**RPG : Report program generator .**

This is basically high-level language which is used in IBMI or AS400.

Recent RPG type is RPG IV .

Previous versions of RPG like RPG III source type was RPG. It has certain limitation like no expanded opcodes/no definition specs etc.

Standard source physical file used for RPGLE sources is **QRPGLESRC.**

Under QRPGLESRC all rpgle source members will be present.

**--------------------------------------------------------------------------------------------------------------------**

**F-SPEC:** This is used to declare files:

**1.    File name**: Mention here name of the file that you are going to use in your program.

**2.      File Type**: Mention the file type as I,O,U,C where I= INPUT,(data base file) O=OUTPUT(data base and ;printer files but we will use mostly for printer files) ,U=UPDATE(base file) ,C=COMBINED.(Screens)

**3.      File Designation**: Mention theFile Designation as P,S,F where P=PRIMARY,S=SECONDARY,**F=FULL PROCEDURAL**.

**Primary File =** Record will be processed in the order; from start to end by rpg program cycle i.e. OPEN, READ, PROCESS, CLOSE. User can't change this order. There can be only one primary file in the program.

**Secondary Files =**Secondary files apply to programs that do multifile processing. All of the files involved in multifile processing, except the primary file, are secondary files.

**Full Procedural file(Most commonly used in rpgle) =** User can control any order by rpg program opcode. User can change any order by rpg opcode. With full procedural files the programmer determines which record or a block of records to be read by way of the value of the key field used.

**4.      File Addition**: Mention ‘A’ if you want to add record to the DISK FILE. In update mode of file, use ‘A’ file designation.

**5.      File Format**: Mention if the file is program described or externally described.

**6.      Record Address Type**: Mention this field entry as ‘K’ if the file is a keyed file, blank if the file is a sequential access file or based on RRN.

**7.      Device**: Mention the device as DISK, PRINTER or WORKSTN

**C SPEC** : This is used for any operation /calculation .

**D SPEC :** This is used for field definition. Fields can be standalone fields which is normal variable, Arrays or data structures.

**Opcodes in rpgle** :-

Different commands which we use for processing are called as operation codes or opcodes.

RPGLE opcode will contain factor 1 /factor 2 and result but it is not necessary that every opcode will need that

Most of the processes like calculations/assignment use **EVAL opcode**

RPGLE LITERALS :-

\*Zeros will be used for assigning values with 0 similarly \*blanks will be used for blanks.

End of RPGLE program :

Either Seton Indicatiob LR or use return.

**Sample of simple RPGLE program :**.

\* These are field definition

DA s 10 0

db s 10 0

dc s 10 0

\* These are calculations

c eval a = 10

c eval b = 20

c eval c = a + b

\* Initialize these variables.

c eval a = \*Zeros

c eval b = \*Zeros

c eval c = \*Zeros

\* End of program

c EVAL \*INLR = \*on

Built in functions in rpgle:-

1. **%SUBST(string :sart position : length**)

It is used to extract part of string from starting position given with this function in second parameter. It will extract total length equivalent to 3rd parameter.

Length parameter is not mandatory if whole string needs to be extracted from start position till end.

1. **%SCAN :** This is used to search character inside a string.

%scan(char to be scanned : string where char needs to be scanned :start position )

1. **%CHAR :** To change numeric/date value to a character.

%char(value which needs to be converted)

This also has second parameter, If we are converting date to a character value second parameter can be used to define date format which input variable is having.

1. **%XLATE :** This is used for translation purpose.

%xlate(from:To:String)

This will translate all character from one value to another value in string.

It can also be used to translate set of characters within a constant.

**Note** constant is the something where value will remain same throughout program cycle whereas variable values can be manipulated/changed during program cycle.

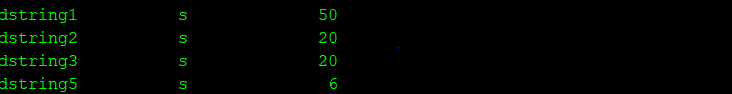
In rpgle like standalone variables constant also will be defined in D-SPEC. Value in constant can be specified under const keyword..

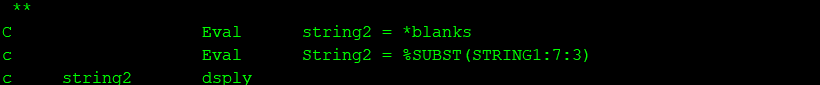
Declaration type will be ‘C’ for constant as compared to ‘S’ for standalone variables.

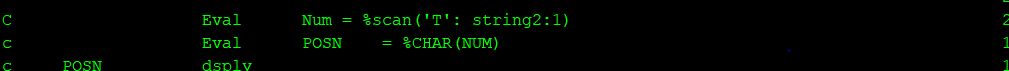
**Constant sample declaration :-**

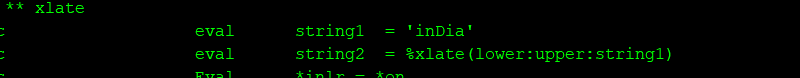


**Variables sample declaration: -**









1. **%len () :** This built in function is used to get length of string**.**
2. **%trim() , %triml() , %trimr():** This built in functions are used to trim spaces in particular string from both ends/ left/right.

