# National University of Computer and Emerging Sciences

## CL-118 Programming Fundamentals Lab

### *Lab Manual* 04

---

**Objective:** Introduction to Basic Program structure, data types, operators and math.h library functions

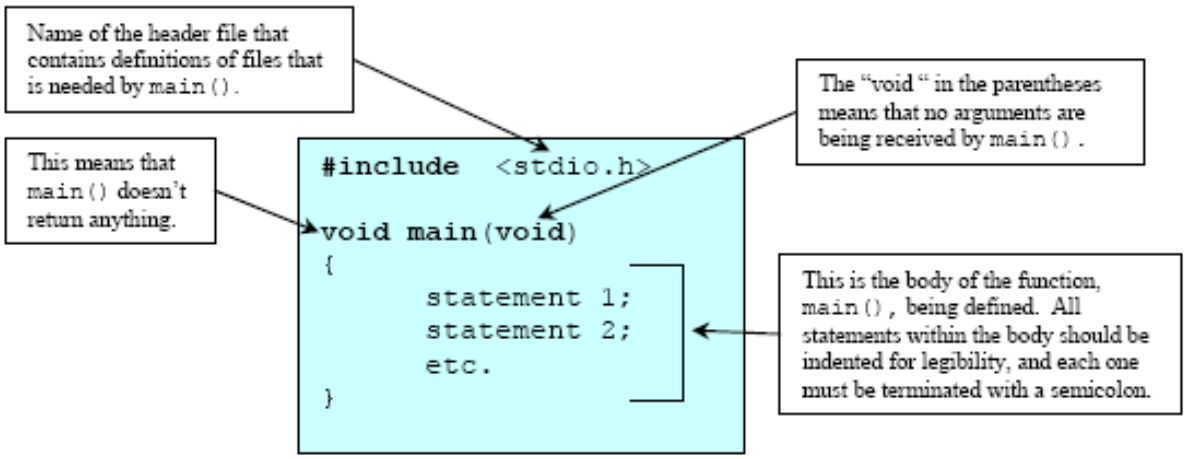
## Compiler

Computer programs are written using high-level programming languages. The source code is converted into machine-understandable machine code. A compiler is used for this conversion. Compiler is a translator that converts the source code from high-level programming language to a lower level machine language in order to create an executable program.

## IDE

IDE stands for **Integrated Development Environment**. It is a software application that provides facilities for developing software. It consists of tools such as source code editor, automation tools, and debugger. Most IDEs have compilers and interpreters. Therefore, it is easier to write the code and compile it. Some IDEs support various languages.

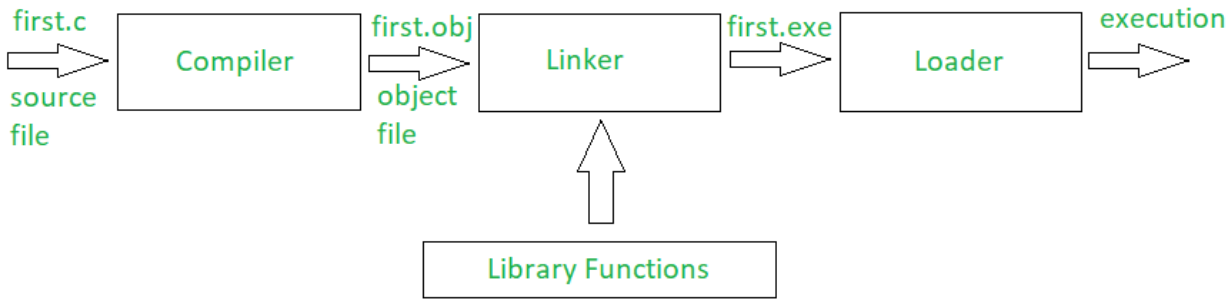
## C - Program Structure




Program 1: Basic Structure of the C/C++ program

# HOW C PROGRAM IS COMPILED?

C program file is compiled and executed, the compiler generates some files with the same name as that of the C program file but with different extensions.




1. Suppose a program file is named, first.c.
2. The file first.c is called the source file which keeps the code of the program.
3. When we compile the file, the C compiler looks for errors.
4. If the C compiler reports no error, then it stores the file as a .obj file of the same name, called the object file. So, here it will create the first.obj.
5. This .obj file is not executable.
6. The process is continued by the Linker which finally gives a .exe file which is executable.

