# Basic C Handbook

C pirates

**Written by**

MD Mahmudullah
MD Razib
MD Rakibul Islam
MD Shihab Uddin Mullah
Nur-E-Jannat Meherin

# Basic C Handbook

From the course

Problem solving lab

CSE123

PC-L

Written by C pirates

**Prepared by**

MD Razib
201-15-3637
MD Mahmudullah
201-15-3407
MD Rakibul Islam
201-15-3461
MD Shihab Uddin Mullah
201-15-3074
Nur-E-Jannat Meherin
201-15-3529

**Supervised by**

Mushfiqur Rahman
Lecturer
Department of Computer Science & Engineering
Faculty of Science & Information Technology
**Daffodil international University**

**Design**

MD Razib

**Implementation & Coding**

MD Mahmudullah

**Research & Management**

MD Rakibul Islam
MD Shihab Uddin Mullah
Nur-E-Jannat Meherin

# Contents

# Chapter 1

# Introduction of C programming

## 1.0 Quick History of C

- Developed at Bell Laboratories in the early seventies by Dennis Ritchie.
- Born out of two other languages – BCPL (Basic Control Programming Language) and B.
- C introduced such things as character types, floating point arithmetic, structures, unions and the preprocessor.
- The principal objective was to devise a language that was easy enough to understand to be "high-level" – i.e. understood by general programmers, but low-level enough to be applicable to the writing of systems-level software.
- The language should abstract the details of how the computer achieves its tasks in such a way as to ensure that C could be portable across different types of computers, thus allowing the UNIX operating system to be compiled on other computers with a minimum of re-writing.
- C as a language was in use by 1973, although extra functionality, such as new types, was introduced up until 1980.
- In 1978, Brian Kernighan and Dennis M. Ritchie wrote the seminal work The C Programming Language, which is now the standard reference book for C.
- A formal ANSI standard for C was produced in 1989.

- Using C language scientific, business and system-level applications can be developed easily.

## 1.1 Why to use C?

C was initially used for system development work, in particular the programs that make up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Databases
- Language Interpreters
- Utilities

## 1.2 C program

A C program can vary from 3 lines to millions of lines and it should be written into one or more text files with extension ".c"; for example, hello.c. You can use "vi", "vim" or any other text editor to write your C program into a file.

This tutorial assumes that you know how to edit a text file and how to write source code using any programming language.

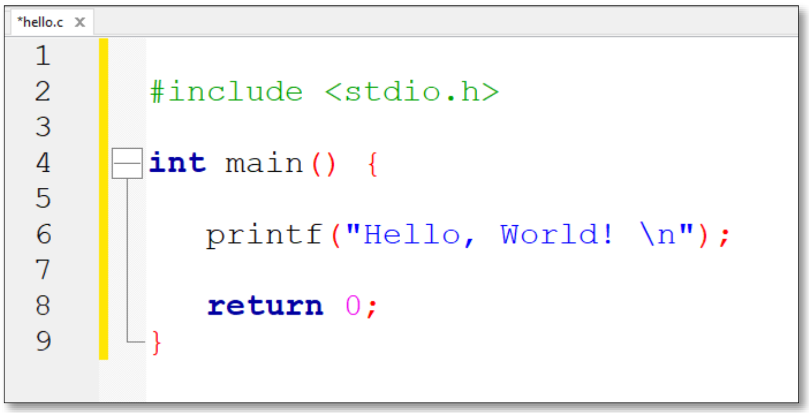
# Chapter 2

# Getting started with C Language

## 2.0 Your first Program

To create a simple C program which prints "Hello, World" on the screen, use a text editor to create a new file (e.g. hello.c — the file extension must be .c) containing the following source code:

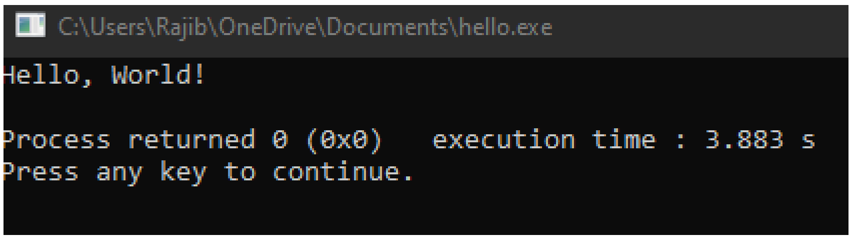
Program:



*hello.c

```c

#include <stdio.h>

int main() {

    printf("Hello, World! \n");

    return 0;
}
```

**Output:**



C:\Users\Rajib\OneDrive\Documents\hello.exe

```
Hello, World!

Process returned 0 (0x0)   execution time : 3.883 s
Press any key to continue.
```

Every C program uses libraries, which give the ability to execute necessary functions. For example, the most basic function called printf, which prints to the screen, is defined in the stdio.h header file.

To add the ability to run the printf command to our program, we must add the following include directive to our first line of the code:

```
#include <stdio.h>
```

The second part of the code is the actual code which we are going to write. The first code which will run will always reside in the main function.

```
int main() {
  ... our code goes here
}
```

The int keyword indicates that the function main will return an integer - a simple number. The number which will be returned by the function indicates whether the program that we wrote worked correctly. If we want to say that our code was run successfully, we will return the number 0. A number greater than 0 will mean that the program that we wrote failed.

For this tutorial, we will return 0 to indicate that our program was successful:

```
return 0;
```

Notice that every line in C must end with a semicolon, so that the compiler knows that a new line has started.

Last but not least, we will need to call the function printf to print our sentence.

## 2.1 Identifiers

A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9).

C does not allow punctuation characters such as @, $, and % within identifiers. C is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in C. Here are some examples of acceptable identifiers:

```
mohd        zara    abc   move_name a_123
myname50    _temp   j     a23b9      retVal
```

## Rules for naming identifiers

1.A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.

2.The first letter of an identifier should be either a letter or an underscore.

3.You cannot use keywords as identifiers.

4.There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.

## 2.2 Keywords

The following list shows the reserved words in C. These reserved words may not be used as constant or variable or any other identifier names.

| | | | |
|---|---|---|---|
| auto | else | Long | switch |
| break | enum | register | typedef |
| case | extern | return | union |

| | | | |
|---|---|---|---|
| char | float | short | unsigned |
| const | for | signed | void |
| continue | goto | sizeof | volatile |
| default | if | static | while |
| do | int | struct | _packed |
| double | | | |

## 2.3 C Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive. Based on the basic types explained in previous chapter, there will be the following basic variable types:

| **Type** | **Description** |
|---|---|
| Char | Typically, a single octet (one byte). This is an integer type. |
| Int | The most natural size of integer for the machine. |
| Float | A single-precision floating point value. |
| Double | A double-precision floating point value. |
| Void | Represents the absence of type. |

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

### Rules for naming a variable

1.A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.

2.The first letter of a variable should be either a letter or an underscore.

3.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.

## 2.4 Literals

### 2.4.1 Floating-point

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing decimal form, you must include the decimal point, the exponent, or both; and while representing exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Here are some examples of floating-point literals −

```
3.14159       /* Legal */
```

```
314159E-5L   /* Legal */

510E         /* Illegal: incomplete exponent */

210f         /* Illegal: no decimal or exponent */

.e55         /* Illegal: missing integer or fraction */
```

### 2.4.2 Character Constants

Character literals are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of char type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C that represent special meaning when preceded by a backslash for example, newline (\n) or tab (\t).

Following is the example to show a few escape sequences characters −

```
#include <stdio.h>
int main () {
  printf("Hello\tWorld\n\n");
  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Hello World
```

### 2.4.3 String Literals

String literals or constants are enclosed in double quotes "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating them using white spaces.

Here are some examples of string literals. All the three forms are identical strings.

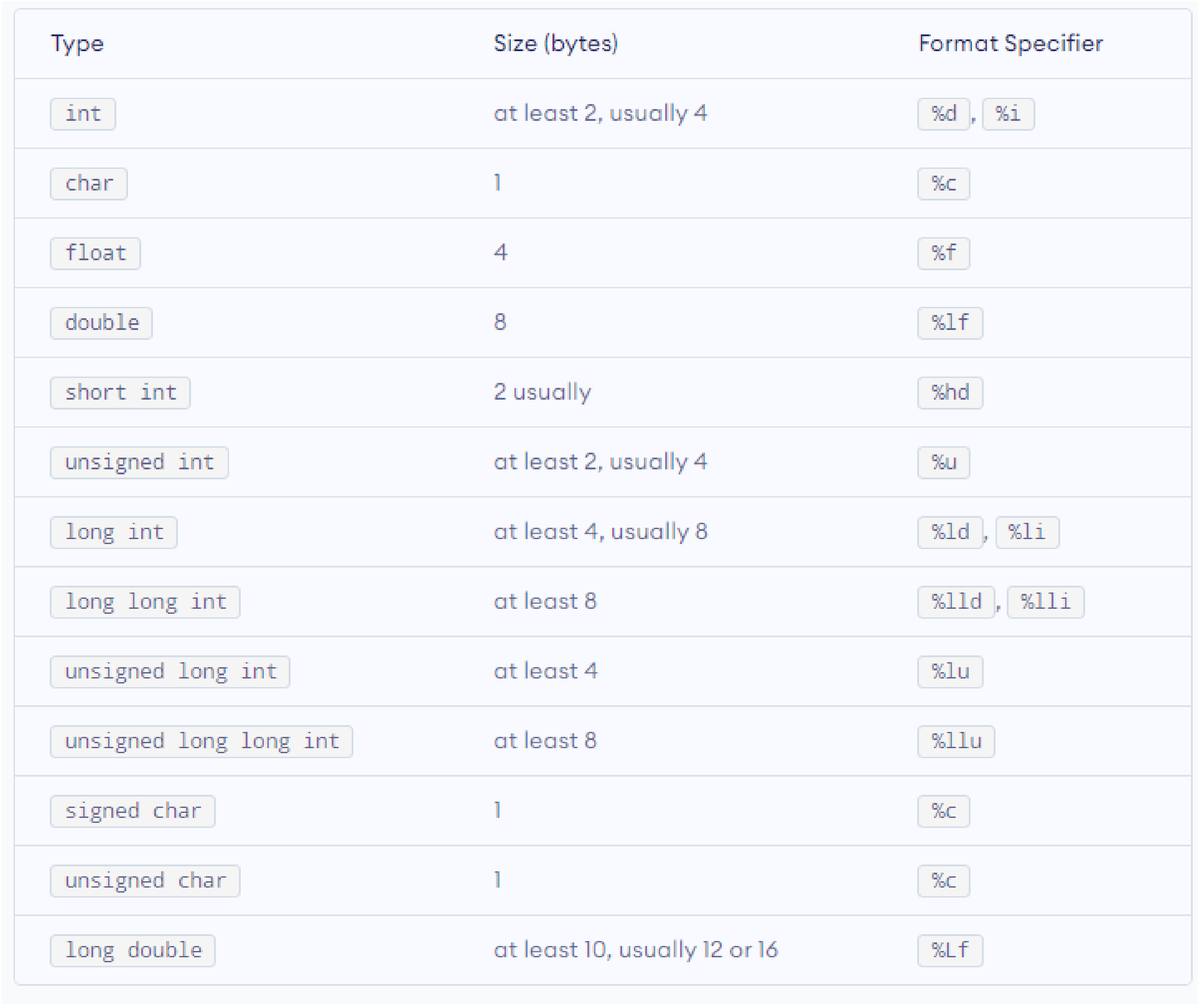
"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

## 2.5 Data Type



| Type | Size (bytes) | Format Specifier |
|---|---|---|
| `int` | at least 2, usually 4 | `%d`, `%i` |
| `char` | 1 | `%c` |
| `float` | 4 | `%f` |
| `double` | 8 | `%lf` |
| `short int` | 2 usually | `%hd` |
| `unsigned int` | at least 2, usually 4 | `%u` |
| `long int` | at least 4, usually 8 | `%ld`, `%li` |
| `long long int` | at least 8 | `%lld`, `%lli` |
| `unsigned long int` | at least 4 | `%lu` |
| `unsigned long long int` | at least 8 | `%llu` |
| `signed char` | 1 | `%c` |
| `unsigned char` | 1 | `%c` |
| `long double` | at least 10, usually 12 or 16 | `%Lf` |

In C programming, data types are declarations for variables. This determines the type and size of data associated with variables.

**1.int=**Integers are whole numbers that can have both zero, positive and negative values but no decimal values. For example,

int DIU;

Here, DIU is a variable of int (integer) type. The size of int is 4 bytes.

Here's a table containing commonly used types in C programming for quick access

**2.float and double=**float and double are used to hold real numbers.

**3.char=**Keyword char is used for declaring character type variables.

**4.short and long=**If you need to use a large number, you can use a type specifier long or If you are sure, only a small integer ([−32,767, +32,767] range) will be used, you can use short.

**5. signed and unsigned**

In C, signed and unsigned are type modifiers. You can alter the data storage of a data type by using them. For example,

unsigned int x;

int y;

Here, the variable x can hold only zero and positive values because we have used the unsigned modifier.

Considering the size of int is 4 bytes, variable y can hold values from -231 to 231-1, whereas variable x can hold values from 0 to 232-1.

## 2.6 Special Characters

Special Characters in C Programming

, < > . _

( ) ; $ :

% [ ] # ?

