

Unit – 1

Computer Language

As we know, to communicate with a person, we need a specific language, similarly to communicate with computers, programmers also need a language is called Programming language.

What is Language?

Language is a mode of communication that is used to **share ideas, opinions with each other**. For example, if we want to teach someone, we need a language that is understandable by both communicators.

What is a Programming Language?

A programming language is a **computer language** that is used by **programmers (developers) to communicate with computers**. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

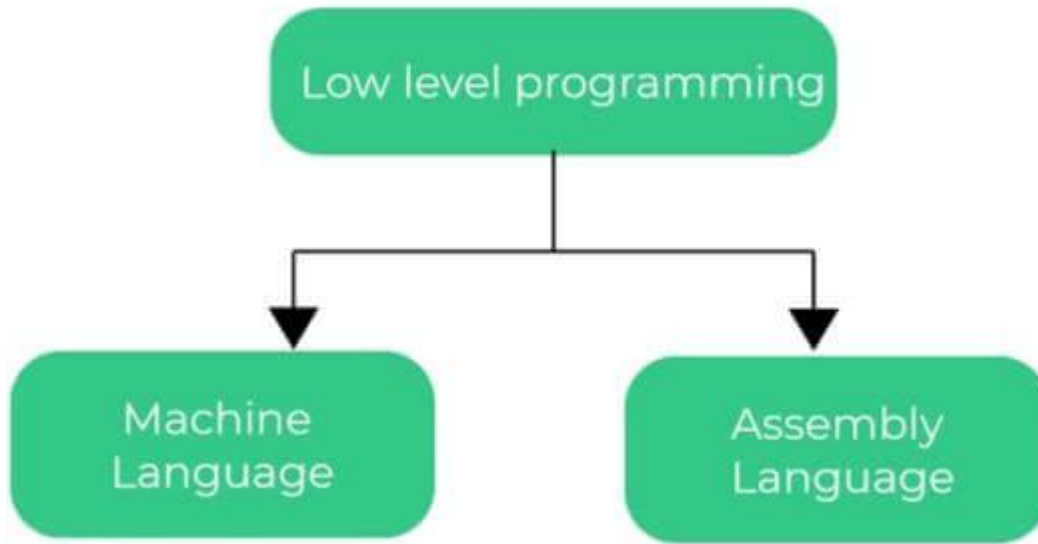
Types of programming language

1. Low-level programming language

Low-level language is **machine-dependent (0s and 1s)** programming language. The processor runs low-level programs directly without the need of a compiler or interpreter, so the programs written in low-level language can be run very fast.

Low-level language is further divided into two parts –

Low Level Programming In C Language



i. Machine Language

The machine-level language is a language that consists of a set of instructions that are in the binary form 0 or 1. As we know that computers can understand only machine instructions, which are in binary digits, i.e., 0 and 1, so the instructions given to the computer can be only in binary codes. Creating a program in a machine-level language is a very difficult task as it is not easy for the programmers to write the program in machine instructions. It is error-prone as it is not easy to understand, and its maintenance is also very high. A machine-level language is not portable as each computer has its machine instructions, so if we write a program in one computer will no longer be valid in another computer.

Advantages

1. Fast execution
2. No need of translator

Disadvantages

1. Difficult to learn
2. Difficult to find errors

3. Difficult to modify
4. Machine dependent

ii. Assembly Language

The assembly language contains some human-readable commands such as mov, add, sub, etc. Since assembly language instructions are written in English words like mov, add, sub, so it is easier to write and understand. As we know that computers can only understand the machine-level instructions, so we require a translator that converts the assembly code into machine code. The translator used for translating the code is known as an assembler.

The assembly language code is not portable because the data is stored in computer registers, and the computer has to know the different sets of registers.

Advantages

1. Easy to learn
2. Easy to find errors
3. Easy to modify

Disadvantages

1. Slow Execution
2. Limited Mnemonics
3. Machine Dependent

2. High-level programming language

The high-level language is a programming language that allows a programmer to write the programs which are independent of a particular type of computer. The high-level languages are considered as high-level because they are closer to human languages than machine-level languages.

When writing a program in a high-level language, then the whole attention needs to be paid to the logic of the problem. A compiler is required to translate a high-level language into a low-level language.

Advantages

1. High level languages are programmer friendly. They are easy to write, debug and maintain.

2. It is machine independent language.
3. Easy to learn.
4. Less error prone, easy to find and debug errors.
5. High level programming results in better programming productivity.

Disadvantages

1. Slow Execution
2. Required a translator
3. High Memory required

Translators

Mostly languages like Java, C++, Python, Assembly and more are used to write the programs, called source code. These source codes need to translate into machine language to be executed because they cannot be executed directly by the computer. Hence, a special translator program, a language processor, is used to convert source code into machine language.

Types of language processors (Translators)

There are mainly three kinds of language processors, which are discussed below:

1. **Assembler:** An assembler converts programs written in assembly language into machine code. It is also referred to assembler as assembler language by some users. The source program has assembly language instructions, which is an input of the assembler. The assembler translates this source code into a code that is understandable by the computer, called object code or machine code.



2. **Compiler:** The language processor allows the computer to run and understand the program by reading the complete source program in one time, which is written in a high-level language. The computer can then interpret this code because it is translated into machine language. While working on the Harvard

Mark I computer, Grace Hopper created the first compiler.



3. **Interpreter:** An interpreter is a computer program that allows a computer to interpret or understand what tasks to perform. The programs written with the help of using one of the many high-level programming languages are directly executed by an interpreter without previously converting them to an object code or machine code, which is done line by line or statement by statement. When the interpreter is translating the source code, it displays an error message if there is an error in the statement and terminates this statement from translating process. When the interpreter removed errors on the first line, then it moves on to the next line.



Introduction to C Language

The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc. C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways:

1. Mother language
2. System programming language
3. Procedure-oriented programming language
4. Structured programming language
5. Mid-level programming language

History of C Language

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

Dennis Ritchie is known as the **founder of the c language**. It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Initially, C language was developed to be used in **UNIX operating system**.

These programming languages that were developed before C language -

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee
C11	2011	Standardization Committee
C18	2018	Standardization Committee

Features of C Language

1. Simple and Efficient

C is a simple language in the sense that it provides a **structured approach** (to break the problem into parts), **the rich set of library functions, data types**, etc.

2. Portability

Unlike assembly language, c programs **can be executed on different machines** with some machine specific changes. Therefore, C is a machine independent language or portable language.

3. Structured Programming language

C is a structured programming language in the sense that **we can break the program into parts using functions**. So, it is easy to understand and modify. Functions also provide code reusability.

4. Rich Standard Library

C **provides a lot of inbuilt functions** that make the development fast.

5. Mid-level programming language

Although, C is **intended to do low-level programming**. It is used to develop system applications such as kernel, driver, etc. It **also supports the features of a high-level language**. That is why it is known as mid-level language.

6. Memory Management

It supports the feature of **dynamic memory allocation**. In C language, we can free the allocated memory at any time by calling the **free()** function.

7. Pointer

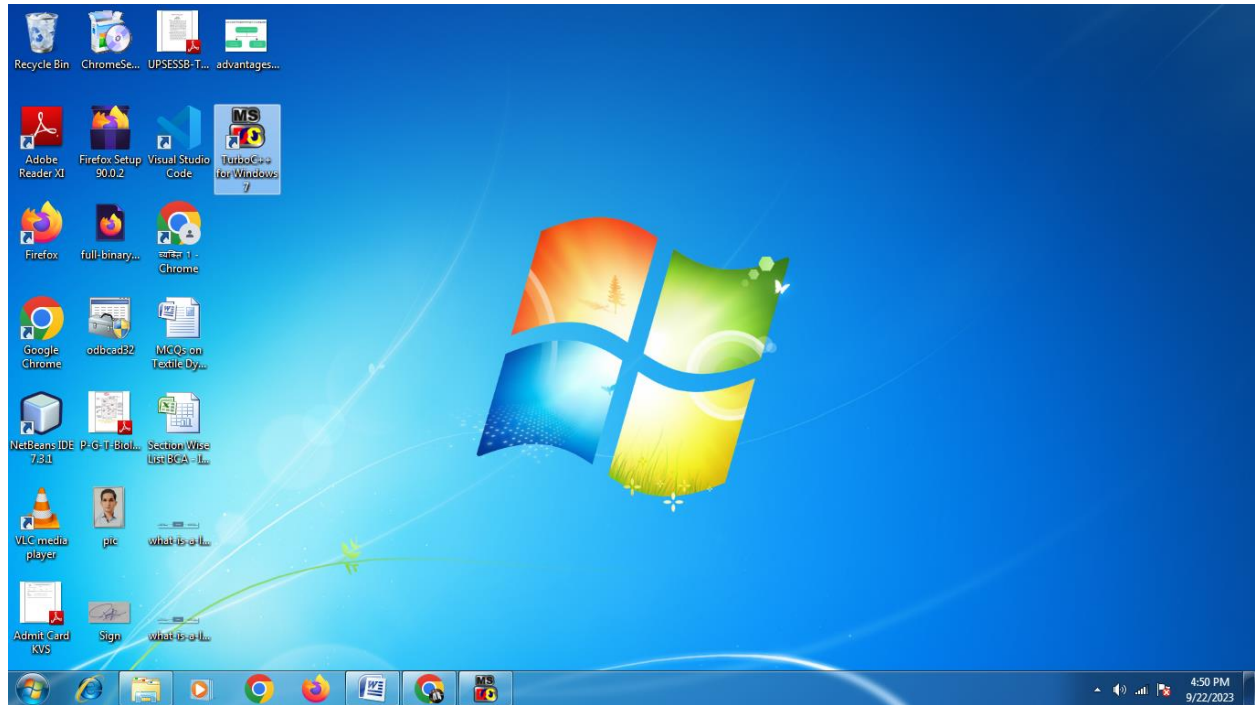
C provides the feature of pointers. We can directly interact with the memory by using the pointers. We **can use pointers for memory, structures, functions, array**, etc.

8. Recursion

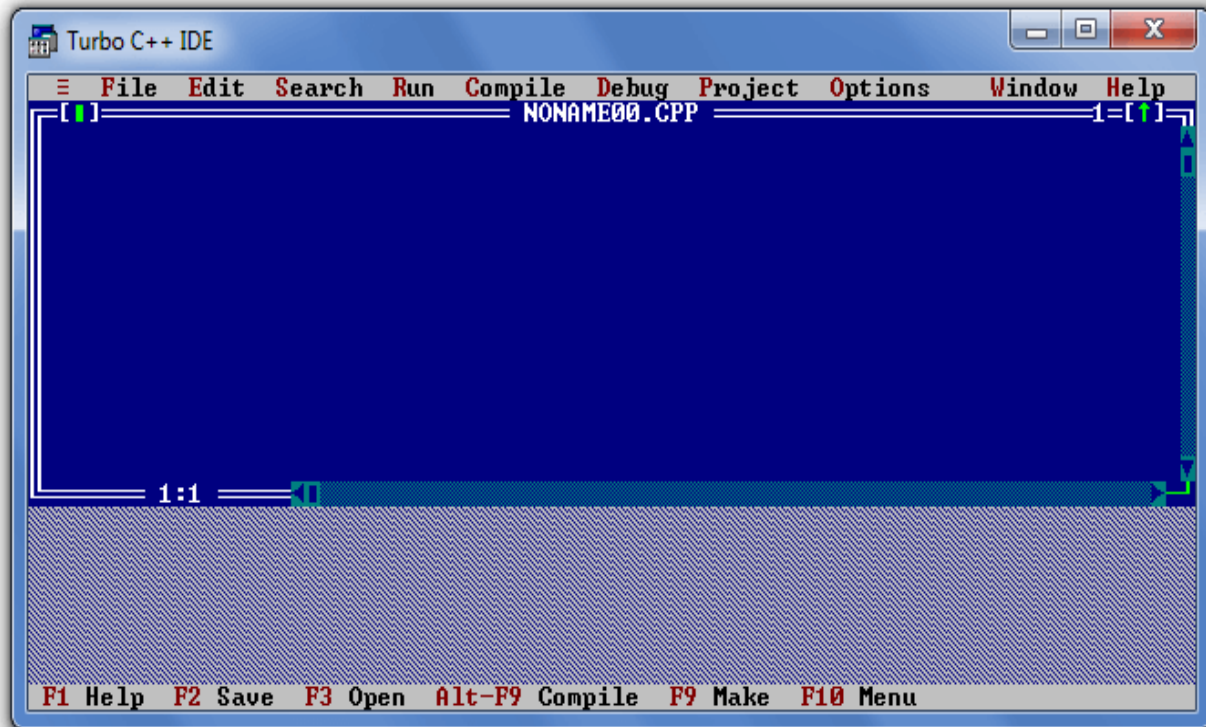
In C, we **can call the function within the function**. It provides code reusability for every function. Recursion enables us to use the approach of backtracking.

How to open Turbo C++

Double click on Turbo C++ icon from the desktop.



It open Turbo C++ IDE as-



First C Program

To write the first c program, open the Turbo C++ IDE and write the following code:

```
#include <stdio.h>
void main()
{
printf("Hello C Language");
}
```

Before compile the program, it must be saved with any name but with extension .c-

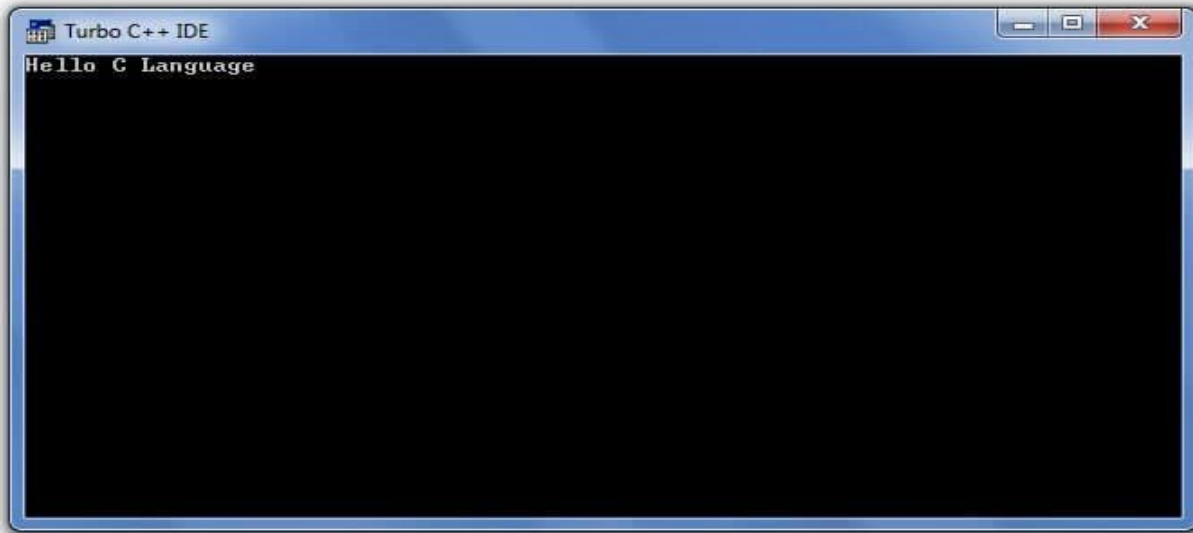
Save the Program : Demo.c

To Compile: Press Alt + F9

To Run: Ctrl + F9

To Show Output: Alt + F5

You will see the following output on user screen.



Now **press any key** to return to the turbo c++ screen.

Here,

#include <stdio.h> includes the **standard input output** library functions. The printf() function is defined in stdio.h .

void main() The **main() function is the entry point of every program** in c language.

printf() The printf() function is **used to print data** on the console screen.

Other Examples

Program 1: WAP to print “Hello C” Message

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
```

```
printf("Hello C");  
getch();  
}
```

Output:

Hello C

Program 2: WAP to print Your Name

```
#include <stdio.h>  
#include <conio.h>  
void main()  
{  
clrscr();  
printf("Pramod Kumar");  
getch();  
}
```

Output:

Pramod Kumar

Program 3: WAP to print “Hello How Are You” Message

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

```
void main()
{
clrscr();
printf("Hello How Are You");
getch();
}
```

Output:

Hello How Are You

Program 4: WAP to print “C is a Middle Level Language” Message

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
printf("C is a Middle Level Language");
getch();
}
```

Output:

C is a Middle Level Language

Program 5: WAP to print a Message as- Hello How Are

You

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("Hell0");
    printf("\n How");
    printf("\n Are");
    printf("\n You");
    getch();
}
```

Output:

Hello

How

Are

You

Program 6: WAP to print a Message as-

Hello How Are You

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("Hell0");
    printf("\t How");
```

```
printf("\t Are");  
printf("\t You");  
getch();  
}
```

Output:

Hello How Are You

Program 6: WAP to print a Message as- Hello

How

Are

You

```
#include <stdio.h>  
#include <conio.h>  
void main()  
{  
clrscr();  
printf("Hell0");  
printf("\n\t How");  
printf("\n\t\t Are");  
printf("\n\t\t\t You");  
getch();  
}
```

Output:

Hello

How

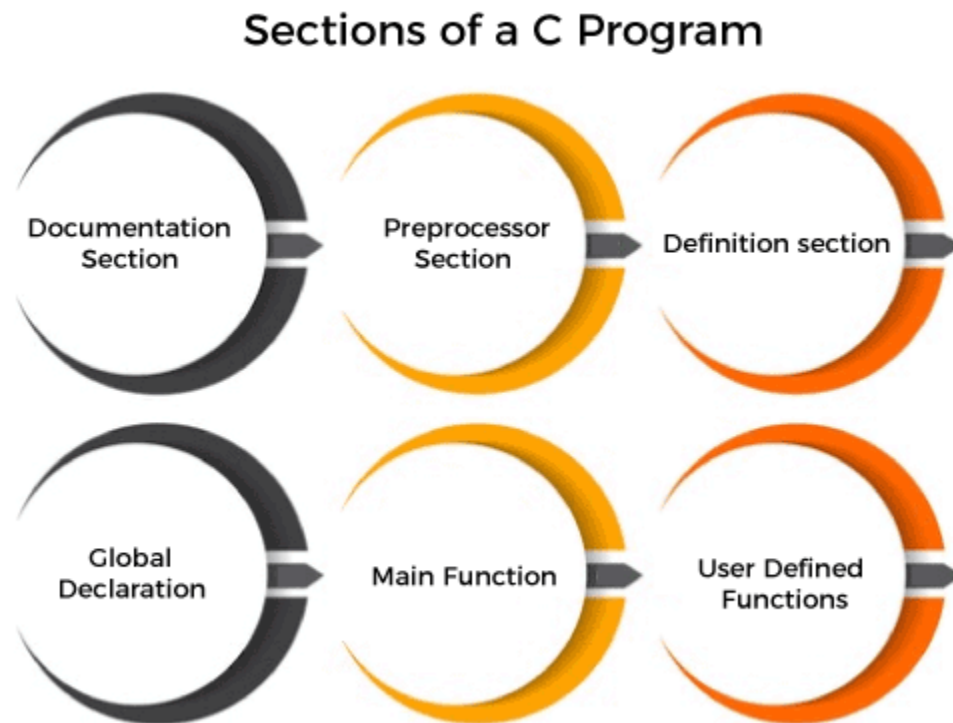
Are

You

Structure of a C program

The structure of a C program means the specific structure to start the programming in the C language. Without a proper structure, it becomes difficult to analyze the problem and the solution. It also gives us a reference to write more complex programs.

