Programming in C

Introduction

C is a general-purpose, structured programming language. Its instruction consists of terms that resemble algebraic expression, augmented by certain English keywords such as if, else, for, do, and while. It is used to develop system programming (e.g., For writing operating system) as well as for application programming (e.g., for writing a program to solve a complicated system of mathematical equations or for writing a program to bill customers).

History

A high-level programming language developed by Dennis Ritchie at Bell Labs in the mid 1970s. Although originally designed as a systems programming language, C has proved to be a powerful and flexible language that can be used for a variety of applications, from business programs to engineering. C is a particularly popular language for personal computer programmers because it is relatively small -- it requires less memory than other languages.

The first major program written in C was the UNIX operating system, and for many years C was considered to be inextricably linked with UNIX. Now, however, C is an important language independent of UNIX.

Although it is a high-level language, C is much closer to assembly language than are most other high-level languages. This closeness to the underlying machine language allows C programmers to write very efficient code. The low-level nature of C, however, can make the language difficult to use for some types of applications.

In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

The UNIX operating system, the C compiler, and essentially all UNIX application programs have been written in C. C has now become a widely used professional language for various reasons –

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around the early 1970s.

- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- The UNIX OS was totally written in C.
- Today C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art software have been implemented using C.
- Today's most popular Linux OS and RDBMS MySQL have been written in C.

Why use C?

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

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

Advantages of C Language

a. C language is a building block for many other currently known languages.
   C language has variety of data types and powerful operators. Due to this, programs written in C language are efficient, fast and easy to understand.
b. C is highly portable language. This means that C programs written for one computer can easily run on another computer without any change or by doing a little change.
c. There are only 32 keywords in ANSI C and its strength lies in its built-in functions. Several standard functions are available which can be used for developing programs.
d. Another important advantage of C is its ability to extend itself. A C program is basically a collection of functions that are supported by the C library this makes us easier to add our own functions to C library. Due to the availability of large number of functions, the programming task becomes simple.
e. C language is a structured programming language. This makes user to think of a problem in terms of function modules or blocks. Collection of these modules makes a complete program. This modular structure makes program debugging, testing and maintenance easier.
f. C is portable language.
g. C language is easy for beginners.
h. C language support system programming.
i. C language support no of operators.

j. C is the collection of lot of library files.

Disadvantages of C Language

a. C does not have concept of OOPs, that's why C++ is developed.
b. There is no runtime checking in C language.
c. C doesn't have the concept of namespace.
d. C doesn't have the concept of constructor or destructor.
e. C language has no script type checking.
f. C language has no run time checking mechanism.
g. C does not support oops features.

Structure of C Program

A C program basically has the following form:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

The following program is written in the C programming language. Open a text file hello.c using TC editor.

```
#include <stdio.h>
int main()
{
/* My first program */
printf("Hello, World! \n");
return 0;
}
```

Preprocessor Commands: These commands tell the compiler to do preprocessing before doing actual compilation. Like #include <stdio.h> is a preprocessor command which tells a C compiler to include stdio.h file before going to actual compilation. You will learn more about C Preprocessors in C Preprocessors session.

Functions: are main building blocks of any C Program. Every C Program will have one or more functions and there is one mandatory function which is called main() function. This function is prefixed with keyword int which means this function returns an integer value when it exits. This integer value is returned using return statement.

The C Programming language provides a set of built-in functions. In the above example printf()is a C built-in function which is used to print anything on the screen. Check Built-in function section for more detail.

You will learn how to write your own functions and use them in Using Function session.

Variables: Variables are used to hold numbers, strings and complex data for manipulation. You will learn in detail about variables in C Variable Types.

Statements & Expressions: Expressions combine variables and constants to create new values. Statements are expressions, assignments, function calls, or control flow statements which make up C programs.

Comments: are used to give additional useful information inside a C Program. All the comments will be put inside /*...*/ as given in the example above.

This will produce following result

Hello, World

Congratulations!! you have written your first program in "C". Now believe me its not difficult to learn "C".

C Program Compilation Process

To compile a C program you would have to Compiler name and program files name. Assuming your compiler's name is cc and program file name is hello.c, give following command at Unix prompt.

$cc hello.c

This will produce a binary file called a.out and an object file hello.o in your current directory. Here a.out is your first program which you will run at Unix prompt like any other system program. If you don't like the name a.out then you can produce a binary file with your own name by using -o option while compiling C program. See an example below:

$ cc-ohello hello.c

Now you will get a binary with name hello. Execute this program at Unix prompt but before executing / running this program make sure that it has execute permission set.

This will produce following result

Hello, World

Congratulations!! you have written your first program in "C". Now believe me its not difficult to learn "C".

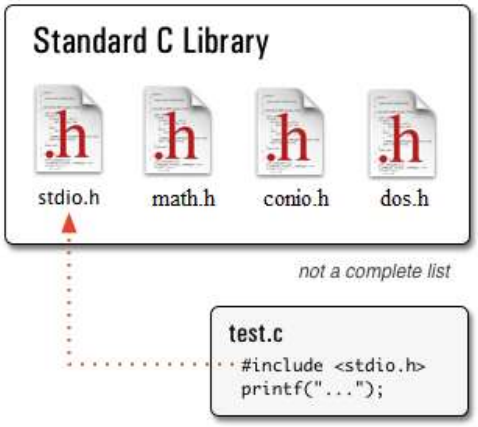
Header Files

Header files contain definitions of functions and variables, which is imported or used into any C program by using the pre-processor #include statement. Header file have an extension ".h" which contains C function declaration and macro definition.

Each header file contains information (or declarations) for a particular group of functions. Like stdio.h header file contains declarations of standard input and output functions available in C which is used for get the input and print the output. Similarly, the header file math.h contains declarations of mathematical functions available in C.

Header Files

Header files contain definitions of functions and variables, which is imported or used into any C program by using the pre-processor #include statement. Header file have an extension ".h" which contains C function declaration and macro definition.