Trim will remove spaces from both end (left and right)

TRIML will remove spaces from left

TRIMR will remove spaces from right

Example/

DA. 10 INZ(‘ABC’)

DB 3 0

B = %LEN(a) , In this case value of B will be 10.

B = %LEN(%TRIM(A)) , In this case value of B will b 3

# **%CHECK**

%CHECK function is used to find the position for non-occurrence of a character in a string.

The format of this function is %CHECK(comparator : base string {: Start position}).

1. **%CHECKR** function is used to find the position for non-occurrence of a character in a string from Right.

The format of this function is %CHECK(comparator: base string {: Start position})

1. **%DEC :** %DEC converts the value of the first parameter to decimal (packed) format.
2. **%DATE** is used to convert char/decima value to date.

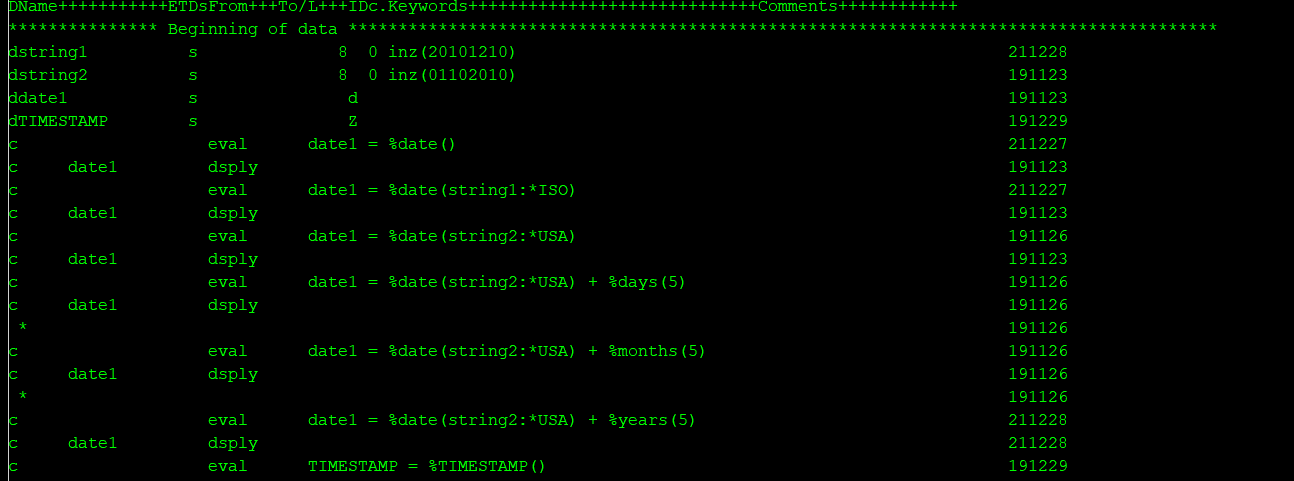
%date(first parameter , second paramete)

First parameter will be string/expression which needs to be converted to date.

Second parameter will be the date format of input.

1. **%days .** This is used to convert number into a duration which is number of days. This can be later used to add/subtract days from date field.
2. **%months .** This is used to convert number into a duration which is number of months. This can be later used to add/subtract months from date field..
3. **%years .** This is used to convert number into a duration which is number of years. This can be later used to add/subtract months from date field.
4. **Timestamp** : This will covert expression to TIMESTAMP , if nothings is specified this will return current time stamp.

**Simple examples showing date functions**





**Concatenation of strings**

Eval variable = string1 + string2 + string3

String 1 = ‘AB’

String 2 = ‘BC’

String 3 = ‘CD’

Varible = ‘ABBCCD’

Eval variable = string1 + ‘ ‘ + string2 + ‘ ‘ + string3+ ‘ ‘

1. Take name as input and extract first name .middle name and last name

Get first space

Posn = %scan(‘ ‘ :Name:1)

FisrtName = %subst(Name:1 : (posn – 1))

Previous posn = Posn

Get second space

Posn = %scan(‘ ‘ :Name : (previous posn + 1)