Sections of a C program



The sections of a C program are listed below:

1. Documentation section
2. Preprocessor section
3. Definition section
4. Global declaration
5. Main function
6. User defined functions

1. Documentation section

It includes the statement specified at the beginning of a program, such as a program's **name**, **date**, **description**, and **title**. It is represented as:

```
//name of a program
```

Or

```
/*
```

Overview of the code

```
.....
```

```
.....
```

```
*/
```

2. Preprocessor section (Linking Section)

The preprocessor section contains all the header files used in a program. It informs the system to link the header files to the system libraries. It is given by:

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

```
#include<conio.h>
```

3. Define section (Declaration Section)

The define section comprises of different constants declared using the define keyword. It is given by:

```
#define a = 2
```

```
#define PI = 3.14
```

```
void sum();
```

```
int result();
```

4. Global declaration

The global section comprises of all the global declarations in the program. It is given by:

```
float num = 2.54;
```

```
int a = 5;
```

```
char ch ='z';
```

5. Main function

main() is the first function to be executed by the computer. It is necessary for a code to include the main(). It is like any other function available in the C library.

```
main()
```

We can also use int or main with the main (). The void main() specifies that the program will not return any value. The int main() specifies that the program can return integer type data.

```
int main()
```

Or

```
void main()
```

6. User defined functions

The user defined functions specified the functions specified as per the requirements of the user. For example, color(), sum(), division(), etc.

```
void sum()  
{  
.....  
.....  
.....  
}
```

Compilation process in c

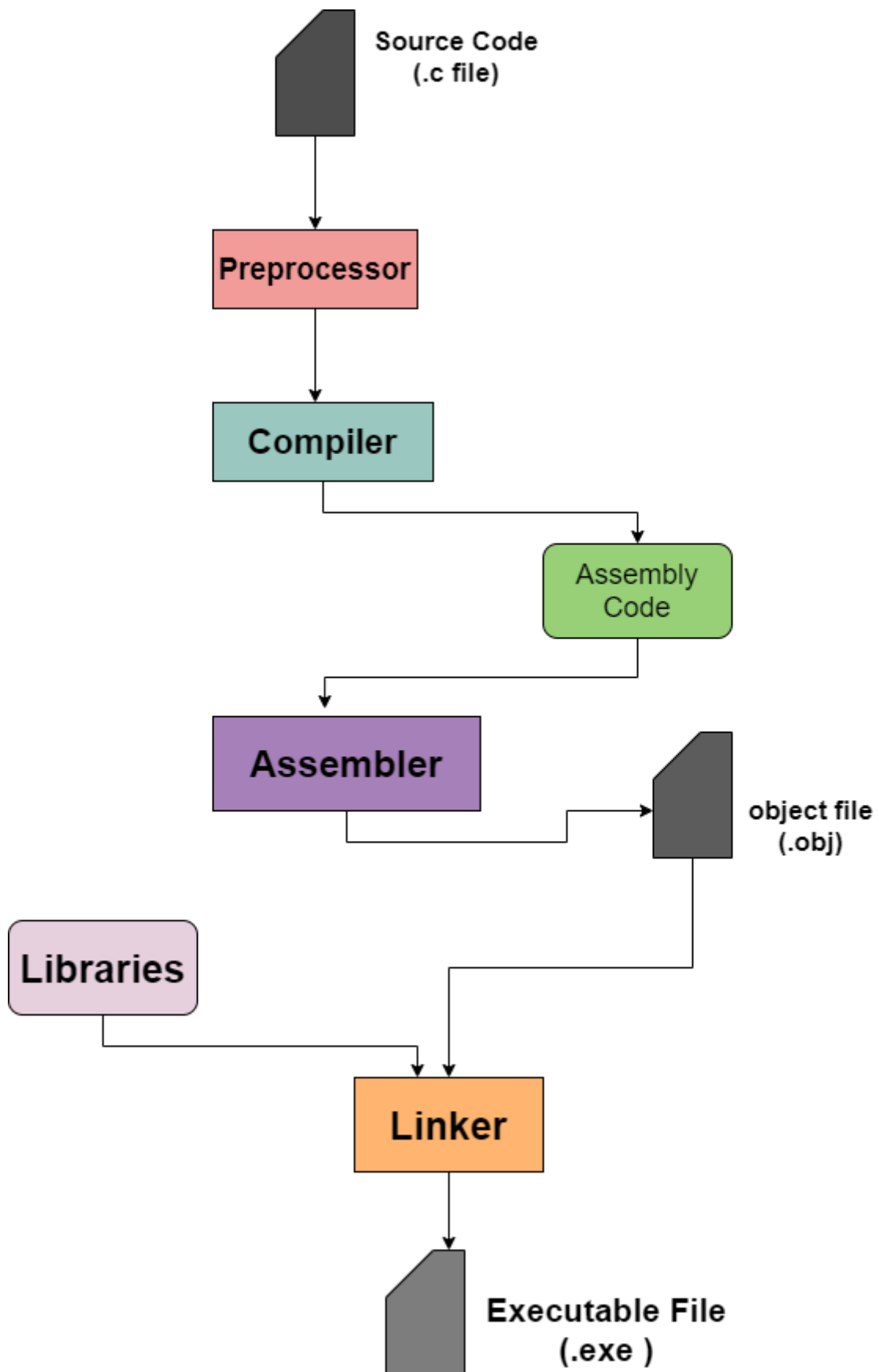
What is a compilation?

The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.

The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

The following are the phases through which our program passes before being transformed into an executable form:

- **Preprocessor**
- **Compiler**
- **Assembler**
- **Linker**



Preprocessor

The source code is the code which is written in a text editor and the source code file is given an extension ".c". This source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

Compiler

The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

Assembler

The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. The extension of the object file in DOS is '.obj,' If the name of the source file is '**hello.c**', then the name of the object file would be 'hello.obj'.

Linker

Mainly, all the programs written in C use library functions. These library functions are pre-compiled, and the object code of these library files is stored with '.lib' extension. The main working of the linker is to combine the object code of library files with the object code of our program.

C Character Set

As every language contains a set of characters used to construct words, statements, etc., C language also has a set of characters which include alphabets, digits, and special symbols. C language supports a total of 256 characters.

Every C program contains statements. These statements are constructed using words and these words are constructed using characters from C character set. C language character set contains the following set of characters...

1. Alphabets
2. Digits
3. Special Symbols

Alphabets

The C programming language provides support for all the alphabets that we use in the English language. Thus, in simpler words, a C program would easily support a total of 52 different characters- 26 uppercase and 26 lowercase.

Type of Character	Description	Characters
Lowercase Alphabets	a to z	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
Uppercase Alphabets	A to Z	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

Digits

The C programming language provides the support for all the digits that help in constructing/ supporting the numeric values or expressions in a program. These range

from 0 to 9. Thus, the C language supports a total of 10 digits for constructing the numeric values or expressions in any program.

Type of Character	Description	Characters
Digits	0 to 9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special Characters

We use some special characters in the C language for some special purposes, such as logical operations, mathematical operations, checking of conditions, backspaces, white spaces, etc.

The C programming language provides support for the following types of special characters:

Type of Character	Examples
Special Characters	` ~ @ ! \$ # ^ * % & () [] { } < > + = _ - / \ ; : ' " , . ?

White Spaces

The white spaces in the C programming language contain the following:

- Blank Spaces
- Carriage Return
- Tab
- New Line

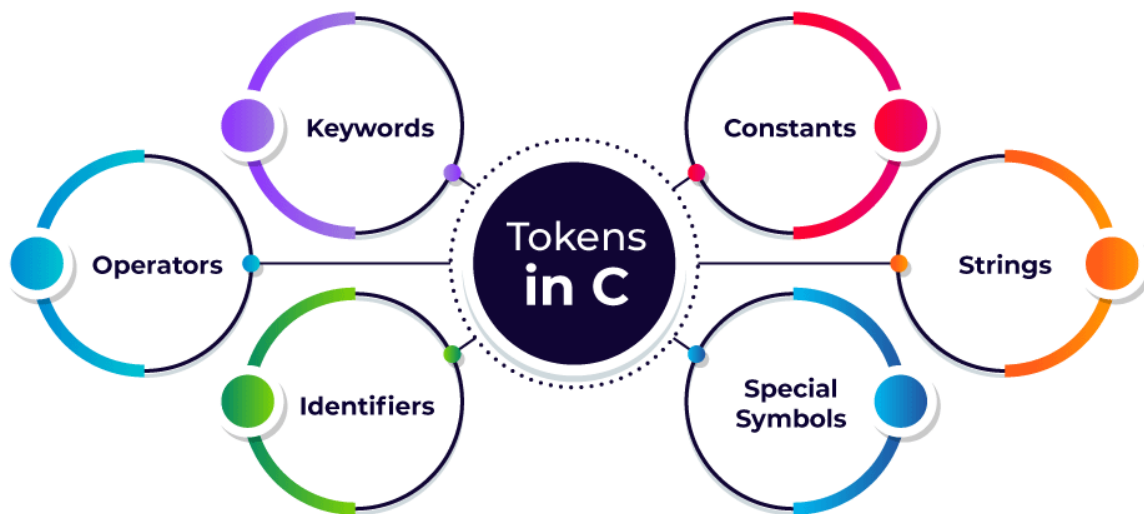
Tokens in C

Tokens in C is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language.

Types of tokens in C

Tokens in C language can be divided into the following categories:

,



Keywords

Keywords in C can be defined as the **pre-defined** or the **reserved words** having its own importance, and each keyword has its own functionality. C language supports 32 keywords given below:

auto	double	int	struct
break	else	long	switch

case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers

Identifiers in C are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words. It can be composed of uppercase letters, lowercase letters, underscore, or digits, but the starting letter should be either an underscore or an alphabet. Identifiers cannot be used as keywords. Rules for constructing identifiers in C are given below:

1. The first character of an identifier should be either an alphabet or an underscore.
2. It should not begin with any numerical digit.
3. In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
4. Commas or blank spaces cannot be specified within an identifier.
5. Keywords cannot be represented as an identifier.
6. The length of the identifiers should not be more than 31 characters.

Strings

Strings in C are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of

the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

```
char a[10] = "Welcome"; // The compiler allocates the 10 bytes to the 'a' array.
```

```
char a[] = "Welcome"; // The compiler allocates the memory at the run time.
```

```
char a[10] = {'W','e','l','c','o','m','e','\0'}; // String is represented in the form of characters.
```

Operators

Operators in C is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

Unary Operator

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.

Binary Operator

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators

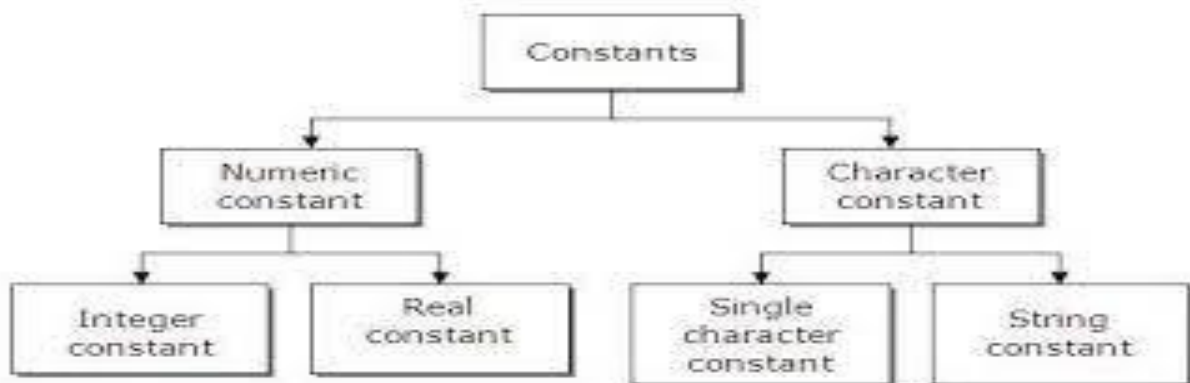
- Bitwise Operators
- Conditional Operators
- Assignment Operator

Constants

A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed.

There are two ways of declaring constant:

- Using `const` keyword
- Using `#define` pre-processor



Constant	Example
Integer constant	10, 11, 34, etc.
Floating-point constant	45.6, 67.8, 11.2, etc.
Octal constant	011, 088, 022, etc.
Hexadecimal constant	0x1a, 0x4b, 0x6b, etc.
Character constant	'a', 'b', 'c', etc.

String constant	"java", "c++", ".net", etc.
-----------------	-----------------------------

Special characters

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

- **Square brackets []:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ():** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call.
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.

Variables in C

A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Syntax to declare a variable:

```
datatype variablename;
```

Example of declaring the variable is given below:

```
int a;
```

```
float b;
```

```
char c;
```

Here, a, b, c are variables. The int, float, char are the data types.

We can also declare multiple variables at the same time as:

```
int a, b, c;
```

```
float x, y;
```

Initialization of variables:

```
A=10;
```

```
B=20.56;
```

```
C='x';
```

We can also declare and initialize a variable together as-

```
int a=10;
```

```
float b=20.56;
```

```
char c='A';
```

Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Valid variable names:

```
int a;
```

```
int _ab;
```

```
int a30;
```

Invalid variable names:

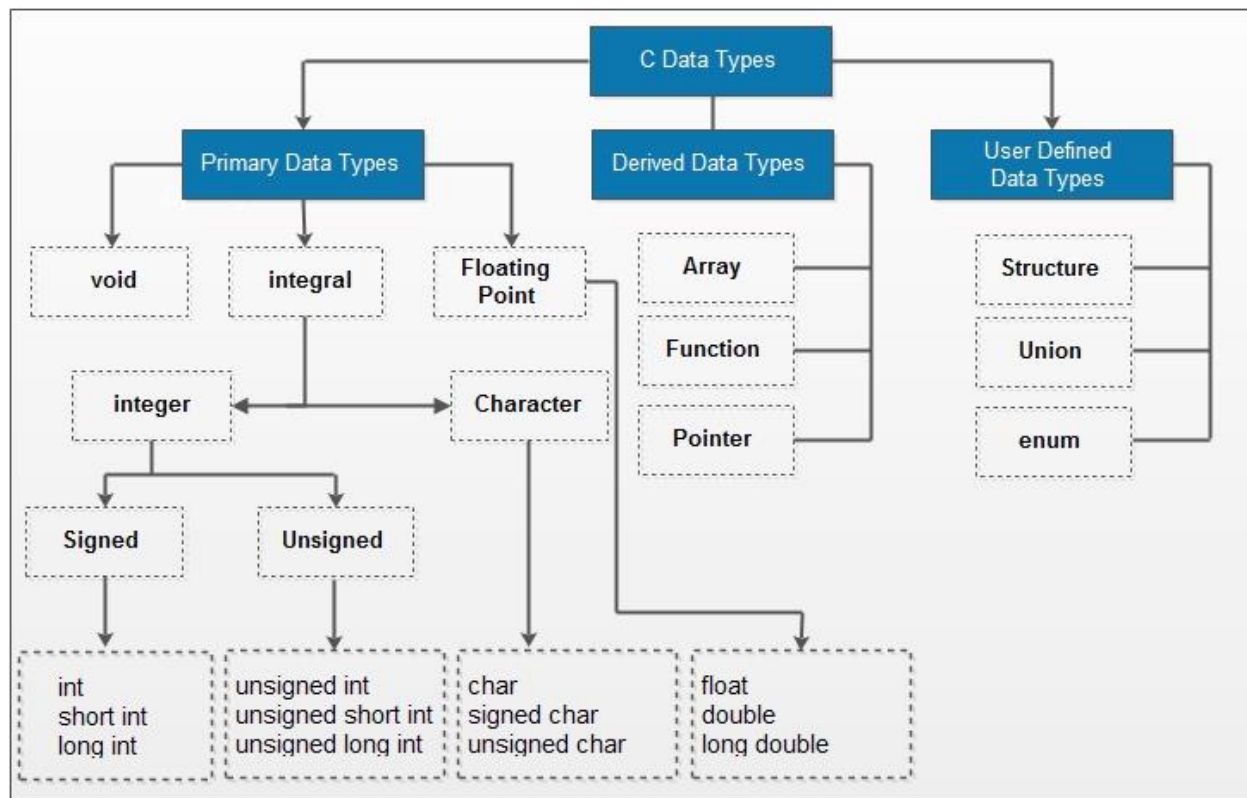
```
int 2;
```

```
int a b;
```

```
int long;
```

Datatypes in c

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc. Each data type requires different amounts of memory and has some specific operations which can be performed over it. The data type is a collection of data with values having fixed values, meaning as well as its characteristics.



There are the following data types in C language.

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union

Enumeration Data Type	Enum
Void Data Type	Void

Basic Data Types

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given **according to 32-bit architecture**.

Data Types	Memory Size	Range
Char	1 byte	–128 to 127
signed char	1 byte	–128 to 127
unsigned char	1 byte	0 to 255
Short	2 byte	–32,768 to 32,767
signed short	2 byte	–32,768 to 32,767
unsigned short	2 byte	0 to 65,535
Int	2 byte	–32,768 to 32,767
signed int	2 byte	–32,768 to 32,767
unsigned int	2 byte	0 to 65,535

short int	2 byte	–32,768 to 32,767
signed short int	2 byte	–32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
Float	4 byte	3.4E-38 to 3.4E+38
double	8 byte	1.7E-308 to 1.7E+308
long double	10 byte	3.4E-4932 to 1.1E+4932

Format Specifier

The Format specifier is a string used in the formatted input and output functions. The format string determines the format of the input and output. The format string always starts with a '%' character.

The commonly used format specifiers in printf() function are:

Format specifier	Description
%d or %i	It is used to print the signed integer value where signed integer means that the variable can hold both positive and negative values.
%u	It is used to print the unsigned integer value where the unsigned integer means that the variable can hold only positive value.
%o	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value.

%x	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value. In this, alphabetical characters are printed in small letters such as a, b, c, etc.
%X	It is used to print the hexadecimal unsigned integer, but %X prints the alphabetical characters in uppercase such as A, B, C, etc.
%f	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'.
%e/%E	It is used for scientific notation. It is also known as Mantissa or Exponent.
%g	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.
%p	It is used to print the address in a hexadecimal form.
%c	It is used to print the unsigned character.
%s	It is used to print the strings.
%ld	It is used to print the long-signed integer value.

Escape Sequence in C

An **escape sequence** in the C programming language consists of a **backslash** (\) and a character that stands in for a **special character** or **control sequence**. During the compilation process, the **C compiler** substitutes any escape sequences it comes across with the **relevant character** or **control sequence**.

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark

Comments in C

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments
2. Multi-Line Comments

Single Line Comments

Single line comments are represented by double slash `\\`. Let's see an example of a single line comment in C.

Example 1:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    printf("Pramod");
    // printf("\nKumar");
    printf("\nSharma");
    getch();
}
```

Output:

Pramod

Sharma

Example 2:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    printf("Pramod");
    // printf("\nKumar");
    // printf("\nSharma");
    getch();
}
```

Output:

Pramod

Mult Line Comments

Multi-Line comments are represented by slash asterisk * ... *\ . It can occupy many lines of code, but it can't be nested. Syntax:

```
/*
```

```
code
```

```
to be commented
```

```
*/
```

Example :

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

```
#include<conio.h>
```

```
void main()
```

```
{
```

```
    clrscr();
```

```
    /* printf("Pramod");
```

```
    printf("\nKumar"); */
```

```
    printf("\nSharma");
```

```
    getch();
```

```
}
```

Output:

Sharma

Unit – 2

Operators

An operator is a symbol that tells to the compiler what operation is performed on the operands. C language is rich in built-in operators.

For example, '+' is an operator used for addition, as shown below:

```
c = a + b;
```

Here, '+' is the operator known as the addition operator and 'a' and 'b' are operands. The addition operator tells the compiler to add both of the operands 'a' and 'b'.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operator
- Shorthand Operator
- Comma Operator
- Increment and Decrement Operator
- Bitwise Operators
- Ternary or Conditional Operators

Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values.

Operators	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Example:

Suppose, a=20, b=3

Operand 1	Operand 2	Expression	Result
a	b	a+b	23
a	b	a-b	17
a	b	A*b	60
a	b	a/b	6
a	b	a%b	2

Program: WAP to perform arithmetic operations

```
#include<stdio.h>

#include<conio.h>

void main()
{
    int a=20, b=3;

    clrscr();

    printf("Addition is =%d", a+b);

    printf("\nSubtraction is =%d", a-b);

    printf("\nMultiplication is =%d", a*b);

    printf("\nDivision is =%d", a/b);

    printf("\nModulus is =%d", a%b);

    getch();
}
```

Output:

Addition is=23

Subtraction is =17

Multiplication is =60

Division =6

Modulus is =2

Relational Operators:

These types of operators check the relationship between two of the available operands. In case the relation happens to be true, then it returns to 1. But in case this relation turns out to be false, then it returns to the value 0. We use the relational operators in loops and in the cases of decision making.

Types of Relational Operators

If P=5 and Q=3, then:

Meaning	Operator	Details	Example
Equal to	==	(P == Q) is not true	4 == 5 gets evaluated to 0 4 == 4 gets evaluated to 1
Not equal to	!=	(P != Q) is true	4 != 5 gets evaluated to 1 4 != 4 gets evaluated to 0
Less than	<	(P < Q) is not true	4 < 2 gets evaluated to 0 4 < 6 gets evaluated to 1
Greater than	>	(P > Q) is true	4 > 2 gets evaluated to 1 4 > 9 gets evaluated to 0
Less than or equal to	<=	(P <= Q) is not true	4 <= 2 gets evaluated to 0 4 <= 6 gets evaluated to 1 4 <= 4 gets evaluated to 1
Greater than or equal to	>=	(P >= Q) is true	4 >= 2 It gets evaluated to 1 4 >= 9 It gets evaluated to 0 4 <= 4 gets evaluated to 1

Here, 0 means false and 1 means true.

