Table of Contents:

@ References:

Basic

1. **Define library?**
    - ✔ A Library is a collection of objects.
    - ✔ Type *LIB that is used to group related object and to find objects by name.
    - ✔ A library is a directory to a group of objects.
    - ✔ The number of objects contained in a library and the number of libraries on the system are limited only by the amount of storage available.
    - ✔ All libraries are placed in the system library QSYS.
    - ✔ Libraries provide a method for organizing objects.
    - ✔ A library is an open-ended directory.
    - ✔ A library can never become 'FULL' as if has no finite size.
    - ✔ Libraries themselves are objects.
    - ✔ A library contain the object name, type, and the address
    - ✔ **Library list**
        - ➢ System library- 15 (QSYSLIB)

          QSYS

          QHLPSYS

          QUSRSYS
        - ➢ Product library –2
        - ➢ Current library –1
        - ➢ User library - 25 (QUSRLIB)

          QGPL

          QTEMP

          MYLIB

When you logon the first library to be load is QSYS. The system library is loaded at the first time.

1. **Define object?**
    - ✔ Everything that can be stored or retrieved on the system is known as an "OBJECT". Objects exit to make users independent of the implementation used in the machine.
    - ✔ The create object instruction establish the object's name and its type.

- ✔ All objects are structured with a common object header, and a type dependent functional portion.
- ✔ A user is not concerned with the space his object occupies.
- ✔ The system allocate space automatically
- ✔ WRKOBJPDM is used to display all object in such a library
- ✔ The library the object name and its type is Unique.

1. What is the difference between OPM, EPM and RPGLE?

OPM	EPM	RPGLE
Original program model is the old RPG/400 system, which will not allow a program type to call another program type. Like CL, RPG, COBOL, PL/I, BASIC only supported.	Extended program model will support C, PASCAL, FORTRAN and other programming concepts.	It supports mixed program support in which you can combine any program with another type of program. It supports modularity, copy book, better call performance.
	Version is V1R2	Version is V2R3

2. What are the disadvantages of using CL over RPG?

- ❖ We can able to read only records but we cannot able to write or update or delete records.
- ❖ We can have only one file to be used in a CL program
- ❖ We cannot able to use printer files in CL
- ❖ We cannot able to use subfile in a CL program

1. How you can read and write single command in CL?

By using SNDRCVF command.

Example

Type: CLP

```
                              SKANDASAMO/CLP

                                 ADD

*************** Beginning of data *******************************

0000.01 /*ADDING TWO NUMBERS */
```

```
0001.00        PGM
0002.00        DCLF     FILE (SKANDASAMO/CLPSCR) RCDFMT (SECLP)
0003.00        SNDRCVF   RCDFMT (SECLP)
0004.00        CHGVAR    VAR (&RES) VALUE (&NUM1 + &NUM2)
0005.00        SNDRCVF   RCDFMT (SECLP)
0006.00        ENDPGM
     ****************** End of data **********************************
```

```
OUTPUT

FIRST NUMBER:  12
 SECOND NUMBER: 12
       ----------
RESULT=    0000024
       ----------
```

2. How to retrieve a date in CL?

By using RTVSYSVAL command we can get the system dates. For getting date QDATE.

The various format of date are

*DMY, *MDY, *YMD, *YYMD, *JOL, *JOB

3. How to copy a record in existing object to another object?

By using CPYF command if you want to copy a data one position to another position. We can give the records copying position starting and ending of the records. We want particular records means. We can give the command in sq position.

```
CPYF take F4

File name (source file) : PF01

Lib-name            :SKANDASAMO

New file name              :PF02

Lib-name            :SKANDASAMO
```

:*FIRST

Replace :*ADD

:*NO

:*CHAR

Start position :1000

End position :2000

Sql command :

FILED EMPNO

CONDITION *GT

VALUE 40

Record format mapping:*MAP(add field)

+DROP (delete field)

4. **How you will avoid multiple users updating the same records?**
The displaying the records in the screen we will be getting the timestamp along with the actual data. Store this in output data structure and while updating check whether the previous time stamp is the same timestamp before updating. If the record is updated by another user than the time stamp will be changed and if it does not matches then throw the error message ‘Record is already updated by another user’ else update the records with current time stamp.

Program 1

Store the time stamp and this time stamp will come as an input to the second program

Program2

Here wstmst1 contains the input time stamp and check this matches with the database. If matches update else send error message.

5. **Explain WRKOBJPDM and DSPOBJD?**

- WRKOBJPDM
  If we want to list all the source PF or files of particular type the WRKOBJPDM with file type as PF-SRC for source PF IOR *file for listing all the files extra can be given.

- DSPOBJD
  If we know library name and object name and we want to know the source PF where it is residing then DSPOBJD with option as *services instead of basic will give the source PF name.

1. How to create RPG, RPGLE, CL, CLLE, PF, LF, PRN, and display file?

RPG	-by using CRTRPGPGM command
RPGLE	-by using CRTBNDRPG Command (or) 14
RPGLE	-by using CRTRPGMOD (or) 15 /CRTPGM command
CL	-by using CRTCLPGM command
CLLE	-by using CRTBNDCL
CLLE	-by using CRTCLMOD/CRTPGM command
PF	-by using CRTPF command
LF	-by using CRTLF command
PRN	- by using CRTPRTF command
DSPF	-by using CRTDSPF command

2. What are the advantages of using AS/400 system?

AS/400 is designed and builds as a total system. This means that facilities such as relational database and networking capability (and much more) are fully integrated into the operating system and machine. The user communication with all these functions through a single control language

- Layered machine architecture
- Object orientation
- Single-level storage
- Hierarchy of microprocessors
- Security levels
  - ✔ Layered machine architecture
    This insulates users from hardware characteristics. It enables them to move to new hardware technology at any time, without disrupting their application programs. We can able to change any layer without affecting the other layer. If any problem occurs in OS, then we can work with application program independently and this is the major advantage of AS/400 system.
  - ✔ Object orientation

Every that can be stored or retrieved on the system is known as an “objects”. Objects exist to make users independent of the internal structure of the machine.

- ✔ Single- level storage
  It provides contiguous memory between main storage and disk storage. It provides authority to add any disk space so that user can access it without any problem. There is no need for the user to think where to store the application program.
- ✔ Hierarchy of microprocessors
  Various types of microprocessors are used in AS/400. Each and every microprocessor is allocated for specific purpose. If one chip is for input operation and other for output then we can do both input and output operation since both the microprocessor can perform independently.
- ✔ Security levels
  It will list the various security provided by the system.
  - No security
  - Password security
  - Resource security
  - OS security
  - Certifiable security

1. **What are the various types of Security in AS/400?**
   AS/400 is designed for business that requires levels of security ranging from nothing at all to full government certifiable security. By setting a system value, we can configure five increasing level of security.

- ❖ No security
- ❖ Password security
- ❖ Resource security
- ❖ OS security
- ❖ Certifiable security

When AS/400 is configured, three system values dealing with security need to be specified. These values are QAUDJRL, QMAXSIGN & QSECURITY.

QSECURITY:

This system value determines the level of security enforcement. S/38 and the original AS/400 only had three of system security. At VIR3 of OS/400 the fourth level of security was added, and the fifth level of security was added at V2R3. The valid values for QSECURITY are 10,20,30,40,50.

QMAXSIGN:

This system value determines the maximum number of sign on attempts allowed. If the number of unsuccessful attempts to sign on to the system exceeds this number, the terminal or device that attempted the sign on is varied off.

QAUDJRL:

AS/400 supports an optional security auditing function. If this function is specified, certain security events are journal. The specific events that are logged in the security audit journal are determined by the value specified in the QAUDJRL system value and the level of system security specified.

Level 10: No security

System is shipped with minimum-security level and doesn't require any password to sign on. If user profile doesn't exists with the same name as the

User id the system creates the user profile with that name.

Level 20: Password security

Minimum security is active and password is required to sign on. The user profile must already exist for the user before we can sign on the system

Level 30: Resource security

Password security is active and user must specify given authority to resources. This level is recommended because the system doesn't give the user authority to access the entire object on the system after the user sign on.

Level 40: Operating system security

Password security, resource security and OS integrity are active. User must be especially given authority to resources this level providing more security than level 30.

- All attempts to access object using interfaces that are not supported fail.
- Programs that contains restricted instructions will not compile

- ➢ Users submitting jobs using the job description containing the user profile name must have *USE authority to user profile.

Level 50: C2 level security

All the level 40 security attributes are included at level 50, and in addition some of the interfaces are modified to meet the C2 standards.

1. **Explain user profile and group profile?**
   - ✔ User profiles are used to identify users to the systems and verify authorities on the system (DSPUSRPRF, CHGUSRPRF, EDTOBJAUT)
   - ✔ User profiles tell the system who can sign on and what functions the user can perform on the system on the system resources after signing on.
   - ✔ The security officer or security administrator can create it.
   - ✔ The user profile defines the following capabilities for a particular user
     - ➢ User class
     - ➢ Object owned and authorized
     - ➢ Authorization of objects
     - ➢ Privileged instructions
     - ➢ Password
     - ➢ Current library
     - ➢ Initial program and menu
     - ➢ Delimited-capability user
     - ➢ Limit device session
     - ➢ Maximum storage allowed
     - ➢ Priority limit
     - ➢ Special environment

- ✔ **User class**

  When identifying a user on the system you can specify the user class in the user profile. AS/400 has five user classes that determine the level of system's access a user is permitted. The five user classes, starting the highest level of access, are

  - ✔ Security officer (*SECOFR)
  - ✔ Security administrator (*SECADM)
  - ✔ Programmer (*PGMR)
  - ✔ System operator (*SYSOPR)
  - ✔ User (*USER)

- ✔ **Authorization of objects**

Object authority, or the right to user to use or control an object comes in two categories.

- Object rights
- Data rights

✔ **Object rights**

Object rights are concerned with the object itself.

Object rights assign a user the following authority

- Operational rights (*OPER)
- Object management rights (*OBJMGT)
- Object existence rights (*OBJEXT)

Operational right (*OPER)

The authority to use an object, looks at its description, and restores it. A user must have operational rights to a program to execute it.

Object management rights (*OBJMGT)

The authority to grant and revoke and user rights move and rename object, and members to database file.

Object existence rights (*OBJEXT)

The authority to delete, free storage, save restore or transfer ownership of an object.

✔ **Data rights**

Data rights apply to the data contained within the object.

Types of data rights

- Read (*READ)

  The authority is to retrieve the contents of an object entry.

- Add (*ADD)

  The authority is to add entries to an object. For example adding records to a database file requires ADD right for the library.

- Update (*UPD)

  The authority to change the entries in an object requires UPD right for the file.

- Delete (*DLT)

  The authority is to remove object in an object. For example deleting a program from a library requires DLT right for the library.

Deleting records for a database that requires DLT rights the database file.

1. **What is Lock? How you achieve in AS/400?**

To give the access permission for database file. The Locks are provided by AS/400 system itself.

Type of lock

- ❖ Share lock
  The share locks only reading operation (PF file).
- ❖ Exclusive lock
  The exclusive lock to perform insert, update, and delete operations.

1. **How you will release the lock?**

By using RCLRSC [Reclaim Resource] command we can release the resources only. UNLOCK or CHAIN (N) command also helps to release the lock.

By using WRKOBJLCK command and take F4.

2. **Explain about RTNCSRLOC?**

Type Y (Yes) in the Select parameters prompts to define parameters for the RTNCSRLOC keyword on the Define Return Cursor Location display.

3. **How you execute CL command in RPGLE?**

By using QCMDEXC command we can execute CL command in RPGLE. Two parameters will be called use in the CL command to be executed and second is the length of the command.

4. **What's new in V4R4 and RPG IV?**

There are a few significant enhancements in RPG IV in OS/400 Version 4, Release 4.

The **%CHAR** built-in function has been fixed. It now functions like it was supposed to in the first place. You can wrap a numeric value in %CHAR and a nicely edited character form of the number is returned. The edited form includes the decimal, trimmed off leading blanks, and a negative sign.

The **FOR loop** provide free format version of DO operation code. With the FOR operation, you can begin a loop operation and continue iterating through the loop until a variable equals a limit value. The syntax for the FOR operation is enhanced with the TO, BY and DOWNTO keywords. The TO operation indicators the upper limit for the looping, while the BY keyword identifies the increment value for the loop counter. Alternatively, you can specify the DOWNTO keyword to loop backwards from a large value to a small value.

The **OPENOPT** keyword is added to the Header specification. This keyword can be used along with its one and only keyword *INZOFL to cause overflow indicators to be set off when their corresponding printer file is closed and then re-opened during the program.

In subroutines, the **LEAVESR** operation can now be used to exit a subroutine immediately. Effectively this is a "glorified goto" operation that branches to the ENDSR statement of a subroutine.

5. **Can you clear up the confusion in the different releases of RPG IV and OS/400 and ILE?**

RPG IV is the next generation of the RPG language. RPG III is the original version of AS/400 RPG/400. The name "AS/400 RPG/400" is that given to the IBM compiler package for distribution on the AS/400. This compiler package compiles various versions of RPG, including RPGII and at least two releases of RPGIII.As of OS/400 Version 3 release 1, IBM changed the name of this compiler package to "AS/400 ILE RPG/400". The reason for this name change was to identify that fact that the compile now includes a version of RPG that targets the Integrated Language Environment (ILE), which is RPG IV.ILE was first shipped in OS/400 Version 2, Release 3. However, only the C language compiler produced code that targeted this environment.

First, a word about ILE, ILE is the new, "native" runtime environment for Programs, on the AS/400. Under OS/400 Version 2 Release 3, IBM introduced a new program model. This basically means that new features and interfaces became available. However, IBM did not just port some runtime environment to the OS/400 operating system, it actually re-wrote code, and wrote new code that, essentially, changed the way OS/400 works. This new code provides support for a mixed set of high-level languages. Previously, RPG and CL had their own little runtime environment, COBOL had its own; C had its own, and so on. Under ILE, all programming languages run in ILE. The same "environment" is used for COBOL, C, RPG and CL. However, to take advantage of ILE, new compilers needed to be created. As for RPG, rather than convert the existing RPGII and RPGIII compilers, IBM, who was designing a new version of RPG anyway, decided to target ILE with the new compiler. This would simultaneously provide a new version of RPG and an ILE targeted compiler.

Names Are Important

A good friend of mine once said, "Names are important" in the programming world. If a field is called "Rhinoceros", does it represent its use or purpose? Okay, so perhaps in traditional RPG "Iguana" is a better choice for this example. (Shorter name) During the development of RPG IV, two distinct issues arose. First, the internal name for RPG IV was "ILE RPG". This was not a code name, but rather the name IBM used to refer to the new compiler. After all, it was targeting ILE; why not refer to it as "ILE RPG"? Second, the re-architecture of RPG came into question. Unfortunately, the internal name "ILE RPG" began to be leaked out to the public. Several magazine writers and IBMers not involved in the development of RPG IV continued to use the term "ILE RPG" when referring to RPG IV. I suppose these people still refer to the AS/400 as *SilverLake* or perhaps even *Olympic*.

Then when IBM announced the compiler package or product name as "AS/400 ILE RPG/400" it only added to the confusion. IBM dropped the ball when promoting the RPG IV name. They are, after all, set up to market their products with their product names. The name of one programming language included in a product that contains nearly seven full compilers isn't high priority.

RPG IV is the version of RPG that targets ILE. OS/400 V3R1 compatible RPG IV can also target what is now called "the original program model" or simply OPM. OPM is just a name that has been given to the original runtime environment of RPG and CL under OS/400. This is the environment in which RPGIII and CL run. Under ILE, however, the original native environment is emulated, that is, ILE isn't an environment at all, it is *native* OS/400, whereas, OPM is now an environment under ILE. Some very clever programming and design went into this, don't you think? Not very many other operating systems, if any, provide this kind of continuity.

RPG IV -- Release what?

RPG IV was first shipped with OS/400 Version 3, Release 1. This is now referred to as RPG IV release 1. But don't worry about remembering releases of RPG IV. Under OS/400 Version 3, Release 6, IBM enhanced RPG with procedures, many more built-in functions, and several new data types. This is referred to as RPG IV release 2. Then, OS/400 Version 3, Release 2 was announced. It brought the original release of RPG IV (on the CISC boxes) up to the same level as RPG IV under V3R6. Are you confused yet? Me too! Under OS/400 Version 3, Release 7, IBM added a couple of enhancements, most notably they increased the length of a field name to a number so large not even magazine authors that don't write real-world code could complain about it anymore. They also added one or two new data types, rounding out RPG IV so that it supports all AS/400 data types, except variable length fields. This version of RPG IV is known as RPG IV Release 3.

The following table identifies the current releases of RPG IV. Note that RPG IV releases do not necessarily coincide with releases of the operating system.

RPG IV Release	OS/400 Version/Release	CISC or RISC	
1	V3 R1	CISC	
2	V3 R6		RISC
2	V3 R2	CISC	
3	V3 R7		RISC
4	V4 R2		RISC
4	V3 R5 (speculation) See note 1	CISC	
5	V4 R3		RISC
5	V4 R4 (February 1999)		RISC
6	V4 R5 (Summer 2000)		RISC

NOTE 1: It is speculated that IBM may ship a final "clean up" release of OS/400 for CISC that would included a large level of compatibility with OS/400 V4 R5.

The release levels of RPG IV are only important if you want to keep track of that kind of thing. One disappointing issue is that unless you stay on the most current release of OS/400, you don't get all the cool new features in RPG IV. Even if you stay current, you can't target prior releases if you use any of the new features. In fact, even if you use a new feature that doesn't depend on an operating system enhancement, it can't be used for back releases. This is because of the way the TGTRLS (target release) feature has been implemented. Basically, if you're on V4 R2 and you do a TGTRLS(V3R2M0) the compiler calls the actual compiler for V3 R2. It doesn't have a built-in syntax checker that says "This feature requires an OS/400 upgrade so don't allow it, or this one is okay so accept it." It is calling the same "binary" compiler code that is on any old V3 R2 system. This means, for example, that if you want to take advantage of the new compiler directives, but you often have to target a prior release, you can't use those directives. For example, /IF DEFINED does nothing for the executable code that's generated, but is not supported when TGTRLS(V3R2M0) is specified. ;(Bummer!)

So now we know about RPG IV release levels and how the term "ILE RPG" got into our vocabulary. So let's clear up another term, the name of the RPG language. The big one is the term "RPG/400". There is not programming language called "RPG/400". The language most often called "RPG/400" is RPGIII. However, back in the System/38 days, the System/38 RPG language was called RPGIII. When the AS/400 was announced, programmers wanted to give themselves an advantage on their résumé. So they began calling AS/400 RPGIII, "RPG/400". Then to make matter worse, when RPG IV was announced, programmers thought that the number "IV" in "RPG IV" was less than the "400" in "RPG/400". So they decided to call RPG IV, "ILE RPG". Well let's set the record straight. The table below lists the RPG language names, their incorrect name, and the proper name.

Commonly used Wrong Name	**Formal Name**	**Proper (correct) Name**
RPG/36	System/36-compatible RPGII	RPGII
RPG/38	System/38-compatible RPGIII	RPGIII
RPG/400	RPGIII	RPGIII
ILE RPG	RPG IV	RPG IV

ILE Concepts

1. Integrated Language Environment (ILE)

ILE is an architectural change to language compilers and the runtime characteristics of AS/400 programs. It is an extension to the architecture which means that your existing programs continue to run without changing and recompiling. ILE is available with Version 2 Release 3 of OS/400.

Integrated Language Environment is tightly integrated into the Operating System/400. The key benefits for the new ILE environment are:

- **Language Integration**: Application programs are developed using the language mix best suited to perform each required function.
- **Reusability**: Code from supported languages is divided into smaller, reusable, more logical modules that compile faster and require less maintenance over their life.
    - **Performance**: Capability is provided to optimize code in compute-intensive applications and to reduce the time to perform inter-program calls.

Integrated Language Environment increases developer productivity by providing the capability to divide code into smaller, more logical units that compile faster.

The system binder combines the compiled modules to create the application program. In addition, the separation of compilation and bind steps provides more flexibility packaging the application.

The new source level debug tool that supports the ILE languages provides enhanced capability over the system debugger with the new feature to debug at the source or listing level of the program. Step, breakpoint, and conditional breakpoint functions have been provided. Expressions are entered and evaluated using the syntax of the programming language being debugged. The current system debug facility remains unchanged for programs developed outside ILE.

ILE offers numerous benefits not found on previous releases of the AS/400 system. These benefits include:

1. Better call performance
2. Modularity
3. Multiple-language integration
4. Enhancements to the ILE compilers
5. Reusable components
6. Control over application run-time environment
7. Code optimization
8. Tool availability
9. Foundation for the future

In addition, ILE offers common run-time routines used by the ILE-conforming languages. Many of the application program interfaces (APIs) are also provided as bind able (service) programs. This allows your applications to use APIs and to get faster ILE call performance. These off-the-shelf components provide such services as:

Date manipulation

Message handling

Math routines

Activation Groups

Service Programs

1. Better Call Performance

An ILE compiler does not produce a program that can be run. Instead, it produces a module object (*MODULE) that can be combined, or bound, with other modules to form a single run able unit, or program. ILE programs are called just as you call programs in your current applications.

A benefit of this binding process is that it helps to reduce the overhead associated with calling programs by reducing the number of external calls.

Before ILE, only dynamic (or external) program calls were available to the application programmer. With ILE, two kinds of calls are available:

Dynamic (or external program) calls [E.g.: Program Calls]

Static (or bound) calls [E.g.: Service Program Calls]

The performance of dynamic calls in ILE programs is fairly close to existing call performance. However, bound calls offer better performance than dynamic calls. Thus, the binding capability and the improved call performance that results may encourage you to develop your applications with a more modular design.

2. Modularity

A more modular approach to programming provides numerous benefits to you, including:

- Faster compilation because the units of code to compile are smaller (Especially recompiling during development).
- Better programmer work load distribution.
- Opportunities to both purchase and sell individual modules of code.
- Increased reusability: Modules written for a specific function can be bound into several program objects.
- Simplified maintenance: Maintenance may be required in only a single module.

3. Multiple-Language Integration

With your current application, you can mix different language programs, such as RPG, COBOL, and C. However, to access code written in another language, your current application must perform a dynamic call to a separate program. The performance cost of the dynamic call to a program and the inconsistencies between language behaviors sometimes complicate the mixing of languages.

With ILE, modules written in any ILE language can be bound to modules written in the same or any other ILE language. For example, a module of code written in ILE C/400 (perhaps a floating-point calculation) can be bound with modules written in ILE RPG/400, ILE COBOL/400, ILE C/400, or ILE CL.

This produces a better performing, and more easily managed application. In addition, you can acquire modules written in a variety of languages, without needing to produce the code yourself. The APIs that IBM provides for ILE are just the beginning. Vendors have more freedom to sell (and application programmers to buy) libraries of routines for any commonly used function, such as tax calculations. They can be written in any language and can be bound for better performance.

4. Enhancements to the ILE Compilers

The ILE compilers have some significant new function included as part of the language. This is particularly true for ILE RPG/400, which is based on the RPG IV language definition. Many long-standing requests from RPG programmers have been addressed in the ILE RPG/400 compiler, including the following:

- 10-character field names
- Free-form logical and math expressions
- Date and time data types and operations
- External data items (data export)
- Uppercase and lowercase source
- File-level field prefix support
- Pointers

For many programmers, the primary motivation for moving to ILE is to get access to the function that ILE language support provides.

5. Reusable components

ILE allows you to select packages of routines that can be blended into your own programs. Routines written in any ILE language can be used by all AS/400 ILE compiler users. The fact that programmers can write in the language of their choice ensures you the widest possible selection of routines.

The same mechanisms that IBM and other vendors use to deliver these packages to you are available for you to use in your own applications. Your installation can develop its own set of standard routines, and do so in any language it chooses.

Not only can you use off-the-shelf routines in your own applications. You can also develop routines in the ILE language of your choice and market them to users of any ILE language.

6. Control over Application Run-Time Environment

ILE allows you to use better control over your application and the resources it uses. You can specify that a given ILE program run in a particular area within a job. This area within a job is called an activation group. You can assign a name to the activation group within the job. Then, ILE programs and service programs can be created to use the named activation group. Thus, you can use activation groups to set up logical boundaries within the job to separate the applications.

Within these boundaries, an activation group has exclusive use of the resource, such as open data paths for the files used in the application.

Using activation groups to isolate applications can also make it easier to end an application in a job. It aids in cleaning up its resources (such as open files and active programs) without disturbing resources associated with other applications active in the job. RPG programmers might think of this technique as a kind of application-level LR indicator. For example, it is a way to end an entire application rather than ending one program at a time.

7. Code Optimization

The new ILE compilers and the associated OS/400 translator have more advanced optimization techniques built into them. In some cases, these new levels of optimization may lead to improved performance of existing code. At compilation time, the programmer can select the desired level of optimization for each piece of code.

8. Tool Availability

The majority of tools for developers in the computer industry today are written in the C language. With ILE binding capability and improved optimization, these C language applications run faster. In addition, they perform better than they did with the previous C/400 compiler.

Therefore, we anticipate that many tool vendors will begin to add their tools to the AS/400 to attract a new marketplace for their products.

Making use of the C language offers you a greater choice of:

- CASE tools, Fourth-generation languages (4GLs), Editors, Debuggers.

9. Foundation for the Future

In addition to the increased opportunity to optimize your applications with the current ILE compilers, you can look forward to even more significant enhancements. The move toward object-oriented programming languages and visual programming tools increases the need for the capabilities provided by ILE.

Applications constructed of large numbers of small, modularized, reusable components that efficiently transfer control among themselves offer you maximum flexibility. You can use them multiple ways in multiple applications.

1. A **procedure** is a set of self-contained high-level language statements that performs a particular task and then returns to the caller. For example, an ILE C/400 function is an ILE procedure.

2. A **sub procedure** is a procedure specified after the main source section. It can only be called using a bound call. Sub procedures differ from main procedures in several respects, the main difference being that sub procedures do not (and cannot) use the RPG cycle while running.

3. A **module object** is a non runable object that is the output of an ILE compiler. A module object is represented to the system by the symbol *MODULE. A module object is the basic building block for creating run able ILE objects.

4. All ILE programs and service programs are activated within a substructure of a job called an **activation group**. This substructure contains the resources necessary to run the programs.

5. A **service program** is a collection of runable procedures and available data items easily and directly accessible by other ILE programs or service programs. In many respects, a service program is similar to a subroutine library or procedure library.

Service programs provide common services that other ILE objects may need; hence the name service program. An example of set of service programs pro-vided by OS/400 is the run-time procedures for a language. These run-time procedures often include such items as mathematical procedures and common input/output procedures.

6. A **binding directory** contains the names of modules and service programs that you may need when creating an ILE program or service program. Modules or service programs listed in a binding directory are used only if they provide an export that can satisfy any currently unresolved import requests. A binding directory is a system object that is identified to the system by the symbol *BNDDIR.

1. What is a Module?

Modules are objects of *MODULE type that are created by the compiler when the create RPG Module (CRTRPGMOD) command is performed. A module can be composed of a main procedure (also referred to as main program) and/or one or more sub procedures.

The term “procedure” often designates a sub procedure or a main procedure. A module is sometimes called “compilation unit” as it comes from compilation of one source member. Modules are not executable; they only serve as building blocks for program creation.

The process of program creation is called binding. Bound programs are executable objects of *PGM type. To bind modules into a program, the Create Program (CRTPGM) command is used. If an RPG IV program does not call sub procedures, or external modules, the Create Bound RPG Program (CRTBNDRPG) command will do for both compilation and binding. This is the case, for example, of an RPG IV program resulting from converting an RPG III program by the CVTRPGSRC command.

A module is a non-executable program and it contains one or more procedures. If you have modules without procedure then it means that it is having only one default procedure and in case we can use CALLB. We are creating a RPGLE module by CRTRPGMOD and a CL module by CRTCLMOD commands.

2. What is a Service Program?

If you have procedures that are called by more than one program, you could bind them individually to each of the programs. In such a case, they would occupy space in each program and would be difficult to maintain. If you group the procedures in a service program instead, the procedures occur only once and can be easily maintained. Service programs are objects of *SRVPGM type which are created by the Create Service Program (CRTSRVPGM) command. A service program is simply a collection of modules especially those containing sub procedures.

Service programs cannot be directly called; however, the procedures contained in it may be called by ILE programs.

Service programs are built by binding, much like programs, but they need to be further bound to a program before they are used. This is done by the CRTPGM command. Service programs can also be bound to other service programs. The top service program in such a group is eventually bound to a program using the CRTPGM command.

3. What is a binding Directory?

Binding directories are objects of *BNDDIR type. Binding directories can be used as an additional source of exports. A binding directory contains a list of modules and service programs that are candidates for automatic binding. Not all items of the list in the binding directory are necessarily bound. Only those required by imports that cannot otherwise be resolved are bound. Modules and service programs listed in a binding directory often contain standard procedures, for example mathematical functions or other system procedures. We can create our own binding directories using special CL command CRTBNDDIR.

4. Why Import and Export?

A service program makes its own modules and procedures available to external users through a mechanism called export. The external users are modules and sub procedures in external programs and other service programs that use (call) the modules and sub procedures of the service program to call them. The external users are also called “public” or “clients”.

Main procedures of modules comprising the service program are exported automatically (implicitly), the programmer do not need to use any special specifications to make a main procedure available to external users. A service program exports its own sub procedures by specifying the EXPORT keyword in the sub procedure definition. However, in order to bring this specification in effect, the binding command (CRTSRVPGM) specifies which of the exported procedures are actually made available to external users. Besides modules and sub procedures, variables may be exported (by specifying the EXPORT keyword). Exported modules, sub procedures and variables are collectively called exports. Exports are used in other procedures where they are referred to. The references are also called imports as opposed to the exports that are sometimes called definitions.

5. What is Activation Group?

Activation groups are temporary storage structures placed inside jobs (which themselves are also temporary structures). There are three types of activation groups:

- Default
- named
- New

Default activation groups exist automatically and are never deleted. There are two default activation groups. Many system programs run in the default activation group 1. RPG IV programs created with the parameter DFTACTGRP(*YES) of the CRTBNDRPG command run in the default activation group.

The other types of activation groups are specified by the parameter ACTGRP in program and service program creation commands - CRTPGM and CRTSRVPGM. Thus, the type of an activation group is determined by the program or service program at creation time.

An activation group is created when the program is started. An activation group may include:

Static and automatic variables

The variables types of programs running in the activation group: Static variables are those defined in a main procedure. They come from external sources such as DDS or SQL specifications, or they are defined as RPG variables (fields, indicators). One more place you will find static variables is as local variables in sub procedures declared with the STATIC keyword. Automatic variables are local variables defined in sub procedures.

Open data paths (ODP)

ODP are temporary objects representing open files to programs. Data buffer and pointer to a record are part of the ODP.

Dynamically allocated storage

Temporary object created by the ALLOC operation in the RPG IV program.

Error handling routines

System or user programs (modules) handling error messages. Programmers can write their own modules to handle error messages coming from any procedure in the call stack, no matter in which programming language the procedure is written. Notice that the “program stack” has been renamed to “call stack”.

6. Name Some ILE API’s? And tell something about them?

The List Module Information (**QBNLMODI**) API lists information about modules. The information is placed in a user space specified by you. This API is similar to the Display Module (DSPMOD) command. You can use the QBNLMODI API to:

• List the symbols defined that can be exported to other modules

• List the symbols that are defined external to the module

• List procedure names and their type

• List objects that are referenced when the module is bound into an ILE program or service program

• List copyright information

2. The List Service Program Information (**QBNLSPGM**) API gives information about service programs, similar to the Display Service Program (DSPSRVPGM) command. The information is placed in a user space specified by you. You can use the QBNLSPGM API to:

• List modules bound into a service program

• List service programs bound to a service program

• List data items exported to the activation group

• List data item imports that are resolved by weak exports that were exported to the activation group

• List copyrights of a service program

• List procedure export information of a service program

• List data export information of a service program

• List signatures of a service program

3. The List ILE Program Information (**QBNLPGMI**) API gives information about ILE programs, similar to the Display Program (DSPPGM) command. The information is placed in a user space specified by you. You can use the QBNLPGMI API to:

• List modules bound into an ILE program

• List service programs bound to an ILE program

• List data items exported to the activation group

• List data item imports that are resolved by weak exports that were exported to the activation group

• List copyrights of an ILE program

You can, for example, list signatures of service programs bound in a program using the **QBNLSPGM** API to get the "old" signatures. You can also list all "new" signatures of these service programs using the QBNLPGMI API and compare the two lists if they match. If there is some mismatch, you can trigger a new binding of the program (by performing the CRTPGM command). Be prepared to inspect lists of lists in some cases because the information retrieved by these APIs is organized hierarchically.

7. What are activation groups?

Activation group is the environment where the ILE jobs are executed. You can specify the activation group in CRTPGM or CRTSRVPGM command.
You cannot create the activation group by command CRTACTGRP.

*ENTMOD: The program entry procedure module (ENTMOD parameter) is checked. If the module attribute is RPGLE, CBLLE, or CLLE, then ACTGRP is QILE or QILETS.QILE is used when STGMDL (*SNGLVL) is specified, and QILETS is used when STGMDL (*TERASPACE) is specified.

*NEW   : When the program gets called, a new activation group is created.

*CALLER: When the program gets called, the program is activated into the caller's activation group.

Name   : Specify the name of the activation group to be used when this program is called.

8. How do I create and use a service program

http://faq.midrange.com/data/cache/614.html

A small sample service program, with source, and instructions for how to create and use it is given here.

a. Create the 6 source members below, changing MYLIB to your library name in the entire source.
b. Use CRTRPGMOD to create the two srvpgm modules SRVSAMP1 and SRVSAMP2.
c. Use CRTCLPGM to create the CL program SRVSAMPCRT.
d. Call SRVSAMPCRT to create the srvpgm MYLIB/SRVSAMP.
e. Create a binding directory
   ===> CRTBNDDIR MYLIB/SRVSAMPBND
f. Add your service program to the binding directory
   ===> ADDBNDDIRE MYLIB/SRVSAMPBND OBJ((MYLIB/SRVSAMP *SRVPGM))
g. Create your test program. Since it has the BNDDIR keyword in the
   H spec, it will automatically find your service program.
   ===> CRTBNDRPG MYLIB/SRVSAMPTST SRCFILE(MYLIB/QRPGLESRC)
h. Try calling your test program. Enter a value like 06.03.31 or 060331
i. Try adding a new procedure to module SRVSAMP1.

- say a procedure to get a numeric value similar to get Answer
- add the prototype to SRVSAMPPR
- code the procedure in SRVSAMP1
- add an EXPORT line to QSRVSRC SRVSAMPBND
- call your CL to create the srvpgm again
- reclaim the activation group that your test program runs in
  (Use DSPPGM to see what activation group it is)
- call your test program again to make sure it still runs ok with the new version of the service program

j. Add some code to your test program to call the new procedure, and recompile and test.

Source files:

1. Binder source MYLIB/QSRVSRC SRVSAMPBND type BND:

```
/*--------------------------------------*/
/* Rules:                               */
/* 1. Never change the order of exports */
/* 2. Add new exports at the end        */
/*--------------------------------------*/
strpgmexp signature('SRVSAMP')
   export symbol('getAnswer')     /*   1 */
   export symbol('chkDate')       /*   2 */
endpgmexp
```

2. CL to create service program MYLIB/QCLSRC SRVSAMPCRT type CLP:

```
crtsrvpgm mylib/srvsamp         +
   module(mylib/srvsamp1        +
          mylib/srvsamp2)       +
   srcfile(mylib/qsrvsrc) srcmbr(srvsampbnd)
```

3. RPG prototype source MYLIB/QRPGLESRC SRVSAMPPR type RPGLE:

```
 /if defined(SRVSAMPPR_COPIED)
 /eof
 /endif
 /define SRVSAMPPR_COPIED
D getAnswer       pr            25a   varying
D                                     extproc('getAnswer')
D   question                    25a   const varying

D chkDate         pr              n
D                                     extproc('chkDate')
D   input                       10a   const varying
D   output                        d
```

```
D   formatParm                  10a   const varying
D                                   options(*nopass)
```

4. RPG test program source MYLIB/QRPGLESRC SRVSAMPTST type RPGLE:

```
H dftactgrp(*no) bnddir('MYLIB/SRVSAMPBND')
 /copy srvsamppr
D ans             s             10a      varying
D date            s               d
D ok              s               n

 /free
     ans = getAnswer ('Give a date in ymd format');
     ok = chkDate (ans : date : '*YMD');
     if ok;
        dsply ('That was ok, date was ' + %char(date));
     else;
        dsply 'Oops, was not valid';
     endif;
     *inlr = '1';
```

5. RPG srvpgm module 1 MYLIB/QRPGLESRC SRVSAMP1 type RPGLE:

```
H nomain
 /copy srvsamppr

P getAnswer       b                   export
D getAnswer       pi            25a   varying
D   question                    25a   const varying
D answer          s             25a   varying
 /free
    dsply question ' ' answer;
    return answer;
 /end-free
P getAnswer       e
```

6. RPG srvpgm module 2 MYLIB/QRPGLESRC SRVSAMP2 type RPGLE:

```
H nomain
 /copy srvsamppr

P chkDate         b                   export
D chkDate         pi              n
D   input                       10a   const varying
D   output                        d
D   formatParm                  10a   const varying
D                                   options(*nopass)
```

```
D format          s             10a   varying inz('*ISO')
D sep             s              1a
D sepPos          s             10i 0
D haveSep         s               n
D standardSep     s             10a   varying
 /free
   // check for optional parameter
   if %parms > 2;
      format = formatParm;
   endif;

   // check for separators
   sepPos = %check('0123456789' : input);
   if sepPos > 0;
      sep = %subst(input : sepPos : 1);
      haveSep = *on;
   endif;

   if format = '*ISO';
      if haveSep;
         standardSep = %xlate(sep:'-':input);
         output = %date(standardSep : *iso);
      else;
         output = %date(input : *iso0);
      endif;

   elseif format = '*YMD';
      if haveSep;
         standardSep = %xlate(sep:'/':input);
         output = %date(standardSep : *ymd/);
      else;
         output = %date(input : *ymd0);
      endif;
   endif;

   return *on; // it was ok

   begsr *pssr;
      return *off;  // some error occurred
   endsr;
 /end-free
P chkDate         e
```

9. Modules - How to write and reuse them

Here we explore the concept of modules which helps us to reuse the procedures without applying the service program concept.

Name of the program that binds all the modules is TSTMOD. There won't be any source for TSTMOD before CRTPGM. After program creation if you debug this program and see it will have the source of the entry module (here PGMENTMOD)

There are three modules PGMMOD1, PGMMOD2 and PGMENTMOD where PGMENTMOD is the entry module (it contains the code to call the exportable procedures present in PGMMOD1 and PGMMOD2).

