Problem Solving

Review: C Programming Language

Instructor: Jae-Pil Heo (허재필)

Review: C-Programming Language

- Flow of control
- Function
- Data type
- Pointer
- Array
- Recursion
- Structure

C-Programming Framework

- Syntax

```
#include<header_file>

int  main(arguments)
{
     statement1
     statement2
          :
      return 0;
}
```

[Ex]

```
#include<stdio.h>

int main(void)
{
  printf("Hello, World!\");

  return 0;
}
```

- Variables
  - A symbolic name associated with a value and whose associated value may be changed
  - Variable type: int, long, float, double, char

Flow of Control

- *if… else* syntax

```
if (expression)
{
        statement1;
        statement2;
        :
}
else
{
        statement1;
        statement2;
        :
}
```

- The nearest rule

x=y=1

x=1, y=2

```
[Ex]
if (x == y)
{
        printf("x is equal to y");
        e_count += 1 ;
}
else
{
        printf("x is not equal to y");
        ne_count += 1 ;
}
```

a=1 b=2

a=1 b≠2

a≠1 b=2

a≠1 b≠2

```
if (a == 1)
   if (b == 2)
      printf("***\n");
else
   printf("###\n");
```

Flow of Control

- *Switch* syntax: multiple conditional statement

```
switch ( expression ) {
        case constant-expression :  statements
        case constant-expression :  statements
        case constant-expression :  statements
        ……
        default :  statements
}
```

grade=3
grade=2
grade=1
grade=0
grade=5

```
[Ex] switch (grade) {
        case 3 :
        case 2 :
        case 1 : printf("Passing\n");   break;
        case 0 : printf("Failing\n");   break;
        default : printf("Illegal grade\n");   break;
}
```

Flow of Control

- Conditional operator syntax

  `expr1 ? expr2 : expr3`

  - Ternary
  - After calculation of *expr1*, *expr2* will be executed if *expr1* is true; otherwise *expr3* will be executed

if-else statement

```
if ( y < z)
   x = y;
else
   x = z;
```

conditional operator

```
x = ( y < z ) ? y : z ;
```

```
( y < z ) ? x=y : x=z ;
```

Flow of Control

- *while* syntax

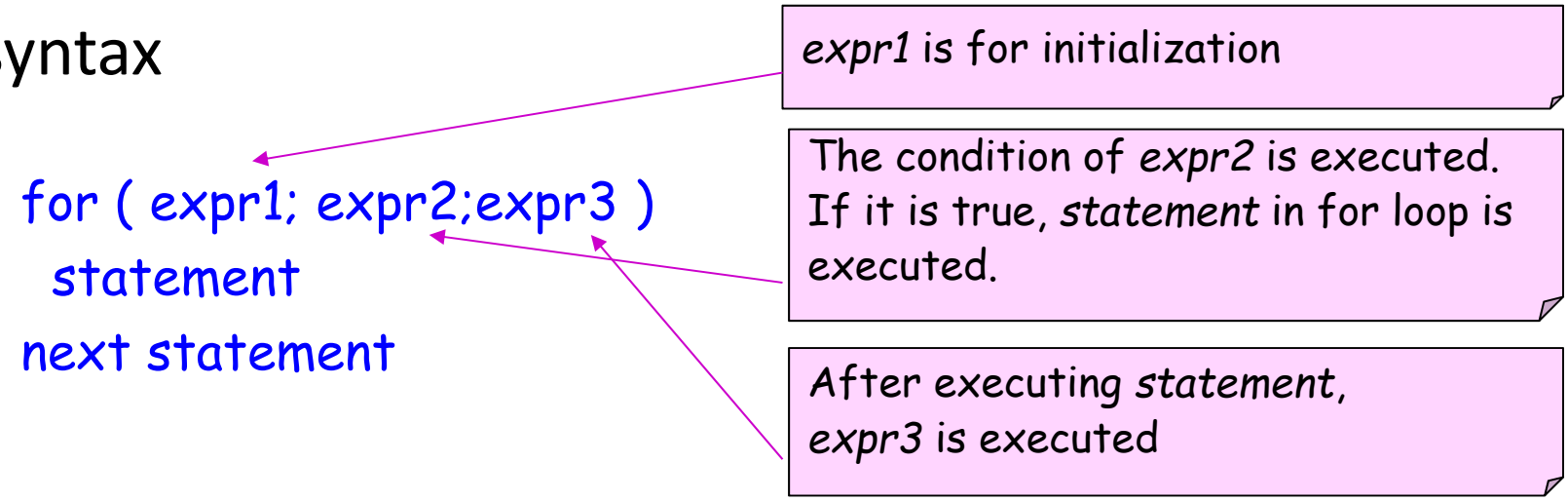
```
while (expr)
  statement
next statement
```