Program of Relational Operators

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



```
int main()
{
int p = 9;
int b = 4;

printf("p > q: %d \n", p > q);
printf("p >= q: %d \n", p >= q);
printf("p <= q: %d \n", p <= q);
printf("p < q: %d \n", p < q);
printf("p == q: %d \n", p == q);
printf("p != q: %d \n", p != q);
}
```

The output generated here would be:

```
p > q: 1
p >= q: 1
p <= q: 0
p < q: 0
p == q: 0
p != q: 1
```

Logical Operators

Logical operators perform logical operations on a given expression by joining two or more expressions or conditions. It can be used in various relational and conditional expressions. This operator is based on Boolean values to logically check the condition, and if the conditions are true, it returns 1. Otherwise, it returns 0 (False).

Types of Logical operator:

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

Logical AND Operator

The logical AND operator is represented as the '&&' double ampersand symbol. It checks the condition of two or more operands by combining in an expression, and if all the conditions are true, the logical AND operator returns the Boolean value true or 1. Else it returns false or 0.

Syntax

(condition1 && condition2)

There are two conditions in the above syntax, condition1 and condition2, and in between the double (&&) ampersand symbol. If both the conditions are true, the logical AND operator returns Boolean value 1 or true. Otherwise, it returns false.

Truth table of the Logical AND (&&) operator

A	B	A && B
1	1	1
1	0	0
0	1	0
0	0	0

Program to demonstrate the Logical AND Operator in C

```

#include <stdio.h>
#include <conio.h>
int main ()
{
    int n = 20;
    clrscr();
    printf (" %d \n", (n == 20 && n >= 8));
    printf (" %d \n", (n >= 1 && n >= 20));
    printf (" %d \n", (n == 10 && n >= 0));
    printf (" %d \n", (n >= 20 && n <= 40));
    getch();
}

```

Output

```

1
1
0
1

```

Logical OR Operator

The logical OR operator is represented as the '||' double pipe symbol. It checks the condition of two or more operands by combining in an expression, and if all the conditions are false, the logical OR operator returns the Boolean value false or 0. Else it returns true or 1.

Syntax

```
(condition1 || condition2)
```

There are two conditions in the above syntax, condition1 and condition2, and in between the double (||) pipe symbol. If both the conditions are false, the logical OR operator returns Boolean value 0 or false. Otherwise, it returns true.

Truth table of the Logical OR (||) operator

A	B	A && B
---	---	--------

1	1	1
1	0	1
0	1	1
0	0	0

Program to demonstrate the Logical OR Operator in C

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int n = 20;
    clrscr();
    printf (" %d \n", (n == 20 || n >= 8));
    printf (" %d \n", (n >= 1 || n >= 20));
    printf (" %d \n", (n == 10 || n >= 0));
    printf (" %d \n", (n > 30 || n <= 40));
    getch();
}
```

Output

```
1
1
1
0
```

Logical NOT operator

The logical NOT operator is represented as the '!' symbol, which is used to reverse the result of any given expression or condition. If the result of an expression is non-zero or true, the

result will be reversed as zero or false value. Similarly, if the condition's result is false or 0, the NOT operator reverses the result and returns 1 or true.

For example, suppose the user enters a non-zero value is 5, the logical NOT (!) operator returns the 0 or false Boolean value. And if the user enters a zero (0) value, the operator returns the true Boolean value or 1.

Syntax of the logical NOT operator

`!(condition);`

Here, the '!' symbol represents the logical NOT operator, which inverses the result of the given condition.

The truth table of the logical NOT operator:

Following is the truth table of the logical not operator in C

Condition	!(condition)
1	0
0	1

Program to use the logical NOT operator in C

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int x = 5;
    clrscr();
    printf (" The value is = %d", ! (x == 5));
    printf (" \n The value = %d", ! (x != 5));
    printf (" \n The value = %d", ! (x >= 3));
    printf (" \n The value = %d", ! (x < 3));
    getch();
}
```

Output:

```
The value = 0
The value = 1
The value = 0
The value = 1
```

Assignment Operator

The assignment operator is used to assign the value, variable and function to another variable.

`A = 5; // use Assignment symbol to assign 5 to the operand A`

`B = A; // Assign operand A to the B`

`B = &A; // Assign the address of operand A to the variable B`

`A = 20 \ 10 * 2 + 5; // assign equation to the variable A`

Simple Assignment Operator (=):

It is the operator used to assign the right side operand or variable to the left side variable.

Syntax

int a = 5;

or

int b = a;

Program

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

```
#include <conio.h>
```

```
void main ()
```

```
{
```

```
    int n1, n2, c, x, y;
```

```
    clrscr();
```

```
    n1 = 5;
```

```
    n2 = n1;
```

```
    c = n1 + n2;
```

```
    x = 20 / 4 * 2 + 5;
```

```
    printf ("\n The value of n1: %d", n1);
```

```
printf (" \n The value of n2: %d", n2);
printf (" \n The value of c: %d", c);
printf (" \n The value of x: %d", x);
getch();
}
```

Output

```
The value of n1: 5
The value of n2: 5
The value of c: 10
The value of x: 15
```

Shorthand Assignment Operators

Plus and Assign Operator (+=):

The operator is used to add the left side operand to the left operand and then assign results to the left operand.

Syntax

A += B;

Or

A = A + B;

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int n1, n2, c;
    clrscr();
    n1 = 5;
    n2 = 10;
    n2 += n1;
    printf (" \n The value of n1: %d", n1);
    printf (" \n The value of n2: %d", n2);
```

```
getch();  
}
```

Output

```
The value of a: 5  
The value of b: 15
```

Subtract and Assign Operator (-=):

The operator is used to subtract the left operand with the right operand and then assigns the result to the left operand.

Syntax

```
A -= B;  
Or  
A = A - B;
```

Program

```
#include <stdio.h>  
#include <conio.h>  
  
int main ()  
{  
    int n1, n2, c;  
clrscr();  
    n1 = 5;  
    n2 = 10;  
    n2 -= n1;  
    printf (" \n The value of n1: %d", n1);  
    printf (" \n The value of n2: %d", n2);  
  
    getch();  
}
```

Output


```
The value of n1: 5
The value of n2: 5
```

Multiply and Assign Operator (*=)

The operator is used to multiply the left operand with the right operand and then assign result to the left operand.

Syntax

A *= B;

Or

A = A * B;

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int n1, n2, c;
    clrscr();
    n1 = 5;
    n2 = 10;
    n2 *= n1;
    printf (" \n The value of n1: %d", n1);
    printf (" \n The value of n2: %d", n2);

    getch();
}
```

Output

```
The value of n1: 5
The value of n2: 50
```

Divide and Assign Operator (/=):

An operator is used between the left and right operands, which divides the first number by the second number to return the result in the left operand.

Syntax

A /= B;

Or

A = A / B;

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int n1, n2, c;
    clrscr();
    n1 = 5;
    n2 = 10;
    n2 /= n1;
    printf (" \n The value of n1: %d", n1);
    printf (" \n The value of n2: %d", n2);
    getch();
}
```

Output

```
The value of n1: 5
The value of n2: 2
```

Modulus and Assign Operator (%=):

An operator used between the left operand and the right operand divides the first number (n1) by the second number (n2) and returns the remainder in the left operand.

Syntax

A %= B;

Or

A = A % B;

Program

```

#include <stdio.h>
#include <conio.h>
void main ()
{
    int n1, n2, c;
    clrscr();
    printf (" Enter the value of n1: ");
    scanf ("%d", &n1);
    printf (" \n Enter the value of n2: ");
    scanf ("%d", &n2);
    n1 %= n2;
    printf (" \n The modulus value of n1: %d", n1);
    getch();
}

```

Output

```

Enter the value of n1: 23
Enter the value of n2: 5
The modulus value of n2: 3

```

sizeof() operator

The **sizeof()** operator is commonly used in C. It determines the size of the expression or the data type specified in the number of char-sized storage units. The **sizeof()** operator contains a single operand which can be either an expression or a data typecast where the cast is data type enclosed within parenthesis.

```

#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("\nsize of the character data type is %d",sizeof(char));
    printf("\nsize of the integer data type is %d",sizeof(int));
    printf("\nsize of the floating data type is %d",sizeof(float));
}

```

```
printf("\nsize of the double data type is %d",sizeof(double));
```

```
getch();  
}
```

Output:

size of the character data type is 1

size of the integer data type is 2

size of the float data type is 4

size of the double data type is 8

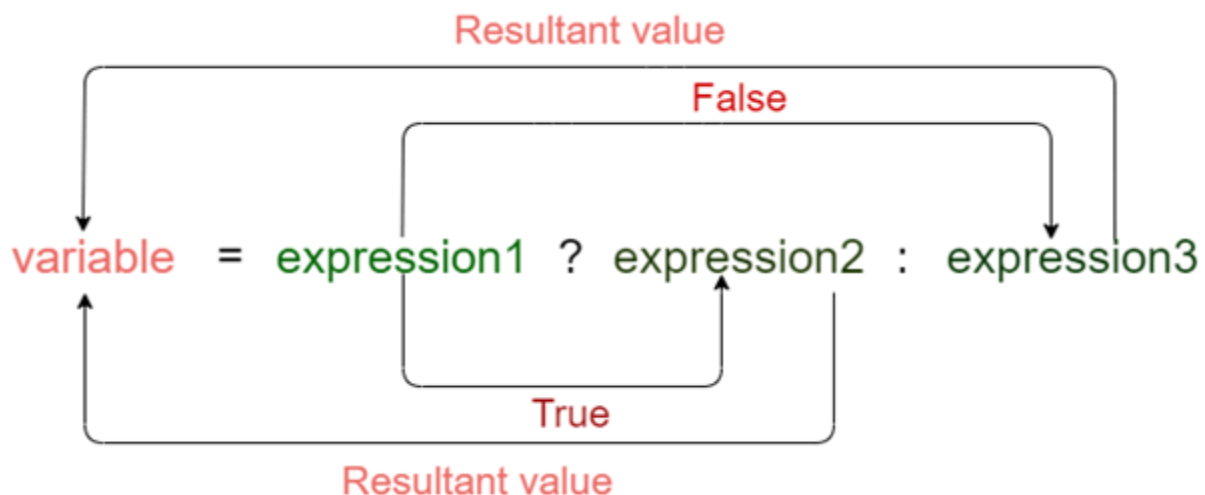
Conditional Operator

The conditional operator is also known as a **ternary operator**. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and ':'.

As conditional operator works on three operands, so it is also known as the ternary operator.

Syntax of a conditional operator

Expression1 ? expression2 : expression3;



Meaning of the above syntax.

- In the above syntax, the expression1 is a Boolean condition that can be either true or false value.
- If the expression1 results into a true value, then the expression2 will execute.
- The expression2 is said to be true only when it returns a non-zero value.
- If the expression1 returns false value then the expression3 will execute.
- The expression3 is said to be false only when it returns zero value.

Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("Enter two numbers =");
    scanf("%d%d",&a,&b);
    c = a>b ? a : b;
    printf("The greater number is =%d",c);
    getch();
}
```

Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("Enter two numbers =");
    scanf("%d%d",&a,&b);
    c = a<b ? a : b;
    printf("The smaller number is =%d",c);
    getch();
}
```

Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int age;
    clrscr();
    printf("Enter your age");
    scanf("%d",&age);
    (age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting"));
    getch();
}
```

Comma Operator

The comma sign is used for mainly two different purposes in the C language – as an operator and as a separator.

The Comma as a Separator

When we want to declare multiple numbers of variables in any program and provide different arguments in any function, we use the comma in the form of a separator in the program.

For instance,

```
int x, y, z;
```

In the statement mentioned above, the comma acts as a separator and informs the compiler that the x, y, and z variables are three different types of variables.

The Comma as an Operator

We use the comma in the form of an operator when we want to assign multiple numbers of values to any variable in a program with the help of a comma.

For example,

```
int a = (40, 50, 60, 70, 80, 90);
```

The value of a will be equivalent to 90. It is because the values 40, 50, 60, 70, 80 and 90 are enclosed in the form of braces () and these braces have a higher priority, as compared to the equal to (=) assignment operator.

Increment and Decrement Operators

Increment Operator

Increment Operators are the unary operators used to increment or add 1 to the operand value. The Increment operand is denoted by the double plus symbol (++). It has two types, Pre Increment and Post Increment Operators.

Pre-increment Operator

The pre-increment operator is used to increase the original value of the operand by 1 before assigning it to the expression.

Syntax

`X = ++A;`

In the above syntax, the value of operand 'A' is increased by 1, and then a new value is assigned to the variable 'B'.

Example 1: Program to use the pre-increment operator in C

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int x, y, z;
    printf (" Input the value of X: ");
    scanf (" %d", &x);
    printf (" Input the value of Y: ");
    scanf (" %d", &y);
    printf (" Input the value of Z: ");
    scanf (" %d", &z);
    ++x;
    ++y;
    ++z;
    printf (" \n The updated value of the X: %d ", x);
    printf (" \n The updated value of the Y: %d ", y);
    printf (" \n The updated value of the Z: %d ", z);
    getch();
```

```
}
```

Output

```
Input the value of X: 10
Input the value of Y: 15
Input the value of Z: 20

The updated value of the X: 11
The updated value of the Y: 16
The updated value of the Z: 21
```

Post increment Operator

The post-increment operator is used to increment the original value of the operand by 1 after assigning it to the expression.

Syntax

1. `X = A++;`

In the above syntax, the value of operand 'A' is assigned to the variable 'X'. After that, the value of variable 'A' is incremented by 1.

Example 2: Program to use the post-increment operator in C

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



```
void main ()
{
int x, y, z, a, b, c;
printf (" Input the value of X: ");
scanf (" %d", &x);
printf (" Input the value of Y: ");
scanf (" %d", &y);
printf (" Input the value of Z: ");
scanf (" %d", &z);
a = x++;
b = y++;
c = z++;
printf (" \n The original value of a: %d", a);
printf (" \n The original value of b: %d", b);
printf (" \n The original value of c: %d", c);
printf (" \n\n The updated value of the X: %d ", x);
printf (" \n The updated value of the Y: %d ", y);
printf (" \n The updated value of the Z: %d ", z);
getch();
}
```

Output

Input the value of X: 10
Input the value of Y: 15
Input the value of Z: 20

The original value of a: 10
The original value of b: 15
The original value of c: 20

The updated value of the X: 11
The updated value of the Y: 16
The updated value of the Z: 21

Decrement Operator

Decrement Operator is the unary operator, which is used to decrease the original value of the operand by 1. The decrement operator is represented as the double minus symbol (--). It has two types, Pre Decrement and Post Decrement operators.

Pre Decrement Operator

The Pre Decrement Operator decreases the operand value by 1 before assigning it to the mathematical expression. In other words, the original value of the operand is first decreases, and then a new value is assigned to the other variable.

Syntax

```
B = --A;
```

In the above syntax, the value of operand 'A' is decreased by 1, and then a new value is assigned to the variable 'B'.

Example 3: Program to demonstrate the pre decrement operator in C

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int x, y, z;
    printf (" Input the value of X: ");
    scanf ("%d", &x);
    printf (" \n Input the value of Y: ");
    scanf ("%d", &y);
    printf ("\n Input the value of Z: ");
    scanf ("%d", &z);
    --x;
    --y;
    --z;
    printf (" \n The updated value of the X: %d ", x);
    printf (" \n The updated value of the Y: %d ", y);
    printf (" \n The updated value of the Z: %d ", z);
```

```
getch();  
}
```

Output

```
Input the value of X: 5  
Input the value of Y: 6  
Input the value of Z: 7  
  
The updated value of the X: 6  
The updated value of the Y: 7  
The updated value of the Z: 8
```

Post decrement Operator:

Post decrement operator is used to decrease the original value of the operand by 1 after assigning to the expression.

Syntax

1. `B = A--;`

In the above syntax, the value of operand 'A' is assigned to the variable 'B', and then the value of A is decreased by 1.

Example 4: Program to use the post decrement operator in C

```
#include <stdio.h>  
#include <conio.h>  
void main ()  
{  
    int x, y, z, a, b, c;  
    printf (" Input the value of X: ");  
    scanf ("%d", &x);  
    printf (" Input the value of Y: ");  
    scanf ("%d", &y);  
    printf (" Input the value of Z: ");  
    scanf ("%d", &z);  
    a = x--;
```

```

b = y--;
c = z--;
printf (" \n The original value of a: %d", a);
printf (" \n The original value of b: %d", b);
printf (" \n The original value of c: %d", c);
printf (" \n\n The updated value of the X: %d ", x);
printf (" \n The updated value of the Y: %d ", y);
printf (" \n The updated value of the Z: %d ", z);
getch();
}

```

Output

```

Input the value of X: 6
Input the value of Y: 12
Input the value of Z: 18

The original value of a: 6
The original value of b: 12
The original value of c: 18

The updated value of the X: 5
The updated value of the Y: 11
The updated value of the Z: 17

```

Bitwise Operator in C

The bitwise operators are the operators used to perform the operations on the data at the bit-level. When we perform the bitwise operations, then it is also known as bit-level programming. It consists of two digits, either 0 or 1. It is mainly used in numerical computations to make the calculations faster.

We have different types of bitwise operators in the C programming language. The following is the list of the bitwise operators:

Operator	Meaning of operator
&	Bitwise AND operator

	Bitwise OR operator
^	Bitwise exclusive OR (XOR) operator
~	One's complement operator or Bitwise Not
<<	Left shift operator
>>	Right shift operator

Bitwise AND operator

Bitwise AND operator is denoted by the single ampersand sign (&). Two integer operands are written on both sides of the (&) operator. If the corresponding bits of both the operands are 1, then the output of the bitwise AND operation is 1; otherwise, the output would be 0.

For example,

We have two variables a and b.

a =6;

b=4;

The binary representation of the above two variables are given below:

a = 00000000 00000110

b = 00000000 00000100

When we apply the bitwise AND operation in the above two variables, i.e., a&b, the output would be:

Result = 0100

As we can observe from the above result that bits of both the variables are compared one by one. If the bit of both the variables is 1 then the output would be 1, otherwise 0.

Program:

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

```
#include <conio.h>
```

```
void main()
```

```
{
```

```
    int a=6, b=14,c;
```

```

clrscr();
c=a&b;
printf("The output of the Bitwise AND operator is %d",c);
getch();
}

```

Output:

The output of the Bitwise AND operator is 6

Bitwise OR operator

The bitwise OR operator is represented by a single vertical sign (`|`). Two integer operands are written on both sides of the (`|`) symbol. If the bit value of any of the operand is 1, then the output would be 1, otherwise 0.

For example,

We consider two variables,

a = 23;

b = 10;

The binary representation of the above two variables would be:

a = 00000000 00010111

b = 00000000 00001010

When we apply the bitwise OR operator in the above two variables, i.e., `a|b`, then the output would be:

Result = 00000000 00011111

Program:

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

```
int main()
```

```
{
```

```
    int a=23,b=10, c;
```

```
    clrscr();
```

```
    c=a|b;
```

```
printf("The output of the Bitwise OR operator is %d",c);  
getch();  
}
```

Output:

The output of the Bitwise OR operator is 31

Bitwise exclusive OR (XOR) operator

Bitwise exclusive OR operator is denoted by (^) symbol. Two operands are written on both sides of the exclusive OR operator. If the corresponding bit of any of the operand is 1 then the output would be 1, otherwise 0.

For example,

We consider two variables a and b,

a = 12;

b = 10;

The binary representation of the above two variables would be:

a = 00000000 00001100

b = 00000000 00001010

When we apply the bitwise exclusive OR operator in the above two variables (a^b), then the result would be:

Result = 00000000 00001110

Program

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

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

```
void main()
```

```
{
```

```
    int a=12,b=10, c;
```

```
    clrscr();
```

```

c=a^b;
printf("The output of the Bitwise XOR operator is %d",c);
getch();
}

```

Output

The output of the Bitwise XOR operator is 31

Bitwise complement operator

The bitwise complement operator is also known as one's complement operator. It is represented by the symbol tilde (~). It takes only one operand or variable and performs complement operation on an operand. When we apply the complement operation on any bits, then 0 becomes 1 and 1 becomes 0.

