



IBM System i™

## CL (Control Language) Update

한국IBM 시스템사업본부 노기선(keysn@kr.ibm.com)

*i want stress-free IT.  
i want control.  
i want an **i**.*

© 2006 IBM Corporation

# Trademarks and Disclaimers

© IBM Corporation 1994-2005. All rights reserved.

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AS/400

iSeries

System i5

eServer

IBM

i5/OS



IBM (logo)

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Other company, product or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

# 목차

- Control Flow 기능
- Subroutines
- DCLF 복수 개 파일 지원
- 새로운 유형의 변수
  - Integer 변수
  - Pointer 변수
  - Defined 변수
  - Based 변수
- 다양해진 파일 I/O
- Miscellaneous
  - Pass parameter by value
  - 길어진 \*CHAR 변수
  - 추가된 파라미터
  - V5R2에서 수행하기

## 참고자료

- [www.iseries.ibm.com/infocenter](http://www.iseries.ibm.com/infocenter) > 버전 선택 > 프로그래밍 > CL
  - CL Command Finder
  - Alphabetical list of commands
  - CL Concept and reference
  - CL Programming

# Control Flow 기능



- DO 루프 명령을 사용할 수 있는 세 가지 방법
  - DOWHILE, DOUNTIL, 그리고 DOFOR
  - 최대 25 단계의 Doxxx nesting 지원
  
- Loop control flow 명령어
  - LEAVE 그리고 ITERATE
  
- Case/subcase 명령어
  - SELECT, WHEN, OTHERWISE, 그리고 ENDSELECT
  - 최대 25 단계의 SELECT nesting 지원

## Notes:

Control flow는 CL에 있어서는 “nice to have” 기능이라고 할 수 있겠다...IF, ELSE 그리고 GOTO 문의 조합으로 DO loop와 같은 보다 수준 높은 control flow 기능을 사용하는 것처럼 구현할 수 있다. 이때의 문제점은 그 프로그램을 읽는 사람은 그 코드가 loop를 구현하고 있다는 것을 알기 위해서는 굉장히 주의 깊게 들여다 봐야 한다는 것이다.

이러한 문제에 대한 해결책으로 V5R3에서 세 가지의 새로운 DO loop 명령어가 추가되었다:

- DOWHILE
- DUNTIL
- DOFOR

또한, loop 내에서의 보다 깔끔한 control flow를 위해서, LEAVE와 ITERATE 명령어가 추가되었다. 이를 사용함으로써 loop 내에서 GOTO 명령어를 사용하는 것을 피할 수 있다.

마지막으로, CL에서 case 구조를 사용하기 위해서 새로운 네 개의 명령어가 추가되었다: SELECT, WHEN, OTHERWISE 그리고 ENDSELECT.

## DOWHILE Loop

- CL의 IF문과 같은 COND
- Loop의 “꼭대기”에서 COND를 평가
- Old-style 코딩 예:

```
DCL VAR(&LGL) TYPE(*LGL)
:
CHECK: IF COND(*NOT &LGL) THEN(GOTO DONE)
: (group of CL commands)
GOTO CHECK
DONE:
```

- New-style 코딩 예:

```
DOWHILE COND(&LGL)
: (group of CL commands) ← 0번 이상 실행됨
ENDDO
```

## Notes:

새로 추가된 DOWHILE 명령어는 일정 조건이 논리적으로 참이라고 평가되는 동안 looping이 계속되도록 해준다.

조건을 조정하는 COND 파라미터는 IF 명령어에서 사용하던 것과 같다.

다른 high-level 언어의 Loop 구조와 마찬가지로 CL의 DOWHILE도 조건을 loop의 꼭대기에서 테스트한다. 그러므로 DOWHILE 명령이 첫번째 실행될 때 조건이 '거짓'으로 평가되면 DOWHILE loop은 단 한 번도 실행되지 않는다.

## DOUNTIL Loop

- CL의 IF 문과 유사한 COND 지원
- Loop의 “바닥”에서 COND를 평가한다
- Old style 코딩 예:

```
DCL VAR(&LGL) TYPE(*LGL)
:
LOOP:
: (group of CL commands)
IF COND(*NOT &LGL) THEN(GOTO LOOP)
```

- New style 코딩 예:

```
DOUNTIL COND(&LGL)
: (group of CL commands) ← 한 번 이상 실행됨
ENDDO
```

## Notes:

새로 추가된 DOUNTIL 명령어는 DOWHILE과 유사해 보인다.

두 명령어의 차이는 loop 조건에 대한 시험이 loop의 마지막에서 이루어지며 COND 파라미터에 대한 평가가 참이 될 때까지 loop가 계속 된다는 것이다. Loop의 조건이 '거짓'으로 시작된다고 가정한다는 점에서 조건에 대한 검사가 DOWHILE과는 반대이다.

예를 들어, 맞는 레코드를 찾을 때까지 레코드들을 조사한다. Loop 검사가 마지막에서 이루어지기 때문에 Loop가 한 번은 무조건 실행되게 된다.

## DOFOR Loop

- Syntax:

**DOFOR VAR( ) FROM( ) TO( ) BY( )**

- BY의 기본값은 '1'이며, 다른 파라미터들은 필수 입력 해야함
- VAR는 \*INT 또는 \*UINT 변수
- FROM 과 TO는 integer 상수, 수식 또는 변수 가능
- BY는 반드시 integer 상수 (음수 가능)
- FROM/TO는 loop가 시작될 때 평가되고 TO는 increment 이후에 평가됨
- DOWHILE 문과 마찬가지로 loop를 빠져나갈지 여부를 loop의 “꼭대기”에서 확인

## Notes:

새로 추가된 DOFOR 명령어는 기본적인면서도 강력한 counting-type loop이다.

조건을 조정하는 CL 변수는 반드시 integer 변수여야 한다. 초기값과 종료를 나타내는 값은 integer이거나 integer 수식(integer로 표현될 수 있는 상수 혹은 수식)일 수 있다. 증가값은 기본으로 1이며, 양수 또는 음수의 상수일 수 있다. 증가값 0을 허용하지만, 목적에 위배된다.

종료를 나타내는 값에 수식을 사용할 수 있으며 수식은 increment가 발생할 때마다 다시 계산되기 때문에 종료를 나타내는 값은 움직이는 표적이 될 수 있다.

DOWHILE 문과 마찬가지로, DOFOR도 loop의 “꼭대기”에서 loop를 빠져나갈지 여부를 확인한다. 이는 FROM 값이 TO 값보다 크거나 (BY가 양수이거나 0인 경우) TO 값보다 작은 경우(BY가 음수일 때) DOFOR loop은 한번도 실행되지 않는다.

## DOFOR Loop (계속)

- Old-style 코딩 예:

```
DCL &LOOPCTL TYPE(*DEC) LEN(5 0)
DCL &LOOPLMT TYPE(*DEC) LEN(5 0)
:
CHGVAR &LOOPCTL VALUE(1)
CHECK: IF COND(&LOOPCTL *GT &LOOPLMT) THEN(GOTO DONE)
: (group of CL commands)
CHGVAR &LOOPCTL VALUE(&LOOPCTL+1)
GOTO CHECK
DONE:
```

- New-style 코딩 예:

```
DCL &LOOPCTL TYPE(*INT) LEN(4)
DCL &LOOPLMT TYPE(*INT) LEN(4)
:
DOFOR VAR(&LOOPCTL) FROM(1) TO(&LOOPLMT) BY(1)
: (group of CL commands) ← 0번 이상 실행됨
ENDDO
```

## Notes:

IF와 GOTO 명령어를 이용해 counting-type loop을 구현한 예를 보여준다.

이와 같은 loop을 DOFOR 명령어를 이용해 보다 간단하고 명료하게 코딩하는 예를 보여준다.