  - After calculation of *expr*, if *expr* is true, *statement* will be executed and the control point will be come back to the beginning of the while statement; otherwise, *next statement* will be executed.

```
[Ex]
i = 1; sum=0;
while ( i <= 10 ) {
  sum += i;                    printf("%d",++i);
  ++i;              }          printf("%d",i++);
```

Flow of Control

- *for* syntax




```
[Ex]
for ( i = 10; i > 0; --i)
  printf(" T minus %d and counting\n", i );
```

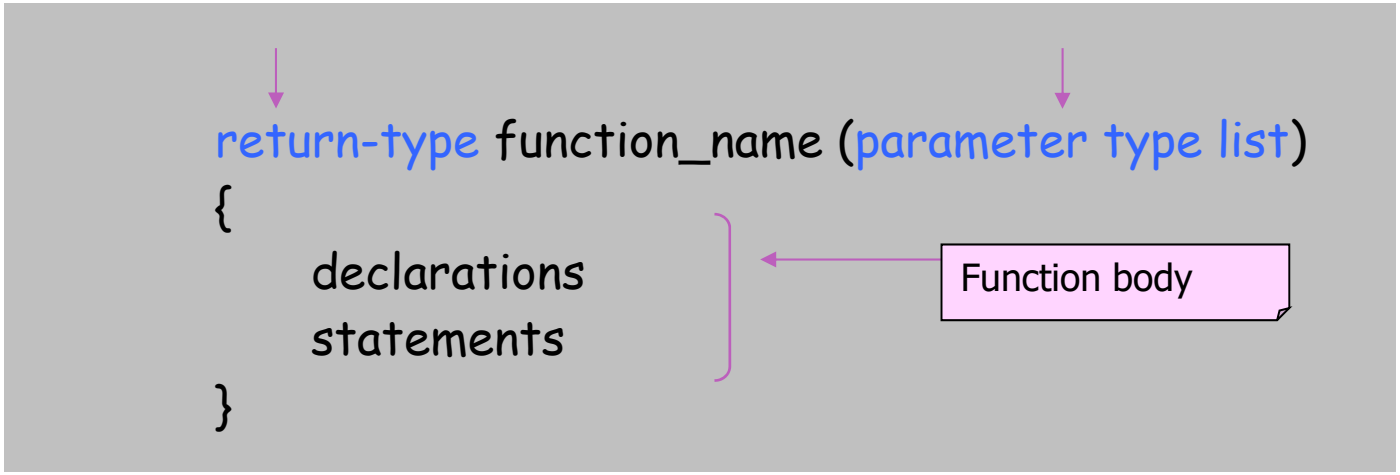
```
for ( ;  ; ++i)
for ( ;  ; )
```

- The above *for* statement is equivalent to the following *while* statement

```
expr1;
while (expr2)  {
  statement
  expr3;
}
next statement
```

Function

- Function definition




```
[Ex]    double power(double x)
        {
            double y = x*x;
            return y;
        }
```

Function

- Function prototype
  - Declaration for using a function
    - **return-type function_name (parameter type list);**

```
[Ex]#include<stdio.h>

      double power(double x);

      int main(void) {

            int y=4;
            double result = power(y);
            printf("sqrt(%d) = %f\n", y, result);
            return 0;
      }
```

Parameter - double type

Argument – int type
There is a promotion of int -> double

double y, int x ?