For example,

If we have a variable named 'a',

```
a = 8;
```

The binary representation of the above variable is given below:

```
a = 1000
```

When we apply the bitwise complement operator to the operand, then the output would be:

```
Result = 0111
```

Program

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int a=8,b;
    clrscr();
    b=~a;
    printf("The output of the Bitwise NOT operator is %d",b);
}

```



```
getch();  
}
```

Output

The output of the Bitwise NOT operator is -9

Bitwise shift operators

Two types of bitwise shift operators exist in C programming. The bitwise shift operators will shift the bits either on the left-side or right-side. Therefore, we can say that the bitwise shift operator is divided into two categories:

- Left-shift operator
- Right-shift operator

Left-shift operator

It is an operator that shifts the number of bits to the left-side.

Syntax of the left-shift operator is given below:

1. Operand << n

Where,

Operand is an integer expression on which we apply the left-shift operation.

n is the number of bits to be shifted.

For example,

Suppose we have a statement:

```
int a = 5;
```

The binary representation of 'a' is given below:

```
a = 00000000 00000101
```

If we want to leftshift the above representation by 2, then the statement would be:

```
a << 2;
```

```
00000000 00000101<<2 = 00000000 00010100
```

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

```
#include <conio.h>
void main()
{
    int a=5,b; // variable initialization
    clrscr();
    b=a<<2;
    printf("The value of a<<2 is : %d ", b);
    getch();
}
```

Output

The value of a<<2 is : 20

Right-shift operator

It is an operator that shifts the number of bits to the right side.

Syntax of the right-shift operator is given below:

Operand >> n;

Where,

Operand is an integer expression on which we apply the right-shift operation.

N is the number of bits to be shifted.

For example,

Suppose we have a statement,

int a = 7;

The binary representation of the above variable would be:

a = 00000000 00000111

If we want to rightshift the above representation by 2, then the statement would be:

a>>2;

00000000 00000111 >> 2 = 00000000 00000001

Program

```
#include <stdio.h>
int main()
{
    int a=7,b;
    clrscr();
    b=a>>2;
    printf("The value of a>>2 is : %d ",b);
    getch();
}
```

Output

The value of a>>2 is : 1

What is a Flowchart?

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as “flowcharting”.

Rules or guidelines of Flow chart:

The various Rules or Guidelines for drawing the flowchart are given below–

- Only conventional flowchart symbols should be used.
- Proper use of names and variables in the flowchart.
- If the flowchart becomes large and complex, use connector symbols.
- Flowcharts should have start and stop points.

Flowchart symbols:

The different flowchart symbols have different conventional meanings.

The various symbols used in Flowchart Designs are given below.

- **Terminal Symbol:** In the flowchart, it is represented with the help of a circle for denoting the start and stop symbol. The symbol given below is used to represent the terminal symbol.



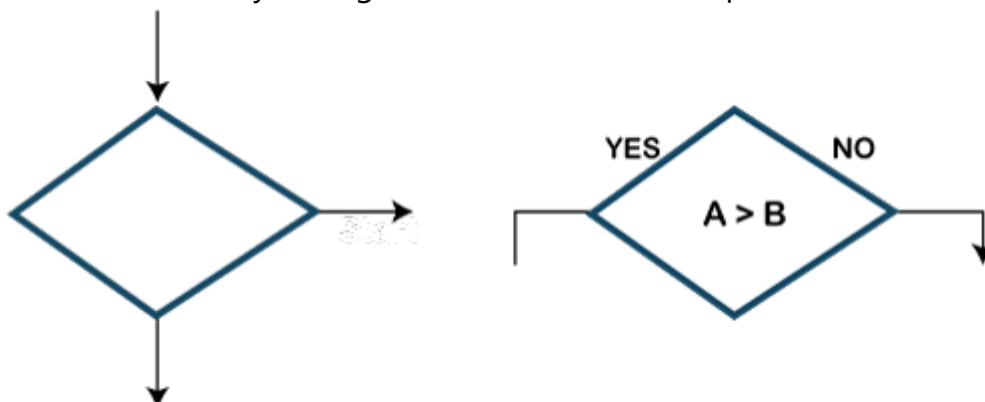
- **Input/output Symbol:** The input symbol is used to represent the input data, and the output symbol is used to display the output operation. The symbol given below is used for representing the Input/output symbol.



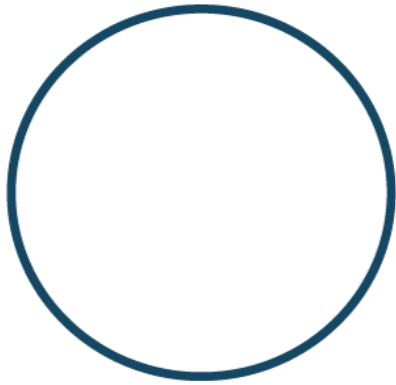
- **Processing Symbol:** It is represented in a flowchart with the help of a rectangle box used to represent the arithmetic and data movement instructions. The symbol given below is used to represent the processing symbol.



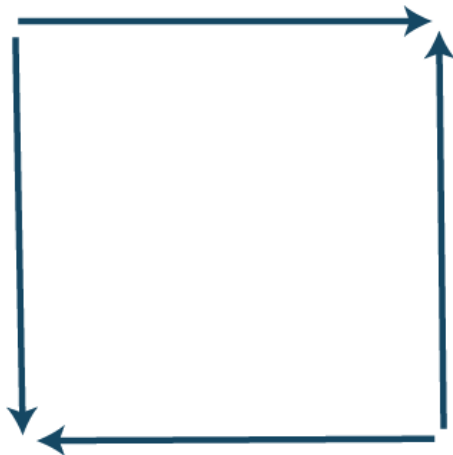
- **Decision Symbol:** Diamond symbol is used for represents decision-making statements. The symbol given below is used to represent the decision symbol.



- **Connector Symbol:** The connector symbol is used if flows discontinued at some point and continued again at another place. The following symbol is the representation of the connector symbol.



- **Flow lines:** It represents the exact sequence in which instructions are executed. Arrows are used to represent the flow lines in a flowchart. The symbol given below is used for representing the flow lines:



Advantages of Flowchart in C:

Following are the various advantages of flowchart:

- **Communication:** A flowchart is a better way of communicating the logic of a program.
- **Synthesis:** Flowchart is used as working models in designing new programs and software systems.
- **Efficient Coding:** Flowcharts act as a guide for a programmer in writing the actual code in a high-level language.

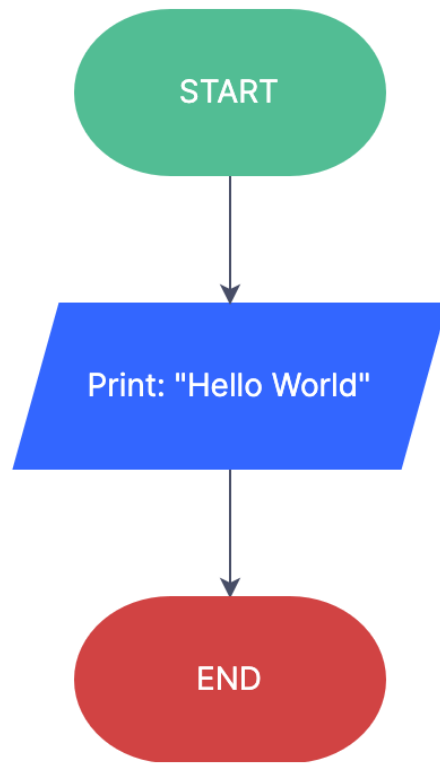
- **Proper Debugging:** Flowcharts help in the debugging process.
- **Effective Analysis:** Effective analysis of logical programs can be easily done with the help of a related flowchart.
- **Proper Documentation:** Flowchart provides better and proper documentation. It consists of various activities such as collecting, organizing, storing, and maintaining all related program records.
- **Testing:** A flowchart helps in the testing process.
- **Efficient program maintenance:** The maintenance of the program becomes easy with the help of a flowchart.

Disadvantages of Flowchart in C:

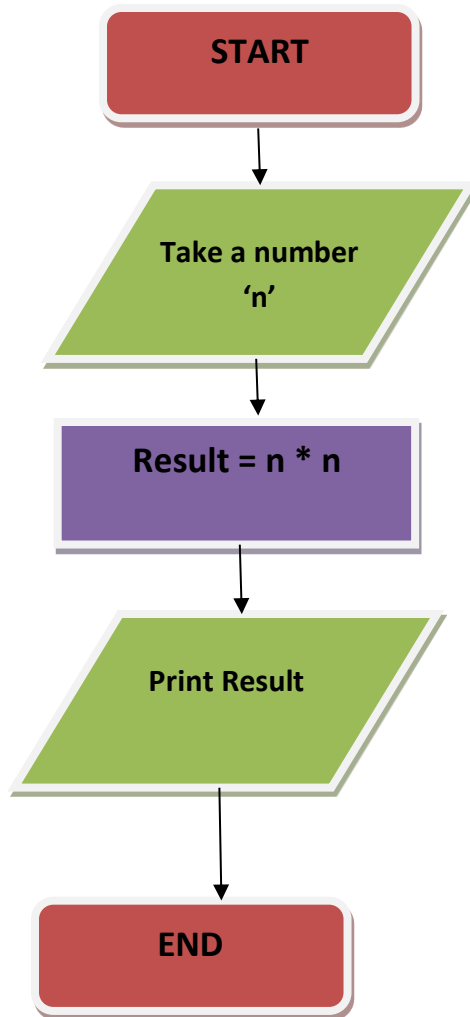
Following are the various disadvantages of flowchart:

- **Time-consuming:** Designing a flowchart is a very time-consuming process.
- **Complex:** It isn't easy to draw a flowchart for large and complex programs.
- **There is no standard** in the flowchart; there is no standard to determine the quantity of detail.
- **Difficult to modify:** It is very difficult to modify the existing flowchart.

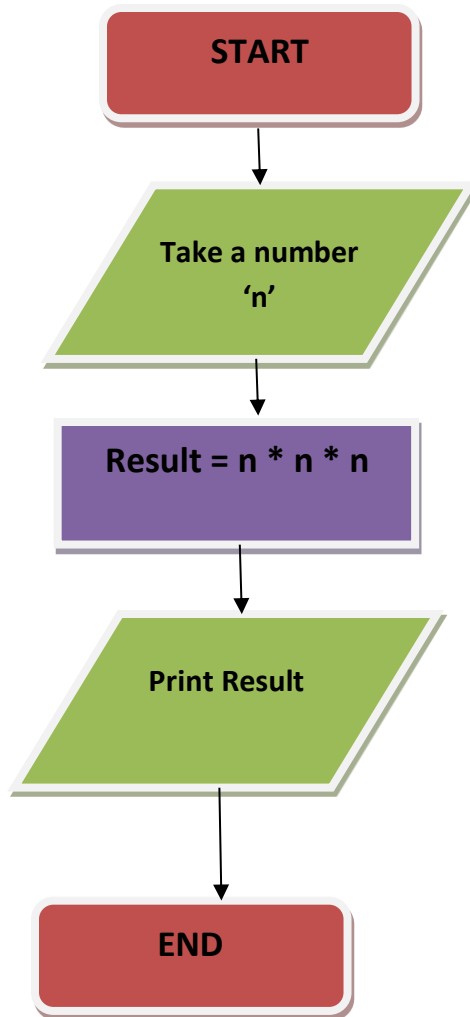
1. Draw a flowchart to print "Hello World" Message



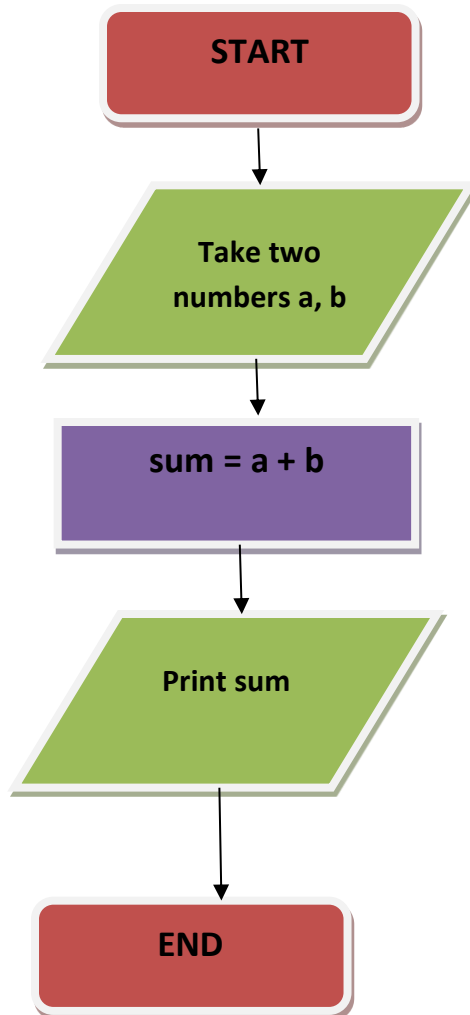
2. Draw a flowchart to print the square of a given number



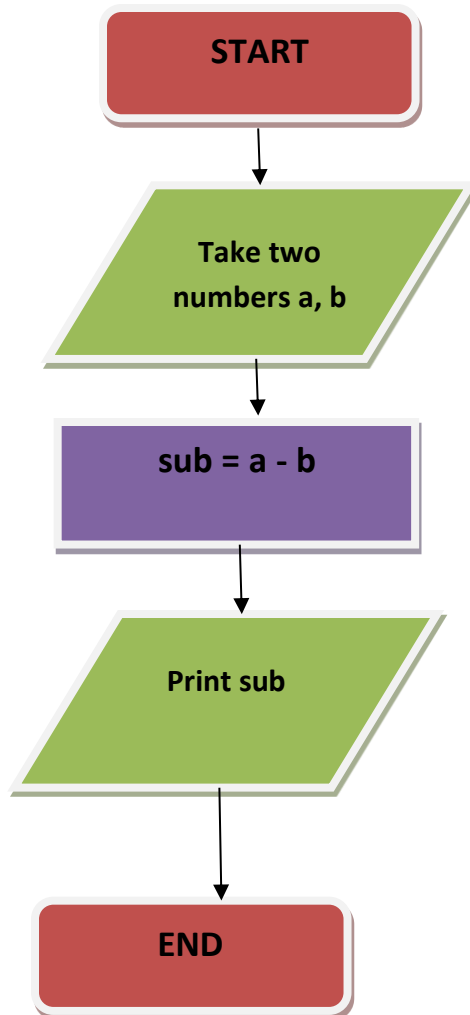
3. Draw a flowchart to print the cube of a given number



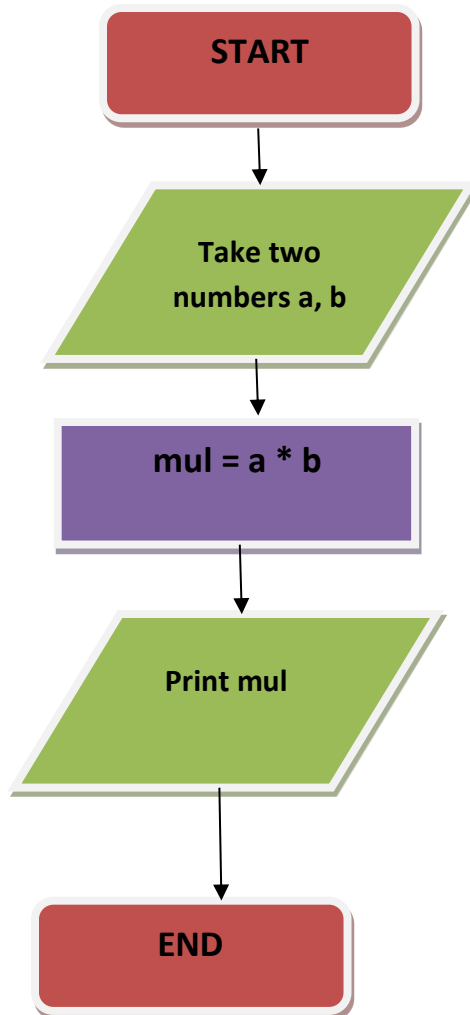
4. Draw a flowchart to print the addition of two numbers



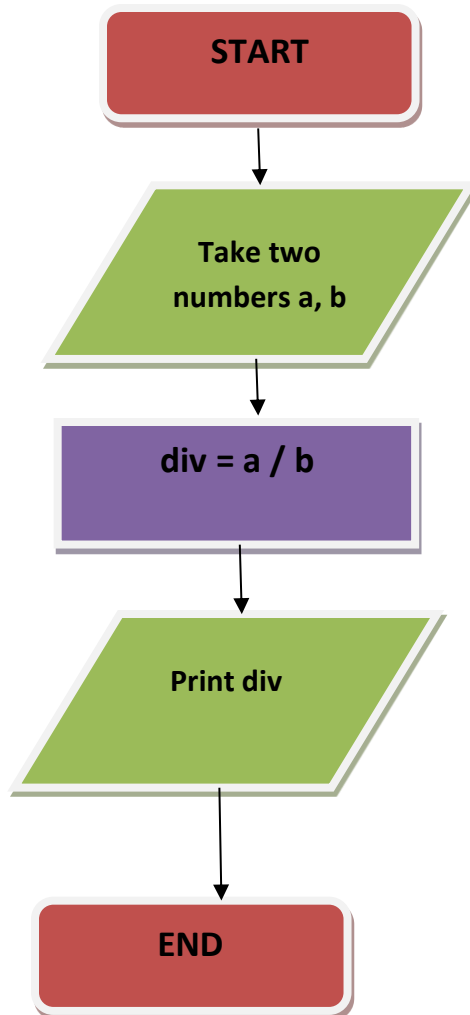
5. Draw a flowchart to print the subtraction of two numbers



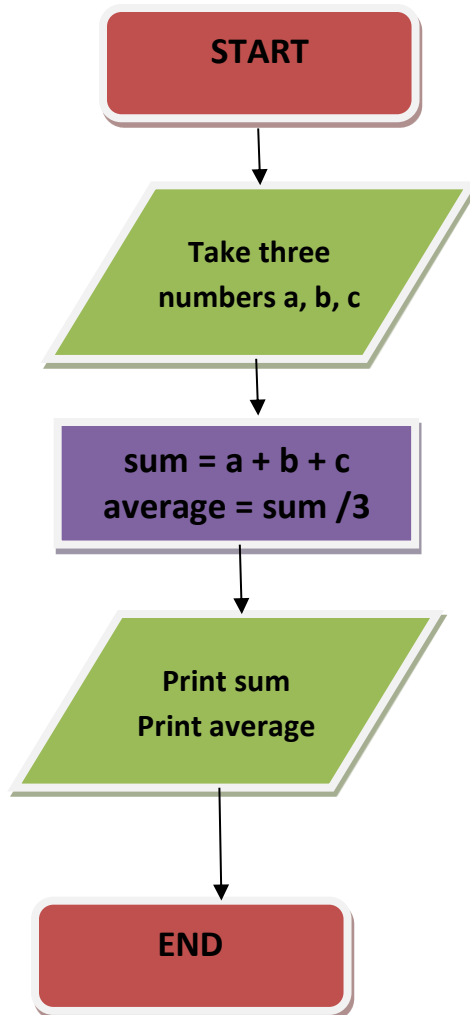
6. Draw a flowchart to print the multiplication of two numbers



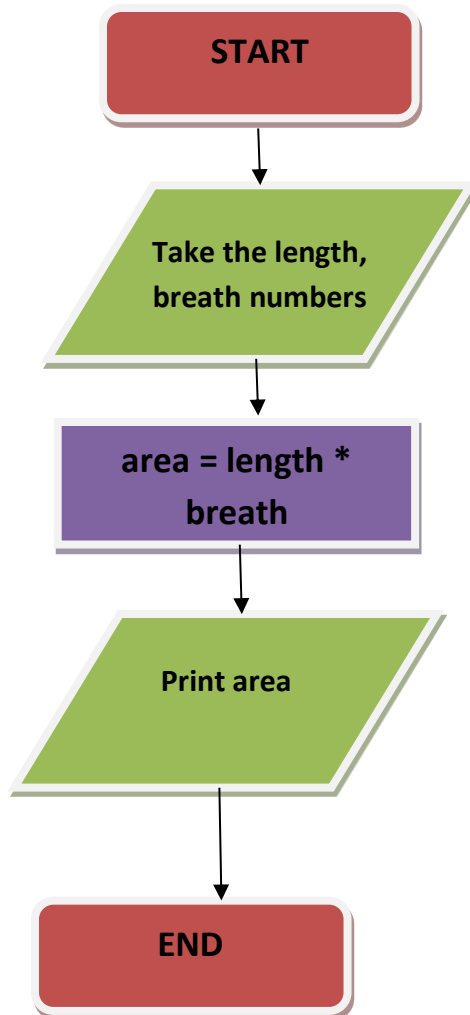
7. Draw a flowchart to print the division of two numbers



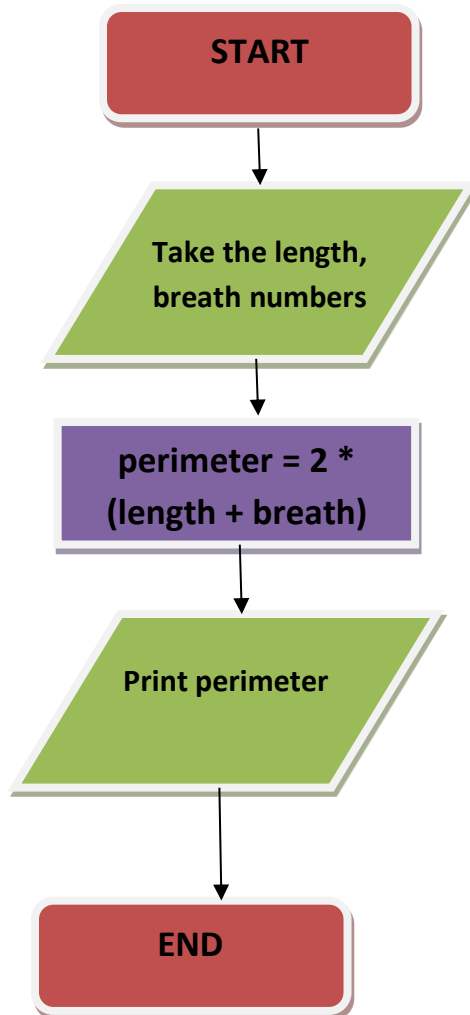
8. Draw a flowchart to print the average of three numbers



9. Draw a flowchart to calculate the area of rectangle



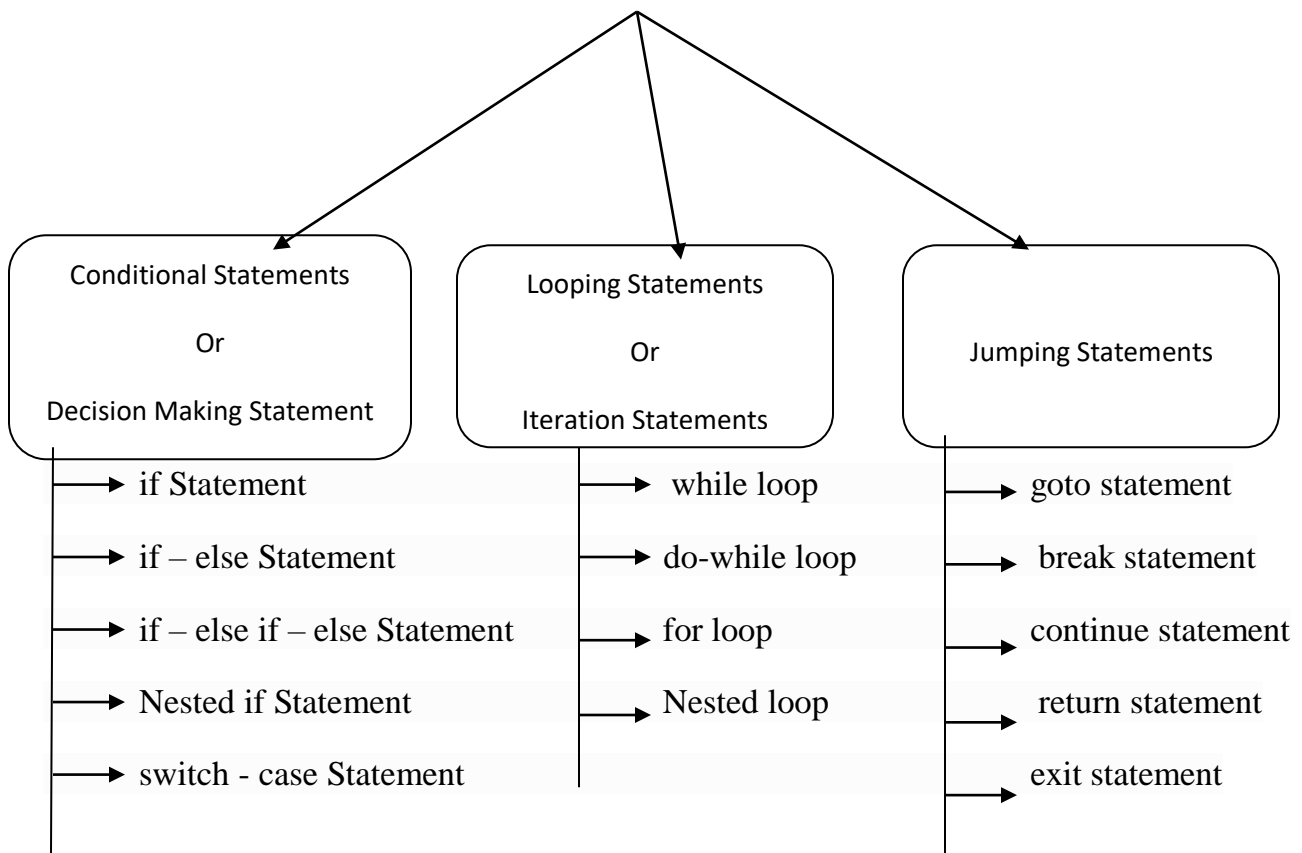
10. Draw a flowchart to calculate the perimeter of rectangle



Unit – 3

Control Statements

Every statement in a computer is executed based on pre-defined rules. The control flow is also based on logic. You find a necessity to execute a few customized logics. The control enters the statements block and gets executed if the logic is satisfied. Hence, they are called control statements. In simple words, Control statements in [C](#) help the computer execute a certain logical statement and decide whether to enable the control of the flow through a certain set of statements or not.



Conditional Statements

Conditional statements help you to make a decision based on certain conditions. These conditions are specified by a set of conditional statements having Boolean expressions which are evaluated to a Boolean value true or false. There are following types of conditional statements in C.

If Statement

The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions. The syntax of the if statement is given below.

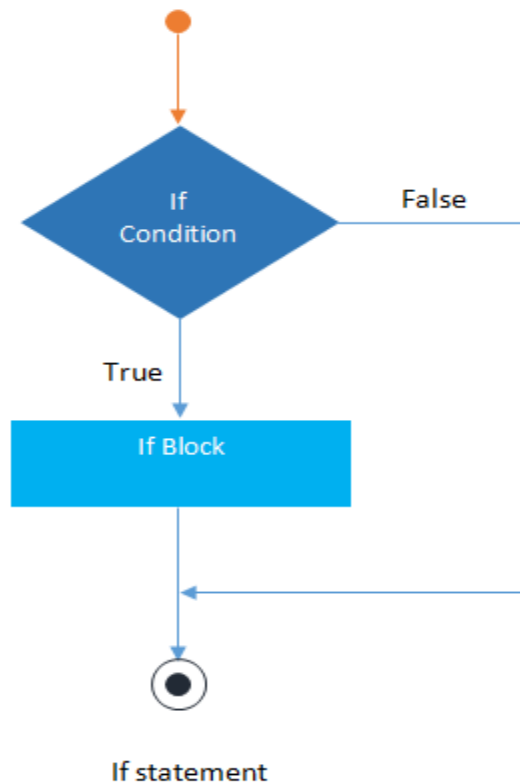
Syntax 1:

```
if(condition)
Statement;
```

Syntax 2:

```
if (condition)
{
Statement 1;
}
Statement x;
```

Flow chart of if statement:



Example 1:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character :");
    scanf("%c", &ch);
    if(ch=='a')
        printf("Amit Kumar");
    getch();
}
```

Output:

Run 1:

Enter a character : a

Amit Kumar

Run 2:

Enter a character : r

Example 2:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a number :");
    scanf("%d", &n);
    if(n>100)
        printf("Number is greater than 100");
    getch();
}
```

Output:

Run 1:

Enter a number : 140

Number is greater than 100

Run 2:

Enter a number : 40

Example 3:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a number :");
    scanf("%d", &n);
    if(n<100)
        printf("Number is smaller than 100");
}
```

```
        getch();  
    }
```

Output:

Run 1:

Enter a number : 50

Number is smaller than 100

Run 2:

Enter a number : 150

Example 4:

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
    int n1, n2;  
    clrscr();  
    printf("Enter two numbers :");  
    scanf("%d%d", &n1, &n2);  
    if(n1<n2)  
        printf("%d is less than %d", n1, n2);  
    getch();  
}
```

Output:

Run 1:

Enter two numbers : 140 150

140 is less than 150

Run 2:

Enter two numbers : 240 50

Example 5:

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{
```

```
int n1, n2;  
clrscr();  
printf("Enter two numbers :");  
scanf("%d%d", &n1, &n2);  
if(n1>n2)  
printf("%d is greater than %d", n1, n2);  
getch();  
}
```

Output:

Run 1:

Enter two numbers : 160 150

160 is greater than 150

Run 2:

Enter two numbers : 20 50

If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition.

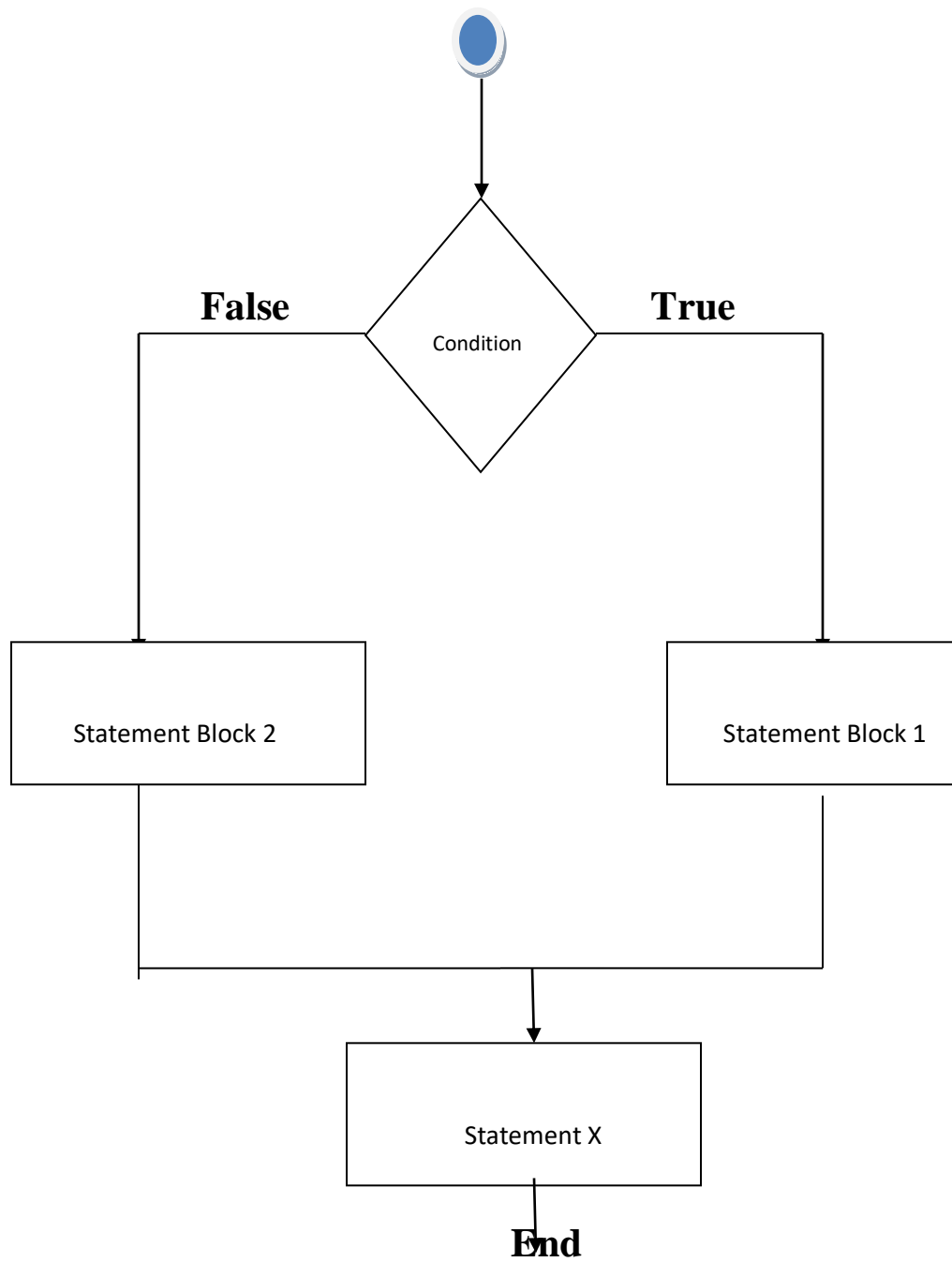
Syntax 1

```
if(condition)
    statement 1;
else
    statement 2;
statement x;
```

Syntax 2

```
if(condition)
{
----- } Statement Block 1
----- }
----- }
}
else
{
----- } Statement Block 2
----- }
----- }
}
Statement x;
```


Flow chart of if – else statement



Example 1:

WAP to check the given number is greater or smaller than 100.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a number =");
    scanf("%d", &n);
    if(n>100)
        printf("Number is greater than 100");
    else
        printf("Number is smaller than 100");
    getch();
}
```

Output:

Run 1:

Enter a number = 120

Number is greater than 100

Run 2:

Enter a number = 20

Number is smaller than 100

Example 2:

WAP to check the greater number between has given two numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n1, n2;
    clrscr();
    printf("Enter two numbers =");
    scanf("%d%d", &n1, &n2);
    if(n1>n2)
        printf("n1 is greater");
    else
        printf("n2 is greater");
    getch();
}
```

Output:

Run 1:

Enter two numbers = 120 220

n2 is greater

Run 2:

Enter two numbers = 220 120

n1 is greater

Example 3:

WAP to check whether given number is Even or Odd

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a number =");
    scanf("%d", &n);
    if(n%2==0)
        printf("Number is Even");
    else
        printf("Number is Odd");
    getch();
}
```

Output:

Run 1:

Enter a number = 20

Number is Even

Run 2:

Enter a number = 21

Number is Odd

Example 4:

WAP to check whether given year is Leap or not leap

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a year =");
    scanf("%d", &n);
    if(n%4==0)
        printf("Leap Year");
    else
        printf("Not a Leap Year");
    getch();
}
```

Output:

Run 1:

Enter a year = 2020

Leap Year

Run 2:

Enter a year = 2021

Not a Leap Year

Example 5:

WAP to check whether given number is +ve or -ve.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a number =");
    scanf("%d", &n);
    if(n>0)
        printf("Positive Number");
    else
        printf("Negative Number");
    getch();
}
```

Output:

Run 1:

Enter a number = 20

Positive Number

Run 2:

Enter a year = -21

Negative Number

Example-6:

C program to check whether a character is an alphabet or not

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character =");
    scanf("%c", &ch);
    if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z') )
        printf("It is an alphabet");
    else
        printf("It is not an alphabet");
    getch();
}
```

Output:

Run 1:

Enter a character = R

It is an alphabet

Run 2:

Enter a year = 6

It is not an alphabet

Example-7:

Program to check whether a person is eligible to vote or not.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int age;
    clrscr();
    printf("Enter your age =");
    scanf("%d",&age);
    if(age>=18)
    {
        printf("You are eligible to vote");
    }
    else
    {
        printf("Sorry! you can't vote");
    }
    getch();
}
```

Output:

Run 1:

Enter your age = 20

You are eligible to vote

Run 2:

Enter your age = 17

Sorry! Yor cant's vote

if – else if – else Statement

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible.

Syntax 1:

```
if(condition 1)
    Statement 1;
else if(condition 2)
    Statement 2;
else if(condition 3)
    Statement 3;
.
.
.
else if(condition n)
    Statement n;
else
    Statement s;
Statement X;
```

Syntax 2:

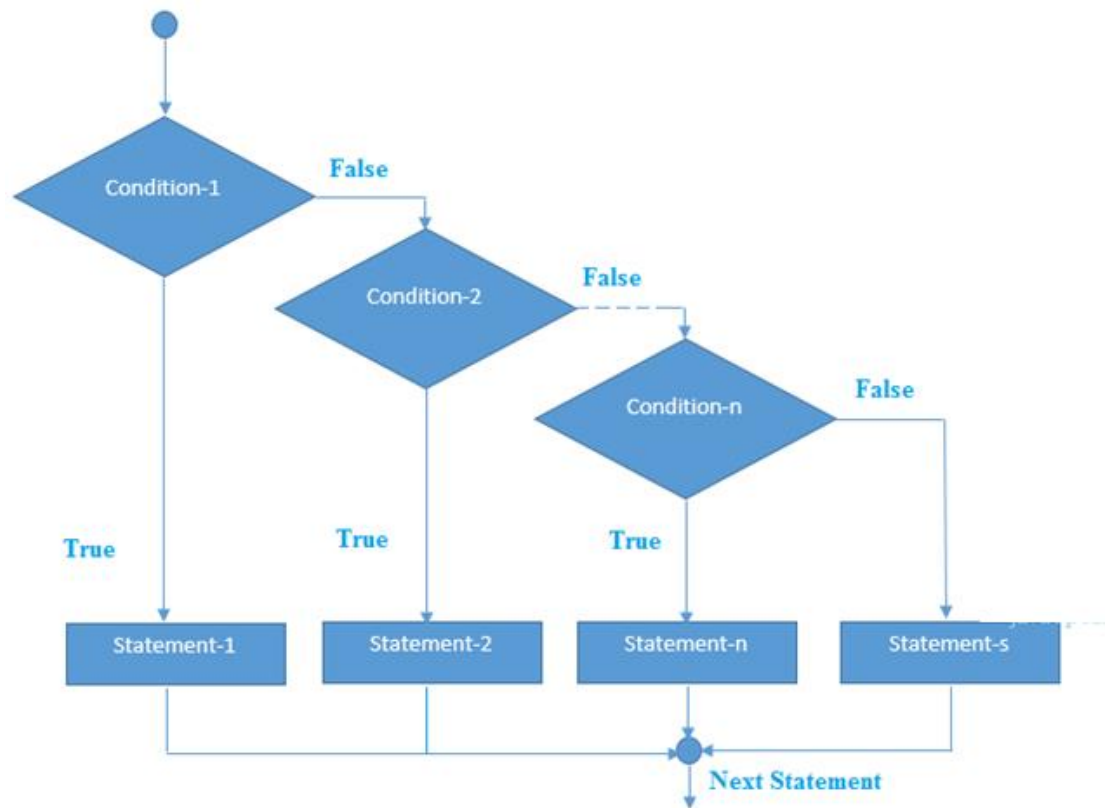
```
if(condition 1)
{
-----
-----
}
else if(condition 2)
{
-----
-----
}
else if(condition 3)
{
```

```

-----
-----
}
.
.
.
else if(condition n)
{
-----
-----
}
else
{
-----
-----
}
Statement X;

```

Flow Chart of if – else if – else statement



Example 1:**WAP to check given number is +ve, -ve or zero.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a number =");
    scanf("%d", &n);
    if(n>0)
        printf("Positive Number");
    else if(n<0)
        printf("Negative Number");
    else
        printf("Number is zero");
    getch();
}
```

Output:**Run 1:**

Enter a number: 10

Positive Number

Run 2:

Enter a number: -10

Negative Number

Run 3:

Enter a number: 0

Number is zero

Example 2:

WAP to find out largest number among three numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n1, n2, n3;
    clrscr();
    printf("Enter any three number =");
    scanf("%d%d%d", &n1, &n2, &n3);
    if(n1>n2 && n1>n3)
        printf("%d is greater", n1);
    else if(n2>n1 && n2>n3)
        printf("%d is greater", n2);
    else
        printf("%d is greater", n3);
    getch();
}
```

Output:

Run 1:

Enter any three number = 30 20 25

30 is greater

Run 2:

Enter any three number = 30 40 25

40 is greater

Run 3:

Enter any three number = 30 20 45

45 is greater

Example 3:

WAP to check given number is one digit, two digit, three digit and four digit number

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    clrscr();
    printf("Enter a number =");
    scanf("%d", &n);
    if(n>=0 && n<=9)
        printf("One digit number");
    else if(n>=10 && n<=99)
        printf("Two digit number");
    else if(n>=100 && n<=999)
        printf("Three digit number");
    else if(n>=1000 && n<=9999)
        printf("Four digit number");
    else
        printf("Invalid Number");
    getch();
}
```