' & { } "

^ ! * / |

- \ ~ +

White space Characters

Blank space, newline, horizontal tab, carriage return and form feed.

## 2.7 Escape Sequences

Sometimes, it is necessary to use characters that cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc.

In order to use these characters, escape sequences are used.

Escape Sequences

| Escape Sequences | Character |
|---|---|
| \b | Backspace |
| \f | Form feed |
| \n | Newline |
| \r | Return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \\ | Backslash |
| \’ | Single quotation mark |
| \" | Double quotation mark |
| \? | Question mark |
| \0 | Null character |

For example: \n is used for a newline. The backslash \ causes escape from the normal way the characters are handled by the compiler.

## 2.8 Comment

**Rules of Comment**

Program Contain Any number of comments at any place

```
void main(/*Main Function*/)
{
int age; // Variable
printf("Enter Your Age : ");
scanf("%d",/*Type Age*/&age);
printf("Age is %d : ",/*Your age here*/age);
getch()
}
```

2 . Nested Comments are not allowed i.e Comment within Comment

```
/* This is /*Print Hello */ */
printf("Hello");
```

3 . Comments can be split over more than one line.

```
printf("Hello");    /* This
                    is
                    Comment */
```

4 . Comments are not Case-Sensitive.

```
printf("Hello");
   /* This is Comment */
   /* THIS IS COMMENT */
```

5 . Single line Comments Starts with '//'

```
printf("Hello");
   // This is Single Line Comment
```

# Chapter 3

# Function

## 3.1 Function Basic

A function is a set of statements that take inputs, do some specific computation and produces output.

The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.

**Types of function:**

Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming

**Library Functions :** functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.

**User Defined Functions:** functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.

**Advantage of functions in C:**

By using functions, we can avoid rewriting same logic/code again and again in a program.

We can call C functions any number of times in a program and from any place in a program.

We can track a large C program easily when it is divided into multiple functions.

Reusability is the main achievement of C functions.

However, Function calling is always a overhead in a C program.

## 3.2 Syntax and example

There are three aspects of a C function.

**Function declaration:** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.

**Function call:** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.

**Function definition:** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

**Function Syntax**

```
return_type function_name( type1 argument1, type2 argument2, ... )
{
   body of the function
}
```

**Return Type:** A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

**Function Name:** This is the actual name of the function. The function name and the argument(parameter) list together constitute the function signature.

**Arguments:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

**Function Body:** The function body contains a collection of statements that define what the function does.

**Example**:

```
#include<stdio.h>
int main ()
{
    char s[30];
    printf("Enter the string? ");
    gets(s);
    printf("You entered %s",s);
return 0;
}
```

**Output:**

```
Enter the string?
I love Bangla.
You entered I love Bangla.
```

## 3.3 Standard Header File

Header files are helping file of your C program which holds the definitions of various functions and their associated variables that needs to be imported into your C program with the help of pre-processor #include statement.

All the header file have a '.h' an extension that contains C function declaration and macro definitions. In other words, the header files can be requested using the preprocessor directive #include.

The default header file that comes with the C compiler is the stdio.h.

Including a header file means that using the content of header file in your source program. A straightforward practice while programming in C or C++ programs is that you can keep every macro, global variables, constants, and other function prototypes in the header files.

The basic syntax of using these header files is:

#include <file>

| **Header Files** | **Description** |
|---|---|
| stdio.h | Input/output functions |
| conio.h | Console Input/output function |
| stdlib.h | General utility functions |
| math.h | Mathematics function |
| string.h | String functions |
| ctype.h | Character handling functions |
| time.h | Date and time functions |

# Chapter 4

# Operators

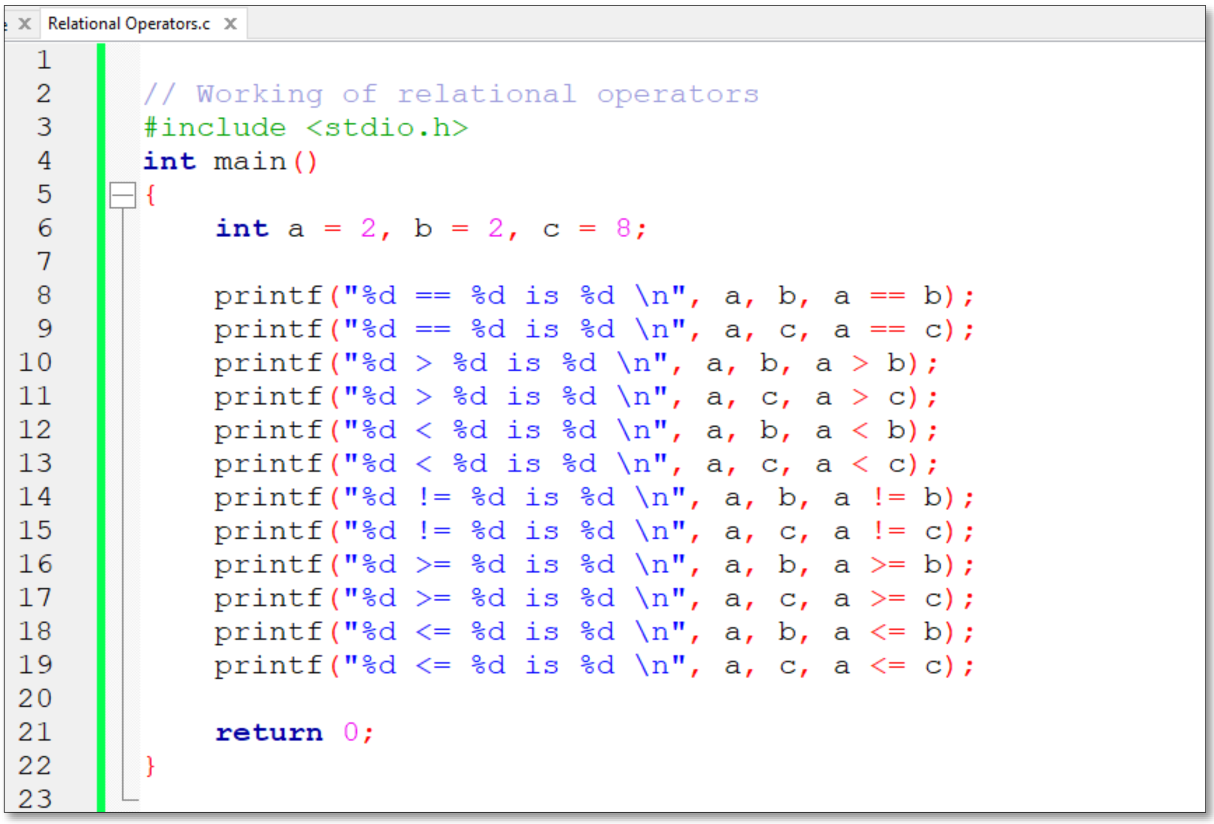
An operator in a programming language is a symbol that tells the compiler or interpreter to perform a specific mathematical, relational or logical operation and produce a final result. C has many powerful operators. Many C operators are binary operators, which means they have two operands. For example, in a / b, / is a binary operator that accepts two operands (a, b). There are some unary operators which take one operand (for example: ~, ++), and only one ternary operator?:

## 4.1: Relational Operators

Relational operators check if a specific relation between two operands is true. The result is evaluated to 1 (which means true) or 0 (which means false). This result is often used to affect control flow (via if, while, for), but can also be stored in variables.

| 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 |

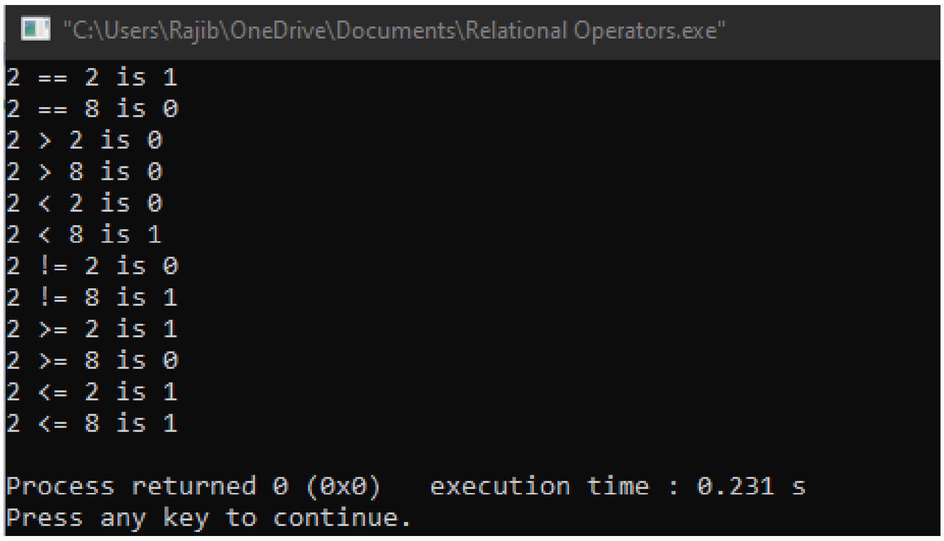
## Relational operators Example:



Relational Operators.c

```c

// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 2, b = 2, c = 8;

    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;
}

```

## Output:



"C:\Users\Rajib\OneDrive\Documents\Relational Operators.exe"

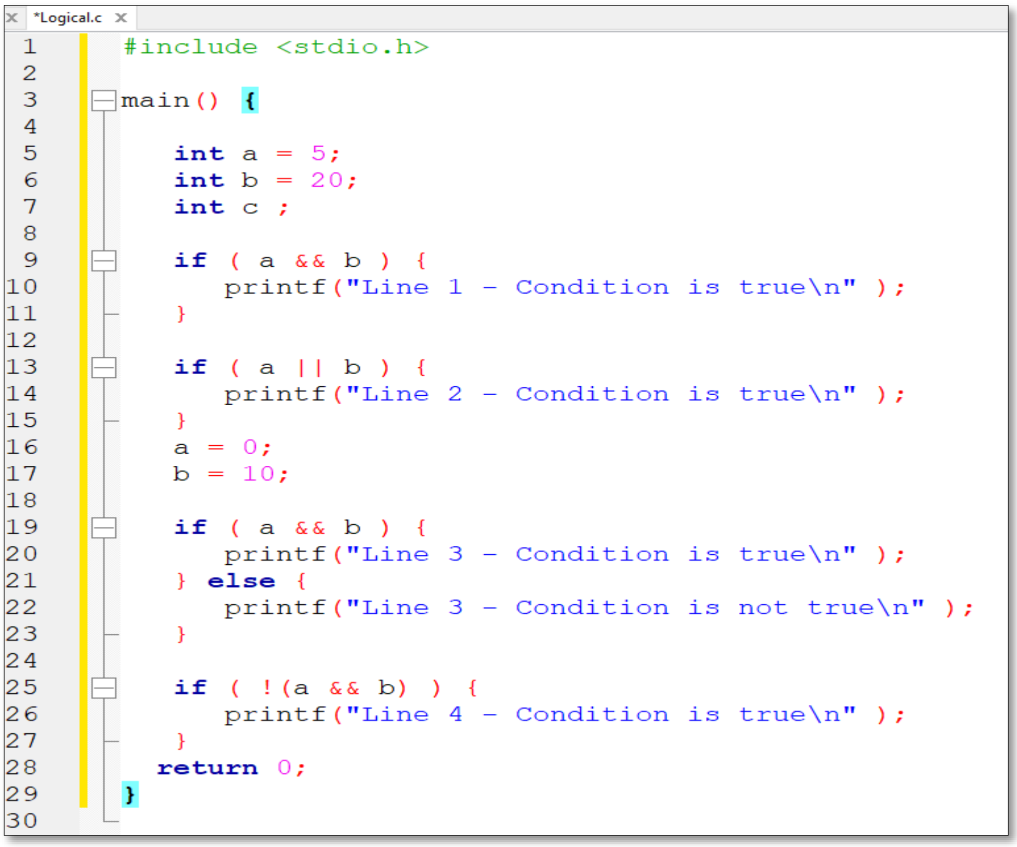
```
2 == 2 is 1
2 == 8 is 0
2 > 2 is 0
2 > 8 is 0
2 < 2 is 0
2 < 8 is 1
2 != 2 is 0
2 != 8 is 1
2 >= 2 is 1
2 >= 8 is 0
2 <= 2 is 1
2 <= 8 is 1

Process returned 0 (0x0)   execution time : 0.231 s
Press any key to continue.
```

## 4.2: Logical Operator

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. |

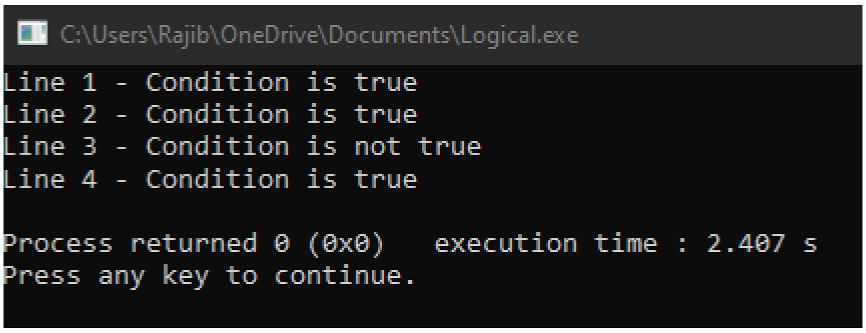
### Logical Operator Example:



*Logical.c

```c
#include <stdio.h>

main() {

    int a = 5;
    int b = 20;
    int c ;

    if ( a && b ) {
       printf("Line 1 - Condition is true\n" );
    }

    if ( a || b ) {
       printf("Line 2 - Condition is true\n" );
    }
    a = 0;
    b = 10;

    if ( a && b ) {
       printf("Line 3 - Condition is true\n" );
    } else {
       printf("Line 3 - Condition is not true\n" );
    }

    if ( !(a && b) ) {
       printf("Line 4 - Condition is true\n" );
    }
  return 0;
}
```

Output:



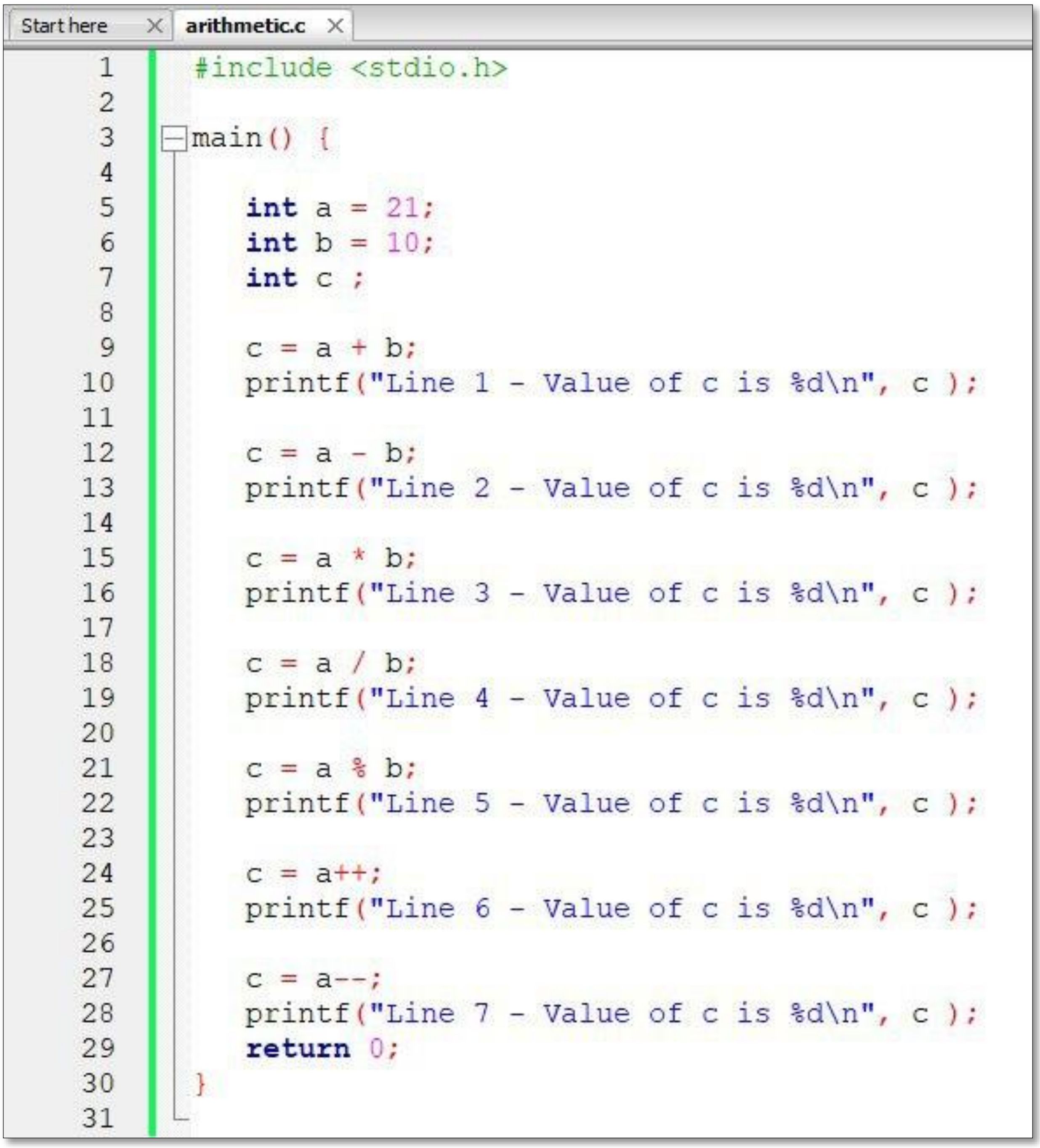
```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true

Process returned 0 (0x0)   execution time : 2.407 s
Press any key to continue.
```

## 4.3: Arithmetic Operators

Return a value that is the result of applying the left-hand operand to the right-hand operand, using the associated mathematical operation. Normal mathematical rules of commutation apply (i.e. addition and multiplication are commutative, subtraction, division and modulus are not).

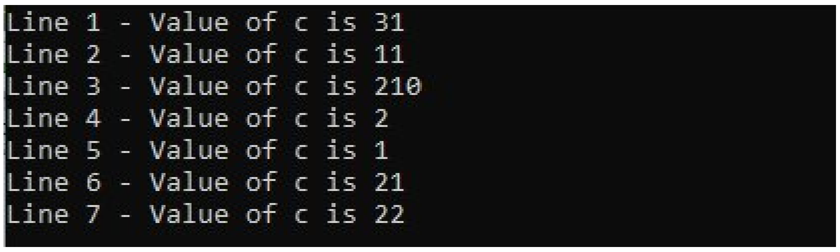
| Operator | Meaning of Operator |
|---|---|
| + | addition or unary plus |
| - | subtraction or unary minus |
| * | multiplication |
| / | division |
| % | remainder after division (modulo division) |

## Arithmetic Operators Example:



```c
#include <stdio.h>

main() {

    int a = 21;
    int b = 10;
    int c ;

    c = a + b;
    printf("Line 1 - Value of c is %d\n", c );

    c = a - b;
    printf("Line 2 - Value of c is %d\n", c );

    c = a * b;
    printf("Line 3 - Value of c is %d\n", c );

    c = a / b;
    printf("Line 4 - Value of c is %d\n", c );

    c = a % b;
    printf("Line 5 - Value of c is %d\n", c );

    c = a++;
    printf("Line 6 - Value of c is %d\n", c );

    c = a--;
    printf("Line 7 - Value of c is %d\n", c );
    return 0;
}
```

Output:



```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 21
Line 7 - Value of c is 22
```

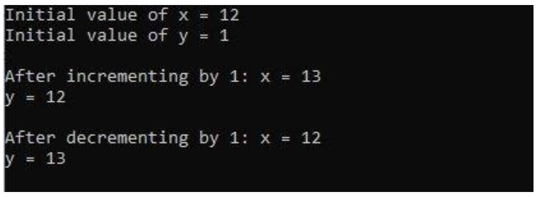
## 4.4: Increment / Decrement Operators

The increment (a++) and decrement (a--) operators are different in that they change the value of the variable you apply them to without an assignment operator. You can use increment and decrement operators either before or after the variable. The placement of the operator changes the timing of the incrementation/decrementation of the value to before or after assigning it to

the variable. Example:

```
#include<stdio.h>

int main ()
{
    int x = 12, y = 1;

    printf("Initial value of x = %d\n", x); // print the initial value of x
    printf("Initial value of y = %d\n\n", y); // print the initial value of y

    y = x++; // use the current value of x then increment it by 1

    printf("After incrementing by 1: x = %d\n", x);
    printf("y = %d\n\n", y);

    y = x--; // use the current value of x then decrement it by 1

    printf("After decrementing by 1: x = %d\n", x);
    printf("y = %d\n\n", y);

    // Signal to operating system everything works fine
    return 0;
}
```

**Output:**



```
Initial value of x = 12
Initial value of y = 1

After incrementing by 1: x = 13
y = 12

After decrementing by 1: x = 12
y = 13
```

## 4.5: Assignment Operators

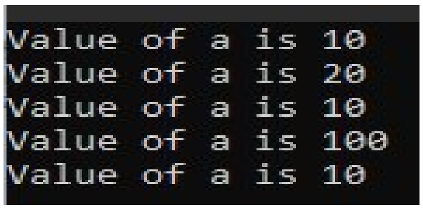
The following table lists the assignment operators supported by the C language –

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator. Assigns values from right side operands to left side operand | C = A + B will assign the value of A + B to C |
| += | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand. | C /= A is equivalent to C = C / A |

|  |  |  |
|---|---|---|
| %= | Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand. | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | Bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| \|= | Bitwise inclusive OR and assignment operator. | C \|= 2 is same as C = C \| 2 |

```
#include <stdio.h>

int main ()
{

    // Assigning value 10 to a
    // using "=" operator
    int a = 10;
    printf("Value of a is %d\n", a);

    // Assigning value by adding 10 to a
    // using "+=" operator
    a += 10;
    printf("Value of a is %d\n", a);

    // Assigning value by subtracting 10 from a
    // using "-=" operator
    a -= 10;
    printf("Value of a is %d\n", a);

    // Assigning value by multiplying 10 to a
    // using "*=" operator
    a *= 10;
    printf("Value of a is %d\n", a);

    // Assigning value by dividing 10 from a
    // using "/=" operator
    a /= 10;
    printf("Value of a is %d\n", a);

    return 0;
}
```

**Output:**



```
Value of a is 10
Value of a is 20
Value of a is 10
Value of a is 100
Value of a is 10
```

## 4..6 Input & output

**C Input**

In C programming, scanf() is one of the commonly used function to take input from the user. The scanf() function reads formatted input from the standard input such as keyboards.

**C Output**

In C programming, printf() is one of the main output functions. The function sends formatted output to the screen.

### Example 1: C Output

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

### Output:

```
C Programming
```

How does this program work?

1.All valid C programs must contain the main() function. The code execution begins from the start of the main() function.

2.The printf() is a library function to send formatted output to the screen. The function prints the string inside quotations.

3.To use printf() in our program, we need to include stdio.h header file using the #include <stdio.h> statement.

4.The return 0; statement inside the main() function is the "Exit status" of the program. It's optional.

## Example 2: Integer Input/output

```
#include <stdio.h>
int main()
{
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d", &testInteger);
    printf("Number = %d",testInteger);
    return 0;
}
```

## Output:

```
Enter an integer: 4
Number = 4
```

Here, we have used %d format specifier inside the scanf() function to take int input from the user. When the user enters an integer, it is stored in the testInteger variable.

Notice, that we have used &testInteger inside scanf(). It is because &testInteger gets the address of testInteger, and the value entered by the user is stored in that address.

## Example 3: Float and Double Input/output

```
#include <stdio.h>
int main()
{
    float num1;
    double num2;

    printf("Enter a number: ");
    scanf("%f", &num1);
    printf("Enter another number: ");
    scanf("%lf", &num2);

    printf("num1 = %f\n", num1);
    printf("num2 = %lf", num2);

    return 0;
}
```

## Output:

```
Enter a number: 12.523
Enter another number: 10.2
num1 = 12.523000
num2 = 10.200000
```

We use %f and %lf format specifier for float and double respectively.

## Example 4: C Character I/O

```
#include <stdio.h>
int main()
{
    char chr;
    printf("Enter a character: ");
    scanf("%c",&chr);
    printf("You entered %c.", chr);
    return 0;
}
```

## Output:

```
Enter a character: g
You entered g
```

When a character is entered by the user in the above program, the character itself is not stored. Instead, an integer value (ASCII value) is stored.

And when we display that value using %c text format, the entered character is displayed. If we use %d to display the character, it's ASCII value is printed.