1. First create all the modules using option 15 like this.

```
CRTRPGMOD MODULE (SHAILESH/PGMMOD1) SRCFILE (SHAILESH/TESTPGMS)
CRTRPGMOD MODULE (SHAILESH/PGMMOD2) SRCFILE (SHAILESH/TESTPGMS)
CRTRPGMOD MODULE (SHAILESH/PGMENTMOD) SRCFILE
(SHAILESH/TESTPGMS)
```

2. Then create the program (CRTPGM) by binding all the modules like this.

```
Program  . . . . . . . . . . . . > TSTMOD
  Library  . . . . . . . . . . . >   SHAILESH
Module . . . . . . . . . . . . . > PGMMOD1
  Library  . . . . . . . . . . . >   SHAILESH
                                 > PGMMOD2
                                 >   SHAILESH
              + for more values > PGMENTMOD
                                 >   SHAILESH
Text 'description' . . . . . . .   *ENTMODTXT
                          Additional Parameters
Program entry procedure module   > PGMENTMOD
  Library  . . . . . . . . . . . >   SHAILESH
```

3. Now call the created program in debug mode like this.

```
STRDBG PGM(SHAILESH/TSTMOD) UPDPROD(*YES) OPMSRC(*YES)
CALL PGM(SHAILESH/TSTMOD)
```

Source code:

PGMMOD1:

```
D ADD           PR
D  VAR1                      5S 0 VALUE
D  VAR2                      5S 0 VALUE
```

```
D  SUM                          6S 0

C                  EVAL      *INLR = *ON

P ADD             B                   EXPORT
D ADD             PI
D  VAR1                          5S 0 VALUE
D  VAR2                          5S 0 VALUE
D  SUM                          6S 0

C                  EVAL      SUM = VAR1 + VAR2
P ADD             E
```

PGMMOD2:

```
D SUB             PR
D  VAR1                          5S 0 VALUE
D  VAR2                          5S 0 VALUE
D  DIFF                         5S 0

C                  EVAL      *INLR = *ON

P SUB             B                   EXPORT
D SUB             PI
D  VAR1                          5S 0 VALUE
D  VAR2                          5S 0 VALUE
D  DIFF                         5S 0

C                  EVAL      DIFF = VAR1 - VAR2
P SUB             E
```

PGMENTMOD:

```
D VAL1            S              5S 0 INZ(99999)
D VAL2            S              5S 0 INZ(99999)
D TOTAL           S              6S 0 INZ
D BALANCE         S              5S 0 INZ

D ADD             PR
D  VAL1                          5S 0 VALUE
D  VAL2                          5S 0 VALUE
D  TOTAL                         6S 0

D SUB             PR
```

```
D  VAL1                        5S 0 VALUE
D  VAL2                        5S 0 VALUE
D  BALANCE                        5S 0

C     TOTAL         DSPLY
C                CALLP     ADD(VAL1: VAL2: TOTAL)
C     TOTAL         DSPLY
C     BALANCE       DSPLY
C                CALLP     SUB(VAL1: VAL2: BALANCE)
C     BALANCE       DSPLY

C                EVAL      *INLR = *ON
```

1. **What are the ILE RPG coding programming considerations?**

Coding Considerations

This section presents some considerations that you should be aware of before you begin designing applications with multiple-procedure modules. The items are grouped into the following categories:

* General

* Program Creation

* Main Procedures

* Sub procedures

1. General Considerations

* When coding a module with multiple procedures, you will want to make use of /COPY files, primarily to contain any prototypes that your application may require. If you are creating a service program, you will need to provide both the service program and the prototypes, if any.

* Maintenance of the application means ensuring that each component is at the most current level and that any changes do not affect the different pieces. You may want to consider using a tool such as Application Development Manager to maintain your applications.

For example, suppose that another programmer makes a change to the /COPY file that contains the prototypes. When you request a rebuild of your application, any module or program that makes use of the /COPY file will be recompiled automatically. You will find out quickly if the changes to the /COPY file affect the calls or procedure interfaces in your application. If there are compilation errors, you can then decide whether to accept the change to prototypes to avoid these errors, or whether to change the call interface.

2. Program Creation

* If you specify that a module does not have a main procedure then you cannot use the CRTBNDRPG command to create the program. (A module does not have a main procedure if the NOMAIN keyword is specified on a control specification.)

This is because the CRTBNDRPG command requires that the module contain a program entry procedure and only a main procedure can be a program entry procedure.

* Similarly, when using CRTPGM to create the program, keep in mind that a

NOMAIN module cannot be an entry module since it does not have a program entry procedure.

* A program that is created to run in the default OPM activation group (by specifying

DFTACTGRP(*YES) on the CRTBNDRPG command) cannot contain bound procedure calls.

3. Main Procedure Considerations

* Because the main procedure is the only procedure with a complete set of specifications available (except P specification), it should be used to set up the environment of all procedures in the module.

* A main procedure is always exported, which means that other procedures in the program can call the main procedure by using bound calls.

* The call interface of a main procedure can be defined in one of two ways:

1. Using a prototype and procedure interface

2. Using an *ENTRY PLIST without a prototype

* The functionality of an *ENTRY PLIST is similar to a prototyped call interface.

However, a prototyped call interface is much more robust since it provides parameter checking at compile time. If you prototype the main procedure, then you specify how it is to be called by specifying either the EXTPROC or EXTPGM keyword on the prototype definition. If EXTPGM is specified, then an external program call is used; if EXTPROC is specified or if neither keyword is specified, it will be called by using a procedure call.

* You cannot define return values for a main procedure, nor can you specify that its parameters be passed by value.

4. Sub procedure Considerations

* Any of the calculation operations may be coded in a sub procedure. However, all files must be defined globally, so all input and output specifications must be defined in the main source section. Similarly, all data areas must be defined in the main procedure, although they can be used in a sub procedure.

* The control specification can only be coded in the main source section since it controls the entire module.

* A sub procedure can be called recursively. Each recursive call causes a new invocation of the procedure to be placed on the call stack. The new invocation has new storage for all data items in automatic storage, and that storage is unavailable to other invocations because it is local. (A data item that is defined in a sub procedure uses automatic storage unless the STATIC keyword is specified for the definition.)

2. What Opcodes are added in ILE?

ADDDUR, SUBDUR, EVAL

3. What are the behavioral differences b/w OPM RPG/400 and ILE?

Compared to OPM, ILE provides RPG users with improvements or enhancements in the following areas of application development:

* Program creation

* Program management

* Program call

* Source debugging

* Bindable application program interfaces (APIs)

Each of the above areas is explained briefly in the following paragraphs and discussed further in the following chapters.

4. ILE advantages over RPG?

- ✔ Better call performance
- ✔ Modularity

- ✔ Multiple-language integration
- ✔ Enhancements to the ILE compilers
- ✔ Reusable components
- ✔ Control over application run-time environment
- ✔ Code optimization
- ✔ Tool availability
- ✔ Foundation for the future

1. **Define binder program?**
   The binder program means binding the procedure it is called binder program.

2. **How to the create module?**
   A module is created as a separate object type (*MODULE). Using the CRTRPGMOD command creates an RPGLE module. A module object cannot be run directly. You must use the CRTPGM command to bind module object into a program object. Do option 15 or CRTRPGMOD command to create a module. The CRTPGM command is used to create a program from one or more module.

```
                                        SKANDASAMO/RPGILE

                                            MAIN

       *************** Beginning of data *******************************

0001.00 C              CALLB    'ADD'

0002.00 C              CALLB    'SUB'

0003.00 C              CALLB    'MUL'

0004.00 C              SETON                          LR

       ****************** End of data **********************************

                                        SKANDASAMO/RPGILE

                                            ADD

       *************** Beginning of data *******************************

0002.00 C              Z-ADD    4        A            4 0

0002.01 C              Z-ADD    5        B            4 0

0004.00 C     A         ADD     B        C            4 0

0005.00 C     C         DSPLY
```

```
0006.00 C                  SETON                                    LR
        ****************** End of data *********************************
                                                         SKANDASAMO/RPGILE
                                                                    SUB
       *************** Beginning of data *******************************
002.00 C                  Z-ADD     10           A                  4 0
002.01 C                  Z-ADD     5            B                  4 0
004.00 C     A            SUB       B            C                  4 0
005.00 C     C            DSPLY
006.00 C                  SETON                                    LR
       ****************** End of data *********************************
                                                         SKANDASAMO/RPGILE
                                                                    MUL
        *************** Beginning of data ******************************
0002.00 C                  Z-ADD     10           A                  4 0
0002.01 C                  Z-ADD     5            B                  4 0
0004.00 C     A            MULT      B            C                  4 0
0005.00 C     C            DSPLY
0006.00 C                  SETON                                    LR
        ****************** End of data *********************************
```

CRTPGM Take F4

```
 Program  . . . . . . . . . . . . > MAIN         Name
   Library  . . . . . . . . . . . >   SKANDASAMO  Name, *CURLIB
 Module . . . . . . . . . . . . . > MAIN          Name, generic*, *PGM, *ALL (PEP)
   Library  . . . . . . . . . . . >   SKANDASAMO  Name, *LIBL, *CURLIB...
```

```
                                   > ADD
                                   >   SKANDASAMO
                                   > SUB
                                   >   SKANDASAMO
                         + for more values > MUL
                                   >   SKANDASAMO
      Text 'description' . . . . . . .   *ENTMODTXT
     OUTPUT
     DSPLY     9
      DSPLY     5
      DSPLY    50
```

3. **What are the differences in CALL, CALLB and CALLP?**

CALL is a dynamic call where the control will be transferred when the program is executed. (Control will be transfer the another program (run time) so it is dynamic call).

Where as CALLB and CALLP are static calls. A module is a non-executable program and it contains one or more procedures. If you have modules without procedure then it means that it is having only one default procedure and in case we can use **CALLB.**

A module is having more than one procedure then we can give explicitly the procedure name to be called in case of **CALLP** out of these three CALLP is the most efficient one. (Using the CALLB, CALLP a program or module is bind in the program so it is static).

4. **What is the difference between Bind by value and Bind by reference?**

Bind by value	Bind by Reference
Here the entire modules to be bounded are physically copied into the main program object.	In this case we are binding the programs by using service programs, which contain a reference to the module that has been called, and the modules are

	not physically copied into the program object.
The program will be executed even when you delete the entire module that has been called.	The program will not execute when the bind modules are deleted.
Bind by value is faster than bind by reference. (All the modules to be bind in the main program, so it is fast)	It is not as faster as bind by value. (All the modules can't bind the main program it is refer the pointer)

5. Define pass by value and pass by reference?

❖ **Pass by reference:**

Pass by reference we are passing the address of the parameters and not the actual value and so the changes in the called procedure will affect the value in the calling programs. In OPM programs we are using only call by reference.

❖ **Pass by value:**

Pass by value we are passing the value of the parameter, changes made to the formal arguments in the called function have no effect on the values of the actual arguments in the calling function it is used in c program.

In RPGLE we have the option to pass the parameter by value by giving the keyword **VALUE.**

1. What are Program Entry Procedure (PEP) and User Entry Procedure (UEP)?

If we are binding many modules together to form a program then we have to specify which module has to take control first when it has been called and that module is called as **PEP** for that program.

User entry procedure **(UEP)** is the first statement that takes the control when a program has been called. For example in C programs main () will be executed first when it has been called and likewise in RPG the statement coded in C Spec will take the control first.

2. **Define Copybook in RPGLE?**
   It will copy a Subroutine (or) any group of codes physically into the program, which is copying it.

```
                                          SKANDASAMO/RPGILE
                                                        COP
     *************** Beginning of data *****************************
0002.00 C                DSPLY              A            5 0
0002.01 C                EXSR     ADD
0004.00 C                SETON                                  LR
0005.00 C/COPY RPGILE, COPY
     ****************** End of data ********************************
                                              SKANDASAMO/RPGILE
                                                     COPY
     *************** Beginning of data *****************************
0000.01
0001.00 C     ADD       BEGSR
0002.00 C                ADD       5           A
0003.00 C     A         DSPLY
0004.00 C                ENDSR
     ****************** End of data *******************************
OUTPUT
13
DSPLY     18
```

3. **How to create a service program and what are the steps involved in this?**

- ✔ The service program means most commonly used modules are grouped (binding) together to form it is called service program.
- ✔ A service program is not bound to its caller until activation time

While creating service program we can create a binder program where we can refer the modules (or) procedures (or) even data types to be used by the program which is using service program.

➢ ***Advantages of service programs***

- ❖ They do not take up auxiliary storage space. There is only one copy for all users.
- ❖ There is only a single copy of the read-only code in main storage for all users in this service programs is the same as a program that you call dynamically.
- ❖ Each user of the service program has an independent work area.
- ❖ You can pass parameters to a service programs by using the traditional parameter list (or) by importing and exporting variables.
- ❖ Service programs can be maintained independently of the programs that use the functions. In most cases, changing a service programs does not cause a program using the function to be changed or re-created.

➢ ***Disadvantages of service programs***

- ❖ Service programs are less desirable for a function you may or may not need. The reason is that it is slower to call a main program that refer to a service program

1. Explain procedure used in RPGLE?

A procedure is a non-executable program. If a module is having more than one procedure then we can give explicitly the procedure name to be called in case of CALLP.

- ❖ Defining the prototype:

  Prototype will specify the following things

  - ✔ Parameter type
  - ✔ Sequence of the parameter
  - ✔ Return variable and its type
  - ✔ It tells the name of the procedure and also the type of the call.

  It will avoid all the run time problems like parameter mismatch by specifying the prototype.

- ❖ Prototype interface

  It is like *entry parameter where we will specify the parameters that are received in this program.

- ❖ Import and export

  If you want to specify the procedures to be the external programs then we can specify EXPORT in your procedure.

- ❖ Global and local variables
  If you declare a variable in main procedure then it will be accessible in all sub procedure and this is global declaration and if you specify the declaration in the sub procedures then it will not be accessed in other procedures or in the main procedure.
- ❖ Return
  If we specify return in the sub procedure then it means that we are returning something to the calling program. We can return a maximum of only one variable to the calling program.
- ❖ Recursion
  A procedure calling to itself is known as recursion.
- ❖ Pass by value/pass by reference
  In case of pass by reference we are passing the address of the parameters and not the actual value and so the changes in the called procedure will affect the value in the calling program. In OPM program we are using only call by reference and in RPGLE we have the option to pass the parameter by giving the keyword VALUE.
- ❖ CALLP/Expression
  We can call the procedure by using CALLP command if it is not having any return type and by an expression if it returns any value.

Database

1. **Define source physical file?**

Source physical file is also a file, which has one, or more files included in it. It is just like a directory and it contains many members. The members are like a various programs residing in the directory **CRTSRCPF** is used to create source physical file.

2. **Physical Files and Logical File**

Physical files hold the actual data of a database file. The data is written in arrival sequence.

Physical files are not required to have keyed fields. If a physical file has key fields, this is the order that an RPG program will read the data if the File Spec in the program indicates to read the data in keyed sequence. Also, with a keyed field, an RPG program can CHAIN, SETLL, READE and READP.

A simple logical file is a different view of the physical file. It is actually a list of pointers to the physical file. Most of the time, a logical file is nothing more than a way of accessing the physical file with different key fields.
Logical file does not occupies any memory space and logical file be derived from physical file. One or more logical file can be derived from a single physical file. A logical file can contain up to 32 record formats. It selects records dynamically. It cannot exist without a physical file. We can filter the data with criteria by using

select and **omit** command. **CRTLF** command is used to create a LF. It accesses the data by creating access path.

A logical file does not contain any data but provides the 'VIEWS' of the data to satisfy end-user's needs. There are two types, 1. Non - join logical file, 2. Join logical file With the standard AS/400 supplied tools, it is hard to see the logical file. One way is to use the copy file CPYF to copy the logical file to a new physical file. Then, look at the physical file... it will be in the same order as the logical file. The AS/400 Database is full featured. Logical files can join multiple files and select and create new fields. Maximum number of fields included in a PF is 8000. Maximum no of key fields included is 120.

There are two types of logical files,

- ✔ Non join logical file
- ✔ Join logical file

1. List the differences between physical file and logical file.

Physical file	Logical file
1. Occupies the portion of memory. It's containing data.	Does not occupy any memory space. Does not contain any data.
2. A physical file contains one record format and can contain many members	A logical file can contain up to 32 record formats but only one member.
3.Can be exist even without LF	Cannot exist without PF
4. The PF cannot be deleted without deleting the associated LFs.	If the LF is deleted then the PF need not be deleted.
5.**CRTPF** command is used to create such object	**CRTLF** command is used to create such type object
6.The object type is PF	The object type is LF

2. What are the four levels of entries in physical file?

(I). **File level entries (optional)**: File level entries give the system information of the entire file. (UNIQUE, LIFO, FIFO, FCFO, REF)

UNIQUE: A record cannot be entered or copied into a file if its key value is same as the key value of a record already existing in the file.

LIFO: Last in first out

FIFO: First in first out

FCFO: First change first out.

REF: This keyword is used to specify the name of the file from which the field descriptions are retrieved.

Ex: you can specify whether the key is unique.

(ii). **Record format level entries**: Record format level entries give the system information about specific record format in the file. For a **PF** the record format name is specified along with an optional text description. (FORMAT, TEXT)

(i) FORMAT:

This record-level keyword specifies that the record format being define is to share the field specifications of a previously defined record format. The name of the record format being defined must be the name of the previously defined record format.

The format of this keyword is:

FORMAT (LIB-NAME / FILE-NAME)

(ii) TEXT:

This record level keyword is used to supply a text description of the record format and it is used for documentation purposes only.

The format of this keyword is:

TEXT (‘description’)

(iii) **Field level entries:** The field names and field lengths are specified along with and optional text description for each field. (ALIAS, ALWNULL, CCSID, CHECK, CHKMSGID, CMP, COLHDG, COMP, DATFMT, DATSEP, DFT, EDTCDE, EDTWRD, REFFLD, REFSHIFT, TEXT, TIMEFMT, TIMESEP, VALUES, VARLEN)

(iv) **Key field level entries**: The field names used as key fields are specified. (DESCEND, SIGNED, ABSVAL, UNSIGNED, ZONE, NOALTSEQ, DIGIT)

1. **What are the six levels of entries in logical file?**

(i) **File level entries (optional**): File level entries give the system information of the entire file. You can specify whether the key is same as physical file.

(ii) **Record format level entries**: Record format level entries give the system information about specific record format in the file. For examples, for a logical file when a record format is described we can specify the physical file it is based on. Example: Jfile, PFile.

(iii) **JOIN Level entries:** Join level entries give the system information about **PF** used in a JOIN LOGICAL FILE. (It is not applicable to NON JOIN LOGICAL FILES).
Join, JFLD,

(iv) **Field level entries** (**optional):** The field level entries give the system information about individual fields in the record format. Example: JRef

(v) **Key field level entries:** The key field level entries give the system information about the key fields of a file. The field names used as key fields are specified.

(vi) **Select / Omit level entries:** These entire give the system information about which records are to be returned to the program when processing the file. These specifications apply to logical file only.
Examples: Range(field), values(recordformat)

1. **Explain JDUPSEQ and JDFTVAL.**

JDUPSEQ:

This join –level keyword is used to specify the order in which records with duplicate join fields are presented when the JLF is read.

The format for this keyword is:

JDUPSEQ (Sequencing field-name [*DESCEND])

- ✔ This keyword has no effect on the ordering of records with unique keys.
- ✔ If *DESCEND is not specified then the default is sequencing in ascending order.

JDFTVAL:

- ✔ When this file-level keyword is used the system provides default values for all for fields when a join to a secondary file does not produce any records.

✔ If this keyword is not specified a record in the primary file for which there is no corresponding record in the secondary file is skipped.

1. **What are the different between non-join logical files and join logical files.**

Non join logical file	Join logical file
We can able to insert or delete or update records using non-logical file.	Insertion, updating or deletion of records is not possible in join logical files.
DFU can be used to display non-join logical file.	DFU is not available
1-32 record format is specified	Only one record format can be specified
Commitment control is used	Commitment control cannot be used.

2. **How many record formats can have physical & logical file.**

- ✔ In physical file only one record format can be specified.
- ✔ In logical file 1-32 record formats are specified

1. **What is the advantage open query file?**
   - ✔ Dynamic selection of records
   - ✔ It will sort the records based on the field values.
   - ✔ We can retrieve records based on Virtual fields.
   - ✔ Can create join logical files

1. **Explain non-join logical file?**

   Non-join logical files can either be a simple logical file, which contains only one record format or a multiple record format logical file, which contains more than one record format.

   Logical files can be derived from 1 to 32 physical files and so a logical file can have a maximum of 32 record formats.

- ❖ **Single record format logical file:**

  If a logical file is derived from single physical file it is called simple logical file.

- ❖ **Multiple record format logical file:**

  Multiple record non-join logical files will select records from 2 or more physical files by referring to only one logical file.

  - ✔ Each record format is always associated with one or more physical file.
  - ✔ The same PF can be used in more than one record format.

- ❖ **Specify the entries in single or multiple format logical files:**

  1. File-level entries (optional): (REFACCPTH, DYNSLT)

REFACCPTH: The access path information for this logical file is to be copied from another PF or LF.

Format of the keyword is:

REFACCPTH (LIB name / DATABASE name)

DYNSLT: This keyword is selection and omission tests in the file. This keyword specifies dynamic select/omit.

2. Record – level entries :(PFILE)

PFILE: The physical files containing the data to be accessed through the

Record formats being defined.

Format of the keyword is:

PFILE (LIB name / PF name)

3. Field-level entries (optional)

4. Key field –level entries (optional)

5. Select and Omit –field level entries (optional)

REFACCPTH—it is applicable for non-join logical file only and meaning is referring the access path from the PF or LF in the file – level entries.

PFILE--- it is applicable only for non-join logical file in record level entries.

1. **It is possible to insert record to JOIN LF?**
   NO, insertion, updating or deleting of records is not possible in JOIN LF.

2. **Explain join logical file?**

A join-logical file is a logical file that combines two or more PF. In the record format not all the fields need to exist in all the PF.

- ✔ A PF cannot be changed through a JLF.
- ✔ DFU cannot be used to display a JLF.
- ✔ Only one record format can be specified in a JLF.
- ✔ Commitment control cannot be used with a JLF.
- ✔ Key fields must be fields defined in the join record format and must be fields from the PRIMARY FILE.

Specify the entries in join logical file:

1. File-level entries (optional): (JDFTVAL)

2. Record-level entries: (JFILE)

3. join-level entries :(JOIN, JFLD, JDUPSEQ)

4. Field –level entries (optional): (JREF, ALL, CONCAT, DYNSLT, RENAME, SST, TRNTBL)

5. Key field –level entries (optional)

6. Select and Omit field level entries. (Optional)

- ➢ JFILE----It is similar to indicate that this is a join logical field and it must have more than 2 physical files.
- ➢ JOIN: It is similar that this file level entries to be represent the position of the files .There must one primary file and can have more than I secondary files..
- ➢ JFLD: Which feels we are going to join.

- JREF: represents the primary file reference field
- JDFTVAL: represents that it as a left outer join.

SELECT OMIT

```
Columns . . . 1 71          Edit                  KSENTHILS/EXAMPLE
 SEU==> SELOMIT
 FMT PF.
        *************** Beginning of data *******************
0001.00              R RECSEL
0002.00                EMPNO         5P 0
0003.00                EMPNAME        20A
0004.00              K EMPNO
        ****************** End of data ************************
         EMPNO   EMPNAME
 000001 10,001   SHYAM
 000002 10,002   SANKA
 000003 10,003   SHYAM
 000004 10,004   SENTH
 000005 10,005   SANKA
 000006 10,006   SHYAM
 000007 10,007   SANKA
 000008 10,008   SENTH
 000009 10,009   SHYAM
 000010 10,010   SENTH
 ****** ******** End of report ********
Columns . . . 1 71          Edit                  KSENTHILS/EXAMPLE
```

```
SEU==> SELOMILF
FMT LF...
       *************** Beginning of data *********************
0001.00            R RECSEL               PFILE (SELOMIT)
0002.00            K EMPNO
0003.00            S EMPNO                CMP (GT 10003)
0004.00            O EMPNAME                CMP (EQ 'SHYAM')
       ****************** End of data **************************
                        Display Report
EMPNO   EMPNAME
 000001 10,002   SANKA
 000002 10,004   SENTH
 000003 10,005   SANKA
 000004 10,006   SHYAM
 000005 10,007   SANKA
 000006 10,008   SENTH
 000007 10,009   SHYAM
 000008 10,010   SENTH
 ****** ******** End of report ********
Columns . . . 1 71         Edit               KSENTHILS/EXAMPLE
 SEU==> SELOMILF
 FMT LF
       *************** Beginning of data ************************
0001.00            R RECSEL               PFILE (SELOMIT)
0002.00            K EMPNO
```

```
0003.00              O EMPNAME                CMP (EQ 'SHYAM')
0004.00              S EMPNO                  CMP (GT 10001)
        ****************** End of data ************************
EMPNO   EMPNAME
  000001 10,002   SANKA
  000002 10,004   SENTH
  000003 10,005   SANKA
  000004 10,007   SANKA
  000005 10,008   SENTH
  000006 10,010   SENTH
  ****** ******** End of report ********
 Columns . . . 1 71          Edit                 KSENTHILS/EXAMPLE
  SEU==> SELOMIT
  FMT PF.
        *************** Beginning of data *******************
 0001.00              R RECSEL
 0002.00                EMPNO        5P 0
 0003.00                EMPNAME       20A
 0004.00              K EMPNO
        ****************** End of data ************************
EMPNO   EMPNAME
  000001     20   SHYAM
  000002     30   RAM
  000003     40   TOM
  000004     50   RAMESH
  000005     60   SHYAM
  000006     70   SHYAM
  000007     80   TOM
  000008     90   TOM
  000009    100   VASU
  ****** ******** End of report ********
Columns . . . 1 71           Edit                  KSENTHILS/EXAMPLE
  SEU==> SELOMILF1
  FMT LF
        *************** Beginning of data ************************
0001.00               R RECSEL                  PFILE (SELOMIT1)
0002.00               K EMPNO
0003.00               S EMPNO                   CMP (GT 50)
0004.00               S EMPNAME                   VALUES ('SHYAM')
        ****************** End of data *************************
EMPNO   EMPNAME
  000001     20   SHYAM
  000002     60   SHYAM
  000003     70   SHYAM
  000004     80   TOM
  000005     90   TOM
  000006    100   VASU
  ****** ******** End of report ********
Columns . . . 1 71           Edit                  KSENTHILS/EXAMPLE
  SEU==> SELOMILF1
  FMT LF.
        *************** Beginning of data ************************
0001.00              R RECSEL                  PFILE (SELOMIT1)
0002.00              K EMPNO
0003.00              S EMPNO                   CMP (GT 50)
0004.00              O EMPNO                   RANGE (20 40)
        ****************** End of data ***************************
EMPNO   EMPNAME
  000001     50   RAMESH
  000002     60   SHYAM
  000003     70   SHYAM
  000004     80   TOM
  000005     90   TOM
  000006    100   VASU
  ****** ******** End of report ********
Columns . . . 1 71          Edit                   KSENTHILS/EXAMPLE
  SEU==> SELOMILF1
  FMT LF
        *************** Beginning of data ****************
0001.00              R RECSEL                  PFILE (SELOMIT1)
0002.00              K EMPNO
0003.00              S EMPNO                   CMP (GT 50)
0004.00              S EMPNAME                   VALUES ('SHYAM')
0005.00              O EMPNO                   RANGE (70 90)
        ****************** End of data ********************
```

```
EMPNO   EMPNAME
  000001     20   SHYAM
  000002     30   RAM
  000003     40   TOM
  000004     50   RAMESH
  000005     60   SHYAM
  000006     70   SHYAM
  000007     80   TOM
  000008     90   TOM
  000009    100   VASU
  ****** ******** End of report ********
 Columns . . . 1 71            Edit                    KSENTHILS/EXAMPLE
  SEU==> SELOMILF1
  FMT LF
        *************** Beginning of data *****************
 0001.00               R RECSEL                   PFILE (SELOMIT1)
 0002.00               K EMPNO
 0003.00               S EMPNO                    CMP (GT 50)
 0005.00               O EMPNO                     RANGE (70 90)
        ****************** End of data ************************
EMPNO   EMPNAME
  000001     20   SHYAM
  000002     30   RAM
  000003     40   TOM
  000004     50   RAMESH
```

```
000005     60   SHYAM
000006     70   SHYAM
000007     80   TOM
000008     90   TOM
000009    100   VASU
****** ******** End of report ********
```

1. Explain self join?

Joining a file to it-self is known as self-join.

(Or)

A physical file can be joined to itself to read records that are formed by combining two or more records from the PF itself.

```
          Columns . . . 1 71           Edit                  KSENTHILS/EXAMPLE
SEU==> SEJOIN
FMT PF
       *************** Beginning of data ********************
0001.00                R EMP
0002.00                  EMPID         5P 0
0003.00                  EMPNAME       20A
0004.00                  MGRID         5P 0
0005.00                K EMPID
       ****************** End of data ***************************
```

```
        EMPID   EMPNAME              MGRID
000001 10,001   SEBI JOSEPH C.        50,001
```

```
 000002 10,002   PURUSHOTTAM        50,002
 000003 10,003   SAMEER DIGHE        50,003
 000004 10,004   SHARATA              50,004
 000005 10,005   PAUL                    50,005
 000006 50,001   SHIVARAM             90,001
 000007 50,002   GAURAV                90,002
 000008 50,003   KING                     90,003
 000009 50,004   SAM                      90,004
 000010 50,005   ANIL                     90,005
 ****** ******** End of report ********
Columns . . . 1 71           Edit                  KSENTHILS/EXAMPLE
 SEU==> SELJOIN
 FMT LF
        *************** Beginning of data *************************
0001.00              R EMP                     JFILE (SEJOIN SEJOIN)
0002.00              J                               JOIN (1 2)
0003.00                                            JFLD (MGRID EMPID)
0004.00                EMPID                   JREF (1)
0005.00                EMPNAME             JREF (1)
0006.00                MANAGER            RENAME (EMPNAME) JREF (2)
0007.00                                             COLHDG ('MANAGER')
        ****************** End of data *****************************
         EMPID   EMPNAME              MANAGER
 000001 10,001   SEBI JOSEPH C.       SHIVARAM
 000002 10,002   PURUSHOTTAM          GAURAV
 000003 10,003   SAMEER DIGHE          KING
 000004 10,004   SHARATA                SAM
 000005 10,005   PAUL                    ANIL
 ****** ******** End of report ********
Columns . . . 1 71           Edit                   KSENTHILS/EXAMPLE
 SEU==> SELJOIN1
 FMT LF
               ***************  Beginning  of  data************************0001.00
R EMP                      JFILE (SEJOIN SEJOIN)
0002.00                J                                JOIN (1 2)
0003.00                                                  JFLD (MGRID EMPID)
0004.00                  EMPID                   JREF (1)
0005.00                  EMPNAME                   RENAME (EMPNAME)
0006.00                                                         JREF (1)
0007.00                  MGRID                          JREF (2)
        ****************** End of data *************************
EMPID   EMPNAME               MGRID
  000001 10,001   SEBI JOSEPH C.        90,001
  000002 10,002   PURUSHOTTAM           90,002
  000003 10,003   SAMEER DIGHE          90,003
  000004 10,004   SHARATA                 90,004
  000005 10,005   PAUL                       90,005
  ****** ******** End of report ********
Columns . . . 1 71           Edit                   KSENTHILS/EXAMPLE
 SEU==> SELJOIN1
 FMT LF
```

```
        *************** Beginning of data************************
    0001.00          R EMP              JFILE (SEJOIN SEJOIN)
    0002.00          J                     JOIN (1 2)
    0003.00                                JFLD (MGRID EMPID)
    0004.00            EMPID            JREF (1)
    0005.00            MANAGER          RENAME (EMPNAME)
    0006.00                                JREF (2) COLHDG ('MANAGER')
    0007.00            MGRID             JREF (2)
        ****************** End of data ****************************
    EMPID   MANAGER            MGRID
     000001 10,001   SHIVARAM          90,001
     000002 10,002   GAURAV             90,002
     000003 10,003   KING                  90,003
     000004 10,004   SAM                   90,004
     000005 10,005   ANIL                   90,005
     ****** ******** End of report ********
```

2. **Explain normalization?**

It is the process of segregating and decomposing information held within a system into logically grouped, related. Uniquely identifiable entities

3. **Explain the command ADDPFCST?**

ADDPFCST is a command that is used to define the Constraint on your physical file. The constraint has several types. These are REFCST, UNQCST and PRIKEY. By the by, this command is helps to define Update rules and Delete rules.

4. **How to send the message to the screen SNDPGMMSG?**

BY passing unique message ID message data and message file.

SNDPGMMSG syntax

```
SNDPGM MSG        MSGID (MSG0001)  MSGF (MSGSUB)
```

5. **How you can list all the LF of a PF?**

By using DSPDBR command it is to list all the files, which are related to a PF. It displays all the LF that is referring the PF and also lists the child table if it is having a relation through ADDPFCST.

6. **What is use of DSPFFD and DSPFD?**

❖ DSPFD (display file description)
  - ✔ It is used to display the details about the file when it is created.

❖ DSPFFD (display file field description)
  - ✔ It is used for listing details about individual fields.

1. **Explain inner join or natural join and left outer join?**

❖ Inner join

Inner join means the matching records in between the joining file will be selected.

```
Columns . . . 1 71          Edit                  KSENTHILS/EXAMPLE

 SEU==> JPF01

 FMT PF

        *************** Beginning of data ****************************

0001.00              R JP1REC

0002.00                EMPNO         5S 0

0003.00                EMPNAME       20A

0004.00              K EMPNO

        ****************** End of data *****************************

Columns . . . 1 71          Edit                  KSENTHILS/EXAMPLE

 SEU==> JPF02

 FMT PF.
```

```
        *************** Beginning of data ***************************
0001.00              R JP2REC
0002.00                EMPNO         5S 0
0003.00                EMPSAL       10P 2
0004.00              K EMPNO
        ****************** End of data ********************************
```

```
                         Display Report
EMPNO   EMPNAME
  000001 1,001   SHYAMBABU
  000002 1,002   SENTHILKUMAR
  000003 1,003   RAMESH
  ****** ******** End of report  ********
```

```
EMPNO         EMPSAL
  000001 1,001         100.00
  000002 1,002          20.00
  000003 1,004         300.00
  ****** ******** End of report  ********
```

```
Columns . . . 1 71           Edit                    KSENTHILS/EXAMPLE
 SEU==> JOFILE
 FMT LF.
        *************** Beginning of data ***********************
0001.00              R JREC1                   JFILE (JPF01 JPF02)
0002.00              J                             JOIN (1 2)
0003.00                                           JFLD (EMPNO EMPNO)
0004.00                EMPNO                   JREF (JPF01)
0005.00                EMPNAME
```

```
0006.00                  EMPSAL
0007.00                K EMPNO
        ****************** End of data*********************
EMPNO   EMPNAME                   EMPSAL
  000001 1,001   SHYAMBABU                   100.00
  000002 1,002   SENTHILKUMAR                  20.00
  ****** ******** End of report  ********
```

Left outer join

Left outer join all the records from primary file and matching records from the secondary file will be selected.

```
Columns . . . 1 71           Edit                    KSENTHILS/EXAMPLE
 SEU==> JOOUT
 FMT LF.
     *************** Beginning of data*********************
0001.00                                       JDFTVAL
0002.00                R JREC1                     JFILE (JPF01 JPF02)
0003.00                J                      JOIN (1 2)
0004.00                                       JFLD (EMPNO EMPNO)
0005.00                  EMPNO                     JREF (JPF01)
0006.00                  EMPNAME
0007.00                  EMPSAL
0008.00                K EMPNO
        ****************** End of data********************
EMPNO   EMPNAME                   EMPSAL
  000001 1,001   SHYAMBABU                   100.00
  000002 1,002   SENTHILKUMAR                  20.00
 000003 1,003   RAMESH                          .00
                    ****** ******** End of report  ********
```

SEQUENCING DUPLICATE RECORDS

```
Columns . . . 1 71           Edit                  KSENTHILS/EXAMPLE
 SEU==> SEQ1
 FMT PF.
        *************** Beginning of data ***********************
0001.00                R SEQREC1
0002.00                  EMPNO          5P 0
0003.00                  EMPNAME1       20A
0004.00                  ADDRESS        20A
0005.00                K EMPNO
        ****************** End of data **************************
Columns . . . 1 71           Edit                  KSENTHILS/EXAMPLE
 SEU==> SEQ2
 FMT PF
        *************** Beginning of data****************
0001.00                R SEQREC2
0002.00                  EMPNO          5P 0
0003.00                  EMPNAME        20A
0004.00                  TEL         10P 0
        ****************** End of data ******************

EMPNO   EMPNAME1              ADDRESS
 000001 10,001   BOB                  23,OLD MADIWALA
```

```
000002 10,002   DANNY              50,LONG ISLAND
000003 10,003   PRINC              90,ATTUR
****** ******** End of report  ********
  EMPNO   EMPNAME                      TEL
000001 10,001   BOB                    825,777
000002 10,001   BOB                    825,999
000003 10,001   BOB                    825,888
000004 10,002   DANNY                  4,222,600
****** ******** End of report  ********
Columns . . . 1 71           Edit                 KSENTHILS/EXAMPLE
 SEU==> JDFTSEQ
 FMT LF
       *************** Beginning of data ******************
0001.00             R RECSEQ                 JFILE (SEQ1 SEQ2)
0002.00             J                        JOIN (1 2)
0003.00                                      JFLD (EMPNAME1 EMPNAME)
0004.00                                      JDUPSEQ (TEL)
0005.00               EMPNO                  JREF (2)
0006.00               EMPNAME1
0007.00               ADDRESS
0008.00               TEL
       ****************** End of data************************
       EMPNO   EMPNAME1            ADDRESS                        TEL
000001 10,001   BOB              23,OLD MADIWALA           825,777
000002 10,001   BOB              23,OLD MADIWALA           825,888
```

```
000003 10,001   BOB                23,OLD MADIWALA          825,999
000004 10,002   DANNY                50,LONG ISLAND          4,222,600
****** ******** End of report  ********
```

JOIN DESCEND

```
Columns . . . 1 71           Edit                  KSENTHILS/EXAMPLE
 SEU==> JDFTSEQ
 FMT LF
       *************** Beginning of data **********************
0001.00                R RECSEQ                 JFILE (SEQ1 SEQ2)
0002.00                J                        JOIN (1 2)
0003.00                                         JFLD (EMPNAME1 EMPNAME)
0004.00                                         JDUPSEQ (TEL *DESCEND)
0005.00                  EMPNO                  JREF (2)
0006.00                  EMPNAME1
0007.00                  ADDRESS
0008.00                  TEL
       ****************** End of data ************************
```

```
  EMPNO   EMPNAME1            ADDRESS                    TEL
000001 10,001   BOB                23,OLD MADIWALA          825,999
000002 10,001   BOB                23,OLD MADIWALA          825,888
000003 10,001   BOB                23,OLD MADIWALA          825,777
000004 10,002   DANNY                50,LONG ISLAND          4,222,600
****** ******** End of report  ********
```

1. **How to create a trigger in AS/400?**

The trigger is an event to be performing before or after any change to a database. When a trigger is added to a physical file, three attributes need to be defined.

- ✔ The first is the event that will cause the trigger to fire. A trigger event can be an insert, an update, or a delete a record from the file.
- ✔ The second attribute to define is when to fire the trigger-before or after the event.
- ✔ The third attribute to define is the identification of the trigger program to by run.

We can infer that up to six triggers can be defined for each PF.

- ✔ For each update, insert, and delete
- ✔ Two triggers can be defined
  - ➢ One that runs before the event
  - ➢ One that runs after the event

These trigger are added using the ADDPFTRG

- ✔ Can be removed with the remove PF trigger command (RMVPFTRG)

The command ADDPFTRG takes F4

PF	PF001	
LIB	SKANDASAMO	
TRIGGER TIME	*AFTER	*BEFORE *AFTER
TRIGGER EVEVT	*UPDATE	*UPDATE *INSERT *DELETE
PROGRAM	PGM001	NAME
LIB	SKANDASAMO	
REPLACE TRIGGER	*NO	*NO *YES
ALLOW REPEATED	*NO	*NO *YES

1. How will be establishing REFERENTIAL INTEGRITY in as/400 systems?

- ❖ By using ADDPFCST command establish in as/400 system.
- ❖ Referential integrity concepts
  - ➢ Referential constraint
  - ➢ Parent and dependent files
  - ➢ Unique key and primary keys
  - ➢ Parent and foreign keys
  - ➢ Delete rule
    Propagate delete from parent file to dependent file

    Restrict delete on parent file
  - ➢ Update rule
    Restrict inconsistent updates

- Insert operations on dependent file are checked

- Referential constraint
  - ✔ A referential constraint is a relation between two files, the parent file and the dependent file.
  - ✔ This relationship establishes that every record in the dependent file has to have a matching record in the parent file.
  - ✔ The key value of any record in the dependent file must match a key value in the parent file.
  - ✔ We call parent key the key of the parent file and foreign key the key in the dependent file.
  - ✔ The parent key has to be unique and cannot contain null values.
  - ✔ The foreign key value has to match one (any only one) value of the parent key. Otherwise the foreign key can contain a null value.
  - ✔ A record in the parent file may be related to multiple records in the dependent files; each record in the dependent file has to have just one" parent" or have a null foreign key.
  - ✔ Primary key we means a unique and ascending key, which is the primary, access path for a PF and cannot contain null values.
  - ✔ Primary access path for a database file on the AS/400 in the access path used to access the file by using OPNDBF command.
- Delete rule

  We can delete a record from parent file or dependent file first check for delete rule.