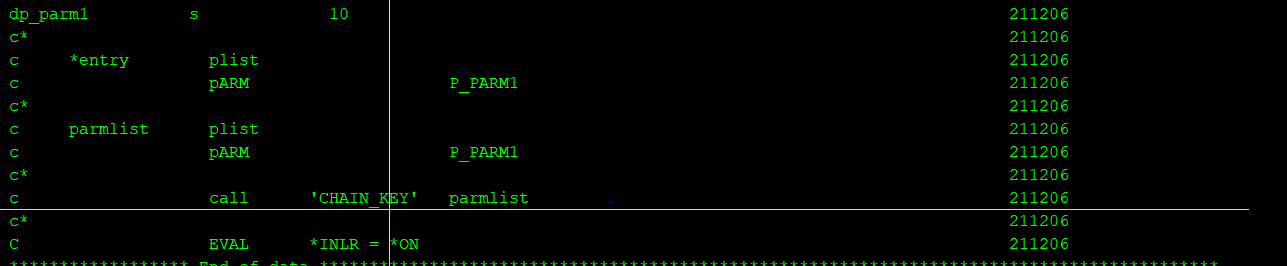
Middle name = %subst(Name : (prev posn + 1): (posn –(prev posn + 1)))

1. **Take input as first name . middle name and last name and formulate Name,**.

Name = first name + ‘ ‘ + Middle Name+ ‘ ‘ + Last Name.

**How to accept Parameters in RPGLE :-**

1. **PLIST(Parameter list** ) : This is list of parameters, defined in combination of PARM opcode. This is used while receiving parameters via \*ENTRY or to call any programs.

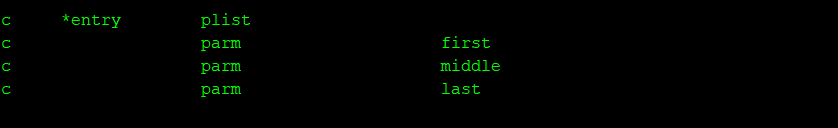


\*ENTRY is parameter list which program is accepting and PARMLIST is parameter list with which chain\_key is cal

1. PARM (These are parameters) This is to define parameters.

**Example if program A is calling program B , we will define parameter list which program B needs and call program B using that list**..

In rpgle we have \*ENTRY parameter list which we use to accept parameters.



,

**Loops in rpgle :**

1. **DOWHILE : -DO WHILE**

DOW (Do while till the condition remains true)

ENDDO

B)  **DOU : DO UNTILL**

DOU(Do until till the condition becomes true)

ENDDO

**C)**

For count = 1 till 100 by 3

**=**

DOW ( COUNT < = 100)

**{**

PROCESSING

COUNT = COUNT + 3

}ENDDO

FOR (For loop)

§  The For loop in RPGLE seems similar to a ‘FOR’ loop in any other programming languages.

§  The initial counter, the final counter and the increment value. The counter of the loop begins from 1 and is incremented/decremented by the step(2) till the counter becomes greater than (or less than) the max value(6

Example : ‘I I I I I’

Get all positions of I

FOR COUNTER I TILL 20 by 1

Posn = %scan(‘I’:String : Counter)

Endfor

Posn = 1

Startposn = 1

DOW Posn <> 0

Posn = %scan(‘I’ :String: Startposn)

Starposn = posn + 1 ;

Enddo

**LEAVE** : To jump out of loop immediately

**ITER:** To pass control to first line of loop.

Write a rpgle program using DOW loop to get all positions of a particular character in a string.

# **SETLL (Set Lower Limit)**

The SETLL operation positions a file at the next record that has a key or relative record number that is greater than or equal to the search argument (key or relative record number) operand specified (*search-arg*). The file must be a [full procedural file](https://www.ibm.com/docs/en/ssw_ibm_i_72/rzasd/fullprocfiles.htm).

**Example1: -** EMPPF is physical file with key as EMPID , this file is defined in program by key. Hence setll will search based on key

Record access type is ‘K ‘ .

F EMPF K

EVAL WEMPID = 100

WEMPID SETLL EMPPF (This will position at 100)

|  |  |
| --- | --- |
| **EMPPF** | |
| EMPID | EMPNAME |
| **100** | **A** |
| 200 | B |
| 300 | C |

**Example2:- -** EMPPF is physical file with key as EMPID , this file is accessed in program by arrival sequence Hence setll will search based on rrn(relative record number)

Record access type is ‘ ‘ .

F EMPF disk

EVAL WRECNO = 2

WRECNO SETLL EMPPF (This will position at record with record number 2)

|  |  |
| --- | --- |
| **EMPPF** | |
| EMPID | EMPNAME |
| 100 | A |
| **200** | **B** |
| 300 | C |

# **SETGT (Set GREATER Limit)**

The SETGT operation positions a file at the next record with a key or relative record number that is greater than the key or relative record number specified in factor 1.

**Example: -** EMPPF is physical file with key as EMPID , this file is defined in program by key. Hence setll will search based on key

Record access type is ‘K ‘ .