# Chapter 5

# Problem solving

## 5.1 Basic Problem

**Problem 1:** Write a program that print “I Love Bangladesh”.

```
#include <stdio.h>
int main ()
{
    printf("I Love Bangladesh\n ");
    return 0;
}
```

**Output:**

```
I Love Bangladesh
```

**Problem 2:** write a program that read 2 integer value and print them.

```
#include <stdio.h>
int main ()
{
    int Num1, Num2;
    printf("Enter two integers: ");
    scanf("%d %d", & Num1,&Num2);
    printf("Num1=%d Num2=%d\n", & Num1,&Num2);
    return 0;
}
```

**Output:**

```
Enter two integer: 5 10
Num1=5 Num2=10
```

**Problem 3:** Write a C program to input two numbers from user and calculate their sum.

Sample:

| **Input** | **Output** |
|---|---|
| Input first number: 20<br>Input second number: 10 | Sum= 30 |

```
#include <stdio.h>

int main()
{
    int num1, num2, sum;

    /*
     * Read two numbers from user
     */
    printf("Enter first number: ");
    scanf("%d", &num1);
    printf("Enter second number:");
    scanf("%d", &num2);

    /* Adding both number is simple and fundamental */
    sum = num1 + num2;

    /* Prints the sum of two numbers */
    printf("Sum of %d and %d = %d", num1, num2, sum);

    return 0;
}
```

**Output:**

```
Input first number: 20
Input second number: 10

Sum = 30
```

**Problem 4:** Write a C program to input two numbers and perform all arithmetic operations.

Sample:

| Input | Output |
|---|---|
| First number: 10<br>Second number: 5 | Sum = 15<br>Difference = 5<br>Product = 50<br>Quotient = 2<br>Modulus = 0 |

```
#include <stdio.h>
int main()
{
    int num1, num2;
    int sum, sub, mult, mod;
    float div;
    /*
     * Input two numbers from user
     */
    printf("Enter any two numbers: ");
    scanf("%d%d", &num1, &num2);

    /*
     * Perform all arithmetic operations
     */
    sum = num1 + num2;
    sub = num1 - num2;
    mult = num1 * num2;
    div = (float)num1 / num2;
    mod = num1 % num2;

    printf("SUM = %d\n", sum);
    printf("DIFFERENCE = %d\n", sub);
    printf("PRODUCT = %d\n", mult);
    printf("QUOTIENT = %f\n", div);
    printf("MODULUS = %d", mod);

    return 0;
}
```

**Output:**

```
Enter any two numbers: 10 5
Sum = 15
Difference = 5
Product = 50
Quotient = 2
Modulus = 0
```

**Problem 5:** The formula to calculate the area of a circumference is defined as A = π . R2. Considering to this problem that π = 3.14159:

| **Input** | **Output** |
|---|---|
| 2.00 | A=12.5664 |

```
#include<stdio.h>
int main()
{
    double b=0,R=0 ,p=3.14159;
    scanf("%lf",&R);
    b=p*(R*R);
    printf("A=%.4lf\n",b);
    return 0;
}
```

**Output:**

```
2
A=12.5664
```

**Problem 6:** Write a C program to input length and width of a rectangle and calculate perimeter of the rectangle.

Sample:

| **Input** | **Output** |
|---|---|
| Enter length: 5<br><br>Enter width: 10 | Perimeter of rectangle = 30 units |

```c
#include <stdio.h>

int main()
{
    float length, width, perimeter;

    /*
     * Input length and width of rectangle from user
     */
    printf("Enter length of the rectangle: ");
    scanf("%f", &length);
    printf("Enter width of the rectangle: ");
    scanf("%f", &width);

    /* Calculate perimeter of rectangle */
    perimeter = 2 * (length + width);

    /* Print perimeter of rectangle */
    printf("Perimeter of rectangle = %f units ", perimeter);

    return 0;
}
```

**Output:**

```
Enter length: 5
Enter width: 10
Perimeter of rectangle = 30 units
```

**Problem 7:** Write a C program to input temperature in Centigrade and convert to Fahrenheit.

Sample:

| Input | Output |
|---|---|
| Enter temperature in Celsius = 100 | 100 Celsius = 212.00 Fahrenheit |

```
#include <stdio.h>

int main()
{
    float celsius, fahrenheit;

    /* Input temperature in celsius */
    printf("Enter temperature in Celsius: ");
    scanf("%f", &celsius);

    /* celsius to fahrenheit conversion formula */
    fahrenheit = (celsius * 9 / 5) + 32;

    printf("%.2f Celsius = %.2f Fahrenheit", celsius, fahrenheit);

    return 0;
}
```

## Output:

```
Enter temperature in Celsius: 100
100 Celsius = 212.00 Fahrenheit
```

**Problem 8:** Write a C program to input principle, time and rate (P, T, R) from user and find Simple Interest.

Sample:

| Input | Output |
|---|---|
| Enter principle (amount): 1200<br>Enter time: 2<br>Enter rate: 5.4 | Simple Interest = 129.600006 |

```c
#include <stdio.h>

int main()
{
    float principle, time, rate, SI;

    /* Input principle, rate and time */
    printf("Enter principle (amount): ");
    scanf("%f", &principle);

    printf("Enter time: ");
    scanf("%f", &time);

    printf("Enter rate: ");
    scanf("%f", &rate);

    /* Calculate simple interest */
    SI = (principle * time * rate) / 100;

    /* Print the resultant value of SI */
    printf("Simple Interest = %f", SI);

    return 0;
}
```

## Output:

```
Enter principle (amount): 1200
Enter time: 2
Enter rate: 5.4
Simple Interest = 129.600006
```

**Problem 9:** Read three inputs. The input file contains a text (employee's first name), and two double precision values, that are the seller's salary and the total value sold by him/her and Print the seller's total salary, according to the given example.

Sample

| Input | Output |
|---|---|
| 500.00<br>1230.30 | TOTAL = R$ 684.54 |

```c
#include<stdio.h>
int main()
{
   char NAME;
   double FIXED_SALARY,SELL,TOTAL;
   scanf("%S %lf %lf",&NAME,&FIXED_SALARY,&SELL);
   TOTAL=FIXED_SALARY+(SELL*0.15);
   printf("TOTAL = R$ %.2lf\n",TOTAL);
   return 0;
}
```

**Output:**

```
500.00
1230.30
TOTAL = R$ 684.54
```

**Problem 10:** Write a C program to input a number and find square root of the given number using inbuilt sqrt() function.

Sample:

| Input | Output |
|---|---|
| Enter any number: 9 | Square root of 9 = 3 |

```
#include <stdio.h>
#include <math.h>

int main()
{
    double num, root;

    /* Input a number from user */
    printf("Enter any number to find square root: ");
    scanf("%lf", &num);

    /* Calculate square root of num */
    root = sqrt(num);

    /* Print the resultant value */
    printf("Square root of %.2lf = %.2lf", num, root);

    return 0;
}
```

**Output:**

```
Enter any number to find square root: 144
Square root of 144.00 = 12.00
```

## 5.2 If statement in C

The Syntax of if statement is-

```
if(Boolean expression)

{

  // body of if

}
```

In above syntax if Boolean expression evaluates true, then statements inside if body executes otherwise skipped.

**Example:**

Write a program to input user age and check if he is eligible to vote in India or not. A person in Bangladesh is eligible to vote if he is 18+.

```
#include <stdio.h>

int main()
{
    /* Variable declaration to store age */
    int age;

    /* Input age from user */
    printf("Enter your age: ");
    scanf("%d", &age);

    /* Use relational operator to check age */
    if(age >= 18)
    {
        /* If age is greater than or equal 18 years */
        printf("You are eligible to vote in Bangladesh.");
    }

    return 0;
}
```

**Output:**

```
Enter your age: 24
You are eligible to vote in India.
```

## 5.3 if...else statement in C

The Syntax of if...else statement is-

```
if(Boolean expression)
{
   // Body of if
   // If expression is true then execute this
}
else
{
   // Body of else
   // If expression is false then execute this
}
```

In above syntax if the given Boolean expression is true then, execute body of if part otherwise execute body of else part. In any case either body if or body of else is executed. In no case both the blocks will execute.

**Example:** Write a program to input two numbers from user. Print maximum between both the given numbers.

```
#include <stdio.h>

int main()
{
    int num1, num2;
    /* Input two number from user */
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    /* Compare both number using relational operator */
    if(num1 > num2)
    {
        printf("First number is maximum.");
    }
    else
    {
        printf("Second number is maximum.");
    }
    return 0;
}
```

**Output:**

```
Enter two numbers: 10
20
Second number is maximum.
```

## 5.4 if...else...if statement

syntax of if...else...if statement is-

```
if (boolean_expression_1)
{
   // If expression 1 is true then execute
   // this and skip other if
}
else if (boolean_expression_2)
{
   // If expression 1 is false and
   // expression 2 is true then execute
   // this and skip other if
}
else if (boolean_expression_n)
{
   // If expression 1 is false,
   // expression 2 is also false,
   // expression n-1 is also false,
   // and expression n is true then execute
   // this and skip else.
}
else
{
   // If no expressions are true then
   // execute this skipping all other.
}
```

**Example:** write a simple C program to input an integer from user. Check if the given integer is negative, zero or positive?

```
#include <stdio.h>

int main()
{
    /* Declare integer variable */
    int num;

    /* Input an integer from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    if(num < 0)
    {
        /* If number is less than zero, then it is negative */
        printf("NUMBER IS NEGATIVE.");
    }
    else if(num == 0)
    {
        /* If number equal to 0, then it is zero */
        printf("NUMBER IS ZERO.");
    }
    else
    {
        /* If number is greater then zero, then it is positive */
        printf("NUMBER IS POSITIVE.");
    }
    return 0;
}
```

**Output:**

```
Enter any number: -22
NUMBER IS NEGATIVE.
```

## 5.5 nested if...else

Syntax of nested if...else statement is-

```
if (boolean_expression_1)
{
   if(nested_expression_1)
   {
      // If boolean_expression_1 and
      // nested_expression_1 both are true
   }
   else
   {
      // If boolean_expression_1 is true
      // but nested_expression_1 is false
   }
   // If boolean_expression_1 is true
}
else
{
   if(nested_expression_2)
   {
        // If boolean_expression_1 is false
        // but nested_expression_2 is true
    }
    else
    {
        // If both boolean_expression_1 and
        // nested_expression_2 is false
    }
    // If boolean_expression_1 is false
}
```

In the above syntax I have nested one if...else statement inside another. However, any two-decision statement can be inside other.

**Example:**

write a program to input three numbers from user. Find maximum between given three numbers.

```c
#include <stdio.h>

int main()
{
    /* Declare three integer variables */
    int num1, num2, num3;

    /* Input three numbers from user */
    printf("Enter three numbers: ");
    scanf("%d%d%d", &num1, &num2, &num3);

    if(num1 > num2)
    {
        if(num1 > num3)
        {
            /* If num1>num2 and num1>num3 */
            printf("Num1 is max.");
        }
        else
        {
            /* If num1>num2 but num1<num3 */
            printf("Num3 is max.");
        }
    }
    else
    {
        if(num2 > num3)
        {
            /* If num1<num2 and num2>num3 */
            printf("Num2 is max.");
        }
        else
        {
            /* If num1<num2 and num2<num3 */
            printf("Num3 is max.");
        }
    }

    return 0;
}
```

**Output:**

```
Enter three numbers: 10
20
30
Num3 is max.
```

**Explanation:** 1.Suppose user inputs three numbers as num1=10, num2=20 and num3=30.

2.The first outer if condition if(num1 > num2) is false since 10 > 20 is false. Hence, outer if statement is skipped, executing the outer else part.

3.Inside the outer else, condition if(num2 > num3) is also false, since 20 > 30 is false. Hence, the inner if statement is skipped, executing inner else part.

4.Inside the inner else there is nothing much to do. Just a simple printf() statement, printing "Num3 is max**."**

## 5.6: Some Conditional problem

**Problem 1:** Write a C program to accept two integers and check whether they are equal or not.

```
#include <stdio.h>
int main()
{
    int int1, int2;

    printf("Input the values for Number1 and Number2 : ");
    scanf("%d %d", &int1, &int2);
    if (int1 == int2)
        printf("Number1 and Number2 are equal\n");
    else
        printf("Number1 and Number2 are not equal\n");
return 0;}
```

**Output:**

```
Input the values for Number1 and Number2 : 15 15
Number1 and Number2 are equal
```

**Problem 2:** Write a C program to check whether a given number is even or odd.

```
#include <stdio.h>
int main()
{
    int num1, rem1;
    printf("Input an integer : ");
    scanf("%d", &num1);
    rem1 = num1 % 2;
    if (rem1 == 0)
        printf("%d is an even integer\n", num1);
    else
        printf("%d is an odd integer\n", num1);
return 0;
}
```

**Output:**

```
Input an integer: 15
15 is an odd integer
```

**Problem 3:** Write a C program to find whether a given year is a leap year or not. Go to the editor.

```
#include <stdio.h>
int main()
{
    int chk_year;

    printf("Input a year :");
    scanf("%d", &chk_year);
    if ((chk_year % 400) == 0)
        printf("%d is a leap year.\n", chk_year);
    else if ((chk_year % 100) == 0)
        printf("%d is a not leap year.\n", chk_year);
    else if ((chk_year % 4) == 0)
        printf("%d is a leap year.\n", chk_year);
    else
        printf("%d is not a leap year \n", chk_year);
return 0;
}
```

**Output:**

```
Input a year :2016
2016 is a leap year.
```

**Problem 4:** Write a C program to find the largest of three numbers.

Test Data: 12 25 52

Expected Output:

1st Number = 12,   2nd Number = 25,    3rd Number = 52

The 3rd Number is the greatest among three

```
#include <stdio.h>
void main()
{
    int num1, num2, num3;

    printf("Input the values of three numbers : ");
    scanf("%d %d %d", &num1, &num2, &num3);
    printf("1st Number = %d,\t2nd Number = %d,\t3rd Number = %d\n", num1,
num2, num3);
    if (num1 > num2)
    {
        if (num1 > num3)
        {
            printf("The 1st Number is the greatest among three. \n");
        }
        else
        {
            printf("The 3rd Number is the greatest among three. \n");
        }
    }
    else if (num2 > num3)
        printf("The 2nd Number is the greatest among three \n");
    else
        printf("The 3rd Number is the greatest among three \n");
return 0;
}
```

**Output:**

```
Input the values of three numbers : 12 25 52
1st Number =12,2nd Number = 25, 3rd Number = 52
The 3rd Number is the greatest among three
```

**Problem 5:** Write a C program to calculate the root of a Quadratic Equation.

Test Data: 1 5 7 Expected Output: Root are imaginary;