Each header file contains information (or declarations) for a particular group of functions. Like stdio.h header file contains declarations of standard input and output functions available in C which is used for get the input and print the output. Similarly, the header file math.h contains declarations of mathematical functions available in C.

TC Editor

TC Editor is very simple and easy to use; here i will give you all tips related to TC Editor and some shortcut keys related to TC Editor which is very useful at the time of coding. Turbo C is a most common C language compiler. Below i will discuss all about its Interfaces.

TC Editor

The interface of Turbo C is very simple. When IDE screen appears, the menu bar is activated. It contains various menus such as;

- File: This menu contains group of commands used for save , edit , print program, exit from Turbo C editor etc.
- Edit: This menu contains group of commands used for editing C program source code. Example Copy, Cut, Paste, Undo etc.
- Search: This menu contains group of commands used for searching specific word as well as replacing it with another one.
- Run: This menu contains group of commands used for running C program.
- Compile: This menu contains group of commands used for compiling C program.
- Debug: This menu contains group of commands used for debugging C program.
- Project: This menu contains group of commands used for opening, closing and creating projects.
- Options: This menu contains group of commands used for configuring IDE of Turbo C and setting up directories etc.
- Windows: This menu contains group of commands used for opening, closing various windows of IDE.
- Help: This menu is used to get help about specific topic of C language. Similarly to get help about a specific keyword or identifier of C.

Shortcut keys Related to TC Editor

- Alt + x : Close TC Editor.
- Clt + f9 : Run C Program.
- Alt + f9 : Compile C Code.
- Alt + Enter : Get Full Screen or Half Screen TC Editor.
- Clt + y : Delete complete line above the cursor.
- Shift + Right arrow : Select Line of Code.
- Clt + Insert : Copy.
- Shift + Insert : Paste.
- Shift + Delete : Delete.

Fundamentals of C

Character Set used in C

Character set: - The character set is the fundamental raw material of any language and they are used to represent information. Like natural languages, computer language will also have well defined character set, which is useful to build the programs.

The characters set in C are grouped into the following two categories:

1. **Source character set**
    a. Alphabets
    b. Digits
    c. Special Characters

d. White Spaces

2. Execution character set

1. Source character set

ALPHABETS

Uppercase letters	A-Z
Lowercase letters	a-z

DIGITS 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

SPECIAL CHARACTERS

~ tilde	% percent sign	| vertical bar	@ at symbol
+ plus sign	< less than	_underscore	- minus sign
>greater than	^ caret	# number sign	= equal to
& ampersand	$ dollar sign	/ slash	(left parenthesis
* asterisk	\back slash	)right parenthesis	'apostrophe
: colon	[left bracket	" quotation mark	;semicolon
]right bracket	!exclamation mark	, comma	{ left flower brace
? Question mark	.dot operator	}right flower brace	

Use of Comments

In the C Programming Language, we can place comments in your source code that are not executed as part of the program.

Comments provide clarity to the C source code allowing others to better understand what the code was intended to accomplish and greatly helping in debugging the code. Comments are especially important in large projects containing hundreds or thousands of lines of source code or in projects in which many contributors are working on the source code.

A comment starts with a slash asterisk /* and ends with a asterisk slash */ and can be anywhere in your program. Comments can span several lines within your C program. Comments are typically added directly above the related C source code.

Adding source code comments to your C source code is a highly recommended practice. In general, it is always better to over comment C source code than to not add enough.

Syntax

The syntax for a comment is:

```
/* comment goes here */
```

OR

```
/*
 * comment goes here
 */
```

Note

• It is important that we choose a style of commenting and use it consistently throughout your source code. Doing so makes the code more readable.

Example - Comment in Single Line
You can create an comment on a single line.
For example:

```
/* Author: Dennis Ritchie */
```

Example - Comment Spans Multiple Lines
We can create a comment that spans multiple lines. For example:

```
/*
 * Author: TechOnTheNet.com
 * Purpose: To show a comment that spans multiple lines.
 * Language:  C
 */
```

The compiler will assume that everything after the /* symbol is a comment until it reaches the */ symbol, even if it spans multiple lines within the C program.
Example - Comment at End of Code Line
You can create a comment that displays at the end of a line of code.
For example:

```
#define AGE 6  /* This constant is called AGE */
```

C tokens:

C tokens are the basic buildings blocks in C language which are constructed together to write a C program.Each and every smallest individual units in a C program are known as C tokens.

C tokens are of six types. They are,

1. Keywords (eg: int, while),
2. Identifiers (eg: main, total),
3. Constants (eg: 10, 20),
4. Strings (eg: “total”, “hello”),
5. Special symbols (eg: (), {}),
6. Operators (eg: +, /,-,*)

C tokens example program:

```
int main()
{
int x, y,total;
x = 10, y = 20;
total = x + y;
printf (“Total = %d \n”, total);
}
```

Identifier

Identifiers are the names we can give to entities such as variables, functions, structures etc.Identifier names must be unique. They are created to give unique name to a C entity to identify it during the execution of a program.Both uppercase and lowercase letter are permitted, through common usage favors the use of lowercase letters for most types of identifiers.

For example:
int money;
double accountBalance;
Here, money and accountBalance are identifiers.

Also remember, identifier names must be different from keywords. we cannot use int as an identifier because int is a keyword.

Rules for writing an identifier

a. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscore only.
b. The first letter of an identifier should be either a letter or an underscore. However, it is discouraged to start an identifier name with an underscore. It is because identifier that starts with an underscore can conflict with system names. In such cases, compiler will complain about it. Some system names that start with underscore are _fileno, _iob, _wfopen etc.
c. There is no rule on the length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler. So, the first 31 letters of two identifiers in a program should be different.

The following names are not valid identifiers.

d. 4th the first character must be a letter.
e. "X" illegal characters(")
f. order-no illegal characters(-)
g. error flag illegal characters(blank space)

Keyword

Keyword is a predefined or reserved word in C library with a fixed meaning and used to perform an internal operation. C Language supports 32 keywords. Every Keyword exists in lower case latter like auto, break, case, const, continue, int etc. Keywords are predefined, reserved words used in programming that have special meaning. Keywords are part of the syntax and they cannot be used as an identifier. For example:

int money;

Here, int is a keyword that indicates 'money' is a variable of type integer.

