

COSC1101 – Programming Fundamentals

Maham Khan

Lecture - 6

Do not Confuse between C and C++

- C is a structured language
- C++ is an object oriented language
- C++ supports most of the features of structure language hence you may say C is a subset of C++ (with few exceptions that you need not to worry at this stage)

1. Declaration, definition and initialization of variables
2. User input/output functions scanf() and printf()
3. printf () Function
4. Format specifier
5. Field width specifier
6. Escape sequence
7. Scanf() function
8. Address operator
9. For loop
 - Structure of the for loop
 - Initialization, test, increment expression
 - Body of the for loop
 - Operation of the for loop
 - Multiple statements in for loop
 - Multiple initialization in a for loop

Defining and Declaring variables

- A variable declaration only specifies the name and type of the variable without allocating memory to it
- Declarations are important in multi file programs where a variable defined in one file must be referred to in a second file.
- Variable definition on the other hand specifies the name and type of the variable and also sets aside memory for it. When a data is assigned first time to the variable it is defined.

(We will discuss it in detail when study functions)

Initializing variables

Examples

```
int num=2;  
int char='d',  
float length=34.5677;
```

By initializing a variable we provide:

- Declaration of variable
- Definition of a variable
- Initial value loaded in the variable

User Input and output functions

- The `printf()` function is used to display output on the screen
- The `scanf()` function is used to read input from the keyboard
- `printf()` and `scanf()` both are defined in `stdio.h` called header file which we include using:
 `# include <stdio.h>`

The printf() function

- Printf() is a function like main()
- It caused the string written within the quotes “ ” inside printf() function on the screen
- The string which is phrase in quotes is the function argument passed to printf()
- “Hello World” is a string of characters. C compiler recognizes a string when it is enclosed within quotes.

The definition of printf() and scanf() functions

- We have called the function printf() but have not defined it anywhere
- The declaration is done in the header file called stdio.h
- Its definition is provided in a standard Library which we will study later on.
- The linker links our program with the `***.lib` file at the time of linking. This happens for all C library functions
- Remember the name of a function is like variable name also called an identifier.

Exploring the printf() function

- Printing numbers (integers)

```
void main (void)
```

```
{
```

```
    printf ("Printing number: %d", 10);
```

```
}
```

Here printf took two arguments:

Printing number --- argument 1

10 (what is that ?)--- argument 2

what is %d ---- **Format specifier**

Format Specifiers

- Tells the compiler where to put the value in string and what format to use for printing the value
- %d tells the compiler to print 10 as a decimal integer
- Why not write 10 directly inside the string?
Possible !! However, to give printf() more capability of handling different values, we use format specifiers and variables

Different Format Specifiers

Format specifiers	used for
%c	single character
%s	string
%d	signed decimal integer
%f	floating point (decimal notation)
%e	exponential notation
%g	floating point (%f or %e whichever is shorter)
%u	unsigned decimal integer
%x	unsigned hexadecimal
%o	unsigned octal integer
l	prefix used with %d, %u, %x %o to specify long integer e.g. %ld

Field width Specifiers

Format specifier: `%(-)w.nf`

- `f` is a format specifier used for float data type
e.g `%f`
- `.n` means number of digits after decimal point
e.g `%.2f` `27.25` & `%.3f` means `27.250`
- The digit (`w`) preceding the decimal point specifies the amount of space that number will use to get displayed:

e.g	<code>printf ("Age is % 2d" 33);</code>	<code>33</code>
	<code>printf ("Age is % 4d" 33);</code>	<code>- - 33</code>
	<code>printf ("Age is % 6d" 33);</code>	<code>- - - - 33</code>

Field width Specifiers

```
void main (void)
```

```
{
```

```
    printf("8.1f %8.1f %8.1f\n", 3.0, 12.5, 523.3);
```

```
    printf("8.1f %8.1f %8.1f\n", 300.0, 1200.5, 5300.3);
```

```
}
```

3.0 12.5 ' '523.3

300.0 ' '1200.5 ' '5300.3

Field width Specifiers

```
void main (void)
```

```
{
```

```
    printf("-8.1f %-8.1f %-8.1f\n", 3.0, 12.5, 523.3);
```

```
    printf("-8.1f %-8.1f %-8.1f\n", 300.0, 1200.5, 5300.3);
```

```
}
```