No solution.

```
#include <stdio.h>
#include <math.h>

int main()
{
   int a,b,c,d;
   float x1,x2;

   printf("Input the value of a,b & c : ");
   scanf("%d%d%d",&a,&b,&c);
   d=b*b-4*a*c;
   if(d==0)
   {
     printf("Both roots are equal.\n");
     x1=-b/(2.0*a);
     x2=x1;
     printf("First  Root Root1= %f\n",x1);
     printf("Second Root Root2= %f\n",x2);
   }
   else if(d>0)
  {
     printf("Both roots are real and diff-2\n");
     x1=(-b+sqrt(d))/(2*a);
     x2=(-b-sqrt(d))/(2*a);
     printf("First  Root Root1= %f\n",x1);
     printf("Second Root root2= %f\n",x2);
  }
  else
      printf("Root are imeainary;\nNo Solution. \n");
return 0;
}
```

**Output:**

```
Input the value of a,b & c : 1 5 7
Root are imaginary;
No Solution.
```

**Problem 6**:  Write a C program to read temperature in centigrade and display a suitable message according to temperature state below :

Temp < 0 then Freezing weather

Temp 0-10 then Very Cold weather

Temp 10-20 then Cold weather

Temp 20-30 then Normal in Temp

Temp 30-40 then Its Hot

Temp >=40 then Its Very Hot

Test Data:

42

Expected Output:

It’s very hot.

```
#include <stdio.h>
int main()
{
     int tmp;

    printf("Input days temperature : ");
    scanf("%d",&tmp);
   if(tmp<0)
             printf("Freezing weather.\n");
   else if(tmp<10)
            printf("Very cold weather.\n");
            else if(tmp<20)
                        printf("Cold weather.\n");
                    else if(tmp<30)
                               printf("Normal in temp.\n");
                            else if(tmp<40)
                                         printf("Its Hot.\n");
                                    else
                                           printf("Its very hot.\n");
return 0;

}
```

**Output:**

```
Input days temperature: 42
It’s very hot.
```

**Problem 7**: Write a program in C to calculate and print the Electricity bill of a given customer. The customer id., name and unit consumed by the user should be taken from the keyboard and display the total amount to pay to the customer. The charge are as follow:

Unit Charge/unit

upto 199 @1.20

200 and above but less than 400 @1.50

400 and above but less than 600 @1.80

600 and above @2.00

If bill exceeds Rs. 400 then a surcharge of 15% will be charged and the minimum bill should be of Rs. 100/-

Test Data :

1001

James

800

Expected Output :

Customer IDNO :1001

Customer Name :James

unit Consumed :800

Amount Charges @Rs. 2.00 per unit : 1600.00

Surchage Amount: 240.00

Net Amount Paid by the Customer: 1840.00

```
#include <stdio.h>
#include <string.h>
int main()
{
   int custid, conu;
   float chg, surchg=0, gramt, netamt;
   char connm[25];

   printf("Input Customer ID :");
   scanf("%d",&custid);
   printf("Input the name of the customer :");
   scanf("%s",connm);
   printf("Input the unit consumed by the customer : ");
   scanf("%d",&conu);
   if (conu <200 )
  chg = 1.20;
   else  if (conu>=200 && conu<400)
    chg = 1.50;
  else if (conu>=400 && conu<600)
      chg = 1.80;
    else
      chg = 2.00;
   gramt = conu*chg;
   if (gramt>300)
  surchg = gramt*15/100.0;
   netamt = gramt+surchg;
   if (netamt  < 100)
  netamt =100;
   printf("\nElectricity Bill\n");
   printf("Customer IDNO                       :%d\n",custid);
   printf("Customer Name                       :%s\n",connm);
   printf("unit Consumed                       :%d\n",conu);
   printf("Amount Charges @Rs. %4.2f  per unit :%8.2f\n",chg,gramt);
   printf("Surchage Amount                     :%8.2f\n",surchg);
   printf("Net Amount Paid By the Customer     :%8.2f\n",netamt);
return 0;
}
```

**Output:**

```
Input Customer ID :10001
Input the name of the customer: James
Input the unit consumed by the customer : 800
```

```
Electricity Bill
Customer IDNO                       :10001
Customer Name                      : James
unit Consumed                       :800
Amount Charges @Rs. 2.00 per unit: 1600.00
Surchage Amount                    :  240.00
Net Amount Paid By the Customer     : 1840.00
```

## 5.7 switch...case statement

Syntax of switch...case statement is-

```
switch(expression)
{
  case 1:
    /* Statement/s */
    break;
  case 2:
    /* Statement/s */
    break;
  case n:
    /* Statement/s */
    break;
  default:
    /* Statement/s */
}
```

working process of switch case-

Expression inside switch must evaluate to integer, character or enumeration constant. switch...case only works with integral, character or enumeration constant.

The case keyword must follow one constant of type evaluated by expression. The case along with a constant value is known as switch label.

You can have any number of cases.

Each and every case must be distinct from other. For example, it is illegal to write two case 1 label.

You are free to put cases in any order. However, it is recommended to put them in ascending order. It increases program readability.

You can have any number of statements for a specific case.

The break statement is optional. It transfers program flow outside of switch...case. break statement is covered separately in this C tutorial series.

an example to demonstrate the working of switch...case statement.

```
int num = 2;

switch(num)
{
   case 1: printf("I am One");
      break;
   case 2: printf("I am Two");
      break;
   case 3: printf("I am Three");
      break;
   default: printf("I am an integer. But definitely I am not 1, 2 and 3.");
}
```

Initially I declared an integer variable num = 2.

switch(num) will evaluate the value of num to 2.

After switch(num) got evaluated, switch knows the case to transfer program control.

Instead of checking all cases one by one. It transfers program control directly to case 2. If value of num is not matched with any case then switch transfers control to default case, if defined.

After the control has been set to case 2. It executes all statements inside the case. The case contains two statement first printf("I am Two"); and second break. printf("I am Two"); will print "I am Two" on console and transfers control to break.

break statement terminates switch...case and transfer program control to statement after switch.

What if I don’t use break keyword? If you don’t use break keyword, it executes all below cases until break statement is found. It doesn’t matter whether the remaining case matches or not, it will execute all below case from matching case in the absence of break keyword.

**Example:** write a C program to input week number from user and print the corresponding day name of week.

```
#include <stdio.h>

int main()
{
    /* Declare integer variable to store week number */
    int week;

    /* Input week number from user */
    printf("Enter week number (1-7): ");
    scanf("%d", &week);

    switch(week)
    {
        case 1:
            /* If week == 1 */
            printf("Its Monday.\n");
            printf("Its a busy day.");
            break;
        case 2:
            /* If week == 2 */
            printf("Its Tuesday.");
            break;
        case 3:
            /* If week == 3 */
            printf("Its Wednesday.");
            break;
        case 4:
            /* If week == 4 */
            printf("Its Thursday.\n");
            printf("Feeling bit relaxed.");
            break;
        case 5:
            /* If week == 5 */
            printf("Its Friday.");
            break;
        case 6:
            /* If week == 6 */
            printf("Its Saturday.\n");
            printf("It is weekend.");
            break;
        case 7:
            /* If week == 7 */
            printf("Its Sunday.\n");
            printf("Hurray! Its holiday.");
            break;
        default:
            /* If week < 1 or week > 7 */
            printf("Um! Please enter week number between 1-7.");
    }
    return 0;
}
```

**Output:**

```
Enter week number (1-7): 6
Its Saturday.
It is weekend.
```

### 5.8 Nesting of switch...case statement

Syntax of nested switch...case statement is -

```
switch(expression)
{
  case 1:
    /* Statement/s */
    break;
  case 2:
    /* Statement/s */
    switch(inner_expression)
    {
      case 1:
        /* Statement/s */
        break;
      case 2:
        /* Statement/s */
        break;
      case n:
        /* Statement/s */
        break;
      default:
        /* Statement/s */
        break;
    }
    break;
  case n:
    /* Statement/s */
    break;
  default: /* Default statement/s */
}
```

**Example:**

```
#include <stdio.h>
int main() {
        int ID = 500;
        int password = 000;
        printf("Plese Enter Your ID:\n ");
        scanf("%d", & ID);
        switch (ID) {
            case 500:
                printf("Enter your password:\n ");
                scanf("%d", & password);
                switch (password) {
                    case 000:
                        printf("Welcome Dear Programmer\n");
                        break;
                    default:
                        printf("incorrect password");
                        break;
                }
                break;
            default:
                printf("incorrect ID");
                break;
        }
}
```

**Output:**

```
Plese Enter Your ID:
 500
Enter your password:
 000
Welcome Dear Programmer
```

1.In the given program we have explained initialized two variables: ID and password

2.An outer switch construct is used to compare the value entered in variable ID. It execute the block of statements associated with the matched case(when ID==500).

3.If the block statement is executed with the matched case, an inner switch is used to compare the values entered in the variable password and execute the statements linked with the matched case(when password==000).

4.Otherwise, the switch case will trigger the default case and print the appropriate text regarding the program outline.

# Chapter 6

# Loop’s

**What are Loops?**

A Loop executes the sequence of statements many times until the stated condition becomes false. A loop consists of two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false. The purpose of the loop is to repeat the same code a number of times.

Types of Loops:

1.For loop

2.While loop

3.do...while loop

4.Nested loops

## 6.1: For loop

**Syntax of for loop**

```
for (variable-initialization; condition; variable-update)
{
   // Body of for loop
}
```

Parts of for loop

Any repetition contain two important part - What to repeat and number of repetition? variable-initialization, condition and variable-update define number of repetition and body of loop define what to repeat.

**1.**Variable-initialization contain loop counter variable initialization statements. It define starting point of the repetition (where to start loop).

2.Condition contain boolean expressions and works like if...else. If boolean expression is true, then execute body of loop otherwise terminate the loop.

3.Body of loop specify what to repeat. It contain set of statements to repeat.

4.Variable-update contains loop counter update (increment/decrement) statements.

**Important note:** All four parts of a for loop is optional. Hence you can write a for loop without initialization, condition, update or body. However, you must follow the syntax and specify semicolons.

**How for loop works?**

1.Initially variable-initialization block receive program control. It is non-repeatable part and executed only once throughout the execution of for loop. After initialization program control is transferred to loop condition.

2.The loop condition block evaluates all boolean expression and determines loop should continue or not. If loop conditions are met, then it transfers program control to body of loop otherwise terminate the loop. In C we specify a boolean expression using relational and logical operator.

3.Body of loop execute a set of statements. After executing all statements, it transfers program control to variable-update block.

4.Variable-update block updates loop counter variable and transfer program control again back to condition block of loop.

**Example:** write a C program that print natural numbers from 1 to 10.

```
#include <stdio.h>

int main()
{
    /* Declare loop counter variable */
    int count;

    /* Run a loop from 1 to 10 */
    for(count=1; count<=10; count++)
    {
        /* Print current value of count */
        printf("%d ", count);
    }

    return 0;
}
```

**Output:**

```
1 2 3 4 5 6 7 8 9 10
```

**Explanation of above program:**

1.We have declared a variable of an int data type to store values.

2.In for loop, in the initialization part, we have assigned value 1 to the variable number. In the condition part, we have specified our condition and then the increment part.

3.In the body of a loop, we have a print function to print the numbers on a new line in the console. We have the value one stored in number, after the first iteration the value will be incremented, and it will become 2. Now the variable number has the value 2. The condition will be rechecked and since the condition is true loop will be executed, and it will print two on the screen. This loop will keep on executing until the value of the variable becomes 10. After that, the loop will be terminated, and a series of 1-10 will be printed on the screen.

## 6.2 while loop

**Syntax of while loop**

```
while(condition)

{

   // Body of while loop

}
```

Parts of while loop

Unlike for loop, while does not contain initialization and update part. It contains only two parts - condition and body of loop.

condition is a Boolean expression evaluating an integer value. It is similar to if...else condition and define loop termination condition.

Body of loop contains single or set of statements. It define statements to repeat.

At this point, you might be thinking about loop counter variable-initialization and variable-update part. Where to put these? You are free to initialize loop counter variables anywhere in the program before its use. However, best practice is to initialize all important loop variable just before the loop. Likewise, you can keep your loop update part just before the end of loop.

**How while loop works?**

Simplicity of while loop exists in its working mechanism. while loop works in two steps.

Initially program control is received by condition block. It contains set of relational and logical expressions. If result of the conditional expression is 1 (true) then while transfers program control to body of loop. Else if result of conditional expression is 0 (false) then it exits from loop.

Body of loop contain single or set of statements to repeat. It execute all statements inside its body and transfer the program control to loop condition block.

Step 1 and 2 are repeated until the loop condition is met.

The above two steps are repeated, until loop condition is true.

**Example:** write a C program that print natural numbers from 1 to 10.

```
#include <stdio.h>

int main()
{
    /* Loop counter variable declaration and initialization*/
    int n = 1;

    /* Loop condition */
    while(n <= 10)
    {
        /* Body of loop */
        printf("%d ", n);

        /* Update loop counter variable */
        n++;
    }
    return 0;
}
```

**Output:**

```
1 2 3 4 5 6 7 8 9 10
```

**Explanation of above program:**

1.We have declared a variable of an int data type to store values.

2.In for loop, in the initialization part, we have assigned value 1 to the variable number. In the condition part, we have specified our condition and then the increment part.

3.In the body of a loop, we have a print function to print the numbers on a new line in the console. We have the value one stored in number, after the first iteration the value will be incremented, and it will become 2. Now the variable number has the value 2. The condition will be rechecked and since the condition is true loop will be executed, and it will print two on the screen. This loop will keep on executing until the value of the variable becomes 10. After that, the loop will be terminated, and a series of 1-10 will be printed on the screen.

