

webProgramming

함수형 언어

in-hee Kim,
school of Computer Engineering
inhee.kim@hansung.ac.kr



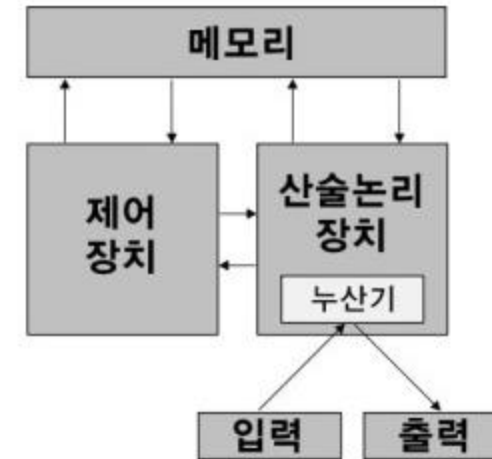
- 명령형 언어의 특징을 알고 문제점을 설명할 수 있다.
- 함수형 언어의 특징을 이해할 수 있다.
- 순수 함수 언어(Lisp)의 문법을 사용해서 Lisp 프로그램을 제작할 수 있다
- 명령형 언어와 함수형 언어를 비교할 수 있다.

명령형 언어의 특징

명령형 언어의 특징

- 폰노이만 컴퓨터 구조(Computer Architecture)에 기반

- 메인 메모리, CPU 가 있는 구조
- 프로그램의 작업 단위는 문장(statement)



폰 노이만 구조

프로세서와 메모리로 구성되는 폰 노이만 구조



요한 루트비히 폰 노이만
Johann Ludwig von
Neumann, 1903~1957)

- 명령형 언어의 특징 : 순차계산, 반복계산

- 변수 : 메모리 셀(자료 저장 공간), 셀마다 이름 부여(변수명)
- 배정 연산(assignment)
- 반복문

프로그램의 작업 단위 비교

- 문장(statement) vs. 식(expression)

```
if x > y then max := x  
           else max := y
```

< 문장(statement) 중심 표현 >



```
max := if x > y then x else y
```

< 식(expression) 중심 표현 >

- 식 지향적(expression-oriented) 언어
- 예) Algol68

```
max := begin int x, y;  
        read((x, y));  
        if x > y then x else y fi  
      end
```



- 프로그램의 정확성을 증명하기 어렵다
- 계산의 진행 과정을 살펴보려면 명령어 실행의 각 단계에서 기억장소(변수)의 내용을 조사해야 한다.
- 변수를 사용했다면 변수의 내용까지 같이 봐야지 어떻게 진행되는지 속사정을 알 수 있다.

→ 참조 투명성(referential transparency) 문제

<해결방안> 함수형 프로그래밍 언어 : 변수, 배정문, 반복문이 없는 언어

함수형 언어의 특징

- lambda calculus(수학 이론에서 출발)
 - 프로그램의 작업단위가 expression 중심
 - 변수, 배정문, 반복문 없음
 - 참조 투명성 확보
 - 의미적으로 우아하고, 구문이 간결하며, 표현력이 강하다
→ 프로그램의 생산성을 높일 수 있다.



- 함수형 프로그래밍
 - $\text{square}(x) = x * x$ (단, x 는 정수)

```
λ x . x * x
( λ x . x * x ) 2
( λ x . x * x ) 2 ≡ 2 * 2 ≡ 4
```

```
(lambda (x) (* x x)) → anonymous function
```

```
(defun (square (lambda (x) (* x x)))) → square 함수명 정의(기본 구조)
```


명령형 언어와 함수형 언어 비교

- **명령형 언어**

- 기계 의존적 : 폰노이만 구조
- 실행시간 효율적
- 기계의 세부사항에도 주의를 기울여야 함
 - 메모리 셀 지정, 값 부여, 반복 연산 등

- **함수형 언어**

- 단순하고 일정한 자료구조 : 리스트, 배열 등
- 기억장치 셀에 관계없이 자료구조 설계
- 함수 적용에 의해 값이 생성되며, 이 값은 다른 함수로 전달
- 더 높은 수준의 프로그래밍 방식 : 프로그래밍이 용이하다

간단한 순수 함수 언어 : Lisp(List processor)