**Linker**: First of all, **library functions** are not a part of any C program but of the C software. Thus, the compiler doesn't know the operation of any function, whether it be printf or scanf. The definitions of these functions are stored in their respective library which the compiler should be able to link. This is what the Linker does. So, when we write #include, it includes stdio.h library which gives access to Standard Input and Output. The linker links the object files to the library functions and the program becomes a .exe file. Here, first.exe will be created which is in an executable format.

**Loader**: Whenever we give the command to execute a particular program, the loader comes into work. The loader will load the .exe file in RAM and inform the CPU with the starting point of the address where this program is loaded.

## Variables

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

```
int playerScore = 95;
    char ch = 'a';
// some code
ch = 'l';
```

## Rules for naming a variable

- A variable name can have only letters (both uppercase and lowercase letters), digits and underscore.
- The first letter of a variable should be either a letter or an underscore.
- There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.
- **Note:** You should always try to give meaningful names to variables. For example: firstNameis a better variable name than fn

## C - DATATYPES:

Variables are classified according to their data type, which determines the kind of information that May be stored in them.

| Data Type | Description | C-language Keyword | Format Specifier |
|---|---|---|---|
| Integer | Integers are whole numbers that can have both positive and negative values but no decimal values. | int | %d or %i |
| Float | Floating type variables can hold real numbers precision of 6 digits | float | %f |
| Double | can hold real numbers with precision of | double | %f |
| Character | Character data type allows a variable to store only one character. | char | %c |

## Constants:

If you want to define a variable whose value cannot be changed, you can use the constkeyword. This will create a constant.
For example:

```
const double PI = 3.14;
```
Notice keyword const.

Here, PI is a symbolic constant; its value cannot be changed.
```
const double PI = 3.14;
PI = 2.9; //Error
```

## Libraries for managing Input/output

*stdlib* is the standard C library for input-output operations. While dealing with input-output operations in C, two important streams play their role. These are:

1. Standard Input (stdin) used for taking input from devices such as the keyboard as a data stream.

2. Standard Output (stdout) used for giving output to a device such as a monitor.

## Display a value of variable on console

C Output Example

```
#include <stdio.h>
int main()
{
   // Displays the string inside quotations
   printf ("C Programming");
   return 0;
}
```

**Syntax**

```
printf("format specifier ",<variable name>);
```

# Format specifiers

It defines the type of data to be printed on standard output. Whether to print formatted output or to take formatted input we need format specifiers. Format specifiers are also called as format string. Format specifier is used during input and output. It is a way to tell the compiler what type of data is in a variable during taking input using scanf () or printing using printf ().

| Format specifiers | Description |
|---|---|
| %d | Integer Format Specifier |
| %f | Float Format Specifier |
| %c | Character Format Specifier |
| %s | String Format Specifier |
| %u | Unsigned Integer Format Specifier |
| %ld | Long Int Format Specifier |

**For example:** to show integer value

```
printf ( "%d" , age);
```

```
#include <stdio.h>
int main()
{
   int testInteger = 5;
   printf("Number = %d", testInteger);
   return 0;}
```

**Receiving Input**

In above program we have fixed value of age but if we want to take this value from user then we use scanf () receives them from the keyboard. This is illustrated in the program shown below.

**Syntax**

```
scanf("format specifier ",&<variable name>);
```

**For example**: to show integer value

```
scanf ( "%d" ,& age);
```

**Note** that the ampersand (&) before the variables in the scanf( ) function is a must. & is an 'Address of' operator. It gives the location number used by the variable in memory. When we say &a, we are telling scanf( ) at which memory location should it store the value supplied by the user from the keyboard.

# OPERATORS:

## C Arithmetic Operators

There are many operators in C for manipulating data which include arithmetic Operators, Relational Operators, Logical operators and many more which will be discussed accordingly. Some of the fundamental operators are:

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands. | A + B = 30 |
| − | Subtracts second operand from the first. | A − B = -10 |
| * | Multiplies both operands. | A * B = 200 |
| / | Divides numerator by de-numerator. | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division. | B % A = 0 |
| ++ | Increment operator increases the integer value by one. | A++ = 11 |
| -- | Decrement operator decreases the integer value by one. | A-- = 9 |

**Example code:**

```
// Working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
```

```
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

## C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

| Operator | Meaning of Operator | Example |
|---|---|---|
| == | Equal to | `5 == 3` is evaluated to 0 |
| > | Greater than | `5 > 3` is evaluated to 1 |
| < | Less than | `5 < 3` is evaluated to 0 |
| != | Not equal to | `5 != 3` is evaluated to 1 |
| >= | Greater than or equal to | `5 >= 3` is evaluated to 1 |
| <= | Less than or equal to | `5 <= 3` is evaluated to 0 |

**Example code:**