## LEAVE 와 ITERATE

- DOWHILE, DOUNTIL 또는 DOFOR 그룹 내에서만 사용 가능
- LEAVE는 loop의 ENDDO 다음 CL문으로 control을 전달한다
- ITERATE는 loop의 마지막으로 control을 전달하여 loop의 종료 조건을 검사한다
- 둘 다 CMDLBL (Command label) 파라미터를 지원하여 여러 개의 (nested) loop에서 빠져나올 수 있다
  - 둘 다 기본 값은 \*CURRENT loop
- 예:

```
LOOP1: DOFOR &COUNTER FROM(1) TO(&LIMIT)
```

```
LOOP2: DOUNTIL COND(&FOUND)
```

```
IF (%SST(&NAME 1 10) *EQ '*NONE') THEN(LEAVE LOOP1)
```

```
ELSE (DO)
```

```
IF (%SST(&NAME 11 10) *EQ '*LIBL') THEN(ITERATE)
```

```
ENDDO
```

```
ENDDO /* End of DOUNTIL loop */
```

```
ENDDO /* End of DOFOR loop */
```

## Notes:

DO loop CL 명령어와 더불어 loop control 명령어가 두 개 더 추가되었다. RPG의 LEAVE 및 ITERATE op code와 유사하다고 생각하면 된다.

이 새로운 CL 명령어의 강력한 기능은, 이 명령어를 사용하는 loop를 포함하고 있는 loop에 대해서도 사용할 수 있다는 것이다. 이는 상위 DO loop 명령어에 사용된 LEAVE나 ITERATE 명령어에 label을 설정함으로써 가능하다. 예에 나오는 LEAVE LOOP1은 LOOP2와 LOOP1을 도우 빠져나오며 control을 LOOP1의 ENDDO 다음에 있는 명령어로 넘겨준다.

Label을 정의하지 않은 경우 LEAVE 및 ITERATE는 그 명령어가 쓰인 DO loop문에서 쓰이는 것으로 가정한다. 예에 있는 ITERATE 명령어는 control이 LOOP2의 ENDDO 명령어도 넘어가도록 하여 loop에서 빠져나갈지 조건을 검사하게 된다.

## SELECT Group

- SELECT 시작; 파라미터 없음
- ENDSELECT 문으로 그룹을 종료; 파라미터 없음
- 하나의 그룹은 최소 하나의 WHEN이 있어야 함
  - WHEN 명령어에는 IF와 마찬가지로 COND와 THEN 파라미터 있음
- OTHERWISE (optional)는 WHEN COND = True가 없을 때 수행
  - OTHERWISE 명령어는 ELSE문처럼 CMD 파라미터만 있음
- Example:

```
SELECT
  WHEN COND(&TYPE *EQ *CMD) THEN(DO)
    : (group of CL commands)
  ENDDO
  WHEN COND(&TYPE *EQ *PGM) THEN(DO)
    : (group of CL commands)
  ENDDO
  OTHERWISE CMD(CHGVAR &BADTYPE '1')
ENDSELECT
```

## Notes:

가	SELECT	IF THEN ELSE IF	WHEN	IF
ELSE	case structure	. SELECT		
SELECT	THEN	COND가	가	‘ ’
. Null THEN	, WHEN	가 control	ENDSELECT	.
WHEN	‘ ’	‘ ’	OTHERWISE	가
. WHEN	OTHERWISE	SELECT	OTHERWISE	OTHERWISE
가		가	ENDSELECT	control
.				



# Subroutines

- SUBR과 ENDSUBR 명령어 사이에 있는 code block
- 새로 추가된 CALLSUBR 명령어를 이용해 시작
  - argument/parameter passing 안 됨
  - 선택사항으로 RTNVAL 이용해 4 바이트 \*INT 변수 지정 가능
  - Subroutine 변수의 local scoping 안 됨
  - 레이블에 대한 local scoping 지원
  - Nesting 지원되지 않음 (subroutines 내의 subroutines)
- Return to caller는 RTNSUBR 또는 ENDSUBR
- Subroutine으로 들어가거나 subroutine에서 빠져 나오기 위해 GOTO 사용 불가. Subroutine 내에서는 GOTO 사용 가능

## Notes:

그동안 CL 프로그래머들은 CL 프로그램 여러 곳에서 수행될 수 있는 한 묶음의 코드를 정의할 수 있는 기능이 필요하다는 제안을 해왔다. 이 기능은 V5R4에서 CL subroutine을 통해 구현되었다.

CL subroutine은 local 스토리지나 파라미터를 가지지 않는다. 모든 DCL, DCLF 및 COPYRIGHT 문은 CL 프로그램의 첫부분에 들어가야 한다. 이를 CL subroutine에서 사용할 수는 없다.

CL subroutine은 4 바이트 integer 값을 반환할 수 있다. 이를 이용해서 subroutine이 올바르게 수행되었는지 여부를 표시할 수 있다.

Subroutine은 nesting할 수 없다. 다시 말해 subroutine 내에 SUBR 명령어를 사용할 수 없다. Subroutine에서 CALLSUBR 명령어를 이용해 다른 subroutine은 호출하거나 순환적으로 같은 subroutine을 호출할 수 있다.

CL 컴파일러는 subroutine 내에서 사용된 GOTO 명령어는 같은 subroutine 내에 정의된 LABEL을 참조하도록 강제한다. Subroutine 밖에 있는 코드에서 subroutine 안에 있는 LABEL을 참조할 수는 없다. 하나 이상의 subroutine에서 같은 이름의 LABEL 사용 가능하다.

## pseudo-subroutine 을 이용한 CL 프로그램

```

PGM
DCL &RTNLBL  *CHAR 10
:
CHGVAR &RTNLBL 'HERE'
GOTO PSEUDOSUBR /* 1st invocation of pseudo-subroutine */
HERE: /* Return label from 1st invocation of pseudo-subroutine */
:
CHGVAR &RTNLBL 'THERE'
GOTO PSEUDOSUBR /* 2nd invocation of pseudo-subroutine */
THERE: /* Return label from 2nd invocation of pseudo-subroutine */
:
GOTO GO_AROUND /* Branch around the pseudo-subroutine code */
PSEUDOSUBR:
/* Body of pseudo-subroutine PSEUDOSUBR */
:
/* Return-from-subroutine logic */
IF (&RTNLBL *EQ 'HERE') THEN(GOTO HERE)
IF (&RTNLBL *EQ 'THERE') THEN(GOTO THERE)
/* Need to handle other value for &RTNLBL as an error */
GO_AROUND:
:
ENDPGM

```

## Notes:

CL subroutine이 없었을 때도, 창조적인 CL 프로그래머들은 subroutine을 흉내내는 방법들을 만들어냈다. DOFOR, DOWHILE과 DOUNTIL이 없을 때 loop을 흉내 냈듯이 말이다.

예를 들어, CL 변수에 LABEL 이름을 지정하고, pseudo-subroutine 코드로 분기함으로써, CL 변수의 값은 어디로 분기할 것인지를 검사할 수 있다. 이러한 방법의 문제점으로는:

- Subroutine의 새로 시작하려 할 때마다 subroutine이 새로운 반환 LABEL을 처리하기 위한 로직이 추가되어야 한다.
- Subroutine 코드 앞에 GOTO를 추가해 subroutine으로 “falling in”하는 걸 막아야 한다
- GOTO가 많아지면서 코드가 덜 구조적으로 된다.