**F EMPF K**

EVAL WEMPID = 100

WEMPID SETGT EMPPF (This will position at 200 empid record)

|  |  |
| --- | --- |
| **EMPPF** | |
| EMPID | EMPNAME |
| 100 | A |
| **200** | **B** |
| 300 | C |

# **READ**

Read operation **reads the records of a full procedural file**. First of all it reads the record where currently the pointer is and then advances the pointer to the next record. The READ operation applies a record lock to files that are open in update mode

This will be used when there is a requirement to read all records of a file

# **READP**

READP **moves the pointer to the previous record and reads the record** and again moves the pointer to next previous position. If there are no more records it sets EOF \*ON..

# **READE**

The READPE operation retrieves the next sequential record from a full procedural file if the key of the record matches the search argument If there are no more records it sets EOF \*ON

This will be used when there is a requirement to read specific records in a file.

Example : Requirement is to get all orders of a particular country.

We will have a logical file with country as key. Example ORDERLF1

|  |  |
| --- | --- |
| ORDERPF | |
|  |  |
| Order num | Country |
|  |  |
| 100 | IND |
| 200 | UK |
| 400 | IND |
| 300 | UK |
| 500 | IND |

|  |  |
| --- | --- |
| Orderlf1 with Country as key in below order record will be accessed by program. | |
|  |  |
| 100 | IND |
| 400 | IND |
| 500 | IND |
| 200 | UK |
| 300 | UK |

If requirement is to read UK orders first set a pointer or position to first UK record and then issue READE

First instance of READE should be used with SETLL

EVAL WCOUNTRY = ‘UK’

WCOUNTRY’ SETLL ORDERLF1

WCOUNTRY’ READE ORDERLF1

DOW TILL END OF FILE

WCOUNTRY’ READE ORDERLF1

ENDDO

‘

# **READPE**

The READPE operation retrieves the next prior sequential record from a full procedural file if the key of the record matches the search argument If there are no more records it sets EOF \*ON

**CHAIN**

The CHAIN command does a SETLL and a READE in order to find a match. CHAIN is best used to locate a unique record (like a customer record) from a full ...

**%EOF(FILE NAME)**

%EOF built in function returns '1' if the most recent read operation or write to a subfile ended in an end of file or beginning of file condition; otherwise, it returns '0'.

**%FOUND{(file\_name)}**

%FOUND returns '1' if the recent operation finds a relevant/matching record; yet it is not necessary that there will be exact match. If the match is not found then it returns '0'.



**Update opcode :-**

This is used to update a data base file record..

1. File type in F-spec should be U
2. Record should be read (by using either read/reade/chain) for update

**Delete opcode :-**

This is used to delete record of a data base file.

1. File type in F-spec should be U
2. Record should be read (by using either read/reade/chain) for delete

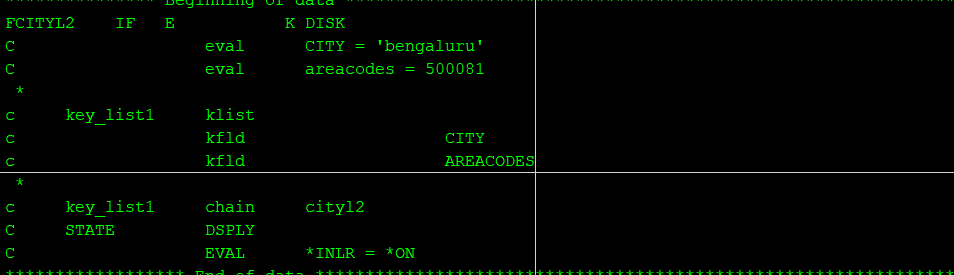
**Write opcode: -**

This is used to write records in a database file.

**KLIST opcode: -**

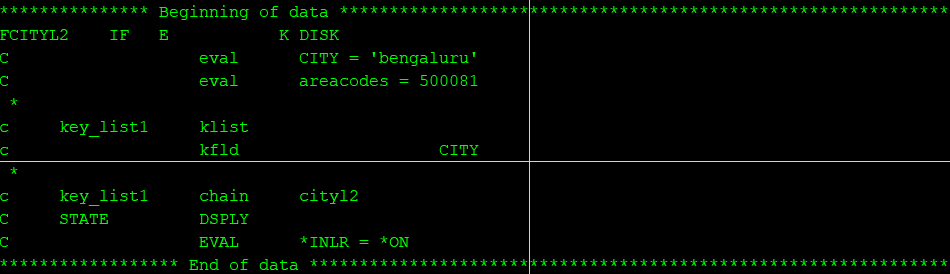
The KLIST operation is a declarative operation that gives a name to a list of KFLDs. This list can be used as a search argument to retrieve records from files that have a composite key.

This file has city and area codes as key. In this program chain is performed by using both keys.



If a file has more than one key. You can read that file based on initial keys also.

In below example CITY is the first key field of file, this file is being read only on basis of first key field .This is also called as accessing based on partial keys.



**PLIST (Identify a Parameter List):-**

·         It is used to define a list of parameters that will be used for calling a program.

