**RPG**

(Link: <https://www.ibm.com/docs/en/i/7.4?topic=functions-operation-codes>)

1. **ACQ(Acquired)**

* Dùng để kết nối 1 thiết bị được chỉ định (device-name) cho tệp làm việc (WORKSTN file) được chỉ định (workstn-file)
* ACQ DEVICE(device-name) FILE(workstn-file)

1. **Add(add)**

* Thực hiện cộng 2 giá trị
* Thành phần 1 và Thành phần 2:
  + Thành phần 1 và Thành phần 2 là các giá trị sẽ được cộng lại với nhau.
  + Các giá trị này phải là số.
* Nếu Thành phần 1 được chỉ định:
  + Nếu Thành phần 1 được chỉ định, nó sẽ được cộng vào Thành phần 2.
  + Tổng của Thành phần 1 và Thành phần 2 sau đó được đặt vào trường kết quả.
* Nếu Thành phần 1 không được chỉ định:
  + Nếu Thành phần 1 không được chỉ định, nội dung của Thành phần 2 sẽ được cộng vào trường kết quả.
  + Tổng sau đó được đặt vào trường kết quả.

Ví dụ:

**C ADD 1 RECNO**

* Dòng này thực hiện phép cộng giữa giá trị 1 vào giá trị hiện tại của biến RECNO.
* Kết quả của phép cộng được lưu vào biến RECNO.

**C ADD EHWRK CURHRS**

* Dòng này thực hiện phép cộng giữa giá trị hiện tại của biến EHWRK và giá trị hiện tại của biến CURHRS.
* Kết quả của phép cộng được lưu vào biến CURHRS.

**C OVRTM ADD REGHRS TOTPAY**

* Dòng này thực hiện phép cộng giữa giá trị hiện tại của biến OVRTM và giá trị hiện tại của biến REGHRS.
* Kết quả của phép cộng được lưu vào biến TOTPAY.

1. **ADDDUR(Add Duration)**

* Thêm một khoảng thời gian cụ thể vào một ngày, thời gian, hoặc dấu thời gian, và nhận giá trị kết quả.

Ví dụ:

**C LOANDATE ADDDUR XX:\*YEARS DUEDATE**

**C ADDDUR YY:\*MONTHS DUEDATE**

**C ADDDUR ZZ:\*DAYS DUEDATE**

* Các dòng này tính toán ngày đáo hạn (**DUEDATE**) dựa trên ngày vay (**LOANDATE**) và một số năm (**XX**), tháng (**YY**), và ngày (**ZZ**) được chỉ định.
* **ADDDUR** được sử dụng để thêm số năm, tháng, và ngày vào **LOANDATE** để tạo ra **DUEDATE**.

**C ADDDUR 23:\*D DUEDATE**

Dòng này thêm 23 ngày vào **DUEDATE** từ giá trị hiện tại của nó.

**C ADDDUR 1234:\*MS timestamp**

Dòng này thêm 1234 microseconds vào **timestamp**.

**C T'00:00 am' ADDDUR 12:\*Hours answer**

**C ADDDUR 16:\*Minutes answer**

Dòng này thêm 12 giờ và 16 phút vào giờ 00:00 (nửa đêm) và lưu kết quả vào biến **answer**.

**C ADDDUR -30:\*D LOANDUE**

Dòng này trừ đi 30 ngày khỏi ngày đáo hạn (**LOANDUE**).

1. **ALLOC(Allocal Storage)**

* Lệnh **ALLOC** được sử dụng để cấp phát bộ nhớ trong bộ nhớ heap mặc định của hệ thống. Bộ nhớ được cấp phát có độ dài được chỉ định bởi Thành phần 2

D Ptr1 S \*

D Ptr2 S \*

Đây là cách khai báo hai con trỏ Ptr1 và Ptr2. Chúng được sử dụng để trỏ đến các vùng bộ nhớ được cấp phát sau này.

C ALLOC 7 Ptr1

Dòng này cấp phát 7 byte bộ nhớ trong heap mặc định của hệ thống và đặt con trỏ Ptr1 để trỏ tới vùng bộ nhớ này.

C ALLOC (E) 12345678 Ptr2

* Dòng này cấp phát một lượng lớn bộ nhớ có độ dài 12345678 byte.
* Trong trường hợp bộ nhớ không có sẵn (do hạn chế về tài nguyên hoặc các vấn đề khác), một lỗi sẽ xảy ra và **%ERROR** sẽ trả về '1'.
* Trạng thái của hàm **ALLOC** sẽ được đặt thành '00426', và **%STATUS** sẽ trả về '00426'.

1. **ANDxx(And)**

-Phép toán AND

**C ACODE IFEQ 'A'**

**C \*IN50 ANDEQ \*ON**

**C MOVE 'A' ACREC**

**C WRITE RCRSN**

Dòng này kiểm tra xem giá trị của biến ACODE có bằng 'A' không và chỉ số 50 có bật không. Nếu cả hai điều kiện này đều đúng, thì một giá trị 'A' được chuyển sang biến ACREC và sau đó ghi ra tập tin RCRSN.

**C ELSE**

**C ACODE IFEQ 'A'**

**C \*IN50 ANDEQ \*OFF**

**C ACREC ANDEQ 'D'**

**C MOVE 'A' ACREC**

**C ENDIF**

**C ENDIF**

Đoạn này là phần ELSE của điều kiện trước đó. Nó kiểm tra xem ACODE có bằng 'A' không, chỉ số 50 có tắt không và giá trị của biến ACREC có bằng 'D' không. Nếu tất cả các điều kiện đều đúng, thì giá trị 'A' được chuyển sang biến ACREC.

1. **BEGSR (Beginning of Subroutine)**

* Xác nhận điểm bắt đầu của 1 function trong chương trình

1. **BITOFF (Set Bits Off)**

* chỉ định các bit cần tắt đi (=0).

**C BITOFF '046' FieldG**

* Phép toán BITOFF được sử dụng để tắt các bit 0, 4, và 6 trong trường FieldG. Các bit 1, 2, 3, 5, và 7 không thay đổi.
* Trong trường hợp này, FieldG ban đầu có giá trị là 01001111 và sau khi thực hiện phép toán BITOFF, giá trị trở thành 01000101.