## 6.3 Do-While loop

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format of while loop is as follows:

```
 do {
  statements
} while (expression);
```

As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop.

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

Similar to the while loop, once the control goes out of the loop the statements which are immediately after the loop is executed

The critical difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

The following program illustrates the working of a do-while loop:

We are going to print a table of number 2 using do while loop.

```
#include<stdio.h>
#include<conio.h>
int main()
{
  int num=1;  //initializing the variable
  do  //do-while loop
  {
    printf("%d\n",2*num);
    num++;    //incrementing operation
  }while(num<=10);
  return 0;
}
```

## Output:

```
2
4
6
8
10
12
14
16
18
20
```

**Explanation of above program:**

In the above example, we have printed multiplication table of 2 using a do-while loop. Let's see how the program was able to print the series.

1.First, we have initialized a variable 'num' with value 1. Then we have written a do-while loop.

2.In a loop, we have a print function that will print the series by multiplying the value of num with 2.

3.After each increment, the value of num will increase by 1, and it will be printed on the screen.

4.Initially, the value of num is 1. In a body of a loop, the print function will be executed in this way: 2*num where num=1, then 2*1=2 hence the value two will be printed. This will go on until the value of num becomes 10. After that loop will be terminated and a statement which is immediately after the loop will be executed. In this case return 0.

## 6.4 Nested loop

**Syntax of nested loop is-**

outer_loop

{

```
    inner_loop
    {
        // Inner loop statement/s
    }

    // Outer loop statement/s
}
```

**Example:** write a C program to print multiplication table from 1 to 5.

```c
#include <stdio.h>

int main()
{
    /* Loop counter variable declaration */
    int i, j;

    /* Outer loop */
    for(i=1; i<=10; i++)
    {
        /* Inner loop */
        for(j=1; j<=5; j++)
        {
            printf("%d\t", (i*j));
        }

        /* Print a new line */
        printf("\n");
    }

    return 0;
}
```

**Output:**

```
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
5       10      15      20      25
6       12      18      24      30
7       14      21      28      35
8       16      24      32      40
9       18      27      36      45
10      20      30      40      50
```

**Explanation of the program**

To understand the above program easily let us first focus on inner loop.

1.First the initialization part executes initializing j=1. After initialization it transfer program control to loop condition part i.e. j<=5.

2.The loop condition checks if j<=5 then transfer program control to body of loop otherwise terminated the inner loop.

3.Body of loop contains single printf("%d\t", (i*j)); statement. For each iteration it print the product of i and j.

4.Next after loop body, the loop update part receives program control. It increment the value of j with 1 and transfer program control back to loop condition.

From the above description it is clear that the inner loop executes 5 times.

For i=1 it prints the product of i and j

```
1    2    3    4    5
```

Similarly for i=2 it prints

```
2    4    6    8    10
```

Next let us now concentrate on outer loop.

1.In the outer loop first variable i is initialized with 1. Then it transfer program control to loop condition i.e. i<=10.

2.The loop condition part checks if i<=10 then transfer program control to body of loop otherwise terminate from loop.

3.Body of loop does not contain any statement rather it contain another loop i.e. the inner loop we discussed above.

4.The program control is transferred to loop update part i.e. i++ after inner loop terminates. The loop update part increment the value of i with 1 and transfer the control to loop condition.

From the above description of outer loop, it is clear that outer loop executes 10 times. For each time outer loop repeats, inner loop is executed with different value of i printing the below output.

```
1     2     3     4     5

2     4     6     8     10

3     6     9     12    15

4     8     12    16    20

5     10    15    20    25

6     12    18    24    30

7     14    21    28    35

8     16    24    32    40

10    20    30    40    50
```

## 6.5 Break Statement

The break statement is used mainly in in the switch statement. It is also useful for immediately stopping a loop.

We consider the following program which introduces a break to exit a while loop:

```
#include <stdio.h>
int main() {
int num = 5;
while (num > 0) {
  if (num == 3)
    break;
  printf("%d\n", num);
  num--;
}}
```

**Output:**

```
5
4
```

## 6.6 Continue Statement

When you want to skip to the next iteration but remain in the loop, you should use the continue statement.

For example:

```
#include <stdio.h>
int main() {
int nb = 7;
while (nb > 0) {
  nb--;
  if (nb == 5)
    continue;
 printf("%d\n", nb);
}}
```

## Output:

```
6
4
3
2
1
```

## 6.4 Some Loop Problem

**Problem 1**:  Write a C program to find the sum of first 10 natural numbers.

```
#include <stdio.h>
int main()
{
    int  j, sum = 0;

    printf("The first 10 natural number is :\n");

    for (j = 1; j <= 10; j++)
    {
        sum = sum + j;
        printf("%d ",j);
    }
    printf("\nThe Sum is : %d\n", sum);
}
```

**Output:**

```
The first 10 natural number is:
1 2 3 4 5 6 7 8 9 10
The Sum is : 55
```

**Problem 2**: Write a program in C to display the cube of the number upto a given integer.

```
#include <stdio.h>
int main()
 {
    int i,ctr;
    printf("Input number of terms : ");
    scanf("%d", &ctr);
    for(i=1;i<=ctr;i++)
    {
         printf("Number is : %d and cube of the %d is :%d \n",i,i, (i*i*i));
    }
 }
```

**Output:**

```
Input number of terms : 5
Number is : 1 and cube of the 1 is :1
Number is : 2 and cube of the 2 is :8
Number is : 3 and cube of the 3 is :27
Number is : 4 and cube of the 4 is :64
Number is : 5 and cube of the 5 is :125
```

**Problem 3**: Write a program in C to display the multiplication table of a given integer.

```
#include <stdio.h>
int main()
{
   int j,n;
   printf("Input the number (Table to be calculated) : ");
   scanf("%d",&n);
   printf("\n");
   for(j=1;j<=10;j++)
   {
     printf("%d X %d = %d \n",n,j,n*j);
   }
}
```

**Output:**

```
Input the number (Table to be calculated) : 15

15 X 1 = 15
15 X 2 = 30
15 X 3 = 45
15 X 4 = 60
15 X 5 = 75
15 X 6 = 90
15 X 7 = 105
15 X 8 = 120
15 X 9 = 135
15 X 10 = 150
```

**Problem 4**: Write a program in C to display the multiplication table vertically from 1 to n.

```
#include <stdio.h>
int main()
{
   int j,i,n;
   printf("Input upto the table number starting from 1 : ");
   scanf("%d",&n);
   printf("Multiplication table from 1 to %d \n",n);
   for(i=1;i<=10;i++)
   {
     for(j=1;j<=n;j++)
     {
       if (j<=n-1)
           printf("%dx%d = %d, ",j,i,i*j);
          else
            printf("%dx%d = %d",j,i,i*j);

      }
     printf("\n");
    }
}
```

**Output:**

```
Input upto the table number starting from 1 : 8
Multiplication table from 1 to 8
1x1 = 1, 2x1 = 2, 3x1 = 3, 4x1 = 4, 5x1 = 5, 6x1 = 6, 7x1 = 7, 8x1 = 8
1x2 = 2, 2x2 = 4, 3x2 = 6, 4x2 = 8, 5x2 = 10, 6x2 = 12, 7x2 = 14, 8x2 = 16
1x3 = 3, 2x3 = 6, 3x3 = 9, 4x3 = 12, 5x3 = 15, 6x3 = 18, 7x3 = 21, 8x3 = 24
1x4 = 4, 2x4 = 8, 3x4 = 12, 4x4 = 16, 5x4 = 20, 6x4 = 24, 7x4 = 28, 8x4 = 32
1x5 = 5, 2x5 = 10, 3x5 = 15, 4x5 = 20, 5x5 = 25, 6x5 = 30, 7x5 = 35, 8x5 = 40
1x6 = 6, 2x6 = 12, 3x6 = 18, 4x6 = 24, 5x6 = 30, 6x6 = 36, 7x6 = 42, 8x6 = 48
1x7 = 7, 2x7 = 14, 3x7 = 21, 4x7 = 28, 5x7 = 35, 6x7 = 42, 7x7 = 49, 8x7 = 56
1x8 = 8, 2x8 = 16, 3x8 = 24, 4x8 = 32, 5x8 = 40, 6x8 = 48, 7x8 = 56, 8x8 = 64
1x9 = 9, 2x9 = 18, 3x9 = 27, 4x9 = 36, 5x9 = 45, 6x9 = 54, 7x9 = 63, 8x9 = 72
1x10 = 10, 2x10 = 20, 3x10 = 30, 4x10 = 40, 5x10 = 50, 6x10 = 60, 7x10 = 70,
8x10 = 80
```

**Problem 5**: Write a program in C to display the n terms of odd natural number and their sum like:1 3 5 7 ... n

```
#include <stdio.h>
int main()
{
   int i,n,sum=0;

   printf("Input number of terms : ");
   scanf("%d",&n);
   printf("\nThe odd numbers are :");
   for(i=1;i<=n;i++)
   {
     printf("%d ",2*i-1);
     sum+=2*i-1;
   }
   printf("\nThe Sum of odd Natural Number upto %d terms : %d \n",n,sum);
}
```

**Output:**

```
Input number of terms : 10

The odd numbers are :1 3 5 7 9 11 13 15 17 19
The Sum of odd Natural Number upto 10 terms : 100
```

**Problem 6:** Write a program in C to make such a pattern like a pyramid with a number which will repeat the number in the same row.

The pattern is as follows:

```
   1
  2 2
 3 3 3
4 4 4 4
```

```
#include <stdio.h>

int main()
{
   int i,j,spc,rows,k;
   printf("Input number of rows : ");
   scanf("%d",&rows);
   spc=rows+4-1;
   for(i=1;i<=rows;i++)
   {
         for(k=spc;k>=1;k--)
            {
              printf(" ");
            }

           for(j=1;j<=i;j++)
           printf("%d ",i);
        printf("\n");
    spc--;
   }
}
```

**Output:**

```
Input number of rows : 5
        1
       2 2
      3 3 3
     4 4 4 4
    5 5 5 5 5
```

**Problem 7:** Write a program in C to find the sum of the series [ 1-X^2/2!+X^4/4!-.........].

```
#include <stdio.h>

int main()
{
        float x,s,t,num=1.00,fac=1.00;
        int i,n,pr,y=2,m=1;

        printf("Input the Value of x :");
        scanf("%f",&x);
        printf("Input the number of terms : ");
        scanf("%d",&n);
        s=1.00; t=1.00;

        for (i=1;i<n;i++)
        {
            for(pr=1;pr<=y;pr++)
                 {
                   fac=fac*pr;
                   num=num*x;

                 }
          m=m*(-1);
          num=num*m;
          t=num/fac;
          s=s+t;
          y=y+2;
          num=1.00;
          fac=1.00;
        }
        printf("\nthe sum = %f\nNumber of terms = %d\nvalue of x =
%f\n",s,n,x);
}
```

**Output:**

```
Input the Value of x :2
Input the number of terms : 5
the sum = -0.415873
Number of terms = 5
value of x = 2.000000
```

**Problem 8:** Write a program in C to display the n terms of harmonic series and their sum. The series is : 1 + 1/2 + 1/3 + 1/4 + 1/5 ... 1/n terms

```
#include <stdio.h>
int main()
{
   int i,n;
   float s=0.0;
   printf("Input the number of terms : ");
   scanf("%d",&n);
   printf("\n\n");
   for(i=1;i<=n;i++)
   {
       if(i<n)
       {
     printf("1/%d + ",i);
     s+=1/(float)i;
       }
     if(i==n)
     {
     printf("1/%d ",i);
     s+=1/(float)i;
     }
     }
        printf("\nSum of Series upto %d terms : %f \n",n,s);
}
```

**Output:**

```
Input the number of terms : 5

1/1 + 1/2 + 1/3 + 1/4 + 1/5
Sum of Series upto 5 terms : 2.283334
```

**Problem 9:** Write a c program to check whether a given number is a perfect number or not.

```
/*Perfect number is a positive number which sum of all positive divisors
excluding that number is equal to that number. For example 6 is perfect number
since divisor of 6 are 1, 2 and 3.  Sum of its divisor is 1 + 2+ 3 = 6*/
#include <stdio.h>

int main()
{
  int n,i,sum;
  int mn,mx;

  printf("Input the  number : ");
  scanf("%d",&n);
    sum = 0;
 printf("The positive divisor  : ");
    for (i=1;i<n;i++)
      {
      if(n%i==0)
         {
         sum=sum+i;
         printf("%d  ",i);
         }
       }
printf("\nThe sum of the divisor is : %d",sum);
    if(sum==n)
      printf("\nSo, the number is perfect.");
    else
      printf("\nSo, the number is not perfect.");
printf("\n");
}
```

**Output:**

```
Input the  number : 56
The positive divisor  : 1  2  4  7  8  14  28
The sum of the divisor is : 64
So, the number is not perfect.
```

**Problem 10:** Write a program in C to display the number in reverse order.

```c
#include <stdio.h>

int main(){
    int num,r,sum=0,t;

    printf("Input a number: ");
    scanf("%d",&num);

    for(t=num;num!=0;num=num/10){
         r=num % 10;
         sum=sum*10+r;
    }
printf("The number in reverse order is : %d \n",sum);
}
```

**Output:**

```
Input a number: 12345
The number in reverse order is: 54321
```

# Chapter 7

# Array, String &pointer

## 7.1 Array

Array in memory is stored as a continuous sequence of bytes. Like variables we give name to an array. However, unlike variables, arrays are multi-valued they contain multiple values. Hence you cannot access specific array element directly.