As C is a case sensitive language, all keywords must be written in lowercase. Here is a list of all keywords allowed in ANSI C.

Keywords used in C Language

auto	Double	int	switch
break	Else	union	long
case	enum	void	register
char	extern	while	return
continue	For	volatile	signed
do	unsigned	if	struct
default	goto	sizeof	typedef
const	float	short	static

Along with these keywords, C supports other numerous keywords depending upon the compiler.

All these keywords, their syntax and application will be discussed in their respective topics. However, if you want a brief information on these keywords without going further, visit list of all keywords in C programming.

Data types in C

C supports several different types of data types, each of which may be represented differently, within the computer's memory. In C programming, variables or memory locations should be declared before it can be used. Similarly, a function also needs data type to be declared before use. Data types simply refers to the type and size of data associated with variables and functions.

Data types in C

1. Fundamental Data Types

- Integer types
- Floating type
- Character type

2. Derived Data Types

- Arrays
- Pointers
- Structures
- Enumeration

Data type	Description	Typical memory Requirements
int	integer quantity	2 bytes or one word (varies from one compiler to another)
char	single character	1 Byte
float	floating point number (i.e. a number containing a decimal point and / or an exponent)	1 word(4 Bytes)
double	double precision floating point number (i.e.more significant figure, and an exponent which may be larger in magnitude)	2 words (8 bytes)

a) Integer data types

Integers are whole numbers that can have both positive and negative values, but no decimal values. The format specifier is %d Example: 0, -5, 10

In C programming, keyword int is used for declaring integer variable. For example:

int id;

Here, id is a variable of type integer.

we can declare multiple variable at once in C programming. For example:

int id, age;

The size of int is 2 bytes.

Examples

```
{
int a;
a=23;
printf("%d",a);
}
```

output 23

b) Character data types

The Keyword char is used to represent individual character type variable. It is placed in single quotes (' '). It occupies 1 byte space in memory of computer. The format specifier is %c. it may be categorized into three types:

char, signed char and unsigned char.

Examples

```
char first='a';
char second='b';
printf("%c%c",first,second);
```

output: ab

c) Floating data type

Floating data type variables can hold real numbers such as: 2.34, -9.382, 5.0 etc. which can hold interger as well as fractional part as its values. It occupies 4 bytes. We can declare a floating point variable in C by using either float or double keyword. The format specifier is %f For example:

```
float accountBalance;
double bookPrice;
```

Here, both accountBalance and bookPrice are floating type variables.

```
float a;
a=3.457
printf("%f",a);
```

if we used larger range of data then we can use double, which occupies 8 bytes in memory.
The format specifier is %1f.

```
double a;
a=45.56789;
printf("%1f",a);
```

another type of data i.e. long double.
Format specifier is %2f.

Constants and Variables

Constants:

In C, a quantity which does not change during the execution of the program is called constant.

For e.g. radius = 3.14 is a constant
#define pi 3.1415
#define length 5.5
There are four basic types of constants in C;

a) integer constants b) floating point constants c) character constants d) string constant

a) **Integer Constants**:

An integer constant is an integer-valued number which consists of digits. It may contain either + or – sign. Commas cannot appear in an integer constant (e.g. 12,000).

E.g. 54312 7777

b) **Floating point Constants**:

A floating point constant is a number that can be written either in decimal form or an exponent form (or both).

It must have at least one digit before decimal point and at least one digit after the decimal.

It may also have either + or - sign.

E.g.

Valid	Invalid
7.0	7
17.6	17.
-14.5	-14.5.456
0.8	.8

c) **Character Constant**:

A character constant is a single character enclosed in apostrophes (single quotation marks).

Example: 'B', '2', '$', 'f'

d) **String Constant**:

A string constant is a set of characters enclosed in double quotation marks. It may be alphabet, number, special characters and blank spaces.

E.g. "Green", "Washington", "D.C. 2005", "2*(I+3)/j", " "

Variables:

Variable is a symbolic name which is used to store different types of data in the computer's memory.Variable is an identifier that is used to represent a single data item, i.e. a numerical quantity or a character constant. The data item must be assigned to the variable at some point in the program.

E.g. int a, b, c;
char d;
a=3;
b=5;
c=d+b;
d= ‘a’;
a,b,c are integer variable
d is char-type variable.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables. Here all the variables a, b and c are local to main() function.

```
#include <stdio.h>
 int main ()
{
   /* local variable declaration */
   int a, b;   int c;
 /* actual initialization */
a = 10;   b = 20;   c = a + b;
printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
return 0;
}
```

Global Variables

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

```
#include <stdio.h>
   /* global variable declaration */
int g;
int main ()
{
 /* local variable declaration */
int a, b;
/* actual initialization */
a = 10;
b = 20;
 g = a + b;
printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
return 0;
}
```

Static variable:

A static variable is a variable that has been allocated statically—whose lifetime or "extent" extends across the entire run of the program. This is in contrast to the more ephemeral automatic variables (local variables are generally automatic), whose storage is allocated and deallocated on the call stack; and in contrast to objects whose storage is dynamically allocated in heap memory.

When a program (executable or library) is loaded into memory, static variables are stored in the data segment of the program's address space (if initialized), or the BSS segment (if uninitialized), and are stored in corresponding sections of object files prior to loading.

The static keyword is used in C and related languages both for static variables and other concepts.

Example:

```
#include <stdio.h>
/* function declaration */
void func(void);
static int count = 5;
/* global variable */
main() {
while(count--)
{
func();
}
return 0;
}
/* function definition */
void func( void )
{
 static int i = 5;
/* local static variable */
i++;
printf("i is %d and count is %d\n", i, count);
}
```

Output:

```
i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0
```

Type of specifier:

It is also known as format specifier. While the data is being outputted or inputted, it must be modified with some identifiers and their format specifiers. Format specifiers are the characters starting with % sign and followed with a character. It is also called “conversion specification”.