**C BITOFF BITNC FieldI**

* Phép toán BITOFF được sử dụng để tắt các bit 0, 2, 4, và 6 trong trường FieldI. Các bit 1, 3, 5, và 7 không thay đổi.
* Trong trường hợp này, FieldI ban đầu có giá trị là 11001110 và sau khi thực hiện phép toán BITOFF, giá trị trở thành 01000100.
* Chú ý rằng BITNC được sử dụng như một giá trị đại diện cho '4567', có mẫu bit là 00001111.

**C BITOFF HEXNC2 FieldK**

**C RETURN**

* Phép toán BITOFF được sử dụng để tắt các bit 4, 5, 6, và 7 trong trường FieldK. Các bit 0, 1, 2, và 3 không thay đổi.
* Trong trường hợp này, FieldK ban đầu có giá trị là 10000000 và sau khi thực hiện phép toán BITOFF, giá trị trở thành 00000000.
* Sau khi thực hiện phép toán BITOFF, chương trình trả về.

1. **BITON (Set Bits On)**

* chỉ định các bit cần được bật lên (=1)

**C BITON '3' FieldB**

Phép toán BITON được sử dụng để bật bit 3 trong trường FieldB. Các bit 0, 1, 2, 4, 5, 6, và 7 không thay đổi.

**C BITON FieldE FieldF**

Phép toán BITON được sử dụng để bật các bit trong FieldF tương ứng với các bit được bật trong FieldE. Các bit không được xác định trong FieldE sẽ không thay đổi.

**C BITON X'C1' FieldH**

* Phép toán BITON được sử dụng để bật các bit trong FieldH tương ứng với các bit được bật trong giá trị X'C1'. Các bit không được xác định trong X'C1' sẽ không thay đổi.
* Trong trường hợp này, giá trị của X'C1' tương ứng với giá trị '0', '1', và '7' trong HEXNC. Do đó, các bit 0, 1, và 7 trong FieldH sẽ được bật, và sau khi thực hiện phép toán BITON, giá trị của FieldH trở thành 11001001.

**C BITON HEXNC FieldJ**

* Phép toán BITON được sử dụng để bật các bit trong FieldJ tương ứng với các bit được bật trong HEXNC. Các bit không được xác định trong HEXNC sẽ không thay đổi.
* Trong trường hợp này, giá trị của HEXNC tương ứng với giá trị '4', '5', '6', và '7'. Do đó, các bit 4, 5, 6, và 7 trong FieldJ sẽ được bật, và sau khi thực hiện phép toán BITON, giá trị của FieldJ trở thành 10001111.

1. **CABxx (Compare and Branch)**

* Phép toán CABxx so sánh factor 1 với factor 2. Nếu điều kiện được chỉ định bởi xx là đúng, chương trình sẽ nhảy đến phép toán TAG hoặc ENDSR được liên kết với nhãn được chỉ định trong trường kết quả. Nếu không, chương trình sẽ tiếp tục với phép toán tiếp theo trong chuỗi

**C FieldA CABLE FieldB TAGX**

Nếu giá trị của FieldA nhỏ hơn hoặc bằng giá trị của FieldB, chương trình sẽ nhảy đến nhãn TAGX.

**C FieldA CABLE FieldB TAGX 16**

Nếu giá trị của FieldA nhỏ hơn hoặc bằng giá trị của FieldB và chỉ báo 16 đã tắt, chương trình sẽ nhảy đến nhãn TAGX.

**C FieldA CAB FieldB TAGX 1718**

Nếu giá trị của FieldA bằng giá trị của FieldB và chỉ báo 17 đã tắt và chỉ báo 18 đã bật, chương trình sẽ nhảy đến nhãn TAGX

1. **CALL (Call a Program)**

* Phép toán CALL chuyển quyền điều khiển đến chương trình được chỉ định

**C CALL 'PROGA'**

**C PARM FieldA**

**C PARM FieldB**

* Phép toán CALL được sử dụng để gọi chương trình có tên 'PROGA'.
* PARM được sử dụng để truyền các tham số cho chương trình gọi. Trong trường này, FieldA được truyền như một tham số cho chương trình 'PROGA'.
* Tiếp tục sử dụng PARM để truyền tham số tiếp theo, trong trường hợp này là FieldB.

1. **CALLB (Call a Bound Procedure)**

* Gọi 1 thủ tục ràng buộc

1. **CALLP (Call a Prototyped Procedure or Program)**

* Phép toán CALLP được sử dụng để gọi các thủ tục hoặc chương trình đã được prototype (lập mẫu) trước.

1. **CASxx (Conditionally Invoke Subroutine)**

* Phép toán CASxx cho phép bạn lựa chọn một tiểu chương trình để xử lý theo điều kiện
* **CASGT**: Chọn tiểu chương trình nếu factor 1 lớn hơn factor 2.
* **CASGE**: Chọn tiểu chương trình nếu factor 1 lớn hơn hoặc bằng factor 2.
* **CASEQ**: Chọn tiểu chương trình nếu factor 1 bằng factor 2.
* **CASNE**: Chọn tiểu chương trình nếu factor 1 không bằng factor 2.
* **CASLE**: Chọn tiểu chương trình nếu factor 1 nhỏ hơn hoặc bằng factor 2.
* **CASLT**: Chọn tiểu chương trình nếu factor 1 nhỏ hơn factor 2.

**C FieldA CASGE FieldB Subr01**

Nếu **FieldA** lớn hơn hoặc bằng **FieldB**, tiểu chương trình **Subr01** sẽ được thực thi.

1. **CAT (Concatenate Two Strings)**

* Phép toán CAT (Concatenate) nối chuỗi được chỉ định trong factor 2 vào cuối chuỗi được chỉ định trong factor 1 và đặt nó vào trường kết quả

**C MOVE 'MR.' NAME 3**

**C MOVE ' SMITH' FIRST 6**

**C NAME CAT FIRST RESULT 9**

**Leading Blanks in Factor 2**: Trong ví dụ này, chúng ta muốn nối chuỗi **'MR.'** từ **NAME** và chuỗi **' SMITH'** từ **FIRST** vào **RESULT**. **NAME** có chiều dài 3 ký tự và **FIRST** có chiều dài 6 ký tự. Sau phép toán CAT, **RESULT** sẽ chứa chuỗi **'MR.␢SMITH'**. Trong trường hợp này, các dấu cách trước **'SMITH'** không bị mất sau khi nối.