Output:

Run 1:

Enter a number = 6

One digit number

Run 2:

Enter a number = 16

Two digit number

Run 3:

Enter a number = 612

Three digit number

Run 4:

Enter a number = 6123

Four digit number

Run 5:

Enter a number = 12612

Invalid Number

Example 4:

WAP to print week day name according to the given weekday number

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int day;
    clrscr();
    printf("Enter the weekday number =");
    scanf("%d", &day);
    if(day==1)
        printf("Sunday");
    else if(day==2)
        printf("Monday");
    else if(day==3)
        printf("Tuesday");
    else if(day==4)
        printf("Wednesday");
    else if(day==5)
```

```
    printf("Thursday");
    else if(day==6)
    printf("Friday");
    else if(day==7)
    printf("Saturday");
    else
    printf("Invalid Week day Number");
    getch();
}
```

Output:

Run 1:

Enter the weekday number : 1

Sunday

Run 2:

Enter the weekday number : 2

Monday

Run 3:

Enter the weekday number : 3

Tuesday

Run 4:

Enter the weekday number : 4

Wednesday

Run 5:

Enter the weekday number : 5

Thursday

Run 6:

Enter the weekday number : 6

Friday

Run 7:

Enter the weekday number : 7

Saturday

Run 8:

Enter the weekday number : 8

Invalid Weekday number

Example 4:

WAP to print month name according to the given month number

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int month;
    clrscr();
    printf("Enter the month number =");
    scanf("%d", &month);
    if(month==1)
        printf("January");
    else if(month==2)
        printf("February");
    else if(month==3)
        printf("March");
    else if(month==4)
        printf("April");
    else if(month==5)
```



```
    printf("May");
    else if(month==6)
    printf("June");
    else if(month==7)
    printf("July");
    else if(month==8)
    printf("August");
    else if(month==9)
    printf("September");
    else if(month==10)
    printf("October");
    else if(month==11)
    printf("November");
    else if(month==12)
    printf("December");
    else
    printf("Invalid month Number");
    getch();
}
```

Output:

Run 1:

Enter the month number: 1

January

Run 2:

Enter the month number: 2

February

Run 3:

Enter the month number: 3

March

Run 4:

Enter the month number: 4

April

Run 5:

Enter the month number: 5

May

Run 6:

Enter the month number: 6

June

Run 7:

Enter the month number: 7

July

Run 8:

Enter the month number: 8

August

Run 9:

Enter the month number: 9

September

Run 10:

Enter the month number: 10

October

Run 11:

Enter the month number: 11

November

Run 12:

Enter the month number: 12

December

Run 13:

Enter the month number: 13

Invalid month number

Exercise for students:

1. WAP to find out smallest number among three numbers.
2. WAP to check given character is vowel or consonant.
3. WAP to check given alphabet is lower case, upper case, digit or special alphabet.

C Switch Statement

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

```
switch(expression)
{
case value1:
    //code to be executed;
    break;
case value2:
    //code to be executed;
    break;
.....
.
.
.
case value n:
    //code to be executed;
    break;
default:
    code to be executed if all cases are not matched;
}
```

Rules for switch statement in C language

- 1) The *switch expression* must be of an integer or character type.
- 2) The *case value* must be an integer or character constant.
- 3) The *case value* can be used only inside the switch statement.
- 4) The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case.

We are assuming that there are following variables.

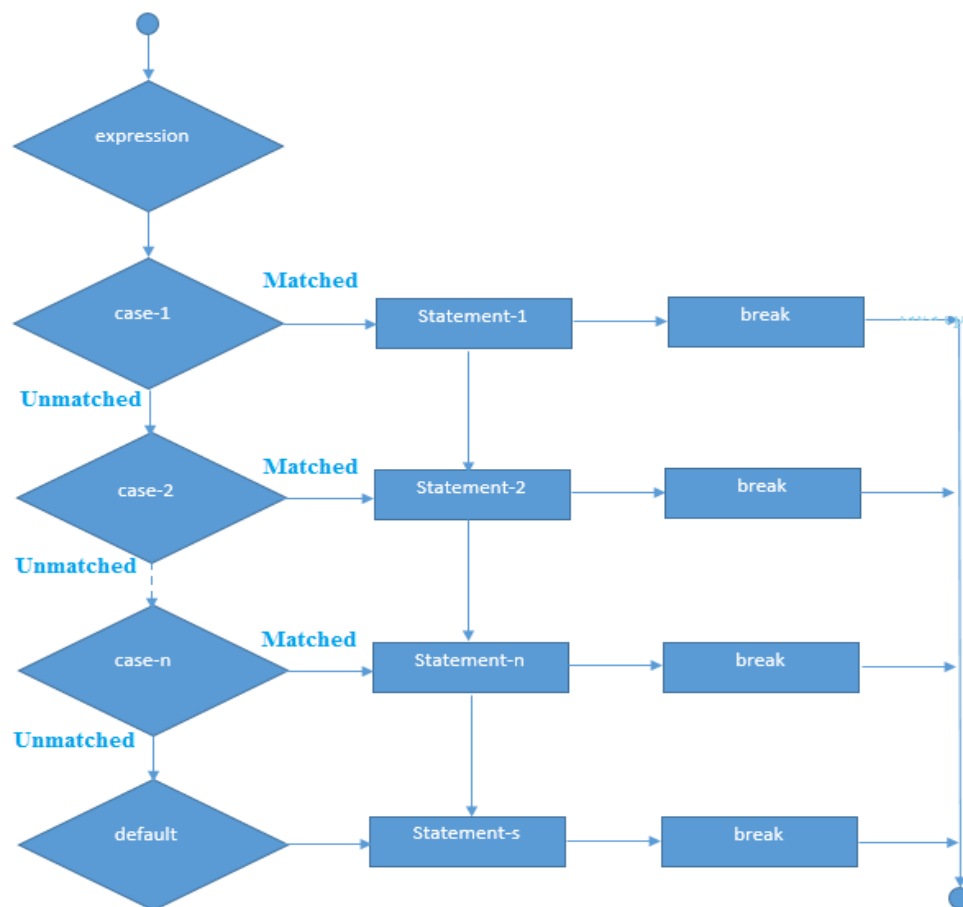
int x,y,z;

char a,b;

float f;

Valid Switch	Invalid Switch	Valid Case	Invalid Case
switch(x)	switch(f)	case 3;	case 2.5;
switch(x>y)	switch(x+2.5)	case 'a';	case x;
switch(a+b-2)		case 1+2;	case x+2;
switch(func(x,y))		case 'x'>'y';	case 1,2,3;

Flow Chart of Switch Case:



Example 1:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int number;
printf("Enter a number:");
scanf("%d", &number);
switch(number)
{
case 10:
printf("number is equals to 10");
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
getch();
}
```

Output

Run 1:

```
Enter a number:4
number is not equal to 10, 50 or 100
```

Run 2:

```
Enter a number:50
number is equal to 50
```

Example 2:

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

void main()
{
    int day;
    clrscr();
    printf("Enter the week day number =");
    scanf("%d", &day);
    switch(number)
    {
        case 1:
            printf("Sunday");
            break;
        case 2:
            printf("Monday");
            break;
        case 3:
            printf("Tuesday");
            break;
        case 4:
            printf("Wednesday");
            break;
        case 5:
            printf("Thursday");
            break;
        case 6:
            printf("Friday");
            break;
        case 7:
            printf("Saturday");
            break;
        default:
            printf("Invalid Weekday number");
```

```
}  
getch();  
}
```

Example 3:

```
#include<stdio.h>  
#include<conio.h>  
  
void main()  
{  
    char ch;  
    clrscr();  
    printf("Enter a character =");  
    scanf("%c", &ch);  
    switch(ch)  
    {  
        case 'a':  
        case 'A':  
            printf("Vowel");  
            break;  
        case 'e':  
        case 'E':  
            printf("Vowel");  
            break;  
        case 'i':  
        case 'I':  
            printf("Vowel");  
            break;  
        case 'o':  
        case 'O':  
            printf("Vowel");  
            break;  
        case 'u':  
        case 'U':  
            printf("Vowel");
```


break;

default:

```
printf("Consonants");  
}  
getch();  
}
```

Example 4:

```
#include<stdio.h>  
#include<conio.h>  
  
void main()  
{  
    char op;  
    clrscr();  
    printf("Enter an arithmetic operator (+, -, *, /, %) =");  
    scanf("%c", &op);  
    printf("Enter two numbers =");  
    scanf("%d%d", &a, &b);  
    switch(op)  
    {  
        case '+':  
            c=a+b;  
            printf("\n The sum is = %d", c);  
            break;  
        case '-':  
            c=a-b;  
            printf("\n The sub is = %d", c);  
            break;  
        case '*':  
            c=a*b;  
            printf("\n The mul is = %d", c);  
            break;  
        case '/':  
            c=a/b;
```

```

printf("\n The div is = %d", c);
break;
case '%':
c=a%b;
printf("\n The mod is = %d", c);
break;

default:
printf("Invalid Operator");
}
getch();
}

```

Nested Loops in C

C supports nesting of loops in C. **Nesting of loops** is the feature in C that allows the looping of statements inside another loop. Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define '**while**' loop inside a '**for**' loop.

Syntax of Nested loop

```

Outer_loop
{
    Inner_loop
    {
        // inner loop statements.
    }
    // outer loop statements.
}

```

Outer_loop and **Inner_loop** are the valid loops that can be a 'for' loop, 'while' loop or 'do-while' loop.

Syntax 1

The syntax for a **nested for loop** statement in C is as follows –

```
for ( initialization; condition; increment/decrement )
```

```

{
    for ( initialization; condition; increment/decrement )
    {
        //statements of inner loop;
    }
    // statements of outer loop;
}

```

Syntax 2

The syntax for a **nested while loop** statement in C programming language is as follows –

```

while(condition)
{
    while(condition)
    {
        //statements of inner loop
    }
    //statements of inner loop
}

```

Syntax 3

The syntax for a **nested do...while loop** statement in C programming language is as follows –

```

do
{
    //statements of inner loop
    do
    {
        //statements of inner loop
    }while( condition );
}while( condition );

```

Example of nested while loop

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int i=1, j=2;
    clrscr();
    while(i<=10)
    {

```

```

        while(j<=20)
        {
            printf("\n%d", j);
            J=j+2;
        }
    printf("\n%d", i);
    i++;
}
getch();
}

```

Example of nested do-while loop

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int i=1, j=2;
    clrscr();
    do
    {
        printf("\n%d", i);
        i++;
        do
        {
            printf("\n%d", j);
            j=j+2;
        } while(j<=20);
        printf("\n");
    } while(i<=10);
    getch();
}

```

Example of nested for loop

```

#include <stdio.h>
#include <conio.h>
void main()

```

```

{
    int i, j;
    clrscr();
    for(i=1;i<=5;i++)
    {
        for(j=1;j<=i; j++)
        {
            printf("  %d ", j);
        }
        printf("\n");
        getch();
    }
}

```

Jumping Statement

Jump Statement makes the control jump to another section of the program unconditionally when encountered. It is usually used to terminate the **loop** or **switch-case** instantly. It is also used to escape the execution of a section of the program.

break statement

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

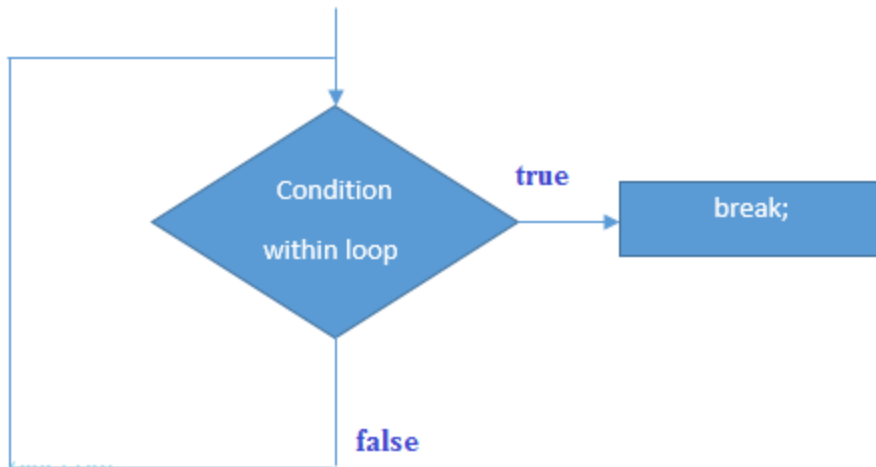


Figure: Flowchart of break statement

Example

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
    int i;
    clrscr();
    for(i = 1; i<=10; i++)
    {
        printf("%d ",i);
        if(i == 5)
            break;
    }
    printf("came outside of loop i = %d",i);
    getch();
}
```

Output

```
1 2 3 4 5 came outside of loop i = 5
```

Example

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

```

#include<conio.h>

void main ()
{
    int n=2,i,choice;
    do
    {
        i=1;
        while(i<=10)
        {
            printf("%d X %d = %d\n",n,i,n*i);
            i++;
        }
        printf("do you want to continue with the table of %d , enter any nonzero value to continue.",n+1);
        scanf("%d",&choice);
        if(choice == 0)
        {
            break;
        }
        n++;
    }while(1);
    getch();
}

```

Output

```

2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
do you want to continue with the table of 3 , enter any non-zero value to continue.3
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18

```

```
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
do you want to continue with the table of 4 , enter any non-zero value to continue.0
```

continue statement

The **continue statement** in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i=1;
    clrscr();
    for(i=1;i<=10;i++)
    {
        if(i==5)
        {
            continue;
        }
        printf("%d \n",i);
    }
    getch();
}
```

Output

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


goto statement

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statement can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement.

Syntax:

```
label:  
//some part of the code;  
goto label;
```

goto example

Let's see a simple example to use goto statement in C language.

```
#include <stdio.h>  
#include <conio.h>  
void main()  
{  
    int num,i=1;  
    clrscr();  
    printf("Enter the number whose table you want to print?");  
    scanf("%d",&num);  
    table:  
    printf("%d x %d = %d\n",num,i,num*i);  
    i++;  
    if(i<=10)  
        goto table;  
    getch();  
}
```

Output:

```
Enter the number whose table you want to print?10  
10 x 1 = 10  
10 x 2 = 20  
10 x 3 = 30  
10 x 4 = 40  
10 x 5 = 50  
10 x 6 = 60
```

10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100

exit() function in C

The **exit()** function is used to terminate a process or function calling immediately in the program. It means any open file or function belonging to the process is closed immediately as the exit() function occurred in the program. The exit() function is the standard library function of the C, which is defined in the **<stdlib.h>** or **<process.h>** header file. So, we can say it is the function that forcefully terminates the current program and transfers the control to the operating system to exit the program. The exit(0) function determines the program terminates without any error message, and then the exit(1) function determines the program forcefully terminates the execution process.

The **exit()** function has no return type.

Example:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main ()
{
    int i, num;
    clrscr();
    printf ( " Enter the last number: ");
    scanf ( " %d", &num);
    for ( i = 1; i<num; i++)
    {
        if ( i == 6 )
        {
            exit(0);
        }
        else
        {
            printf (" \n Number is %d", i);
        }
    }
    getch();
}
```

```
}
```

Output

```
Enter the last number: 10
```

```
Number is 1
```

```
Number is 2
```

```
Number is 3
```

```
Number is 4
```

```
Number is 5
```

Looping Statements

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

- 1) It provides code reusability.
- 2) Using loops, we do not need to write the same code again and again.

Types of C Loops

There are three types of loops in C language that is given below:

1. while loop
2. do while loop
3. for

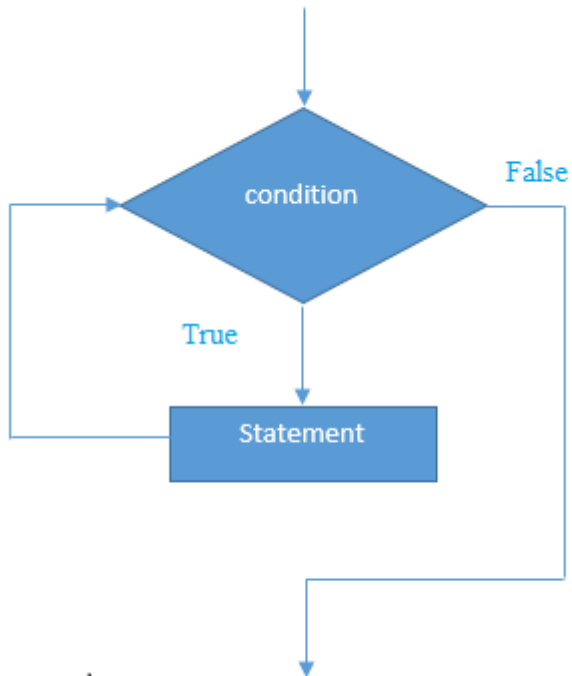
while loop

The while loop in c is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop or entry controlled loop.

Syntax

```
while(condition)
{
    //statements
}
```

Flowchart of while loop



Statement – X

Example 1:

WAP to print the counting from 1 to 10.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n=1;
    clrscr();
    while(n<=10)
    {
        printf("\n%d" ,n);
        n++;
    }
    getch();
}
```

Output

1
2
3
4
5
6
7
8
9
10

Example 2:

WAP to print the counting from 10 to 1.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n=10;
clrscr();
while(n>=1)
{
printf("\n%d" ,n);
n--;
}
getch();
}
```

Output

10
9
8
7
6
5
4
3
2
1

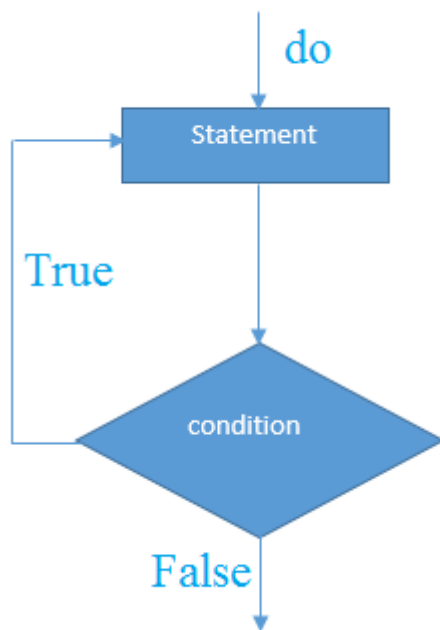
do-while loop in C

The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once. This loop is also called post tested loop or exit-controlled loop.

Syntax

```
do  
{  
//statements  
}while(condition);
```

Flowchart of do-while loop



Statement – X

Example 1:

WAP to print the counting from 1 to 10.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n=1;
clrscr();
do
{
printf("\n%d" ,n);
n++;
} while(n<=10);
getch();
}
```

Output

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

Example 2:

WAP to print the counting from 10 to 1.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n=10;
clrscr();
```



```
do
{
printf("\n%d" ,n);
n--;
} while(n>=1);
    getch();
}
```

Output

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

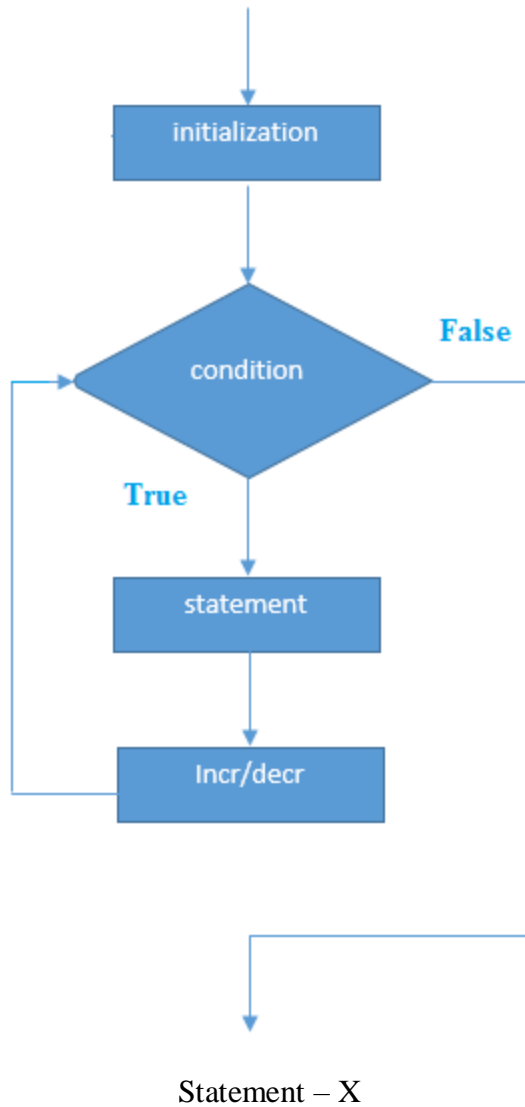
for loop in c

The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. It is better to use for loop if the number of iteration is known in advance.

Syntax