Function

- Call-by-value

```
[Ex]#include <stdio.h>

    int function(int i, int j) {
        i = 10;
        j = 10;
        printf("in function : i=%d, j=%d \n", i, j);
        return j;
    }

    int main(void) {
        int i = 1;
        int j = 1;
        j = function(i, j);
        printf("in main : i=%d, j=%d \n", i, j);
        return 0;
    }
```

These are stored in a place different from i, j in main()

i, j in main( ) do not change.

j will be assigned a returned value by function( )

```
in function : i=10, j=10
in main : i=1, j=10
```

Data Type

- *char* type
  - 1 byte (8 bits), translated into ASCII, interchangeable with int

```
[Ex]    int c;
        c= 'A'+5;          /* 'A' ASCII code : 65 */
        printf("%c %d\n", c, c);
```

F 70

Data Type

Left Digit(s) \ *Right Digit*	*ASCII* 0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	□	!	“	#	$	%	&	’
4	(	)	*	+	,	–	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Data Type

- Integral type
  - int, short, long, unsigned
- Floating type
  - float, double, long double
- Type conversion: case operator

a is casted to double, yielding (dobule/int) expression.
Then, int is promoted to double.
The final result of (double/double) is double 3.0 / 2.0 = 1.5

```
[Ex]    int a=3, b=2;
        double c = (double) a / b;
        printf("c=%f\n", c);
```

c=1.500000

double c = a/b

int c = a/b

int c = (double)a/b

Pointer

- Declarations

```
data_type * pointer_varaible;
```

```
[Ex]    int *p;
        float *fp;
        p = NULL;
        p = 0;
```

Declaration of fp, a float-type variable whose value is a memory address.

The same expression; point nothing

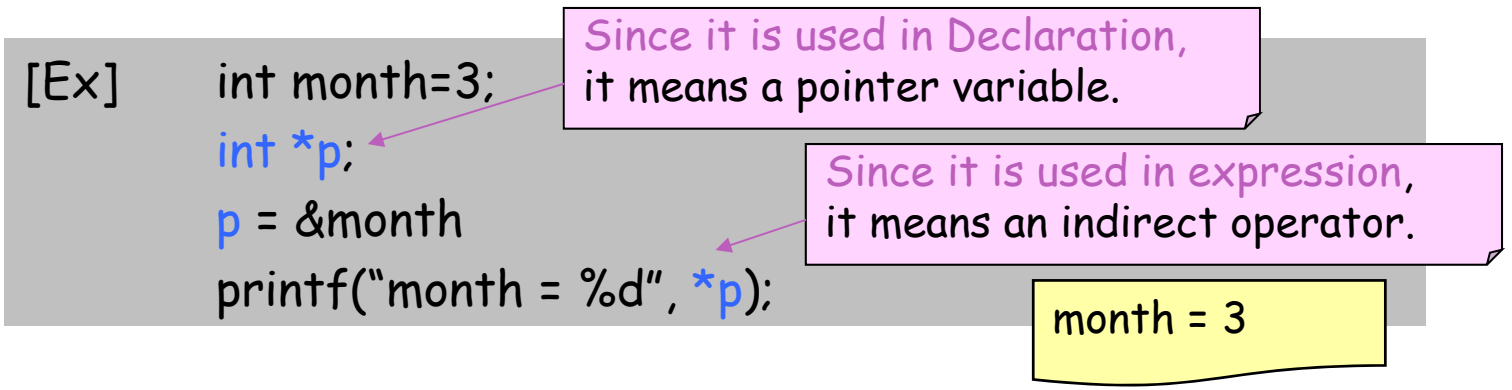
- & (reference) operator
  - "address of" variable

```
[Ex]    int *p;
        int month=3;
        p = &month;
```

Assign a memory address of month to a pointer variable p

Pointer

- * (indirect or dereference) operator
  - Different meaning from * for pointer variable declaration
  - Access a value of a place where a pointer variable points



Pointer

- Call-by-reference

```
[Ex]  void swap(int *p, int *q)  {
                    int temp = *p;
                    *p = *q;
                    *q = temp;
}
```

```
int main(void) {
    int a=3, b=7;
    swap(&a, &b);
    return 0;
}
```

- Call-by-value

```
[Ex]  void swap(int p, int q)  {
                    int temp = p;
                    p = q;
                    q = temp;
}
```

*p,*q

```
int main(void) {
    int a=3, b=7;
    swap(a, b);
    return 0;
}
```

Question

- What is the return type of conditional operation?
  - For example, if(a>1)
  - printf(“%d”,1<2);
  - printf(“%c”, (1<2)+64);
  - printf(“%hd”,1<2);

- The values other than 0 and 1 are allowed?
  - if (2)
  - if (-1)
  - if (1.1)
  - if (0.0)
  - if (0.1)

```
if () {
        printf(“A”);
} else {
        printf(“B”);
}
```