For example, you can write sum = 432; to access sum. But you cannot access specific array element directly by using array variable name. You cannot write marks to access 4th student marks.

In array, we use an integer value called index to refer at any element of array. Array index starts from 0 and goes till N - 1 (where N is size of the array). In above case array index ranges from 0 to 4.

To access individual array element we use array variable name with index enclosed within square brackets [ and]. To access first element of marks array, we use marks[0]. Similarly to access third element we use marks[2].

How to declare an array?

Syntax to declare an array.

data_type array_name[SIZE];

data_type is a valid C data type that must be common to all array elements.

array_name is name given to array and must be a valid C identifier.

SIZE is a constant value that defines array maximum capacity.

Example to declare an array

```
int marks[5];
```

How to initialize an array?

**There are two ways to initialize an array.**

Static array initialization - Initializes all elements of array during its declaration.

Dynamic array initialization - The declared array is initialized some time later during execution of program.

Static initialization of array

We define value of all array elements within a pair of curly braces { and } during its declaration. Values are separated using comma , and must be of same type.

Example of static array initialization

```
int marks[5] = {90, 86, 89, 76, 91};
```

Note: Size of array is optional when declaring and initializing array at once. The C compiler automatically determines array size using number of array elements. Hence, you can write above array initialization as.

```
int marks[] = {90, 86, 89, 76, 91};
```

Dynamic initialization of array

You can assign values to an array element dynamically during execution of program. First declare array with a fixed size. Then use the following syntax to assign values to an element dynamically.

```
array_name[index] = some_value;
```

**Problem 1:**

write a C program to declare an array capable of storing 10 student marks. Input marks of all 10 students and find their average.

```
#include <stdio.h>

#define SIZE 10 // Size of the array

int main()
{
    int marks[SIZE]; // Declare an array of size 10
    int index, sum;
    float avg;

    printf("Enter marks of %d students: ", SIZE);
/* Input marks of all students in marks array */
    for(index=0; index<SIZE; index++)
    {
        scanf("%d", &marks[index]);
    }

    /* Find sum of all marks */
    sum = 0;
    for(index=0; index<SIZE; index++)
    {
        sum = sum + marks[index];
    }

    /* Calculate average of marks*/
    avg = (float) sum / SIZE;

    /* Print the average marks */
    printf("Average marks = %f", avg);

    return 0;
}
```

**Output:**

```
Enter marks of 10 students: 90 86 89 76 91 95 80 77 82 93
Average marks = 85.900002
```

**Problem 2:** Write a program in C to store elements in an array and print it.

Test Data:

Input 10 elements in the array:

element - 0: 1

element - 1: 1

element - 2: 2

```
#include <stdio.h>

int  main()
{
    int arr[10];
    int i;
       printf("\n\nRead and Print elements of an array:\n");
       printf("-----------------------------------------\n");

    printf("Input 10 elements in the array :\n");
    for(i=0; i<10; i++)
    {
      printf("element - %d : ",i);
        scanf("%d", &arr[i]);
    }

    printf("\nElements in array are: ");
    for(i=0; i<10; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}
```

**Output:**

```
Read and Print elements of an array:
-----------------------------------------
Input 10 elements in the array :
element - 0 : 1
element - 1 : 1
element - 2 : 2
element - 3 : 3
element - 4 : 4
element - 5 : 5
element - 6 : 6
element - 7 : 7
element - 8 : 8
element - 9 : 9

Elements in array are: 1  1  2  3  4  5  6  7  8  9
```

**Problem 3:** Write a program in C to read n number of values in an array and display it in reverse order.

Test Data:

Input the number of elements to store in the array :3

Input 3 number of elements in the array:

element - 0: 2

element - 1: 5

element - 2: 7

Expected Output:

The values store into the array are:

2 5 7

The values store into the array in reverse are:

7 5 2

```
#include <stdio.h>

int main()
{
   int i,n,a[100];

       printf("\n\nRead n number of values in an array and display it in
reverse order:\n");
```

```
       printf("-------------------------------------------------------------
---------\n");

   printf("Input the number of elements to store in the array :");
   scanf("%d",&n);

   printf("Input %d number of elements in the array :\n",n);
   for(i=0;i<n;i++)
      {
    printf("element - %d : ",i);
    scanf("%d",&a[i]);
    }

   printf("\nThe values store into the array are : \n");
   for(i=0;i<n;i++)
     {
     printf("% 5d",a[i]);
   }

   printf("\n\nThe values store into the array in reverse are :\n");
   for(i=n-1;i>=0;i--)
      {
     printf("% 5d",a[i]);
    }
   printf("\n\n");
return 0;
}
```

**Output:**

```
Read n number of values in an array and display it in reverse order:
------------------------------------------------------------------------
Input the number of elements to store in the array :3
Input 3 number of elements in the array :
element - 0 : 2
element - 1 : 5
element - 2 : 7

The values store into the array are :
    2    5    7

The values store into the array in reverse are :
    7    5    2
```

**Problem 4:** Write a program in C to copy the elements of one array into another array.

```
#include <stdio.h>

int main()
{
    int arr1[100], arr2[100];
    int i, n;


       printf("\n\nCopy the elements one array into another array :\n");
       printf("----------------------------------------------------\n");

       printf("Input the number of elements to be stored in the array :");
       scanf("%d",&n);

       printf("Input %d elements in the array :\n",n);
       for(i=0;i<n;i++)
        {
        printf("element - %d : ",i);
        scanf("%d",&arr1[i]);
      }
    /* Copy elements of first array into second array.*/
    for(i=0; i<n; i++)
    {
        arr2[i] = arr1[i];
    }
    /* Prints the elements of first array   */
    printf("\nThe elements stored in the first array are :\n");
    for(i=0; i<n; i++)
    {
        printf("% 5d", arr1[i]);
    }
    /* Prints the elements copied into the second array. */
    printf("\n\nThe elements copied into the second array are :\n");
    for(i=0; i<n; i++)
    {
        printf("% 5d", arr2[i]);
    }
         printf("\n\n");
return 0;
}
```

**Output:**

```
Input the number of elements to be stored in the array :3
Input 3 elements in the array :
element - 0 : 15
element - 1 : 10
element - 2 : 12
Expected Output :
The elements stored in the first array are :
15 10 12
The elements copied into the second array are :
15 10 12
```

**Problem 5:** Write a program in C to find the maximum and minimum element in an array.

Test Data :

Input the number of elements to be stored in the array :3

Input 3 elements in the array :

element - 0 : 45

element - 1 : 25

element - 2 : 21

Expected Output :

Maximum element is : 45

```
#include <stdio.h>

void main()
{
    int arr1[100];
    int i, mx, mn, n;


       printf("\n\nFind maximum and minimum element in an array :\n");
       printf("--------------------------------------------------\n");

       printf("Input the number of elements to be stored in the array :");
       scanf("%d",&n);
```

```
        printf("Input %d elements in the array :\n",n);
        for(i=0;i<n;i++)
            {
         printf("element - %d : ",i);
         scanf("%d",&arr1[i]);
       }


    mx = arr1[0];
    mn = arr1[0];

    for(i=1; i<n; i++)
    {
        if(arr1[i]>mx)
        {
            mx = arr1[i];
        }


        if(arr1[i]<mn)
        {
            mn = arr1[i];
        }
    }
    printf("Maximum element is : %d\n", mx);
    printf("Minimum element is : %d\n\n", mn);
}
```

**Output:**

```
Find maximum and minimum element in an array :
--------------------------------------------------
Input the number of elements to be stored in the array :3
Input 3 elements in the array :
element - 0 : 45
element - 1 : 25
element - 2 : 21
Maximum element is : 45
```

**Problem 6:** Write a program in C to separate odd and even integers in separate arrays.

```
#include <stdio.h>

int main()
 {
    int arr1[10], arr2[10], arr3[10];
    int i,j=0,k=0,n;


       printf("\n\nSeparate odd and even integers in separate arrays:\n");
       printf("------------------------------------------------------\n");

       printf("Input the number of elements to be stored in the array :");
       scanf("%d",&n);

       printf("Input %d elements in the array :\n",n);
       for(i=0;i<n;i++)
            {
        printf("element - %d : ",i);
        scanf("%d",&arr1[i]);
      }

    for(i=0;i<n;i++)
    {
  if (arr1[i]%2 == 0)
  {
     arr2[j] = arr1[i];
     j++;
  }
  else
  {
     arr3[k] = arr1[i];
     k++;
  }
    }

    printf("\nThe Even elements are : \n");
    for(i=0;i<j;i++)
    {
  printf("%d ",arr2[i]);
    }

    printf("\nThe Odd elements are :\n");
    for(i=0;i<k;i++)
    {
  printf("%d ", arr3[i]);
    }
    printf("\n\n");
return 0;
 }
```

**Output:**

```
Separate odd and even integers in separate arrays:
------------------------------------------------------
Input the number of elements to be stored in the array :5
Input 5 elements in the array :
element - 0 : 25
element - 1 : 47
element - 2 : 42
element - 3 : 56
element - 4 : 32

The Even elements are :
42 56 32
The Odd elements are :
25 47
```

## 7.2 Multi-dimensional array in

Two-dimensional array

Two-dimensional array is a collection of one-dimensional array. Two-dimensional array has special significance than other array types. You can logically represent a two-dimensional array as a matrix. Any matrix problem can be converted easily to a two-dimensional array.

**Syntax to declare two-dimensional array:**

type array_name[row-size][col-size];

1.type is a valid C data type.

2.array_name is a valid C identifier that denotes name of the array.

3.row-size is a constant that specifies matrix row size.

4.col-size is also a constant that specifies column size. col-size is optional when initializing array during its declaration.

**Example program to use two-dimensional array**

Write a C program to declare a two-dimensional array of size 4x3. Read values in each element of array from user and display values of all elements.

```
/
 * C program to input and display two-dimensional array
 */

#include <stdio.h>

#define ROW_SIZE 4 // Define constant row size
#define COL_SIZE 3 // Define constant column size

int main()
{
    int matrix[ROW_SIZE][COL_SIZE];
    int row, col;

    printf("Enter elements in matrix of size %dx%d \n", ROW_SIZE, COL_SIZE);

    /* Outer loop to iterate through each row */
    for(row=0; row<ROW_SIZE; row++)
    {
        /* Inner loop to iterate through columns of each row */
        for(col=0; col<COL_SIZE; col++)
        {
            /* Input element in array */
            scanf("%d", &matrix[row][col]);
        }
    }
    /*
     * Print all elements of array
     */
    printf("\nElements in matrix are: \n");
    for(row=0; row<ROW_SIZE; row++)
    {
        for(col=0; col<COL_SIZE; col++)
        {
            printf("%d ", matrix[row][col]);
        }
        printf("\n");
    }

    return 0;
}
```

**Output:**

```
Enter elements in matrix of size 4x3
10 20 30
40 50 60
70 80 90
100 110 120

Elements in matrix are:
10 20 30
40 50 60
70 80 90
100 110 120
```

## Example program to use three-dimensional array

```
int main()
{
    int arr[SIZE1][SIZE2][SIZE3];
    int i, j, k;

    /*
     * Input elements in array
     */
    printf("Enter elements in three-dimensional array of size %dx%dx%d \n",
SIZE1, SIZE2, SIZE3);
    for(i = 0; i < SIZE1; i++)
    {
        for(j = 0; j < SIZE2; j++)
        {
          for (k = 0; k < SIZE3; k++)
          {
              scanf("%d", &arr[i][j][k]);
          }
        }
    }
/*
     * Print elements of array
     */
    printf("\nElements in three-dimensional array are: \n");
    for(i = 0; i < SIZE1; i++)
    {
        for(j = 0; j < SIZE2; j++)
        {
          for (k = 0; k < SIZE3; k++)
          {
              printf("%d\n", arr[i][j][k]);
          }
        }
    }

    return 0;
}
```

**Output:**

```
Enter elements in three-dimensional array of size 2x2x3
1
2
3
4
5
6
7
8
9
10
11
12
```

```
Elements in three-dimensional array are:
1
2
3
4
5
6
7
8
9
10
11
12
```

**Problem 6:** Write a program in C for a 2D array of size 3x3 and print the matrix. Go to the editor  Test Data :

Input elements in the matrix :

element - [0],[0] : 1

element - [0],[1] : 2

element - [0],[2] : 3

element - [1],[0] : 4

element - [1],[1] : 5

element - [1],[2] : 6

element - [2],[0] : 7

element - [2],[1] : 8

element - [2],[2] : 9

Expected Output :

The matrix is :

1 2 3

4 5 6

7 8 9

```
#include <stdio.h>

int main()
{
  int arr1[3][3],i,j;

       printf("\n\nRead a 2D array of size 3x3 and print the matrix :\n");
       printf("------------------------------------------------------\n");


    /* Stored values into the array*/
       printf("Input elements in the matrix :\n");
  for(i=0;i<3;i++)
  {
      for(j=0;j<3;j++)
      {
        printf("element - [%d],[%d] : ",i,j);
        scanf("%d",&arr1[i][j]);
      }
  }

 printf("\nThe matrix is : \n");
  for(i=0;i<3;i++)
  {
      printf("\n");
      for(j=0;j<3;j++)
           printf("%d\t",arr1[i][j]);
  }
 printf("\n\n");
return 0;
}
```