## 진짜 subroutine을 이용한 CL 프로그램

```
PGM
:
CALLSUBR  REALSUBR  /* 1st call to the subroutine */
:
CALLSUBR  REALSUBR  /* 2nd call to the subroutine */
:
/* End of program mainline code */

SUBR  REALSUBR
/* Body of subroutine REALSUBR */
:
RTNSUBR
:
ENDSUBR

ENDPGM
```

## Notes:

subroutine

- CALLSUBR 가 control flow
- CL
- Subroutine
- CALLSUBR
- Subroutines subroutine , CL subroutine stack

## Subroutines을 이용한 CL 예제

```
PGM /* A simple program with two subroutines */
  DCLPRCOPT SUBRSTACK(25)
  DCL VAR(&SRTNVAR) TYPE(*INT) LEN(4)
  DCL VAR(&SDEC) TYPE(*DEC) LEN(5 2)
  MONMSG MSGID(CPF0822) EXEC(GOTO CMDLBL(DUMP))

START:
  CALLSUBR SUBR(SUBR1) RTNVAL(&SRTNVAR)
DUMP:
  DMPCLPGM

START: SUBR SUBR(SUBR1)
        CALLSUBR SUBR(SUBR2)
        ENDSUBR RTNVAL(&SDEC)

START: SUBR SUBR(SUBR2)
        CALLSUBR SUBR(SUBR2)
        RTNSUBR RTNVAL(-1)
        ENDSUBR

ENDPGM
```

## Notes:

. SUBR2가 SUBR2 . CL subroutine recursion  
 , 가 subroutine loop가 CL .  
 CL runtime subroutine call stack 가 , subroutine stack  
 99 entry가 . Subroutine stack entry가 가 , subroutine  
 stack entry .  
 DCLPRCOPT (Declare Processing Options) subroutine stack 20~9999  
 . 26 subroutine stack entry가 가 CPF0822 (Subroutine stack overflow  
 at statement &1)가 .  
 MONMSG CPF0822 escape message가 control  
 DMPCLPGM (Dump CL Program) . DMPCLPGM  
 가 .



## Notes:

Dump CL Program (DMPCLPGM) 명령어는 당신의 친구이다! 모든 CL 프로그래머가 익숙해져야 할 편리한 명령어이다.

예제에 있는 CL 프로그램이 수행되면서 CPF0822가 발생했을 때의 DMPCLPGM output의 일부이다.

Exception message 정보가 나타남을 주목하라.

또한 subroutine stack 정보가 dump되어 나타남도 주목하라. 첫 10 subroutine stack entry만이 나타나긴 하지만, CALLSUBR loop 안에 있다면, 이것으로 충분할 것이다.

덤프된 정보에는 프로그램 내의 모든 CL 변수의 현재 값도 나타난다.

## 향상된 파일 지원



- DCLF에서 최고 5개의 파일 “instance” 지원
  - Instances는 같은 파일이거나 다른 파일
- DCLF문에 OPNID (Open Identifier) 파라미터 추가됨
- OPENID의 기본값은 \*NONE
  - OPENID(\*NONE)인 경우 하나의 DCLF만 허용
- OPNID는 10 character까지의 simple name 사용

## Notes:

CL 개발자들은 binary integer 변수가 없을 때 %BINARY built-in 기능을 사용했고, DO loop이 없을 때는 IF와 GOTO 명령어를 혼합해서 사용했습니다. 하지만 하나의 CL 프로시저가 하나의 파일만을 선언해서 쓸 수 있는 제한을 극복할 수 있는 방법은 없다고 얘기합니다.

이를 개선하기로 결정했고, 무한대로 많은 파일을 사용하는 것을 필요치 않을 것이라 예상되었습니다. DCLF 명령이 다섯 개의 파일을 지원하도록 지원을 확장했습니다.

각 DCLF 명령어에 정의된 OPNID는 모두 달라야 하며, 같은 파일을 참조하는 복수 개의 DCLF 명령을 사용할 수 있습니다. 예를 들어, 데이터베이스 파일 FILEA에 대해서 OPNID PASS1과 PASS2를 정의한 DCLF문이 있을 수 있다는 것입니다. 이를 활용함으로써 RCVF문에서 OPNID(PASS1)을 이용해 한번 읽고, end-of-file을 만나면 RCVF 문에 OPNID(PASS2)을 지정함으로써 다시 한번 파일을 읽을 수 있습니다. 아주 아름다운 건 아니지만, it works!

## 파일 지원 (계속)

- OPNID가 지정되면, 선언된 CL 변수들은 그 이름과 밑줄이 접두사로 붙는다(e.g. &MYTESTFILE\_CUSTNAME )
- 기존 파일 I/O CL 명령어들에 OPNID 추가
  - RCVF
  - ENDRCV
  - SNDF
  - SNDRCVF
  - WAIT

## Notes:

기존에 DCLF를 사용하고 있는 CL 프로그램과의 호환을 가능하게 하기 위해 open identifier를 지정하지 않은 DCLF 하나를 허용한다. OPNID(\*NONE)으로 지정된 DCLF 하나 + OPNID(name)으로 지정된 DCLF 네 개까지 지원되며, OPNID(name)으로 지정된 DCLF만 다섯 개를 사용할 수도 있다.

모든 I/O 명령어들이 단일 파일을 가정하고 있기 때문에, 각 I/O 명령어에 open identifier를 추가하여 어느 파일에 대해 작동해야 하는 건지를 확인할 수 있게 했다.

## 새로운 변수

- Integer 변수



V5R3

- Pointer 변수

- Defined 변수

- Based 변수



V5R4

# Binary (Integer) 변수



V5R3

- DCL 명령어의 TYPE에 \*INT와 \*UINT 추가됨  
\*\* 1980년 이후 처음으로 추가된 새로운 CL 변수의 데이터 유형\*\*
- LEN(2)와 LEN(4) 지원
- %BIN built-in을 사용하는 것과 비교했을 때 훨씬 더 "cleaner"
- API로 파라미터 전달할 때 사용
- 다른 HLL 프로그램으로 파라미터 전달할 때 또는 다른 HLL 프로그램으로부터 integer 파라미터를 전달 받을 때 사용

## Notes:

V5R3에서 추가된 기능 중 하나는 true binary integer 변수를 쓸 수 있게 됐다는 것이다. Character data를 2-바이트나 4-바이트 integer로 view하기 위해서 컴파일러가 %BINARY built-in function을 제공하였으나 이 방법은 다른 HLL들처럼 바로 integer 변수를 선언하여 사용하는 것에 비해 불편했다. V5R3에서는 2-바이트 또는 4-바이트의 signed 또는 unsigned integer 변수를 선언할 수 있다. 이 새로운 변수는 API를 호출하거나 integer 파라미터를 사용하는 다른 HLL 프로그램을 호출할 때 사용된다.

흥미로운 것은, 이는 System/38 release 1이 발표된 1980년 이후 처음 새로 추가된 데이터 유형이라는 것입니다!

# Pointer CL Variables



V5R4

- DCL 문에 TYPE(\*PTR) 추가됨
- Pointer를 지정하기 위해 %ADDRESS built-in 추가됨
- Pointer 변수의 offset 부분을 지정하거나 retrieve할 위해 %OFFSET built-in 추가
- DLC문에 STG(\*BASED) 속성 추가
- Pointer 파라미터는 사용하는 프로그램을 호출하거나, 호출될 수 있게됨
- 많은 프로시저 API들을 ILE CL에서 사용할 수 있게 됨
  - Full record-level file I/O
  - String functions

## Notes:

CL과 포인터가 같은 문장 안에서 거론되는 것을 기대한 사람은 많지 않을 것이다. 이 기능 향상은 CL이 API와 더 친해지라고 추가된 것이다. 사실 많은 API들은 포인터 파라미터를 기대하거나 array로 정보를 반환하는데, 이도 포인터와 based 변수를 이용해서 조정해야 한다.

V5R4에 포인터에 대한 지원을 추가했으며, 이와 관련된 based 변수, %ADDRESS 및 %OFFSET built-in들을 추가하여 포인터 변수를 조작할 수 있게 했다. CL과 ILE CL 모두 이 변수를 사용할 수 있으나, CL 컴파일러가 활용할 수 없는 많은 API들을 ILE CL 프로그램에서 사용할 수 있게 되므로 더 혜택이 된다.