- 1958년 미국 MIT의 존매카시 개발 : 인공지능 연구 프로젝트에서 Fortran에서 목록 작업 수행하는 서브루틴의 패키지로 Fortran List Processing Language을 구현. 이것이 모태가 됨
- 리스트 형태로 데이터 처리
- 데이터와 프로그램이 S-expression(S식). 일반화된 리스트 형태로 기술
→ 프로그램이 데이터처럼 취급됨
- 람다 대수학에 이론적 근거를 두고 다양한 함수를 어셈블리 언어로 컴파일링
- 함수 자체를 데이터로 취급하며, 함수 재귀호출(recursive call)을 사용함
- 일련의 지식들을 규칙적으로 모아서 사용하는 인공지능 분야에 많이 사용
- 식의 전개, 인수분해, 미적분, 정리증명, 게임문제, 자연어 처리 등에 적합함




common Lisp 다운로드

- URL : <http://clisp.org>
- e-class 에서 다운받기
 - clisp-2.49-win32-mingw-big.exe

Welcome to [CLISP](#)

This is [GNU CLISP](#) - an [ANSI Common Lisp](#) Implementation



Current version: 2.49 (2010-07-07) [NEWS](#)

About CLISP	Get CLISP
<p>implementation of a great language!</p> <p>Common Lisp and CLISP</p> <p>Common Lisp such a great programming environment?</p> <p>for UNIX</p> <p>Options for other platforms are very similar (see <code>clisp.html</code> in your build directory)</p> <p>Notes [TOC]</p> <p>CLISP implements and extends the ANSI standard INCITS 226-1994 (R1999) "Information Programming Language - Common Lisp", available as the Common Lisp</p>	<p>Home</p> <ul style="list-style-type: none">• http://clisp.org/• http://clisp.sourceforge.net/• http://www.gnu.org/software/clisp/ <p>Our official distribution sites</p> <ul style="list-style-type: none">• http://SF• ftp://ftp.gnu.org/pub/gnu/clisp/• http://ftp.gnu.org/pub/gnu/clisp/

Lisp 실습

함수형 언어 : Lisp (List Processor)

- LISP 객체 : atom, list, string

- atom : 숫자, 문자, 문자열

```
> 10  
> A  
> PEOPLE
```

- List : 원자 혹은 다른 리스트의 집합(공백으로 원자 분리, 괄호로 묶는다).

```
> (1 2 3 (4 5) (6 7))  
> ((MEAT CHICKEN) (SPINACH POTATOES TOMATOES) (WATER))
```