  - ✔ CASCADE: If we want to delete a record from the parent file and its parent key is matching records in a dependent file, the DBMS will delete all the matching records of the dependent file.
  - ✔ SETNULL: If we delete a record from the parent file and parent key is matching some records in a dependent file, the DBMS will set to null the matching keys in the dependent file.
  - ✔ SETDEFAULT: This is like previous case, but matching occurrences in the foreign key are set to their default values. The default value for the foreign key has to match a record in the parent file.
  - ✔ RESTRICT: The DBMS will prevent any attempt to delete records in the parent file if its key is matching some records in the dependent file.
  - ✔ NO ACTION: This has the same meaning as restrict, but different timing. When we use *NOACTION and an invalid delete operation is about to take place, DB2 /400 will delay any error message until the end of the operation itself, allowing for instance the activation of a before trigger attached to the PF.
    - If *restrict is in use, the exception message is sent immediately.
    - Deleting records in a dependent file is always permitted.
- Update rule

- ✔ RESTRICT: We cannot change the value in a parent key if the old values are matching some records in the dependent file.
    - ➢ The remaining portion of the record can always be updated.
    - ➢ We cannot update a foreign key in a dependent file if the new value for the key is not null and does not match any value of the parent key.
- ✔ NO ACTION: This is same as * restrict but with different timing considerations. Refer above; where we describe no action delete operations.

❖ Inserts

- ✔ There is no insert rule to be chosen, but referential integrity prevents any insert in the dependent file if the new record has no match in the parent file and its foreign key is not null.

1. **What RUNSQLSTM will do?**

- ✔ If we want to execute set of SQL statement then we can write all the SQL statement to be including with the source.
- ✔ Type as SQL
- ✔ Only insert, update and delete and no select is allowed.
- ✔ Then use STRSQLSTM to execute the program
- ✔ RUNSQLSTM SRCFILE (LIB/TEST) SRCMBR (SQL01) COMMIT (*NONE)

1. **What is a field reference file?**
   This is PF, which does not have any data and contains only the field descriptions and these fields are referred in other PF by using REF and REFFLD
2. **What are the various ways creating access path?**

Access path means the records are to be retrieved from a file. The records can be retrieved from a PF or a LF either ARRIVAL SEQUENCE or by KEYED SEQUENCE. For LF you can also select records using select or omit keywords

Arrival sequence access path

- ✔ Sequentially, where each record is taken from the next sequential physical position in the file.
- ✔ Directly by relative records number, where the record number is identified by its position from the start of the file.

Keyed sequence access path

It is based on the contents of the key fields as defined in DDS. This type of access path is updated in the contents of a key field is changed.

There are three ways of bounding the access path

- **Immediate**
  Access path is always maintained for every opening in a file.
- **Rebuild**
  Access path is maintained when the file is open and various updates are recorded and the access path is rebuilt every time when the access path is closed.
- **Delayed**
  Access path is maintained when the file is opened and updates recorded. When the file is closed all the updates to the records are closed together but it is not rebuild. When the recorded update percentage exceeds 25% then rebuild of records take place.

1. **How many record formats PF, LF, DSPF and SFL?**
   PF- 1, LF- 32, DSPF-1024 SFL-512
2. **Define KLIST?**
   KLIST operation is a declarative operation that gives a name to list of KFLD. This KLIST can be used as a search argument to retrieve records from files that have a composite key.
3. **Define PLIST?**
   The declarative PLIST operation defines symbolic name for a parameter list to be specified in a CALL operation.
4. **Define composite key?**
   It is a key for a file, which is composed of more than one field.
5. **Is it possible to create a logical file whose Physical file is not in same library?**

Yes...for that u have to specify the actual data path in keyword PFILE
Like

```
   A    R A1      PFILE (AMIT/A2)
   A    K CODE
```

Then this will create the LF A1 for A2 PF...
Yes, just put the user library in which you want to create on top and followed by other library in which physical file is located.
Provided the first library should not have physical file.
Then try to create the logical.

6. **Can you delete the record space permanently in PF through CL?**

USING THE COMMAND 'RGZPFM' The Reorganize Physical File Member (RGZPFM) command compresses (removes deleted records from) one member of a physical file in the database, and it optionally reorganizes that member.

7. **What is the difference between adding keys & constraints into a physical file?**

During the time of compilation of a file, constrains are being removed, from the file .but at the time of compilation keys are not removed in case from pf.
Constraints has no affects on members of a pf but keys has affect on members of pf
A key field defines the order in which the records of a PF member will be sorted (ascending or descending).
On the other hand, a constraint, such as UNIQUE, defines which key field values or records are valid /**allowable** to be written into a file. UNIQUE prevents a record with duplicate key field values to be written into a PF.

8. **How to insert more than one record to a pf at a time? (Bulk insert to a pf)**

Bulk insert can be done in ways
1. Using SQL we can run a query
   Insert into table1 values ('a', 'b', 'c'), ('x', 'y', 'z')
2. Using STRDFU we insert records into PF
3. Using DBU (Database utility) also we can insert records

9. **How to see number of logical files depending on a pf? Can we declare more than 20 logical files from a single pf? Is it possible?**

1. We can see the number of logical files in the following ways

a. DSPDBR (filename) command

b. Using DBU (Filename) and Shift + F2

There is one more way to see the logical USING DBU is just type 'DBR' and press enter.

2. Yes we can declare more than 20 files on a physical file

10. **I want to change the attribute of field or want to add new field in existing PF but condition is format level identifier should not change, is it possible?**

Whenever a file is changed (by adding or deleting a field), Format level identifier is also changed, even if u are using CHGPF. All the programs that are this file needs to be compiled again. You can avoid recompiling programs by setting Format Level Check parameter to *NO in command CRTPF. But this is not a good method, since it may affect the program.

11. **Maximum how many fields we can create under a record format of PF?**

We can give max of 8000 fields in rec format of a PF. But it also depends on the no. of bytes occupied by the record format. Because rec. format of PF may occupy at max 32766 bytes.ie if there is only one field which occupy 32766 bytes then we can't define a new field in record format. So it depends on the no. of bytes occupied by the field defined in the record format.

Ex. Char field - max value (32766 bytes)

Variable length field - max value (32740 bytes)

Allow null field - max value (32765)

Variable & allow null field - max value (32739)

12.How can we write LF using flat file?

BLDINDEX MLR?WS?,28,8,ML.RETUR,,DUPKEY

BLDINDEX ML.HIS?WS?,9,6,ML.INVHS,,DUPKEY,,17,5

TYPE BLDINDEX in system 36 environment & F4

```
 BLDINDEX PROCEDURE                  Optional-*

                                     Ignored-%

Creates an alternate index for a physical file

Name of file to be CREATED . .              NIROSH

 Start posit for 1st field of key 1-4096    6

  Length of first field . .  . . . 1-120    5

 Start posit for 2nd field of key 1-4096    1

 Length of 2nd field            .  1-120    5

 Start posit for 3rd field of key  1-4096

 Length of 3RD field               1-120

 Name of PHYSICAL FILE              . . .    ML.CRDR

 Creation date of physical file

 Allow duplicate keys DUPKEY,NODUPKEY        DUPKEY

 Prefer disk locn A1,A2,A3,A4,block number
```

13.Why we create the Physical File Member?

MEMBER IS USE TO STORE THE DATA, EACH FILE MUST HAVE MEMBER.

14.CHGPF to compile the PF without using the data:

You have added some fields to a physical file that has lots of records. You don't want to lose the data, but you have to compile that to include those fields. In that case, make use of CHGPF.

```
CHGPF FILE (SHAILESH/SFL001PF) SRCFILE (SHAILESH/TESTPGMS)
```

It gives you the result similar compilation (including the added fields) without losing the data. Here SFL001PF is the PF and TESTPGMS is the source physical file.

Note: CHGPF cannot be used to achieve the result similar to compilation in case of deletion of some fields. You have to compile the PF to exclude the deleted fields.

In SQL there is a command called ALTER which performs similar functions (like ADD constraint, DROP constraint, ADD field, ALTER field and DROP field) as CHGPF.

But here also we get error message when we try to alter or delete the fields just like the CHGPF scenario.

15.Multi format Logical file Example:

Here is an example of a multi-format logical file:

```
A       R FORMAT1           PFILE (FILE1)
A         FIELD1
A         FIELD2
A       K FIELD1
A       R FORMAT2           PFILE (FILE2)
A         FIELD1
A         FIELD2
A       K FIELD1
```

In the program, you can chain with the file name and you'll get records from both physical files, or the format name and you'll get records only from the specific PFILE. If you want to update or write a record with this logical file, you must use the format name.

Limitation: Even though logical files allow multiple record formats, we cannot have different record formats for the same physical file (as the record format of the LF has to be the same as PF). Record formats of many physical files can be present in one LF making it multi format LF. We can have up to 32 record formats in the same LF.

If you want to access the same PF with different set of keys you have to go for multiple simple logical files like the one shown below.

LF1:

```
A         R FORMAT1                PFILE (FILE1)

A           FIELD1

A           FIELD2

A         K FIELD1
```

LF2:

```
A         R FORMAT1                PFILE (FILE1)

A           FIELD2

A           FIELD3

A         K FIELD2
```

Here we want to access the PF, File1 with 2 sets of keys. Field1 & Field 2 is one set and Field 2 & Field 3 is the other set. But we cannot mention 2 record formats with these keys in the same logical file. We have to write 2 separate logical files LF1 and LF2 in order to have 2 different record formats for that PF, File1.

16.Access Path – PF and LF

In a program, a record is deleted from a physical file. In the same program, the file is read again (the file is not closed and the delete operation is not committed as well). In that case, we will be accessing the deleted record also, since the refresh has not taken place on that PF. Only after Commit (in case the file is not user open only commit or ending the program can refresh the file) or explicit close, data refresh happens. It will not free up the space though. The record will be empty but will occupy space. Only RGZPFM can free up the space occupied by the deleted record.

But in the case of LF, refresh happens immediately if we associate them with the keywords that can rebuild/refresh the access path. Say for example, we have *IMMED for rebuild, as soon as the operation happens the file will get refreshed. It will not wait for the commit or close or program ending to happen.

Note: After deleting record 2, and not issuing RGZPFM, if we are trying to access the PF with RRN as key starting from the first record, the deleted record 2 will not be processed even though the space is still occupied (i.e., it is not treated as a data record). The compiler treats it as empty space and skips the record.

17. Tell me the differences between DB2 CLI (call Level Interface) and embedded SQL?

An application that uses an embedded SQL interface requires a pre-compiler to convert the SQL statements into code, which is then compiled, bound to the database, and executed. In contrast, a DB2 CLI application does not require pre-compilation or binding, but instead uses a standard set of functions to execute SQL statements and related services at runtime. This difference is important because, traditionally, pre-compilers have been specific to a database product, which effectively ties your applications to that product. DB2 CLI enables you to write portable applications that are independent of any particular database product. This independence means a DB2 CLI application does not have to be recompiled or rebound to access different database products, but rather selects the appropriate one at runtime. DB2 CLI can execute any SQL statement that can be prepared dynamically in embedded SQL. This is guaranteed because DB2 CLI doesn't actually execute the SQL statement itself, but passes it to the DBMS for dynamic execution.

18. General points in DB2/400

1. DB2/400 is an integrated RDBMS.
2. The major parts of a file are Record format and Access path.
3. Record format in a file describes the way the data is actually stored.
4. Access path describes the order in which the records are to be retrieved.
5. We have two types of access patch Keyes sequence access path and Arrival sequence access path.
6. Access Path maintenance specifies how access paths are maintained for closed files. While file is open the system maintains the access path changes.
7. In general we have three types of access path maintenance, *IMMED , *DLY and *REBLD
8. *IMMED must be specified for files that require UNIQUE keys. (Immediate access path maintenance mainly for files used as interactively)
9. *REBLD the access path is completely rebuilt each time a file member is opened. This cannot be specified for file if its access path is being journal.

10. *DLY the maintenance of the access path is delayed until the PF member is opened for use. Updates to the access path are collected from the time the member is closed until it is opened again. When it is opened, only the collected changes are merged into the access path.

11. In CHGPF command we have the keyword *LVLCHK (Level Check). This specifies whether the levels of record format identifiers are checked to verify that the current record format identifier is the same as that specified in the program that opens the physical file. The level identifiers of the record formats are checked when the file is opened. If the level identifiers do not match, an error message is sent to the program requesting the open, and the file is not opened.

12. For physical file and for logical the file type should be "FILE". Attribute for physical file should come as "PF-DATA", logical file come as "LF" and for source physical file attribute should come as "PF-SRC"

13. Let's say one scenario, your source physical file name is "QDDSSRC" and inside this source physical file when you try to create a physical file with name same as that of source physical file "QDDSSRC" what will happen? The source physical file will get deleted.

14. UNIQUE keyword is used in LF and PF to prevent duplicate key values.

15. DESCEND keyword is used to arrange records from the highest to the lowest key field values.

16. RANGE keyword is used to provide a range of valid values.

17. REF keyword specifies the name of file contains the referenced fields.

18. REFFLD keyword copies the field description from the referenced file.

19. FORMAT keyword shares the field description with the existing record format.

20. DFT keywords provide the default value for the fields.

21. CMP keyword provides a comparison value. Example   CMP(GE 0)

22. CHECK keyword provides validity checking.

23. COLHDG keyword provides column heading.

24. EDTCDE and EDTWRD keyword edit code and edit word.

25. TEXT keyword provides the description of record or field.

26. VALUES keyword provides a list of valid values.

27. A physical file can have only one record format.

28. A physical file cannot share the format of logical file.

29. For physical file the default maximum number of member is one, the maximum members we can attain is *NOMAX

30. For physical file the default value of initial number of records which we can add to database is 10000 and maximum increments in a physical file are three.

31. Maximum number of key fields allowed in a PF is 120. Max number of fields in a PF is 8000.

32. Members in physical file, the significance of member is it helps to classify data easier.

33. Example for member concept is, I want to keep data for a year in one file, and I frequently want to process data for one month at a time. For this I can create one physical file with twelve different members for each month.

34. We can copy data of a member to other member of same physical file. (We can't compile any new members)

35. Specific commands for physical file members, ADDPFM, CHGPFM, RMVM, INZPFM (*DFT or *DLT), RGZPFM, DSPPFM.

36. Even if you delete the records in a PF through program, still the space used by the deleted records not used by other purpose. Hence using RGZPFM command we can compress the deleted record space.

37. How we can add a new field to a file which has thousands of data in it? CHGPF command will help you to add new field without losing your current data.

38. We can add member to logical file with command ADDLFM.

39. DSPFD (Display file description) with TYPE *MBRLIST will display all members associated with the specified file. *TRG will give you the trigger program list.

40. An ODP (Open Data Path) is the path through which all input/output operations for the files are performed. It connects the program to a file. If we do not specify the SHARE (*YES) then a new data path is created every time a file is opened.

41. Logical file is used to arrange data from one or more PFs into different formats and sequences. Logical files contain no data. Three different types of

logical file we have, Simple logical file, Multiform at logical file and Join logical file. A multi format logical file is also known as Union file.

42. The record format name of logical file should/should not be same as that of physical file which mentioned in PFILE keyword. PFILE keyword tells the physical file that the logical file based on.

43. There are some keywords specific for LFs, CONCAT, DYNSLT, JFILE, JFLD, JOIN, JREF, PFILE, RENAME, and SST.

44. Column 17 values, R – Record format name, K- Key field name, J – Join specification, S – Select field name, O – Omit field name.

45. In LF the access path we can specify in three ways, keyed sequence access path, Arrival sequence access path and REFACCPTH.

46. In keyed sequence access path,

```
R CUSRCD             PFILE(PFNAME)
         K KEY1
                  K KEY2
```

47. In arrival sequence access path,

```
R CUSRCD              PFILE(PFNAME)
```

48. In REFACCPTH,(Reference access path)

```
                                  REFACCPTH(LIBRNAME/PFNAME)
                         R CUSRCD      PFILE(PFNAME)
```

49. SELECT/OMIT specification apply to logical file only.

50. In multi format logical file the maximum number of record format we can declare is 32.

51. Example of SELECT/OMIT

```
S    FIELD1                VALUES(123  345  567 566)
         S    FIELD2                RANGE(30122  30300)
                  S    FIELD3                CMP(GE  300)
```

52. We have two types of SELECT/OMIT, Access path SELECT/OMIT and Dynamic SELECT/OMIT. Access path SELECT/OMIT is maintained by system when records are added or changed. Dynamic SELECT/OMIT processing is done when records are read by the program.

53. For a multi format logical file if you want to access a particular member of a physical file. Then execute the OVRDBF command with the specific member name before reading the logical file.

54. Dynamic SELECT/OMIT, when a program reads a records from the file, the system only returns those records that meet the select/omit values. That is the actual SELECT/OMIT processing is done when records are done by the program, rather than records are added or changed.

55. DYNSLT is the keyword for dynamic select.

56. Another method of selecting records is using QRYSLT parameter on the open query file (OPNQRYF) command.

57. Join logical file is a logical file that combines fields from two or more physical file. Cannot change physical file using join logical file. Can specify only one record format in join logical file. The record format in join logical file cannot be shared. Commitment control cannot be used with join logical file.

58. We can join a physical file to itself.

59. Join logical files are Read only files.

60. In Join logical file maximum of 31 secondary files can join. (Total 32 files.) Max of two files can be joined at a time.

61. Example of join logical file,

```
                    R           JOINREC         JFILE(PF1 PF2)
                          J                           JOIN(PF1 PF2)
                                                               JFLD(NBR  NBR)
                                                   NBR               JREF(PF1)
                                                         NAME
                                                                SALARY
                                                            K              NBR
```

62. At least two physical files must be specified on JFILE keyword. The first file specified on the JFILE is the primary file the other files are secondary files.

63. JOIN keyword identifies which two files are joined by the JOIN specification. If only two physical files are joined by the join logical file, then JOIN keyword is optional.

64. JFLD keyword identifies the join fields that join records from physical files specified in JOIN. JFLD must be specified at least once for each JOIN specification. Join fields except character type fields must have the same attribute.

65. JREF keyword specifies which physical file to use.

66. Example of join logical file, In this case the NBR filed comes from PF2, because relative file number is specified.

```
                         R   JOINREC          JFILE(PF1  PF2)
                              J                         JOIN(PF1 PF2)
                                                           JFLD(NBR
NBR)                                      NBR      S          JREF(2)
                                               NAME
                                                    SALARY
```

67. Reference file does not contain any data. This is a physical file. It is used as a reference for the field descriptions for other files.

68. OPNQRYF is like a temporary logical file. It will get automatically deleted when we close this query file. OPNQRYF command opens a file that contains a set of database records that satisfies a database query request. OPNQRYF never shares an existing shared ODP in the job or activation group.

69. OVRBDF command must be executed before executing OPNQRYF command.

70. We can copy data from data path opened by OPNQRYF by CPYFRMQRYF.

71. Parameters in OPNQRYF commands, **FILE** – name of file that get processed by OPNQRYF command. **OPTION** – open option used for query file example *INP, *OUT, *UPD, *DLT, *ALL. **QRYSLT** – query select specifies the selection value. **KEYFLD** - *ASCEND, *DESCEND. **UNIQUEKEY** – Specifies whether the query is restricted to records with unique key values.

72. Example for OPNQRYF Command,

```
                    OPNQRYF      FILE(ORDFILE)    OPTION(*ALL)
                              QRYSLT('ORDDATE=%RANGE("920101"
"920131") OR ORDAMT > 100')                        KEYFLD(ORDAMT
*DESCEND)
```

73. Difference between PF and LF

Physical File	**Logical File**
One Record Format	More than one record format
Record will loss, while compile	No record loss while compile
Contain actual data	Doesn't contain data but it provides view from PF
We can update data	We cannot update data

We can use REF keyword	We use REFACCPTH, DYNSLT
We can’t use SELECT/OMIT	We can use except in join logical file.

74. Difference between JLF and LF

Join Logical file	**Logical File**
One record format	One or more record format
Record format name should be different	Record format name should be same as of Physical file
Through JLF we cannot change PF	Through LF we can change PF.

75. Database management commands, DSPFFD, DSPFD, DSPDBR, and DSPPGMREF.

76. A trigger is a set of actions that are run automatically when a specified change operation is performed on a specified physical database file. The change operation can be an insert, update, or delete high level language statement in an application program.

77. On the AS/400 system, a set of trigger actions can be defined in any supported high level language. (C, Cobol. RPG, PLI, SQL)

78. ADDPFTRG command adds a trigger to call a named trigger program to a specified physical file. The trigger program can be specified to be called before or after a change operation occurs. The change operation can be insert, update or delete. A maximum of six triggers can be added to a physical file. Once a trigger is added to a physical file, all members of that specified file are affected by trigger. example, ADDPFTRG FILE (EMP) TRGTIME(*AFTER) TRGEVENT(*INSERT) PGM(LIB2/INSTRG)

79. The Remove Physical File Trigger (RMVPFTRG) command removes the association of files and trigger program.

80. A trigger program **cannot** include the following commands, COMMIT, ROLLBACK, and ENDCMTCTL.

81. A trigger program can call other programs or can be nested (that is, a statement in a trigger program causes the calling of another trigger program.) The maximum trigger nested level for insert and update is 200.
82. A double-byte character set (DBCS) is a character set that represents each character with 2 bytes. The DBCS supports national languages that contain a large number of unique characters or symbols (the maximum number of characters that can be represented with 1 byte is 256 characters). Examples of such languages include Japanese, Korean, and Chinese. There are two general kinds of DBCS data: bracketed-DBCS data and graphic (non-bracketed) DBCS data. **Bracketed-DBCS data** is preceded by a DBCS shift-out character and followed by a DBCS shift-in character. **Graphic-DBCS data** is not surrounded by shift-out and shift-in characters.
83. The DB2/400 database supports the following physical file constraints, Referential constraints, Primary key constraints and Unique constraints.
84. The Add Physical File Constraint (ADDPFCST) command adds all types of physical file constraints. To add unique and primary key constraints, specify *UNQCST for the Type parameter for a unique constraint and *PRIKEY for a primary key constraint. When adding a primary key constraint, the specified key becomes the file's primary access path.
85. The Remove Physical File Constraint (RMVPFCST) command removes a constraint.
86. Physical File Constraint Considerations and Limitations,
    - A file must be a physical file.
    - A file can have a maximum of one member, MAXMBR(1).
    - A constraint can be defined when the file has zero members. A constraint cannot be established until the file has one, and only one, member.
    - There is a maximum of 300 constraint relations per file.
    - Constraints cannot be added to files in the QTEMP library.

1. The CRTSRCPF command creates a physical file, but with attributes appropriate for source physical files. For example, the default record length for a source file is 92 (80 for the source data field, 6 for the source sequence number field, and 6 for the source date field).
2. DB2/400 Database Data Recovery, we have the following methods.

    Journal management, for recording data changes to files

    Commitment control, for synchronizing transaction recovery

    Force-writing data changes to auxiliary storage

    Abnormal system end recovery

3. SQL/400 will not support multi format logical file.

4. In UDB/400 (Universal Data Base) an SQL table is an single member physical file.
5. In UDB/400 an SQL view is a single member logical file.
6. In UDB/400 an SQL index is a single member logical file with keyed access.
7. SQL/400 refreshes records as rows and fields as columns.
8. The SQL related program variables, Known as the SQL communication area (SQLCA), you should code in the working storage section to have the SQL compiler generate SQLCA group and elementary items. EXEC SQL INCLUDE SQLCA END-EXEC. SQLCA works as an error handler.
9. SQLCODE value less than zero means error, Value “100” means no rows found.

1. File pointer – after a failed chain operation

When the CHAIN operation is not completed successfully (for example, an error occurs or no record is found), the file specified in factor 2 must be repositioned (for example, by a CHAIN or SETLL operation) before a subsequent read operation can be done on that file.

2. What are Triggers?

Triggers is a self contained set of transact executable statements which can be invoked during the operations such as insert, update, delete can be performed in a database file.
We can able to add trigger in a physical file: ADDPFTRG
To remove a trigger: RMVPFTRG
In the case of SQL/400 we can create trigger by....

```
>>STRSQL
>>CREATE TRIGGER TRG_Name after INSERT/DELETE/UPDATE On PF1
  Insert Into PF2 Values ('RECORDS DELETED IN PF1')
>>Delete FROM PF1 Where Recordno = 10
>>SElect * FROM PF2.
```

In the above code, the records in the PF1 is deleted , at once the trigger trg_name is fired and the given insert statement is activated in pf2.

3. What is the purpose of USROPN keyword?

The USROPN keyword causes the file not to be opened at program initialization. This gives the programmer control of the file's first open. The file must be explicitly opened using the OPEN operation.
For example, if a file is opened and later closed by the CLOSE operation, the programmer can reopen the file (using the OPEN operation) without having specified the USROPN keyword on the file description specification.

4. What is LEVEL CHECK?

Whenever PF is compiled, the system generates unique code for identifying the file for future reference.

When we compile the Program that uses the PF, it will use that unique code of the PF.
If we call the program, it runs successfully. But if we change the PF and recompile the PF, the system generates a new unique code for that PF. So, our program doesn’t have this unique code and hence terminates abnormally with a Level Check error.
The solution of Level Check error is whenever it happens we have to either compile the PF with Level Check parameter value *NO or we have to compile the program again.

OVRDBF

1. What exactly the OVRDBF does?

It can do a lot below are some basic examples

1) Say you have an internally defined file in your program.

```
OVRDBF     FILE (CR311H) TOFILE (CB.CR1H)
CALL CR1H
DLTOVR FILE (CR311H)
```

2) Say you want to use a different file (with the same record level) in a program.

```
OVRDBF     FILE (PAYROLL) TOFILE (MYPAYROLL)
CALL GIVERAISE
DLTOVR FILE (MYPAYROLL)
```

3) Say you want to use OpnQryF (ug hate old school).

```
OVRDBF     FILE(OCEANHDR)   TOFILE(OCEANHDR) SHARE(*YES)
OPNQRYF    FILE((OCEANHDR))                    +
           QRYSLT('HSTATUS ¬= "C" & HCUST = 08177')  +
           KEYFLD((HCUSTREF))
           QRYSLT('HCUST = 08177')  +
CALL       PGM(AB010R)
CLOF       OPNID(OCEANHDR)
DLTOVR     FILE(*ALL)
```

4) Say you have a specific member that you want to use

```
OvrDbf     File (TaRpt0103p) ToFile(TaRpt0103p) +
           OvrScope(*CallLvl)  Mbr(&PoMember)
CALL TaRpt0103
DltOvr     FILE(*ALL) LVL(*)
```

To be theoretical,
The OVRDBF is used to

1) To temporarily change the attributes of a file like member, position of rrn ,sharing the Open Data Path etc....
2) To Redirect the references made for one file to other file.

Actually OVRDBF is used for multimember concept. A pf having multiple member and u need to access one of the members from that, OVRDBF is used. Also, in CL u can access only 1 file at a time. For more than 1 file also OVRDBF is used

OPNQRYF

5. **What is the open query file?**

It is a dynamic record selection. The OPNQRYF command acts as a filter between the processing program and the database records. The database file can be a PF or LF. It will create open data pathway to access (retrieve) data file.

If you want to specify any SQL operation within a CL we have to use OPNQRYF

- ❖ Functions supplied by OPNQRYF are:
  - ✔ Dynamic record selection.
  - ✔ Dynamic keyed sequence access path
  - ✔ Dynamic keyed sequence access path over a join
  - ✔ Dynamic join
  - ✔ Handling missing records in secondary join files
  - ✔ Unique-key processing
  - ✔ Mapped field definitions
  - ✔ Group processing
  - ✔ Final total-only processing
  - ✔ Improving performance
  - ✔ Open query identifier (ID)

1. **What is the different between OPNQRYF and SQLRPG?**

OPNQRYF	**SQLRPG**
OPNQRYF will come along with OS/400 system and no need to have any additional package needed to execute it	We need to have SQLRPG installed in as/400 system which involves additional cost to the programmers
OPNQRYF is faster as compared to SQLRPG	It is slower

OPNQRYF is nothing but a dynamic logical files will be created and the records	SQLRPG is imbedding SQL statements directly within SQL statement

2. What are the various steps in creating OPNQRYF?

Totally five steps involved in creating OPNQRYF

- ❖ OVRDBF
  - ✔ FILE (file PF) TOFILE (LIB/ PF) SHARE (*Yes)
    - ➢ If a PF is having 100 records and if we want to override the PF so that it continues only the specific number of records we are using OVRDBF
- ❖ OPNQRYF
  - ✔ FILE (LIB / PF) QRYSLT (‘EMPNO *EQ ‘ *BCAT &A)
    - ➢ If you want to perform any SQL operation we have to declare in OPNQRYF command only.
    - ➢ In case of OPNQRYF we can perform expression only based on characters but not on numeric.
  - ✔ *BCAT

    If you want to perform any charter expression are using *BCAT expression which will provide a blanks in between the 2 variables.
  - ✔ %WLDCRD
    - ➢ It is similar to %LIKE in SQL
    - ➢ QRYSLT (‘EMPNAME *EQ %WLDCRD (“S* “)’)

      It will fetch all the records whose empname starts from S.
  - ✔ *CT
    - ➢ It will fetch all the records, which conditions the particular charter.
    - ➢ QRYSLT (‘EMPNAME *CT “S” ‘)
  - ✔ %RANGE
    - ➢ It will fetch the records within the specific range
    - ➢ QRYSLT (‘EMPNO *Eq %RANGE (100 110)’)
- ❖ CALL PGM (LIB/NAME) PARM ()
- ❖ DLTOVR
  - ✔ As we see early the main file logically overridden and after performing the necessary operation, we have to delete the logical file so that the main file contains the actual records for this DLTOVR will be used.
  - ✔ DLTOVR   FILE (OPNPF)
- ❖ CLOF
  - ✔ We have to close the file, which has been opened

- ✔ CLOF OPNID (OPNPF)

You will copy overridden file records using CPYFRMQRYF

- ❖ CPYFRMQRYF
  - ✔ Since OVRDBF is logical we cannot able to list the variables, which satisfy the query condition. To see the records being selected we have to copy from the source file to a temporary file for this CPYFRMQRYF will be used
  - ✔ CPYFRMQRYF FROMOPNID (OPNPF) TOFILE (LIB/NAME) MBR (*REPLACE) CRTFILE (*YES) FMTOPT (*NOCHK)
- ❖ RUNQRY
  - ✔ We have copied the contents satisfy the query into a temporary file using CPYFRMQRYF. If we run the destination file we got the actual records, which satisfy the query.
  - ✔ RUNQRY QRYFILE (LIN/NAME)

Example:

Database PF

```
                                          SKANDASAMO/CLP
                                                 OPENF
     *************** Beginning of data ****************************
0000.01 C                          UNIQUE
0001.00 C         R OPNQFILE
0002.00 C           OEMPNO      5S 0
0003.00 C           OEMPNAME     20A
0004.00 C           OADDRESS     20A
0005.00 C           ODOB        8S 0
0006.00 C         K OEMPNO
     ****************** End of data ********************************
```

Data file

```
                    Display Report
     OEMPNO   OEMPNAME          OADDRESS              ODOB
000001 1,001   SENTHIL           SALEM1             1,232,002
```

```
000002 1,002   KUMAR              TRICHY              12,123,000
000003 1,003   SHYAM              SALEM               12,345,000
000004 1,004   RAMESH              SALEM                1,010,100
000005 1,005   BALU               SALEM                      222
000006 1,007   KUMAR              JJ                   32,938
****** ******** End of report  ********
```

CL program

```
                                         SKANDASAMO/CLP
                                          OPNQFILE5
   *************** Beginning of data ********************************
0001.00 PGM
0002.00          DCL       VAR(&A) TYPE(*CHAR) LEN(5)
0003.00          DCLF      FILE(SKANDASAMO/OPNQFILE3) RCDFMT(OPNF3)
0004.00          SNDRCVF    RCDFMT(OPNF3)
0005.00          CHGVAR     VAR(&A) VALUE(&OEMPNO)
0006.00          OVRDBF     FILE(OPENF) SHARE(*YES)
0006.01                   OPNQRYF        FILE((SKANDASAMO/OPENF))
QRYSLT('OEMPNO *EQ' +
0006.02                *BCAT &A)
0006.03                              CPYFRMQRYF    FROMOPNID(OPENF)
TOFILE(SKANDASAMO/TEMP) +
0006.04                MBROPT(*REPLACE) CRTFILE(*YES)
0009.00          DLTOVR     FILE(OPENF)
0010.00          CLOF       OPNID(OPENF)
0010.01          RUNQRY     QRYFILE((SKANDASAMO/TEMP))
0011.00          ENDPGM
```

```
                  ****************** End of data*********************************

OUTPUT

            EMPLOYEE NUMBER: 1001

                  OEMPNO   OEMPNAME            OADDRESS                  ODOB

           000001 1,001   SENTHIL             SALEM1               1,232,002

           ****** ******** End of report  ********
```

1. **How the records are accessed for using OPNQRYF?**
   By creating open data pathway to access (retrieve) data file.
2. **What is the difference between FMTDTA and OPNQRYF?**

FMTDTA	OPNQRY
It will sort the records sequentially based on the position of the record	It will sort the records based on the field values.
If any change in the attribute size of a PF then we have to change the program specification also.	If there is any change in the attribute size it will not affect the program specification also.
FMTDTA is bit faster in process than OPNQRYF.	OPNQRYF is slower as compare to FMTDTA if we are processing millions of records.

3. **List out the Differences between a LF and command OPNQRYF?**

LF creates a new object in the system while that is not the case for OPNQRYF.LF creates a permanent data access path to the physical file that will be updated as and when and add, update and delete operation is performed on file Whereas OPNQRYF creates a temporary access data path that is shared by high level pgm for further processing of recs in file.

I agree with Vaiv20. Just want to add that OPNQRYF is used with keyword Share (*Yes) and that's what makes the ODP available to high level pgms.
Also the usage of OPNQRYF is for adhoc jobs that are executed once in a while whereas LF is used in case where the ODP is going to be used pretty regularly. So LF object would be preferred when the usage is going to be regular.
OPNQRYF would be good where the job is going to be once in a while.

LF would make the job faster compared to OPNQRYF though it depends on what kind of maintenance option you use for LF.
The main difference is: Logical file creates permanent object on the system. OPNQRYF creates temporary access path.

4. OPNQRYF - Short explanation with samples in CLP

```
CL-PROGRAM
 FUNCTION.....: RETRIEVE/SELECT DATA WITH THE CL-COMMAND OPNQRYF.
                ILLUSTRATED IN SAMPLES 1 - 3.
 TASK.........: RETRIEVE USERINFO ON FIELDS FROM THE USRPRF-FILE.
                USE THE CL-COMMAND DSPUSRPRF *ALL OUTPUT(*OUTFILE)
                FILE(LIB/QRYSLTPF). LIB IS YOUR OWN TEST-LIBRARY.
                THE PROGRAM ONLY USES 2 FIELDS TO AVOID CONFUSION.
                IN THIS PROGRAM THE TEST-LIBRARY IS JPHLIB.

 INPUT........: DB-FILE:   QRYSLTPF
                FIELDS:    UPUPRF    10 A
                           UPUID     10 P0
 DISCLAIMER...: THE DATA RETRIEVED AND THE COMBINATION OF FIELDS
                IS COMPLETELY NONSENSE AND INTENDED ONLY TO ILLU-
                STRATE THE USE OF THIS COMMAND. CREATE BETTER EX-
                AMPLES ON YOUR OWN !!
 CREATE A FILE IN YOUR TESTLIBRARY CALLED BRUG. THIS IS THE FILE
 THAT RECIEVES OUTPUT-DATA FROM THE OPNQRYF.
 OUTPUT.......: DB-FILE:   BRUG
                FIELDS:    UPUPRF    10 A
```

```
                    UPUID     10 P0
             OPENID IN THE OPNQRYF-STATEMENT IS YOUR REFERENCE TO THE
             INTERNAL OPNQRYF-OUTPUT. USE THIS NAME AS FROM-FILE
             IN THE FINAL CPYFRMQRYF WHERE YOU RETRIEVE THE SELECTED
             DATA TO A PHYSICAL FILE.
 TIP..........: BE ABSOLUTELY SURE TO USE THE RIGHT NUMBER OF
             QUOTES ( ' ) WHEN YOU DEFINE THE SELECT-STATEMENT.
             ALL CHAR-VARIABLES IN THE OPNQRYF SELECT-LINE MUST
             BE EMBEDDED IN TRIPLE-QUOTES AND *CAT:
                ''' *CAT &CHARVAR *CAT '''             OR
                 "' *CAT &CHARVAR *CAT '"
            PGM
           DCL       VAR(&USER) TYPE(*CHAR) LEN(10)
           DCL       VAR(&NR) TYPE(*DEC) LEN(10 0)
           DCL       VAR(&EX) TYPE(*DEC) LEN(1 0)
           DCL       VAR(&NRALF) TYPE(*CHAR) LEN(10)
           DCL       VAR(&X) TYPE(*CHAR) LEN(1) +  VALUE(' ')
           DCLF       FILE(QRYSLTDF) RCDFMT(*ALL)
/* THE VAR X HELPS TO LEAVE THE LIBRARY-LIST UNCHANGED  */
/* WHEN THE PROGRAM HAS FINISHED PROCESSING.            */
           ADDLIBLE   LIB(JPHLIB)
           MONMSG     MSGID(CPF2103) EXEC(CHGVAR VAR(&X) VALUE('X'))
/* PROMPT FOR NAME OG ID AND TYPE OF EXAMPLE            */
 CHOISE:     SNDRCVF    RCDFMT(F0)
/* QRYSLT ONLY OPERATES WITH ALFA-VARIABLES. THE NUME-  */
/* RIC VAR. &NR IS CONVERTED TO CHAR. &NRALF              */
/* USED IN SAMPLE 3.                                       */
             CHGVAR     VAR(&NRALF) VALUE(&NR)
/* F3 WAS PRESSED ON THE SCREEN                            */
             IF         COND(&IN03 = '1') THEN(GOTO CMDLBL(END))
             IF         COND(&EX = 1) THEN(GOTO CMDLBL(ONE))
             IF         COND(&EX = 2) THEN(GOTO CMDLBL(TWO))
             IF         COND(&EX = 3) THEN(GOTO CMDLBL(THREE))
/*****************************************************************/
/* SAMPLE ONE: CHAR CONSTANT AND NUM CONSTANT                    */
/*****************************************************************/
ONE:
 OPNQRYF    FILE((JPHLIB/QRYSLTPF)) +
 QRYSLT(' +
(UPUPRF *EQ ''JPH'') +
*AND +
(UPUID *EQ 338) +
 ') +
OPNID(BRUG)
             GOTO       CMDLBL(OUT)
/*****************************************************************/
/* SAMPLE TWO: CHAR VARIABLE AND NUM CONSTANT                    */
/*****************************************************************/
TWO:
             OPNQRYF    FILE((JPHLIB/QRYSLTPF)) +
QRYSLT(' +
 (UPUPRF *EQ ''' *CAT &USER *CAT ''') +
 *AND +
 (UPUID *EQ 338) +
') +
 OPNID(BRUG)
             GOTO       CMDLBL(OUT)
/*****************************************************************/
/* SAMPLE THREE: CHAR VARIABLE OG NUM VARIABLE.                  */
/*                DIGITS IS A OPNQRYF KEYWORD THAT CONVERTS       */
/*                A FIELD FROM NUMERIC TO ALFA (CHAR.)            */
/*****************************************************************/
THREE:
             OPNQRYF    FILE((JPHLIB/QRYSLTPF)) +
                        QRYSLT(' +
                        (UPUPRF *EQ ''' *CAT &USER *CAT ''') +
                        *AND +
                        (%DIGITS(UPUID) *EQ ''' *CAT &NRALF *CAT ''') +
                        ') +
                        OPNID(BRUG)
             GOTO       CMDLBL(OUT)
/*****************************************************************/
/* MAKE A COPY OF THE OPNQRY OUTPUTFILE TO THE PF BRUG            */
/*****************************************************************/
OUT:         CPYFRMQRYF FROMOPNID(BRUG) TOFILE(JPHLIB/BRUG) +
                       MBROPT(*ADD)
             CLOF       OPNID(BRUG)
             DLTOVR     FILE(*ALL)
             GOTO       CMDLBL(CHOISE)
/* LIBRARY-LIST IS RESTORED                      */
END:       IF        COND(&X = ' ') THEN(RMVLIBLE LIB(JPHLIB))
             RCLRSC
             RETURN
             ENDPGM
DISPLAY-FILE:
    A                              DSPSIZ(24 80 *DS3)
    A         R F0
    A                              CF03(03 'Afslut')
    A                           1 69TIME
    A                           1 63'Time:'
    A                           1 30'Retrieve user and id'
    A                              DSPATR(HI)
    A                           2 69DATE
    A                              EDTCDE(Y)
    A                           2 63'Date:'
    A                           4 12'User name . . . . . :'
    A                              DSPATR(HI)
    A           PROMPT         1A  I 24  4DSPATR(ND)
    A                          24  6'Enter=Run      F3=Exit'
    A                              COLOR(BLU)
```

```
A            USER         10A  I  4 35
A                              6 12'User id . . . . . . :'
A                                 DSPATR(HI)
A            NR          10S 0I  6 35CHECK(FE)
A                                 CHECK(RZ)
A                              8 12'Sample  . . . . . . :'
A                                 DSPATR(HI)
A                             10  8'1 = CHAR CONSTANT og NUM CONSTANT'
A                             11  8'2 = CHAR VARIABLE og NUM CONSTANT'
A            EX           1S 0I  8 35RANGE(1 3)
A                             12  8'3 = CHAR VARIABLE og NUM VARIABLE'
```

PHYSICAL FILE BRUG:

```
 *****************************************************************
 *  PHYSICAL FILE BRUG TO HOLD SELECTED RECORDS IN OPNQRYF
 *****************************************************************
 *________________________________________________________________
A          R SELECT
 *________________________________________________________________
A            UPUPRF        10          TEXT('USER')
A                                 COLHDG('USER')
A            UPUID         10P 0       TEXT('ID')
A                                 COLHDG('ID')
 *________________________________________________________________
```

I hope this will help you understanding some of the basics in OPNQRYF.