# Defined CL 변수



V5R4

- DCL 명령어에 STG(\*DEFINED) keyword 추가됨
- Defined-over CL 변수에 대해 이름을 부여해야 한다 (DCL 명령어에 DEFVAR keyword 추가)
- Defined-over CL 변수의 starting position (기본값=1)을 제공할 수 있다
- 복잡한 substring 사용할 필요 없음
- Varying-character fields에 유용하고 CL에 structure에 해당되는 기능 제공

## Notes:

보다 큰 CL 변수의 일부를 CL 변수로 선언할 수 있으므로, CL에서 structure와 유사한 지원을 제공한다.

예를 들어, 데이터베이스 파일 레코드와 같은 크기의 일반적인 character variable을 선언하고서 그 record format에 있는 가 field를 defined CL 변수로 선언하는 것이다.

하지만 모든 필드를 defined 변수로 선언해야 하는 것은 아니다. 프로그램에서 사용해야 하는 필드들에 대해서만 defined 변수를 선언해주면 된다. 뒤에 나올 예제에서 이 기능을 보여주고 있다.

## V5R4 추가기능에 대한 예제

- \*DEFINED CL 변수 선언하기

```
DCL &QUALOBJ *CHAR LEN(20)
```

```
DCL &OBJ *CHAR LEN(10) STG(*DEFINED) DEFVAR(&QUALOBJ 1)
```

```
DCL &LIB *CHAR LEN(10) STG(*DEFINED) DEFVAR(&QUALOBJ 11)
```

- Pointer CL 변수 선언하기

```
DCL &CHARVAR *CHAR LEN(10)
```

```
DCL &PTRVAR *PTR ADDRESS(&CHARVAR)
```

- \*BASED CL 변수 선언하기

```
DCL &ENTRYPTR *PTR
```

```
DCL &CHARENTRY *CHAR 10 STG(*BASED) BASPTR(&ENTRYPTR)
```

- \*DEFINED Pointer CL 변수 선언하기

```
DCL &CHARSTRUCT *CHAR LEN(48)
```

```
DCL &PTRVAR2 *PTR STG(*DEFINED) DEFVAR(&CHARSTRUCT 17)
```

## Notes:

이 예문들은 처음에는 간단하지만 뒤로 갈수록 복잡해진다.

첫번째 예는 20 character짜리 qualified object name의 두 부분을 DEFINED로 선언함으로써 %SUBSTRING 사용을 피할 수 있는 예이다. 라이브러리 이름을 지정하기 위해서는 &QUALOBJ 변수를 substring하기 보다 단순히 &LIB을 변경하면 된다. 새로운 STG 파라미터와 DEFVAR 파라미터를 주목하라.

두번째 예는 간단한 pointer 변수를 선언하고, 이를 &CHARVAR 변수의 스토리지 위치를 가리키도록 초기화하는 것을 보여주고 있다. 초기화할 때 VALUE 파라미터를 사용하지 않고 ADDRESS 파라미터를 사용한 것에 주목하라.

세번째 예는 간단한 based 변수를 선언한 것을 보여주고 있다. 이 변수에 대해서는 CL 프로그램이나 프로시저가 호출될 때 program working stroage가 할당되지 않는다; 변수의 basing pointer가 가리키는 스토리지 위치를 보여준다. STG(\*BASED)로 선언되는 모든 변수들은 DCL문에서 BASPTR 파라미터에 basing pointer를 반드시 지정해주어야 한다. Based 변수를 사용할 때는 basing pointer를 이미 알고 있기 때문에, pointer qualification이 필요하지 않다.

마지막 예는 새로운 CL 변수들을 함께 사용하는 방법을 보여준다. 예를 들어, 포인터를 보다 큰 structure의 일부로 선언할 수도 있다. 그 structure도 포인터에 기반한 것일 수도 있다. 이 예에서의 &CHARSTRUCT 변수는 CL 프로그램 또는 프로시저가 호출될 때 할당되는 간단한 변수로 선언되었다.

## DCL 추가 기능에 대한 예

PGM

```
DCL &QUALOBJ *CHAR LEN(20)
DCL &OBJ *CHAR LEN(10) STG(*DEFINED) DEFVAR(&QUALOBJ 1)
DCL &LIB *CHAR LEN(10) STG(*DEFINED) DEFVAR(&QUALOBJ 11)
DCL &CHARVAR *CHAR LEN(10)
DCL &PTRVAR *PTR ADDRESS(&CHARVAR)
DCL &ENTRYPTR *PTR
DCL &CHARENTRY *CHAR 10 STG(*BASED) BASPTR(&ENTRYPTR)
DCL &CHARSTRUCT *CHAR LEN(32) VALUE('FIRSTENTRY')
DCL &PTRVAR2 *PTR STG(*DEFINED) DEFVAR(&CHARSTRUCT 17)
```

/\* CL 변수에 value 지정하기 \*/

```
CHGVAR &OBJ 'MYMSGF'
CHGVAR &LIB 'MYLIB'
CHGVAR &CHARVAR 'ABCDEFGHJIJ'
CHGVAR &PTRVAR2 (%ADDRESS(&CHARSTRUCT))
CHGVAR &ENTRYPTR &PTRVAR2
```

/\* CL 변수의 값을 보기 위해 프로그램 덤프하기 \*/

```
DMPCLPGM
ENDPGM
```

# Notes:

, 가 CL 가 .  
 , 가 CHGVAR sub-string .  
 built-in runtime 가 CHGVAR VALUE %ADDRESS .  
 VAR 가 . CHGVAR VALUE

## Compiler Cross-Reference Listing

```

LPAR4TLM
File Edit View Communication Actions Window Help
Display Spooled File
File . . . . . : TESTDCL Page/Line 2/7
Control . . . . . Columns 1 - 78
Find . . . . .
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...
Cross Reference

Declared Variables
Name Defined Type Length Reference
&CHARENTRY 800 *CHAR 10
* CPD0726 10 Variable &CHARENTRY declared but not referred to.
&CHARSTRUCT 900 *CHAR 32 1000
&CHARVAR 500 *CHAR 10 600
&ENTRYPTR 700 *PTR 16 800
&LIB 400 *CHAR 10 1300
&OBJ 300 *CHAR 10 1200
&PTRVAR 600 *PTR 16
* CPD0726 10 Variable &PTRVAR declared but not referred to.
&PTRVAR2 1000 *PTR 16 1500
&QUALOBJ 200 *CHAR 20 300
* CPD0791 00 No labels used in program.
* * * * * E N D O F C R O S S R E F E R E N C E
More...

F3=Exit F12=Cancel F19=Left F20=Right F24=More keys

```

MA a

I902 - Session successfully started

## Notes:

CL

spooled file cross-reference section

CL

# DMPCLPGM Spooled File Output

LPAR4TLM

File Edit View Communication Actions Window Help

Display Spooled File

File . . . . . : QPPGMDMP Page/Line 1/14  
 Control . . . . . : Columns 1 - 78  
 Find . . . . . :  
 \*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...  
 Variables

Variable	Type	Length	Value
&CHARENTRY	*CHAR	10	'FIRSTENTRY'
&CHARSTRUCT	*CHAR	32	'FIRSTENTRY 0 6'
		+26	'{ °ú ^'
&CHARVAR	*CHAR	10	'ABCDEFGHIJ'
&ENTRYPTR	*PTR		
&LIB	*CHAR	10	'MYLIB '
&OBJ	*CHAR	10	'MYMSGF '
&PTRVAR	*PTR		
&PTRVAR2	*PTR		
&QUALOBJ	*CHAR	20	'MYMSGF MYLIB '
			* * * * * E N D O F D U M P

Bottom

F3=Exit F12=Cancel F19=Left F20=Right F24=More keys

MA a

I902 - Session successfully started

## Notes:

새로운 유형의 CL 변수에 값을 할당해줬을 때 DMPCLPGM 명령어로 생성된 스폴 파일에 어떻게 나타나는지 보여준다.

변수들은 알파벳 순서로 나열된다. 할당된 값을 hexa값도 화면 오른쪽에 나타난다.

Dump CL Program과 친해지는 것을 잊지 말라!