1. **CHAIN (Random Retrieval from a File)**

* lệnh **CHAIN** được sử dụng để **tìm kiếm một bản ghi trong một tập hợp dữ liệu theo một khóa xác định** và sau đó **đọc nội dung của bản ghi đó vào các biến hoặc cấu trúc dữ liệu đã được xác định**

**C CHAIN EmployeeID EmployeeFile**

**EmployeeID** là giá trị của khóa chính mà chúng ta muốn sử dụng để truy xuất bản ghi từ tập tin **EmployeeFile**.

1. **CHECK (Check Characters)**

* Phép toán CHECK được sử dụng để xác nhận rằng mỗi ký tự trong chuỗi cơ sở (factor 2) nằm trong các ký tự được chỉ định trong chuỗi so sánh (factor 1)

**D Field S 10A INZ('ABCD1234')**

**D ValidChars C 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'**

**C CHECK ValidChars Field InvalidCharPos**

* **Field** là một biến chuỗi có chiều dài 10, được khởi tạo với giá trị **'ABCD1234'**.
* **ValidChars** là một hằng số chuỗi chứa tất cả các ký tự chữ cái từ A đến Z.
* Phép toán CHECK sẽ kiểm tra từng ký tự trong biến **Field** và so sánh chúng với các ký tự trong chuỗi **ValidChars**.
* Nếu một ký tự không nằm trong chuỗi **ValidChars**, chỉ mục của ký tự không hợp lệ sẽ được đặt trong biến **InvalidCharPos**.

1. **CHECKR (Check Reverse)**

* Kiểm tra bắt đầu từ ký tự bên phải

**D BaseString S 10A INZ('ABCD1234')**

**D ValidChars C 'ABCDEFGHIJKLMNOPQRSTUVWXYZ**

**C CHECKR ValidChars BaseString InvalidCharPos**

* **BaseString** là một biến chuỗi có chiều dài 10, được khởi tạo với giá trị **'ABCD1234'**.
* **ValidChars** là một hằng số chuỗi chứa tất cả các ký tự chữ cái từ A đến Z.
* Phép toán CHECKR sẽ kiểm tra từng ký tự trong biến **BaseString** và so sánh chúng với các ký tự trong chuỗi **ValidChars**.
* Kiểm tra bắt đầu từ ký tự bên phải nhất của **BaseString** và tiếp tục với mỗi ký tự từ phải sang trái.
* Nếu một ký tự không nằm trong chuỗi **ValidChars**, chỉ mục của ký tự không hợp lệ sẽ được đặt trong biến **InvalidCharPos**.

1. **CLEAR (Clear)**

* Phép toán CLEAR được sử dụng để thiết lập các phần tử trong một cấu trúc
* Đối với trường số (numeric): Giá trị mặc định là 0.
* Đối với trường ký tự (character): Giá trị mặc định là khoảng trắng (' ').
* Đối với trường đồ họa (graphic): Giá trị mặc định là khoảng trắng (' ').
* Đối với trường UCS-2: Giá trị mặc định là khoảng trắng (' ').
* Đối với trường chỉ số (indicator): Giá trị mặc định là \*OFF.
* Đối với con trỏ (pointer): Giá trị mặc định là \*NULL.
* Đối với ngày/giờ/biểu đồ thời gian (date/time/timestamp): Giá trị mặc định là ngày/giờ/biểu đồ thời gian không xác định, thường là 0001-01-01 00:00:00.

**DCL VAR1 CHAR(10) INZ('Hello')**

**/\* Khai báo một biến ký tự và khởi tạo giá trị \*/**

**DCL VAR2 PACKED(5:2) INZ(123.45)**

**/\* Khai báo một biến số và khởi tạo giá trị \*/**

**CLEAR VAR1 /**

**\* Thiết lập biến ký tự VAR1 về giá trị mặc định (khoảng trắng) \*/**

**CLEAR VAR2**

**/\* Thiết lập biến số VAR2 về giá trị mặc định (0) \*/**

1. **CLOSE (Close Files)**

* Sử dụng để đóng một hoặc nhiều tập tin hoặc thiết bị và ngắt kết nối chúng khỏi module.

1. **COMMIT (Commit)**

* Xác nhận thay đổi và giải phóng khoá bản ghi

1. **COMP (Compare)**

* Phép toán COMP được sử dụng để so sánh factor 1 với factor 2 trong ngôn ngữ lập trình RPG.

**C NUM1 COMP 100**

**/\* So sánh giá trị của NUM1 với literal 100 \*/**

**C STATUS COMP 'OK'**

**/\* So sánh giá trị của STATUS với chuỗi 'OK' \*/**

**C ARRAY(3) COMP 50**

**/\* So sánh giá trị của phần tử thứ 3 trong mảng ARRAY với literal 50 \*/**

1. **DATA-GEN (Generate a Document from a Variable)**

* Tạo ra 1 cấu trúc mới với kiểu dữ liệu được chỉ định

**DATA-GEN(BienDuLieu: ChuongTrinhTaoRa);**

1. **DATA-INTO (Parse a Document into a Variable)**

* Phép toán DATA-INTO trong RPG được sử dụng để nhập dữ liệu từ một tài liệu có cấu trúc vào một biến RPG.

1. **DEALLOC (Free Storage)**

* Phép toán DEALLOC giải phóng một phân bổ trước đó của bộ nhớ heap

1. **DEFINE (Field Definition)**

* Định nghĩa một trường dữ liệu dựa trên các thuộc tính (độ dài và số thập phân) của một trường khác (\*LIKE DEFINE).
* Định nghĩa một trường dữ liệu như là một khu vực dữ liệu (\*DTAARA DEFINE).

**D DataArea S 20A DTAARA('MYDATA')**

Trong ví dụ này, trường dữ liệu DataArea được định nghĩa như một khu vực dữ liệu có tên 'MYDATA'.