Format specifier	Description	Supported data types
%c	Character	char unsigned char
%d	Signed Integer	short unsigned short int long
%e or %E	Scientific notation of float values	float double
%f	Floating point	float
%g or %G	Similar as %e or %E	float double
%hi	Signed Integer(Short)	short
%hu	Unsigned Integer(Short)	unsigned short
%i	Signed Integer	short unsigned short int,long
%l or %ld or %li	Signed Integer	long
%lf	Floating point	double
%Lf	Floating point	long double
%lu	Unsigned integer	unsigned int unsigned long
%lli, %lld	Signed Integer	long long
%llu	Unsigned Integer	unsigned long long
%o	Octal representation of Integer.	short, unsignedshort int, unsigned, int, long
%p	Address of pointer to void void *	void *
%s	String	char *
%u	Unsigned Integer	unsigned int unsigned long
%x or %X	Hexadecimal representation of Unsigned Integer	short unsigned short int unsigned int long
%n	Prints nothing	
%%	Prints % character	

For example:

```
scanf("%d",&a);
printf("Area%d",a);
```

Statements: Simple and Compound Statements

A statement causes the computer to carry out some action. There are three different classes of statements in C;

a) Simple or expression statements b) Compound statements
c) Control statements

a) **Simple statement**:

Simple statements are those statements which are ended with semicolon ready to perform some action.

Example:

```
 a=33;
c=a+b;
printf("Sum=%d",sum);
```

Note: ; null statement

b) **Compound statement:**

A compound statement consists of several individual statements enclosed within a pair of braces { }. There should not be semicolon at the end of compound statement.

Example:

```
{
pie=3.141593;
circumference=2*pie*radius;
area = pie*radius*radius;
}
```

c) **Control Statement:**

They are used to create special program feature such as logical tests, loops and branches.

Example:

```
main()
{
int n=5,count=1;
int sum;
while (count<=n)
{
printf("n=");

sum=sum+n;
count++;
}
Printf("%d",sum);
}
```

Operators and Expressions

Operators: precedence & Associativety

Operator is a special symbol that tells the compiler to perform specific mathematical or logical Operation. Operator means to operate the operands.

Example: p = a+b ; Here, '=' & '+' are operators, 'a' & 'b' are operand and 'p = a+b' is an operation.

The data item act upon operator are called operands. The portion which indicates the action to be performed on operands is called operators.

There are different types of operators:

1) Arithmetic operators
2) Relational operators
3) Equality operator
4) Logical operators
5) Assignment operators
6) Unary operators
7) Conditional operators
8) Size of operator
9) Comma operator

1) **Arithmetic operator:**

These operators are used to perform the mathematical calculations. It usually takes two operands. C contains five arithmetic operators. Let us suppose variable a hold 11 and b hold -3

Operators	Purpose	Example	Result (a=11, b=-3)
+	Addition	a+b	8
-	Subtraction	a-b	14
*	Multiplication	a*b	-33
/	Divide	a/b	-3
%	Modulus	a%b	2 (Remainder after division)

2) **Relational operator:**

Relational operator determines the relationship between two different operands. It is also called comparison operator. Which can be used to check the Condition, it always return true or false. C supports four relational operator.

Operator	Meaning	Example
<	less than	a<b
<=	less than or equal to	a<=b
>	greater than	a>b
>=	greater than or equal to	a>=b

3) **Equality operator**

Closely associated with the relational operator, there are more two operator called equality operator.

==	equal to	a==b
!=	not equal to	a!=b

These operators represent condition that are either True or False; 1 for true and 0 for false.

Examples:

Suppose i, j and k are integer variable whose value are 1, 2 and 3.

Expression	Interpretation	Value
i<J	True	1
(i+j)>=k	True	1
(j+k)>(i+5)	False	0
k!=3	False	0
j==2	True	1

4) **Logical operator**

Logical operators logically connect the logical expressions, i.e. it is used to connect two or more expressions. It is used to give logical value either true or false.

Operator	Meaning	Examples
&&	logical AND	(a>b)&&(a>c)
||	logical OR	(a>b)||(a>c)
!	logical NOT	(a!=b)

Examples:

The result of logical AND (&&) operation will be True only if both operands are True.

The result of logical OR(||) operation will be True if either operand is True or if both operands are True. On the other hand, the result of a logical OR operation will be false only if both operands are false.

Example: i = 7, f = 5.5, c=w=119, p=112

Expression	interpretation	Value
(i>=6)&&(c=='w')	True	1
(i>=6)||(c==119)	True	1
(f<11)&&(i>100)	False	0
(c!='p')||((i+f)<=10)	True	1

5) **Assignment operator**

Assignment operator means to assign the value of an expression to an identifier.

The most commonly used assignment operator is '='.

It can be written, identifier = expression

When identifier generally represents a variable and an expression represents a constant, variable, or a more complex expression.

E.g.
a=3;

x=y;
delta=0.01;
sum=a+1;
area=length*breadth;

6) **Conditional operator**

C offers a conditional operator (?:). An expression that makes use of the conditional operator is called a conditional expression.

A conditional expression is written in the form,

expression1?expression2:expression3

expression1 is evaluated first. If the expression1 is true then expression2 is evaluated otherwise expression3 is evaluated.

a=5;
b=20;
y=(a>b)?a:b;

Output: y=20 if expression1 is false then expression3 is evaluated.

y=5 if expression1 is true as expression2 is evaluated.

Here, 20>5 is true so the output is y=20

7) **Comma operator**

C allows us to put multiple expression in the same statement, sperated by comma(,). It is used in loop.

E.g.
```
int x=5, y=99;
int x,y,z;
printf("%d%f",x,y);
scanf("%d%f",&a,&b);
```

8) **Unary operator**

C includes a class of operators that act upon a single operand to produce a new value, which is called unary operator. It is mostly used in loops.

There are two types of unary operators:

i) **increment operator**, ++, increment operator causes its operand to be increased by 1.

ii) **decrement operator**, --, decrement operator causes its operand to be decreased by 1.