- String

```
> "Hello, World!!!"
```

- 연산 : 전위 표기법

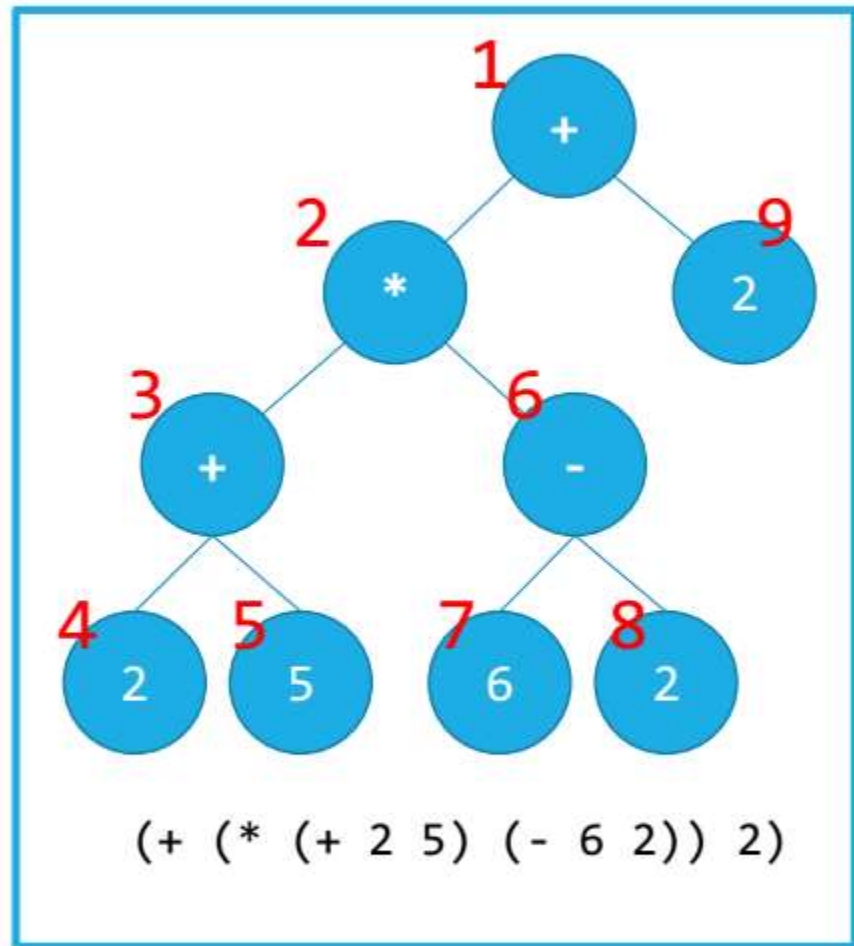
```
> (+ A B)
```



- > 999999 ; 999999 (integers)
- > #b111 ; 7 (binary)
- > #o111 ; 73 (octal)
- > #x111 ; 273 (hexadecimal)
- > (+ 1 2) ; 3
- > (quote (+ 1 2)) ; (+ 1 2)
- > (quote a) ; A
- > ' (+ 1 2) ; (+ 1 2)
- > 'a ; A



> (+ 1 1)	; 2
> (+ 2 5 10)	; 17
> (expt 2 3)	; 8
> (mod 5 2)	; 1
> (/ 12 4)	; 3
> (/ 1 3)	; 1 / 3
> (+ (* (+ 2 5) (- 6 2)) 2)	; 30





- > #\u0041 ; #\u0041 (문자)
- > #\u03BB ; #\u03BB (유니코드)
- > (concatenate 'string "Hello, " "world!!!!") ; "Hello, world!!!"
- > (concatenate 'list '(1 2) '(3 4)) ; '(1 2 3 4)
- > (elt "Hansung" 3) ; #\u0041

atom/list 추출(1)

- **Lisp 객체** : atom, list, string

- **car** : 첫번째 원자 추출

(car '(1 2 3)) ; 1

- **cdr** : 첫번째 원자를 제외한 나머지 리스트 추출

(cdr '(1 2 3)) ; (2 3)

- **cons** : 첫번째 원자를 포함시킨 새로운 리스트 생성

(cons '(1 2) '(3 4)) ; ((1 2) 3 4)

- **list** : 여러 개의 원자를 리스트로 생성

(list 1 2 3 4) ; (1 2 3 4)

'(1 2 3 4) ; (1 2 3 4)

```
> (cons 'comp 'webp )  
(COMP . WEBP)
```

```
> (car (cons 'comp 'webp ))  
COMP
```

```
> (cdr (cons 'comp 'webp ))  
WEBP
```

```
> (cons 1 (cons 2 (cons 3 nil) ) )  
(1 2 3)
```

```
> (cons 4 '(1 2 3))  
(4 1 2 3)
```



```
> '((a b)(c d)(e f))  
((A B) (C D) (E F))
```

```
> (car '((a b)(c d)(e f)) )  
(A B)
```

```
> (car (car  
      '((a b)(c d)(e f))  
    ))  
A
```

```
> (cdr '((a b)(c d)(e f)) )  
((C D) (E F))
```

```
> (car (cdr  
      (car (cdr  
            '((a b)(c d)(e f))  
          ))  
    ))  
D
```

```
> (cadadr '((a b)(c d)(e f))) ; 간략화  
D
```

```
> (caddr '((a b)(c d)(e f)) )  
(E F)
```

```
> (cadr (caddr '((a b)(c d)(e f)) ))  
F
```

```
> (cons  
    (caddr '((a b)(c d)(e f)))  
    '(x y z)  
  )  
((E F) X Y Z)
```

```
> (mapcar #' + '(1 2 3) '(10 20 30))  
(11 22 33)
```

```
> (remove-if #'evenp '(8 2 4 5 6 3))  
(5 3)
```

```
> (remove-if-not #'evenp '(8 2 4 5 6 3))  
(8 2 4 6)
```



```
> (list 1 2 3)
(1 2 3)

> (quote (1 2 3))
(1 2 3)

> '(1 2 3)
(1 2 3)

> (append
    '(1 2 3) '(11) '(22 33 44))
(1 2 3 11 22 33 44)

> (concatenate 'list
    '(1 2 3) '(11) '(22 33 44))
(1 2 3 11 22 33 44)
```

```
> (length
    (append '(1 2 3) '(11) '(22 33 44)))
7

> (nth 0
    (append '(1 2 3) '(11) '(22 33 44)))
1

> (nth 5
    (append '(1 2 3) '(11) '(22 33 44)))
33

> (reverse
    (append '(1 2 3) '(11) '(22 33 44)))
(44 33 22 11 3 2 1)
```

```
> (format nil "~a, ~a!!!" "Hello" "World")  
"Hello, World!!!"
```

```
> (format t "The value is ~a" 30)  
The value is 30
```

```
> (format t "The value is ~a" "good")  
The value is good
```

```
> (format t "The value is ~a" 123)  
The value is 123
```

```
> (format t "~:d" 1000000)  
1,000,000
```

```
> (format t "~:@d" 1000000)  
+1,000,000
```

```
> (format t "~f" pi)  
3.14159263535897932385
```

```
> (format t "~5f" pi)  
3.142
```

```
> (format t "~5f" (/ 13 2))  
6.5
```

```
> (format t "~,5f" (/ 13 2))  
6.50000
```



```
> (defun hello() "Hello World")
```

```
> (hello)
```