Some keywords

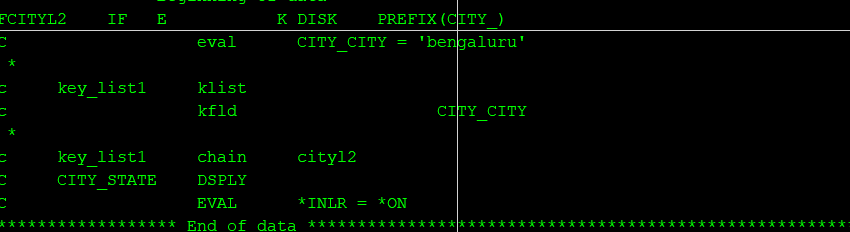
**F-SPEC keywords.**

**Rename:** Rename keyword is used to rename record format of a file in program .

When we use physical file and logical files referring to that physical file in same program, record format will be ignored while compilation due to same name and there will be compilation error. In this case use Rename keyword. Same renamed record format should be used while update/write.

Similar way flat files also can be define in this case file name and record format name will be same. Hence renaming is needed.

**PREFIX:** This will be used to prefix any string to file fields and using them with new names in program ,Basically this is renaming file fields. .



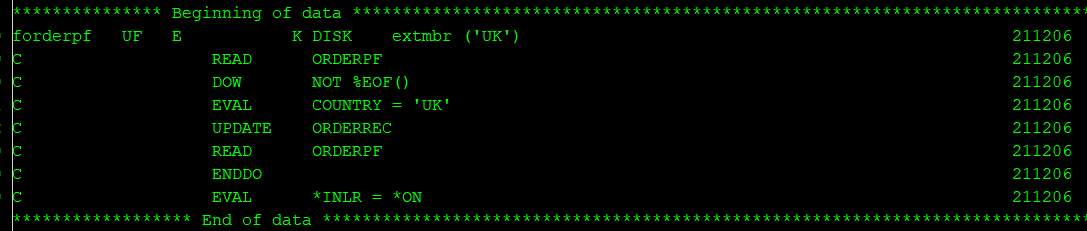
**USROPN**

This will be used to control opening/closing of file programmatically.

File should be open with OPEN opcode and once all processing for that file is done it should be closed.

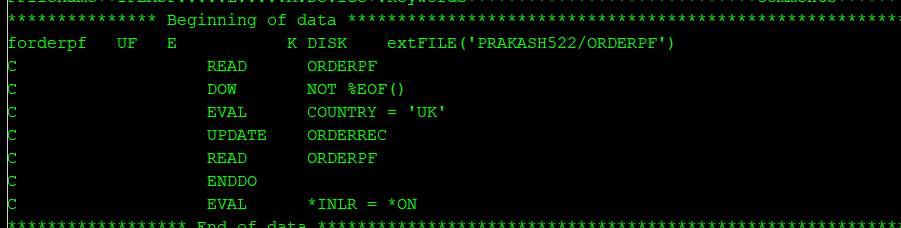
**EXTMBR**

This is used to access particular member of a physical file. By default first member of file will be accessed we can use this keyword to access other members if needed.



This will process UK member of ORDERPF

**EXTFILE :** This will specify which file in which library should be opened.



This will open ORDERPF from Prakash522 library.

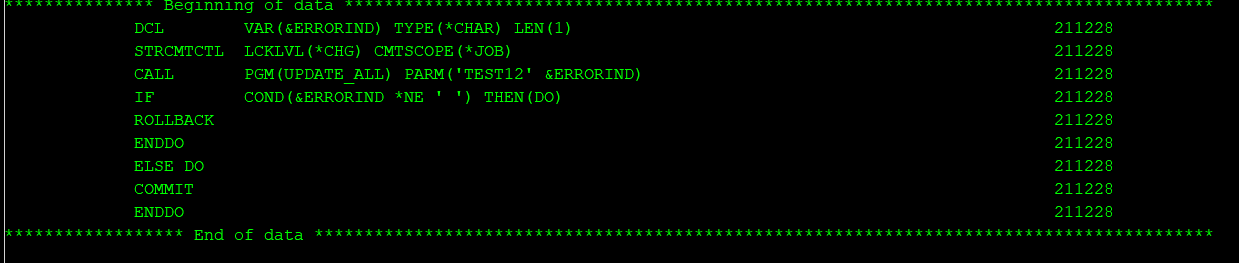
**COMIT keyword:** -

This keyword is used in F-SPEC for files which should be operated under commitment control.