However there is a small thing usually wrapped in plastic or hidden on a CD called a manual. This could be a great help but sometimes very hard to understand and with some stupid examples that don't work. My samples can be typed in on your AS/400 and it works.

Don't hesitate to send a mail if you want more help or want to discuss some of the above mentioned topics.

BR JPH and GL.

Here are some reference sites:

Database programmers guide

http://publib.boulder.ibm.com/iseries/v5r2/ic2924/info/dbp/rbafomst02.htm

http://publib.boulder.ibm.com/iseries/v5r2/ic2924/info/dbp/rbafomst199.htm#HDROPNQF

http://www.geocities.com/SiliconValley/Hills/6632/opnqryf.html

5. OPNQRY Example

http://publib.boulder.ibm.com/html/as400/v4r5/ic2979/info/db2/rbafomst140.htm#Header_199

TESTPF data:

EMPNO	EMPNAME	EMPPHONE
1	Agnie	1,234,567
2	Amudha	3,456,789

TESTPF1 data:

EMPNUM	EMPNAM	EMPADDR
1	Agnie	Coimbatore
3	Varun	Bangalore

Format of OPNQRYRES – Note that it contains the fields of TESTPF as well as TESTPF1

EMPNUM	EMPNAM	EMPPHONE	EMPADDR

SHAILESH/TESTPGMS/TSTOPNQRYF – Program to test OPNQRYF command

```
    PGM

/* To select the records present in TESTPF & TESTPF1 and copy that to a new file
(which contains all the fields of TESTPF and TESTPF1 */

              OPNQRYF    FILE((SHAILESH/TESTPF) (SHAILESH/TESTPF1)) +

                     FORMAT(SHAILESH/OPNQRYRES) +

                     JFLD((TESTPF/EMPNO TESTPF1/EMPNUM *EQ)) +

                     JDFTVAL(*NO)

           CPYFRMQRYF FROMOPNID(TESTPF) TOFILE(SHAILESH/OPNQRYRES) +

                     MBROPT(*REPLACE) CRTFILE(*YES) FMTOPT(*NOCHK)

           CLOF       OPNID(TESTPF)

/* To select the records present in TESTPF & not present in TESTPF1 */

           OPNQRYF    FILE((SHAILESH/TESTPF) (SHAILESH/TESTPF1)) +

                     FORMAT(SHAILESH/TESTPF) +

                     JFLD((TESTPF/EMPNO TESTPF1/EMPNUM)) +

                     JDFTVAL(*ONLYDFT)

           CPYFRMQRYF FROMOPNID(TESTPF) TOFILE(QTEMP/RESULT) +

                     MBROPT(*REPLACE) CRTFILE(*YES) FMTOPT(*NOCHK)

           CLOF       OPNID(TESTPF)

/* To select the records present in TESTPF1 & not present in TESTPF */

           OPNQRYF    FILE((SHAILESH/TESTPF1) (SHAILESH/TESTPF)) +

                     FORMAT(SHAILESH/TESTPF1) +

                     JFLD((TESTPF1/EMPNUM TESTPF/EMPNO)) +

                     JDFTVAL(*ONLYDFT)

           CPYFRMQRYF FROMOPNID(TESTPF1) TOFILE(QTEMP/RESULT1) +

                     MBROPT(*REPLACE) CRTFILE(*YES) FMTOPT(*NOCHK)
```

```
            CLOF       OPNID(TESTPF1)
        ENDPGM
```

SQLRPGLE

6. **SQLRPGLE Example**

```
D PGMEMPNAME      S             25A
D EMPID           S              5S 0 INZ(00003)
D TSTEMPNAME      S                  LIKE(PGMEMPNAME)
C/EXEC SQL
C+ INSERT INTO SHAILESH/TESTPF VALUES(00003, 'Varun')
C/END-EXEC
C/EXEC SQL
C+ SELECT EMPNAME INTO :PGMEMPNAME FROM SHAILESH/TESTPF
C+ WHERE EMPNO = :EMPID
C/END-EXEC
C/EXEC SQL
C+ UPDATE SHAILESH/TESTPF SET EMPNAME = 'AGNIE'
C+ WHERE EMPNAME = 'AGNI'
C/EXEC SQL
C+ DELETE FROM SHAILESH/TESTPF WHERE EMPNO = 00003
C/END-EXEC
C                   EVAL      TSTEMPNAME = PGMEMPNAME
C                   EVAL      *INLR = *ON
```

7. **SQL Cursor:**

We have RPG programs that use SQL cursors to sequentially retrieve data. If a program cancels and I call it again, the program resumes processing a cursor where it left off. I have to sign off and back on in order to restart from the top. Why doesn't the program start over from the beginning of the returned data set?

The behavior you're witnessing comes from three contributing factors. First, your program was compiled to close the cursor when the activation group is destroyed. Second, your program is running in the default activation group. Third, you are not checking the open of the cursor to determine whether it succeeds or fails. Fortunately, this is an easy problem to fix.
Let's look at a program like the ones Lynne is talking about.

```
Fqsysprt   o    f  132        printer
F
D zInput          ds                  inz
D  zCustNumber                        6p 0
D  zLastName                        8a
D  zInitials                     3a
D  zBalanceDue                       7p 2
D  zCreditDue                      7p 2

D  Ratio          s              3p 0

 /free
     exec sql
        declare c1 cursor for
              SELECT cusnum, lstnam, init, baldue, cdtdue
                FROM qiws/qcustcdt
               ORDER BY 1;

     exec sql
        open c1;

     dow '1';
        exec sql
           fetch c1 into :zInput;
        if sqlstt >= '02000';
           leave;
        endif;
        eval(h) Ratio = zCreditDue / zBalanceDue * 100;
        except pline;
     enddo;

     *inlr = *on;
 /end-free

Oqsysprt   e           pline       1
```

```
O                    zCustNumber
O                    zLastName       +   1
O                    zInitials       +   1
O                    zCreditDue    j  +   1
O                    zBalanceDue   j  +   1
O                    Ratio        j  +   1
```

Notice the eval within the do-while loop. I've included this line of code in order to make the program cancel.
Here's the result set from running the query. This is the data that the program reads as input.

```
CUSNUM   LSTNAM    INIT    BALDUE     CDTDUE
192837   Lee       F L     489.50        .50
389572   Stevens   K L      58.75       1.50
392859   Vine      S S     439.00        .00
397267   Tyron     W E        .00        .00
475938   Doe       J W     250.00     100.00
583990   Abraham   M T     500.00        .00
593029   Williams  E D      25.00        .00
693829   Thomas    A N        .00        .00
839283   Jones     B D     100.00        .00
846283   Alison    J S      10.00        .00
938472   Henning   G K      37.00        .00
938485   Johnson   J A   3,987.50      33.50
```

The first time I call the program, I get the following output, followed by escape message MCH1211, which tells me that the program attempted to divide by zero.

```
192837 Lee      F L       .50     489.50    0
389572 Stevens  K L      1.50      58.75    3
392859 Vine     S S       .00     439.00    0
```

The second time I call the program, I get the following output before I get another MCH1211.

```
475938 Doe      J W    100.00     250.00   40
583990 Abraham  M T       .00     500.00    0
593029 Williams E D       .00      25.00    0
```

Notice that the second run of the program did not begin with the first record of the result set.
The program was compiled to close the cursor when the activation group is destroyed. The system destroys named activation groups when the last program in the activation group ends. However, the default activation group, which is intended for use only with OPM programs, is destroyed only when the job ends. Therefore, the cursor remained open between invocations.
If I had checked the SQL status variable, SQLSTT, after the open during the first call, I would have found that SQLSTT contained a value of five zeros, meaning that the open succeeded. But SQLSTT would have had a value of 24502 after the open in the second call, meaning that the cursor was already open in the activation group.

So, how do I fix the problem?
The simplest fix is to change the program so that it closes the cursor when the module ends. I do that by adding the following code to the top of the calcs in the RPG program.

```
exec sql
   set option closqlcsr=*endmod;
```

You should also consider running the program in a named activation group. You can easily do this by adding an H spec to the program.

```
 H dftactgrp(*no) actgrp(??????)
```

I'll leave it to you to think about what activation group name you should replace the questions marks with. If an RPG program is a standalone application, you can use *NEW.
Also, I encourage you to check the status of the open and to end the program gracefully if it has a value greater than or equal to 02000.

```
exec sql
   open c1;
if sqlstt >= '02000';
   // do something to handle the failed open
```

8. Sample imbedded SQLRPGLE program

Sample imbedded, or embedded, SQLRPGLE program:

```
     H ActGrp(*CALLER)

     H DftActGrp(*NO)

     D OpenCursor      PR              n

     D FetchCursor     PR              n

     D CloseCursor     PR              n

     D MyLib           s             10a

     D MyFile          s             10a

      /free

       *inlr=*on;

       if not OpenCursor();

         // perform error routine to alert the troops

         // ...
```

```
  Else;
    Dow FetchCursor();
      // putting the fetchcursor on the do loop allows the user of
      // iter, and thus iter will not perform an infinite loop
      // normal processing here...
    EndDo;
    CloseCursor();
  EndIf;
  return;
 /end-free
P OpenCursor      B
D OpenCursor      PI                  like(ReturnVar)
D ReturnVar       s               n
C/EXEC SQL
C+ Set Option
C+     Naming    = *Sys,
C+     Commit    = *None,
C+     UsrPrf    = *User,
C+     DynUsrPrf = *User,
C+     Datfmt    = *iso,
C+     CloSqlCsr = *EndMod
C/END-EXEC
C/EXEC SQL
C+ Declare C1 cursor for
C+  Select System_Table_Schema as library,
C+         System_Table_Name   as file
C+  from qsys2/systables
C/END-EXEC
C/EXEC SQL
C+ Open C1
C/END-EXEC
 /free
  Select;
    When SqlStt='00000';
      return *on;
    Other;
      return *off;
  EndSl;
 /end-free
P OpenCursor      E
 /eject
P FetchCursor     B
D FetchCursor     PI                  like(ReturnVar)
D ReturnVar       s               n
C/EXEC SQL
C+ Fetch C1 into :MyLib, :MyFile
C/END-EXEC
 /free
  Select;
    When sqlstt='00000';
```

```
        // row was received, normal
        ReturnVar=*on;
      When sqlstt='02000';
        // same as %eof, sooner or later this is normal
        ReturnVar=*off;
      Other;
        // alert the troops!
        ReturnVar=*off;
    EndSl;
    return ReturnVar;
   /end-free
  P FetchCursor     E
   /eject
  P CloseCursor     B
  D CloseCursor     PI                  like(ReturnVar)
  D ReturnVar       s               n
  C/EXEC SQL
  C+ Close C1
  C/END-EXEC
   /free
    Select;
      When sqlstt='00000';
        // cursor was closed, normal
        ReturnVar=*on;
      Other;
```

```
        // alert the troops!

        ReturnVar=*off;

     EndSl;

     return ReturnVar;

    /end-free

   P CloseCursor
```

9. **Embedded SQL:**

Integration of sql in traditional sql rpg application.

Static:

- SQL statement defined when program is compiled.
- Files and fields are established before program executes.
- SQL Pre-compilation takes place.

Dynamic:

- SQL statement defined during program execution.
- Files and fields are not established before program executes.
- No SQL Pre-compilation takes place.

Journal

1. **What is the journal?**

   Any changes in PF will be recorded. A journal is an object of type *JRN which detects and records that cause a PF to change.

   The information recorded by the journal is stored in an object *JRNRCV called journal receiver.

2. **What are the various steps creating journal?**
   - The steps to start journaling
     - ✔ Create a journal receiver - CRTJRNRCV
     - ✔ Create a journal - CRTJRN
     - ✔ Start journaling of a PF - STRJRNPF
     - ✔ Backup the PF - SAVOBJ
   - The steps to end journaling
     - ✔ End journaling a PF -ENDJRNPF
     - ✔ Delete a journal -DLTJRN
     - ✔ Delete the last journal receiver -DLTJRNRCV
     - ✔ Save journal receivers -SAVOBJ
   - The commands used for housekeeping purposes

- ✔ Change journal -CHGJRN
- ✔ Save object -SAVOBJ
- ✔ Delete a journal receiver -DLTJRNRCV

❖ The commands used in case of a failure are

- ✔ Display journal entries -DSPJRN
- ✔ Apply journal changes -APYJRNCHG
- ✔ Remove journal changes -RMVJRNCHG
- ✔ Restore a saved object -RSTOBJ

1. **Explain Commitment Control?**

The AS/400 system has an integrated transaction recovery function called commitment control. Commitment control is an extension of the journal function on the system.

The records used during a complex transaction are locked from other user and at the end of the transaction; the program issues the **commit operation,** updating the records.

If the system fails before the commit operation is performed, all database changes are **rolled back** to the previous commit operation and all the affected records are unlocked.

COMMIT-The transactions are updated in the data file. Commit occurs on **COMMIT** command

ROLLBACK- The transactions are NOT updated in the data file. Rollback occurs if there are uncommitted transactions and on **ROLLBACK** command.

2. **Can anybody tell why Journaling is compulsory before Commitment Control?**

Commitment ctrl is use to save /rollback the group of changes and Journaling is use to save the changed records in Journal receiver.

3. **Commitment control Implementation and controlling commitment control from external program.**

1. CRTJRNRCV
2. CRTJRN
3. STRJRN
4. STRCMTCTL (in CL program)
5. Declare file with keyword COMMIT and in program use
Commit and Rollback based on processing logic (i.e. after successful execution use Commit and when invoked exception handling then use rollback).
6. ENDCMTCTL

Data Areas, Queues, Arrays & Structures:

1. What is the data area?

A data area is an object used to store data for access by any job running on the system. It is permanent storage. A data area can be used whenever you need to store information of limited size, independent of the existence of the programs or files.

- Typical uses of data areas are:
    - To provide an area to pass information within job.
    - To provide a field that is easily and frequently changed to control references within a job such as supplying the next check number.
    - To provide a constant field for use in several jobs, such as tax rate
    - To provide limited access to a large process that requires the data area.
    - A data area can be locked to a single user, thus preventing other users from processing at the same time.
- To create a general data area use the command (CRTDTAARA)
- To retrieve values from data area use (RTVDTAARA)
- To change this value, use (CHGDTAARA)
- To display the current value, use (DSPDTAARA)
- To delete a data area use (DLTDTAARA)
- Type of data area created by the system
    - Local data area
    - Group data area
    - Program initialization parameter (PIP) data area

```
                                   SKANDASAMO/RPGILE

                                           DAREA

   *************** Beginning of data ******************************

0001.00 DS              S            10A

0002.00 DG              S            10A   INZ ('I HATE YOU')

0003.00 C     *DTAARA       DEFINE    DATA1         S

0004.00 C     *LOCK         IN        S

0005.00 C                   EVAL      S=G

0006.00 C                   OUT       S

0007.00 C     S             DSPLY

0008.00 C                   SETON                                   LR
```

```
****************** End of data ***********************************
```

OUTPUT

DSPLY I HATE YOU

AUTO NUMBER GENERATION

```
                                                        SKANDASAMS/TEST
                                                              TESTEX19
      *************** Beginning of data *******************************
0001.00 DA           S          4S 0
0002.00 C     *DTAARA     DEFINE    DATA2       A
0003.00 C     *LOCK       IN        A
0004.00 C                 EVAL      A=A+1
0005.00 C                 OUT       A
0006.00 C     A           DSPLY
0007.00 C                 SETON                               LR
      ****************** End of data ***********************************
```

OUTPUT

DSPLY 5

DSPLY 6

2. **Define LDA, GDA, and PIP?**
   - ❖ LOCAL DATA AREA (LDA)
     - ✔ A local data area is created for each job in the system automatically, when you submit a job.
     - ✔ Only one LDA can be created by submitting a job.
     - ✔ The system create a local data area, which is initially filled with blanks, with a length of 1024 and type ***CHAR.**
       - ✔ When you submit a job using SBMJOB command, the value of the submitting job's local data area is copied into the submitted job's local data area.
     - ✔ You can refer to your job's local data area by specifying *LDA for the DTAARA keyword on the CHGDTAARA, RTVDTAARA, and

DSPDTAARA commands or *LDA for the sub string built-in function (%SST)

- ✔ The following is true of a local data area:
  - ➢ The local data area cannot be referred to from any other job.
  - ➢ You cannot create, delete or allocate a local data area.
  - ➢ We can to change the contents of LDA by the by using CHGDTAARA command.
  - ➢ No library is associated with the local data area.
- ✔ ACCESSING LDA:
  - ➢ CHGVAR VAR (%SST (*LDA 3 5)) VALUE(123)

    OR
  - ➢ CHGDTAARA DTAARA (*LDA (3 5)) VALUE(123)
  - ➢ CHGVAR VAR (&ROLNO) VALUE (%SST (*LDA   3 5))

    OR
  - ➢ RTVDTAARA DTAARA (*LDA (3   5)) RTNVAR (&ROLNO)

❖ GROUP DATA AREA (GDA)

- ✔ The system creates a group data area when an interactive job becomes a group job.
- ✔ Only one group data area can exist for a group.
- ✔ The group data area is deleted when the last job in the group is ended, or when the job is no longer part of the group job.
- ✔ A group data area, which is initially filled with blanks, has a length of 512 and type *CHAR.
- ✔ The following is true for a group data area
  - ➢ You cannot use the group data area as a substitute for a character variable on the sub string built-in function.
  - ➢ A group data area cannot be referred by jobs outside the group.
  - ➢ You cannot create, delete, or allocate a group data area
  - ➢ No library is associated with a group data area.
- ✔ Example

CHGDTAARA DTAARA (*GDA) VALUE (‘DECEMBER 1996’)

RTVDTAARA DTAARA (*GDA) RTNVAR (&GRPARA)

❖ PROGRAM INITIALIZATION PARAMETER **(PIP**) DATA AREA

- ✔ A PIP data area is created for each pre-started job when the job is started.
- ✔ The object sub-type of the PDA is different than a regular data area.
- ✔ The PIP can only be referred to by the special value name *PDA.

- ✔ The size of the PDA is 2000 bytes but the number of parameter contained in it is not restricted.

2. What is the data queue?

- ✔ It is a temporary storage. We can able to store and retrieve the data, but once data is retrieved the data is lost.
- ✔ First create the data queue by using CRTDTAARA command
- ✔ Sending a message to a data queue (QSNDDTAQ, QRCVDTAQ, And CLRDTAQ)

Data query is nothing but a queue in which are program can send a data and other program or the same program can receive the program. QSNDDTAQ is stored in QSYS.

2. Explain QSNDDTAQ and QRCVDTAQ?

- ❖ QSNDDTAQ
  - ✔ By using this command sent data same / another program.
  - ✔ QSNDDTAQ PARM (QUEUE NAME LIB &LEN &DAT)
- ❖ QRCVDTAQ
  - ✔ By using this command receive data same /another program
  - ✔ QRCVDTAQ PARM (QUEUE NAME LIB &LEN &DAT &WAIT)

2. What are the mandatory parameters for declaring a Data queue?

- ✔ QUEUE NAME
- ✔ LIB NAME
- ✔ LENGTH
- ✔ DATA
- ✔ WAIT

2. What is the command to create menu?

CRTMNU – Create Menu

3. What is the difference between CALL and Transfer Control (TFRCTL)?

CALL	TFRCTL
1.Call will transfer the control according with the CALL STACK	Transfer Control (TFRCTL) will remove the CALL STACK and transfer the control to the calling program.
2. The CALL is used to different types of programs. Ex: RPGILE/400, CL/400, C/400, COBOL/400.	TFRCTL is only used in CL programs.

4. **Explain Multi Dimensional Array?**
   - The multi dimension data structure array will be implemented in occur class.
   - The similar elements of same data type and same attributes size repeating many times this time using Occur opcode.
   - Can only be used with a multiple occurrence data structure, allow you to specify which occurrence of data structure is used for subsequent operation within the programs.

2. **Define data structure and types of data structure?**

The different types of fields and sub fields are stored in a single area. This area in storage is called data structure. Data structure means program allows you to define an area in storage and the layout of the fields, called sub fields, with the area. This area in storage is called a data structure.

Data structure can be used for

- ✔ Group non-contiguous data into contiguous internal storage locations
- ✔ Define the same internal area multiple times using different data formats.
- ✔ Operate on a field and change its contents
- ✔ Divide a field into sub fields without using the MOVE or MOVEL
- ✔ Define a data structure and its sub fields in the same way a record is defined.
- ✔ Define multiple occurrences of a set of data

There are four different types of data structure commonly used.

- ❖ General data structure
- ❖ Data area data structure
- ❖ File information data structure
- ❖ Program status data structure

Data structure can be specified in D spec

Type IPDS

Data structure name

I –Globally initialized data structure

S--Program status data structure

U--Data area data structure

Blanks—**G**eneral (or) **F**ile status data structure

- ❖ **Data area data structure**

  A data area data structure, identified by a **U** in position 18 of the data structure statement, indicates to the RPGLE program that if should read in and lock the data area of the same name at program at program initialization and should write out and unlock the same data area at end of the program.

The data area and data area structure must have the same name unless you rename the data within the program by using the *NAMVAR DEFINE statement.

❖ **File information data structure**

A file information data structure provides you with status information on file exception /error occurs. This data structure name must be unique for each file. It consists of pre defined sub fields that provide information on the file exception/error that occurred.

❖ **Program status data structure**

This data structure is identified by as S in position 18 of the data structure statement, provides program exception/error information to the program. The *ROUTINE, *STATUS, *PARM keywords mostly preferred to determine the PS DS.

Example

```
        SKANDASAMO/DATASTR
                                                    DUMP
      *************** Beginning of data *************************
0001.00 HDEBUG (*YES)
0002.00 DPSSR           SDS
0004.00 DSTATUS            *status
0005.00 DROUTINE           *routine
0005.01 DPARMS             *parms
0005.02 DRES            S            2S 0
0007.00 C               Z-ADD   1          NUM1            2 0
0007.01 C               Z-ADD   0          NUM2            2 0
0010.00 C               EVAL    RES=NUM1/NUM2
0011.00 C    'NOTCOM'     DSPLY
0012.00 C    A            TAG
0013.00 C    'COMING'     DSPLY
0014.00 C               SETON                              LR
0015.00 C    *PSSR        BEGSR
```

```
0016.00 C     STATUS        DSPLY
0017.00 C     ROUTINE        DSPLY
0018.00 C     PARMS         DSPLY
0019.00 C                   DUMP
0020.00 C                   GOTO      A
0021.00 C                   ENDSR
        ***************** End of data *****************************
```

Output

```
DSPLY    102
 DSPLY *DETC
 DSPLY    0
 DSPLY  COMING
```

2. **How do I declare an array with a dynamic number of elements?**

In RPG IV, the new (V3 R7) ALLOC, REALLOC and DEALLOC operation codes can be used to allocate memory. This means that at run time, you can go out to the system and ask it to assign storage to the program that was not allocated to the program when it was evoked.

These operation codes can be used to allocate memory up to 16MB. The allocation can be assigned to a pointer variable. In RPG IV, pointers have the data-type of asterisk (*). All that is needed is to allocate memory to a pointer that is used with the BASED keyword of the desired dynamic array. The example that follows illustrates this technique:

```
.....DName+++++++++++EUDS.......Length+TDc.Functions+++++++++++++
+++++
     D DynoArr         S              7P 0 Dim(10000) based( pDynoArr)
     D nSize           S             10i 0

.....CSRn01.............OpCode(ex)Extended-factor2++++++++++++++++++
     C                   Eval      nsize = %size(DinoArr) * 64
.....CSRn01Factor1+++++++OpCode(ex)Factor2+++++++Result+++++++
+Len++DcHiLoEq
      C                  Alloc     nSize         pDynoArr
     **  We now have enough storage allocated for 64 elements.
```

```
C                  Eval      nsize = %size(DinoArr) * 70
C                  ReAlloc   nSize         pDynoArr
** We have changed the allocation to enough storage for 70 elements
C* ... code to use the array goes here...
C                  Dealloc(N)              pDynoArr
** We have just returned the storage to the system.
```

To increase or decrease the number of elements in the dynamic array, use the REALLOC operation code. Simply change the number of bytes that need to be allocated, and call REALLOC with the new size in Factor 2 and the original pointer variable in the Result field. REALLOC allocates new storage of the size specified, and copies the original data to that new location. Then it frees ("deallocates") the original storage.

IMPORTANT: Always remember to DEALLOC anything you ALLOC. That is always free up memory that you have allocated otherwise memory leaks will be created.

If you are not on V3 R7, you can still use dynamic memory by calling one of the system APIs or linking into the QC2LE binding directory and calling the C runtime MALLOC and DEALLOC functions.

3. Data structure array basics

It's all pretty straightforward. If *MyDS* is a data structure array of 10 elements, then *MyDS(5)* will access the fifth element of that data structure array. But how do you access the subfields? That requires a bit more discussion.
When you define a data structure array, you not only use the **Dim** keyword, but you must also specify the **qualified** keyword. The **Qualified** keyword was introduced in V5R1, and it allows you to specify the name of a data structure subfield qualified by the name of the data structure. For example, if data structure *MyDS* is defined with the **qualified** keyword and it contains a subfield named *Subfield1*, then you would use the following notation to access the subfield:
MyDS.Subfield1
If data structure *MyDS* is also defined with the **Dim** keyword, then something like the following notation is used to access subfields in a specific element:
MyDS(5).Subfield1
In this example, you would be accessing the subfield named *Subfield1* in the fifth element in the data structure array *MyDS*.

4. **Clear up the confusion over multiple-occurrence data structures**
   over the past few months I have detected a hint of confusion about

> multiple-occurrence data structures with regard to data structure arrays. While the two have some dissimilarities, data structure arrays are essentially and technically the same as multiple-occurrence data structures.

More precisely, multiple-occurrence data structures are, *and have always been*, data structure arrays; they were just implemented poorly and given a goofy IBM name. Of course, this was the result of the six-character limit on field names and on the *Result Field* column. Under the covers, data structure arrays and multiple-occurrence data structures are implemented almost identically. You see, a data structure is a high-level language (HLL) construct; there is no such thing as a data structure down at the machine interface (MI) level. That means *all* data structures must be implemented in a simulated fashion, regardless if they are simple data structures, multiple-occurrence data structures or data structure arrays. Interestingly, the MI language *does* contain arrays -- and fairly sophisticated ones at that. So, once a data structure is defined at the MI level, making it into an array is fairly straightforward.

Thankfully, we (i.e., the RPG language) have evolved to where we can actually call data structure arrays "arrays" instead of "multiple-occurrences." Unfortunately, most, if not all, of the documentation has not. I have yet to find one definitive statement that explains, for example, that multiple-occurrence data structures are often interchangeable with data structure arrays, such as when passed as parameters.

Arrays (this includes multiple-occurrence data structures because remember, they are arrays) are stored in one chunk of contiguous storage. There are no extra spaces in between the individual elements of the array unless you specify alignment; that is, the compiler will not add any extra space that you don't ask for. Therefore, in storage, a multiple-occurrence data structure looks exactly like a data structure array.

Another good example of the documentation being out of date is the *SQL Reference*. In the "ILE RPG," it discusses using multiple-occurrence data structures for multiple-record fetches. However, it makes no mention of using data structure arrays as an alternative, never mind that using data structure arrays is actually a *better* alternative.

Always use data structure arrays

if you are in charge of setting programming practices and standards in your shop, you should insist that data structure arrays are used instead of multiple-occurrence data structures whenever possible. They are much simpler to code, and they are much less prone to bugs.

"CLEAR" advantage over resetting work fields

Use the "CLEAR" operation code to clear the fields in a single-occurrence data structure or multiple-occurrence data structures.
Pass larger amounts of data between programs
when using separate programs to retrieve and display data, use arrays or multiple-occurrence data structures to pass back larger amounts of data strings.

5. Data area, Data Queue and Message Queue:

Data area size is defined while creation and if the data of the same size is put on the data area the previous data would get overwritten. For example we have a data area of size 512. First time you have moved the value of size 512. If you try to move some other data of the same size (512) the second time, then the first data would be lost since it gets overwritten by the latest one. But data will be present in the data area even after several retrievals.

On the other hand, we define the size of the data the data queue can hold at the time of creation. The data queue will grow as and when new data comes in. The number of data in the data queue is not fixed while creation and the number of data it can hold depend upon the system space. It is just like the normal queue, it grows when new data comes in. We just set the size of one data and not the size of the data queue.
For example you have created a data queue of size 100. If you move data for the first time it will be stored in the first position. Then if you move the second data it will be stored in the second position and so on. Depending upon the method (FIFO, LIFO or keyed) specified at the time of creation, for retrieving data the data will be fetched.
After the data is retrieved once (via receive data queue command) it will be lost. It will be present in the data till the first retrieval.
In the message queue the message will be present even after several retrievals.

6. Group Jobs and Group data area:

Group data area is created by the system when an interactive job becomes a group job.
Group job is nothing but another job with the same job name but with a different job number. You will have 2 sessions in the same session itself.
Example:

Assume that you have a session with the Job: QPADEV0008 and the Number: 946578 (Do a Shift+F3 to get this information after logging in). After creating multiple jobs the job name will be the same but we will have different job numbers for each session.
Try to display the contents of *LDA and *GDA.
DSPDTAARA *LDA
Will display the contents of local data area
But DSPDTAARA *GDA will give an error message (DTAARA(*GDA) not valid because job not group job) since this job is not an interactive job.
Change the interactive job to group job (TSTGRPJOB) like this.
CHGGRPA GRPJOB(TSTGRPJOB) MSGQ(*NONE)
Now try DSPDTAARA *GDA. It will work, because the interactive job is now changed to a group job.
Now we have one group job TSTGRPJOB. We are creating another group job like this to explore the group job concept.
Give the command TFRGRPJOB GRPJOB(*SELECT) and press Enter. Then Press F6 to Start a new group job.
Give the following details and Press Enter.

```
Group job  . . . . . . . . . . .   GRPTEST2
Initial group program  . . . . .  QCMD
```

Note: Instead of QCMD you can give a program name as well. After the program ends the job will end and control will come back to the previous job from where the TFRGRPJOB is issued.
If you Signoff from any group job session then all the jobs will be ended since signing off from that job closes the *GDA which internally ends all the jobs present in that group.
Note:
If you do it with Shift + Esc + 1 we can have 2 sessions only with the same job name.
Shift + Esc + 1 will take you to sign on screen. You can login to go to the second session (meaning second job with the same job name and a new job number), you will be in the same session even now. If you sign off from that session you will go the first session (place where you gave Shift + Esc + 1) and not to the Sign on screen – But it is not a group job, we are just accessing 2 sessions from the same session itself and there is no provision of switching between the sessions. After completing the work in the second session you have to signoff to come to the first session. It is just an option to have one more session inside the same session rather than opening a new session. That's all.

But with group job concept we can have up to 32 sessions in the same session. With transfer group job command we can go to different sessions.

7. **Data Structure Array and Example:**

```
D Ds_Array       DS                Dim(25) Qualified
D  Ds_Fld1                   5A
D  Ds_Fld2                   5S 0
C                Eval     Ds_Array(1).Ds_Fld1 = 'TEST'
C                Eval     Ds_Array(1).Ds_Fld2 = 12345
C                Eval     *Inlr = *On
```

8. **Difference between Data area and data queue:**

Data area: Data will not be lost after it is accessed.
Data queue: Data will be lost after QRCVDTAQ is executed, meaning data will be lost from the data queue after it is received.

9. **Difference between data-structure array and multi occurrence data structure**

Data-structure array: Subfield names can contain more than six characters.
MODS: Subfield names cannot contain more than six characters.

10. **RPG data structure arrays improvement over multiple-occurrence data structures**

In case you haven't yet heard the news, the ILE RPG compiler now supports -- as of V5R2 data structure arrays. That is, you can now specify the Dim keyword on the definition of a data structure to dimension that data structure. Just as you can dimension a stand-alone field to create an array of that type of field, you can now dimension a data structure to create an array of that kind of data structure.

11. **Compile time array, pre run time array run time array**

Compile time array is the array which is declared with all the actual values in the program using CTDATA

Pre run time array: It is loaded from a column of physical file

Run time array: values are loaded in the array during execution time

You define array on "D" Specification you can load data into an array at compile, with value entered at end of the program is called = Compile time array.

You define array on "D" Specification you can load data into a array at compile, with value obtained from a File is called = Pre-Runtime Array

Arrays can be loaded with values during the program execution this is called = Runtime Array

Small correction about pre run time array:

The file used as input must be a sequential program described file. DURING INITIALIZATION, BUT BEFORE ANY INPUT, CALCULATION, OR OUTPUT OPERATIONS ARE PROCESSED the array is loaded with initial values from the file. By modifying this file, you can alter the array’s initial values on the next call to the program, WITHOUT RECOMPILING the program.

12.RNF7701 data structure not allowed

I'm using a data structure for me READ/WRITE/READP operations but I keep getting RNF7701 - Data structure not allowed. Here's the code:

```
FWMERLGP   if a e           k disk

d Q            e DS                 EXTNAME(WMERLGP:WMERLG:*INPUT)
d                                   QUALIFIED

d next_eid        pr            11s 0
d $err            s              n

 /free
```

```
        setgt *hival wmerlgp;
        // read into a ds so as not to disturb the
        // global file fields
        readp wmerlg Q;
    // RNF7701 here!
```

The answer is that the compiler wants you to use LIKEREC instead of EXTNAME.
Try

```
     d Q              Ds                  LikeRec(WMERLG : *Input )
```

DEBUG

1. How to Debug a Batch ILE RPG?

1. Compile your ILE source using DBGVIEW (*SOURCE) OPTION (*SRCSTMT:*NODEBUGIO)

2. Hold the job queue where the program will be submitted

3. Submit the job

4. Hold the submitted job

5. Release the jobq, and if required change the jobq.

6. Start servicing the submitted job using STRSRVJOB JOB (number/user/name) (Start Service job), where "number", "user' and "name" are attributes retrieved from the submitted job.

7. Start the debugger using STRDBG PGM (library/program) UPDPROD (*YES) (Start Debug)

8. You are now in the display module source. Don't try to debug or add breakpoints. It's too early. The job must be active before. Leave this screen using F12.

9. Release the held job.

10. When the job is activated, the start service job window comes. Use

F10 to enter debug command.

11. Use DSPMODSRC (Display module source) command to go back to the source and add breakpoint wherever you want, using F6

12. Use F12 to resume the job, and F12 once again to go back to the start service job window.

13. Use Enter to launch the job.

14. The process breaks at the first breakpoint installed and gives you the control.

15. Debug....my friend

16. When the job is finished, you receive a 'Job being serviced ended' message.

17. Use ENDDBG command to end debug.

18. Use ENDSRVJOB to end servicing

2. Debug value of pointer?

When debugging ILE RPG code, to see what a pointer is pointing to, you can use :x or :c when using the EVAL debugger command on the pointer itself. This means you can debug a pointer's value without having an RPG BASED variable.

This shows the first 10 bytes that "ptr" is pointing to, as character data: ===> EVAL ptr:c 10

This shows the first 10 bytes that "ptr" is pointing to, in hex: ===> EVAL ptr:x 10

This is useful when the pointer is pointing to data that is not readable as character data—

3. How do I debug ILE programs? STRISDB doesn't work!

Be sure you compile your program with an option that allows the debugger to "see" the source: DBGVIEW(*LIST) will require access to the source at run-time, while DBGVIEW(*LIST) inserts a copy of the compile listing right into the object code, so no access to the source is needed at run-time.

After that, the basic sequence of events is to STRDBG, set the breakpoints, hit F3 back to the command line and call your program.

Detailed steps:

1. STRDBG - you may need to specify UPDPROD(*YES), but please be very careful debugging in a production environment!
2. Set breakpoints. Position the cursor on a line where you want a breakpoint. Press F6. The debugger will stop before this line is executed. Set as many as you need.
3. If you are debugging a service program, press F14 to see a screen where you can add the service program (*SRVPGM) to the list of modules under debug control. Use option 5 on the module line to enter the source and set breakpoints. Press F14 to get back to the original source.
4. Exit the STRDBG session and get back to the command line.
5. Call your program the way you normally would; from a menu or command line. You don't necessarily need to call the program under debug; if it's the third one in a series, you can call the first one and let the programs progress as normal.

There are a few differences between STRISDB and STRDBG: STRISDB WATCH has no equivalent. WATCH under STRDBG will break when a watched variable changes.

You don't display variables with the "D" command; it's EVAL under STRDBG. To see a variable in hex, do an eval varname:x. You don't set a value with "c", you use EVAL; eval varname = 'OPEN'. Look at the help for commands on the command line for a quick reference of the available commands. Check the language reference manual for details of how to use the debugger with your target language.