The operand used with each of these operators must be single variable.

E.g. i=5=integer value

The expression ++i, is equivalent to i = i+1 ; i =5+1=6

The expression --i, is equivalent to i=i-1; i=5-1=4

If the operator (++i) then the operand will be changed in value before it is utilized for its intended purpose within the program.

If the operator (i++), then the value of the operand will be changed after it is utilized.

Example:

```
i=1;
printf("i=%d\n",i);
printf("i=%d\n",++i);
printf("i=%d\n",i);
```

Output:

i) causes the original value of i to be displayed, ie i =1
ii) second statement inrement i and then display its value, i.e. i =2
iii) final value of i is displayed by the last statement, i.e. i=2

Example:

```
i=1;
printf("i=%d\n",i);
printf("i=%d\n",i++);
printf("i=%d\n",i);
```

Output:

i. i = 1; first statement causes the original value of to be dispalyed
ii. i = 1; second statement causes the current value of i to be displayed and then increment (i =1+1=2)
iii. i = 2; final statement displays the value of i, i.e. 2

Example:

```
int x=5,y;
y=x++; /* stores 5 in y and then increases x to 6 from 5*/
y=++x; /*first increases x by 1 then stores in y, i.e. y = 6*/
```

Note: a*=b+1 is same as a=a*(b+1); x*=4 is same as x=x*4; x+=2 is same as x=x+2; x-=2 is same as x=x-2

9) **Sizeof operator**

It returns the number of bytes the operand occupies. To determine the number of bytes occupied by the integer, we can use sizeof operator.

Example:

```
int i, float x
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i;
float x;
char c;
double d;
```

```
printf("integer %d\n", sizeof i);
printf("character %d\n", sizeof c);
printf("double %d\n", sizeof d);
printf("float %d\n", sizeof x);
getch();
}
```

OR,

```
void main()
{
clrscr();
printf("integer %d\n", sizeof (int));
printf("character %d\n", sizeof (char));
printf("double %d\n", sizeof (double));
printf("float %d\n", sizeof (float));
getch();
}
```

Expressions:

An expression is a combination of variables, constants and operators written according to the syntax of C language.

Algebraic expression	C expression
ab+b(c+d)	a*b+b*(c+d)
(x+y)(x-y)	(x+y)*(x-y)
3x+5	3*x+5
$\frac{a+b}{c+d}$	(a+b)/(c+d)
x^4+x^3	x*x*x*x+x*x*x or pow(x,4)+pow(x,3)
$x^3(x+1)$	x*x*x*(x+1) or pow(x,3)*(x+1)

pow(x,3) means that power of (pow) x is 3 ,i.e. x^3

Type Casting and Conversions

Type casting is process to convert a variable from one data type to another data type. For example if we want to store a integer value in a float variable then we need to typecast integer into float.

In an expression, when we try to evaluate two or more types of data types, then one data type should be converted to another type and final result can only be represented into one data type.

For example:

a+b i.e. expression

a is type of **int** and b is **float**.

In this case, smaller data type is converted to higher data type automatically by the compiler, which is called automatic type conversion. In this case, final result is float.

Example

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b;
float sum;
clrscr();
printf("Enter two no. ");
scanf("%d%d",&a,&b);
sum=a+b;
printf("Sum: %f",sum);
getch();
}
```

Output

Enter two no. 3
4
Sum: 7.000000

long double ← double ← float ← int ← char

There are two types of conversion;

a) **Implicit type conversion**

C converts automatically by the compiler itself is called implicit type conversion.

long double←double←float←int←char

If either of the type long double, then other data are also converted to long double.

If either of the type double, then other data are also converted to double.

If either of the type are float type, then other data are also converted to float.

If either of the data type are int type, then other data are also converted to int.

Example:

```
float a;
int b,c;
b=3,c=2;
a=b/c;
printf(“%f”,a);
```

Output: 1.000000

b and c are integer, and division is also integer, but final value is converted to float because a is float.

b) **Explicit type conversion**

If we want to convert any type of single variable or mixed variable into another type then it is known as explicit type conversion.

Example:

```
float a;
int b,c;
b=3,c=2;
a=(float)b/c; /* b is converted into float*/
printf("%f",a); /*The result is 1.500000 */
float a;
int b,c;
b=3,c=2;
a=(float)(b/c); /* The result of b/c is converted into float */
printf("%f",a); /*The result is 1.000000 */
```

and,

```
float a;
int b,c;
b=3,c=2;
a=b/(float)c; /* c is converted into float*/
printf("%f",a); /* The result is 1.500000 */
float x;
int num1, num2;
num1=10, num2=3;
x=(float)num1/(float)num2; /* num1 and num2 are converted into float */
printf("%f",x); /* The result is 3.333333 */
```

Introduction to Library Function

Introduction to library functions:

Library functions are those which are predefined in C compiler.The functions that are pre-defined in programming language are called library functions. Some library functions are used for input/output operations, some are used for string processing operations and some are used for mathematical operation. printf(), scanf(), clrscr(), pow() etc. are pre-defined functions.

Library function:

It is part of C compiler package. Some of the library function are printf(), scanf(), clrscr(), getch(), strlen(), sqrt(), etc.

Library function	Type	Meaning
abs(i)	int	Returns the absolute value of i
cos(d)	double	Returns the cosine of d
exp(d)	double	Raise to power
getchar()	int	Enter the character from the standard input device.
log(d)	double	Return the natural logarithm of d.

pow(d1,d2)	double	Return d1 raised to the d2 power, i.e. $d1^{d2}$
printf()	int	Send value to the standard output device
scanf()	int	Enter value from the standard input device
strlwr()		converts into lowercase
strupr()		converts into uppercase
sqrt()		To find square root

Input/Output (I/O) Functions

Input Function:

Input functions or streams are used to read different type of data(such as numeric, string or character) from the keyboard. We can use more than one type of input functions in the program.

scanf():

scanf() is the input function of standard library. In C language, scanf() function can be used without giving reference to standard library but we need to include the stdio.h header file. scanf() function is used to read data from keyboard. The syntax of scanf() statement is

scanf("list of format specifier",&list of variables);

The first part of the general syntax of scanf() fnction is the list of format specifier which depends on the data types of variables to be read enclosed in double quotes. Moreover, the list of format specifier can also used with printf function. The % is used to tell or instruct compiler that nest coming character is a format specifier.

The second part of the syntax of scanf function is the list of variables. Each variable name start with a symbol @ (called adreess operator) is known as address operator which refers to the address of that variable which is going to be read.

Example:

To read two integer variable a and b of int data type
scanf(%d%d",&a,&bb);

Example:

```
#include<stdio.h>
main()
{
char item[20];
int partno;
float cost;
…….
….……
scanf("%s %d %f", item, &partno, &cost);
```

……
}

Within the scanf function the control string is "%s %d %f".

%s indicates that argument (item) represents a string

%d indicates that argument(partno) represents an integer value

%f indicates that argument(cost) represents a floating point value

Note: The numerical variable partno and cost are preceded by ampersands within the scanf function.

An ampersand does not precede item, however, since item is an array name.

scanf("%s %d %f", item, &partno, &cost);

The following data can item can be entered from the standard input device, when the program is executed.

Fastener 12345 0.05

Output Function

Output function are used to display different type of data(such as numeric, string or character)and results on the screen. We can use more than one types of output function in the program

printf();

printf() is used the output function of the standard library. In C language printf() function can be used without giving reference of any standard library but we need to include stdio.h hrader file.

Syntax:

printf("list of format specifier and escape sequence",list of variables);

First part of printf() syntax refers a list of format specifier which help to display data of variable according to their data types and list of escape sequence to be printed data . In C \(backslash)is considered as a escape character.

Example

If x is a variable with integer value 5
printf("%d",x);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
float i = 2.0, j = 3.0;
printf("%f %f %f %f",i, j, i+j, sqrt(i+j));
```

}

Output: 2.000000 3.000000 5.000000 2.236068

Example2:

```
#include<stdio.h>
main()
{
char item[20];
int partno;
float cost;
……..
…..……
printf(“%s %d %f”, item, partno, cost);
……
}
```

Within the printf function the control string is “%s %d %f”.

%s indicates that argument (item) represents a string

%d indicates that argument(partno) represents an integer value

%f indicates that argument(cost) represents a floating point value

Q. Solve for the roots of the quadratic equation $ax^2+bx+c=0$ using the well-known formula,

$$\boldsymbol{x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}}$$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
        clrscr();
        float a,b,c,d,x1,x2;
        /*input data*/
        printf("enter a,b,c");
        scanf("%f%f%f",&a,&b,&c);
        d=sqrt(b*b-4*a*c);
        x1=(-b+d)/(2*a);
        x2=(-b-d)/(2*a);
        printf("\nx1 = %e",x1);
        printf("\nx2 = %e",x2);
        getch();
}
```

Control Structures

A statement that is used to control the flow of execution in a program is called control structure. It combines instruction into logical unit. Logical unit has one entry point and one exit point.

Types of control structures

1. Sequence
2. Selection
3. Repetition/Looping

1. Sequence:

Statements are executed in a specified order. No statement is skipped and no statement is executed more than once.

Syntax:

Start

Instruction 1;

Instruction 2;

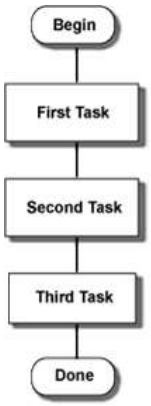
Instruction 3;

…..

…..

Instruction n;

End



Example:

A C program to read two numbers and find sum of their square:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a,b;
int sum=0;
printf("enter two numbers");
scanf("%d%d",&a,&b);
sum=a*a+b*b;
printf("\nThe sum of squares of two nos. = %d",sum);
getch();
}
```

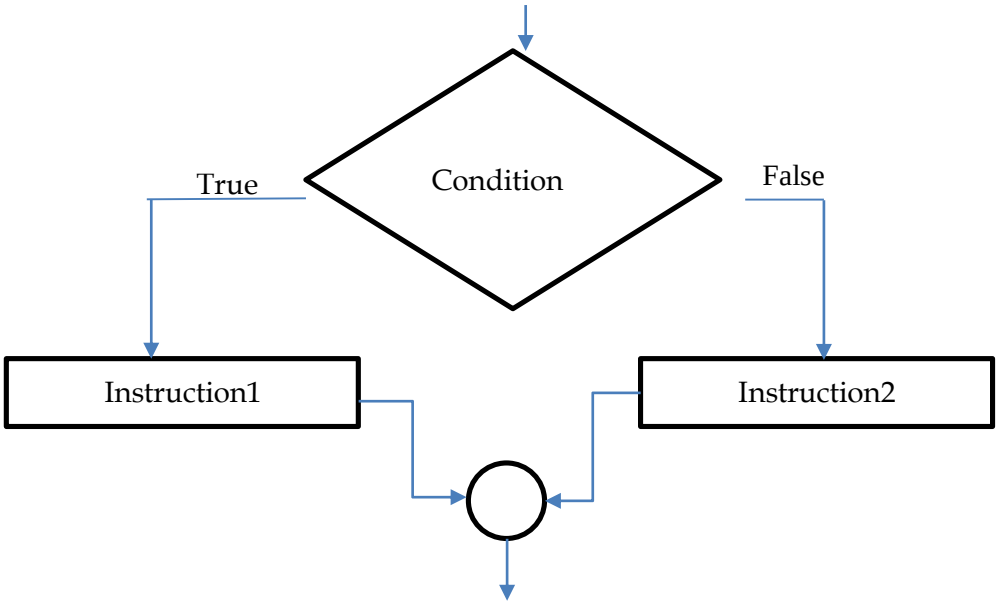
2. Selection:

It selects a statement to execute on the basis of condition. Statement is executed when the condition is true and ignored when it is false. Eg. if, if else, switch structures.