"Hello World"

```
> (defun hello(name)
  (format nil "Hello, ~a" name)
)
```

```
> (hello "Henry")
```

"Hello, Henry"

```
> (defun sum(n1 n2) (+ n1 n2))
```

```
> (sum 1 2)
```

3

```
> (defun hello (name &optional from)
  (if from
    (format t "Hello, ~a, from ~a" name from)
    (format t "hello, ~a" name)
  )
)
```

```
> (hello "Kim")
```

Hello, Kim

```
> (hello "Kim" "Seoul")
```

Hello, Kim, from Seoul

```
> (defun hello
  (name &optional (from "The world"))
  (format nil "Hello, ~a, from ~a" name from)
)
```

```
> (hello "Lee")
```

"Hello, Lee, from The world"



```
> ( defstruct family name age )  
> ( defparameter f[0] (make-family :name "henry" :age 4 ) )  
> ( defparameter f[1] (make-family :name "emma" :age 8 ) )
```

```
> f[0]  
#S ( FAMILY :NAME "henry" :AGE 4 )
```

```
> ( family-p f[0] )  
T
```

```
> ( family-name f[0] )  
"henry"
```

```
> ( family-age f[1] )  
8
```



- > (setf y 2) ; **y=2**
- > (setf y (+ 2 5)) ; **y=7**
- > (setf y quote(+ 2 5)) ; **(+ 2 5)**
- > (setf y '(+ 2 5)) ; **(+ 2 5)**

- > (setf x 5 y 10)
- > (list x y) ; **(5 10)**
- > (setf x (+ x 10)) ; **15**

- > (setf x (random 10)) ; **0~9 random**



```
> (setf x 34 y 9)
```

```
> (> x y)
```

```
T
```

```
> (< x y)
```

```
NIL
```

```
> (if (> x y) x y)
```

```
34
```

```
> (dotimes (x 5) ; 5번 반복
```

```
  (format t "~3a" x) ; 3칸 왼쪽 정렬
```

```
)
```

```
0 1 2 3 4
```

```
> (dotimes (x 5) ; 5번 반복
```

```
  (format t "~3d" x) ; 3칸 오른쪽 정렬
```

```
)
```

```
0 1 2 3 4
```

```
> (dotimes (x 25)
```

```
  ; 25번 반복
```

```
  (if (= 0 (mod x 5))
```

```
    ; 5 나눈 나머지 0이면
```

```
    (format t "~3d" x)
```

```
  )
```

```
)
```

```
0 5 10 15 20
```

```
> (dotimes (x 3)
```

```
  ; 3번 반복
```

```
  (dotimes (y 5)
```

```
    ; 5번 반복
```

```
    (format t "~3d" y)
```

```
  )
```

```
  (format t "~%")
```

```
  ; 라인 개행
```

```
)
```

```
0 1 2 3 4
```

```
0 1 2 3 4
```

```
0 1 2 3 4
```




```
> (dotimes (x 2)
  (dotimes (y 5)
    (format t "~3d" y)
  )
  (format t "~%")
)
0 1 2 3 4
0 1 2 3 4
```

```
> (dotimes (x 25)
  (if (= 0 (mod x 5))
    (format t "~3d" x)
  )
)
0 5 10 15 20
```

```
> (do
  ((x 0 (+ 2 x))) ; (초기값 (증감))
  ((>= x 10)) ; (최종값)
  (print x) ; 실행문
)
0
2
4
6
8
```

```
> (do
  ((x 0 (+ 2 x))) ; (초기값 (증감))
  ((>= x 25)) ; (최종값)
  (format t "~3d" x) ; 실행문
  (if (= 0 (mod x 10))
    (format t "~%")
  )
)
0
2 4 6 8 10
12 14 16 18 20
22 24
```