## Based & Defined 변수 사용하기

- %SUBSTRING을 이용해 API의 결과 데이터에 접근하기

```
DCL &LISTHDR *CHAR 192
```

```
IF COND(%SST(&LISTHDR 104 1) = 'C') THEN(DO)
```

- \*DEFINED 변수를 이용해 API의 결과 데이터에 접근하기

```
DCL &LISTHDR *CHAR 192
```

```
DCL &LISTSTS *CHAR 1 STG(*DEFINED) DEFVAR(&LISTHDR 104)
```

```
IF COND(&LISTSTS = 'C') THEN(DO)
```

- User space에 대한 포인터를 이용해 API 결과 데이터에 접근(\*BASED 변수 이용)

```
DCL &USRSPCPTR *PTR
```

```
DCL &LISTHDR *CHAR 192 STG(*BASED) BASPTR(&USRSPCPTR)
```

```
DCL &LISTSTS *CHAR 1 STG(*DEFINED) DEFVAR(&LISTHDR 104)
```

```
IF COND(&LISTSTS = 'C') THEN(DO)
```

## Notes:

이곳에 발췌된 코드들은 \*DEFINED 변수를 사용하는 것이 아주 간단하며, 이를 하용함으로써 CL 프로그램이 보다 읽기 쉬워진다는 것을 보여준다.

첫번째 코드는 지금 만약 'List-type' API(i5/OS에 포함되어 있는 API중 오브젝트나 job 과 같은 것의 list를 반환하는 API들)를 CL에서 호출해야 한다면 사용하는 방법이다. 반환된 데이터의 첫부분에 헤더가 있고 포지션 104에 한 캐릭터짜리 필드가 있어서 가용한 모든 entry가 반환되면 그 자리가 'C'로 채워진다. V5R4 이전까지는 이를 위해서 헤더에 대해서 substring을 해서 status field를 확인해야 했다.

V5R4에서는 헤더 위에 정의된 한 캐릭터짜리 CL 변수를 선언할 수 있다. 이 방법으로 list의 상태 필드를 확인하는 것은 속도도 조금 빠르지만, 필드를 substring된 데이터 바이트가 아닌 이름으로 검사할 수 있어 더 알아보기 쉽다.

한 단계 더 높여 본다면, 아주 긴 리스트를 다룰 때는 List API를 호출해서 돌아온 정보를 \*CHAR 변수에 저장하는 대신 user space에 저장할 수도 있다. 이렇게 할 경우, 헤더는 user space의 시작을 가리키는 basing pointer를 가지는 \*BASED 변수로 선언하게 된다. List status 변수를 선언하는 방법과 그 변수를 참조하는 방식은 같다는 것에 주목하라.

## 포인터 변수를 이용하는 예

```
PGM PARM(&SCANVALUE &NEWVALUE)
/* &STRING 내에 있는 모든 &SCANVALE를 &NEWVALUE로 대체한다 */
DCL VAR(&SCANVALUE) TYPE(*CHAR) LEN(1)
DCL VAR(&NEWVALUE) TYPE(*CHAR) LEN(1)
DCL VAR(&STRING) TYPE(*CHAR) LEN(26) +
    VALUE('This is the string to scan')
DCL VAR(&BASEPTR) TYPE(*PTR) ADDRESS(&STRING)
DCL VAR(&SCANFOUND) TYPE(*CHAR) LEN(1) +
    STG(*BASED) BASPTR(&BASEPTR)
DCL VAR(&CONTROLS) TYPE(*CHAR) LEN(8) +
    VALUE(X'0100000000000001A') +
/* String 길이 26바이트로 스캔 시작 */
DCL &RTNCDE TYPE(*INT) LEN(4)
```

## Notes:

가 callable ILE .V5R4 CL 가 string \_SCANX based . \_SCANX

## 포인터 변수를 이용하는 예(계속)

```
CHGVAR VAR(%SST(&CONTROLS 4 1)) VALUE(&SCANVALUE)
CALLPRC PRC('_SCANX') PARM((&BASEPTR) +
(&CONTROLS) (X'48000000' *BYVAL)) +
          RTNVAL(&RTNCDE) /* Scan the string */
/* &SCANVALUE를 모두 찾을 수 있도록 loop */
DOWHILE COND(&RTNCDE=0) /* Scan value found? */
  CHGVAR VAR(&SCANFOUND) VALUE(&NEWVALUE)
  CALLPRC PRC('_SCANX') PARM((&BASEPTR) +
(&CONTROLS) (X'48000000' *BYVAL)) +
          RTNVAL(&RTNCDE) /* Scan string again */
ENDDO
/* 결과 string을 메시지로 뿌린다 */
SNDPGMMSG MSG(&STRING) TOPGMQ(*EXT)

ENDPGM
```

## Notes:

\_SCANX                    &BASEPTR가 가                    string                    &SCANVALUE character  
 .  
 가                    .                    , non-zero return code가                    &SCANVALUE    &NEWVALUE  
 , \_SCANX    &BASEPTR  
 based        &SCANFOUND                    CHGVAR (change variable) .

## Debug using Existing Tools

The screenshot shows a debugger window titled "LPAR4TLM" with a menu bar (File, Edit, View, Communication, Actions, Window, Help) and a toolbar. The main area displays the source code for a program named "SCANX" from the "VIG" library. The code includes conditional execution, variable assignment, and a call to a procedure. A status bar at the bottom indicates the session is successfully started.

```

Program:      SCANX          Library:      VIG          Module:      SCANX
 20      DOWHILE      COND(&RTNCDE=0) /* Scan value found? */
 21          CHGVAR   VAR(&SCANFOUND) VALUE(&NEWVALUE)
 22          CALLPRC  PRC('_SCANX')  PARM((&BASEPTR) (&CONTROLS) +
 23              (X'48000000' *BYVAL)) +
 24              RTNVAL(&RTNCDE) /* Scan string again */
 25      ENDDO
 26      /* Output the resulting string in a message      */
 27      SNDPGMMSG   MSG(&STRING) TOPGMQ(*EXT)
 28      ENDPGM
  
```

Bottom

Debug . . .

---

F3=End program    F6=Add/Clear breakpoint    F10=Step    F11=Display variable  
 F12=Resume        F17=Watch variable    F18=Work with watch    F24=More keys  
 &STRING = 'This,is,the string to scan'

MA a MW

I902 - Session successfully started

## Notes:

이 화면은 앞 페이지의 예제 CL 코드를 CRTBNDCL (create bound ILE CL program) 명령에 DBGVIEW(\*SRC)를 지정해서 컴파일 한 후, STRDBG 명령으로 source-level debug 기능을 시작했을 때 나타나는 화면이다.

이 debug 기능은 pointer, based, 그리고 defined 변수를 지원하는 다른 언어의 프로그램과 똑같이 작동한다.

## \*BASED와 \*DEFINED 변수를 List API와 함께 사용하기

```
/* Generic header information */
DCL    VAR(&USRSPCPTR) TYPE(*PTR)
DCL    VAR(&LISTHDR) TYPE(*CHAR) STG(*BASED) +
      LEN(192) BASPTR(&USRSPCPTR)
DCL    VAR(&LISTSTS) TYPE(*CHAR) STG(*DEFINED) +
      LEN(1) DEFVAR(&LISTHDR 104)
DCL    VAR(&LISTENTOFS) TYPE(*INT) STG(*DEFINED) +
      DEFVAR(&LISTHDR 125)
DCL    VAR(&LISTENTNBR) TYPE(*INT) STG(*DEFINED) +
      DEFVAR(&LISTHDR 133)
DCL    VAR(&LISTENTSIZ) TYPE(*INT) STG(*DEFINED) +
      DEFVAR(&LISTHDR 137)

/* List object detail information */
DCL    VAR(&LISTENTPTR) TYPE(*PTR)
DCL    VAR(&LISTENT) TYPE(*CHAR) STG(*BASED) +
      LEN(30) BASPTR(&LISTENTPTR)
DCL    VAR(&OBJNAME) TYPE(*CHAR) STG(*DEFINED) +
      LEN(10) DEFVAR(&LISTENT 1)
DCL    VAR(&OBJTYPE) TYPE(*CHAR) STG(*DEFINED) +
      LEN(10) DEFVAR(&LISTENT 21)
```