```
for(initialization; condition; incr/decr)
{
//statements
}
```

Flowchart of for loop



Example 1:

WAP to print the counting from 1 to 10.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
```

```
clrscr();
for(n=1; n<=10; n++)
{
printf("\n%d" ,n);
}
getch();
}
```

Output

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

Example 2:

WAP to print the counting from 10 to 1.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
clrscr();
for(n=10; n>=1; n--)
{
printf("\n%d" ,n);
}
getch();
}
```

Output

```
10
9
```

8
7
6
5
4
3
2
1

Problems related to Looping statements:

1. WAP to print the counting from 1 to 10.
2. WAP to print the counting from 10 to 1.
3. WAP to print the counting from 1 to last number.
4. WAP to print the counting from starting to end number.
5. WAP to print the natural number up to given number.
6. WAP to print the sum of 1 to 10 counting.
7. WAP to print the sum of 1 to last number.
8. WAP to print the sum of starting number to end number.
9. WAP to print the table of any number.
10. WAP to print the all even numbers up to has given number.
11. WAP to print the all odd numbers up to has given number.
12. WAP to print the reverse of given number.
13. WAP to count the digits of a given number.
14. WAP to print the sum of all digits of a given number.
15. WAP to print the sum of first and last digit of a given number.
16. WAP to count even and odd digits from a number.
17. WAP to print the sum of even digits and odd digits of a given number.
18. WAP to calculate the factorial of given number.
19. WAP to print the Fibonacci series.
20. WAP to check whether given number is prime or not.
21. WAP to check whether given number is Armstrong or not.
22. WAP to check whether given number is palindrome or not.
23. WAP to check whether given number is perfect or not.
24. WAP to calculate the power of given number.
25. WAP to print the sine series.
26. WAP to print the cosine series.
27. WAP to print the factors of given number.

Program 1: WAP to print “Hello C” Message

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("Hello C");
    getch();
}
```

Output:

Hello C

Program 2: WAP to print Your Name

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("Pramod Kumar");
    getch();
}
```

Output:

Pramod Kumar

Program 3: WAP to print “Hello How Are You” Message

```
#include <stdio.h>
#include <conio.h>
void main()
{
```

```
clrscr();
printf("Hello How Are You");
getch();
}
```

Output:

Hello How Are You

Program 4: WAP to print “C is a Middle Level Language” Message

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("C is a Middle Level Language");
    getch();
}
```

Output:

C is a Middle Level Language

Program 5: WAP to print a Message as-

Hello

How

Are

You

```
#include <stdio.h>
#include <conio.h>
void main()
{
```

```
clrscr();
printf("Hell0");
printf("\n How");
printf("\n Are");
printf("\n You");
getch();
}
```

Output:

Hello

How

Are

You

Program 6: WAP to print a Message as-

Hello How Are You

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
printf("Hell0");
printf("\t How");
printf("\t Are");
printf("\t You");
getch();
}
```

Output:

Hello How Are You

**Program 6: WAP to print a Message as-
Hello**

How

Are

You

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("Hell0");
    printf("\n\t How");
    printf("\n\t\t Are");
    printf("\n\t\t\t You");
    getch();
}
```

Output:

Hello

How

Are

You

**Program 7: WAP to print a Message as-
Hello**

How

Are

You

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("/t/t/tHell0");
    printf("\n\t/tHow");
    printf("\n\tAre");
    printf("\nYou");
    getch();
}
```

Output:

Hello

How

Are

You

Program 8: WAP to print the square of a number

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num=10, res;
    clrscr();
    res=num*num;
    printf("%d",res);
    getch();
}
```

Output:

100

Program 9: WAP to print the square of a number

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num=10, res;
    clrscr();
    res=num*num;
    printf("The Sqaure is =%d",res);
    getch();
}
```

Output:

The Square is =100

Program 10: WAP to print the cube of a number

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num=10, res;
    clrscr();
    res=num*num*num;
    printf("%d",res);
    getch();
}
```

Output:

1000

Program 11: WAP to print the cube of a number

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num=10, res;
    clrscr();
    res=num*num*num;
    printf("The Cube is =%d",res);
    getch();
}
```

Output:

The Cube is =1000

Program 12: WAP to print the addition of two numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1=10,num2=20, res;
    clrscr();
    res=num1+num2;
    printf("%d",res);
    getch();
}
```

Output:

30

Program 13: WAP to print the addition of two integer numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1=10,num2=20, res;
    clrscr();
    res=num1+num2;
    printf("The Sum is =%d",res);
    getch();
}
```

Output:

The Sum is =30

OR

Program 13: WAP to print the addition of two real numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
float num1=10.50, num2=5.25, res;
clrscr();
res=num1+num2;
printf("The Sum is =%f",res);
getch();
}
```

Output:

The Sum is =15.750000

Program 14: WAP to print the subtraction of two numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
int num1=30,num2=20, res;
clrscr();
res=num1 - num2;
printf("The Sub is =%d",res);
getch();
}
```

Output:

The Sub is =10

Program 15: WAP to print the multiplication of two numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1=10,num2=20, res;
    clrscr();
    res=num1 * num2;
    printf("The Mul is =%d",res);
    getch();
}
```

Output:

The Mul is =200

Program 16: WAP to print the division of two numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1=30,num2=10, res;
    clrscr();
    res=num1 / num2;
    printf("The Div is =%d",res);
    getch();
}
```

Output:

The Sub is =3

Program 17: WAP to print the division of two numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1=30,num2=20, res;
    clrscr();
    res=num1 / num2;
    printf("The Div is =%d",res);
    getch();
}
```

Output:

The Sub is =1

Program 18: WAP to print the reminder of given numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1=15,num2=4, res;
    clrscr();
    res=num1 % num2;
    printf("The Reminder is =%d",res);
    getch();
}
```

Output:

The Sub is =3

Program 18: WAP to performs basic arithmetic operations

```
#include <stdio.h>
#include <conio.h>
void main()
{
int num1=15,num2=5, add, sub, mul, div, rem;
clrscr();
add=num1 + num2;
sub=num1 - num2;
mul=num1 * num2;
div=num1 / num2;
rem=num1 % num2;
printf("The Addition is =%d", add);
printf("The Subtraction is =%d", sub);
printf("The Multiplication is =%d", mul);
printf("The Division is =%d", div);
printf("The Reminder is =%d", rem);
getch();
}
```

Output:

The Addition is =20

The Addition is =10

The Addition is =75

The Addition is =3

The Addition is =0

Program 19: WAP to performs basic arithmetic operations

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1=15,num2=5, res;
    clrscr();
    res=num1 + num2;
    printf("The Addition is =%d", res);
    res=num1 - num2;
    printf("The Subtraction is =%d", res);
    res=num1 * num2;
    printf("The Multiplication is =%d", res);

    res=num1 / num2;
    printf("The Division is =%d", res);
    res=num1 % num2;
    printf("The Reminder is =%d", res);
    getch();
}
```

Output:

The Addition is =20

The Addition is =10

The Addition is =75

The Addition is =3

The Addition is =0

Program 20: WAP to performs basic arithmetic operations

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a=15,b=5;
    clrscr();
    printf("The Addition is =%d", a+b);
    printf("The Subtraction is =%d", a-b);
    printf("The Multiplication is =%d", a*b);
    printf("The Division is =%d", a/b);
    printf("The Reminder is =%d", a%b);
    getch();
}
```

Output:

The Addition is =20

The Addition is =10

The Addition is =75

The Addition is =3

The Addition is =0

Program 21: WAP to calculate the area of rectangle

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

```
#include <conio.h>
void main()
{
int length=20, breath=15, area;
clrscr();
area=length*breath;
printf("The area of rectangle is =%d", area);
getch();
}
```

Output:

The area of rectangle is= 300

Program 22: WAP to calculate the area of circle

```
#include <stdio.h>
#include <conio.h>
void main()
{
const float pi=3.14;
float r = 4.50, area;
clrscr();
area = pi * r * r;
printf("The area of circle is =%f", area);
getch();
}
```

Output:

The area of circle is= 14.13

Program 23: WAP to calculate the area of triangle

```
#include <stdio.h>
#include <conio.h>
void main()
{
float b = 20.25, h = 30.58, area;
clrscr();
area = (b * h)/2;
printf("The area of triangle is =%f", area);
getch();
}
```

Output:

The area of circle is= 309.6225

Program 23: WAP to convert the given hours into minutes, seconds

```
#include <stdio.h>
#include <conio.h>
void main()
{
Int h=5, m, s;
clrscr();
m = h * 60;
s = m * 60;
printf("Total hours are =%f", h);
printf("Total minutes are =%f", m);
printf("Total second are =%f", s);
```

```
getch();  
}
```

Output:

The area of circle is= 309.6225

Unit – 5

Program 1: WAP to print the addition of two integers.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1, num2, sum;
    clrscr();
    printf("Enter first number: ");
    scanf("%d", &num1);
    printf("Enter second number: ");
    scanf("%d", &num2);
    sum = num1 + num2;
    printf("Sum of the numbers: %d", sum);
    getch();
}
```

Output:

```
Enter first number: 20
Enter second number: 19
Sum of the numbers: 39
```

Program 2: WAP to print the multiplication of two integers.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1, num2, mul;
    clrscr();
    printf("Enter first number: ");
    scanf("%d", &num1);
    printf("Enter second number: ");
    scanf("%d", &num2);
    mul = num1 * num2;
    printf("Multiplication of the numbers: %d", mul);
    getch();
}
```

Output:

Enter first number: 20

Enter second number: 10

Multiplication of the numbers: 200

Program 3: WAP to determining if a number is +ve or –ve.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num;
    clrscr();
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num > 0)
        printf("%d is a positive number ", num);
    else
        printf("%d is a positive number ", num);
    getch();
}
```

Output:

Run 1:

Enter a number: 10
10 is a positive number

Run 2:

Enter a number: -3
-3 is a negative number

Program 4: WAP to determining whether given number is even or odd.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num;
    clrscr();
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num%2==0)
        printf("%d is an Even number ", num);
    else
        printf("%d is an Odd number ", num);
    getch();
}
```

Output:

Run 1:

Enter a number: 10
10 is an Even number

Run 2:

Enter a number: 13
13 is an Odd number

Program 5: WAP to find out maximum between two numbers.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1, num2;
    clrscr();
    printf("Enter two numbers: ");
    scanf("%d%d", &num1, &num2);
    if (num1>num2)
        printf("%d is maximum ", num1);
    else
        printf("%d is maximum ", num2);
    getch();
}
```

Output:

Run 1:

Enter two number: 100 80
100 is maximum

Run 2:

Enter a number: 80 90
90 is maximum

Program 6: WAP to find out maximum number among three numbers.

Solution 1:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1, num2, num3;
    clrscr();
    printf("Enter three numbers: ");
    scanf("%d%d%d", &num1, &num2, &num3);
    if (num1>num2 && num1>num3)
        printf("%d is maximum ", num1);
    else if(num2>num1 && num2>num3)
        printf("%d is maximum ", num2);
    else
        printf("%d is maximum ", num3);
    getch();
}
```

Output 1:

```
Enter two number: 100      80      50
100 is maximum
```

Output 2:

```
Enter a number: 80      100      50
100 is maximum
```

Output 3:

```
Enter a number: 80      50      100
100 is maximum
```

Solution 2

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int n1, n2, n3;
    printf("Enter three numbers: ");
    scanf("%d%d %d", &n1, &n2, &n3);

    if (n1 >= n2)
    {
        if (n1 >= n3)
            printf("%d is the largest number.", n1);
        else
            printf("%d is the largest number.", n3);
    }
    else
    {
        if (n2 >= n3)
            printf("%d is the largest number.", n2);
        else
            printf("%d is the largest number.", n3);
    }
    getch();
}
```

Output 1:

Enter two number: 100 80 50
100 is maximum

Output 2:

Enter a number: 80 100 50
100 is maximum

Output 3:

Enter a number: 80 50 100
100 is maximum

Program 7: WAP to Sum of first N natural numbers

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num, i, sum = 0;
    clrscr();
    printf(" Enter a positive number: ");
    scanf("%d", &num);
    for (i = 0; i <= num; i++)
    {
        sum = sum + i;
    }
    printf("\n Sum of the first %d numbers is= %d", num, sum);
    getch();
}
```

Output:

Enter a positive number: 5
Sum of the first 5 numbers is= 15

Program 8: WAP to print the division of two integers.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num1, num2, div;
    clrscr();
    printf("Enter first number: ");
    scanf("%d", &num1);
    printf("Enter second number: ");
    scanf("%d", &num2);
    div = num1 / num2;
    printf("Division of the numbers: %d", div);
    getch();
}
```

Output:

Enter first number: 20

Enter second number: 5

Division of the numbers: 4

Program 9: WAP to print the reverse of the number

```
#include<stdio.h>
void main()
{
    int n, reverse=0, rem;
    clrscr();
    printf("Enter a number= ");
    scanf("%d", &n);
    while(n!=0)
    {
        rem=n%10;
        reverse=reverse*10+rem;
        n=n/10;
    }
    printf("Reversed Number= %d",reverse);
    getch();
}
```

Output:

```
Enter a number= 1234
Reversed Number= 4321
```

Program 10: WAP to print the table of given number.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num, i = 1;
    clrscr();
    printf (" Enter a number to generate the table= ");
    scanf ("%d", &num);
    printf ("\n Table of %d \n ", num);
    while (i <= 10)
    {
        printf (" %d x %d = %d \n", num, i, (num * i));
        i++;
    }
    getch();
}
```

Output:

Enter a number to generate the table= 8

Table of 8

8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80

Program 11: WAP to calculate the factorial of given number

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,fact=1,n;
    clrscr();
    printf("Enter a number: ");
    scanf("%d", &n);
    for(i=1;i<=n; i++)
    {
        fact=fact*i;
    }
    printf("Factorial of %d is: %d", n, fact);
    getch();
}
```

Output:

Enter a number: 5
Factorial of 5 is: 120

Program 12: C program to find the value of nCr(Combination) using function

Logic

To find combination we use the concept of finding factorial of a number and use the standard formula for $nCr = \frac{n!}{r! * (n-r)!}$.

Run of the Program

Take input $n=5$ and $r=3$

$nCr = \frac{\text{fact}(n)}{\text{fact}(r) * \text{fact}(n-r)}$ i.e. $nPr = \frac{\text{fact}(5)}{\text{fact}(3) * \text{fact}(5-3)}$ i.e. $nPr = \frac{\text{fact}(5)}{\text{fact}(3) * \text{fact}(2)}$

Now function fact() is called.

We now calculate fact(5), fact(3) and fact(2)

int fact(int n)

$n=5$

Initialize $f=1$;

1st iteration for($i=1; i \leq n; i++$) i.e. for($i=1; 1 \leq 5; i++$)

$f=f*i$; i.e. $f=1*1$ i.e. **$f=1$**

2nd iteration for($i=2; i \leq n; i++$) i.e. for($i=2; 2 \leq 5; i++$)

$f=f*i$; i.e. $f=1*2$ i.e. **$f=2$**

3rd iteration for($i=3; i \leq n; i++$) i.e. for($i=3; 3 \leq 5; i++$)

$f=f*i$; i.e. $f=2*3$ i.e. **$f=6$**

4th iteration for($i=4; i \leq n; i++$) i.e. for($i=4; 4 \leq 5; i++$)

$f=f*i$; i.e. $f=6*4$ i.e. **$f=24$**

5th iteration for($i=5; i \leq n; i++$) i.e. for($i=5; 5 \leq 5; i++$)

$f=f*i$; i.e. $f=24*5$ i.e. **$f=120$**

Now we break out of the for loop as i will now be greater than $n(5)$.

In a similar way, we calculate fact(3) & fact(2) for which answer will be 2

Hence $5C3 = \frac{120}{6*2} = \mathbf{10}$.

```

#include<stdio.h>
#include<conio.h>

int fact(int);

void main()
{
    int n,r,ncr;
    clrscr();
    printf("Enter the value of n=");
    scanf("%d",&n);
    printf("Enter the value of r=");
    scanf("%d",&r);
    ncr=fact(n)/(fact(r)*fact(n-r));
    printf("Value of %dC%d = %d" ,n ,r, ncr);
    getch();
}

int fact(int n)
{
    int i,f=1;
    for(i=1;i<=n;i++)
    {
        f=f*i;
    }
    return f;
}

```

Output:

Enter the value of n= 5

Enter the value of r= 3

Value of $5C3 = 10$

Program 13: WAP to check whether given number is prime or not

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,i,m=0,flag=0;
clrscr();
printf("Enter the number to check prime:");
scanf("%d", &n);
m=n/2;
for(i=2;i<=m;i++)
{
if(n%i==0)
{
printf("Number is not prime");
flag=1;
break;
}
}
if(flag==0)
printf("Number is prime");
getch();
}
```

Output

Enter the number to check prime:56
Number is not prime

Enter the number to check prime:23
Number is prime

Program 14: WAP to find the factors of given number

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num, i;
    clrscr();
    printf("Enter the number: ");
    scanf("%d",&num);

    printf("Factors of %d are:\n", num);

    for(int i=1; i<=num/2; i++)
    {
        if(num%i==0)
            printf("%d\t", i);
    }

    getch();
}
```

Output:

Enter the number: 12

Factors of 12 are:

1 2 3 4 6

Program 14: WAP to check whether given number is perfect or not

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num, sum = 0, i;
    clrscr();
    printf("Enter a number = ");
    scanf("%d", &num);
    for(i = 1; i < num; i++)
    {
        rem = num % i;
        if (num%i == 0)
        {
            sum = sum + i;
        }
    }
    if (sum == num)
        printf(" %d is a Perfect Number");
    else
        printf("\n %d is not a Perfect Number");
    getch();
}
```

Output

Run 1:

Enter a number= 28

28 is a Perfect Number

Run 2:

Enter a number= 20

20 is not a Perfect Number

Program 15: WAP to find out GCD of two numbers

GCD or HCF of two numbers in C

$$24 - 2 \times 2 \times 2 \times 3$$
$$30 - 2 \times 3 \times 5$$

GCD = Multiplication of common factors

$$= 2 \times 3$$
$$= 6$$

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int n1, n2, i, res;
    clrscr();
    printf ( " Enter any two numbers: \n ");
    scanf ( "%d %d", &n1, &n2);
    for( i = 1; i <= n1 && i <= n2; ++i)
    {
        if (n1 % i == 0 && n2 % i == 0)
            res = i;
    }
    printf (" GCD of two numbers %d and %d is %d", n1, n2, res);
    getch();
}
```

Output

```
Enter any two numbers:
24
30
GCD of two numbers 24 and 30 is 6
```

1.6 Functions

Objectives

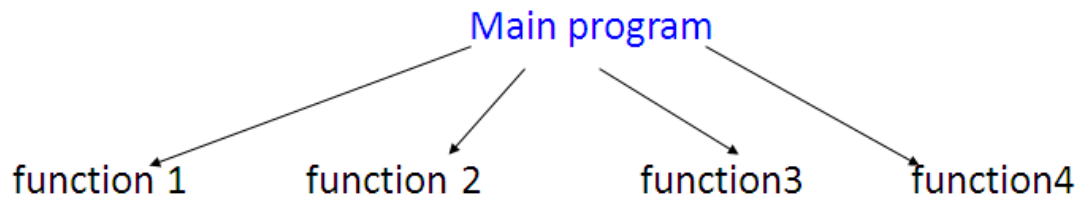
- To understand the concept of modularization.
- To know about the types of functions.
- To study about formal arguments and actual arguments.
- To understand the need of passing arguments to function.