1. How can I debug an ILE program in batch?

Brad Stone has a FAQ entry that addresses this:
http://bvstools.com/faq.html#RPG

Using the green screen debugger:

1. Submit your program to batch. The job MUST be held. You can either hold the job queue (HLDJOBQ) or hold the individual job (HLDJOB) or specify HOLD(*YES) on the SBMJOB command.
2. WRKSBMJOB/WRKUSRJOB/WRKACTJOB and find your submitted job. Note that the SBMJOB command gives you an informational message

with the job name/number. What you need is the job name, user ID and job number - the fully qualified job name. Example: 123456/BUCK/MONTHEND

3. STRSRVJOB on the held batch job.
4. STRDBG on your program. Specify UPDPROD(*YES) if needed. You'll see the source listing if you compiled with DBGVIEW(*LIST) or *SOURCE.
5. Press F12 to exit - you cannot set a breakpoint yet.
6. Release the job so that it becomes STATUS(*ACTIVE).
7. You'll see a display asking if you want to debug or continue. Press F10 to debug.
8. DSPMODSRC to see the source listing again. Alternately, press F10 to step into the first instruction.
9. Now you can add your breakpoints.
10. Press F3 until you're back to the "debug or continue" display. Press Enter to run the program with your breakpoints set.
11. When you're done, do an ENDDBG and ENDSRVJOB.

Using the IBM Distributed Debugger:

1. SBMJOB CMD(CALL PGM(yourlib/yourpgm)) JOBQ(yourlib/yourjobq) HOLD(*YES)
2. Start your Code debugger from Start->Programs->WebSphere Development... ->IBM Distributed Debugger->IBM Distributed Debugger
3. Select the debugger Start up window and key into the job name entry field */##########/* where ########## is your user id.
4. You may have to log in and specify the AS/400 system name.
5. Select the job that is being held in yourjobq.
6. Click the ok push button.
7. Enter the library and program name into the Program entry field
8. Click the Load push button on the debugger Startup information window. A debugger message will appear telling you to start the program.
9. Click Ok on the message push button, even though it tells you to start your program first.

10. Switch to a 5250 emulation window.
11. WRKJOBQ JOBQ(yourlib/yourjobq)

Release your job.

1. How can I debug an OPM program in batch?

You can follow the steps in How can I debug an ILE program in batch? And use the old (no source) debugger STRDBG.
Mike Barton suggests compiling the program with OPTION (*SRCDEBUG) and then using STRDBG OPMSRC(*YES), which should work with the steps given above.
STRISDB won't work unless the job is running, so you can't put it on hold and enter your break points.
Martin Rowe contributed the following idea: Insert a simple CL program into your RPG that waits for you to answer a message. This way, the job is running, but not processing yet. (RPG400-L 24 May 2001)
Here is an adaptation of his idea:
Here's the RPG program you're trying to debug:

```
     H       1
     C                  CALL 'DBGWAIT'
     C                  Z-ADD1        X       50
     C                  SETON                   LR
```

I've inserted "CALL 'DBGWAIT'" and re-compiled.
Here's the source for DBGWAIT:

```
pgm

dcl &reply *char 1

sndusrmsg  msgid(CPF9898) +
           msgf(QCPFMSG)  +
           msgdta('Paused for debug') +
           msgrpy(&reply)

endpgm
```

STRISDB PGM (BATCHOPM) UPDPROD (*NO) INVPGM (*NO) SRVJOB (*SELECT) you'll see a list of all active jobs on your system. Select the one you're trying to debug. You'll get a message saying that the program is in debug mode. Answer the "paused for debug" message, and the source will pop up after the call to DBGWAIT.

2. How can I tell if my program is running in batch or interactive?

Here's a program that uses the QUSRJOBI API. After calling the API, the Job Type field (QUSJT04) will contain a B for batch or I for interactive. If you don't pass the internal job name (don't pass anything) it will retrieve information about the current job.

```
DQUSI020000       DS
D*                                 Qwc JOBI0200
D QUSBR01                1      4B 0
D*                                 Bytes Return
D QUSBA01                5      8B 0
D*                                 Bytes Avail
D QUSJN03                9     18
D*                                 Job Name
D QUSUN03               19     28
D*                                 User Name
D QUSJNBR03             29     34
D*                                 Job Number
D QUSIJID01             35     50
D*                                 Int Job ID
D QUSJS05               51     60
D*                                 Job Status
D QUSJT04               61     61
D*                                 Job Type
D QUSJS06               62     62
D*                                 Job Subtype
D QUSSN                 63     72
D*                                 Subsys Name
D QUSRP01               73     76B 0
D*                                 Run Priority
D QUSSPID00             77     80B 0
D*                                 System Pool ID
D QUSCPUU00             81     84B 0
D*                                 CPU Used
D QUSAIOR               85     88B 0
D*                                 Aux IO Request
D QUSIT                 89     92B 0
D*                                 Interact Trans
D QUSRT                 93     96B 0
```

```
D*                                          Response Time
D QUSFT                  97     97
D*                                          Function Type
D QUSFN15                98    107
D*                                          Function Name
D QUSAJS                108    111
D*                                          Active Job Stat
D QUSNDBLW              112    115B 0
D*                                          Num DBase Lock Wts
D QUSNIMLW              116    119B 0
D*                                          Num Internal Mch Lck Wts
D QUSNDBLW00            120    123B 0
D*                                          Num Non DBase Lock Wts
D QUSTDBLW              124    127B 0
D*                                          Wait Time DBase Lock Wts
D QUSTIMLW              128    131B 0
D*                                          Wait Time Internal Mch L
D QUSNDBLW01            132    135B 0
D*                                          Wait Time Non DBase Lock
D QUSERVED45            136    136
D*                                          Reserved
D QUSCSPID              137    140B 0
D*                                          Current System Pool ID
D QUSTC01               141    144B 0
D*                                          Thread Count
DQUSEC            DS           116    inz
D QUSBPRV                 1      4B 0 inz(116)
D QUSBAVL                 5      8B 0 inz(0)
D QUSEI                   9     15
D QUSERVED               16     16
D QUSED01                17    116

D FormatName      S              8    Inz('JOBI0200')
D InJobName       S             26
D IntJobName      S             16
D JobName         S             26    Inz('*')
D Outcount        S              5  0
D ReceiveLen      S             10i 0 Inz(187)

c     *entry        Plist
c                   Parm                    InJobName
```

```
c                    If       %parms > 0
c                    Eval      JobName = InJobName
c                    Endif

 * Call the api to get the information you want
C                    Call      'QUSRJOBI'
C                    Parm                     QusI020000
C                    Parm                     ReceiveLen
C                    Parm                     FormatName
C                    Parm                     JobName
C                    Parm                     IntJobName
C                    Parm                     QusEc
c                    Eval      *inlr = *o
```

3. How to debug jobs in MSGW without ending it?

You have to process in three steps:
1) Get the Name/User/Number of the job in Message Waiting status.
   You can find this by the WRKACTJOB command, and then press 5 in front of the job. On the top of the screen, you get those information's.
2) Service the job with the command: STRSRVJOB JOB(JOBNAME/USER/NUMBER)
3) Use the usual command STRDBG.
Important note: Do not forget to put the option DBGVIEW(*ALL) when creating the program and/or the module, if you want to be able to see the error inside the source code while debugging.

4. How do you do debugs for ILE programs and Handle Exceptions?

Create the program EMPRPT so that you can debug it using the source debugger. The DBGVIEW parameter on either CRTBNDRPG or CRTRPGMOD determines what type of debug data is created during compilation.

The parameter provides six options which allow you to select which view(s) you want:

* *STMT — allows you to display variables and set breakpoints at statement locations using a compiler listing. No source is displayed with this view.

* *SOURCE — creates a view identical to your input source.

* *COPY — creates a source view and a view containing the source of any /COPY members.

* *LIST — creates a view similar to the compiler listing.

* *ALL — creates all of the above views.

* *NONE — no debug data is created.

The source for EMPRPT is shown in Figure 28 on page 54.

1. To create the object type:

```
CRTBNDRPG PGM (MYLIB/EMPRPT) DBGVIEW (*SOURCE) DFTACTGRP (*NO)
```

The program will be created in the library MYLIB with the same name as the source member on which it is based, namely, EMPRPT. Note that by default, it will run in the default named activation group, QILE. This program object can be debugged using a source view.

2. To debug the program type:

```
STRDBG EMPRPT
```

Programming Concepts

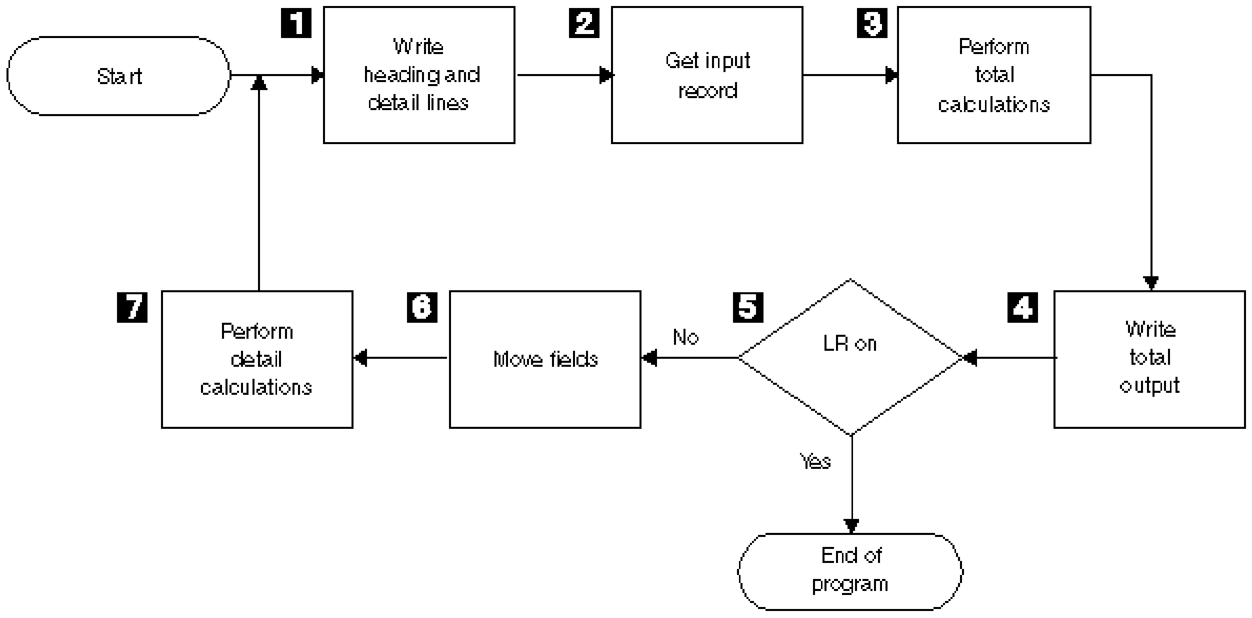
1. General RPG IV Program Cycle

Figure shows the specific steps in the general flow of the RPG IV program cycle. A program cycle begins with step 1 and continues through step 7, then begins again with step 1.

The first and last time a program goes through the RPG IV cycle differ somewhat from the normal cycle. Before the first record is read the first time through the cycle, the program resolves any parameters passed to it, writes the records conditioned by the 1P (first page) indicator, does file and data initialization, and processes any heading or detail output operations having no conditioning indicators or all negative conditioning indicators. For example, heading lines printed before the first record is read might consist of constant or page heading information or fields for reserved words, such as PAGE and *DATE. In addition, the program bypasses total calculations and total output steps on the first cycle.

During the last time a program goes through the cycle, when no more records are available, the LR (last record) indicator and L1 through L9 (control level) indicators are set on, and file and data area cleanup is done.

Figure 5. RPG IV Program Logic Cycle


Start
1
Write heading and detail lines
2
Get input record
3
Perform total calculations
4
Write total output
5
LR on
No
Yes
End of program
6
Move fields
7
Perform detail calculations

1 All heading and detail lines (H or D in position 17 of the output specifications) are processed.

2 The next input record is read and the record identifying and control level indicators are set on.

3 Total calculations are processed. They are conditioned by an L1 through L9 or LR indicator, or an L0 entry.

4 All total output lines are processed (Identified by T at position 17 of output specifications).

5 It is determined if the LR indicator is on. If it is on, the program is ended.

6 The fields of the selected input records are moved from the record to a processing area. Field indicators are set on.

7 All detail calculations are processed (those not conditioned by control level indicators in positions 7 and 8 of the calculation specifications) on the data from the record read at the beginning of the cycle.

2. What are Static bind and Dynamic binds?

Dynamic Bind:

Suppose pgm A is calling pgm B. In current scenario, pgm A will get compiled irrespective of pgm B object (*pgm), is there or not. It will give an error at runtime, but not at compile time since this is a dynamic bind.

Static bind:

Suppose pgm A is calling pgm B. In current scenario, pgm A will not get compiled if pgm B object (*pgm) not there. It will give an error at compile time since this is a static bind.

3. CRTBNDRPG & CRTRPGPGM

CRTBNDRPG: Here the module is been created internally (a temporary one) and the *PGM is been created.

CRTRPGPGM: Here the module a permanent one is created and the *PGM is been created.

4. Hidden Fields:

In the hidden fields, the fields that are been hidden have the previously entered values below and on F5 the values previously entered will be available on the field.

5. Last statement of any RPG pgm is LR?

Not necessary. We can write the codes after SETON LR also and it will execute that too. The main difference is that the program will end after the last line of coding you did and will clear all the access path and variables to blanks or zeros.
But when you give RETURN then the program will not execute the below coding part and will end as soon as the RETURN is encountered, but here it will not clear the access path and the variables. If you run the program again you will find the last value in the variable.

While writing a program Last statement of RPG program need not be LR. It is the last executable statement. I.e. after all processing is done LR (Last Record indicator) will be set on.

NOTE: In a program that does not contain a primary file; you can set the LR indicator on as one method to end the program.

Last statement is RETURN. Means Exit from program LR is second last statement. Means move pointer to end of file.

```
Ex:     CLOS     BEGSR
                 MOVE      '1'         *INLR
                 RETURN
                 ENDSR
```

LR is generally set on after all processing is done i.e. at the end of the program but there is no hard and fast rule for that.

6. **Is Constant can be define as a key field?**

No, But you can give a constant Value in physical file field and you can specify that field as a key field.
In PF

```
A        XXXXX    20    Value ("MYNAME")
         yyyyy     6 0
        Kxxxxx
```

7. **Which keyword is used both in subfile and subfile control record format of a DSPF?**

```
SETOFF      03      'Exit'
 SETOFF      12      'Cancel'
```

8. **Define interactive jobs and batch jobs?**

In Interactive jobs user interaction to called job. But in batch jobs user interaction not required (actions taken by System).
Interactive job using "CALL" command (CALL PROGRAMNAME) to run program and you can't use your pc until program ends. Batch job using "SBTJOB" command (SBMJOB"CALL/PROGRAMNAME") to run program and you can use your pc and the program will run in the batch (QBATCH) you can see this use WRKACTJOB command.

9. **WHAT IS THE DIFFERENCE BETWEEN 'COLHDG' AND 'ALIAS'?**

Both (COLHDG & ALIAS) are used to identify fields. COLHDG & ALIAS is the Description of fields. The difference is that in ALIAS we can access data based on that ALIAS name, while COLHDG is not allowed. Suppose in PF field name as DES78, give ALIAS as Description78, then user can access data from using Description78.

10.What's the difference between CONST and VALUE?

If you pass a parameter by value, you are free to change the variable defined by the procedure interface (you might want to change blanks to zeroes, double quotes, strip leading zeroes). For a CONST parameter you would have to copy to a local variable first.

The point is that in both cases the caller is assured that his variable won't be changed by the called procedure, but the called procedure can use VALUE parameters as work variables, whereas CONST parameters are not allowed to be used in any way except "read-only."

A technical difference is that a CONST parameter is passed by reference, while a VALUE parameter is passed by value.

When a parameter is passed by reference, the called procedure receives a pointer to the parameter. When a parameter is passed by value, the called procedure receives the actual value. Since a call to a program always passes parameters by reference (on the Iseries at least), VALUE is not allowed on program prototypes (EXTPGM keyword), but CONST is allowed.

11.CL – EOF:

When the end of file is reached, message CPF0864 is sent to the procedure or OPM program. The CL variables declared for the record format are not changed by the processing of the RCVF command when this message is sent. You should monitor for this message and perform the appropriate action for end of file. If you attempt to run additional RCVF commands after end of file has been reached, message CPF0864 is sent again.

12.Level Check Error:

All the files will have a value for Level Check after compilation if the parameter Level check (*yes) is given. If the file is modified and compiled again, it will get a new value for level check. If the parameter Level Check(*yes) is given for the file and the program using that file is not compiled after the file is modified and compiled, then the program will encounter level check error at the time of execution.

13.Significance of Return and *INLR = *ON.

If *INLR = *ON is specified in a program, it instructs the compiler that all the operations should be terminated (i.e., if file is opened implicitly it should be closed, data movement to/from data area should happen etc) before exiting the program. Even if there are any executable statements following that they will be executed once.

Now, *INLR = *ON, followed by RETURN will terminate the operations and return to the caller. Any executable statements that follow these 2 statements will not be executed.

If RETURN is given separately, the program will get returned, but the operations will not be terminated (i.e., if file is opened implicitly it will not be closed, data movement to/from data area will not happen etc). Any statements that follow RETURN will never get executed.

14. ***Entry – significance of factor 1, factor 2 and result fields.**

http://publib.boulder.ibm.com/infocenter/iadthelp/v7r0/index.jsp?topic=/com.ibm.etools.iseries.langref.doc/evferlsh320.htm

15. EDTCDE & EDTWRD, OVERLAY, RSTDSP, Command Attention key and Command Function Key and Validity check:

- EDTCDE & EDTWRD

EDTCDE & EDTWRD are key words used for formatting purpose. EDTCDE cannot be applied to Character filed. And EDTCDE has some Codes pre-defined for example, EDTCDE(**Z**) – for suppressing the leading zero Y – for date field.

EDTWRD can be used to define user defined formatting for fields.

- OVERLAY

It allows a record format to be displayed on screen retaining the previous displayed record formats.

- What key word is used when screen is re-displayed?

RSTDSP is a parameter to be specified at compile time for display file.

- Command Attention key and Command Function Key?

With the help of Command attention key we can pass only the indicator status to program not the data from screen. While command function key passes indicator status as well as a data from screen to program.

- How to validate input values in Display file?

With the help of Validity check key words VALUE, RANGE, COMP

1. **What is the difference between CA and CF keys?**
    - CF transfers the changed data and CA transfers only the indicator value.

1. **What is PSDS?**

The Program Status Data Structure is the way the RPG compiler informs the program what is happening at runtime. This is explained in the RPG reference under File and Program Exception/Errors. For V5R1 English, here is the link: http://publib.boulder.ibm.com/iseries/v5r1/ic2924/books/c092508380.htm#HDRPROGXPE

Here is an example of RPG D specs to describe the PSDS provided by Simon Coulter:

```
    D              SDS
    D PROC_NAME        *PROC                                  * Procedure
name
    D PGM_STATUS       *STATUS                                 * Status code
    D PRV_STATUS           16     20S 0                       * Previous
status
    D LINE_NUM             21     28                        * Src list line nu
    D ROUTINE          *ROUTINE                               * Routine name
    D PARMS            *PARMS                               * Num passed
parms
    D EXCP_TYPE            40     42                        * Exception type
    D EXCP_NUM             43     46                         * Exception
number
    D PGM_LIB              81     90                       * Program library
    D EXCP_DATA            91    170                          * Exception data
    D EXCP_ID             171    174                        * Exception Id
    D DATE                191    198                       * Date (*DATE fmt)
    D YEAR                199    200S 0                     * Year (*YEAR
fmt)
    D LAST_FILE           201    208                         * Last file used
    D FILE_INFO           209    243                         * File error info
    D JOB_NAME            244    253                           * Job name
    D USER                254    263                       * User name
    D JOB_NUM             264    269S 0                       * Job number
```

```
     D JOB_DATE             270    275S 0                              * Date (UDATE
fmt)
     D RUN_DATE             276    281S 0                              * Run date
(UDATE)
     D RUN_TIME             282    287S 0                              * Run time
(UDATE)
     D CRT_DATE             288    293                                 * Create date
     D CRT_TIME             294    299                                 * Create time
     D CPL_LEVEL            300    303                                 * Compiler level
     D SRC_FILE             304    313                                 * Source file
     D SRC_LIB              314    323                                 * Source file lib
     D SRC_MBR              324    333                                 * Source file mbr
     D PROC_PGM             334    343                                 * Pgm Proc is in
     D PROC_MOD             344    353                                 * Mod Proc is in
     D CURR_USER            358    367                                 * Mod Proc is in

      * Status codes
      * Normal Codes
      * Code      Condition
      * 00000     No exception/error occurred
      * 00001     Called program returned with the LR indicator on.
      * Exception/Error Codes
      * Code      Condition
      * 00100     Value out of range for string operation
      * 00101     Negative square root
      * 00102     Divide by zero
      * 00103     an intermediate result is not large enough to contain the
result.
      * 00104     Float underflow. An intermediate value is too small to be
contained in the
      *           Intermediate result field
      * 00112     Invalid Date, Time or Timestamp value.
      * 00113     Date overflow or underflow. (For example, when the result of
a Date calculation
      *           results in a number greater than *HIVAL or less than *LOVAL.)
      * 00114     Date mapping errors, where a Date is mapped from a 4
character year to a 2
      *           Character year and the date range are not 1940-2039.
      * 00120     Table or array out of sequence.
      * 00121     Array index not valid
      * 00122     OCCUR outside of range
      * 00123    Reset attempted during initialization step of program
      * 00202    Called program or procedure failed; halt indicator (H1 through
H9) not on
      * 00211    Error calling program or procedure
      * 00222    Pointer or parameter error
      * 00231    Called program or procedure returned with halt indicator on
      * 00232    Halt indicator on in this program
      * 00233    Halt indicator on when RETURN operation run
      * 00299    RPG IV formatted dump failed
      * 00333    Error on DSPLY operation
      * 00401    Data area specified on IN/OUT not found
      * 00402    *PDA not valid for non-prestart job
      * 00411    Data area type or length does not match
      * 00412    Data area not locked for output
      * 00413    Error on IN/OUT operation
      * 00414    User not authorized to use data area
      * 00415    User not authorized to change data area
      * 00421    Error on UNLOCK operation
      * 00425    Length requested for storage allocation is out of range
      * 00426    Error encountered during storage management operation
      * 00431    Data area previously locked by another program
      * 00432    Data area locked by program in the same process
      * 00450    Character field not entirely enclosed by shift-out and shift-in
characters
      * 00501    Failure to retrieve sort sequence.
      * 00502    Failure to convert sort sequence.
      * 00802    Commitment control not active.
      * 00803    Rollback operation failed.
      * 00804    Error occurred on COMMIT operation
      * 00805    Error occurred on ROLBK operation
      * 00907    Decimal data error (digit or sign not valid)
      * 00970    the level number of the compiler used to generate the
program does not agree
      *          With the level number of the RPG IV run-time subroutines
      * 09998    internal failure in ILE RPG/400 compiler or in run-time
subroutines
      * 09999    Program exception in system routine.
```

2. What is the file information data structure?

The File information data structure is the way the RPG runtime environment informs the program about the status of a file, seen from the operating system perspective. Sometimes called a feedback data structure, it is updated after every I/O operation (RPG IV) or after a block of records is transferred (RPG/400).

Information can be found in the RPG reference under File and Program Exception/Errors. For V5R1 English, here is the link: http://publib.boulder.ibm.com/iseries/v5r1/ic2924/books/c092508377.htm#HDRFILEXPE

Do note that there are example D specs at the above link.

Simon Coulter posted one example of D specs to the RPG400-L list:

```
 * Standard RPG feedback area 1-80
DINFDS           ds
D File          *FILE                                  * File name
D OpenInd             9      9                           * File open?
D EOFInd             10     10                           * File at eof?
D FileStatus     *STATUS                                   * Status code
D OpCode          *OPCODE                                    * Last opcode
D Routinr        *ROUTINE                                  * RPG Routine
D ListNum             30     37                          * Listing line
D SpclStat           38     42S 0                         * SPECIAL status
D RecordFmt       *RECORD                                       * Record name
D MsgID              46     52                           * Error MSGID

D* the next 4 fields are available after POST
D Screen_P        *SIZE                                  * Screen size
D NLSIn_P        *INP                                  * NLS Input?
D NLSOut_P        *OUT                                     * NLS Output?
D NLSMode_P        *MODE                                       * NLS Mode?

 * Open feedback area 81-240
 * NOTE that getting data beyond column 80 is expensive
 *      In terms of program opens...
D ODP_TYPE            81     82                           * ODP Type
D FILE_NAME           83     92                           * File name
D LIBRARY            93    102                          * Library name
D SPOOL_FILE         103    112                             * Spool file
name
D SPOOL_LIB         113    122                             * Spool file lib
```

```
     D SPOOL_NUM            123    124I 0                                   * Spool file
num
     D RCD_LEN              125    126I 0                                  * Max record len
     D KEY_LEN              127    128I 0                                  * Max key len
     D MEMBER               129    138                                     * Member name
     D TYPE                 147    148I 0                                  * File type
     D ROWS                 152    153I 0                                  * Num PRT/DSP
rows
     D COLUMNS              154    155I 0                                   * Num PRT/DSP
cols
     D NUM_RCDS             156    159I 0                                   * Num of
records
     D ACC_TYPE             160    161                                     * Access type
     D DUP_KEY              162    162                                     * Duplicate key?
     D SRC_FILE             163    163                                     * Source file?
     D VOL_OFF              184    185I 0                                  * Vol label offs
     D BLK_RCDS             186    187I 0                                   * Max rcds in bl
     D OVERFLOW             188    189I 0                                   * Overflow line
     D BLK_INCR             190    191I 0                                  * Blk increment
     D FLAGS1               196    196                                     * Misc flags
     D REQUESTER            197    206                                      * Requester
name
     D OPEN_COUNT           207    208I 0                                   * Open count
     D BASED_MBRS           211    212I 0                                   * Num based
mbrs
     D FLAGS2               213    213                                     * Misc flags
     D OPEN_ID              214    215                                     * Open identifie
     D RCDFMT_LEN           216    217I 0                                   * Max rcd fmt
le
     D CCSID                218    219I 0                                  * Database CCSID
     D FLAGS3               220    220                                     * Misc flags
     D NUM_DEVS             227    228I 0                                   * Num devs
defin

     D* I/O feedback area 241-366
     D* NOTE that this area is shared with the POST feedback area below!
     D                                                              * 241-242 not used
     D WRITE_CNT            243    246I 0                                   * Write count
     D READ_CNT             247    250I 0                                   * Read count
     D WRTRD_CNT            251    254I 0                                   * Write/read
count
```

```
     D OTHER_CNT             255    258I 0                                  * Other I/O
count
     D OPERATION             260    260                                    * Cuurent
operatio
     D IO_RCD_FMT            261    270                                    * Rcd format
name
     D DEV_CLASS             271    272                                    * Device class
     D IO_PGM_DEV            273    282                                    * Pgm device
name
     D IO_RCD_LEN            283    286I 0                                 * Rcd len of I/O

     D* POST area 241-nnn
     D* Display
     D PGM_DEV_P             241    250                                    * Program
device
     D DEV_DSC_P             251    260                                    * Dev
description
     D USER_ID_P            261    270                                    * User ID
     D DEV_CLASS_P            271    271                                    * Device class
     D DEV_TYPE_P            272    277                                    * Device type
     D REQ_DEV_P             278    278                                    * Requester?
     D ACQ_STAT_P            279    279                                    * Acquire
status
     D INV_STAT_P           280    280                                    * Invite status
     D DATA_AVAIL_P          281    281                                     * Data
available
     D NUM_ROWS_P            282    283I 0                                   * Number of
rows
     D NUM_COLS_P            284    285I 0                                  * Number of
cols
     D BLINK_P              286    286                                   * Allow blink?
     D LINE_STAT_P           287    287                                    * Online/offline?
     D DSP_LOC_P             288    288                                    * Display
location
     D DSP_TYPE_P            289    289                                    * Display type
     D KBD_TYPE_P            290    290                                    * Keyboard
type
     D CTL_INFO_P           342    342                                    * Controller info
     D COLOR_DSP_P            343    343                                    * Color
capable?
     D GRID_DSP_P            344    344                                    * Grid line dsp?
```

```
      * The following fields apply to ISDN.
     D ISDN_LEN_P            385    386I 0                                   * Rmt number
len
     D ISDN_TYPE_P           387    388                                      * Rmt number
type
     D ISDN_PLAN_P           389    390                                      * Rmt number
plan
     D ISDN_NUM_P            391    430                                      * Rmt number
     D ISDN_SLEN_P           435    436I 0                                   * Rmt sub-
address
     D ISDN_STYPE_P          437    438                                      * Rmt sub-
address
     D ISDN_SNUM_P           439    478                                      * Rmt sub-
address
     D ISDN_CON_P            480    480                                      * Connection
     D ISDN_RLEN_P           481    482I 0                                   * Rmt address
len
     D ISDN_RNUM_P           483    514                                      * Rmt address
     D ISDN_ELEN_P           519    520                                      * Extension len
     D ISDN_ETYPE_P          521    521                                      * Extension
type
     D ISDN_ENUM_P           522    561                                      * Extension
num
     D ISDN_XTYPE_P          566    566                                      * X.25 call
type
     D* ICF
     D PGM_DEV_P             241    250                                      * Program
device
     D DEV_DSC_P             251    260                                      * Dev
description
     D USER_ID_P             261    270                                      * User ID
     D DEV_CLASS_P           271    271                                      * Device class
     D DEV_TYPE_P            272    272                                      * Device type
     D REQ_DEV_P             278    278                                      * Requester?
     D ACQ_STAT_P            279    279                                      * Acquire
status
     D INV_STAT_P            280    280                                      * Invite status
     D DATA_AVAIL_P          281    281                                      * Data
available
     D SES_STAT_P            291    291                                      * Session status
     D SYNC_LVL_P            292    292                                      * Synch level
     D CONV_TYPE_P           293    293                                      * Conversation
typ
     D RMT_LOC_P            294    301                                      * Remote
location
     D LCL_LU_P            302    309                                     * Local LU name
     D LCL_NETID_P          310    317                                      * Local net ID
     D RMT_LU_P            318    325                                     * Remote LU
     D RMT_NETID_P          326    333                                       * Remote net
ID
     D APPC_MODE_P          334    341                                       * APPC Mode
     D LU6_STATE_P          345    345                                      * LU6 conv
state
     D LU6_COR_P            346    353                                      * LU6 conv
                                                                  *    correlator
      * The following fields apply to ISDN.
     D ISDN_LEN            385    386I 0                                    * Rmt number
len
     D ISDN_TYPE           387    388                                      * Rmt number
type
     D ISDN_PLAN           389    390                                      * Rmt number
plan
     D ISDN_NUM            391    430                                      * Rmt number
     D ISDN_SLEN           435    436I 0                                    * sub-addr len
     D ISDN_STYPE          437    438                                      * sub-addr type
     D ISDN_SNUM            439    478                                      * Rmt sub-
address
     D ISDN_CON            480    480                                      * Connection
     D ISDN_RLEN           481    482I 0                                    * Rmt address
len
     D ISDN_RNUM            483    514                                      * Rmt address
     D ISDN_ELEN           519    520                                      * Extension len
     D ISDN_ETYPE          521    521                                      * Extension
type
     D ISDN_ENUM            522    561                                      * Extension
num
     D ISDN_XTYPE          566    566                                      * X.25 call type

      * The following information is available only when program was started
      * as result of a received program start request. (P_ stands for protected)
     D TRAN_PGM            567    630                                      * Trans pgm
name
```

```
     D P_LUWIDLN            631    631                                   * LUWID fld len
     D P_LUNAMELN            632    632                                   * LU-NAME len
     D P_LUNAME             633    649                                   * LU-NAME
     D P_LUWIDIN            650    655                                   * LUWID
instance
     D P_LUWIDSEQ            656    657I 0                                  * LUWID seq
num

      * The following information is available only when a protected
conversation
      * is started on a remote system.  (U_ stands for unprotected)
     D U_LUWIDLN            658    658                                   * LUWID fld len
     D U_LUNAMELN            659    659                                   * LU-NAME len
     D U_LUNAME             660    676                                   * LU-NAME
     D U_LUWIDIN            677    682                                   * LUWID
instance
     D U_LUWIDSEQ            683    684I 0                                  * LUWID seq
num

     D* Device independent area 367-nnn
     D* NOTE that this area is shared with the POST feedback area above!
     D* Printer
     D CUR_LINE             367    368I 0                                 * Current line
num
     D CUR_PAGE             369    372I 0                                 * Current page
cnt
     D PRT_MAJOR            401    402                                   * Major ret code
     D PRT_MINOR            403    404                                   * Minor ret code

     D* Disk
     D FDBK_SIZE            367    370I 0                                 * Size of DB
fdbk
     D JOIN_BITS            371    374I 0                                 * JFILE bits
     D LOCK_RCDS             377    378I 0                                  * Nbr locked
rcds
     D POS_BITS             385    385                                   * File pos bits
     D DLT_BITS             384    384                                   * Rcd deleted
bits
     D NUM_KEYS             387    388I 0                                  * Num keys
(bin)
     D KEY_LEN              393    394I 0                                 * Key length
```

```
    D MBR_NUM              395    396I 0                                 * Member
number
    D DB_RRN              397    400I 0                                * Relative-rcd-
num
    D KEY                 401   2400                                 * Key value (max
    D*                                                           *   size 2000)

    D* ICF
    D ICF_AID             369    369                                 * AID byte
    D ICF_LEN             372    375I 0                               * Actual data len
    D ICF_MAJOR            401    402                                 * Major ret code
    D ICF_MINOR            403    404                                 * Minor ret code
    D SNA_SENSE            405    412                                  * SNA sense rc
    D SAFE_IND            413    413                                 * Safe indicator
    D RQSWRT              415    415                                 * Request write
    D RMT_FMT             416    425                                 * Remote rcd
fmt
    D ICF_MODE            430    437                                 * Mode name

    D* Display
    D DSP_FLAG1            367    368                                  * Display flags
    D DSP_AID             369    369                                 * AID byte
    D CURSOR              370    371                                 * Cursor location
    D DATA_LEN             372    375I 0                                * Actual data
len
    D SF_RRN              376    377I 0                              * Subfile rrn
    D MIN_RRN             378    379I 0                               * Subfile min rrn
    D NUM_RCDS             380    381I 0                                * Subfile num
rcds
    D ACT_CURS             382    383                                  * Active window
    D*                                                           *  cursor location
    D DSP_MAJOR            401    402                                  * Major ret code
    D DSP_MINOR            403    404                                  * Minor ret
code

     * Status codes
     * Normal Codes
     * Code      � Device    � RC       � Condition
     * 00000     �           �          � No exception/error.
     * 00002     � W         � n/a      � Function key used to end display.
     * 00011     � W,D,SQ    � 11xx     � End of file on a read (input).
     * 00012    � W,D,SQ   � n/a  � No-record-found condition on a CHAIN,
SETLL, and SETGT
     * 00013    � W        � n/a     � Subfile is full on WRITE operation.

     * Exception/Error Codes
     * Code     � Device   � RC      � Condition
     * 01011    � W,D,SQ   � n/a    � Undefined record type (input record
does not match rec
     *        �          �          � indicator).
     * 01021    � W,D,SQ   � n/a      � Tried to write a record that already
exists (file bein
     *        �          �          � key is duplicate, or attempted to write
duplicate rela
     *        �          �          � subfile).
     * 01022    � D        � n/a      � Referential constraint error detected
on file member.
     * 01023    � D,SQ     � n/a      � Error in trigger program before file
operation perform
     * 01024    � D,SQ     � n/a      � Error in trigger program after file
operation performs
     * 01031    � W,D,SQ   � n/a      � Match field out of sequence.
     * 01041    � n/a      � n/a      � Array/table load sequence error.
     * 01042    � n/a      � n/a      � Array/table load sequence error.
Alternate collating s
     * 01051    � n/a      � n/a      � Excess entries in array/table file.
     * 01071    � W,D,SQ   � n/a      � Numeric sequence error.
     * 01121(4)  � W       � n/a      � No indicator on the DDS keyword for
Print key.
     * 01122(4)  � W       � n/a      � No indicator on the DDS keyword for
Roll Up key.
     * 01123(4)  � W       � n/a      � No indicator on the DDS keyword for
Roll Down key.
     * 01124(4)  � W       � n/a      � No indicator on the DDS keyword for
Clear key.
     * 01125(4)  � W       � n/a      � No indicator on the DDS keyword for
Help key.
     * 01126(4)  � W       � n/a      � No indicator on the DDS keyword for
Home key.
     * 01201    � W        � 34xx     � Record mismatch detected on input.
     * 01211    � all      � n/a      � I/O operation to a closed file.
     * 01215    � all      � n/a      � OPEN issued to a file already opened.
     * 01216(3)  � all      � yes      � Error on an implicit OPEN/CLOSE
operation.
     * 01217(3)  � all      � yes      � Error on an explicit OPEN/CLOSE
operation.
     * 01218     � D,SQ     � n/a      � Record already locked.
     * 01221     � D,SQ     � n/a      � Update operation attempted without
a prior read.
     * 01222     � D,SQ     � n/a      � Record cannot be allocated due to
referential constraint
     * 01231     � SP       � n/a      � Error on SPECIAL file.
     * 01235     � P        � n/a      � Error in PRTCTL space or skip entries.
     * 01241     � D,SQ     � n/a      � Record number not found.  (Record
number specified in
     *           �          �          � present in file being processed.)
     * 01251     � W        � 80xx 81xx � Permanent I/O error occurred.
     * 01255     � W        � 82xx 83xx � Session or device error occurred.
Recovery may be pos
     * 01261     � W        � n/a      � Attempt to exceed maximum number
of acquired devices.
     * 01271     � W        � n/a      � Attempt to acquire unavailable device
     * 01281     � W        � n/a      � Operation to un acquired device.
     * 01282     � W        � 0309     � Job ending with controlled option.
     * 01284     � W        � n/a      � Unable to acquire second device for
single device file
     * 01285     � W        � 0800     � Attempt to acquire a device already
acquired.
     * 01286     � W        � n/a      � Attempt to open shared file with
SAVDS or IND options.
     * 01287     � W        � n/a      � Response indicators overlap IND
indicators.
     * 01299     � W,D,SQ   � yes      � Other I/O error detected.
     * 01331     � W        � 0310     � Wait time exceeded for READ from
WORKSTN file.
     * ------------------------------------------------------------------------------------
     * Notes:
     *   (3) Any errors that occur during an open or close operation
     *       will result in a *STATUS value of 1216 or 1217
     *       regardless of the major/minor return code value.
     *   (4) See Figure 9 in topic 1.3.2.4 for special handling.
     * AID byte map
    D KeyF1           s              1    inz(x'31')
```

```
D KeyF2           s              1    inz(x'32')
D KeyF3           s              1    inz(x'33')
D KeyF4           s              1    inz(x'34')
D KeyF5           s              1    inz(x'35')
D KeyF6           s              1    inz(x'36')
D KeyF7           s              1    inz(x'37')
D KeyF8           s              1    inz(x'38')
D KeyF9           s              1    inz(x'39')
D KeyF10          s              1    inz(x'3A')
D KeyF11          s              1    inz(x'3B')
D KeyF12          s              1    inz(x'3C')
D KeyF13          s              1    inz(x'B1')
D KeyF14          s              1    inz(x'B2')
D KeyF15          s              1    inz(x'B3')
D KeyF16          s              1    inz(x'B4')
D KeyF17          s              1    inz(x'B5')
D KeyF18          s              1    inz(x'B6')
D KeyF19          s              1    inz(x'B7')
D KeyF20          s              1    inz(x'B8')
D KeyF21          s              1    inz(x'B9')
D KeyF22          s              1    inz(x'BA')
D KeyF23          s              1    inz(x'BB')
D KeyF24          s              1    inz(x'BC')
D KeyFClr         s              1    inz(x'BD')
D KeyFEnt         s              1    inz(x'F1')
D KeyFHlp         s              1    inz(x'F3')
D KeyFRDn         s              1    inz(x'F4')
D KeyFRUp         s              1    inz(x'F5')
D KeyFPrt         s              1    inz(x'F6')
D KeyFBksp        s              1    inz(x'F8')
D KeyFAutoEnter   s              1    inz(x'3F')
```

3. CL Parameter Basics

When a variable is declared within a CL program, the system assigns storage for that variable within the program automatic storage area (PASA). If you subsequently use the variable as a parameter within a CALL command, the system does not pass the value of that variable to the called program, but rather a pointer to the PASA of the calling program. This is known as parameter passing by reference.