## \*BASED와 \*DEFINED 사용하기 (계속)

```

/* Miscellaneous variables */
DCL VAR(&CURRENT) TYPE(*INT)
DCL VAR(&TEXT) TYPE(*CHAR) LEN(50)

/* User Space 존재하는지 확인하기 */
CHKOBJ OBJ(QTEMP/OBJLIST) OBJTYPE(*USRSPC) MBR(*NONE)
/* 존재하지 않으면 User Space 생성하기 */
MONMSG MSGID(CPF9801) EXEC(CALL PGM(QUSCRTUS) +
    PARM('OBJLIST QTEMP 'QUSLOBJ ' +
    X'00000001' X'00' *CHANGE 'Output from +
    QUSLOBJ API'))

/* QTEMP 라이브러리에 존재하는 모든 오브젝트의 리스트 가져오기 */
CALL PGM(QUSLOBJ) PARM('OBJLIST QTEMP ' +
    'OBJL0100' *ALL QTEMP ' *ALL')

/* User Space의 포인터 가져오기 */
CALL PGM(QUSPTRUS) PARM('OBJLIST QTEMP ' +
    &USRSPCPTR)
  
```

## \*BASED 와 \*DEFINED 사용하기 (계속)

```

/* List가 성공적으로 만들어졌는가?          */
IF COND((&LISTSTS = 'C') *AND (&LISTENTNBR > 0)) +
THEN(DO)
/* 첫째 리스트의 어드레스 지정하기          */
CHGVAR  VAR(&LISTENTPTR) VALUE(&USRSPCPTR)
CHGVAR  VAR(%OFFSET(&LISTENTPTR)) +
        VALUE(%OFFSET(&LISTENTPTR) + &LISTENTOFS)
/* 반환된 모든 entry들 처리하기             */
DOFOR   VAR(&CURRENT) FROM(1) TO(&LISTENTNBR)
/* 오브젝트 정보 쓰기                      */
CHGVAR  VAR(&TEXT) VALUE('Object: ' +
        *CAT &OBJNAME +
        *CAT ' Type: ' *CAT &OBJTYPE)
SNDPGMMMSG MSG(&TEXT) TOPGMQ(*EXT)
/* 다음 리스트의 어드레스 정하기           */
CHGVAR  VAR(%OFFSET(&LISTENTPTR)) +
        VALUE(%OFFSET(&LISTENTPTR) + &LISTENTSIZ)
ENDDO
SNDPGMMMSG MSG('End of list') TOPGMQ(*EXT)
ENDDO

```

## Notes:

이 예제는 앞에서 소개한 몇 가지 개념들과 코드들을 모았다.

이 프로그램은 output user space가 존재하는지 확인하며, CHKOBJ 명령어가 CPF9801 (Object not found) 메시지를 뿌리면 새로 생성한다.

QUSLOBJ (List Objects) API가 호출되어 결과가 user space로 보내진다. QUSPTRUS API를 호출해서 반환된 데이터의 포인터를 알아낸다. 반환된 데이터의 제일 앞부분에는 모든 entry가 반환됐는지 알려주는 헤더와 첫번째 entry의 offset, entry의 개수, 그리고 이 프로그램에서는 사용하지 않는 여러 가지 정보가 있다. \*DEFINED로 선언된 CL 변수들이 헤더 내용 중에서 프로그램에서 사용해야하는 부분만을 선언하고 있다는 것에 주목하라.

List entry 구조의 basing pointer를 지정하기 위해서 %OFFSET built-in 함수를 사용하고 있다. 여기서도 역시, 프로그래머에게 관심있는 필드만을 \*DEFINED 변수로 선언하고 있다. List entry가 처리되고나면, %OFFSET 함수를 이용해 list entry의 basing pointer를 다음 entry로 덤프한다.

이는 entry의 개수를 terminating value로 하는 DOFOR loop로 처리하며, 이때 terminating value는 list 헤더에 있는 \*DEFINED 변수이다.

The QUSLOBJ (List Objects) API is called and the output is directed to the user space. API

# CL에서 보다 많은 File I/O 하기



## 데이터베이스 파일 FILEA의 record format DDS

```
R RECORD
  KEYVALUE      5 0
  DATA         10
  MOREDATA     10
K KEYVALUE
```

## FILEA에 대해 같은 record format을 정의한 CL

```
DCL &RECORD *CHAR LEN(23) /* Record buffer */
DCL &RECORDSIZE TYPE(*INT) VALUE(23) /* Record length */
DCL &KEY *DEC LEN(5 0) STG(*DEFINED) DEFVAR(&RECORD)
DCL &KEYSIZE TYPE(*INT) VALUE(3) /* Size of key (bytes) */
DCL &DATA *CHAR LEN(10) STG(*DEFINED) DEFVAR(&RECORD 4)
DCL &MOREDATA *CHAR LEN(10) STG(*DEFINED) DEFVAR(&RECORD 14)
```

## Notes:

CL의 input/output 지원은 늘 매우 제한적이었다. 데이터베이스 파일에 대해서 CL은 파일은 sequential하게 읽는 것만 가능했다...데이터베이스로 쓰기도 안되고, keyed I/O도 안되고, 레코드 갱신, 삭제 모두 안 됐다. V5R4에서는 이런 것이 바뀐다. CL에서 운영체제에 포함되어 있는 풍부한 레코드 I/O 프로시저들을 사용할 수 있게 되었기 때문이다.

다음 몇 페이지에서는 CL 코드로 FILEA라는 간단한 keyed physical file에 대해 record-level I/O 하는 것을 보여준다. FILEA의 record format은 단 세 개의 필드로 구성되어 있다.

레코드 I/O API를 써서 I/O할 때의 단점은, 파일의 record format 구조에 맞게 CL 선언을 maunally 해주어야 한다는 것이다.

## 데이터베이스 파일 FILEA OPEN

```

DCL &NULL TYPE(*CHAR) LEN(1) VALUE(X'00')
DCL &FILENAME *CHAR 11      /* 작업할 파일 */
DCL &MODE *CHAR 50          /* Open modes      */
DCL &RFILE *PTR             /* File pointer    */

/* File feedback information */
DCL &FEEDBACK TYPE(*PTR)    /* I/O feedback pointer */
DCL &FDBCKDATA TYPE(*CHAR) STG(*BASED) +
    LEN(64) BASPTR(&FEEDBACK) /* Feedback data */
DCL &NUM_BYTES TYPE(*INT) STG(*DEFINED) +
    DEFVAR(&FDBCKDATA 37) /* 처리된 바이트 수 */

/* _Ropen을 호출하여 FILEA 열기. 이때 파일의 포인터 반환됨 */
CHGVAR VAR(&FILENAME) VALUE('FILEA' *TCAT &NULL)
CHGVAR VAR(&MODE) VALUE('rr+' *TCAT &NULL)
CALLPRC PRC('_Ropen') PARM((&FILENAME) (&MODE)) +
    RTNVAL(&RFILE)

```

## Notes:

Record-level I/O API (open mode) V2R3 ILE C null-terminated string . input string (open  
 \_Ropen API가 ( &RFILE), V5R4 ILE CL .