```
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

## C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

| Operator | Meaning | Example |
|---|---|---|
| && | Logical AND. True only if all operands are true | If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0. |
| \|\| | Logical OR. True only if either one operand is true | If c = 5 and d = 2 then, expression `((c==5) \|\| (d>5))` equals to 1. |
| ! | Logical NOT. True only if the operand is 0 | If c = 5 then, expression `!(c==5)` equals to 0. |

**Example code:**

```
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

## C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.
Bitwise operators are used in C programming to perform bit-level operations.

| Operators | Meaning of operators |
|---|---|
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise exclusive OR |
| ~ | Bitwise complement |
| << | Shift left |
| >> | Shift right |

## Comma Operator

Comma operators are used to link related expressions together. For example:

**int a, c = 5, d;**

## The sizeof operator

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

**Example code:**

```
include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));
    printf("Size of char=%lu byte\n",sizeof(d));

    return 0;
}
```

## Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

| **Category** | **Operator** | **Associativity** |
|---|---|---|
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)* & sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |

**Example code:**

```
#include <stdio.h>

main() {

   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;       // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n",  e );

   e = ((a + b) * c) / d;     // (30 * 15 ) / 5
```

```
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

   e = (a + b) * (c / d);   // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );

   e = a + (b * c) / d;     //  20 + (150/5)
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );

   return 0;
}
```

When compile and execute the above program, it produces the following result :

Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is  : 90
Value of (a + b) * (c / d) is  : 90
Value of a + (b * c) / d is  : 50

*Parentheses for Grouping Subexpressions*

Parentheses are used in C expressions in the same manner as in algebraic expressions. For example, to multiply a times the quantity b + c we write a * ( b + c ).

*Rules of Operator Precedence*

C applies the operators in arithmetic expressions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those in algebra:

1. Operators in expressions contained within pairs of parentheses are evaluated first. Parentheses are said to be at the "highest level of precedence." In cases of nested, or embedded, parentheses, such as

   ```
   ( ( a + b ) + c )
   ```

   the operators in the *innermost* pair of parentheses are applied first.

2. Multiplication, division and remainder operations are applied next. If an expression contains several multiplication, division and remainder operations, evaluation proceeds from left to right. Multiplication, division and remainder are said to be on the same level of precedence.

3. Addition and subtraction operations are evaluated next. If an expression contains several addition and subtraction operations, evaluation proceeds from left to right. Addition and subtraction also have the same level of precedence, which is lower than the precedence of the multiplication, division and remainder operations.

4. The assignment operator (=) is evaluated last.

The rules of operator precedence specify the order C uses to evaluate expressions.[1] When we say evaluation proceeds from left to right, we're referring to the associativity of the operators. We'll see that some operators associate from right to left. Figure 2.10 summarizes these rules of operator precedence for the operators we've seen so far.

## Math library functions

Math library functions allow you to perform certain common mathematical calculations.

Functions are normally used in a program by writing the name of the function followed by a left parenthesis followed by the argument (or a comma-separated list of arguments) of the function followed by a right parenthesis. For example, to calculate and print the square root of 900.0 you might write When this statement executes,

**printf( "%.2f", sqrt( 900.0 ) );**

the math library function sqrt is called to calculate the square root of the number contained in the parentheses (900.0). The number 900.0 is the argument of the sqrt function. The preceding statement would print 30.00. The sqrt function takes an argument of type double and returns a result of type double. All functions in the math library that return floating-point values return the data type double. Note that double values, like float values, can be output using the %f conversion specification.

Error-Prevention Tip Include the math header by using the preprocessor directive #include when using functions in the math library

| Function | Description | Example |
|---|---|---|
| `sqrt( x )` | square root of *x* | `sqrt( 900.0 )` is 30.0<br>`sqrt( 9.0 )` is 3.0 |
| `cbrt( x )` | cube root of *x* (C99 and C11 only) | `cbrt( 27.0 )` is 3.0<br>`cbrt( -8.0 )` is -2.0 |
| `exp( x )` | exponential function $e^x$ | `exp( 1.0 )` is 2.718282<br>`exp( 2.0 )` is 7.389056 |
| `log( x )` | natural logarithm of *x* (base *e*) | `log( 2.718282 )` is 1.0<br>`log( 7.389056 )` is 2.0 |
| `log10( x )` | logarithm of *x* (base 10) | `log10( 1.0 )` is 0.0<br>`log10( 10.0 )` is 1.0<br>`log10( 100.0 )` is 2.0 |
| `fabs( x )` | absolute value of *x* as a floating-point number | `fabs( 13.5 )` is 13.5<br>`fabs( 0.0 )` is 0.0<br>`fabs( -13.5 )` is 13.5 |
| `ceil( x )` | rounds *x* to the smallest integer not less than *x* | `ceil( 9.2 )` is 10.0<br>`ceil( -9.8 )` is -9.0 |
| `floor( x )` | rounds *x* to the largest integer not greater than *x* | `floor( 9.2 )` is 9.0<br>`floor( -9.8 )` is -10.0 |
| `pow( x, y )` | *x* raised to power *y* ($x^y$) | `pow( 2, 7 )` is 128.0<br>`pow( 9, .5 )` is 3.0 |
| `fmod( x, y )` | remainder of *x*/*y* as a floating-point number | `fmod( 13.657, 2.333 )` is 1.992 |
| `sin( x )` | trigonometric sine of *x* (*x* in radians) | `sin( 0.0 )` is 0.0 |
| `cos( x )` | trigonometric cosine of *x* (*x* in radians) | `cos( 0.0 )` is 1.0 |
| `tan( x )` | trigonometric tangent of *x* (*x* in radians) | `tan( 0.0 )` is 0.0 |

**Fig. 5.2** | Commonly used math library functions.

# TASKS

1. Write a C program that accepts an employee's ID, total worked hours of a month and the amount he received per hour. Print the employee's ID and salary (with two decimal places) of a particular month.

   Output:

   Input the Employees ID: 0342

   Input the working hrs: 8

   Salary amount/hr: 15000

   Expected Output:

   Employees ID = 0342

   Salary = 120000.00

2. Write a program that asks the user to enter two numbers, obtains them from the user and prints their sum, product, difference, quotient and remainder.
3. State the order of evaluation of the operators in each of the following C statements and show the value of x after each statement is performed.

   a) x = 7 + 3 * 6 / 2 - 1;

   b) x = 2 % 2 + 2 * 2 - 2 / 2;

   c) x = ( 3 * 9 * ( 3 + ( 9 * 3 / ( 3 ) ) ) );

4. Write a C program to input number of days from user and convert it to years, weeks and days.
5. Write a C program that takes hours and minutes as input, and calculates the total number of minutes.
6. Write a C program to Find out distance, coordinates of midpoint using distance formula, derived from Pythagorean Theorem and value of X by Quadratic formula, as follows:

   Distance = √((x2-x1)^2 + (y2-y1)^2)

   Midpoint= ((x2+x1/2), (y2+y1/2))

   $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ Given (a≠0.)

7. Write c program that find the result of the following operations:
   Observe and write in comment how that operation produce results.
   - 5 + 4
   - 10/2
   - True OR False
   - 20 MOD 3
   - 5< 8
   - 25 MOD 70
   - "A" > "H"
   - NOT True
   - 25/70
   - False AND True
   - 20 * 0.5
   - 35 <= 35

8. Aiman is fond of collecting different country's postage stamps. She is so passionate that she had collected 7 Pakistani stamps, 4 UK stamps, 3 Germany and 3 Australian stamps. Based on the input, identify how many international stamps she has?
9. Write C program to create a BMI calculator application that reads the user's weight in pounds and height in inches (or, if you prefer, the user's weight in kilograms and height in meters), then calculates and displays the user's body mass index. Also, the application should display the following information from the Department of Health and Human Services/National Institutes of Health so the user can evaluate his/her BMI:
   BMI VALUES
   Underweight: less than 18.5
   Normal: between 18.5 and 24.9
   Overweight: between 25 and 29.9
   Obese: 30 or greater

10. Admissions were announced in FAST university. Interested candidates need to fill the admission form which contains his/her name, father's name, CNIC, phone number, address, SSC/Olevel result and Intermediate/A-level result. Based on the given information, administration will decide whether the candidate is eligible to give admission test or not. The eligibility criteria are 60% results in SSC and 70% results in intermediate. According to scenario take the input from user and verify if he/she is eligible to give admission test or not.
11. A company annual policy is to gives a bonus at the end of each year. For an employee to get a bonus, the following must be true:

- The employee has been working at the company for more than six months with no negative reports.
- The employee has earned more than 50,000 during year.