1. A
2. B
3. Error

Array

- Declaration

data_type variable_name[number][number];

Array dimensions
Declaration & Initialization of Arrays
int a[4] = {2, 4, 3, 0};
int b[2][3] = { {1, 6, 4}, {5, 3, 2} };
int c[2][2][3] = { { {1,2,0}, {3,5,4} }, { {9,8,7}, {14,15,16} } };

b[0][1]

c[1][1][1]

c[1][0][0]

Array

- Array access using pointer
  - a[i]: the i-th column of a
  - a is equivalent to &a[0]

Equivalent to a[i]
*(a + i)
*(&a[0]+i)

For a[3][5], equivalent to a[i][j]
*(a[i] + j)
(*(a + i)) [j]
*((*(a + i)) + j)
*(&a[0][0] + 5 * i + j)

Array

- Passing arrays to functions
  - When an array is passed to a function, its address is passed by "call by value."
  - The values of an array is passed by "call by reference."

```
[Ex]
int sum( int a[], int n)
{
  int i, s = 0;

  for ( i = 0; i < n; ++i)
     s += a[ i ];
  return s;
}
```

int a[ ] is equivalent to int *a.

Various ways that sum() might be called	
Invocation	What gets computes and returned
sum(v, 100)	v[0] + v[1] + … + v[99]
sum(v, 88)	
sum(&v[7], k – 7)	v[7] + v[8] + … + v[k -1]
sum(v + 7, 2* k)	v[7] + v[8] + … + v[2 * k + 6]

Array

- Dynamic memory allocation

size of each object

number of objects

size of each object

```
void malloc ( object_size );
void calloc ( n, object_size );
void free(void *ptr);
```

De-allocate a memory block that ptr points

```
#include <stdio.h>
#include <stdlib.h>
int main(void)  {
    int *a, i, n, sum = 0;
    scanf("%d" , &n );
    a = calloc(n, sizeof(int) );        /* get space for n ints */
    for ( i = 0; i < n; ++i )     scanf("%d", &a[ i ] );
    free(a);                            /*free the space */
    return 0;
}
```

calloc(), malloc(), free() belong to stdlib.h

a = malloc(n * sizeof(int));

De-allocate a memory block allocated by calloc()

Recursion

- Recursive problem solving: computing factorial

```
[Ex] /* Recursive version  */
int fact( int n)
{
    if (n <= 1)
        return 1;
    else
        return n * fact(n-1);
}
```

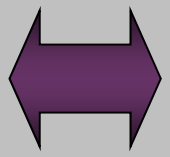
```
[Ex]  /* Iterative version */
int fact( int n )
{
   int result = 1;

   for ( ; n > 1; --n )
        result *= n;
   return result;
}
```

What i = fact(3) returns	
fact (3)	3 * fact (2) 3 * (2 * fact (1)) 3 * (2 * (1)) = 6

Structure

- The difference between array and structure
  - Array
    - All elements in an array should be the same type.
    - We can access individual elements of an array using their index.
  - Structure
    - A structure can consists of elements with different types.
    - Each element has its own name.
    - We can access individual elements of a structure using their name.
- Declaration of structure: collection of members

```
[Ex] struct part{ /* 3-element structure */
        int  number;
        char  name [20] ;
        int  on_hand;
     }  part1;
```



```
struct part {
……
};
struct part  part1;
```

Structure

- Accessing members
  - Struct member operator: "."

```
[Ex] struct part {
        int number;
        char name[20];
        int on_hand;
     } part1;

     part1.number = 258;        /* assignment */
     scanf ("%d", &part1.on_hand); /* reading using scanf() */
     scanf("%s", part1.name);
     part1.on_hand++;           /* increment */
```

Structure

- Accessing members
  - Struct pointer operator: “->”

```
[Ex] typedef struct complex {
        double re;
        double im;
     } complex;

     complex c1, c2, *a=&c1, *b=&c2;
     /* a refers structure c1, b refers c2*/

     a->re = b->re + 2 ;   /* c1.re = c2.re + 2*/
     b->im = a->im - 3;    /* c2.im = c1.im - 3*/

     printf ("value ; %f\n ", a->im);
     scanf("%f", &b->im);
```

Any Question?