## 레코드를 key로 읽는 CL 프로그램

```

PGM  PARM(&KEYIN &DATAIN)
:
DCL  &KEYIN  *DEC LEN(15 5) /* 갱신할 레코드의 key */
DCL  &DATAIN *CHAR LEN(10) /* 갱신할 값 */
DCL  &KEY    *DEC LEN(5 0) STG(*DEFINED) DEFVAR(&RECORD)
DCL  &OPTS   TYPE(*INT) /* Options for _Rreadk */
DCL  &KEY_EQ TYPE(*INT) VALUE(X'0B000100') /* Equal to Key */
DCL  &RTNCDE TYPE(*INT) /* Return code */
:
CHGVAR VAR(&KEY) VALUE(&KEYIN)
CHGVAR VAR(&OPTS) VALUE(&KEY_EQ)

/* _Rreadk를 호출하여 key field = &KEYIN로 레코드 읽는다 */
CALLPRC PRC('_Rreadk') +
      PARM(((&RFILE *BYVAL) (&RECORD) (&RECORDSIZE *BYVAL) +
            (&OPTS *BYVAL) (&KEY) (&KEYSIZE *BYVAL))) +
      RTNVAL(&FEEDBACK)

```

## Notes:

\_Rreadk (Read record by key) API    \_Ropen  
 .    \_Rreadk가    input  
 "by value"    .  
 \_Rreadk가    key    가    가    'Number of bytes  
 processed'    0    key    가    .

## 레코드 추가 또는 갱신하기

```
/* Matching key 레코드가 없으면, 새로운 레코드를 write 하기 */
IF COND(&NUM_BYTES *EQ 0) THEN(DO)
  CHGVAR VAR(&DATA) VALUE(&DATAIN)
  CHGVAR VAR(&MOREDATA) VALUE('      ')
  CALLPRC PRC('_Rwrite') +
    PARM((&RFILE *BYVAL) (&RECORD) (&RECORDSIZE *BYVAL)) +
    RTNVAL(&FEEDBACK)
ENDDO
```

```
/* 레코드가 성공적으로 읽어졌으니, 기존 레코드 갱신하기 */
ELSE (DO)
  CHGVAR VAR(&DATA) VALUE(&DATAIN)
  CALLPRC PRC('_Rupdate') +
    PARM((&RFILE *BYVAL) (&RECORD) (&RECORDSIZE *BYVAL)) +
    RTNVAL(&FEEDBACK)
ENDDO
```

## Notes:

이 CL 프로그램은 `_Rreadk`에서 반환된 피드백을 검사해서 새로운 레코드를 쓰거나 또는 기존에 있는 레코드를 다른 값으로 대체한다.

- Matching key 필드가 파일에 없으면, `_Rwrite` (Write Record) API를 호출하여 새로운 레코드를 작성한다.
- 특정 key에 맞는 레코드를 발견하면, `_Rupdate` (Update Record) API를 호출하여 레코드 값을 대체한다.

# 디렉토리 내용 읽기

(by Scott Klement)

PGM

```
DCL VAR(&NUL) TYPE(*CHAR) LEN(1) VALUE(X'00')
DCL VAR(&DIRNAME) TYPE(*CHAR) LEN(200)
DCL VAR(&EXT) TYPE(*CHAR) LEN(4)
DCL VAR(&POS) TYPE(*INT) LEN(4)
DCL VAR(&SUCCESS) TYPE(*INT) LEN(4)
DCL VAR(&HANDLE) TYPE(*PTR)
DCL VAR(&ENTRY) TYPE(*PTR)
DCL VAR(&NULLPTR) TYPE(*PTR)
DCL VAR(&STMF) TYPE(*CHAR) LEN(640)
```

```
/* readdir로부터 directory entry 반환됨. 796 chars이고 */
/* position 53~56에 &NAMELEN라는 필드 있음 */
/* 그리고 position 57~696에 &NAME이라는 필드 있음 */
DCL VAR(&DIRENT) TYPE(*CHAR) LEN(796) +
DCL VAR(&NAMELEN) STG(*DEFINED) DEFVAR(&DIRENT 53) +
TYPE(*UINT) LEN(4)
DCL VAR(&NAME) STG(*DEFINED) DEFVAR(&DIRENT 57) +
TYPE(*CHAR) LEN(640)
```

## Notes:

이 예제는 저자인 Scott Klement의 허락을 받고 올린다. CL에 포인터 지원이 추가된다는 것을 알았을 때 Scott은 처음엔 별로 관심이 없었다. 그러다가, 강력한 Integrated File System (IFS) 관련 API들을 CL에서 사용할 수 있음을 알고 CL에서의 포인터의 가치를 다시 평가했다.

이 CL 프로그램은 디렉토리를 하나 열어서 그 안에 있는 파일들을 처리하여 특정 파일 확장자를 가진 것들을 찾아낸다. 그 파일들은 하나의 데이터베이스 파일로 합쳐지며, RPG 프로그램 (이 교재에는 없음)이 호출되어 그 데이터베이스의 정보를 처리한다.

## 디렉토리 내용 읽기 (계속)

```
/* **** */
/* /edi/incoming/custdata 디렉토리를 연다 */
/* API는 실패를 나타내기 위해 NULL 포인터를 반환한다 */
/* **** */
CHGVAR VAR(&DIRNAME) VALUE('/edi/incoming/tabdata' *CAT &NUL)
CALLPRC PRC('opendir') PARM((&DIRNAME)) RTNVAL(&HANDLE)
IF (&HANDLE *EQ &NULLPTR) DO
  SNDPGMMMSG MSGID(CPF9897) MSGF(QCPFMSG) MSGDTA('Unable +
  to open directory!') MSGTYPE(*ESCAPE)
ENDDO

/* **** */
/* 디렉토리의 내용 읽기 */
/* **** */
CALLPRC PRC('readdir') PARM((&HANDLE *BYVAL)) RTNVAL(&ENTRY)
```

## Notes:

I/O API terminated string 가 , opendir (Open Directory) API input null-  
 가  
 &HANDLE value가 &NULLPTR 가 null , CL COND IF . opendir \*NONE special local pointer value &HANDLE  
 , it works!  
 opendir API가 , “by value” ( input only) readdir (Read Directory)  
 API

## 디렉토리 내용 읽기 (계속)

```

/* .TAB로 끝나는 모든 stream file을 PF TRANMAST로 복사한다 */
DOWHILE COND(&ENTRY *NE &NULLPTR)
  CHGVAR VAR(&STMF) VALUE(%SST(&NAME 1 &NAMELEN))
/* 파일 이름에서 마지막 네 글자를 추출해낸다 */
  IF (&NAMELEN *GE 4) DO
    CHGVAR VAR(&POS) VALUE(&NAMELEN - 3)
    CHGVAR VAR(&EXT) VALUE(%SST(&STMF &POS 4))
  ENDDO
  ELSE +
    CHGVAR VAR(&EXT) VALUE(' ')
/* 파일의 확장자가 '.TAB'이면, physical file로 복사하고 + */
/* stream file은 삭제한다 (다시 처리되지 않도록) */
  IF ((&EXT *EQ '.TAB') *OR (&EXT *EQ '.tab')) DO
    CPYFRMIMPF FROMSTMF(&STMF) TOFILE(TRANMAST) MBROPT(*ADD) +
      RCDDL(*CRLF) FLDDL(*TAB) RPLNULLVAL(*FLDDFT)
    DEL OBJLNK(&STMF)
  ENDDO
  CALLPRC PRC('readdir') PARM((&HANDLE *BYVAL)) RTNVAL(&ENTRY)
ENDDO

```

## Notes:

DOWHILE loop은 디렉토리의 모든 entry들(예를 들어 디렉토리 내의 모든 파일들)을 읽을 때까지 계속 readdir API를 호출할 것이다.

Loop 안에서 파일의 이름을 확인하기 위해서 readdir이 반환한 directory entry 에 매핑된 \*DEFINED 변수를 사용한다. 관심이 있는 파일이면, CPYFRMIMPF (Copy from Import File) 명령어를 이용해서 데이터를 데이터베이스 파일로 복사한다. 테이블의 데이터는 모든 파일들의 정보를 가지고 있어야 하므로 계속 추가(append)된다.

모든 .TAB 파일들을 복사하고 나면, DEL (Delete Object) 명령어를 이용해 삭제한다.