The most important Selection is the **if… else** statement, which chooses between two alternatives.

Syntax:

if (condition)
instruction1
else
instruction2



Example:

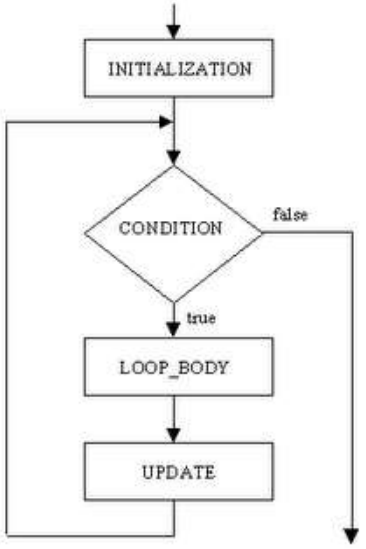
```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int n,r;
printf("enter a number");
scanf("%d",&n);
r=n%2;
if(r==0)
{
printf("\n%d is even number",n);
}
else
{
printf("\n%d is odd number",n);
}
```

getch();
}

3.Repetition/Looping:

In this structure the statements are executed more than one time. It is also known as iteration or loop.Eg. while loop, for loop,do-while loops etc.

Flowchart



For Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=1;
clrscr();
while(a<=5)
{
printf("I Love Nepal\n");
a++;
} //end of while
getch();
} //end of main
```

Decisions (if, if – else if, switch, ?; operator)

There are two types of decision (selection) control structures;

a) **conditional statement**
b) switch-case statement
a) **Conditional statement**

A statement that is executed only when a certain condition within a program has been met is called conditional expressions. It is used for mainly decision making.

C has different decision making instructions. They are as follows:

i) if statement
ii) if else statement
iii) if… else if statement (multi-way condition of if statement)
iv) nested if else statement.

i) if statement

The simplest if structure involves a single executable statement. Execution of the statement occurs only if the condition is true.

Syntax:

```
if (condition)
{
statement;
```

}

Example:

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int marks;
clrscr();
{
printf("Enter your marks:");
scanf("%d",&marks);
if(marks >=50)
printf("CONGRATULATIONS...!!! you have passed.");
}
getch();
}
```

Limitation of if

The statement(s) are executed if the condition is true; if the condition is false nothing happens in other words we may say it is not the effective one. Instead of it we use if-else statements etc.

ii) if…else statement

In if-else statement if the condition is true, then the true statement(s), immediately following the if-statement are executed otherwise the false statement(s) are executed. The use of else basically allows an alternative set of statements to be executed if the condition is false.

Syntax:

```
if (condition)
{
statement(s);
}
else
{
statement(s);
}
```

Q. Write a program to check if the year is leap or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int year;
printf("Enter any year ");
scanf("%d",&year);
```

```
if(year%400==0 || (year%100!=0 && year%4==0))
printf("The given year is leap year.");
else
printf("The given year is not a leap year.");
getch();
}
```

Example: To find largest number among two number

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a,b;
printf("enter the value of a and b");
scanf("%d%d",&a,&b);
if (a>b)
printf("\n %d is greatest",a);
else
printf("\n %d is greatest",b);
getch();
}
```

iii) if ….else if statement

It can be used to choose one block of statements from many blocks of statements. The condition which is true only its block of statements is executed and remaining are skipped.

Syntax:

```
if (condition)
{
statement 1;
}
else if (condition)
{
statement 2;
}
else
{
statement N;
}
```

Example: Program to find whether the entered number is positive, negative or zero.

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

```
{
int n;
clrscr();
printf("Enter a number:");
scanf("%d",&n);
if(n>0)
printf("The number is positive.");
else if (n<0)
printf("The number is negative.");
else
printf("The number is zero.");
getch();
}
```

nested if

In nested-if statement if the first, if condition is true the control will enter inner if. If this is true the statement will execute otherwise control will come out of the inner if and the else statement will be executed.

Syntax:

```
if (condition)
(condition)
{
statement 1;
}
else
{
statement 2;
}
else
{
statement N;
}
```

Example: Program to find the smallest number among three numbers.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int a,b,c;
clrscr();
printf("Enter three number:");
scanf("%d %d %d,&a,&b,&c);
if (a<b)
{
```

```
        if (a<c)
                printf("%d is a smallest number.",a);
      else
         printf("%d is a samallest number.",c);
}
else
{
      if(b<c)
         printf("%d is smallest number.",b);
      else
printf("%d is smallest number.",c);
}
getch();
}
```

Q. Write a program to test whether the entered character is alpha digit or alpha character or special character on the basis of given condition

Character	**Message**
0-9	**Alpha digit**
a-z	**alpha character**
others	**special character**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        char ch;
        printf("enter any character");
        scanf("%c",&ch);
        printf("\n");
        if (ch>='1' && ch<='9')
        printf("Alpha digit");
        else if (ch>='a' && ch<='z')
        printf("Alpha character");
        else if (ch>='A' && ch<='Z')
        printf("Alpha character");
        else
        printf("Special character");
        getch();
}
```

Program to find the largest number among three numbers

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

```
void main()
{
        clrscr();
        int a,b,c;
        printf("enter the three numbers");
        scanf("%d%d%d",&a,&b,&c);
        if (a>b && a>c)
                printf("\n %d is greatest",a);
        else if (b>a && b>c)
                printf("\n %d is greatest",b);
        else (c>a && c>b)
                printf("\n %d is greatest",c);
        getch();
}
```

Q. Write a program to enter any three numbers and find out the middle number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        int a,b,c;
        printf("enter the three numbers");
        scanf("%d%d%d",&a,&b,&c);
        if ((a>b && a<c) || (a<b && a>c))
                printf("\n %d is middle number",a);
        else if ((b>a && b<c)||(b<a && b>c))
                printf("\n %d is middle number",b);
        else ((c>a && c<b)||(c<a && c>b))
                printf("\n %d is middle number",c);
        getch();
}
```

Q. A program to demonstrate if else if statement .

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        int a,b;
        printf("enter a");
        scanf("%d",&a);
        printf("\n enter b \n");
```

```
        scanf("%d",&b);
        if (a==b)
        printf("a is equal to b");
        else if(a>b)
        printf("a is greater than b”);
        else
        printf("a is less than b");
        getch();
}
```

Q. The marks obtained by a student in 7 different subjects are entered through the keyboard. The student gets a division as per the following rules.