1. **DELETE (Delete Record**)

* Phép toán DELETE xóa một bản ghi khỏi một tệp cơ sở dữ liệu

1. **DIV (Divide)**

* sử dụng để thực hiện phép chia

**C DIV TotalHours NumDays**

Trong ví dụ này, TotalHours được chia cho NumDays, và kết quả được đặt vào TotalHours, biểu thị số giờ chia đều vào số ngày.

1. **DO (Do) => for**

* sử dụng để bắt đầu một nhóm các phép toán và chỉ ra số lần nhóm đó sẽ được thực hiện

**D Counter S 2S 0 INZ(1)**

**C DO 10 // Bắt đầu vòng lặp từ 1 đến 10**

**C ENDDO // Kết thúc vòng lặp DO**

1. **DOU (Do Until) => while**

* Các phép toán được điều khiển bởi phép toán DOU được thực hiện cho đến khi biểu thức trong indicator-expression là đúng.

1. **DOUxx (Do Until)**
2. **DOW (Do While)** => while or do.. while

* Các phép toán được điều khiển bởi phép toán DOW được thực hiện trong khi biểu thức trong indicator-expression là đúng (TRUE)

1. **DOWxx (Do While)**
2. **DSPLY (Display Message)**

* Hiển thị data ra màn hình
* **DSPLY FLD1 ' ' FLD2**

FLD1 và FLD2 là hai trường dữ liệu cần hiển thị.

Ký tự khoảng trắng ' ' được sử dụng để tạo ra một khoảng trống giữa FLD1 và FLD2 khi hiển thị trên màn hìn

1. **DUMP (Program Dump)**
2. **ELSE (Else)**

**C IF Condition1**

**C MOVE Value1 Result**

**C ELSE**

**C MOVE Value2 Result**

**C ENDIF**

Trong ví dụ này, nếu **Condition1** được đánh giá là đúng, thì giá trị của **Value1** sẽ được di chuyển vào **Result**. Nếu **Condition1** không đúng (tức là ELSE), thì giá trị của **Value2** sẽ được di chuyển vào **Result**.

1. **ELSEIF (Else If)**
2. **ENDyy (End a Structured Group)**

* Phép toán **ENDyy** đánh dấu sự kết thúc của một cấu trúc điều khiển như một vòng lặp, khối điều kiện hoặc khối xử lý lỗi

1. **ENDSR (End of Subroutine) =>{}**
2. **EVAL (Evaluate expression)**

* Phép toán EVAL thực hiện một câu lệnh gán

**C EVAL TotalAmount = Quantity \* UnitPrice**

Trong ví dụ này, biểu thức **Quantity \* UnitPrice** được đánh giá và kết quả được gán vào trường **TotalAmount**.

1. **EVALR (Evaluate expression, right adjust)**

* **C EVALR LastName = %SUBST(FullName: 1: 20)**

Trong ví dụ này, biểu thức **%SUBST(FullName: 1: 20)** được đánh giá và kết quả được gán vào trường **LastName**. Kết quả sẽ được căn phải và thêm các khoảng trắng vào bên trái nếu cần. Điều này giả định rằng **FullName** và **LastName** là các trường kiểu ký tự có chiều dài cố định.

1. **EVAL-CORR (Assign corresponding subfields)**
2. **EXCEPT (Calculation Time Output)**

* Xử lý các ngoại lệ của chương trình

1. **EXFMT (Write/Then Read Format**)=> hiển thị dữ liệu lên màn hình

* Phép toán EXFMT kết hợp việc ghi (WRITE) và đọc (READ) vào cùng một bản ghi định dạng.

**C WRITE RECORD1**

**C EXFMT RECORD2**

Trong ví dụ này, đầu tiên chúng ta ghi (WRITE) dữ liệu vào RECORD1, sau đó sử dụng phép toán EXFMT để ghi và đọc dữ liệu từ RECORD2.

1. **EXSR (Invoke Subroutine)**
2. **EXTRCT (Extract Date/Time/Timestamp)**

* Phép toán EXTRCT sẽ trả về một trong các phần sau:
* Phần năm, tháng hoặc ngày của một trường ngày hoặc timestamp
* Phần giờ, phút hoặc giây của một trường thời gian hoặc timestamp
* Phần micro giây của trường timestamp

1. **FEOD (Force End of Data)**

* Phép FEOD trong RPG IV thông báo về sự kết thúc logic của dữ liệu cho một tập tin chính, phụ hoặc toàn bộ quy trình.

1. **FOR (For)**

* Vòng lặp for

1. **FOR-EACH (For Each)**
2. **FORCE (Force a Certain File to Be Read Next Cycle)**

* Phép FORCE cho phép lựa chọn tập tin mà bản ghi tiếp theo sẽ được đọc từ đó. Phép này chỉ có thể được sử dụng cho các tập tin chính hoặc tập tin phụ.

**C FORCE MasterFile**

Trong ví dụ này, **MasterFile** là tên của tập tin từ đó bản ghi tiếp theo sẽ được đọc.

1. **GOTO (Go To)**

* cho phép chương trình chuyển hướng thực thi đến một phép toán tính toán cụ thể trong chương trình

**C 10 GOTO RTN1**

**C 15 GOTO RTN2**

* Nếu chỉ số 10 được bật, chương trình sẽ nhảy tới nhãn TAG **RTN1**.
* Nếu chỉ số 15 được bật, chương trình sẽ nhảy tới nhãn TAG **RTN2**.

1. **IF (If)**

* Kiểm tra điều kiện

1. **IFxx (If)**
2. **IN (Retrieve a Data Area)**

* Phép toán IN thường được sử dụng để lấy giá trị được lưu trữ trong một khu vực dữ liệu (data area)

**D DataArea S 50 INZ**

**D DataAreaName S 10A INZ('MYDATAARA')**

**C IN DataAreaName**

**C DataArea DSPLY**

* **DataArea** được định nghĩa là một trường có độ dài 50 ký tự để lưu trữ giá trị của khu vực dữ liệu.
* **DataAreaName** được định nghĩa là một trường có độ dài 10 ký tự được khởi tạo với tên của khu vực dữ liệu cần lấy.
* Phép toán **IN** lấy giá trị được lưu trữ trong khu vực dữ liệu được chỉ định bởi **DataAreaName** và đặt nó vào **DataArea**.
* Phép toán **DSPLY** sau đó hiển thị giá trị của **DataArea** cho người dùng.