**Output:**

```
Read a 2D array of size 3x3 and print the matrix :
------------------------------------------------------
Input elements in the matrix :
element - [0],[0] : 1
element - [0],[1] : 2
element - [0],[2] : 3
element - [1],[0] : 4
element - [1],[1] : 5
element - [1],[2] : 6
element - [2],[0] : 7
element - [2],[1] : 8
element - [2],[2] : 9

The matrix is :

1       2       3
4       5       6
7       8       9
```

**Problem 8:** Write a program in C for subtraction of two Matrices.

Test Data :

Input the size of the square matrix (less than 5): 2

Input elements in the first matrix :

element - [0],[0] : 5

element - [0],[1] : 6

element - [1],[0] : 7

element - [1],[1] : 8

Input elements in the second matrix :

element - [0],[0] : 1

element - [0],[1] : 2

element - [1],[0] : 3

element - [1],[1] : 4

Expected Output :

The First matrix is :

5 6

7 8

The Second matrix is :

1 2

3 4

The Subtraction of two matrix is :

4 4

4 4

```
#include <stdio.h>

int main()
{
  int arr1[50][50],brr1[50][50],crr1[50][50],i,j,n;

       printf("\n\nSubtraction of two Matrices :\n");
       printf("-----------------------------\n");
        printf("Input the size of the square matrix (less than 5): ");
       scanf("%d", &n);

    /* Stored values into the array*/
       printf("Input elements in the first matrix :\n");
       for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
             printf("element - [%d],[%d] : ",i,j);
             scanf("%d",&arr1[i][j]);
            }
        }

       printf("Input elements in the second matrix :\n");
       for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
             printf("element - [%d],[%d] : ",i,j);
             scanf("%d",&brr1[i][j]);
            }
        }
   printf("\nThe First matrix is :\n");
  for(i=0;i<n;i++)
    {
      printf("\n");
      for(j=0;j<n;j++)
           printf("%d\t",arr1[i][j]);
    }

  printf("\nThe Second matrix is :\n");
  for(i=0;i<n;i++)
    {
      printf("\n");
      for(j=0;j<n;j++)
      printf("%d\t",brr1[i][j]);
    }
/* calculate the subtraction of the matrix */
   for(i=0;i<n;i++)
       for(j=0;j<n;j++)
            crr1[i][j]=arr1[i][j]-brr1[i][j];
   printf("\nThe Subtraction of two matrix is : \n");
   for(i=0;i<n;i++){
       printf("\n");
       for(j=0;j<n;j++)
            printf("%d\t",crr1[i][j]);
   }
   printf("\n\n");
return 0;
}
```

**Output:**

```
Subtraction of two Matrices :
------------------------------
Input the size of the square matrix (less than 5): 2
Input elements in the first matrix :
element - [0],[0] : 5
element - [0],[1] : 6
element - [1],[0] : 7
element - [1],[1] : 8
Input elements in the second matrix :
element - [0],[0] : 1
element - [0],[1] : 2
element - [1],[0] : 3
element - [1],[1] : 4

The First matrix is :
5       6
7       8
```

```
The Second matrix is :
1       2
3       4
The Subtraction of two matrix is :
4       4
4       4
```

## 7.3 String

What is a String?

String is nothing but a collection of characters in a linear sequence. 'C' always treats a string a single data even though it contains whitespaces. A single character is defined using single quote representation. A string is represented using double quote marks.

Example, "Welcome to the world of programming!"

Declare and initialize a String

A string is a simple array with char as a data type. 'C' language does not directly support string as a data type. Hence, to display a string in 'C', you need to make use of a character array.

The general syntax for declaring a variable as a string is as follows,

char string_variable_name [array_size];

The classic string declaration can be done as follow:

char string_name[string_length] = "string";

The size of an array must be defined while declaring a string variable because it used to calculate how many characters are going to be stored inside the string variable. Some valid examples of string declaration are as follows, char first_name[15];    //declaration of a string variable

char last_name[15];

The above example represents string variables with an array size of 15. This means that the given character array is capable of holding 15 characters at most. The indexing of array begins from 0 hence it will store characters from a 0-14 position. The C compiler automatically adds a NULL character '\0' to the character array created.

String Input: Read a String

When writing interactive programs which ask the user for input, C provides the scanf(), gets(), and fgets() functions to find a line of text entered from the user.

When we use scanf() to read, we use the "%s" format specifier without using the "&" to access the variable address because an array name acts as a pointer. For

**example:**

```
#include <stdio.h>
int main() {
char name[10];
int age;
printf("Enter your first name and age: \n");
scanf("%s %d", name, &age);
printf("You entered: %s %d",name,age);
}
```

**Output:**

```
Enter your first name and age:
John_Smith 48
```

The problem with the scanf function is that it never reads an entire string. It will halt the reading process as soon as whitespace, form feed, vertical tab, newline or a carriage return occurs. Suppose we give input as "Hello World" then the scanf function will never read an entire string as a whitespace character occurs between the two names. The scanf function will only read Hello.

In order to read a string contains spaces, we use the gets() function. Gets ignores the whitespaces. It stops reading when a newline is reached (the Enter key is pressed).

**For example:**

```
#include <stdio.h>
int main() {
char full_name[25];
printf("Enter your full name: ");
gets(full_name);
printf("My full name is %s ",full_name);
return 0;
}
```

**Output:**

```
Enter your full name: Kazi umar
My full name is Kazi umar
```

**String Output: Print/Display a String**

The standard printf function is used for printing or displaying a string on an output device. The format specifier used is %s

Example,

printf("%s", name);

String output is done with the puts() and printf() functions.

## puts function

The puts function prints the string on an output device and moves the cursor back to the first position. A puts function can be used in the following way,

```
#include <stdio.h>
int main() {
char name[15];
gets(name);        //reads a string
puts(name);        //displays a string
return 0;}
The syntax of this function is comparatively simple than other functions.
```

## 7.4 Pointer

**What is a pointer?**

A pointer is a variable that stores memory address. If it is a variable, it must have a valid C data type. Yes, every pointer variable has a data type associated with it. Which means an integer pointer can hold only integer variable addresses.

**How to declare pointer variable**

Once you got basics of memory addresses, reference and dereference operator. Let us declare our first pointer variable.

Pointer variable declaration follows almost similar syntax as of normal variable.

**Syntax to declare pointer variable**

data-type * pointer-variable-name;

data-type is a valid C data type.

* symbol specifies it is a pointer variable. You must prefix * before variable name to declare it as a pointer.

pointer-variable-name is a valid C identifier i.e. the name of pointer variable.

Example to declare pointer variable

int * ptr;

**How to initialize pointer variable**

There are two ways to initialize a pointer variable. You can use reference operator & to get memory location of a variable or you can also directly assign one pointer variable to other pointer variable.

Examples to initialize pointer variable

int num   = 10;

int *ptr  = &num;   // Assign address of num to ptr

// You can also assign a pointer variable to another

int *ptr1 = ptr;    // Initialize pointer using another pointer

How pointers are stored in memory

You got a basic picture of pointer working. Let us take a closer look on how pointer variables are stored in memory. Consider the following statements

int num  = 10;

int *ptr = &num;

**Example program to use pointers**

Write a C program to demonstrate the use of pointers in C programming.

```
#include <stdio.h>

int main()
{
    int num = 1;
    int *ptr = &num;    // ptr points to num

    printf("Value of num    = %d \n", num);
    printf("Address of num = %x \n\n", &num);


    printf("Value of ptr          = %x \n", ptr);
    printf("Address of ptr        = %x \n", &ptr);
    printf("Value pointed by ptr = %d \n\n", *ptr);


    /* Change the value of num directly */
    num = 10;
    printf("After changing value of num directly. \n");
    printf("Value of num          = %d \n", num);
    printf("Value pointed by ptr = %d \n\n", *ptr);


    /* Assigns 100 at the address pointed by ptr */
    *ptr = 100;
    printf("After changing value pointed by ptr. \n");
    printf("Value of num          = %d \n", num);
    printf("Value pointed by ptr = %d \n", *ptr);

    return 0;
}
```

**Output:**

```
Value of num   = 1
Address of num = 60ff0c

Value of ptr          = 60ff0c
Address of ptr        = 60ff08
Value pointed by ptr = 1

After changing value of num directly.
Value of num          = 10
Value pointed by ptr = 10

After changing value pointed by ptr.
Value of num          = 100
Value pointed by ptr = 100
```

Working of above program

1.int *ptr = &num; declares an integer pointer that points at num.

2.The first two printf() in line 12 and 13 are straightforward. First prints value of num and other prints memory address of num.

3.printf("Value of ptr = %x \n", ptr); prints the value stored at ptr i.e. memory address of num. Hence, the statement prints memory address of num.

4.printf("Address of ptr = %x \n", &ptr); prints the address of ptr.

Don't confuse with address of ptr and address pointed by ptr. First ptr is a variable so it will have a memory address which is retrieved using &ptr. And since it is a pointer variable hence it stores memory address which is retrieved using ptr.

5.printf("Value pointed by ptr = %d \n\n", *ptr);, here * dereferences value pointed by ptr and prints the value at memory location pointed by ptr.

6.Next, we made some changes to num i.e. num=10. After changes printf("Value of num = %d \n", num); prints 10.

7.Since we made changes to our original variable num, hence changes are reflected back to pointer that points to the num. *ptr in line 23, dereferences value pointed by ptr i.e. 10.

8.*ptr = 100; says assign 100 to memory location pointed by ptr. Which means, assign 100 to num indirectly.

9.Since, we again modified the value of num using *ptr = 100. Hence, num and *ptr in line 28 and 29 will evaluate to 100.

## 7.5 structure

**What is structure in C?**

Structure is a user defined data type. It is a collection of different data type, to create a new data type.

For example, You can define your custom type for storing student record containing name, age and mobile. Creating structure type will allow you to handle all properties (fields) of student with single variable, instead of handling it separately.

**How to declare?**

Syntax to define a structure

```
struct structure_name
{
   member1_declaration;
   member2_declaration;
   ...
   ...
   memberN_declaration;
};
```

Here, structure_name is name of our custom type. memberN_declaration is structure member i.e. variable declaration that structure will have.

**Example to define a structure**

```
struct student
{
   char name[40];      // Student name
   int  age;           // Student age
   unsigned long mobile; // Student mobile number };
```

## 7.6 Functions

## What is function

A function is a collection of statements grouped together to do some specific task. In series of learning C programming, we already used many functions unknowingly. Functions such as - printf(), scanf(), sqrt(), pow() or the most important the main() function. Every C program has at least one function i.e. the main() function.

Function declaration

Like variable declarations, functions are also declared. Function declaration tells the compiler that there exists a function with some name that may be used later in the program. Function declaration is also known as function prototype or function signature.

Syntax of function declaration

return_type function_name( parameter_list );

1.Return type - Return type defines the data type of value returned by the function.

A function does some calculation and may return a resultant value. So that the result of one function can be used by another function. For example - sqrt() function returns square root of given number, which is later used by calling function.

You must mention return type as void if your function does not return any value.

2.Function name - Function name is a valid C identifier that uniquely identifies the function. You must follow identifier naming rules while naming a function.

3.Parameter list - A function may accept input. Parameter list contains input type and variable name given to the function. Multiple inputs are separated using comma ,.

A function declaration must be terminated using semicolon ;. You can declare a function anywhere in the program. However it is best practice to declare functions below pre-processor directives.

Syntax of function definition

```
return_type function_name(parameter list)
{
   // Function body
}
```

Function declaration and definition syntax must be same. You are free to put function definition anywhere in your program, below its declaration. However, I recommend you to put all declarations below main() function.

**Function calling**

The most important part of function is its calling. Calling of a function in general is execution of the function. It transfers program control from driver (current) function to called function. We can optionally pass input to the calling function.

**Syntax of function call**

function_name( parameter_list );

Function name - Name of the function to execute.

Parameter list - Comma separated input given to the function. Parameter list must match type and order mentioned in function declaration. Leave parameter list as blank if function does not accept any input.

**Function example program**

Let us write a C program to input two numbers from user. Find sum of the given two number using function.

```
#include <stdio.h>

/* Addition function declaration */
int add(int num1, int num2);


/* Main function definition */
int main()
{
    /* Variable declaration */
    int n1, n2, sum;

    /* Input two numbers from user */
    printf("Enter two numbers: ");
    scanf("%d%d", &n1, &n2);

    /*
     * Addition function call.
     * n1 and n2 are parameters passed to add function.
     * Value returned by add() is stored in sum.
     */
    sum = add(n1, n2);

    /* Print value of sum */
    printf("Sum = %d", sum);

    return 0;
}

/**
 * Addition function definition.
 *
 * Return type of the function is int.
 * num1 - First parameter of the function of int type.
 * num2 - Second parameter of the function of int type.
 */
int add(int num1, int num2)
{
    int s = num1 + num2;

    /* Return value of sum to the main function */
    return s;
}
```

**Output:**

```
Enter two numbers: 10 20
Sum = 30
```

T h a n k y o u