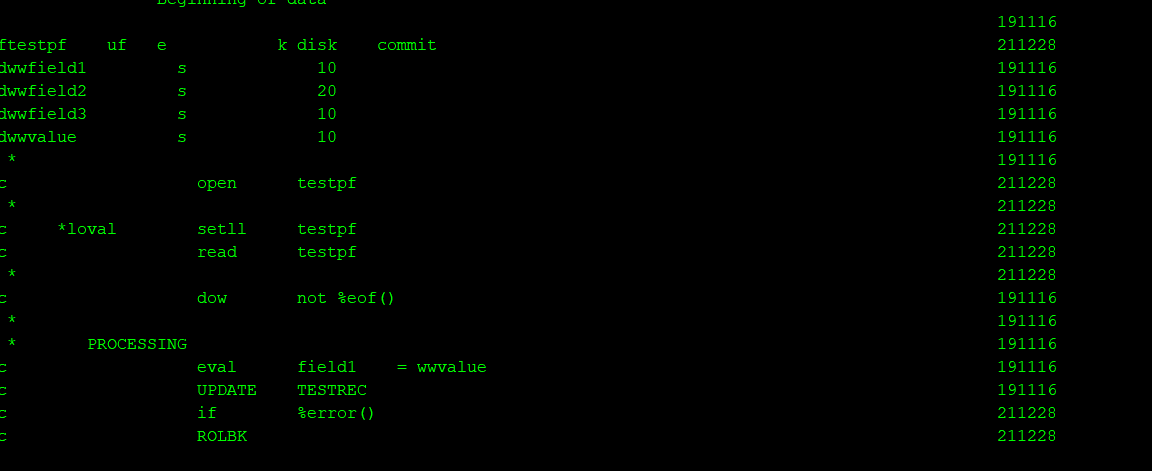
**Basic flow will be**

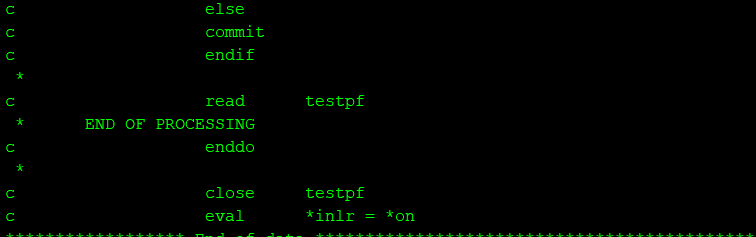
1. Drive CL will start commitment control with strcmtctl command. The scope can be defined at activation group level or job level.
2. Call rpg program which does file updates.
3. Declare files needing to be operated under commitment control with COMMIT keyword.
4. Handle error conditions for update operations on those files and rollback in case of errors.
5. Do ENDCMTCTL at the end of CL
6. In rpgle use ROLBK /commit opcodes

**CL commiting all changes at once.**



**RPGLE commitin changes record by record.**





**.**

**COMMIT and ROLBK opcodes are used.**

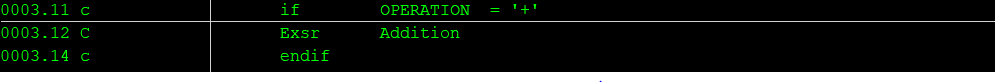
**D-SPEC keywords.**

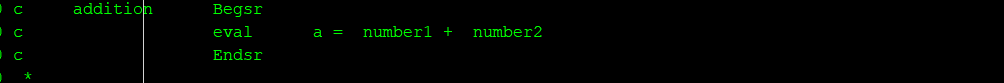
1. **INZ:** This is used to initialize variable, if no value is specified it will be initialized to default, if we need to initialize to some other value, we can define that in INZ KEYWORD
2. **CONST:** This is used to define constant

**Subroutines:** Sub steps of programs. Every program can be divided into multiple processes and one subroutine can be written for each process.

Subroutine logic is defined between BEGSR **(Begin subroutine)** and **ENDSR (End Subroutine)**

Subroutine can be executed by **EXSR (Execute subroutine)**





**LeaveSr**: Opcode is used to leave from subroutine, Pass control to next statement following subroutine call(Exsr)

\***INZSR** : This is initialization subroutine. This can be utilized only when program is not ended by return.

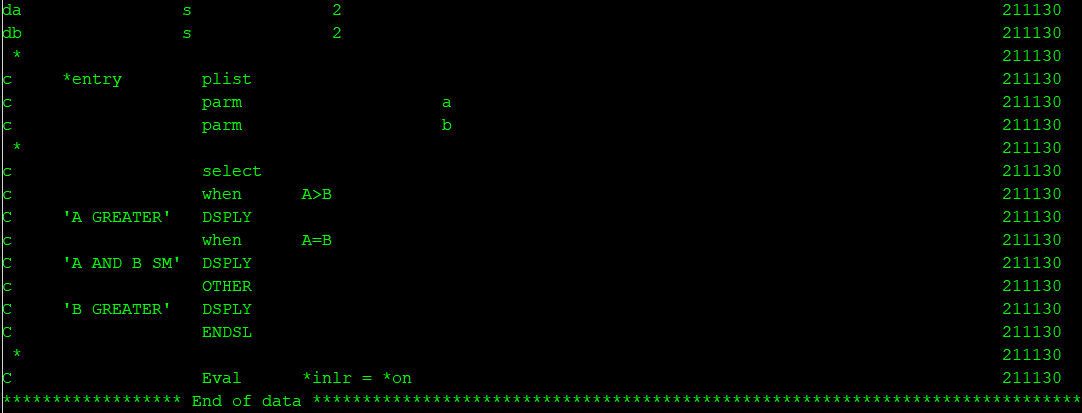
This will be first subroutine which will be executed in RPG program.