For this reason, it is very important that both programs declare the parameter to be of the same type and size. To illustrate, let's look at the following example:

```
PgmA: Pgm

  DCL &Var1 *CHAR 2 Inz( 'AB' )
  DCL &Var2 *CHAR 2 Inz( 'YZ' )

  Call PgmB Parm( &Var1 &Var2)

EndPgm
```

```
PgmB: Pgm Parm( &i_Var1 &i_Var2 )

  DCL &i_Var1 *CHAR 4
  DCL &i_Var2 *CHAR 2

EndPgm
```

Hopefully, you've noticed that the first parameter is declared to be larger in PgmB than it was in PgmA. Although you might expect &i_Var1 to contain 'AB ' after the call, the following is what the input parameters in PgmB actually contain:

&i_Var1 = 'ABYZ'
&i_Var2 = 'YZ'

&i_Var1 shows the contents of the first parameter, and the second, because the second parameter is immediately adjacent to the first within the storage area. If the second parameter was not contiguous to the first, then the last two bytes of &i_Var1 would show whatever happened to be in the storage area at that time.

You can think of &i_Var1 as a 4-byte "window" into the storage area of the calling program. It's passed a pointer that tells it where the view begins, and it accesses anything in storage from that point up to the parameter's declared length.

Looking at Literals

There are several ways that a program can be called, other than from another program. Examples include the command line, SBMJOB, job scheduler etc. In the case of an interactive call from the command line, you specify the parameters as literals, ie:

Call PgmB Parm('AB' 'YZ')

Consider that when we do this, there is no PASA. We'll look at the implications of that in a minute, but for now; just make a note of it. Submitting a job from the command line isn't any different. If you're submitting a CALL, then you'll be specifying any associated parameters as literals. However, things can get a bit deceiving when you submit a job from *within* a program, as the following example illustrates:

```
PgmC: Pgm

 DCL &Var1 *CHAR 2 Inz( 'AB' )
 DCL &Var2 *CHAR 2 Inz( 'YZ' )

 SbmJob Cmd(Call PgmB Parm( &Var1&Var2))

EndPgm
```

Clearly, we're not passing literals here. Or are we?

Let's think about how things would work if we passed variables:

- PgmC submits a call to PgmB, passing two variables as parameters.
- PgmC immediately ends as a result of the EndPgm statement.
- PgmB begins running in batch and receives pointers to PgmC's PASA.
- PgmB crashes when it attempts to use the pointers.

We have invalid pointers because PgmC is no longer running. If you've ever tried this personally, you know that it doesn't happen in practice. The reason for that is that the system is converting those variables to *literals* before issuing the CALL command. It’s very sneaky, but effective.

Now that we've seen some examples of where literals are used, and why, it's time to talk about the PASA again. When we discussed the basics of CL parameter passing, we learned that the called program expects to receive a pointer to a storage area within the PASA for each input parameter. This requirement hasn't changed. So now we have a situation where the CALL command is passing literals, but the called program is still expecting pointers.

Obviously, it's time for the system to perform some more magic behind the scenes. In order to accommodate the requirements of the called program, the system creates a space in temporary storage for each literal being passed, and moves the value of the literal into that storage space. Now it can pass pointers to the called program, and everyone is happy.

Except you that is, because none of this changes the fact that you're getting "garbage" in the input variables of your called program! Fair enough. I'm getting to that now, but you needed the background in order to understand the next part.

Sizing It All Up

Now that you know the system is creating variables behind the scene, you might wonder how it knows what size those variables need to be. The answer is that it doesn't. Instead, the designers have imposed some specific rules about how literals are transformed to variables, and thereby passed as parameters.

CL supports only three basic data types: *character*, *decimal*, and *logical*. For the purposes of this discussion, you can consider the logical data type equivalent to the character type, because it's treated in the same manner.

The simplest rule is the one that handles decimal literals. All decimal literals will be converted to packed decimal format with a length of (15 5), where the value is 15 digits long, of which 5 digits are decimal places. Therefore, any program that you expect to call from the command line, or SBMJOB etc., needs to declare its numeric input parameters as *DEC(15 5).

Character literals are a little bit more complicated, but still fairly straightforward. There are two rules to remember. The first is that any character literal up to 32 characters in length will be converted to a 32 byte variable. The value is left justified, and padded on the right with blanks.

So if you were to pass the following literal:
Call PgmB 'AB' the associated storage space for that literal would contain:
'ABxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' *(where "x" represents a blank space)*

The second rule is that character literals longer than 32 bytes are converted to a variable of the same length as the literal value itself, as in the following example:

Call PgmB 'This is a long character literal that will exceed 32 bytes.' the associated storage space for that literal would contain:

'This is a long character literal that will exceed 32 bytes.'

Finally, since the logical data type follows the same rules as the character type, and the only possible values for a logical data type are '0' or '1', we know that a logical literal will always be created as a 32 byte, left justified, padded character variable.

Parameter Problems

In the beginning of this explanation, you learned that it was important for the parameter declarations to match between a called program and its caller. Then you discovered that the system sometimes has to take it upon itself to declare the parameters of the caller on your behalf. If the two declarations don't match, we have the potential for trouble.

In the case of a decimal value, the result is immediate and obvious; you get a data decimal error. Character variables are more difficult to debug because they don't generate any immediate errors. What actually happens depends upon the length of the parameter in the called program. If the length of the parameter in the called program is **less than** the length of the parameter being passed, the extra characters are effectively truncated, as follows:

```
Call SomePgm ('ABCDEFG') /* system creates 32 byte *CHAR*/

SomePgm: Pgm Parm( &i_Var1 )

 DCL &i_Var1 *CHAR 4

EndPgm
```

What happens is that the system passes 'ABCDEFGxxxxxxxxxxxxxxxxxxxxxxxxx' *('x' is a blank)*, but because of the declared length of &i_Var1, SomePgm only see's 'ABCD'. For most of us, this is the behavior that we would expect.

Things get nasty when the declared length of the variable is **longer than** what is being passed in. Using the same example as we've just seen above:

```

SomePgm: Pgm Parm( &i_Var1 )

 DCL &i_Var1 *CHAR 34

EndPgm

```

In this case, the system will still allocate 32 bytes of storage and assign 'ABCDEFGxxxxxxxxxxxxxxxxxxxxxxxxx' to it, but because &i_Var1 is now declared to be 34 bytes long, SomePgm will see more storage than it was intended to. It will see the 32 bytes that were allocated for it, **plus** two additional bytes. It's those two additional bytes that can cause the infamous "unpredictable results" which IBM's documentation often refers to.

If the extra bytes contain blanks, chances are that you won't notice a problem, but if they contain something else, your input parameter will contain "garbage".

As you can see, *when dealing with literals*, the magic number for character parameters is 32. If the called program declares the parameter to be less than or equal to 32, you'll never see "garbage" in the parameter. Once you cross that 32 byte threshhold, you need to take extra care to ensure that the size of the literal being passed is equal to the declared size of the input parameter.

Things to Remember

- Always match the type/size of parameters on your pgm to pgm calls.
- Remember that the system converts literals to variables in the background.
- Remember that decimal literals are always converted to *DEC(15 5) and that char literals less than or equal to 32 bytes are converted to *CHAR(32) and that char literals greater than 32 bytes are converted to variables of equivalent size.

And last, but not least: the called program "sees" as much storage as it declares for an input parameter, regardless of whether or not the caller actually allocated that much storage for it.

Here is an example of using a command with numeric parameters:
http://faq.midrange.com/data/cache/197.html

How to easily create a CMD object to start your CL program and avoid these errors:
http://faq.midrange.com/data/cache/576.html

1. **Calling program TSTCALL code:**

```
D Result          S             5A   Inz('A')
D Factor1         S             5A   Inz('B')
D Factor2         S             5A   Inz('C')
```

```
D Var1             S             5A   Inz('X')

C                      Call      'TSTENTRY'
C     Factor1       Parm      Factor2       Result

C                      Eval      Result = 'TEST'
C     Result         Dsply

C                      Eval      *Inlr = *on
```

Called program (TSTENTRY) code:

```
D Res              S             5A
D Fact1            S             5A
D Fact2            S             5A

C     *Entry        Plist
C     Fact1          Parm      Fact2         Res
C                      Eval      *Inlr = *on
```

Description of the values in Factor 1, Factor 2 and Result:
While calling TSTENTRY
Factor1 = B
Factor2 = C
Result = A

Soon after entering TSTENTRY - Factor 2 gets moved to Result. Factor 1 and Factor 2 gets cleared
Factor1 = ' '
Factor2 = ' '
Result = C

Before leaving TSTENTRY - Result is moved to Factor 1. Now Result is not cleared after value is moved.
Factor1 = C
Factor2 = ' '
Result = C

After executing *Inlr = *on of the called program TSTENTRY
Factor1 = C
Factor2 = ' '
Result = ' '

After returning to the calling program TSTCALL - Factor 1 gets moved to Factor 2. Factor 1 gets cleared.
Factor1 = ' '
Factor2 = C
Result = ' '

2. Display Program References (DSPPGMREF)

The Display Program References (DSPPGMREF) command provides a list of the system objects referred to by the specified programs. The following list shows the system objects provided for the respective program types:

CL
*FILE, *PGM, and *DTAARA
CLE
*SRVPGM
CLLE
*FILE, *PGM, *DTAARA, and *SRVPGM
RPG
*FILE, *DTAARA, and *PGM
RPGLE
*FILE, *PGM, *DTAARA, and *SRVPGM

This information can be displayed, printed, or placed in a database output file.
If the information is shown or printed, a list (by library) of the specified user-authorized programs, along with the objects referenced by each program, is created. For files, information about how each file is used (input, output, update, unspecified, or any combination of these four) is also shown or printed.

If the information is written to a database file, the database file will have a record format named QWHDRPPR. The fields in record format QWHDRPPR are the same as the fields in the IBM-supplied format QWHDRPPR in file QADSPPGM in the library QSYS. The following information is contained in the database file:

- The name of the program and its text description
- The name of the library containing the program
- The number of objects referenced by the program

- The qualified name of the system object
- The information retrieval dates
- The object type of the referenced object

For files, the record contains the following additional fields:

- The name of the file in the program (possibly different from the system object name if an override was in effect when the program was created)
- The program use of the file (1=input, 2=output, 4=update, 8=unspecified, or a number representing a combination of any of these four; for example, a code of 11 is a combination of 1, 2, and 8, which is input, output, and unspecified)
- The number of record formats referenced, if any
- The name of the record format used by the file and its record format level identifier
- The number of fields referenced for each format

Note: This command lists which objects are referenced when the object is created or updated using UPDPGM or UPDSRVPGM. The referenced object names and libraries listed may be different than the actual names of the objects, since this information is stored when the program is created. Entries can be added as the ILE program or service program is updated using UPDPGM or UPDSRVPGM, but entries are never removed. If the object has been moved since the program was created, or an override was in effect during creation, the names listed may differ from the actual names.

Restrictions:

1. The user must have object operational authority for the program.
2. Also, of the libraries specified by the library qualifier, only the libraries for which the user has read authority are searched for the programs.

Examples:

```
DSPPGMREF   PGM(LIBRARY1/*ALL)  OUTPUT(*OUTFILE)
            OUTFILE(LIB2/FILE2)
```

This command creates a list of all authorized programs found in LIBRARY1, and of the files and other system objects that the programs reference. It stores the list in a database file named FILE2 in LIB2.

```
DSPPGMREF   PGM(LIBRARY1/BILLING)  OUTPUT(*PRINT)
```

This command creates a list of system objects that are referenced by the BILLING program in LIBRARY1. The output is spooled for printing.

1. **Difference between *Omit and *no pass:**

Omitting Parameters:
When calling a program or procedure, you may sometimes want to leave out a parameter. It may be that it is not relevant to the called procedure.

If you need to omit a parameter on a call, you have two choices:
- Specify OPTIONS(*OMIT) and pass *OMIT
- Specify OPTIONS (*NOPASS) and do not pass the parameter.

The primary difference between the two methods has to do with how you check to see if a parameter has been omitted. In either case, an omitted parameter cannot be referenced by the called procedure; if it is, unpredictable results will occur. So if the called procedure is designed to handle different numbers of parameters, you will have to check for the number of parameters passed. If *OMIT is passed, it will 'count' as a parameter.

Passing *OMIT
You can pass *OMIT for a prototyped parameter if the called procedure is aware that *OMIT might be passed. In other words, you can pass *OMIT if the keyword OPTIONS (*OMIT) is specified on the corresponding parameter definition in the prototype. When *OMIT is specified, the compiler will generate the necessary code to indicate to the called procedure that the parameter has been omitted.

Note:
*OMIT can only be specified for parameters passed by reference.
To determine if *OMIT has been passed to an ILE RPG procedure, use the %ADDR built-in function to check the address of the parameter in question. If the address is *NULL, then *OMIT has been passed. You can also use the CEETSTA (Check for Omitted Argument) bind able API.
Ex:

```
 *-----------------------------------------------------------------
 * CEETSTA (Test for omitted argument) -- ILE CEE API
 *     1. Presence flag                  Output   Binary(4)
 *     2. Argument number                    Input    Binary(4)
 *-----------------------------------------------------------------
D CEETSTA         PR                  EXTPROC('CEETSTA')
D   Present                  10I 0
D   ArgNum                    10I 0   CONST
```

```
 D   Feedback                  12A     OPTIONS(*OMIT)
...
 D HaveParm        S            10I 0
...
 C                   CALLP    CEETSTA(HaveParm : 3 : *OMIT)
 C                   IF       HaveParm = 1
 *       do something with third parameter
 C                   ENDIF
```

Leaving Out Parameters
The other way to omit a parameter is to simply leave it out on the call. This must be expected by the called procedure, which means that it must be indicated on the prototype. To indicate that a prototyped parameter does not have to be passed on a call, specify the keyword OPTIONS (*NOPASS) on the corresponding parameter definition. Note that all parameters following the first *NOPASS one must also be specified with OPTIONS (*NOPASS).

You can specify both *NOPASS and *OMIT for the same parameter, in either order, that is, OPTIONS (*NOPASS:*OMIT) or OPTIONS (*OMIT:*NOPASS).

As an example of OPTIONS (*NOPASS), consider the system API QCMDEXC (Execute Command) which has an optional third parameter.

```
 *------------------------------------------------------------
 * This prototype for QCMDEXC defines three parameters:
 *   1- a character field that may be shorter in length
 *      Than expected
 *   2- Any numeric field
 *   3- An optional character field
 *------------------------------------------------------------
 D qcmdexc         PR                  EXTPGM('QCMDEXC')
D   cmd                     3000A   OPTIONS(*VARSIZE)  CONST
D   cmdlen                     15P 5 CONST
D                           3A   CONST OPTIONS(*NOPASS)
```

1. **What do we mean by externalizing?**

By externalizing we mean that all READ, CHAIN, and other database operations are located in separate routines and programs that require I/O make requests to these routines to perform the operation on their behalf.

2. What will FOR opcode will do?

SKANDASAMO/DOLOOP

FOR

```
*************** Beginning of data *****************************
di              s              5p 0 inz(1)
dn              s              5p 0 inz(10)
c                   for       i=1 to n
c     i             dsply
c                   endfor
c                   for       i=n downto 1
c     i             dsply
c                   endfor
*for(I=5;I<40;i=i+10)
c                   for       i=5 by n to 40
c     i             dsply
c                   endfor
c                   for       i=5 to 40 by n
c     i             dsply
c                   endfor
c                   seton                                        lr
****************** End of data *****************************
```

OUTPUT

```
DSPLY 1       DSPLY    10
DSPLY 2       DSPLY    9
DSPLY 3       DSPLY    8
DSPLY 4       DSPLY    7
DSPLY 5       DSPLY    6
DSPLY 6      DSPLY    5
DSPLY 7      DSPLY    4
DSPLY 8      DSPLY    3
DSPLY 9      DSPLY    2
DSPLY 10     DSPLY    1
DSPLY 5      DSPLY   5
DSPLY 15     DSPLY  15
DSPLY 25     DSPLY   25
DSPLY 35     DSPLY    35
```

3. What are the various stages for a job after it is submitted?

Job queue, Active job, and OUTQ are the three stages after the job has been submitted.

4. What is an activation group?

Activation group is the boundary set for similar programs. Activation group is also a storage space in memory.

- ✔ CLP has a OVRDBF command and calls a RPGLE program
- ✔ RPGLE program performs a read operation and the pointer is now in the second program and now call the program 3
- ✔ RPGLE program that also do a read operation which will read the second record
- ✔ Since the pointer is in the second position and then return to 2nd program in the above situation when the control transfer from 3rd record since the 2 rd record is already read in program 3. But we need to need the 2nd record according to the logic but this is not possible in any OPM programs. But in RPGLE there is a solution for this problem by giving a common activation group for 1st and 2nd program and have a separate activation group for the 3rd program while creating the program itself and this will avoid the entire problem we faced before.
- ✔ In some situation we want to share between 2 program then we can give the activation group in **job* level in which the changes in one program will be affected in another program.

❖ **Types of activation group levels:**

  - ✔ ***New:** In this case every time you call the program an new activation group will be created which this case will not be used mostly.
  - ✔ ***caller:** If we don't know the type of the program that is calling then we can specify *caller where the activation group will be the same of the program that is calling.
  - ✔ **Named activation group:** We can give our own named for different activation group**.**

1. **What are the statements that are affected by activation group?**
   - ✔ OVRDBF
   - ✔ OPNDBF
   - ✔ OPNQRYF
   - ✔ STRCMTCTL
   - ✔ DLTOVR

1. **How to see source of copybooks include in a program while compiling or debugging?**

While compiling the program give *list instead of *source which will expand all the copybooks.

2. **Explain keyword in ILE?**

- ❖ Overlay
- ❖ Rename
- ❖ Prefix
- ❖ Options
- ❖ Const

1. **How you can schedule a job to run periodically?**

We have to create a job scheduler for running a job periodically. Here we can make a job to run once or periodically at a given date and time. We can create a job scheduler by using ADDJOBSCDE command.

We can list all the job scheduler running by using the command WRKJOBSCDE command and we can delete a job scheduler by using RMVJOBSCDE or we can reschedule the job by using CHGJOBSCDE.

2. **How you can import and export a data type between 2 programs?**

If you are using an export statement when declaring a variable then the data type can be imported in any modules that is bind either by value or by reference. So in this case we can pass values in between modules instead of using PLIST and *ENTRY.

3. **Navigation between two screens**

```
                                  SKANDASAMS/TABLES

                                          EX21

   *************** Beginning of data *****************************

0001.00 FEXDSPF    CF   E            WORKSTN

0002.00 C                  Z-ADD     1           SCR1             2 0

0003.00 C     *IN03        DOWEQ     *OFF
```

```
0004.00 C     SCR1          DOWEQ     1
0005.00 C                 EXFMT     DSPF1
0006.00 C   03              LEAVE
0007.00 C                 IF        *IN08=*ON
0008.00 C                 Z-ADD     2             SCR1
0009.00 C                 LEAVE
0010.00 C                 ENDIF
0011.00 C                 ENDDO
0012.00 C     SCR1          DOWEQ     2
0013.00 C                 EXFMT     DSPF2
0014.00 C   03              LEAVE
0015.00 C                 IF        *IN07=*ON
0016.00 C                 Z-ADD     1             SCR1
0017.00 C                 LEAVE
0018.00 C                 ENDIF
0019.00 C                 ENDDO
0020.00 C   03              LEAVE
0021.00 C                 ENDDO
0022.00 C                 SETON                                   LR
        ****************** End of data *******************************
```

4. Define indicator & MOVEA?

It is a 1-bit flag where value will be either 0 or 1 AS/400 provider 99 indicators for the business user.

- 1-24 ->assigned functions keys
- 25-99 ->our own purpose

1. Define ITER / LEAVE/DO/Dow?

ITER-> Transfer the control before do loop.

LEAVE-> Transfer the control after do loop.

DOU →Checks after entering the loop, it is performed at least once.

DOW ▪ Checks before entering into the loop.

2. Explain Assume and Overlay?

Assume

Type Y (Yes) to select the ASSUME keyword. It causes the AS/400 system to assume that this record appears on the display when the file is opened. Use this keyword to receive data that a previous program has left on the display.

Overlay

Type Y (Yes) to allows the overlaying of fields on the record without erasing the entire display. Note: If you type anything other than Y or blank, your entry will be ignored. You must specify the OVERLAY keyword to select the other keywords on this display, with the exception of PUTOVR.

3. Why externalize?

Why would you want to do this? Perhaps we can best answer that question by posing one of our own. Suppose that during discussions with your users, it becomes apparent to you that the business needs have changed. After having studied the new requirements for a while, you realize that you need to redesign the database to accommodate these changes.

Would you:

• Modify your database and then locate all relevant I/O operations and modify them as required.

• Decide that doing the right thing is just too much work, and just "hack" the database one more time.

Externalizing the I/O operations of databases provides one way of helping to ensure that your applications can adapt quickly and (relatively) painlessly to changing business needs. Instead of coding a READ, CHAIN, or whatever at each point in the program where database access is required, you invoke a routine to perform the I/O for you.

4. What is the disadvantage of using Validity Check keyword? How to overcome these disadvantages?

If invalid values are entered,

- The option filed is displayed in reverse image.
- System defined message is displayed which may not be user friendly.
- Keyboard is locked, we have to reset it.
- To overcome above disadvantage validations is done within program and user friendly/defined message is displayed.

1. **Chain:**

   Chain is a combination of SETLL and READE

2. **Which of the following operations does NOT zero the field FLDA defined as 4,0?**

   C MOVE *ZEROS FLDA
   C Z-ADD *ZEROS FLDA
   C Z-ADD 0 FLDA
   C MOVE *ALL'0' FLDA
   C SUB FLDA FLDA
   C MOVE '0000' FLDA
   C CLEAR FLDA
   C MOVE *BLANKS FLDA

   The last instruction does NOT zero the field FLDA.

3. **How can you check for a records existence without causing and I/O (CHAIN/READ)?**
   With the help of File Information Data Structure, we can check existence of records in a physical file. The code is described below:
   In File description continuation line (IPFK)

   KINFDS RCDS

   IRCDS DS
   I *RECORD #RCDS
   with the above code we can check the existence of records in a file without causing I/O operation.

4. **What is the difference between UDATE and the system date?**
   UDATE supports two-digit year. The format is *MDY (MMDDYY).
   *DATE (system date) supports four digit year. The format is *MDYY (MMDDYYYY).

5. **Describe the difference between the DOWxx and DOUxx operations?**
   DOWxx : If the condition becomes true, then only the group of instructions allowed executing.
   DOUxx : Irrespective of condition, it will execute at least one time.

6. **Define the purpose of the ITER operation?**
   If you specify the ITER, the groups of statements are allowed to execute repeatedly.

7. **List the steps/commands necessary to accomplish the following:**
   a. Copy data from the file ORDHDR into file ORDHIST
   b. The file ORDHIST may or may not exist
   c. If the file ORDHDR does exist, it may or may not contain data
   d. The file ORDHIST may or may not contain data, if the file does contain data the old data should be erased

   Commands: a. CPYF FILE(ORDHDR) TOFILE(ORDHIST)
   b. CPYF FILE (ORDHDR) TOFILE (ORDHIST) CRTFILE (*YES)
   c. CPYF FILE (ORDHDR) TOFILE (ORDHIST) *ADD
   d. CPYF FILE (ORDHDR) TOFILE (ORDHIST) *REPLACE

8. **What is the purpose of the following?**

FORDHDR1 IF E K DISK
ORDHDRF KRENAMEORDHDRF1

In order to rename the record format of a data base file in a program, we can use the above steps. Purpose of renaming is: If the record format name is similar in two files and if both are used in a same program, the program will not compile. Hence we have to rename either of the file.

9. **What is the purpose of the following**
   C/COPY QRPGSRC,ORDERR

During the compilation the source code of ORDERR copy book is copied into the existing program. Whereas /COPY is compiler directive statement.

10. **What is the purpose of the following? A CSRLOC (F1ROW F1COL)**

Using this record level keyword, you can specify cursor location on an output operation to the record format you are defining. The program sends output after setting the cursor location.

11. **What is the difference between SFLCLR and SFLINZ?**
    SFLCLR: It clears the subfile.
    SFLINZ: First it clears the subfile and initializing the numeric variables with zeros and alphanumeric variables with characters.

12. **Define the purpose/use for SFLRNA?**
    Using this, we can make specified subfile record format inactive.

13. **How can you detect and handle a record lock situation?**
    If you try to read the locked record, we can get system defined message i. e. , the

program will ended abnormally. With the help of File Information Data Structure we can handle record lock situation. Generally it will happen, when the same file of type " U" used in different programs.

14. **How can you detect overflow for a print program that prints multiple lines per cycle?**
You specify the indicators OA through OG and OV in 33 - 34 columns in a printer file. This indicator automatically set on whenever overflow occurs on a type of page.

15. **How would you design the process for a nightly, high volume check producing process that needs to select only records that are flagged to be processed?**
With the help of OPNQRYF Clp command, we can select the records from the data base file. The process involves following steps:

Steps: 1. OVRDBF with SHARE (*YES)
2. OPNQRYF
3. CALL the program
4. DLTOVR
5. CLOF

16. **How would you join 3 separate fields, a first name, middle initial and last name together as 1 field with proper spacing? You can describe in either RPG and/or RPG ILE (Integrated Language Environment)**

```
MOVE 'Dr. ' FNAME 3
MOVE 'JOHN' MNAME 4
MOVE 'WATSON' LNAME 6
FNAME CAT MNAME: 1 VAR1 8
VAR1 CAT LNAME:1 VAR2 15
DSPLY VAR2
MOVE *ON *INLR
```

17. **When PGMA calls PGMB for the first time PGMB executes the *INZSR. PGMB uses the RETRN operation to return to PGMA. When PGMA call PGMB the second time is the *INZSR executed?**

If you specify RETRN in called program, the *INZSR will not execute again.

18. **Show 2 ways to convert a date from YYMMDD to MMDDYY (MULT operation not acceptable)**

1) CVTDAT DATE() RTNVAR( ) FROMFMT( ) TOFMT( )
Source code is required to convert from one date format to another date format. The source code in CLP is given below:

```
PGM
DCL VAR(&VAR1) LENGTH(6) TYPE(*CHAR) VALUE('YYMMDD')
DCL VAR(&RCVD) LENGTH(6) TYPE(*CHAR)
DCL VAR(&VAR2) LENGTH(4) TYPE(*CHAR)
DCL VAR(&VAR3) LENGTH(2) TYPE(*CHAR)
CHGVAR VAR(&VAR2) VALUE(%SST(&VAR1 3 4))
CHGVAR VAR(&VAR3) VALUE(%SST(&VAR1 1 2))
CHGVAR VAR(&RCVD) VALUE(&VAR2 *CAT &VAR3)
SNDMSG MSG(&RCVD) TOUSR(*USRPRF)
ENDPGM
```

19. Determine the value of the result field

    a. Cost = $110. 00
    b. Tax = 20%
    c. MarkUp= 05%
    d. Sale = 10%

    C Eval TotalCost = ((Cost * MarkUp) * Tax)) - Sale
    = 1. 0$

20. **Define the purpose of Factor 1 the Operation Code and *IN15 in following code**
    HI LO EQ
    C *YMD Test(D) yymmddDate 15

    If the factor 1 value matches with factor2 value, the indicator specified in EQ comes *ON.

21.Describe the function of SETLL operation in RPG language?

The SETLL operation positions a file at the next record with a key or relative record number that is greater than or equal to key or relative record number specified in factor1.

22.Describe the function of SETGT operation in RPG language?

The SETGT operation positions a file at the next record with a key or relative record number that is greater than key or relative record number specified in factor 1.

23. What is the purpose of Level Check parameter in a Physical file?

    Specifies whether the level identifiers of the record formats in the physical file are checked when the file is opened by the program.

24. **Define a Job Queue?**
    Job queues are queues of batch jobs waiting to be processed.

25. **Define an Output Queue?**
    Output queues are queues of jobs waiting to be printed.

26. **What is the function of CPYSPLF command?**
    It copies the spooled file to the data base file.

27. **What is the function of CPYF command?**
    To copy the data from the one file to another.

28. **What is the function of CRTDUPOBJ command?**
    To create the replica from the original object.

29. **Define Subsystem?**
    Subsystem is nothing but it provides specialized environment to complete the execution of jobs.

30. **What are different types of Substems?**
    QBATCH, QINTER, QSPL, QCMN, QCTL, QBASE.

31. **Define a Batch Job?**
    * A user requests the job.
    * The job is created (job name is assigned, job attributes are allocated)
    * The job is placed on a job queue
    * The sub system QBATCH takes the job from job queue and starts it.
    * Output generated by the batch job is placed on an output queue.
    * The spool sub system prints the output on the output queue.

32. **Describe about Query/400?**
    Query/400 is a licensed program that uses a query to analyze and select the information contained in the data base files and create a query report.
    A query report can be:
    * displayed on a workstation (screen)
    * printed
    * stored in another database file.

33. **What is the CLP command to access a Query/400?**
    WRKQRY

34. **Purpose of Overrides?**
    The basic purpose of Overrides is to temporarily change the attributes of a file. So you don't have to create permanent files for every combination of attributes your application might need. Overrides gives you the flexibility to use existing model files and dynamically change their attributes.

35. **Define Data Structure?**
    Data structures are specified in the Input specifications of an RPG/400 program to define an area in storage and layouts of related sub fields.

36. **What is the purpose of Data structure?**
* Divide a field in to sub fields
* Change the format of a field
* Group non-contiguous data in a contiguous format
* Define an area of storage in more than one format
* Define Multiple occurrences of data structures.

37. **List and explain the different type of data structures?**
* Data area data structure
when the data area is defined in an RPG/400 program as a data area data structure, its data is implicitly retrieved for processing and written back at the end of the program. In the data area data structure, letter "U" must be entered to define the data structure as a data area data structure.

* File information data structure
a file information data structure provides exception/error information that may be occurred when processing a file during program execution. This type of data structure contains pre defined sub fields that identify
* The name of the file for which the error occurred
* The record processed when the error occurred
* The operation being processed when the error occurred
* The status code number
* The RPG/400 routine in which the error occurred.
Exception errors may be controlled by testing for an error code in the *STATUS field which is included in a file information data structure. Specifically, keywords including *FILE, *RECORD, *OPCODE, *STATUS, *ROUTINE provide the previously named information.

Program status data structure
Program status data structure however identity exception/errors that are generated in the program by RPG/400 operations and not by a file. Note that any code greater than 00099 is flagged as an exception/error. Four keywords - *STATUS, *ROUTINE, *PROGRAM, *PARMS are supported by a program status data structure.

38. **What is the purpose of DYNSLT keyword?**
This is a file level keyword used in a logical file. If you specify this in a file level, the system doesn't perform record selection until the program reads file. Then on the Select/Omit criteria, it selects the records from the specified file.

39. **What is the difference between access path and Dynamic select?**
Dynamic select occurs whenever the program reads file. But access path occurs before the file is read (but not necessarily). Because access path maintenance performed on the file.

40. **Why would you prefer OPNQRYF than logical file?**
The main difference is: Logical file creates permanent object on the system. OPNQRYF creates temporary access path.

41. **What is the difference between Packed decimal and Zoned decimal?**
    Packed decimal: One digit occupies 1 byte.
    Zoned decimal: One digit occupies 2 bytes.

42. **What is default data type (if you define decimals '0') in Physical file?**
    Packed decimal

43. **What is default data type for the fields(sub fields) defined in data structures in RPG?**
    Zoned decimal

44. **When do you explicitly open files and close files in an RPG program?**
    If you specify the letter ' U ' at column 73-74, you need to be open and close files explicitly in a RPG program.

45. **What is Spool file, why is it required?**
    A file that holds output data to be processed, such as information waiting to be printed.

46. **What is Job, What are the attributes of a Job?**
    A Job is a basic unit of work on AS/400.
    The attributes are:
    Job Number Unique system generated sequential number
    Job Name Any user defined name (Max 10 char)
    User Name Who initiated the job.

47. **What is Sub-System?**
    Sub-Systems are specific user defined partitions of the CPU where various jobs may be executed. One subsystem can have more than one active job at a time.

48. **What is a Device file?**
    A device file contains the description of how data is to be presented to a program from a device or vice versa. Device file can be Printer, Disk, Tape and Remote system.

49. **How can a data area be locked after being updated?**
    Using OUT *LOCK

50. **What are the types of object authorities?**
    *USE, *CHANGE, *ADD, *DLT, *READ, *UPD, *ALL, *EXCLUDE, *OBJEXIST, *OBJMGT, *OBJOPR

51. **What is the use of OVRPRTF?**
    Override with Printer file (OVRPRTF) command is used to override certain parameters of the printer files used in the program or to replace the printer file.

52. **What is Subfile?**
    Subfile is group of records of same record format and can be read from or write to the display in a single operation.

53. **What are all the contents of subfile?**
    Subfile Record Format, Subfile Control Record Format, Relative Record Number, Subfile Record Number, Associated Subfile Keywords.

54. What is SFLPAG and SFLSIZ?
    SFLPAG: it is an attribute which specifies the number of records that can be displayed in a screen.
    SFLSIZ: it is an attribute which specifies the number of records can be stored in subfile.

55. **Can more than one subfile record be displayed on one line?**
    Yes, by using SFLLIN keyword.

56. **How do you specify the number of records to roll in a subfile?**
    Use SFLROLVAL keyword in DDS along with number, which specifies the number of records to scroll at a time.

57. **How will you display a particular page in subfile?**
    Move a valid relative record number (RRN) in the field specified using SFLRCDNBR keyword in DDS.

58. **How to pick up the changed records every time in a subfile after the first change made?**
    Seton SFLNXTCHG keyword indicator and update the subfile record.

59. **What is the use of SFLEND keyword?**
    By specifying this keyword, the Bottom/More message could be displayed at end of screen.

60. **How to toggle between single line and Multi - line display of a particular record in a subfile?**
    Using SFLDROP keyword.

61. **Explain the difference between defining Subfile and Message-subfile?**
    Subfile record is defined by SFL keyword, where as Message subfile is defined by SFLMSG keyword.

62. **What are the different types of variables available in CL?**
    DEC, CHAR, LGL

63. **How do you pass parameters in CL?**
    Using PARM keyword.

64. **What is difference between CAT, TCAT, BCAT?**
CAT Concatenate two variables or constants into one continuous string.
BCAT Truncates all trailing blanks in the first character string, one blank is inserted, then the two character strings
are concatenated.
TCAT Truncates all trailing blanks in the first character string, the two character strings
are concatenated.

65. **What are the different types of messages in CL?**
Immediate message, Break message, Program message, User message

66. **How to trap errors in CL?**
By using Monitor Message Command (MONMSG)

67. **What is the maximum length of a variable name in CL?**
Maximum 11 characters (including '&')

68. **What are the limitations of CL (compare to RPG) ?**
You cannot use CL program to ADD or UPDATE records in database files.
Use Printer or ICF files.
Use Program described files.
Use the concept of subfile (to display more than one record), but a single output message subfile is a special type of
subfile that is supported well in CL.
Use subroutines.
You cannot declare more than one object (file) in a CL program.

69. **What is the use of Header Specification in RPG/400?**
It identifies by H in column 6, provides information about generating and running programs.

70. **When will DUMP and DEBUG opcodes be ignored?**
If blank is specified in position 15 of H specs.

71. **Specify different indicators used in RPG?**
Overflow indicators
Record Identifying Indicators
Field Indicators
Resulting Indicators
Control Level Indicators

72. **What are Control level indicators?**
L1 to L9 used to identify certain fields on control fields and then used to condition which operations are to be processed at detail or total calculation or output time.

73. **What is the use of E specification in RPG?**
    Extension Specs describes all record address files, arrays and tables.

74. **What is the use of L specs in RPG?**
    Line counter specification can be used to describe printer file to indicate the length of the form and number of lines per page.

75. **In which specification the report layout can be defined?**
    O Specification.

76. **How many files can be defined in F specs?** 50

77. **How many printer files can be defined in F specs?** 8

78. **Give three main purposes of File specification?**
    To define files, to describe the files, to assign the files to specified devices.

79. **How do you specify page overflow indicator for printer files in RPG?**
    Specify an indicator in position 33-34 of F specification.

80. **What is a Primary File?**
    It is used in RPG Program Cycle to automatically read records in a cycle.

81. **Can an indexed file be accessed in arrival sequence in RPG program?** Yes.

82. **What is a Program Described file in RPG?**
    The field name and length of the fields are defined within the RPG program.

83. **What is externally described file?**
    All information about the fields is specified in DDS and the RPG program can use them within the program.

84. **Can you specify a display file to be used in the following modes Input, Output, or Combined modes?** Yes.

85. **What is match field indicator?**
    Matching record indicator is seton when all the matching fields in the record of a secondary file matches with all the matching fields of a record in a primary file.

86. **What are all the compiler directive statements?**
    /TITLE, /SPACE, /EJECT, /COPY

87. **During execution, an RPG/400 program automatically follows a sequence of operations for each record that is processed. The built-in program cycle includes the following logical steps.**
    1. reading input (READ)

2. processing calculations (PROCESS)
3. writing output (WRITE)

88. **What are the different Opcodes available in RPG for Database access?**
READ, CHAIN, WRITE, UPDAT, DELET, SETLL, SETGT, READE, READP, REDPE, OPEN, CLOSE, FORCE, NEXT, UNLCK.
309. How can database records be read without lock ?
Put 'N' in position 53 of C specs.

89. **What does CHECK opcode do ?**
The check operation verifies that each character in the base string (factor 2) is among the character indicated in the comparator string(factor 1).

90. **How do handle file exception/error**
*INFDS ,*PSSR defining it in F spec

91. **What is OPNQRYF, MONMSG commands**
It is Dynamically creation of access path, and it can have resultant fields i.e. if the expression is A = B + C then B and C are from the file while A is defined in OPNQRYF. We can divert the output of command to an OUTFILE. Command associated with OPNQRYF is CPYFRMQRYF to save the output permanently

It is a CL command to monitor and error/exception message so that in case of an error a dump is avoided and the control is in program. It is also used to monitor user message.

92. **What are the uses of FACTOR1, FACTOR2 and RESULT field for the RPG operation code PARM?**

It is add value of FACTOR1 to FACTOR2 or compare the value of FACTOR 2 with FACTOR1.

93. **How will you find a string using PDM?**

By using FNDSTRPDM.

94. **How do you read changed records backward in subfile?**

NOT POSSIBLE.

95. **What is the difference between normal UPDDTA to PF and updating using DFU program?**

Both are same only difference is DFU allows you to add or change selected fields

133. **What is the syntax for PLIST?**

*ENTRY PLIST PARM

134. **Which are the String Manipulation Opcodes?**

TESTN, SCAN, CHECK, CHECKR, SUBST & CAT

Sub Procedures:

1. **Why Sub procedures are used?**

- Sub procedures can define their own local variables.
- Sub procedures can accept parameters.
- Sub procedures provide parameter checking
- Sub procedures can be used as a user defined function.
- You can call the sub procedure from outside the module, if it is exported.
- Sub procedures provide multiple entry points within the same RPG module.