3.0' '12.5 ' ' 523.3

300.0 ' ' 1200.5 ' '5300.3

Escape Sequence

- `\n` prints carriage return and linefeed combination
- `\t` tab
- `\b` backspace
- `\r` carriage return
- `\'` single quote
- `\"` double quote
- `\\` backslash

Printing Strings using printf () function

```
void main (void)
{
    printf("%s is %d years old","Ahmed",22);
}
```


Printing characters using printf () function

```
void main (void)
```

```
{
```

```
    printf("%c is pronounced as %s", 'j', "jay");
```

```
}
```

- Single quotes are for character whereas the %c is format specifier
- Double quotes are for strings whereas the %s is format specifier

Scanf() function

```
void main (void)
{
    int userAge;
    printf("Enter the the age: ");
    scanf(" %d", &userAge);
}
```

- scanf() reads in data and stores it in one or more variables.
- The first argument (the control string), contains a series of placeholders
- The remaining arguments to scanf() is a variable name proceed with & sign called address operator

scanf() can read data into multiple variables

```
int a, b,  
float c;  
char d;
```

```
scanf(" %d %d %f %c" , &a, &b, &c, &d);
```

- Input is stored in these variables. Each variable name is preceded by & , &a means “the memory address associated with variable a”
- It uses any whitespace (space, newline or tab) character to delimit inputs in case of multiple inputs
- Format specifiers are like the ones that printf() uses e.g. %d = int, %f = float, %c = char, etc.

Comments

- `/* ----- */` multiline comments
- `//` single line comments

Loops

- Computer has the ability to perform set of instructions repeatedly
- This repetitive operation is performed using loops
- It involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied

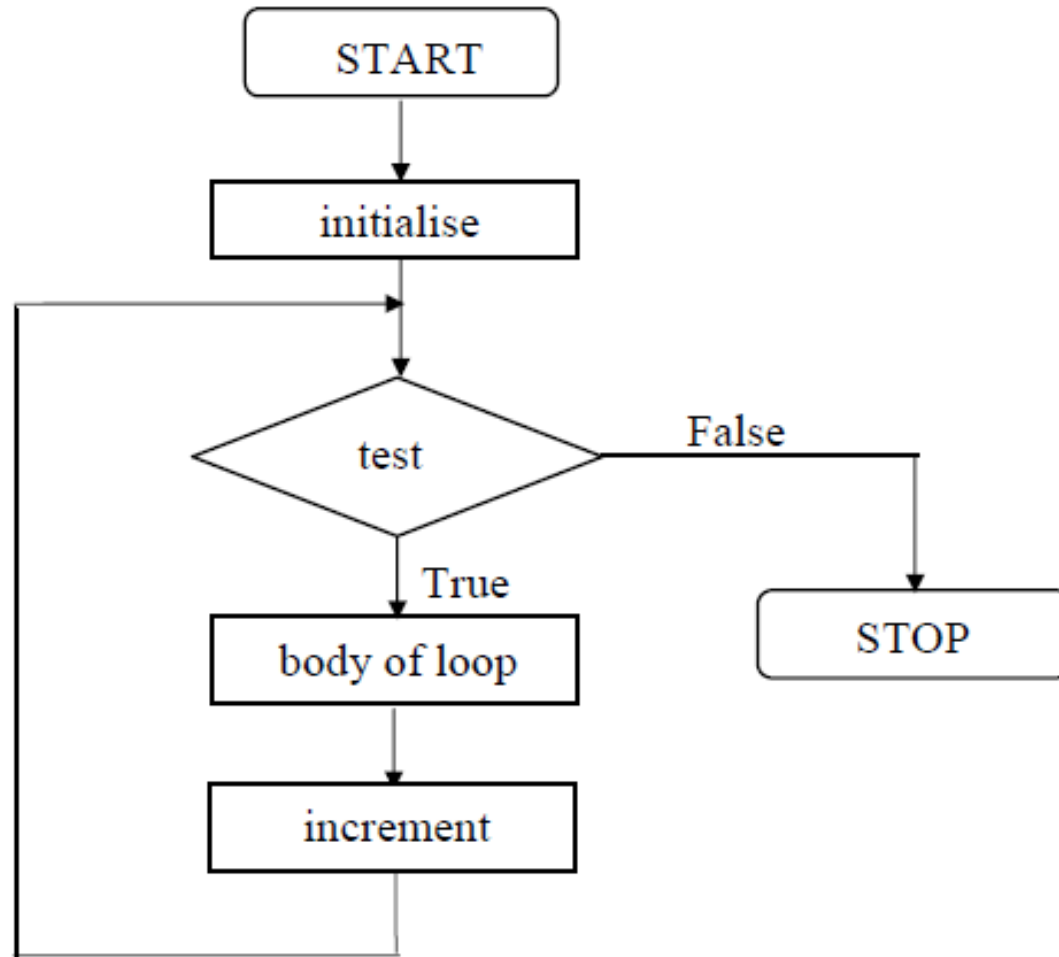
Loops

- There are three types of loops in C
 - For loop
 - While loop
 - Do while loop
- For loop is the most popular looping instruction used by the programmers

For loop

- It specifies three things about a loop in one line
 - Setting a loop counter to an initial value
 - Testing the loop counter to determine whether its value has reached the number of desired repetitions
 - Increasing the value of loop counter each time the program segment within the loop has been executed

Flowchart of the for loop



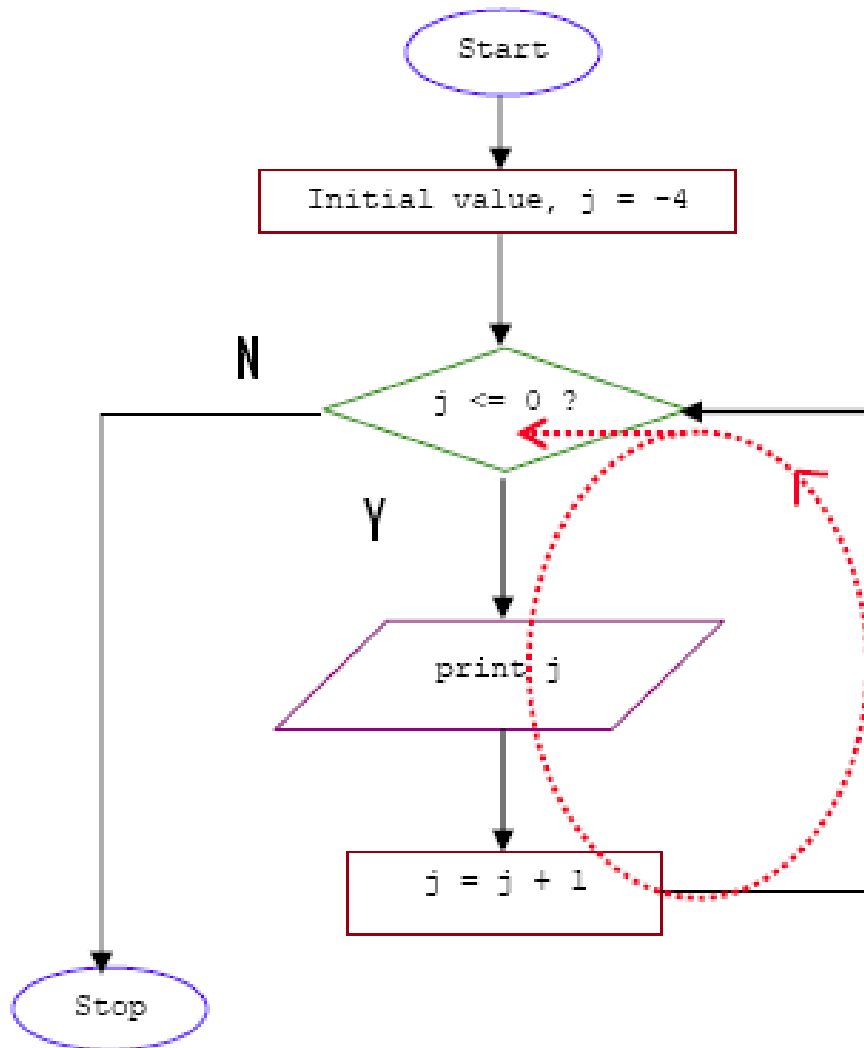
For loop ... continued

The general form of for loop is:

```
for (initialize counter; test counter; increment/decrement  
    counter)  
{  
    Do this;  
    And this;  
    And this;  
}
```

Example:

Print an integer number from -4 to 0



Sample Program

```
void main()
{
    int j;
    for(j = -4; j <= 0 ; j = j + 1)
        printf("%d\n", j);
}
```

print numbers from 1 to 10

```
void main( )  
{  
    int i ;  
    for ( i = 1 ; i <= 10 ; i = i + 1 )  
        printf ( "%d\n", i ) ;  
}
```

- Instead of `i = i + 1`, the statements `i++` or `i += 1` can also be used.