We no need to specially use EXSR opcode to execute this.

**Select when in rpgle :**This is also form of conditioning check in rpgle.

Where different conditions can be checked under WHEN.

Others will denote all conditions other than those specified in when



### **Alternative to SELECT is IF-ELSEIF**

|  |  |
| --- | --- |
| select ;  when (REGCDE = 'ER') ;  DESC = 'East region' ;  when (REGCDE = 'WR') ;  DESC = 'West region' ;  when (REGCDE = 'CA') ;  DESC = 'Canada' ;  other ;  DESC = 'Unknown' ;  endsl ; | if (REGCDE = 'ER') ;  DESC = 'East region' ;  elseif (REGCDE = 'WR') ;  DESC = 'West region' ;  elseif (REGCDE = 'CA') ;  DESC = 'Canada' ;  else ;  DESC = 'Unknown' ;  endif ; |

Multiple conditions in IF and select when

IF COND 1 OR CONDN 2

IF CONDN1 AND CONDN2

IF (CONDN1 OR CONDN2) AND COND(3)

### **Data areas in RPGLE: -**

**IN opcode:** - This opcode is used to retrieve the value from the data area.

For a data area to be retrieved by the IN operation, it must be specified in the result field of an [\*DTAARA DEFINE](https://www.ibm.com/docs/en/ssw_ibm_i_74/rzasd/zznamvr.htm#zznamvr) statement or using the [DTAARA keyword](https://www.ibm.com/docs/en/ssw_ibm_i_74/rzasd/ddtaara.htm#ddtaara) on the Definition specification

We can lock data area if needed while IN operation which will be further released by UNLOCK opcode or OUT opcode.

This is similar to RTVDTAARA in CLP

**Example**

1. **Data area definition using dtaara keyword in D-SPEC.**



In this case ADDRESS is variable which represents data area address.

If we have variable name defined and no value defined inside DTAARA keyword system assumes data area name same variable name.

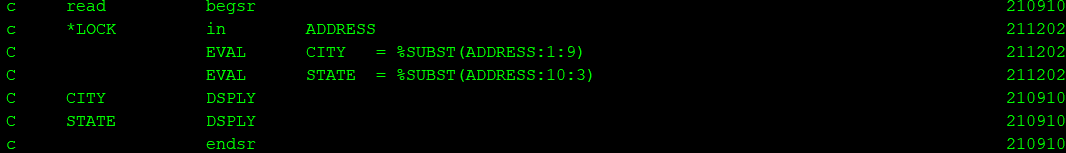
We can also refer external data area by separate name in program



**b) Defining data area using DEFINE opcode this was usually done in old rpg as we didn’t had D-SPEC.**



**c)IN OPCODE example.**



**Out opcode:** It is used to write into data area. This is similar to CHGDTAARA in CLP



**OUT opcode:** - This opcode is used to update the value from the data area.

This is similar to UPDDTAARA in CLP

**Data queues in AS400**

  It is used for making asynchronous communication between the two jobs (2 programs). If we are sending data from one job/program, to another using data queue using QSNDDTAQ API, but at that time the receiver job/program is not active/running, then the data will be there in the queue and whenever the receiver job becomes active, it retrieves the data using the QRCVDTAQ API.

Data queue can be created using **CRTDTAQ** command.

We can create data queue on local system or on remote system by making the data queue type as \*DDM. By default, it will be \*STD. In case of DDM data queue, we need to mention Remote data queue name (RMTDTAQ) and remote system name (RMTLOCNAME) also as parameter.

**QSNDDTAQ** : This API is used to send data from one program to DTAQUEUE

Parametes :

Data queue name

Data queue library

Length of data

Data

**QRCVDTAQ** : This API is used to receive data from DTAQUEUE in receiving program.

Parametes :

Data queue name

Data queue library

Length of data

Data

WAIT TIME

PGM  
DCL VAR(&DQNAME) TYPE(\*CHAR) LEN(10) VALUE('FILEINFO')  
DCL VAR(&DQLIB) TYPE(\*CHAR) LEN(10) VALUE('QGPL')  
DCL VAR(&DQSNDLEN) TYPE(\*DEC) LEN(5 0) VALUE(14)  
DCL VAR(&DQLEN) TYPE(\*DEC) LEN(5 0)  
DCL VAR(&DQSNDDATA) TYPE(\*CHAR) LEN(100)  
DCL VAR(&DQDATA) TYPE(\*CHAR) LEN(100)  
DCL VAR(&DQWAIT) TYPE(\*DEC) LEN(5 0) VALUE(0)  
CHGVAR VAR(&DQSNDDATA) VALUE('THIS IS A TEST')  
CALL QSNDDTAQ PARM(&DQNAME &DQLIB &DQSNDLEN &DQSNDDATA)  
CALL QRCVDTAQ PARM(&DQNAME &DQLIB &DQLEN &DQDATA &DQWAIT)  
SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) MSGDTA(&DQDATA)  
ENDPGM

**Arrays in AS400:-**

Array is collection of elements of same data type. Use arrays in RPGLE program in below format.