1. **ITER (Iterate) => continute**

* Phép toán ITER trong RPG được sử dụng để chuyển quyền điều khiển từ bên trong một nhóm DO hoặc FOR đến câu lệnh ENDDO hoặc ENDFOR của nhóm đó

1. **KFLD (Define Parts of a Key)=> biến**

* Phép toán KFLD trong RPG chỉ ra rằng một trường dữ liệu là một phần của đối số tìm kiếm được xác định bằng tên KLIST (Danh sách Khóa)

1. **KLIST (Define a Composite Key)**

* Phép toán KLIST trong RPG dùng để đặt tên cho một danh sách các trường (KFLD) được sử dụng như là đối số **tìm kiếm để lấy các bản ghi từ các tập tin có một khóa composite.(truy vấn dữ liệu)**

// Định nghĩa một danh sách trường dữ liệu (Key List) cho tập tin

**KLIST KListName;**

**KFLD Field1;**

**KFLD Field2;**

1. **LEAVE (Leave a Do/For Group)** **=> break**

* Phép toán LEAVE trong RPG chuyển quyền kiểm soát từ bên trong một nhóm DO hoặc FOR đến câu lệnh sau câu lệnh ENDDO hoặc ENDFOR.

1. **LEAVESR (Leave a Subroutine)=> return**

* Thoát khỏi 1 subroutine

1. **LOOKUP (Look Up a Table or Array Element)**

* lệnh LOOKUP được sử dụng để tìm kiếm một phần tử cụ thể trong một mảng hoặc bảng.

1. **DIM(number):** kích thước của mảng
2. **MHHZO (Move High to High Zone)**

**C MOVE '123C' NGUỒN 5**

**C MOVE '1234' ĐÍCH 5**

**C NGUỒN MHHZO ĐÍCH**

Trong ví dụ này, phần vùng zone 'C' từ vị trí trái nhất của trường NGUỒN được di chuyển sang vị trí trái nhất của trường ĐÍCH. Sau phép toán, trường ĐÍCH sẽ chứa '123C4'.

1. **MHLZO (Move High to Low Zone)**

**C MOVE '123C' SOURCE 5**

**C MOVE '1234' DESTINATION 5**

**C SOURCE MHLZO DESTINATION**

Trong ví dụ này, phần kỹ hiệu 'C' từ kỹ hiệu trái nhất của trường SOURCE được di chuyển đến kỹ hiệu phải nhất của trường DESTINATION. Sau phép toán, trường DESTINATION sẽ chứa '123C4'.

1. **MLHZO (Move Low to High Zone)**

**C MOVE '12A3' SOURCE 5**

**C MOVE '1234' DESTINATION 5**

**C SOURCE MLHZO DESTINATION**

Trong ví dụ này, phần kỹ hiệu 'A' từ kỹ hiệu phải nhất của trường SOURCE được di chuyển đến kỹ hiệu trái nhất của trường DESTINATION. Sau phép toán, trường DESTINATION sẽ chứa 'A234'.

1. **MLLZO (Move Low to Low Zone)**

**C MOVE '12A3' SOURCE 5**

**C MOVE '1234' DESTINATION 5**

**C SOURCE MLLZO DESTINATION**

Trong ví dụ này, phần ký hiệu 'A' từ ký hiệu phải nhất của trường SOURCE được di chuyển đến ký hiệu phải nhất của trường DESTINATION. Sau phép toán, trường DESTINATION sẽ chứa '12A4'.

1. **MONITOR (Begin a Monitor Group)**

* thực hiện xử lý lỗi có điều kiện dựa trên thông điệp ngoại lệ hoặc mã trạng thái

1. **MOVE (Move)**

* cho phép di chuyển các ký tự từ factor 2 vào result field. Việc di chuyển bắt đầu từ ký tự ở vị trí bên phải nhất của factor 2.

**C MOVE FIELD2 RESULT**

Trong ví dụ này, dữ liệu từ trường FIELD2 sẽ được di chuyển vào trường RESULT. Điều này có nghĩa là giá trị của trường FIELD2 sẽ được sao chép vào trường RESULT.

1. **MOVEA (Move Array)**

* MOVEA được sử dụng để chuyển đổi các giá trị ký tự, đồ họa, UCS-2 hoặc số từ factor 2 vào trường kết quả

**C MOVEA Array Field**

Chuyển giá trị từ mảng array vào filed

1. **MOVEL (Move Left)**

* Phép toán MOVEL chuyển các ký tự từ factor 2 vào trường kết quả. Quá trình di chuyển bắt đầu từ ký tự bên trái nhất trong factor 2. Khi dữ liệu được chuyển đến một trường số, dấu (+ hoặc -) của trường kết quả được giữ nguyên trừ khi factor 2 dài hơn hoặc bằng trường kết quả. Trong trường hợp này, dấu của factor 2 được sử dụng làm dấu của trường kết quả.

**D SourceField S 10 INZ('Hello')**

**D TargetField S 10**

**C MOVEL SourceField TargetField**

Trong ví dụ này, giá trị 'Hello' được di chuyển từ trường SourceField sang trường TargetField bắt đầu từ vị trí bên trái nhất. Khi hoàn thành, giá trị của TargetField sẽ là 'Hello'.

1. **MULT (Multiply)**

* **C MULT AMOUNT RATE TOTAL**

Trong ví dụ này, giá trị của trường AMOUNT được nhân với giá trị của trường RATE và kết quả được đặt vào trường TOTAL.

1. **MVR (Move Remainder)**

* Phép toán MVR trong RPG được sử dụng để di chuyển phần dư từ phép chia trước đó vào một trường riêng được đặt tên trong trường kết quả.
* Factor 1 và factor 2 phải là trống.

**C DIV NUM1 NUM2**

**C MVR REMAINDER**

Trong ví dụ này, phần dư của phép chia NUM1 cho NUM2 được đặt vào trường REMAINDER sau phép chia.