Print numbers from 1 to 10

```
void main( )  
{  
    int i ;  
    for ( i = 1 ; i <= 10 ; )  
    {  
        printf ( "%d\n", i ) ;  
        i = i + 1 ;  
    }  
}
```

- Here, the increment is done within the body of the for loop and not in the for statement. Note that despite of this the semicolon after the condition is necessary.

print numbers from 1 to 10

```
void main( )  
{  
    int i = 1 ;  
    for ( ; i <= 10 ; i = i + 1 )  
        printf ( "%d\n", i ) ;  
}
```

- Here the initialisation is done in the declaration statement itself, but still the semicolon before the condition is necessary.

print numbers from 1 to 10

```
void main( )  
{  
    int i = 1 ;  
    for ( ; i <= 10 ; )  
    {  
        printf ( "%d\n", i ) ;  
        i = i + 1 ;  
    }  
}
```

- Here, neither the initialisation, nor the incrementation is done in the for statement, but still the two semicolons are necessary.

print numbers from 1 to 10

```
void main( )  
{  
    int i ;  
    for ( i = 0 ; i++ < 10 ; )  
        printf ( "%d\n", i ) ;  
}
```

- Here, the comparison as well as the incrementation is done through the same statement, $i++ < 10$. Since the $++$ operator comes after i firstly comparison is done, followed by increment. Note that it is necessary to initialize i to 0.

print numbers from 1 to 10

```
void main( )  
{  
int i ;  
for ( i = 0 ; ++i <= 10 ; )  
    printf ( "%d\n", i ) ;  
}
```

- Here, both, the comparison and the incrementation is done through the same statement, `++i <= 10`. Since `++` precedes `i` firstly incrementation is done, followed by comparison. Note that it is necessary to initialize `i` to 0.

Nesting of Loops

```
/* Demonstration of nested loops */  
void main( )  
{  
    int r, c, sum ;  
    for ( r = 1 ; r <= 3 ; r++ ) /* outer loop */  
    {  
        for ( c = 1 ; c <= 2 ; c++ ) /* inner loop */  
        {  
            sum = r + c ;  
            printf ( "r = %d c = %d sum = %d\n", r, c, sum ) ;  
        }  
    }  
}
```

r = 1 c = 1 sum = 2
r = 1 c = 2 sum = 3
r = 2 c = 1 sum = 3
r = 2 c = 2 sum = 4
r = 3 c = 1 sum = 4
r = 3 c = 2 sum = 5

Multiple Initialisations in the *for Loop*

- The initialisation expression of the for loop can contain more than one statement separated by a comma. For example,
- `for (i = 1, j = 2 ; j <= 10 ; j++)`
- Multiple statements can also be used in the incrementation expression of for loop; i.e., you can increment (or decrement) two or more variables at the same time.
- However, only one expression is allowed in the test expression. This expression may contain several conditions linked together using logical operators.

Solution to the last Practice Quiz

Question -1: Solve the following expressions

a). $3 - 8 / 4$ b). $3 * 4 + 18 / 2$

```
void main( )  
{  
    printf ( "The result of the expression is %d \n",  
            3 - 8 / 4) ;  
    printf ( "The result of the expression is %d \n",  
            (3 * 4 + 18 / 2));  
}
```

Solution to the last Practice Quiz

- Question - 2: If $a=2$, $b=4$, $c=6$, $d=8$, then what will be the result of the following expression?

- $x = a * b + c * d / 4;$

```
void main( )
```

```
{
```

```
    int a =2, b=4, c=6, d=8;
```

```
    printf ( "The result of the expression is  %d \n",
```

```
                x = a * b + c * d / 4) ;
```

```
}
```