1. **Can you use a sub procedure in a sub procedure?**

Yes

2. **What are the specifications used in a sub procedure?**

P, D, C only.

3. **How many ways a sub procedure can pass parameters?**

• By reference (default)

• By value (keyword VALUE on the parameter definition)

• By read-only reference (keyword CONST on the parameter definition)

4. How do you invoke a stored procedure?

To invoke a stored procedure we use the SQL CALL statement. This statement contains the name of the stored procedure and any arguments passed to it. Arguments may be constants, special registers, or host variables. Here is an example how to call stored procedure and pass two arguments:

```
/Exec Sql

CALL mylib/procname (:PARTNUM, :PARTDES)

/End-Exec
```

Easiest way to return a completion status to the SQL program issuing the CALL statement is to code an extra INOUT type parameter and set it prior to returning from the procedure. Another, more complicated method of returning a completion status is to send an escape message to the calling program (operating system program QSQCALL) which invokes the procedure.

5. Is there any cycle code generated for the sub procedure?

No. Not generated.

6. What are the Important frequently used commands in ILERPG environment?

Commonly Used CL Commands

Action	**CL command**	**Result**
Using System Menus	**GO MAIN**	Display main menu
	GO INFO	Display help menu
	GO CMDRPG	List commands for RPG
	GO CMDCRT	List commands for creating
	GO CMDxxx	List commands for 'xxx'
Calling program	**CALL**	program-name runs a
Compiling	**CRTxxxMOD**	Creates xxx Module
	CRTBNDxxx	Creates Bound xxx Program
Binding Modules	**CRTPGM**	Creates a program from ILE
	CRTSRVPGM	Creates a service program
	UPDPGM	Updates a bound program object
Debugging	**STRDBG**	Starts ILE source debugger

	ENDDBG	Ends ILE source debugger
Creating Files	**CRTPRTF**	Creates Print File
	CRTPF	Creates Physical File
	CRTSRCPF	Creates Source Physical File
	CRTLF	Creates Logical File

7. What are CODE/400 / Visual Age??

ADTS CS (for Windows Ñ) is a workstation product that includes two server access programs:

* Cooperative Development Environment/400 (CODE/400)

* Visual Age for RPGÑ

CODE/400 contains features to help edit, compile, and debug: RPG, ILE RPG,

COBOL, ILE COBOL, Control Language (CL), ILE C, and ILE CL host source programs; design display, printer, and database host files; and manage the components that make up your application. This enhances program development and moves the program development workload off the host. The application, when built, runs on an AS/400. For RPG and ILE RPG application development and maintenance, CODE/400 provides:

* Language sensitive editing— includes token highlighting, format lines, a full suite of prompts, and online help.

* Incremental syntax checking— provides immediate error feedback as each line of source is entered

* Program verification— performs, at the workstation, the full range of syntax and semantic checking that the compiler does, without generating object code

* Program conversion— performs, at the workstation, an OPM to ILE RPG conversion

* A windowed environment for submitting host compiles and binds

* Source-level debugging

* A DDS design utility—allows you to easily change screens, reports, and database files

* Access to Application Dictionary Services.

Visual Age for RPG offers a visual development environment on the workstation platform for RPG application developers to develop, maintain, and document client/server applications. Applications can be edited, compiled, and debugged on your workstation. The applications, when built, are started on a workstation and can access AS/400 host data and other AS/400 objects. Its integrated components allow application developers to preserve their current skills and easily develop AS/400 RPG applications with graphical user interfaces.

8. What are Main Procedure and a sub procedure?

An ILE RPG module consists of a main procedure and zero or more sub procedures. (If there are sub procedures, the main procedure is optional.) **A main procedur**e is a procedure that can be specified as the program entry procedure (and so receive control when an ILE program is first called). The main procedure is defined in the **main source sectio**n, which is the set of H, F, D, I, C, and O specifications that begin a module. In V3R1, all ILE RPG modules had a main procedure and no other procedures.

A **sub procedu**re is a procedure that is specified after the main source section. A sub procedure differs from a main procedure primarily in that:

* Names that are defined within sub procedure are not accessible outside the sub procedure.

* No cycle code is generated for the sub procedure.

* The call interface must be prototyped.

* Calls to sub procedures must be bound procedure calls.

* Only P, D, and C specifications can be used.

Sub procedures can provide independence from other procedures because the data items are local. Local data items are normally stored in automatic storage, which means that the value of a local variable is not preserved between calls to the procedure.

Sub procedures offer another feature. You can pass parameters to a sub procedure by value, and you can call a sub procedure in an expression to return a value.

Sub Files

1. **Explain about sub files in AS/400?**
   - ✔ A subfile is a group of records READ from or WRITTEN to a display device file in one single operation.
   - ✔ It is a display file facility
   - ✔ It is a group of records that can be stored in the main memory.
   - ✔ The program can store a group of records in the subfile one by one in a sequence.
   - ✔ **LODING SUBFILES:**
     - ➢ Load all (Size >Page)
     - ➢ Load on demand (Size >Page)
     - ➢ Load on demand (Size =Page)
   - ✔ **Load all (Size >Page)**
     - ➢ All the records from the database file will be loaded in to the subfile in one shot.
     - ➢ The subfile size should be greater than the page size at least by one. (SFLSIZ =5, SFLPAG=4). The subfile size will dynamically grow when the subfile size mentioned is less than the number of records in the database file.
     - ➢ PAGEUP and PAGEDOWN are taken care of by the system.
     - ➢ The total subfile size 9999 records.
   - ✔ **Load on demand (Size > Page)**
     - ➢ The number of records as mentioned in SFLPAG will be loaded initially.
     - ➢ Then the remaining records can be loaded by pressing **PAGEDOWN**, which is taken care of by the programmer.
     - ➢ Same time **PAGEUP** is taken care of by the system.
     - ➢ The subfile size should be greater than the page size at least by one. (SFLSIZ =5, SFLPAG=4). The subfile size will dynamically grow when the subfile size mentioned is less than the number of records in the database file.
     - ➢ All the records loaded will exist in the subfile.
     - ➢ The total subfile size 9999 records.

- ✔ **Load on demand (Size = Page)**
  - ➢ The number of records that will be loaded into the subfile must always be equal to the value mentioned in for SFLSIZ and SFLPAG.
  - ➢ Every time the subfile should be cleared before paging up or paging down.
  - ➢ **PAGEUP** and **PAGEDOWN** are taken care of the programmer.
  - ➢ The number of records that can be in the subfile at any instance will be equal to SFLSIZ and SFLPAG values.

Subfile points:

Record formats:

One display -1024 records formats

One display files –512 subfiles

Record formats are

1. Subfile record format (SFL)
2. Subfile control record format (SFLCTL)

Subfile record format (SFL)

This record format will have the multiple record definitions

-Defining fields.

-Defining database fields.

Subfile Control record format (SFLCTL)

This record format will control the subfile record format.

-Defining texts

-Defining control fields.

Subfile Size (SFLSIZ)

This keyword can be used to specify the maximum number of records that can be in the subfile (buffer)

Default ->2

Maximum ->9999

Subfile Page (SFLPAG)

This keyword can be used to specify the maximum number of records that can be in one subfile page. That is the maximum number of records that the system will display in the screen at a time.

-Default ->1

-Maximum-> depends upon the display record size.

If the subfile size is at least one greater than the subfile page then the subfile size will grow dynamically up to 9999.

General keywords

SFLDSP	-> subfile display
SFLDSPCTL	-> subfile display control
SFLCLR	-> subfile clear
SFLEND	-> subfile end

Define General Keywords

```
Subfile control record . . . . . . . . . :   SENWLT1

Type choices, press Enter.               Keyword

  Related subfile record . . . . . . .   SFLCTL     SENWND1      Name

  Subfile cursor relative record . . .   SFLCSRRRN              Name

  Subfile mode . . . . . . . . . . . .  SFLMODE               Name

                                          Y=Yes       Indicators/+

  Display subfile records  . . . . . .   SFLDSP      Y          25

  Display control record . . . . . . .   SFLDSPCTL    Y          26

  Initialize subfile fields  . . . . .  SFLINZ

  Delete subfile area  . . . . . . . .  SFLDLT

  Clear subfile records  . . . . . . .  SFLCLR                 28

  Indicate more records  . . . . . . .  SFLEND                 30

    SFLEND parameter   . . . . . . . .    *MORE       Y

    SFLEND parameter   . . . . . . . .    *SCRBAR               *MORE...

  Record not active  . . . . . . . . .  SFLRNA

                                                    More...

F3=Exit   F12=Cancel
```

Subfile Display (SFLDSP)

This keyword is used to insert the system that the subfile records format has to be displayed. The subfile record format without any record in it cannot be displayed.

Subfile Display control (SFLDSPCTL)

This keyword is used to instruct the system that the subfile control record format has to be displayed.

```
0009.00 C N30          SETON                              2526
```

Subfile Clear (SFLCLR)

This keyword is used to clear the records in the subfile records format.

An indicator can control this keyword.

```
0005.00 C              SETON                              28
0006.00 C              WRITE    SENWLT1
0007.00 C              SETOFF                             28
```

Subfile End (SFLEND)

- ✔ This keyword is used to get the display of '+' sign or a text 'more' or 'bottom' in the bottom of the subfile.
- ✔ '+ or 'more' indicates the existence of more records in the subfile which can be displayed by pressing **PAGEDOWN** key.
- ✔ **'B**ottom' indicates the end of the subfile.
- ✔ Instead of Enter Key - Help Type a CF or CA key number to specify that the operator is to use the Enter key as a Roll Up key. The specified CF or CA key acts as the Enter key.

Mandatory keywords for subfile

- ✔ SFL
- ✔ SFLCTL
- ✔ SFLSIZ
- ✔ SFLPAG
- ✔ SFLDSP

RRN (Relative record number) (control record level keyword)

- ✔ RRN is the numeric value (1 to 9999) associated with each subfile record for accessing
- ✔ Each record should have a unique RRN value
- ✔ This value has to be giving by the program.
- ✔ This has to be associated with the subfile record format in the F spec continuation line.

- ✔ This has to be declared in the program as numeric variable of maximum length 4 and decimal position 0.
  F spec format in continuation line with SFILE option

```
0002.00 FSENDESFILECF   E            WORKSTN
0003.00 F                                  SFILE (SFL01:RRN1 )
```

Where RRN1 -> RRN variable name

SFL01-> subfile record format name.

RPGLE Opcodes

EXFMT

If SFLDSP and SFLDSPCTL indicators are on, this opcode will send the two subfile record formats to the display device and waits for the user's response.

WRITE – (SFL)

- ➢ This opcode is used to add a record to the subfile record format
- ➢ RRN value should be set with a non-existing value before adding the record in the subfile record format.

WRITE (SFLCTL)

This keyword is used for clearing & display the subfile

SFLDROP (subfile drop)

This control record level keyword is used to assign a CA (command attention) or CF (command function) key. The program first displays the subfile in truncated form; subfile records are truncated to fit on one display line. When the user presses the specified key, the program displays the records in the folded form.

Or

Subfile Initially Truncated - Help Type a command function (CF) or command attention (CA) key number to assign a CF or CA key to specify if a subfile control record requiring more than one display line should be truncated to one line, or should be folded to display on two lines. When this keyword is specified, the subfile is first displayed in truncated form. The operator presses the specified CF or CA key to switch from truncated form to folded form or from folded form to truncated form.

SFLFOLD (subfile fold)

This control record level keyword is used to assign a CA (command attention) or CF (command function) key. The program first displays the subfile in folded form. When the user presses the specified key, the program displays the records again in the truncated form.

Or

Subfile Initially Folded - Help Type a command function (CF) or command attention (CA) key number to assign a CF or CA key to specify if a subfile control record requiring more than one display line should be truncated to one line, or should be folded to display on two lines. When this keyword is specified, the subfile is first displayed in folded form. The operator presses the specified CF or CA key to switch from folded form to truncated form or from truncated form to folded form.

SFLINZ (subfile initialize)

This control record level keyword is used to specify that the program is to initialize all records in the subfile on the output operation (read & updates) to the subfile control record format.

The fields in each subfile record are initialized to

CHAR->BLANKS

NUME->ZEROS

FLOAT->NULLS

SFLNXTCHG (subfile next change)

This record level keyword is used on the subfile control record format to force the user to correct program-detected keying error in the subfile records that have been read by the program. It does this by causing the record to be changed so that a get-next-changed operation must read the record.

Or

Return Record - Help Type Y (Yes) to instruct the system to return this subfile record to your program on a subsequent Get-Next-Changed input operation to the subfile. This record is returned whether or not the operator changes it. Note: You must type Y. If you do not, the entry will not be used. `You can specify condition indicators for the SFLNXTCHG keyword.

SFLROLVAL (subfile roll value)

This field-level keyword is used to specify that the user can key a value into this field to tell the program how many records to PAGEUP or PAGEDOWN when the appropriate paging key is pressed.

SFLRCDNBR (subfile record number)

This field level keyword on the subfile control record format is used to specify that the page of the subfile to be displayed is the page containing the record whose relative record is in this field. If you do not specify this keyword, the program displays the first page of the subfile by default

Example for **load all**

```
                                          SKANDASAMO/SUBFILE

                                                     NEWEMP

                 *************** Beginning of data *******************

0001.00                                UNIQUE

0002.00           R EMPNEW

0003.00             EMPNOM       6P 0      TEXT('EMPLOYEE NUMBER')

0004.00             EMPNA        15A       TEXT('EMPLOYEE NAME')

0005.00             SEX3         1A        TEXT('EMPLOYEE SEX')

0006.00             AGE3         3P 0      TEXT('EMPLOYEE AGE')

0007.00             ADDRESS5     15A       TEXT('ADDRESS')

0008.00             CITY5        10A       TEXT('CITY')

0009.00           K EMPNOM

      ****************** End of data ********************************
```

DATA FILE

Display Report

```
      EMPNOM   EMPNA        SEX3  AGE3   ADDRESS5      CITY5
```

```
000001     101   K.SENTHILKUMAR    M      25   ATTUR            SALEM
000002     102   R.SHYAMSUNDAR     M      27   TRICHY           TRICHY
000003     103   B.MOHAN           M      27   TCODE            SALEM
000004     104   K.KUMAR           M      25   SALEM            SALEM
000005     105   A.ARUL            M      25   SALEM            NAMAKKAL
000006     106   BALU              M      25   SALEM            SALEM
000007     107   SENTHIL           M      35   SALEM            TRICHY
000008     108   RAJ               M      22   TCODWE           SALEM
000009     109   HEMA              M      33   SALEM            CHENNAI
000010     110   VEL               M      67   SALEM            SALEM
000011     111   RAMESH            M      56   ATTUR            SALEM
000012     122   SUDHA             M      28   SALEM            ATTUR
000013     123   KANDASAMY         M      34   SALEM            ATTUR
****** ******** End of report  ********
                                                  SKANDASAMO/SUBFILE
                                                                   SUB04
      *************** Beginning of data ****************************
0002.00 FSENDESFILECF   E             WORKSTN
0015.00 C     *IN03         DOWEQ     *OFF
0016.00 C                   EXFMT     MAIN1
0016.01 C                   IF        *IN04=*ON AND EMPCUR='EMPNOM'
0016.02 C                   CALL      'SELOADALL3'
0016.03 C                   ENDIF
0017.00 C   03                LEAVE
0018.00 C                   ENDDO
```

```
C                  SETON                                   LR
       ****************** End of data ********************************
                                                SKANDASAMO/SUBFILE
                                                           SELOADALL3
       *************** Beginning of data *****************************
FNEWEMP    IF   E            DISK
FSENDESFILECF   E            WORKSTN
F                               SFILE(SENWND1:RRN1)
C                  Z-ADD     1            RRN1             4 0
C                  SETON                                   28
C                  WRITE     SENWLT1
C                  SETOFF                                  28
C                  READ      EMPNEW                                30
C  N30             SETON                                   2526
C                  DOW       *IN30=*OFF
C                  WRITE     SENWND1
C                  ADD       1            RRN1
C                  READ      EMPNEW                                30
C   30             LEAVE
C                  ENDDO
C                  DOW       *IN03=*OFF
C   03             LEAVE
C*                 WRITE     HEATER
C*                 WRITE     FOOTER1
C                  EXFMT     SENWLT1
```

```
0020.00 C            ENDDO
0021.00 C            SETON                          LR

        ****************** End of data *****************************
```

window main1 take 8

```
                 Select Record Keywords
Record . . . :   MAIN1
Type choices, press Enter.
                            Y=Yes
  General keywords  . . . . . . . .   Y
  Indicator keywords  . . . . . . .   Y
  Application help  . . . . . . . .


                 Select General Keywords
Record . . . :   MAIN1
Type choices, press Enter.                      Keyword    Y=Yes
  If this record is not on display, write it
    to the display before issuing read . . . . . . .   INZRCD
  Keep record on display . . . . . . . . . . . . . .   KEEP
  Assume record is on display  . . . . . . . . . . .   ASSUME       Y
  Allow rolling of lines . . . . . . . . . . . . . .   ALWROL
  Retain CLEAR HELP HOME and ROLL keys . . . . . . .   RETKEY
  Retain command function (CFnn and CAnn) keys . . .   RETCMDKEY
  Change input defaults  . . . . . . . . . . . . . .   CHGINPDFT
    Select parameters  . . . . . . . . . . . . . . .
  Menu-Bar display . . . . . . . . . . . . . . . . .   MNUBARDSP
    Select parameters  . . . . . . . . . . . . . . .
  Entry field attribute  . . . . . . . . . . . . . .  ENTFLDATR
    Select parameters  . . . . . . . . . . . . . . .
  Return cursor location . . . . . . . . . . . . . .  RTNCSRLOC    Y
    Select parameters  . . . . . . . . . . . . . . .
                       Define Return Cursor Location
 Record . . . :   MAIN1
                                    Keyword number     Roll
                                      1   of 1         Y     +/-
                                                           F4 for list
 Type parameters, press Enter.
                                          Keyword
 Return cursor location . . . . . . . . . . RTNCSRLOC   Y              Y=Yes
   Type indicator *RECNAME  . . . . . . . .              Y              Y=Yes
   Cursor record  . . . . . . . . . . . . .             RECSD          Name
   Cursor field . . . . . . . . . . . . . .            EMPCUR         Name
   Cursor position  . . . . . . . . . . . .                           Name
CF03                    03
 CF04                    04
```

TAKE 12 AND F4

```
                             Work with Fields
 Record . . . :   MAIN1
 Type information, press Enter.
```

```
  Number of fields to roll . . . . . . . . . . . . . . . . . .     6
 Type options, change values, press Enter.
   1=Select keywords   4=Delete field
 Option   Order   Field       Type Use Length Row/Col Ref Condition Overlap
             70   AGE           C          21   15 014
             80    RECSD          A   H       10
             90   EMPCUR        A   H       10
            100   ----------    C          76   21 003
            110   F3->EXIT      C          37   22 005
            120   EMPNOM        S   B      6,0   09 036   Y
                                                          More...
  Add                           H           Hidden
  Add                           M           Message
  Add                           P           Program-to-system
```

TAKE 12

SENWLT1

```
                         Select Window Keywords
 Window record . . . . . . . . . . . . . :    SENWLT1
 Type choices, press Enter.
                                   Y=Yes
   General keywords  . . . . . . . .   Y
   Select record keywords  . . . . .   Y
   General SFLCTL keywords . . . . .   Y
   Subfile display layout  . . . . .   Y
```

```
  Subfile messages  . . . . . . . .   Y
```

Select General Keywords

```
Window record . . . . . . . . . . . . . :   SENWLT1

Type choices, press Enter.

                                              Keyword    Y=Yes  Indicators/+

  Window parameters  . . . . . . . . . . . .   WINDOW       Y

    Select parameters  . . . . . . . . . . .                Y

  Window borders . . . . . . . . . . . . . .   WDWBORDER

    Select parameters  . . . . . . . . . . .

  Remove windows . . . . . . . . . . . . . .   RMVWDW

  User Restore Display . . . . . . . . . . .   USRRSTDSP
```

Define Window Parameters

```
Record . . . :    SENWLT1

Keyword  . . :    WINDOW

Referenced window  . . . . . . . . . . . .                    Name

-OR-

Window definition

  Default start positioning  . . . . . . .                    Y=Yes

  -OR-

  Start line

    Program-to-system field  . . . . . . .                    Name

    Actual line  . . . . . . . . . . . .      2           1-25

  Start position

    Program-to-system field  . . . . . . .                    Name

    Actual position  . . . . . . . . . . .      2          1-128
  Window lines . . . . . . . . . . . . . .     20            1-25
  Window position  . . . . . . . . . . . .      40           1-128
  Message line . . . . . . . . . . . . . .     Y             Y=Yes
  Restrict cursor to window  . . . . . . .      Y             Y=Yes
```

Select Record Keywords

```
Record . . . :   SENWLT1
Type choices, press Enter.
                              Y=Yes
  General keywords  . . . . . . . .   Y
  Indicator keywords  . . . . . . .   Y
  Application help  . . . . . . . .
  Help keywords . . . . . . . . . .
  Output keywords . . . . . . . . . .


CF12                12
 CF03                03
```

Define General Keywords

```
Subfile control record . . . . . . . . . :   SENWLT1
Type choices, press Enter.              Keyword
  Related subfile record . . . . . . .   SFLCTL     SENWND1      Name
  Subfile cursor relative record . . .   SFLCSRRRN              Name
  Subfile mode . . . . . . . . . . . .   SFLMODE                Name
                                          Y=Yes       Indicators/+
  Display subfile records  . . . . . .   SFLDSP       Y          25
  Display control record . . . . . . .   SFLDSPCTL    Y          26
```

```
   Initialize subfile fields  . . . . .   SFLINZ
   Delete subfile area  . . . . . . . .   SFLDLT
   Clear subfile records  . . . . . . .   SFLCLR                **28**
   Indicate more records  . . . . . . .   SFLEND                  **30**
     SFLEND parameter   . . . . . . . .     *MORE       **Y**
     SFLEND parameter   . . . . . . . .     *SCRBAR                 *MORE ...
   Record not active  . . . . . . . . .   SFLRNA
                                                  More...
 F3=Exit   F12=Cancel
                         Define Display Layout
Subfile control record . . . . . . . . . :   SENWLT1
Type values, press Enter.
                                          Keyword  Number
  Records in subfile . . . . . . . . . .   SFLSIZ      **6**
    Program-to-system field  . . . . . .
  Records per display  . . . . . . . . .   SFLPAG      **5**
  Spaces between records . . . . . . . .   SFLLIN
```

OUTPUT

```
              EMPLOYEE DETAILS
     1.SELECT
     OPT      EMPNUMBER        NAME
              000101       K.SENTHILKUMAR
              000102       R.SHYAMSUNDAR
              000103       B.MOHAN
              000104       K.KUMAR
```

```
          000105        A.ARUL
                           More...


           EMPLOYEE DETAILS:
   1.SELECT


   OPT      EMPNUMBER         NAME


         000106        BALU
         000107        SENTHIL
         000108        RAJ
         000109        HEMA
         000110        VEL
                          More...


            EMPLOYEE DETAILS
    1.SELECT
    OPT      EMPNUMBER         NAME
          000111        RAMESH
          000122        SUDHA
      123      KANDASAMY
```

2. **Message subfile record format keywords**

❖ SFLMSGRCD (subfile message record)
This keyword is used to give the line number to get the first message in the display.

❖ SFLMSGKEY (subfile message key)

This keyword is used to specify that the program message queue is built one at a time.

- SFLPGMQ (subfile program queue)

  It is a message queue created for every program active in the call stack.

  This keyword is used to specify the field that can have the name of the program message queue.

1. **How to create Message subfile?**
   - Create message subfile

CRTMSGF MKSFILE/LIB

WRKMSGF FILE/ANME

OPTION 12

```
                     Add Message Description (ADDMSGD)

 Type choices, press Enter.

 Message identifier . . . . . . . MKS0001              Name

 Message file . . . . . . . . . . > MKSFILE      Name

   Library  . . . . . . . . . . . >   SKANDASAMO  Name, *LIBL, *CURLIB

 First-level message text . . . . EMPLOYEE NUMBER &1 CANNOT BE ZEROS

PAGE DOWN

Message data fields formats:

   Data type  . . . . . . . . . .   *CHAR        *NONE, *QTDCHAR, *CHAR...

   Length . . . . . . . . . . . .   10           Number, *VARY
```

WRKMSGF

```
MKS0001      0    EMPLOYEE NUMBER CANNOT BE ZEROS

MKS0002      0    EMPLOYEE &1 NUMBER AIREADY EXIST

MKS0003      0    EMPLOYEE NAME CANNOT BE BLANKS

MKS0004      0    ADDRESS(1) CANNOT BE BLANKS
```

MKS0005	0	CITY CANNOT BE BLANKS
MKS0006	0	MARITAL STATUS WILL BE WITH (M/S)
MKS0007	0	EMPLOYEE NUMBER &1 ADDED SUCCESFULLY
MKS0008	0	EMPLOYEE NUMBER DOES NOT EXISTS
MKS0009	0	EMPLOYEE UPDATED SUCCESSFULLY
MKS0010	0	EMPLOYEE NUMBERDELETE SUCCESFULLY

- **Create message subfile**

```
10   MSGREC1     SFLMSG                  06/27/02

20   MSGCTL1     SFLCTL   MSGREC1          06/27/02
```

USING 8 SELETION

```
                    Select Subfile Keywords

Subfile record . . . . . . . :   MSGREC1

Type choices, press Enter

                          Y=Yes

  General keywords  . . . . . . . .   Y

  Indicator keywords  . . . . . . .   Y

  Message record  . . . . . . . . .   Y

  TEXT keyword  . . . . . . . . . .
```

Define Message Record

```
Subfile record . . . . . . . :   MSGREC1

Type choices, press Enter.

                              Keyword

  Line number for first message . . . .   SFLMSGRCD   24         1-27

  Message ID field  . . . . . . . . . .   SFLMSGKEY  DUMMY       Name

    (if program message queue is built

     one message at a time)
  Program message queue field . . . . .  SFLPGMQ    QUEUE      Name
    Generate a 276 byte field . . . . .                Y=Yes
```

MESSAGE SUBFILE (SFTCTL)

```
Subfile control record . . . . . . . . . :   MSGCTL1
Type choices, press Enter.
                                   Y=Yes
  General keywords  . . . . . . . .   Y
  Subfile display layout  . . . . .   Y
  Subfile messages  . . . . . . . .   Y
  Select record keywords  . . . . .   Y
  TEXT keyword  . . . . . . . . . .
```

Define General Keywords

```
Subfile control record . . . . . . . . . :   MSGCTL1
Type choices, press Enter.               Keyword
  Related subfile record . . . . . . .   SFLCTL     MSGREC1     Name
  Subfile cursor relative record . . .   SFLCSRRRN              Name
  Subfile mode . . . . . . . . . . . .   SFLMODE                Name
  Program message queue field  . . . .   SFLPGMQ    QUEUE       Name
    Generate a 276 byte field  . . . .              Y=Yes
                                         Y=Yes      Indicators/+
  Display subfile records  . . . . . .   SFLDSP     Y
  Display control record . . . . . . .   SFLDSPCTL  Y
  Initialize subfile fields  . . . . .   SFLINZ     Y
  Delete subfile area  . . . . . . . .   SFLDLT
  Clear subfile records  . . . . . . .   SFLCLR
  Indicate more records  . . . . . . .   SFLEND
    SFLEND parameter   . . . . . . . .     *MORE
    SFLEND parameter   . . . . . . . .     *SCRBAR                 *MORE ...
  Record not active  . . . . . . . . .   SFLRNA
                                                               More...
F3=Exit   F12=Cancel
               Select Record Keywords
Record . . . :   MSGCTL1
Type choices, press Enter.
                                   Y=Yes
  General keywords  . . . . . . . .   Y
  Indicator keywords  . . . . . . .   Y
Overlay keywords  . . . . . . . .   Y


               Select Overlay Keywords
Record . . . :   MSGCTL1
Type choices, press Enter.
                                        Keyword    Y=Yes  Indicators/+   Roll
  Overlay without erasing  . . . . . .   OVERLAY      Y
```

- ❖ Create CL program for SNDPGMMSG

```
                                                    SKANDASAMO/EMPCUSTOR
                                                                  MSG
       *************** Beginning of data *****************************
0001.00            PGM        PARM(&MSGID &MSGFI &MSGDTA)
0002.00            DCL       VAR(&MSGID) TYPE(*CHAR) LEN(7)
0003.00            DCL       VAR(&MSGFI) TYPE(*CHAR) LEN(7)
```

```
0004.00            DCL       VAR(&MSGDTA) TYPE(*CHAR) LEN(10)

0005.00                      SNDPGMMSG    MSGID(&MSGID)  MSGF(&MSGFI)
MSGDTA(&MSGDTA)

0006.00            ENDPGM

      ****************** End of data******************************
```

- **Create CL program for RMVMSG**

```
                                    SKANDASAMO/EMPCUSTOR

                                                    RMO

      *************** Beginning of data *****************************

0001.00            RMVMSG    PGMQ(*PRV) CLEAR(*ALL)

      ****************** End of data *******************************
```

- Design the screen
- Generate RPG program

```
0000.01 FMESSTAB   IF A E          K DISK

0001.00 FMKSSCR    CF   E            WORKSTN

0001.01 C                  MOVEL     MSTATUS       S                1

0002.02 C                  MOVEL     '*'           QUEUE

0003.00 C                  MOVEL     'MKSFILE'     MSGFI

0004.00 C     PL01         PLIST

0005.00 C                  PARM                    MSGID            7

0006.00 C                  PARM                    MSGFI           10

0007.00 C                  PARM                    MSGDTA          10

0008.00 C                  DOW       *IN03=*OFF

0008.01 C                  MOVEL     0             ERROR            1 0

0008.02 C                  MOVEL     0             ERROREMP         1 0

0012.00 C                  WRITE     MSGCTL1

0013.00 C                  EXFMT     MKSSEN
```

```
0013.01 C            CALL     'RMO'
0014.00 C            IF       EMPNUMB=*ZEROS
0014.01 C            MOVEL    1           ERROR
0014.02 C            MOVEL    1           ERROREMP
0015.00 C            MOVEL    'MKS0001'   MSGID
0016.00 C            MOVEL    *ZEROS      MSGDTA
0017.00 C            CALL     'MSG'       PL01
0017.01 C            ENDIF
```

1. **What is active subfile?**
   - ❖ Subfiles, which are in the main memory, are called as active sub files.
   - ❖ A maximum of 12 sub file can be active at a time.

CL Programming

1. **CL commands?**

1. WRKMBRPDM

2. **CRTSRCPF**

By using this command to create source physical file. Default PF size is 92.

3. **DSPFD**

It is used to display the details about the file when it is created.

4. DSPFFD

It is used for listing details about Individual Fields.

5. STRDFU

It is used to add Data into the records.

6. STRSDA

It is used to go into screen Designing or Menu Designing.

7. CRTMSGF

For creating the Message file

8. WRKMSGF

If you want to create or change or delete any message we can use this command.

9. CRTDTAARA

For creating The Data Area

10. DSPDTAARA

Display the contents of Data area.

11. CHGDATARA

Change the contents of a Data area.

12. DSPLIBL

For listing the contents of library

13. ADDLIBLE

Adds a library into current library

14. RMVLIBLE

Removes a library in current library list

15. CHGCURLIB

For changing the Current Library to a new library

16. EDTLIBL

It is used to the edit of the library file. (Change library file it is temporary delete the library file)

For Adding or removing library lists.

17. SNDBRKMSG

Used for sending message to all user.

18. CRTPF

It is used to create a PF. Using this command we can set the maximum number of records, whether delete or update operation is allowed or not, maximum storage allocation, waiting time etc can be determined

19. CRTLF

Creates the logical files

20. STRRLU

To go into the RLU we are using this command. Default length of RLU is 132 if you give it and creating it very first time else it set the page width value of last edited RLU. We can give the page width value from 1 to 378 in page width option.

21. DSPSBS

It is used to list the various subsystems running under AS/400 environment. Various subsystems running are QINTER, QACTIVE, and QBATCH etc.

22. CRTRPGPGM

It is used to create an RPG program. We can determine what sort of source file is generated. For example when we need the entire source compile as it is since source is default. If we give *NOSOURCE only syntax errors is generated NOSECLVL will not allows secondary message where as other setting is allowed it and so on

23. CRTCLPGM

To create CL program

24. EDTOBJAUT

It is used to give authority to a particular source PF. For giving authority to other user we must give authority to library, user profile and the source PF.

25. DSPOBJD

If we know library name and object name and we want to know the source PF where it is residing then DSPOBJD with option as services instant of basic will give the source PF name.

It describes various object descriptions like created Date, Created by, Source Physical file, which it is being created, and so on.

26. STRSQL

Starts SQL.

27. WRKSPLF

To work with Spool file

28. DLTSPLF

To delete the spool file

29. CRTDUPOBJ

This command creates duplicate object. If you want to compile a PF having 1000 of records and when we compile it all the data will be lost else if you want to add or delete a attribute data for other fields will have to copied. For that we a duplicate object.

30. CPYF

Records are being copied from PF to temporary file and after compiling it we have to again copy back from temporary file to the original file. If we Add a attribute we have to give *map and if we want to delete a attribute we have to give *drop in the map entry field.

31. DLTF

After copying into the original file we have to delete the temporary file or if you want to delete any file we are using this command. DLTF will only remove the object

32. RMVM

It is used to delete the member as well as the object.

33. RUNQRY

Displays all the records in a PF

34. CRTCMD

It is the powerful command used to create user define commands.

35. CMPPFM

It is used to compare two programs or files. It can be coded by taking option 54 in Subfile screen.

36. MRGSRC

It is used to merge a file with another file. We have to mention three files one is the root file, which is just a root and does not contain any code even. The second file is target file where we can have the ready-made we can copy the target source.

37. RTVCLSRC

If you delete a CL Source then we can retain the source if you are having the object by using this command.

38. SBMJOB

If you want to submit the job in certain interval i.e. on a specified date and time we can use this command.

39. DSPJOB

It will display all the jobs.

40. DSPUSRPRF

It will display all the entries regarding the particular user profile. It displays information like what is the user. Profile name; date previously, user class, printing and all.

41. CPYSRCF

If we want to copy all the members in a source PF to another source PF this command is used.

42. WRKACTJOB

To display the active jobs running in AS/400 systems.

43. DSPJOBLOG

By using this command display the output console.

44. SNDMSG

This command is to send the message to an user.

45. DSPMSG

This command is display all the message.

46. CHGCURLIB

Change the current library file.

47. ADDLIBLE

It is used to add the library. It is fully administrator authority.

48. RMVLIBLE

This command is to remove a library from the list.

49. CRTUSRPRF

It is used to display the rights given to a user. The system administrator can change authority he can give the authority as a system administrator.

50. DSPDBR (data base relation)

This is used to list all the files, which are related to a PF. It displays all the LF that is referring the PF and also lists the child table if it is having a relation through ADDPFCST.

51. DSPJOB

It will display all the jobs submitted within the specific interval and display the entire user who worked on the system at that time

52. WRKMSGQ

It will list all the messages of different user in the job queue.

53. CPYSRCF

If we want to copy all the members in a source PF to another source PF we can use this command.

54. CPYTODKT

If we want to copy from source PF to a diskette file

55. CPYTOTAP

If we want to copy source PF to a tape then we can use this command.

56. STRDBG

If we want to debug a ILE program then we can use this command .We have to create a ILE program by compiling with 15 which is CRTRPGMOD command and take F10 give debugging values as *Source. This will create a module. Then we have the create the program by giving program name and module name as the same and if we are calling any other modules also in that include that in the CRTPGM command

Ex: CRTPGM PGM (LIB/PNAME) MODULE (LIB/PNAME)

(LIB/SPNAME)

Now the program as well as the module is created. Then we have to start the debug by using the command.

STRDBG PGM (LIB/PNAME) UPDPROD (*YES)

It will show the source code of the program and we have to press F6 set the break point and press F10 key and call the program

CALL PNAME

F11-> display the variable

TYPE ()	**LEN ()**	**VALUE ()**
*DEC	Default (15 5) Max (15 9)	Default (0)
*CHAR	Default (32)	Default (b)
*LGL	1	Default (‘0’)

Shift + F11 -> go to module

2. **Data types in CL?**
   Char, Logical, Numerical

3. **String operation in CL?**

***CAT** ->Concatenate without editing.

***BCAT**->trailing blanks in the first character string is truncated. One blank is inserted, and then the two character strings are concatenated. Any leading blanks of the second operand are not truncated

***TCAT->**All trailing blanks in the first character string is truncated, and then the two character strings are concatenated. Any leading blanks of the second operand are not truncated.

```
                                   SKANDASAMO/CLP

                                       STRING

   *************** Beginning of data ********************************

0000.01 /*STRING *CAT *BCAT *TCAT FUNCTION */

0001.00            PGM       PARM(&STR &STR1 &STR2 &STR3 &STR4)

0001.02            DCL       VAR(&STR) TYPE(*CHAR) LEN(15)

0001.03            DCL       VAR(&STR1) TYPE(*CHAR) LEN(15)

0001.04            DCL       VAR(&STR2) TYPE(*CHAR) LEN(15)

0001.05            DCL       VAR(&STR3) TYPE(*CHAR) LEN(15)

0001.06            DCL       VAR(&STR4) TYPE(*CHAR) LEN(40)

0001.07            CHGVAR    VAR(&STR2) VALUE(&STR *CAT &STR1)

0001.08            CHGVAR    VAR(&STR3) VALUE(&STR *BCAT &STR1)

0001.09            CHGVAR    VAR(&STR4) VALUE(&STR *TCAT &STR1)

0001.10            SNDMSG    MSG(&STR3) TOUSR(SKANDASAMY)

0001.11            SNDMSG    MSG(&STR4) TOUSR(SKANDASAMY)

0001.12            SNDMSG    MSG(&STR2) TOUSR(SKANDASAMY)

0006.00 ENDPGM

   ****************** End of data **********************************
```

Run

call program name (**string)** f4

```
Program . . . . . . . . . . . . > STRING      Name
```

```
 Library  . . . . . . . . . . . >   SKANDASAMO  Name, *LIBL, *CURLIB

Parameters . . . . . . . . . . . > SENTHIL

                                > kumar

                                > ''

                                > ''

                    + for more values > ''
```

DSPMSG

4. **How to set the cursor position in particular field in particular position?**
   Using the Curpos
5. **How will retrieve the data in data area?**
   In –retrieve a data area

   Out-write a data area
6. **Built in function in CL?**

   - %SUBSTRING or %SST
     The sub string built-in function produces a character string that is a subset of an existing character string and can only be used with a CL program.

     %SUBSTRING (Character-variable-name Starting-position length)

     Or

     %SST (Character-variable-name Starting-position length)
   - %SWITCH

2. **Define indicator in CL?**

We can set on or set off the indicator by the command.

CHGVAR (&IN30) VALUE (‘0’) ->setoff

CHGVAR (&IN30) VALUE (‘1’)->set on

3. **Message subfile in CL**
Subfile cannot be used in CL but we can use message subfiles in CL.

4. **CL processing commands & program control commands?**

PROCESSING -> CHGVAR, SNDPGMMSG, OVRDBF, AND DLTF.

PROGRAM CONTROL ->CALL, RETURN, TFRCTL

5. **How to CL code has to change to use a call procedure?**

By using CALLPRC command. This is the bound call in CL that calls a procedure within a module.

6. **What are various steps accessing data area in CL?**
   - ✔ The first create a general data area use the command (CRTDTAARA)
   - ✔ To retrieve values from data area use (RTVDTAARA)
   - ✔ To change this value, use (CHGDTAARA)
   - ✔ To display the current value, use (DSPDTAARA)
   - ✔ To delete a data area use (DLTDTAARA)

2. **What is the equivalent command to *setll *loval* in CL?**
POSDBF with file position as *start will set the file to the beginning (or) using OVRDBF and specify the key field value by RRN value (or) by giving *start.