1. **NEXT (Next)**
2. **OCCUR (Set/Get Occurrence of a Data Structure**) **=> array với n kí tự**
3. **ON-ERROR (On Error**)=> status
4. **ON-EXCP (On Exception**) => exception
5. **ON-EXIT (On Exit**)

-được sử dụng để xác định các lệnh sẽ được thực thi mỗi khi thủ tục kết thúc

1. **OPEN (Open File for Processing)**
2. **ORxx (Or)**

**D Count S 3S 0**

**C MOVE 0 Count**

**C Count ADD 1 ORLT 10**

**C END**

Trong ví dụ này, chúng ta khởi tạo biến Count với giá trị 0. Tiếp theo, chúng ta thực hiện phép cộng 1 vào Count và kiểm tra xem Count có nhỏ hơn 10 không. Nếu điều kiện này không được đáp ứng, quá trình kết thúc. Tuy nhiên, với phép toán ORLT, nếu Count nhỏ hơn 10, quá trình vẫn tiếp tục.

1. **OTHER (Otherwise Select) => default**

* Lệnh OTHER được sử dụng trong một nhóm SELECT để xác định một chuỗi các phép toán sẽ được thực hiện khi không có điều kiện WHEN nào trong nhóm SELECT được thỏa mãn. Điều này cho phép xử lý các trường hợp không nằm trong danh sách các điều kiện cụ thể được xác định bởi các phép toán WHEN.

D Option S 1A

**C SELECT => switch**

**C WHEN Option = 'A' => case**

**C // Xử lý khi Option là 'A'**

**C WHEN Option = 'B'**

**C // Xử lý khi Option là 'B'**

**C OTHER => default**

**C // Xử lý khi Option không phù hợp với bất kỳ điều kiện nào**

**C ENDSL**

Trong ví dụ này, chúng ta có một biến Option để chỉ ra lựa chọn. Trong nhóm SELECT, chúng ta xác định các điều kiện WHEN cho các trường hợp Option có thể là 'A' hoặc 'B'. Tuy nhiên, nếu Option không phù hợp với bất kỳ điều kiện WHEN nào, phép toán OTHER sẽ được sử dụng để xử lý các trường hợp này.

1. **OUT (Write a Data Area)**

* Lệnh OUT thường được sử dụng để ghi dữ liệu vào vùng dữ liệu, cập nhật các giá trị hoặc thực hiện các thao tác khác liên quan đến dữ liệu

**// Cập nhật giá trị của vùng dữ liệu sử dụng lệnh OUT**

**C OUT DataArea**

1. **PARM (Identify Parameters)** => biến

* sử dụng để định nghĩa các tham số trong danh sách tham số (PLIST) để truyền dữ liệu cho các chương trình con hoặc thủ tục được gọi

// Định nghĩa một danh sách tham số (Parameter List) cho chương trình con

**PLIST MySubroutine;**

**PARM Param1;**

**PARM Param2;**

1. **PLIST (Identify a Parameter List)**

* Định nghĩa một danh sách tham số (Parameter List) cho **chương trình con**

**PLIST MySubroutine;**

**PARM Param1;**

**PARM Param2;**

1. **POST (Post)**
2. **READ (Read a Record)**

* lệnh **READ** được sử dụng để đọc **bản ghi hiện tại** từ một tệp thực thể

**C READ FILEA**

=>đọc bản ghi hiện tại từ tệp **FILEA**

1. **READC (Read Next Changed Record)**

* Lệnh **READC** được sử dụng để đọc bản ghi **thay đổi tiếp theo** trong một subfile được mô tả bên ngoài
* **C READC (E) SFCUSR**

1. **READE (Read Equal Key)**

* Lệnh READE được sử dụng để lấy bản ghi tuần tự tiếp theo từ một tệp full procedural **nếu khóa của bản ghi khớp với đối số tìm kiếm**. Nếu khóa của bản ghi không khớp với đối số tìm kiếm, một điều kiện End Of File (EOF) xảy ra, và bản ghi không được trả về cho chương trình. Điều kiện EOF cũng xảy ra khi cuối file được đạt đến.
* **C KEYFLD READE FILEA**  
  Trong ví dụ trên, lệnh READE được sử dụng để đọc bản ghi tiếp theo từ tệp FILEA, với điều kiện là giá trị của trường KEYFLD phải khớp với đối số tìm kiếm. Nếu bản ghi có khóa tương ứng, nó sẽ được đọc vào chương trình cho xử lý tiếp theo. Nếu không, điều kiện End Of File (EOF) xảy ra và không có bản ghi nào được trả về.

1. **READP (Read Prior Record)**

* lệnh READP được sử dụng **để đọc bản ghi trước đó (nhỏ hơn)** từ một tập tin procedural đầy đủ
* **C READP FILEA**

1. **READPE (Read Prior Equal)**

* lệnh đọc bản ghi trước đó từ một tập tin được mô tả đầy đủ. Nó tương tự như lệnh READP, tuy nhiên, READPE chỉ **đọc bản ghi trước đó nếu khóa của bản ghi phù hợp với đối số tìm kiếm**
* **C FieldB READPE FileB Ds1 99**  
  Trong ví dụ này, lệnh READPE được sử dụng để đọc bản ghi trước đó từ tệp FileB. Bản ghi được đọc sẽ được đặt vào cấu trúc dữ liệu Ds1. Nếu không có bản ghi nào được tìm thấy có khóa phù hợp, cờ chỉ thị 99 sẽ được thiết lập.
* **C FieldC READPE RecA 8899**  
  Trong ví dụ này, lệnh READPE được sử dụng để đọc bản ghi trước đó từ định dạng bản ghi RecA. Nếu thao tác đọc thành công nhưng không có bản ghi nào được tìm thấy với khóa phù hợp, cờ chỉ thị 88 sẽ được thiết lập. Nếu không có bản ghi nào được tìm thấy, cờ chỉ thị 99 sẽ được thiết lập.