Agenda

- Introduction to Function
- Types of C Functions
- Function naming rule in C
- General Form of a Function
- Parameters / Arguments
- Scope of a Function
- Returning Value – Control from a Function
- Three Main Parts of a Function
- Categorization based on Arguments and Return value
- Calling Functions – Two Methods
- Creating user defined header files
- Storage classes
- Summary

Introduction to Functions

Functions are the building blocks of C and the place where all program activity occurs.



Benefits of Using Functions:

- It provides modularity to the program.
- Easy code reusability. You just have to call the function by its name to use it.
- In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.

Contd..

A function is independent:

- It is “completely” **self-contained**
- It can be **called at any place** of your code and can be **ported** to another program
 - ✓ **reusable** - Use existing functions as building blocks for new programs
 - ✓ **Readable** - more meaningful
 - ✓ **procedural abstraction** - hide internal details
 - ✓ **factoring of code**- divide and conquer

Contd..

A function:

receives zero or more parameters,
performs a specific task, and
returns zero or one value

A function is invoked / called by name and parameters

Communication between function and invoker code is through the parameters and the return value

➤ In C, no two functions can have the same name

Types of C Functions

- Library function
- User defined function

Library function

- Library functions are the in-built function in C programming system

For example:

- ❖ `main()` - - The execution of every C program
- ❖ `printf()` - `printf()` is used for displaying output in C.
- ❖ `scanf()` - `scanf()` is used for taking input in C.

Some of the math.h Library Functions

- `sin()` → returns the sine of a radian angle.
- `cos()` → returns the cosine of an angle in radians.
- `tan()` → returns the tangent of a radian angle.
- `floor()` → returns the largest integral value less than or equal to x .
- `ceil()` → returns the smallest integer value greater than or equal to x .
- `pow()` → returns base raised to the power of exponent(x^y).

Some of the conio.h Library Functions

- `clrscr()` → used to clear the output screen.
- `getch()` → reads character from keyboard.
- `textbackground()` → used to change text background

Contd..

User defined function

- Allows programmer to define their own function according to their requirement.

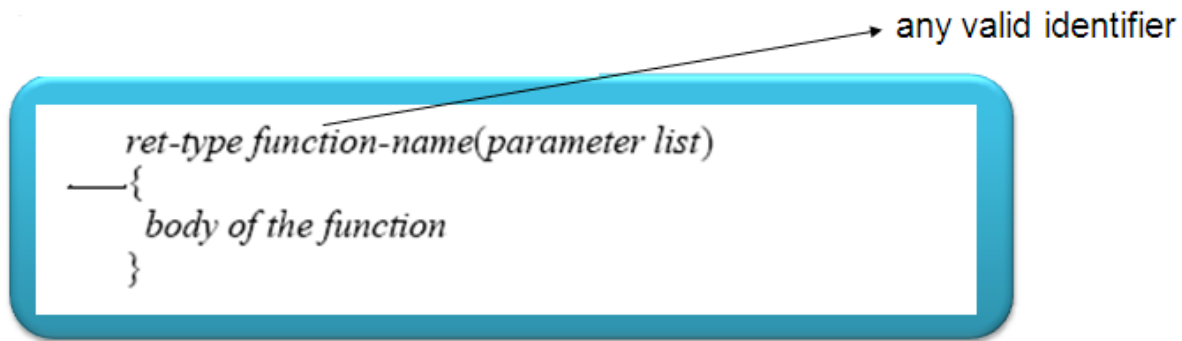
Advantages of user defined functions

- It helps to decompose the large program into small segments which makes programmer easy to understand, maintain and debug.
- If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
- Programmer working on large project can divide the workload by making different functions.

Function naming rule in C

- Name of function includes only alphabets, digit and underscore.
- First character of name of any function must be an alphabet or underscore.
- Name of function cannot be any keyword of c program.
- Name of function cannot be global identifier.
- Name of function cannot be exactly same as of name of function in the same scope.
- Name of function is case sensitive
- Name of function cannot be register Pseudo variable

The General Form a Function



The **ret-type** specifies the type of data that the function returns.

A function may **return any type** (default: `int`) of data **except an array**

The **parameter (formal arguments) list** is a **comma-separated list of variable names** and **their associated types**

The parameters **receive the values of the arguments** when the function is called

A function can be without parameters:

An **empty parameter list** can be explicitly specified as such by placing the **keyword void** inside the parentheses

More about formal argument/parameter list

(type varname1, type varname2, ..., type varnameN)

You can declare **several variables to be of the same type**

By using a comma separated list of variable names.

In contrast, all function parameters **must be declared individually**, each including both the type and name

- `f(int i, int k, int j)`
- `f(int i, k, float j)` Is it correct?
- `/* wrong, k must have its own type specifier */`

Scope of a function

Each function is a **discrete block of code**. Thus, a function **defines a block scope**:

A function's code is private to that function and cannot be accessed by any statement in any other function except through a call to that function.

- **Variables that are defined within a function are local variables**
- A local variable comes into existence when the function is entered and is destroyed upon exit
- A local variable **cannot hold its value between function calls**

Contd..

The **formal arguments / parameters** to a function also fall **within the function's scope:**

is known throughout the entire function comes into existence when the function is called and is destroyed when the function is exited.

Even though they perform the special task of receiving the value of the arguments passed to the function, they **behave like any other local variable**

Returning value, control from a function

If nothing returned

- return;
- or, until reaches right curly brace

If something returned

- return expression;

Only one value can be returned from a C function

A function can **return only one value**, though it can return **one of several values** based on the evaluation of certain conditions.

Multiple return statements can be used within a single function (eg: inside an “if-then-else” statement...)

The return statement not only returns a value back to the calling function,
it **also returns**
control back to the calling function

Three Main Parts of a Function

- Function Declaration (Function prototype)
- Function Definition
- Function Call

Structure of a C program with a Function

Function prototype giving the name, return type and the type of formal arguments

```
main( )  
{  
.....
```

Call to the function:

Variable to hold the value returned by the function = Function name with actual arguments

```
.....  
}
```

Function definition:

Header of function with name, return type and the type of formal arguments as given in the prototype

Function body within { } with local variables declared , statements and return statement

Function Prototype

- Functions should be **declared before they are used**
- Prototype only needed if function definition comes after use in program
- Function prototypes are always declared **at the beginning of the program** indicating :
 - name of the function, data type of its arguments &
 - data type of the returned value

return_type function_name (type1 name1, type2 name2, ..., typen namen);

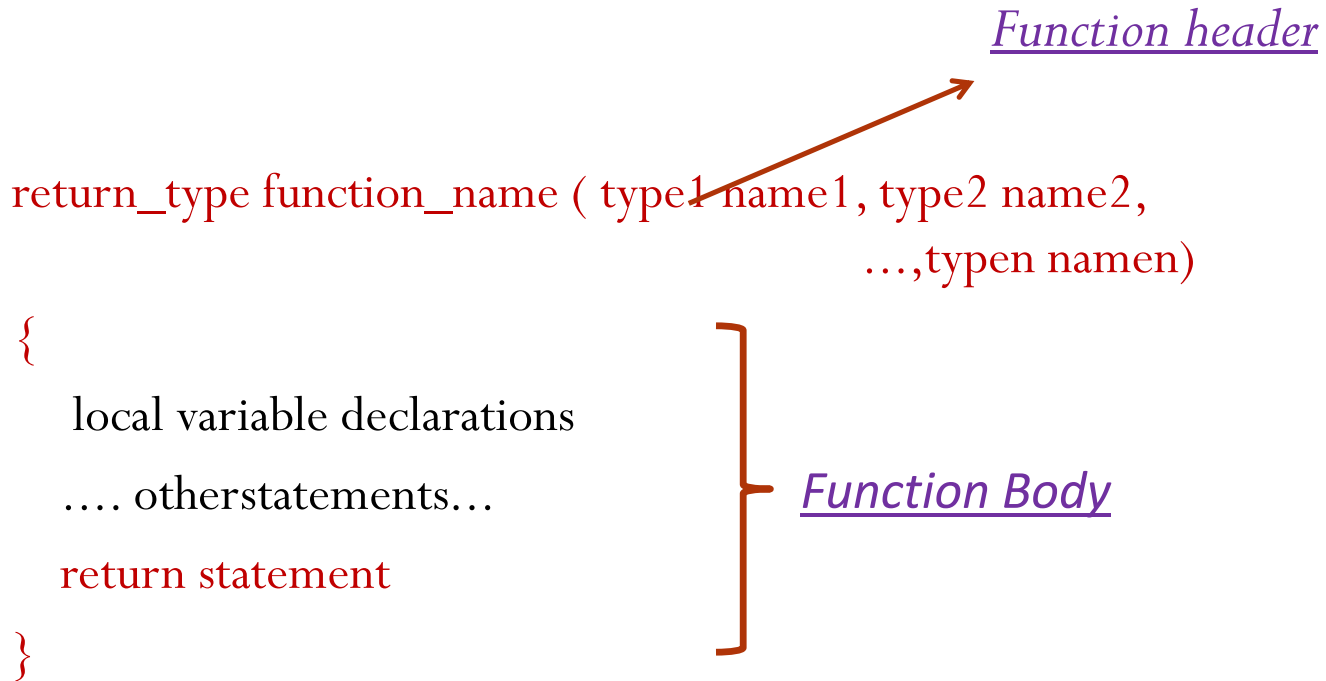
Function Definition

Function header

return_type function_name (type1 name1, type2 name2,
..., typen namen)

{
 local variable declarations
 otherstatements...
 return statement
}

Function Body



Function call

A function is **called from the main()**

A function can in turn **call a another function**

Function call statements **invokes the function** which means the **program control passes to that function**

Once the function completes its task, the **program control is passed back to the calling environment**

Contd..

Variable = function_name (actual argument list);

Or

Function_name (actual argument list);

Function **name and the type and number of arguments must match** with that of the function declaration stmt and the header of the function definition

Examples:

result = sum(5, 8); display(); calculate(s, r, t);

Return statement

To return a value from a C function you must explicitly return it with a return statement

```
return <expression>;
```

The expression can be **any valid C expression** that resolves to the type defined in the function header

```
add( int a, int b)
```

```
{  
  
    return (a + b);  
}
```

```
add( int a, int b)
```

```
{  
    int c = a + b;  
    return ( c );  
}
```

Ex: Function call: `int value = add(5,8)`

Here, `add()` sends back the value of the expression `(a + b)` or value of `c` to `main()`

Examples

Function Prototype Examples

```
double squared (double number);  
void print_report (int);  
int get _menu_choice (void);
```

Function Definition Examples

```
double squared (double number)  
{  
    return (number * number);  
}  
  
void print_report (int report_number)  
{  
    if (report_nmber == 1)  
        printf("Printer Report 1");  
    else  
        printf("Not printing Report 1");  
}
```

Example C program..

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

```
float average(float, float, float);
```

```
int main( )
```

```
{  
    float a, b, c;  
    printf("Enter three numbers please\n");  
    scanf("%f, %f, %f",&a, &b, &c);  
    printf("Avg of 3 numbers = %.3f\n", average(a, b, c) );  
    return 0;  
}
```

Function prototype

Function call

Contd..

The definition of function average:

```
float average(float x, float y, float z) // local variables x, y, z
```

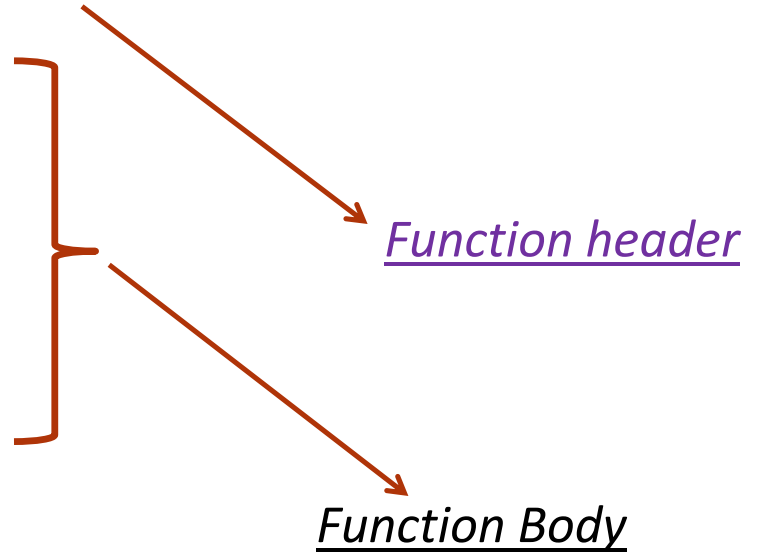
```
{
```

```
    float r; // local variable
```

```
    r = (x+y+z)/3;
```

```
    return r;
```

```
}
```



Categorization based on arguments and return value

- Function with no arguments and no return value
- Function with no arguments and return value
- Function with arguments but no return value
- Function with arguments and return value.

Credits : <http://www.programiz.com/c-programming/types-user-defined-functions>

Calling Functions – Two methods

Call by value

- **Copy of argument passed**
- Changes in function do not effect original
- Use when function does not need to modify argument
 - Avoids accidental changes

Call by reference

- **Passes original argument**
- Changes in function effect original
- Only used with trusted functions

Call by Value

When a function is called by an argument/parameter which **is not a pointer** the **copy of the argument is passed to the function.**

Therefore a **possible change on the copy does not change the original value of the argument.**

Example:

Function call **func1 (a, b, c);**

Function header **int func1 (int x, int y, int z)**

Here, the parameters **x , y and z** are initialized by the values of **a, b and c**

int x = a

int y = b

int z = c

Example C Program

```
void swap(int, int );
```

```
main( )
```

```
{  
    int a=10, b=20;  
    swap(a, b);  
    printf(" %d %d \n", a, b);  
}
```

```
void swap (int x, int y)
```

```
{  
    int temp = x;  
    x= y;  
    y=temp;  
}
```

Intricacies of the preceding example

- In the preceding example, the function `main()` declared and initialized two integers `a` and `b`, and then invoked the function `swap()` by **passing `a` and `b` as arguments** to the **function `swap()`**.
- The **function `swap()` receives the arguments `a` and `b` into its parameters `x` and `y`**. In fact, the function `swap()` receives a **copy of the values** of `a` and `b` into its parameters.
- The **parameters of a function are local to that function**, and hence, any **changes made by the called function to its parameters affect only the copy received by the called function**, and do not affect the value of the variables in the called function. This is the **call by value mechanism**.

Call by Reference

When a function is called by an argument/parameter which **is a pointer** (address of the argument) the **copy of the address of the argument is passed to the function**

Therefore, a **possible change on the data at the referenced address changes the original value of the argument.**

Will be dealt later in detail...

Illustration for creating user defined header files using user created functions

Make Your Own Header File ?

Step1 :Type this Code

```
int add(int a,int b)
{
    return(a+b);
}
```

- In this Code write only function definition as you write in General C Program

Step 2 : Save Code

- Save Above Code with [.h] Extension .
- Let name of our header file be myhead [myhead.h]
- Compile Code if required.

Step 3 :Write Main Program

```
#include<stdio.h>
#include"myhead.h"
main() {
    int num1 = 10, num2 = 10, num3;
    num3 = add(num1, num2);
    printf("Addition of Two numbers : %d", num3);
}
```

Here,

- Instead of writing < myhead.h> use this terminology **“myhead.h”**
- All the Functions defined in the **myhead.h header** file are now ready for use .
- **Directly call function add();** [Provide proper parameter and take care of return type]

Note : While running your program precaution to be taken :

Both files [**myhead.h and sample.c**] should be in same folder.

Storage Classes

- A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.
 - Automatic variables → auto
 - External variables → extern
 - Static variables → static
 - Register variables → register

auto - Storage Class

auto is the default storage class for all local variables.

```
{  
int Count;  
auto int Month;  
}
```

The example above defines two variables with the same storage class. auto can only be used within functions, i.e. local variables.

extern - Storage Class

- These variables are declared outside any function.
- These variables are active and alive throughout the entire program.
- Also known as global variables and default value is zero.

static - Storage Class

- The value of static variables persists until the end of the program.
- It is declared using the keyword static like
static int x;
static float y;
- It may be of external or internal type depending on the place of there declaration.
- Static variables are initialized only once, when the program is compiled.

register - Storage Class

- These variables are stored in one of the machine's register and are declared using register keyword.
eg. `register int count;`
- Since register access are much faster than a memory access keeping frequently accessed variables in the register lead to faster execution of program.
- Don't try to declare a global variable as register. Because the register will be occupied during the lifetime of the program.

1. Predict the output of the program.



```
#include<stdio.h>

int main()
{
    int fun(int);
    int i = fun(10);
    printf("%d\n", --i);
    return 0;
}

int fun(int i)
{
    return (i++);
}
```

1. Solution

Answer: 9

Explanation:

- **Step 1:** `int fun(int);` Here we declare the prototype of the function `fun()`.
- **Step 2:** `int i = fun(10);` The variable `i` is declared as an integer type and the result of the `fun(10)` will be stored in the variable `i`.
- **Step 3:** `int fun(int i){ return (i++); }` Inside the `fun()` we are returning a value `return(i++)`. It returns 10. because `i++` is the post-increment operator.
- **Step 4:** Then the control back to the `main` function and the value 10 is assigned to variable `i`.
- **Step 5:** `printf("%d\n", --i);` Here `--i` denoted pre-increment. Hence it prints the value 9.



2. What will be the output of the program?

```
#include<stdio.h>
int reverse(int);

int main()
{
    int no=5;
    reverse(no);
    return 0;
}
int reverse(int no)
{
    if(no == 0)
        return 0;
    else
        printf("%d,", no);
    reverse (no--);
}
```

2. Solution

Answer: Infinite loop

Explanation:

- **Step 1:** `int no=5;` The variable `no` is declared as integer type and initialized to 5.
- **Step 2:** `reverse(no);` becomes `reverse(5);` It calls the function `reverse()` with '5' as parameter.
- The function `reverse` accept an integer number 5 and it returns '0'(zero) if `(5 == 0)` if the given number is '0'(zero) or else `printf("%d", no);` it prints that number 5 and calls the function `reverse(5);`.
- The function runs infinitely because the there is a post-decrement operator is used. It will not decrease the value of 'n' before calling the `reverse()` function. So, it calls `reverse(5)` infinitely.
- Note: If we use pre-decrement operator like `reverse(--n)`, then the output will be 5, 4, 3, 2, 1. Because before calling the function, it decrements the value of 'n'.

3. Predict the output of the program.



```
#include<stdio.h>
int i;
int fun();

int main()
{
    while(i)
    {
        fun();
        main();
    }
    printf("Hello\n");
    return 0;
}
int fun()
{
    printf("Hi");
}
```


3.Solution

Answer: Hello

Explanation:

- Step 1: *int i*; The variable *i* is declared as an integer type.
- Step 1: *int fun()*; This prototype tells the compiler that the function *fun()* does not accept any arguments and it returns an integer value.
- Step 1: *while(i)* The value of *i* is not initialized so this while condition is failed. So, it does not execute the *while* block.
- Step 1: *printf("Hello\n");* It prints "Hello".
- Hence the output of the program is "Hello".

1. Point out the error in the program.



```
#include<stdio.h>

int main()
{
    int a=10;
    void f();
    a = f();
    printf("%d\n", a);
    return 0;
}

void f()
{
    printf("Hi");
}
```

1. Solution

Answer: Error: Not allowed assignment

Explanation:

The function *void f()* is not visible to the compiler while going through *main()* function. So we have to declare this prototype *void f();* before to *main()* function. This kind of error will not occur in modern compilers.

2. Point out the error in the program.



```
#include<stdio.h>
int f(int a)
{
    a > 20? return(10): return(20);
}
int main()
{
    int f(int);
    int b;
    b = f(20);
    printf("%d\n", b);
    return 0;
}
```

2. Solution

Answer:

Error: return statement cannot be used with conditional operators

Explanation:

In a ternary operator, we cannot use the return statement. The ternary operator requires expressions but not code.

3. There is an error in the below program. Which statement will you add to remove it?



```
#include<stdio.h>

int main()
{
    int a;
    a = f(10, 3.14);
    printf("%d\n", a);
    return 0;
}

float f(int aa, float bb)
{
    return ((float)aa + bb);
}
```

3. Solution

Answer: Add prototype: *float f(int, float);*

Explanation:

- The correct form of function *f* prototype is
float f(int, float);

Summary

- Discussed the modularization techniques in C.
- Illustration of functions with different parts – Prototype, Call and Definition.
- Discussed formal and actual parameters and passing mechanism.