**Array name (element number**)

**DIM keyword**: This is D-SPEC keyword used to provide dimension of array.

Example: If we need to define an array with maximum 10 elements. It will defined with DIM(10) in d-spec function

**Types of arrays: -**

`1. **Runtime array** : This is loaded when your program is running, it is loaded at run time.

A(1) = 'A' ; EVAL A(1) = 'A'

A(2) = 'B' ; EVAL B(2) = 'B'

2. **Compile time array:** This is loaded once you compile program , it is denoted by keyword CTDATA in D-SPEC

You load at end. after last line Just give \*\* and then follow with elements of array

3. **Pre runtime array**: This is loaded before your run , it is done from some file. From file in D-SPEC.

WHENEVER u want to define file from where u want to load elements in an pre runtime array we need to define file designation as 'T'

In Normal file declaration we will define as 'F' fully procedural file

T : 'table file' basically used to load into pre run time array...

**Array keywords**:-

**PERRCD()** represents **the number of entries in one array record**. ...

**DIM :** Dim defines number of elements in array.

**Array Opcodes/functions:**

**sorta** : This will sort an array in ascending order

Array a = 5

6

1

2

7

sorta a

1

2

5

6

7

**LOOKUP**

Lookup OR U can use built in function %lookup to search element inside an array. This will give u position of an element in an array.

Consider above array example: -

i= %lookup(1:a), so i = 1

i = %lookup(5:a),so i = 3

**XFOOT:** This will give sum of all elements in array , it is specially used in numeric arrays.

**Data structures in AS400 :**

A single field can be divided into subfields in AS400. This functionality is achieved using data structures.

The ILE RPG compiler allows you to define an area in storage and the layout of the fields, called subfields, within the area. This area in storage is called a **data structure**.

**H-SPEC in rpgle**

It is used to define the **controlling entity** of the program e.g. date/ time format, default program name, program as PEP, to define the decimal precision etc.

It is also used to define the **compiler options** to be used while we are compiling the program.

e.g. default activation group, authority, compiler source generation options etc.

# **H SPEC Keywords**

***DEBUG (\*Yes)***

Use this keyword to indicate whether or not to perform DUMP operations in your RPG program. The default is \*NO, which means DUMP operations will *not* be performed. If you have an error handler that uses the DUMP op-code, you should specify DEBUG(\*YES) in your H-spec.

**OPTION(\*{NO}XREF \*{NO}GEN \*{NO}SECLVL \*{NO}SHOWCPY \*{NO}EXPDDS \*{NO}EXT \*{NO}SHOWSKP) \*{NO}SRCSTMT) \*{NO}DEBUGIO)**

*{NO}DEBUGIO:*  If you use the interactive source debugger to step through a program, you'll notice that the debugger will break many times on every I/O statement.  This is because a separate breakpoint is inserted for every field returned from the I/O buffer.  This is not a major problem, and very inconvenient.  Coding \*NODEBUGIO will break only once for each I/O statement.

**DATFMT (FMT)**

Specify this keyword to define the default format for date fields within the program.

Choose one of the following date formats:

    \*MDY (mm/dd/yy)

    \*DMY (dd/mm/yy)

    \*YMD (yy/mm/dd)

    \*ISO (yyyy-mm-dd)

    \*USA (mm/dd/yyyy)

    \*EUR (dd.mm.yyyy)

**e.g.    H  datfmt(\*MDY)**

**By default FMT will be \*ISO.**

***TIMFMT (FMT)***

Choose one of the following time formats:

  \*HMS (hh:mm:ss)

  \*ISO (hh.mm.ss)

  \*USA (hh:mm AM or hh:mm PM)

  \*EUR (hh.mm.ss)

  By default FMT will be \*ISO.

***o   ACTGRP (\*NEW | \*CALLER | 'activation-group-name')***

If ACTGRP(\*NEW) is specified, then the program is activated into a new activation group. If ACTGRP(\*CALLER) is specified, then the program is activated into the caller's activation group. If an activation-group-name is specified, then that name is used when this program is called.

H datedit(\*YMD) datfmt(\*ISO) option(\*SRCSTMT \*NODEBUGIO)