1. **REALLOC (Reallocate Storage with New Length)**

* Thay đổi độ dài của vùng nhớ

1. **REL (Release)**
2. **RESET (Reset)**
3. **RETURN (Return to Caller)**
4. **ROLBK (Roll Back)**
5. **SCAN (Scan String)**

* **C 'ABC' SCAN 'XCABCD' RESULT 90**

Lệnh SCAN tìm chuỗi con 'ABC' bắt đầu từ vị trí thứ 3 trong factor 2 ('XCABCD'). Kết quả trả về là 3, và chỉ số 90 được kích hoạt vì chuỗi được tìm thấy. Do không có chỉ định vị trí bắt đầu, vị trí mặc định là 1 sẽ được sử dụng.

1. **SELECT (Begin a Select Group)=> switch**
2. **SETGT (Set Greater Than)**

* Lệnh SETGT dùng để định vị tập tin ở bản ghi kế tiếp có khóa hoặc số bản ghi tương đối **lớn hơn** khóa hoặc số bản ghi tương đối được chỉ định.

**//Đọc bản ghi tiếp theo so với vị trí setgt**

**C**

**C KEY SETGT FILEA**

**C READ FILEA 64**

**C**

**//Đọc bản ghi trùng với setgt**

**C KEY SETGT FILEB**

**C READP FILEB 64**

Dòng mã trên sẽ định vị tập tin FILEB tại bản ghi tiếp theo có khóa lớn hơn khóa trong trường KEY. Sau đó, nó đọc bản ghi này bằng lệnh READP. Nếu không tìm thấy bản ghi nào có khóa lớn hơn, chỉ báo 64 sẽ được đặt lên.

* Nếu dùng biến **LOVAL**: *để đọc bản ghi* ***đầu tiên*** *của tập tin theo thứ tự tăng*
* Nếu dùng biến **HIVAL**: đọc bản ghi **cuối cùng** của tập tin theo thứ tự tăng dần

1. **SETLL (Set Lower Limit)**

* Lệnh SETLL được sử dụng để định vị một tập tin tại bản ghi tiếp theo có khóa hoặc số bản ghi tương đối **lớn hơn hoặc bằng** đối số tìm kiếm được chỉ định

**C ORDER SETLL ORDFIL**

1. **SETOFF (Set Indicator Off)**

* Lệnh SETOFF được sử dụng để **tắt** bất kỳ chỉ báo nào được chỉ định
* **C \*IN01 SETOFF**

Trong ví dụ này, lệnh SETOFF được sử dụng để tắt chỉ báo \*IN01. Sau khi thực thi lệnh này, chỉ báo \*IN01 sẽ được đặt về giá trị '0'.

1. **SETON (Set Indicator On)**

* **C \*IN02 SETON**

Trong ví dụ này, lệnh SETON được sử dụng để bật chỉ báo \*IN02. Sau khi thực thi lệnh này, chỉ báo \*IN02 sẽ được đặt về giá trị '1'.

1. **SHTDN (Shut Down)**
2. **SND-MSG (Send a Message to the Joblog)**
3. **SORTA (Sort an Array)**

* Lệnh SORTA trong RPG để sắp xếp một mảng dữ liệu theo một hoặc nhiều trường con
* **C SORTA ARROs**

1. **SQRT (Square Root)**

* Phép toán SQRT được sử dụng để tính căn bậc hai

**D Number S 5P 2**

**D SquareRoot S 5P 2**

**C MOVEL 25.00 Number**

**C SQRT Number SquareRoot**

* Chúng ta có một biến Number với kiểu dữ liệu số có 2 chữ số thập phân.
* Chúng ta cũng có một biến SquareRoot để lưu kết quả của phép tính căn bậc hai.
* Trong lệnh SQRT, chúng ta truyền giá trị của Number là 25.00, sau đó kết quả căn bậc hai của 25.00 được gán vào biến SquareRoot. Kết quả là 5.00, vì căn bậc hai của 25 là 5.

1. **SUB (Subtract): phép trừ**

**D So1 S 5P 2**

**D So2 S 5P 2**

**D KetQua S 5P 2**

**C MOVEL 10.50 So1**

**C MOVEL 5.25 So2**

**C SUB So1 So2 KetQua**

* Chúng ta có hai biến **So1** và **So2**, đều được định nghĩa là các trường số học có 2 chữ số thập phân.
* Chúng ta cũng có một biến **KetQua** để lưu trữ kết quả của phép trừ.
* Phép trừ **SUB** trừ giá trị của **So2** từ **So1**, và kết quả được lưu trữ trong biến **KetQua**.
* Sau phép tính này, **KetQua** sẽ chứa giá trị **5.25**, vì **10.50 - 5.25 = 5.25**.

1. **SUBDUR (Subtract Duration):** sử dụng để trừ đi một khoảng thời gian từ một khoảng thời gian khác
2. **SUBST (Substring)**

* sử dụng để trích xuất một phần của chuỗi từ một vị trí bắt đầu cụ thể và với độ dài xác định, sau đó đặt phần này vào trường kết quả

**D SourceString S 20A INZ('Hello, World!')**

**D Substring S 10A**

**C SUBST Substring : 1 : 5 : SourceString**

Trong ví dụ này, phép toán SUBST sẽ trích xuất một phần của chuỗi **SourceString** từ vị trí bắt đầu là 1 và với độ dài là 5 ký tự, sau đó đặt kết quả vào **Substring**. Kết quả sẽ là chuỗi "Hello".

1. **TAG (Tag**)

* Phép toán TAG trong RPG IV được sử dụng để gán một nhãn (label) định danh cho điểm đến của một phép toán GOTO (Di chuyển đến) hoặc CABxx (So sánh và nhảy)

1. **TEST (Test Date/Time/Timestamp)**

* Phép toán TEST cho phép người dùng kiểm tra tính hợp lệ của các trường ngày, giờ hoặc dấu thời gian trước khi sử dụng chúng

1. **TESTB (Test Bit)**

* Phép toán TESTB so sánh các bit được xác định trong yếu tố 2 với các bit tương ứng trong trường được đặt tên là trường kết quả

**C TESTB '3' FieldF 16 17**

* Indicator 16 được đặt bật vì bit 3 bị tắt (0) trong FieldF.
* Indicator 17 được đặt tắt.