## 디렉토리 내용 읽기 (계속)

```
/* **** */
/* 디렉토리 닫기 */
/* **** */
CALLPRC PRC('closedir') PARM((&HANDLE *BYVAL)) +
RTNVAL(&SUCCESS)

/* **** */
/* Physical file에 있는 새로운 트랜잭션을 처리하기 위해 */
/* RPG 프로그램을 호출한다 */
CALL PGM(PROCESS)

ENDPGM
```

## Notes:

디렉토리의 모든 entry가 처리되고 난 후에만, closedir (Close Directory) API가 호출되어 디렉토리를 unlink 한다.

Scott은 이제 ILE RPG 프로그램을 호출하여 physical file의 데이터를 처리한다.

## “일반적인” CL 명령어

- 새로운 OS release 마다 새로운 CL 명령어 및 기존 명령어의 변경 있음
- V5R3:
  - 57개의 새로운 CL 명령어
  - 247개의 변경된 CL 명령어
- V5R4:
  - 43개의 새로운 CL 명령어
  - 151개의 변경된 CL 명령어

## Notes:

AS/400, iSeries 및 System i5의 운영체제 및 라이선스 프로그램에 포함되어 있는 CL 명령어들에 대해 지속적인 투자가 있었습니다. Callable API(application programming interfaces)가 늘어나면서 CL 명령어의 증가는 감소하고 있다. 새 CL 명령어나 변경된 CL 명령어가 없는 AS/400, iSeries, System i5는 없었다. 또한 “Work with” 유형의 명령어를 줄여서, iSeries Navigator와 Management Central과 같은 그래픽 시스템 툴을 이용한 시스템 관리를 촉진하고 있다.

V5R3에서는 새로운 CL 명령어에 투자하는 한편 기존의 풍부한 CL 명령어들을 더욱 향상시켰다. 현재까지, 50개가 넘는 새로운 CL 명령어가 추가 되었고, 그의 네 배가 넘는 CL 명령어들이 갱신되었습니다.

V5R4에서도 이와 마찬가지로 새로운 명령어가 여러 개 추가되었고, 그의 몇 배에 해당되는 명령어들이 갱신되었습니다.

V5R3에서부터 달라진 점이라면 새로 추가되거나 변경된 명령어들 중 다수가 CL 프로그램 내에서만 사용 가능한 것들이라는 점입니다. CL은 어플리케이션 개발 언어로서 계속 향상되고 있습니다.

# 길어진 \*CHAR 변수



- TYPE(\*CHAR)의 예전 제한은 9999 바이트
- TYPE(\*CHAR)의 새 제한은 32767 바이트
- DCLF did not generate CL variables for character fields longer than 9999 bytes in a record format (but will in V5R4)
- PARM, ELEM 그리고 QUAL 명령어 정의문에서 사용하는 TYPE(\*CHAR)와 TYPE(\*PNAME)은 5000 바이트
- VALUE (on DCL)는 첫 5000 바이트로 제한

## Notes:

V5R3에서 추가된 또 하나의 기능은 보다 긴 character CL 변수를 선언할 수 있게 됐다는 것이다.

V5R3 이전까지의 제한은 9999 바이트. 이 제한이 32767 바이트로 V5R3에서 증가되었다.

V5R3 이전까지 DCL의 VALUE 파라미터의 제한은 5000 바이트이며, 이는 변경되지 않았다.

Character fields in a record format referenced by a DCLF command that were longer than 9999 bytes did not generate CL variables in V5R3, but will generate TYPE(\*CHAR) CL variables in V5R4.

# More Parameters



- V5R2까지 PGM과 TFRCTL 명령어의 파라미터 개수 제한은 40개, CALL 명령어의 제한은 99개
- V5R3는 PGM, CALL, TFRCTL 모두 파라미터 255개 지원
- CALLPRC (ILE CL 프로시저에서만 지원)의 파라미터 개수는 300개
- CL 명령어의 PARM 문은 75개에서 99개로 증가

## Notes:

또 한 가지의 제약은 CL 프로그램으로 전달되거나 CL 프로그램이 다른 프로그램을 호출(또는 control을 전달)할 때 CL 프로그램이 내보낼 수 있는 파라미터의 개수였다. 이것에 대한 제한은 여러 해 동안 40개였다.

V5R2에서 CALL 명령어의 파라미터는 99개로 변경되었으며, V5R3에서 모든 파라미터의 제한이 255개로 증가하였다.

이와 관련된 변경 사항으로, CL 명령어에 정의할 수 있는 파라미터의 개수가 75개에서 99개로 늘었다.

# “by value”로 파라미터 전달하기



- CALLPRC (Call Procedure) 명령어는 ILE CL 프로시저에서 다른 ILE 프로시저의 호출을 지원
- V5R3 이전의 CALLPRC는 “by reference”로 파라미터 전달하는 것만을 지원
- 전달되는 파라미터마다 \*BYREF 또는 \*BYVAL 지정 가능
- 이로써 ILC CL에서 많은 MI 및 C 함수들 및 다양한 프로시저 API들을 호출할 수 있게 됨
- 파라미터 개수는 최대 300개

## Notes:

V5R3에 추가된 기능들 중 여태까지의 내용은 CL 컴파일러와 ILE CL 컴파일러 모두에 해당된다. 마지막으로 소개하는 이 기능은 ILE CL에만 해당이 된다. CL 컴파일러는 서비스 프로그램의 일부는 ILE 프로시저를 호출할 수 있는 기능이 없기 때문이다.

ILE 컴파일러 중 하나라도 써봤다면, IBM이 수천 개의 procedure-level API들을 제공한다는 것을 알고 있을 것이다. 이 프로시저 중 다수는 파라미터가 “by value”로 전달되어야 한다. 하지만 CL은 “by reference”로만 전달 가능했다.

V5R3에서 CALLPRC 명령어는 “by reference”와 “by value” 중 어떤 방법으로 파라미터가 전달될 것인지 선택할 수 있도록 했다.

# V5R2에서 V5R3 CL 수행하기

- 보통 새로운 CL 기능을 CL 프로시저에서 사용하려면(CL 프로그램 또는 CL 모듈) TGTRLS 파라미터가 \*CURRENT로 설정되어 있어야 한다.
- V5R3은 뭔가 다르다
- CL 개발팀은 CL 프로그래머들이 새로운 CL 컴파일러 기능을 사용하는 데 장애물을 없애길 원했다
- V5R2와 V5R3에 PTF를 적용하면, 새로운 CL 컴파일러 기능(복수 개의 DCLF 기능 제외)들을 V5R3에서 사용하고 TGTRLS(V5R2M0)로 지정해서 컴파일할 수 있도록 허용

# PTF Numbers

- V5R2 (CL runtime)
  - SI13416
  - SI13417
  
- V5R3 (base OS and option 9(\*PRV CL Compiler Support))
  - SI13505
  - SI13508
  - SI13509

## 기능 향상은 계속됩니다...

- V3R1의 ILE CL 컴파일러의 발표 이후 V5R3는 CL 컴파일러의 기능 향상에 있어 가장 획기적인 릴리스
  - System/38 이루 CL에 가장 많은 신기능 추가
- 하지만 **V5R3**은 시작인 분
- **V5R4**에 보다 더 많은 기능 추가
- 그 이후에도 추가 기능 준비 중

## Notes:

여기까지가 V5R3에서 추가된 CL 컴파일러 기능입니다. 하지만 더 있습니다!

다음 자료들은 i5/OS V5R4에서 추가된 기능들을 설명합니다.

V5R4로 CL에 대한 투자가 끝나는 것은 아닙니다. V5R4 이후에 개발될 추가적인 기능들에 대한 평가가 진행 중입니다.

## 요약

- V5R3에서 CL이 향상되면서 보다 나은 어플리케이션 개발 언어가 되었으며 CL 프로그램으로 보다 많은 일을 할 수 있게 되었다.
- V5R4에서 추가된 기능으로 CL이 할 수 있는 일은 무한해졌다.
- 향상된 기능으로 CL 프로그램 또는 ILE CL 프로시저로 할 수 있는 것의 아주 작은 일부만을 예제로 제시했습니다.