Problem Solving Techniques 문제해결

Jinkyu Lee

Dept. of Computer Science and Engineering,
Sungkyunkwan University (SKKU)

Contents

- **C-programming: overview**

Some slides are adapted from Prof. Chang Wook Ahn's slides.

C-programming: overview

- 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;
        :
}
```

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

x=1, y=1
x=1, y=2

- The nearest rule

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

a=1 b=2
a=1 b≠2
a≠1 b=2
a≠1 b≠2

Flow of control

- *Switch* syntax: multiple conditional statement

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

```
[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;
}
```

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

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 conditional operator

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

→

```
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

```
for ( expr1; expr2;expr3 )
  statement
next statement
```

expr1 is for initialization

The condition of *expr2* is executed. If it is true, *statement* in for loop is executed.

After executing *statement*, *expr3* is executed

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

- The above *for* statement is equivalent

to the following *while* statement

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

Function

- Function definition

```
return-type function_name (parameter type list)
{
    declarations
    statements
}
```

Function body

```
[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

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
```

Date 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

Date 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		

Date 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

Date type

- enum (enumeration) type
  - int type, constant

The first element, sun is assigned 0, and then mon, tue, … have 1, 2 …

tag name

```
[Ex]  enum day { sun, mon, tue, wed, thu, fri, sat };

      typedef enum day  day;

      day find_next_day(day d) {
            day next_day;
            next_day =  (day) ( ( (int) d + 1 ) % 7 ) ;
            return  next_day;
      }
```

```
enum day  find_next_day
(enum day d)  {
      ……………………
}
```

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

```
[Ex]    int month=3;
        int *p;
        p = &month
        printf("month = %d", *p);
```

Since it is used in Declaration, it means a pointer variable.

Since it is used in expression, it means an indirect operator.

month = 3

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;
}
```

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

Q & A

- 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