3. **Various types of message available in CL.**
Message is the interface between operating system and the programs or user and program. We can classify the message into two types namely

- ❖ Immediate message
- ❖ Predefined message

❖ **Immediate message**
Which does the program or system user create when they are sent and are not permanently stored in the system?

- ✔ Control language
  - ➢ SNDUSRMSG
  - ➢ SNDPGMMSG
  - ➢ SNDMSG
  - ➢ SNDBRKMSG
- ✔ Display files
  - ➢ ERRMSG
  - ➢ SFLMSG

- ✔ INQUIRY and INFORMATIONAL message:

Using SNDUSRMSG command to send type of message

- ❖ **Predefined message**

They are created before they are used. These messages are placed in a message file (queue) when they are created, and retrieved from the file when they are used.

- ✔ Control language
  - ➢ SNDUSRMSG
  - ➢ SNDPGMMSG
  - ➢ RTVMSG

- ✔ Display files
  - ➢ ERRMSGID
  - ➢ SFLMSGID
  - ➢ MSGCON
  - ➢ MSGID
- ✔ COMPLETION and DIAGNOSTIC message
  - ➢ Using SNDPGMMSG command these of message can be sent to any message queue.
  - ➢ DIAGNOSTIC message tell the calling program about errors detected by the program. Completion message tell the result of work done by the program.
- ✔ STATUS messages

Using SNDPGMMSG command status message can be sent to it's caller's program message queue or to the external message queue for the job. These messages tell the receiving program the status of the work performed by the sending program.

- ✔ ESCAPE message

Using SNDPGMMSG command escape message from a CL program can be sent to its calling program. An escape message tells the calling program ended abnormally and why.

- ✔ NOTTFY message

Notify message from a CL program can be sent to the message queue of calling program or to the external message queue. A notify message tells the calling program about a condition under which processing can continue.

- ✔ Predefined message are stored in message file
  - ➢ To create a message file

CRTMSGF MSGF (MFILE) SIZE () AUT () TEXT ()

- Create and maintain messages
  - ADDMSGD
  - CHGMSGD OR WRKMSGD
  - DSPMSGD
  - RMVMSGD

Message file QCPFMSG in library QSYS contain the system message

2. What will MONMSG command in do?

The monitor message (MONMSG) command monitors the message send to the program message queue for the conditions specified in the command. If condition exists, the CL command specified on the MONMSG command is run.

- Types of monitor message
  - ✓ Escape Message
  - ✓ Status or Notify Message
- Escape Message

  Escape message are send to tell your program of an error condition that forced the sender to end. By monitoring for escape message, you can take corrective actions or clean up and end your program.

- Status or Notify Message

  Status and notify message are send to tell your program of an abnormal condition that is not serious enough for sender to end. By monitoring for status or notify message, your program can detect this condition and not allow the function to continue.

- Two levels of MONMSG command
  - ✓ Program level
  - ✓ Specific command level
- Program level

  The MONMSG is specified immediately following the last declare command in your CL program. You can use as many as 100 program-level MONMSG commands in a program.

- Specific command level

  Here the MONMSG command immediately follows a CL command. You can use as many as 100 commands-level MONMSG commands for a single command.

- Monitor message command syntax

MONMSG MSGID () CMPDTA () EXEC ()

- ✔ MSGID-Required

Ex: MSGID (MCH1211)

- ✔ CMPDTA –(Optional)

Ex: MONMSG MSGID (MCH1211) CMPDTA (LIB)

- ✔ EXEC - -(Optional)

CL command

2. What are the statements, which is not used in CLLE that is used in CLP?

- ✔ RCLRSC which is replaced by RCLACTGRP
- ✔ TFRCTL

2. How to create user define command?

By using the CRTCMD command process the command definition statements to create the command definition object. The CRTCMD command may be run interactively or in a batch job.

Steps for creating CRTCMD commands

1. Enter the command definition statements into the source file

Command type CMD

```
                                        SKANDASAMO/CLP

                                             CMD1

*************** Beginning of data ****************************

0001.00 CMD

****************** End of data *******************************
```

2. Enter source program in any language

```
                                        SKANDASAMO/CLP

                                             DLIB

TYPE          : CLP

*************** Beginning of data ******************************
```

```
0001.00 PGM

0002.00 DSPLIBL

0003.00 ENDPGM

        ****************** End of data **********************************
```

3. Create the command by using CRTCMD take f4

```
          Command  . . . . . . . . . . . . > KS           Name

Library  . . . . . . . . . . . >   SKANDASAMO  Name, *CURLIB

 Program to process command . . . > DLIB          Name, *REXX

 Library  . . . . . . . . . . . >   SKANDASAMO  Name, *LIBL, *CURLIB

Source file  . . . . . . . . . . > CLP          Name

 Library  . . . . . . . . . . . >   SKANDASAMO  Name, *LIBL, *CURLIB

Source member  . . . . . . . . . > CMD1         Name, *CMD

Threadsafe . . . . . . . . . . .   *NO          *YES, *NO, *COND

 Multithreaded job action . . . .   *SYSVAL       *SYSVAL, *RUN, *MSG, *NORUN

Text 'description' . . . . . . .   *SRCMBRTXT
```

3. Info

- ✔ DEFAULT CL MSGID?
  CPF0000
- ✔ Dspf windows type?
  WINDOW
- ✔ What is the level check error?
  The level check error means RPGLE program is compiled and PF or LF are compile suppose the PF or LF compile after the compiling the RPGLE program this type of error is called level check error.
- ✔ If you want to copy a PF without making any modification to it then FORMAT keyword is used.
- ✔ Default access path maintenance is *IMMED
- ✔ Maximum no of printer files included in a RPGLE program is 8

- ✔ Maximum no of files declared in RPGLE is 50 and CL is l
- ✔ Maximum no of key fields included is 120
- ✔ Maximum no of fields included in a PF is 8000
- ✔ Maximum no of arrays included is a RPG is 200
- ✔ Maximum no of parameter passed in a RPG is 255 and CL 40
- ✔ Total no of system library is 15 and user library is 25
- ✔ While logging on the first library to be included is QSYS
- ✔ QGPH and QTEMP are user library
- ✔ Printer file default length is 132.
- ✔ Default size of a member is CRTSRCPF command for ordinary files is 92**.**

2. **What's the difference between CHAIN and SETLL? Is there a performance advantage?**

There are two important differences between CHAIN and SETLL.

1. The CHAIN operation applies a record lock to files that are open or update. The SETLL operation does not apply the lock.
2. The CHAIN operation copies the record's data to the input buffer for the program. The SETLL operation does not.

More Details

The CHAIN operation performs a random GET operation to the database file. If the operation is successful, the data in the record is copied to the input buffer. If the CHAIN operation fails, a record-not-found condition is signaled, typically via Resulting Indicator 1. If the database file has been opened for UPDATE, the CHAIN operation places a record lock on the retrieved record. No other application can access this record for update while this lock is applied. Furthermore, if another program has issued a lock to the recording being accessed, the CHAIN operation will wait for the database time-out period. If the record is released during that period, the CHAIN operation continues. If the other program does not release the record, the CHAIN operation fails with an exception.

CHAIN with NO LOCK

The CHAIN operation supports the NO LOCK operation extender (the old "half-adjust" column). In RPG III you specify an N in the operation extender column, in RPG IV, you specify CHAIN (n) for the operation code. Using NO LOCK allows you to access a record without a record lock being applied, regardless of the way in which the file is open. The record's data, however, is still copied to the input buffer when NO LOCK is specified.

The SETLL operation performs a quasi READ LESS THAN OR EQUAL operation. If the operation is successful, a READ PRIOR is performed. The database record's data, however, is *not* copied to the input buffer, nor is there a record lock applied to the accessed record. Hence, SETLL is probably the operation code to use for testing the existence of a record. However, if the record needs to be retrieved, CHAIN more effective.

Performance

If your requirement is to check for the existence of a record, traditionally the CHAIN operation is used. However, since CHAIN copies the record's data to your program's input buffer, there is additional overhead required for the CHAIN operation. The SETLL can be used to effectively accomplish the same task as the CHAIN test. Use SETLL with resulting indicator 3 (equal). If this indicator is set on, a record exists whose key matches they value specified in Factor 1. If your requirement is that the record eventually be updated, subsequent to the existents test, you should consider using of CHAIN.

2. **How do I debug a remote (i.e. "batch") job from an interactive job?**

The ability to debug another job has been a long-standing requirement for AS/400, now Iseries programmers. It isn't as difficult as it may seem. Whether you need to debug a batch job, another interactive job, or an HTTP server job (browser/CGI program), the following steps can get you started.

Starting Debug for a Remote Job

1. Determine the job name of number for the job you need to debug.
   - ✔ Use WRKACTJOB and note the Job name, number and user profile ID.
   - ✔ If debugging a CGI program, look in the job log of the job for CPF message HTP2001.
1. Run the Start Service Job (STRSRVJOB) command specifying the job to be debugged
   - ✔ E.g., STRSRVJOB JOB (012345/usrid/jobname)
1. Run Start Debug (STRDBG) on the program to be debugged
   - ✔ E.g., STRDBG PGM (libnam/pgmname) UPDPROD(*YES | <u>*NO</u>)
1. At this point the program in the remote job is under debug control from your job
   - ✔ You can now set break points (if you're debugging an RPG IV program, the source will have already been displayed).
   - ✔ Press F12 from within the debugger to return to CMD entry after setting your break points.
1. Evoke the program in the remote job. If you you're doing a web browser session, hit the SUBMIT button.
2. You interactive job will "break" at the debug break points and you can debug application normally.

Ending Debug for a Remote Job

Ending the debug session should be done in the following sequence.

1. From your debugging session, run the End Debug (ENDDBG) command
2. Then run the (End Service Job) ENDSRVJOB command

Your session is no longer controlling the remote job. The remote job continues normally.

Special Considerations when Debugging a Web Browser/CGI Program

To debug a CGI program that is evoked from a Web Browser session running from the standard IBM HTTP Web Server, you need to do the following in addition to the above.

Before Starting Debug for a Web Browser/CGI Session/Program

- ❖ End the HTTP Server using the following CL command:
  - ✔ ENDTCPSVR *HTTP
  - ✔ WARNING!!! -- You MUST include <u>*HTTP</u> as the parameter for ENDTCPSVR otherwise all TCP/IP server jobs (including telnet, ftp, smtp, etc.) will be ended. And this is a bad thing. IBM sucks for making *ALL the default for ENDTCPSVR.
- ❖ Restart the HTTP Server using the following CL command:
  - ✔ STRTCPSVR *HTTP HTTPSVR(DEFAULT '-minat 1 -maxat 1')
  - ✔ This restarts the HTTP server with once instance of each job type (one for CGI, one for Java, etc.)
  - ✔ Using WRKACTJOB in the QHTTPSVR subsystem location the jobs running.
  - ✔ The job whose joblog contains the CPF message HTP2001 is the one to be debugged.

After Finishing the Debug Session

- ❖ End the HTTP server using the following CL command:
  - ✔ ENDTCPSVR *HTTP
- ❖ Restart the HTTP server using the following CL command, unless your shop has another process for starting the HTTP server:
  - ✔ STRTCPSVR *HTTP

Your system should be back to normal.

2. **What is the new E operation extender used for?**

The new (E) operation extender is used to cause the %ERROR and %STATUS built-in functions to be initialized after an operation is performed. That is, these built-in functions and the E operation extender are used in place of Resulting Indicator 2 on all operation codes that currently support Resulting Indicator 2 as an error condition.

For example, to check to see if a record is locked, you would code the following:

```
.....CSRn01Factor1+++++++OpCode(ex)Factor2+++++++Result++++++++
+Len++DcHiLoEq
    C    CustNO        Chain(E)  CustMast
    C                  if        %ERROR = *ON
    C                  Select
    C                  When      %STATUS = 1221
    C                  exsr      UpdateNoRead
    C                  When      %STATUS = 1218
    C                  exsr      RecdLocked
    C                  endSL
    C                  ELSE
    C                  if        %FOUND( CustMast )
    C                  exsr      whatever...
    C                  endif
    C                  endif
```

The concept is to first check %ERROR for a generalized error condition, and then check %STATUS for the specific error. Note that no resulting indicators are used in the previous example. The normal *not-found* condition is checked using the %FOUND built-in function rather than testing Resulting Indicator 1.

3. Why doesn't the %CHAR built-in function work with numeric values?

Under the initial release of OS/400 Version 4, Release 2, the %CHAR built-in function was introduced. However, the function, as designed, only converted DATE values to character values. This proved to be too restrictive a use for this function. In the next release of OS/400 (V4R4) IBM will add function to %CHAR allowing it to convert all forms of non-character data to character. In that release %CHAR will function with numeric values.

```
    D Amount                       7P 2  Inz(123.45)
    C                  Eval      text = 'The amount is: ' + %Char (
                                     amount )
```

The TEXT field would contain the following after the EVAL operation is performed:

'The amount is: 123.45'

Unlike %EDITC, the %CHAR built-in function trims off leading blanks. However, %EDITC provides much more editing power than %CHAR. Use %CHAR for basic number to character conversion.

4. **How does the CONST keyword work with Procedure parameters?**

If you are certain that the called procedure will NOT modify a parameter, the CONST keyword can provide several benefits.

1. It automatically converts a field of a similar data type, to the length and type required by the parameter.

   What this means, is say a parameter is a 15 position pack field, with 5 decimals. Normally, you'd have to specify a Pdk(15,5) field for the parameter. However, if the parameter is read-only, you can specify CONST on the Prototype and Procedure Interface for the parameter. When you do this, the compiler automatically converts the value (say it's a literal of 27) to the size and type required by the parameter. This works really cool with DATE fields. A date for any format can be passed as a parameter value when that parameter value is CONST.

2. Performance is improved because the compiler can generate more optimized code for the CONST parameter.

   CONST can be used on calls to procedures or programs. We use it all the time when calling QCMDEXC from within RPG IV. All three parameters of the QCMDEXC program are CONST values. The example code below can be used as the PROTOTYPE to call QCMDEXC from within RPG IV. To call it using this prototype, specify something like: **CALLP run('addlible myLib' 14)** in your calculation specs.

```
.....DName+++++++++++EUDS.......Length+TDc.Functions+++++++++++++
+
    D Run           PR                ExtPgm('QCMDEXC')
    D cmdstr                    3000A   Const Options(*VarSize)
    D cmdlen                      15P 5 Const
    D cmdDbcs                      3A   Const Options(*NOPASS)
```

Note: if you're using CodeStudio or IBM's Code/400 as your RPG IV editor under Windows, you could simply highlight the above source code within your Internet Browser, and copy it to the Windows clipboard. Then activate CodeStudio (or Code/400) and use the Paste function to insert the code directly into the editor. Pretty cool, huh? <g>

Built-in Functions

1. RPG IV - Built-in Functions

The original release of RPG IV included a set of built-in functions. These built-in functions were:

%ADDR, %PADDR, %SIZE, %ELEM, %SUBST, %TRIM, %TRIML, %TRIMR

In addition, under OS/400 V3R2 and V3R7 the %PARMS built-in function was introduced. Since then, several built-in functions have been added to RPG IV. The following table provides the OS/400 Version and Release that the specific built-in functions were introduced and/or enhanced.

NOTE: IBM Seems to skip-ship the RPG IV compiler. So RPG IV in V4R1, V4R3 and V4R5 has no new functionality. The next scheduled upgrade is OS/400 V5R1 in spring 2001.

Version Release	Built-in Function	Parameters	Return Value Description
V3R7	%ABS	numeric expression	Absolute value of expression
	%ADDR	variable name	Address of variable
V5R1	%ALLOC	memory size	Pointer to the allocated storage.
V4R2 V4R4	%CHAR	graphic, date, time, timestamp, or numeric expression	Value in character data type
V5R1	%CHECK	compare-value : data-to-search { : start-position }	First position in the searched-data that contains a character not in the list of the characters in the compare value.
V5R1	%CHECKR	compare-value : data-to-search { : start-position }	Last position in the searched-data that contains a character not in the list of the characters in the compare value. (Search

			begins with the right-most character and proceeds to the left.
V5R1	%DATE	{ value { : date-format-code }	A date data-type value after converting the "value" to the specified date format. If no value is specified, the current system date is returned.
V5R1	%DAYS	days	A duration value that can be used in an expression to add a number of days to a date value.
V3R7	%DEC	numeric expression {:digits : decpos}	Value in packed numeric format. If digits and decpos are specified the result value is formatted to fit in a variable of the number of digits specified.
V3R7	%DECH	numeric expression : digits : decpos	Half-adjusted value in packed numeric format. The length and decimal positions
V3R7	%DECPOS	numeric expression	Number of decimal digits.
V5R1	%DIFF	start-date : end-date : duration-code	Calculates the difference between two date fields. The type of difference returned is specified by the duration-code.
V4R4	%DIV	Numerator : Denominator	Performs integer division and returns the quotient (result)

			of that division operation.
V3R7	%EDITC	non-float numeric expression : edit code {:*CURSYM | *ASTFILL | currency symbol}	String representing edited value.
V3R7	%EDITFLT	numeric expression	Character external display representation of float.
V3R7	%EDITW	non-float numeric expression : edit word	String representing edited value
	%ELEM	array, table, or multiple occurrence data structure name	Number of elements or occurrences
V4R2	%EOF	{file name}	'1' if the most recent file input operation or write to a subfile (for a particular file, if specified) | ended in an end-of-file or | beginning-of-file condition '0' otherwise.
V4R2	%EQUAL	{file name}	'1' if the most recent SETLL (for a particular file, if specified) or LOOKUP operation found an exact matches '0' otherwise.
V4R2	%ERROR		'1' if the most recent operation code with extender 'E' specified resulted in an error '0' otherwise.

V3R7	%FLOAT	numeric expression	Value in float format.
V4R2	%FOUND	{file name}	'1' if the most recent relevant operation (for a particular file, if specified) found a record (CHAIN, DELETE, SETGT, SETLL), an element (LOOKUP), or a match (CHECK, CHECKR, SCAN) '0' otherwise.
V4R4	%GRAPHIC	Any character value	Converts character data to double-byte character set value.
V5R1	%HOURS	hours	A duration value that can be used in an expression to add a number of hours to a time value.
V3R7	%INT	numeric expression	Value in integer format
V3R7	%INTH	numeric expression	Half-adjusted value in integer format
V3R7	%LEN	any expression	1. Returns the length of a variable or literal value, or the current length of a varying length field. 2. When used on the left side of the equal sign, sets the length of a varying length field.

V5R1	%LOOKUPxx	search-data : array { : start-index { : elements to search }}	An array index of the element in the array where the search-data is located.
V5R1	%TLOOKUPxx	search-data : searched-table { : alternate-table }	*ON if the search is successful, otherwise *OFF. (NOTE: The indexes of the searched-table and alternate-table are set to the index of the search-data if *ON is returned.)
V5R1	%MINUTES	minutes	A duration value that can be used in an expression to add a number of minutes to a time value.
V5R1	%MONTHS	months	A duration value that can be used in an expression to add a number of months to a date value.
V5R1	%MSECONDS	milliseconds	A duration value that can be used in an expression to add a number of milliseconds to a time value.
V3R7	%NULLIND	null-capable field name	Value in indicator format representing the null indicator setting for the null-capable field.
V5R1	%OCCUR	data-structure	The current occurrence of the data structure, or sets the current occurrence of the data structure

V4R2	%OPEN	file name	'1' if the specified file is open '0' if the specified file is closed. Consider this built-in to be an 'Is this file open?" operation.
	%PADDR	procedure name	Address of procedure
V3R2 V3R6	%PARMS		Number of parameters passed to procedure
V5R1	%REALLOC	pointer : new-size	Pointer to the allocated storage.
V4R4	%REM	Numerator : Denominator	Performs integer division and returns the remainder from the division operation.
V4R2	%REPLACE	replacement string: source string {:start position {:source length to replace}}	String produced by inserting replacement string into source string, starting at start position and replacing the specified number of characters.
V3R7	%SCAN	search argument : string to be searched {:start position}	First position of search argument in string or zero, if not found.
V5R1	%SECONDS	seconds	A duration value that can be used in an expression to add a number of seconds to a time value.
V5R1	%SHTDN		*ON if the job is being shut down (e.g., when the

			PWRDWNSYS command is issued) otherwise *OFF is returned.
	%SIZE	variable, data structure, array, or literal {: *ALL}	Number of bytes used by variable or literal. *ALL returns the number of bytes used by all the elements of the array, or all the occurrences of the data structure.
V5R1	%SQRT	expression or value	The square root of the expression or value.
V4R2	%STATUS	{file name}	0 if no program or file error occurred since the most recent operation code with extender 'E' specified most recent value set for any program or file status, if an error occurred if a file is specified, the value returned is the most recent status for that file.
V3R7	%STR	pointer{:maximum length}	Characters addressed by pointer argument up to but not including the first x'00'.
V5R1	%SUBDT	date : duration-code	The extracted component of the date value. (The functional equivalent of the EXTRCT operation code.)
	%SUBST	string: start{:length}	Substring value. If length is not specified, the substring begins with start and

			continues through the end of the string.
V5R1	%THIS		Used for Java integration. Returns an Object reference.
V5R1	%TIME	{ value { : time-format-code }	A time data-type value after converting the "value" to the specified time format. If no value is specified, the current system time is returned.
V5R1	%TIMESTAMP	{value { : *ISO | *ISO0 }	A timestamp data-type value with or without separators.
	%TRIM	string	String with left and right blanks trimmed (removed)
	%TRIML	string	String with left blanks trimmed
	%TRIMR	string	String with right blanks trimmed
V4R4	%UCS2	Any character value	Returns a varying length value.
V4R2	%UNS	numeric expression	Value in unsigned format
V4R2	%UNSH	numeric expression	Half-adjusted value in unsigned format
V5R1	%XLATE	from-table : to-table : string-to-	The converted string is

		convert { starting-position }	: returned.
V4R4	%XFOOT	Array name	Cross foots (totals) all the elements in an array.

1. **Figurative constants in RPGLE**

*HIVAL, *LOVAL, *ZERO, *ZEROS, *BLANKS, SETLL, SETGT.

2. **Explain ADDDUR, SUBDUR, EXTRCT and TEST?**

ADDDUR:
It is a powerful opcode, which is used to add any date related function to a particular date, time or timestamp.
Example:

SKANDASAMO/DATE

ADDDUR

```
*************** Beginning of data *******************************
0000.01 d*date function using the adddur
0001.00 DTIMESTE       S              Z
0001.01 DTIME5          S              Z
0002.00 DDATE1           S              D
0002.01 DDATE2           S              D
0002.02 DDATE3           S              D
```

```
0002.03 DDATE4            S               D
0003.00 DTIME1            S               T
0003.01 DTIME2            S               T
0003.02 DTIME3            S               T
0003.03 DTIME4            S               T
0004.00 C                   MOVEL     *DATE         DATE1
0005.00 C     DATE1         ADDDUR    02:*Y         DATE2
0006.00 C     DATE1         ADDDUR    05:*M         DATE3
0007.00 C     DATE1         ADDDUR    01:*D         DATE4
0007.01 C                   TIME                    TIME1
0007.02 C                   TIME                    TIMESTE
0008.00 C     TIME1         ADDDUR    10:*H         TIME2
0009.00 C*    TIME1         ADDDUR    10:*ML        TIME3
0010.00 C     TIME1         ADDDUR    10:*S         TIME4
0011.00 C     TIMESTE       ADDDUR    10:*MS        TIME5
0012.00 C     DATE2         DSPLY
0013.00 C     DATE3         DSPLY
0014.00 C     DATE4         DSPLY
0014.01 C     TIME1         DSPLY
0014.02 C     TIME2         DSPLY
0014.03 C     TIME3         DSPLY
0014.04 C     TIME4         DSPLY
0014.05 C     TIME5         DSPLY
0015.00 C                   SETON                                        LR
```

```
        ****************** End of data *********************************
OUTPUT
 DSPLY 2004-06-27
 DSPLY 2002-11-27
 DSPLY 2002-06-28
 DSPLY 12.18.36
 DSPLY 22.18.36
 DSPLY 00.00.00
 DSPLY 12.18.46
 DSPLY 2002-06-27-12.18.36.953010
```

SUBDUR:

It is used to find the difference between two date (or) time (or) time stamp

Example

```
                                                    SKANDASAMO/DATE
                                                         SUBDUR
        *************** Beginning of data ********************************
0000.01 d*date function using the SUBDUR
0001.00 DTIMESTE          S              Z
0001.01 DTIME5           S              Z
0002.00 DDATE1           S              D   INZ (D'1977-06-20')
0002.01 DDATE2           S              D
0002.02 DDATE3           S              D
```

```
0002.03 DDATE4            S               D
0003.00 DTIME2            S               T   INZ (T'12. 50.10')
0003.01 DTIME1            S               T
0003.02 DTIME3            S               T
0003.03 DTIME4            S               T
0005.00 C     DATE1         SUBDUR    02:*D         DATE2
0006.00 C     DATE1         SUBDUR    05:*M         DATE3
0007.00 C     DATE1         SUBDUR    01:*Y         DATE4
0007.01 C                 TIME                  TIME1
0008.00 C     TIME2         SUBDUR    10:*H         TIME1
0010.00 C     TIME2         SUBDUR    10:*S         TIME4
0012.00 C     DATE2         DSPLY
0013.00 C     DATE3         DSPLY
0014.00 C     DATE4         DSPLY
0014.01 C     TIME1         DSPLY
0014.02 C     TIME2         DSPLY
0014.03 C     TIME3         DSPLY
0014.04 C     TIME4         DSPLY
0015.00 C                 SETON                                     LR
        ****************** End of data **********************************
```

OUT PUT

```
 DSPLY 1977-06-18
 DSPLY 1977-01-20
 DSPLY 1976-06-20
```

```
DSPLY 02.50.10

DSPLY 12.50.10

DSPLY 00.00.00

DSPLY 12.50.00
```

EXTRCT:

It is used to extract year, month, day, hours, minutes, seconds, and microseconds of a time stamp or date field.

Example:

```
                    SKANDASAMO/DATE

                                                   EXRCT

         *************** Beginning of data ********************************

0000.01 d*FINT THE EXRCT DAY MONTH YEAR

0001.00 DTIMESTE          S               Z

0002.00 DDATE1            S               D   INZ (D'1977-06-20')

0002.01 DDATE3            S               D

0002.02 DDATE2            S              5P 0

0002.04 DDATE4            S              5P 0

0002.05 DDATE5            S              5P 0

0002.06 DDATE6            S              5P 0

0002.07 DDATE7            S              5P 0

0002.08 DDATE8            S              5P 0

0003.00 DTIME0            S               T   INZ (T'12. 50.10')

0003.01 DTIME1            S               T

0003.02 DTIME2            S              5P 0

0003.03 DTIME4            S              5P 0
```

```
0003.04 DTIME5            S              5P 0
0003.05 DTIME6            S              5P 0
0003.06 DTIME3            S             26P 0
0004.00 C                      MOVEL     *DATE         DATE3
0004.01 C                      TIME                    TIME1
0004.02 C                      TIME                    TIMESTE
0005.00 C                      EXTRCT    DATE1:*M      DATE2
0006.00 C                      EXTRCT    DATE3:*M      DATE4
0007.00 C                      EXTRCT    DATE1:*D      DATE5
0007.01 C                      EXTRCT    DATE3:*D      DATE6
0007.02 C                      EXTRCT    DATE1:*Y      DATE7
0007.03 C                      EXTRCT    DATE3:*Y      DATE8
0007.04 C                      EXTRCT    TIME1:*H      TIME2
0007.05 C                      EXTRCT    TIME1:*H      TIME4
0007.06 C                      EXTRCT    TIME0:*H      TIME5
0007.07 C                      EXTRCT    TIME0:*S      TIME6
0011.00 C                      EXTRCT    TIMESTE:*MS   TIME3
0012.00 C     DATE2         DSPLY
0013.00 C     DATE8         DSPLY
0013.01 C     DATE4         DSPLY
0013.02 C     DATE5         DSPLY
0013.03 C     DATE6         DSPLY
0014.00 C     DATE7         DSPLY
0014.01 C     TIME2         DSPLY
```

```
0014.02 C     TIME4          DSPLY
0014.03 C     TIME5          DSPLY
0014.04 C     TIME6          DSPLY
0014.05 C     TIME3          DSPLY
0015.00 C                    SETON                                  LR
```

OUTPUT

```
DSPLY      6
 DSPLY   2002
 DSPLY      6
 DSPLY     20
 DSPLY     27
 DSPLY   1977
 DSPLY     12
 DSPLY     12
 DSPLY     12
 DSPLY     10
 DSPLY                    441000
```

TEST:

Test is the most powerful opcode, which will check a date is a valid, or not .The low level indicator is set on if the date is not valid or set off if the date is a valid one.

Test will be given with extended factor like test (d), test (t), test (z) for date, time and time stamp and if test without extended factor default to date (z).

Example

SKANDASAMO/DATE

```
                                              TEST
   *************** Beginning of data **********************************
0000.01 C*TEST FOR VALID DATE THE DATE VALID SETOFF OR SETON
(NOTVALID)
0001.00 C                MOVEL     '13/03/1999’ A              10
0002.00 C                TEST (D)              A               30
0003.00 C     *IN30        DSPLY
0003.01 C                IF        *IN30=*ON
0003.02 C     'NOTVAILD'    DSPLY
0003.03 C                ELSE
0003.04 C     'VALID'       DSPLY
0003.05 C                ENDIF
0004.00 C                SETON                                 LR
   ****************** End of data ***********************************
```

OUTPUT

```
DSPLY 1
DSPLY NOTVAILD
```

3. **Explain Compile time array, lookup, sort-a, x-foot, and Run time array?**

❖ **Compile time array**

- ✔ The compile time array means the elements of the array will be loaded before the execution of the programs.
- ✔ The value will be static.
- ✔ We must declare in keyword command DIM (), CTDTAT (), and PERRCD ().
- ✔ We are giving the value in after the SETON   LR.

Example

```
                                              SKANDASAMO/ARRAY
                                                   COMILE
    *************** Beginning of data ********************************
0000.01      c*compile time array
0001.00      darr1          s            4    dim(3) ctdata perrcd(1)
0002.00      di             s            2p 0 inz(1)
0003.00      c     i        do        3
0004.00      c     arr1(i)  dsply
0005.00      c              add       1            i
0006.00      c              enddo
0007.00      c              seton
0008.00 **
0009.00 1001
0010.00 20
0011.00 1000
    ****************** End of data *******************************
```

OUTPUT

DSPLY 1001

 DSPLY 20

DSPLY 1000

❖ **Run time array**
- ✔ The run time array means the value will be loaded during the runtime only.
- ✔ The value will be dynamic.

```
                                              SKANDASAMO/ARRAY
                                                   RUNTIME
```

```
        *************** Beginning of data *****************************
0000.01 c*runtime array
0001.00 darr1            s             10    dim(12)
0002.00 di               s              2p 0 inz(1)
0002.01 da               s              3p 0
0002.02 dj               s              2p 0 inz(1)
0003.00 c     i             do        12
0003.01 c                   dsply                  arr1(i)
0003.02 c*                  eval      arr1(i)=a
0003.03 c                   add       1             i
0003.04 c                   enddo
0003.05 c     j             do        12
0004.00 c     arr1(j)    dsply
0005.01 c                   add       1             j
0006.00 c                   enddo
0007.00 c                   seton                                        lr
        ****************** End of data *********************************
```

lookup, sorta, xfoot :

SKANDASAMO/ARRAY

SORTARRAY

```
        *************** Beginning of data ********************************
0000.01 c*lookup,xfoot&sorta examples
0001.00 darr1            s              4  0 dim(3) ctdata perrcd(1)
0002.00 di               s              2p 0 inz(1)
```

```
dj              s             2p 0 inz(1)
dd              s              4p 0
c     i            do        3
c     arr1(i)    dsply
c                  add       1             i
c                  enddo
c                  sorta     arr1
c                  xfoot     arr1          d
c     1000       lookup    arr1                              40
c                  if        *in40=*on
c     'found'   dsply
c                  else
c     'notfou'  dsply
    c              endif
    c     d        dsply
    c     j          do        3
    c     arr1(j)   dsply
    c                  add       1             j
    c                  enddo
    c                  seton
**
1001
2000
1000
```

```
****************** End of data ***********************************
DSPLY  1001
 DSPLY  2000
 DSPLY  1000
 DSPLY  found
 DSPLY  4001
 DSPLY  1000
 DSPLY  1001
 DSPLY  2000
```

❖ **Pre runtime array**

- ✔ Pre runtime array is in between these 2 conditions where the value is static and the value will be retrieved from disk and loaded into the array.
- ✔ As a result there is no need to retrieve the value every time from the disk and usage of pre runtime array makes it fast.
- ✔ We must declare in keyword command DIM (), FROMFILE (), and PERRCD ().

1. **What is the different between READE and CHAIN Opcodes?**

READE	CHAIN
1.The matching records for table	The first matching records only
2. We are using the looping concept	Looping is not necessary
3.The indicator Set In the EQ	The indicator Set In the HI
4. We are most using in SETGT or SETLL	It is not necessary

2. **Explain Build in function in ILE?**

❖ %SUBST **(String name: String position: length)**

- %ABS **(Absolute value by omitting sign)**
- %EDITC **(string: 'X')**

  In a application if we want to concatenate a string with a numeric then we can use this %EDITC

  Example:

```
                                        SKANDASAMO/BULID
                                            EDITC
   *************** Beginning of data *******************************
0001.00 da             s           10a   inz('shyam')
0002.00 db             s           10p 0 inz(20)
0003.00 dc             s           10a   inz('sundar')
0004.00 dd             s           10s 0 inz(12)
0005.00 de             s           10a   inz('rambabu')
0006.00 dout            s           50a
0007.00 c              eval      out=a+%editc(b:'X')+c+%editc(d:'X')+e
0009.00 c     out        dsply
0010.00 c              seton                              lr
0011.00
   ****************** End of data **********************************
```

OUTPUT

```
DSPLY  shyam     0000000020sundar    0000000012rambabu
```

- %REPLACE **(Replacing string, actual string, starting position, offset)**

  Here we are replacing senthilkumar **from position 4 to 3 by** kum. **The output will be** senkumlkumar.

```
                                   SKANDASAMO/BULID
                                       REPLACE
```

```
        *************** Beginning of data ****************************
0000.01 d*replace the string using keyword %replace
0001.00 dc              s            16a   inz('senthilkumar')
0002.00 db              s            20a
0003.00 c                eval     b=%replace('kum':c:4:3)
0004.00 c     b           dsply
0005.00 c                seton                              lr
        ****************** End of data ****************************
```

OUTPUT

```
DSPLY  senkumlkumar
```

❖ %TRIM **(%TRIML, %TRIMR)**

The use of the TRIM functions is very limited, in that they support only the use of character variables and data structures. Numeric fields and zero-fill values are not supported. They do, however, provide some useful function for string handling. For example, in RPG IV, one line of code is all that's needed to left-adjust a value within a field. For example:

```
.....CCRn01Factor1+++++++OpCode(ex)Factor2+++++++Result++++++++Len++DcHiLoEq
C                   ExFmtCustMaint
     C                   Eval      CustName=%TrimL(CustName)
```

Typically, the %TRIM function is the only one of the three that get used. The other two, however, do have their place.

%TRIM removes trailing and leading blanks from a field, and returns the remaining value, in place, within the expression. The returned value is treated similar to a constant value with leading or trailing blanks.

%TRIML removes leading blanks (trim-left) from a field, and returns the value in place, within the expression.

%TRIMR removes trailing blanks (trim-right) from a field, and returns the value in place, within the expression.

❖ %FOUND, %EOF, %EQUAL,%OPEN

OS/400 Version 4, Release 2 RPG IV supports the elimination of the Resulting Indicators. In their place, several new built-in functions have been introduced. Most of these new built-in function provide information about the result of File operations similar to the Result indicators. But instead of coding Resulting indicator 3, for example, to check for the end-of-file condition, you simply check the value of the %EOF built-in function.

The built-in functions that replace the Resulting Indicators include:

%FOUND, %EOF, %EQUAL. In addition, there are %OPEN, %STATUS, and %ERROR. Mysteriously missing is %LOCK to check for a record lock condition.

%FOUND returns an *ON or *OFF condition if the previous File operation returns a record-found condition. This is particularly useful on the CHAIN operation. Realize, however, that when CHAIN sets on Resulting indicator 1, a not-found condition is signaled. Whereas, without coding Resulting Indicator 1, the %FOUND built-in function returns the *found* condition.

%EOF can be used to check for end-of file, beginning of file, or subfile full conditions. A READ and READE return %EOF=*ON if the end of file is reached. READP and READPE return %EOF=*ON if the beginning of file is reached. The WRITE operation returns %EOF=*ON if the WRITE operation to a subfile detail record returned a subfile-full condition.

%EQUAL is used by the SETLL operation to indicate that it detected a record in the file with a key equal to that of the value specified in Factor 1. Since SETLL does not read the record, does not lock the record, and does not copy the data into the input buffer, SETLL is much faster and less of an impact on the performance of the application than other operations, such as CHAIN. Use CHAIN when you need to retrieve the record, use SETLL and %EQUAL when you need to only check for the existence of a record.

%OPEN is used to check to see if a file has already been opened. The built-in function returns *ON if the file is opened, otherwise it returns *OFF.

❖ %ELEM

%ELEM will display the array dimension

```
                                        ELEM
*************** Beginning of data ******************************
0001.00 darr1          s          3s 0 dim(100)
0002.00 dc             s          3s 0
0003.00 c              eval     c=%elem(arr1)
```

```
0004.00 c     c             dsply
0005.00 c                  seton                                        lr
         ****************** End of data **********************************
```

OUTPUT

DSPLY 100

- %SIZE

%SIZE will display the size of the variable

```
                                                            SKANDASAMO/BULID
                                                                   SIZE
         *************** Beginning of data *****************************
0000.01 D*BY USING THIS COMMAND FIND SIZE OF DATA VALUE
0001.00 darr1             s             10p 0 dim(10)
0002.00 dds1              s             10p 0 dim(20)
0003.00 dnum              s             20p 0
0004.00 c                  z-add     2            a                 20 0
0005.00 c                  movel     'senthil'     b                 10
0006.00 c                  eval      num=%size(a)
0007.00 c
0008.00 c     num           dsply
0009.00 c                  eval      num=%size(b)
0010.00 c     num           dsply
0011.00 c                  eval      num=%size(arr1)
0012.00 c     num           dsply
0013.00 c                  eval      num=%size(arr1:*all)
0014.00 c     num           dsply
```

```
0015.00 c                eval     num=%size(ds1)
0016.00 c     num          dsply
0017.00 c                eval     num=%size(ds1:*all)
0018.00 c     num          dsply
0019.00 c                seton                                  lr
       ****************** End of data *********************************
```

OUTPUT

```
DSPLY           6
DSPLY          11
DSPLY          10
DSPLY           6
DSPLY          60
DSPLY           6
DSPLY         120
```

- %EDITW

@ References:

http://www.allinterview.com/Interview-Questions/RPG400.html

http://publib.boulder.ibm.com/iseries/v5r1/ic2924/books/c092508380.htm#HDRPROGXPE

http://publib.boulder.ibm.com/iseries/v5r1/ic2924/books/c092508377.htm#HDRFILEXPE http://faq.midrange.com/data/cache/13.html

http://publib.boulder.ibm.com/iseries/v5r2/ic2924/info/dbp/rbafomst199.htm#HDROPNQF

http://www.geocities.com/SiliconValley/Hills/6632/opnqryf.html

http://www.allinterview.com/Interview-Questions/RPG400.html

http://sumedh.shende.googlepages.com/as400interviewquestionspartii