**C TESTB '36' FieldF 161718**

Indicator 16 được đặt bật vì cả hai bit 3 và 6 đều bị tắt (0) trong FieldF. Các chỉ báo 17 và 18 được đặt tắt.

1. **TESTN (Test Numeric)**

* Phép toán TESTN được sử dụng để kiểm tra một trường kết quả chứa các chữ số zoned decimal và dấu cách

**C TESTN FieldA 21**

Chỉ báo 21 được bật vì FieldA chứa tất cả các ký tự số.

1. **TESTZ (Test Zone)**
2. **TIME (Retrieve Time and Date)**

* Phép toán TIME được sử dụng để lấy thời gian hệ thống hiện tại trong quá trình thực thi chương trình. Thời gian này được dựa trên hệ thống đồng hồ 24 giờ.

1. **UNLOCK (Unlock a Data Area or Release a Record)**
2. **UPDATE (Modify Existing Record)**

- Phép toán UPDATE được sử dụng để sửa đổi bản ghi

1. **WHEN (When True Then Select)** **=> case**

* Phép toán WHEN trong RPG IV được sử dụng để điều khiển việc xử lý các dòng trong một phép toán SELECT dựa trên một biểu thức logic

1. **WHEN-IN (When the SELECT Operand is IN the WHEN-IN Operand)**

* phép toán WHEN-IN khác biệt ở chỗ các thao tác sau WHEN-IN được thực hiện khi biểu thức được chỉ định bởi toán tử của câu lệnh SELECT nằm trong toán tử được chỉ định bởi toán tử của câu lệnh WHEN-IN, theo quy tắc của toán tử IN.

1. **WHEN-IS (When the SELECT Operand is Equal to the WHEN-IS Operand)**

* khác biệt chính là các câu lệnh sau câu lệnh WHEN-IS sẽ được thực hiện khi biểu thức được chỉ định trong câu lệnh WHEN-IS bằng với biểu thức được chỉ định trong câu lệnh SELECT.

1. **WHENxx (When True Then Select)**
2. **WRITE (Create New Records)**

* Phép toán WRITE trong RPG IV được sử dụng để ghi một bản ghi mới vào một tập tin.

**C WRITE FILE1 DS1**  
Phép toán WRITE này ghi các trường trong cấu trúc dữ liệu DS1 vào tập tin FILE1.

1. **XFOOT (Summing the Elements of an Array)**

* Tính **tổng các phần tử** trong mảng

**\* Đoạn mã RPGLE sau đây tính tổng của mảng ARR và đặt kết quả vào trường TOTAL.**

**D XFOOT Example**

**D ARR S 10 DIM(5) INZ(1 2 3 4 5)**

**D TOTAL S 5 2**

**C XFOOT ARR TOTAL**

Trong ví dụ này, mảng ARR có 5 phần tử và được khởi tạo với các giá trị từ 1 đến 5. Phép toán XFOOT được sử dụng để tính tổng của các phần tử trong mảng ARR và đặt kết quả vào trường TOTAL.

1. **XLATE (Translate)**

* Thay đổi kí tự ccura chuỗi From thành các ký tự của chuỗi To trong chuỗi Source

**D FromTo S 10 INZ('abcde:vwxyz')**

**D Source S 20 INZ('abcde123')**

**D Target S 20**

**C XLATE FromTo Source Target**

Trong ví dụ này, chuỗi nguồn **Source** là "abcde123" và chuỗi "From" và "To" là "abcde" và "vwxyz" tương ứng. Sau khi thực hiện phép toán XLATE, chuỗi **Target** sẽ chứa kết quả "vwxyz123" sau khi các ký tự từ "From" được dịch chuyển sang "To".Top of Form

1. **XML-INTO (Parse an XML Document into a Variable)**
2. **XML-SAX (Parse an XML Document)**
3. **Z-ADD (Zero and Add)**

* Phép tính Z-ADD được sử dụng để **cộng Factor 2 vào một trường chứa giá trị không**. Tổng được đặt vào trường kết quả

**D MyArray S 5 Dim(10) Inz(0)**

**D Total S 5 Dim(10) Inz(0)**

**D Counter S 5 Dim(10) Inz(0)**

**C Move 1 Counter**

**C Do Counter 10**

**C Z-Add Counter MyArray(Counter)**

**C Add 1 Counter**

**C EndDo**

**C MoveA MyArray Total**

**C Eval \*InLR = \*On**

Trong ví dụ này, chúng ta có một mảng **MyArray** có kích thước 10, mỗi phần tử được khởi tạo với giá trị 0. Chúng ta sử dụng một biến đếm **Counter** để duyệt qua mảng từ phần tử đầu tiên đến phần tử cuối cùng. Mỗi lần lặp, chúng ta thực hiện phép tính Z-ADD để cộng giá trị của **Counter** vào phần tử tương ứng trong mảng **MyArray**. Sau đó, chúng ta sao chép mảng **MyArray** vào mảng **Total**.

1. **Z-SUB (Zero and Subtract): trừ cho 0**

**D Amount S 5 Inz(1000)**

**D NegativeAmount S 5 Inz(0)**

**C Z-Sub Amount NegativeAmount**

**C Eval \*InLR = \*On**

Trong ví dụ này, chúng ta có một biến **Amount** với giá trị khởi tạo là 1000 và một biến **NegativeAmount** với giá trị khởi tạo là 0. Chúng ta sử dụng phép tính Z-SUB để trừ giá trị của **Amount** khỏi một trường chứa giá trị 0 (trường hợp đặc biệt này thường được sử dụng để đảo ngược dấu của một số). Kết quả, tức là số âm của **Amount**, được lưu vào biến **NegativeAmount**.

1. **DS(Data Structure) => class**

* Data Structure là một cách để tổ chức và lưu trữ dữ liệu liên quan trong chương trình RPG.

// Khai báo cấu trúc dữ liệu

**D MyDS DS**

**D Field1 10A**

**D Field2 5P 2**

**D Field3 8S 0**

1. **BLANK**

* **BLANK** là một phép toán được sử dụng để thiết lập một biến hoặc một phần của một biến thành **giá trị trống** hoặc **giá trị mặc định tương ứng với kiểu dữ liệu của biến đó**.