Percentage greater or equal to 60	**First division**
Percentage between 45 and 59	**Second division**
Percentage between 35 and 44	**Third division**
Percentage less than 35	**Fail**
Marks less than 35 in a subject	**Fail**

Write a program using C language to process result of all students based on the specific state above.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int s1,s2,s3,s4,s5,s6,s7,total;
        float p;
        printf("enter marks in 7 subjects");
        scanf("%d%d%d%d%d%d%d",&s1,&s2,&s3,&s4,&s5,&s6,&s7);
        total = s1+s2+s3+s4+s5+s6+s7;
        if(s1<35 ||s2<35 ||s3<35 ||s4<35 ||s5<35 ||s6<35 ||s7<35)
        printf("Failed.");
        p=(float)total/7;
        printf("\n total = %d",total);
        printf("\n percentage = %f",p);
        if (p>=80)
        printf("Distinction");
        else if (p>=60)
        printf("First Division");
        else if (p>=45)
        printf("Second Division");
        else if (p>=35)
        printf("Third Division");
        else
        printf("Failed");
```

```
        getch();
}
```

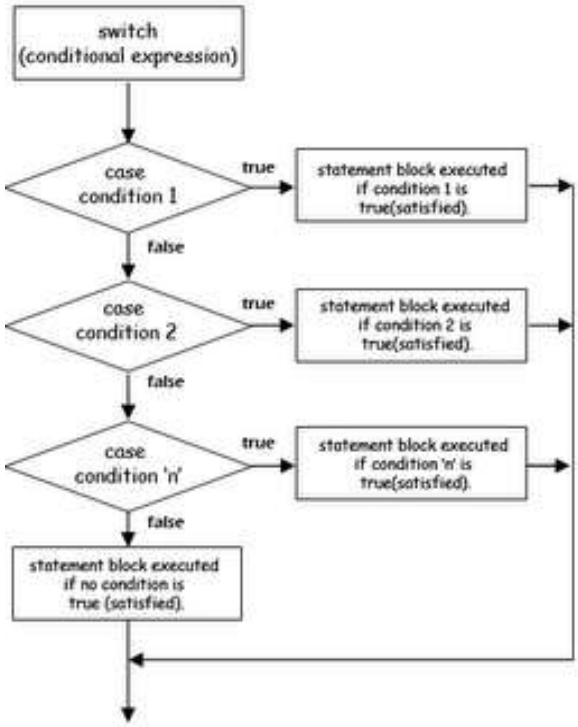
Q. Write a program to find the age group on the basis of age

Age	**group**
0-10	**child**
10-19	**teenage**
19-40	**young**
Above 40	**old**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        int age;
        printf("enter any age");
        scanf("%d",&age);
        if (age>0 && age<=10)
        printf("Child");
        else if (age>10 && age<=19)
        printf("Teenage");
        else if (age>19 && age<=40)
        printf("Young");
        else
        printf("Old");
        getch();
}
```

b) Switch-case statement

Switch statement is alternative of nested if…else.it is executed when there are many choices and only one is to be executed.



Syntax:

```
switch(expression)
{
case 1:
statement 1;
break;
case 2:
statement 2;
break;
.
.
.
.
```

case N:
statement N;
break;
default:
statement;
}
Example:

```
#include<stdio.h>
#include<conio.h>
void main ()
{
char c;
clrscr();
printf("Enter an alphabet:");
scanf("%c",&c);
switch(c)
{
case'a':
case'A':
printf("You entered vowel.");
break;
case'e':
case'E':
printf("You You entered vowel.");
break;
case'i':
case'I':
printf("You entered vowel.");
break;
case'o':
case'O':
printf("You entered vowel.");
break;
case'u':
case:'U'
printf("You entered vowel.");
break;
default:
printf("You entered a consonant.");
}
getch();
}
```

Q. A program to display the name of the day in a week depending on the number, entered through the keyboard using the switch-case statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        int day;
        printf("Enter number any number from 1 to 7 \n");
        scanf("%d",&day);
        switch(day)
        {
                case 1:
                printf("Sunday");
                break;
                case 2:
                printf("\n Monday");
                break;
                case 3:
                printf("\n Tuesday");
                break;
                case 4:
                printf("\n Wednesday");
                break;
                case 5:
                printf("\n Thursday");
                break;
                case 6:
                printf("\n Friday");
                break;
                case 7:
                printf("\n Saturday");
                break;
                default:
                printf("\n Invalid entry");
        }
        getch();
}
```

Q. Write a program that accepts a single character and give output using switch case.

Character	**Colour**
R or r	**Red**
B or b	**Blue**

W or w **White**
Otherwise **no more priority colour**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        char ch;
        printf("Enter your choice R,B,W\n");
        scanf("%c",&ch);
        printf("\n");
        switch(ch)
        {
                case 'r':
                case 'R':
                printf("Red");
                break;
                case 'b':
                case 'B':
                printf("Blue");
                break;
                case 'w':
                case 'W':
                printf("White");
                break;
                default:
                printf("No colour");
        }
        getch();
}
```

The following also can be used

```
switch (ch=toupper(getchar())
{
        case ‘R’:
        printf("Red");
        ……
}
```

Invalid switch case expression

a) The case labels must not be floating point number. They should always be integer constants or character constants.

```
switch(value)
{
case 10.11: /* Error */
```

```
break;
}
```
b) The case label must not be string expression
```
switch(value)
{
case “good”: /*Error*/
….
break;
}
```
c) The case label can’t be expression
```
switch(value)
{
case a+b:
….
break;
}
```
Q. Write a menu driven program to perform following: adding two numbers, difference, product, remainder when one number is divided by another. ‘OR’

Write a program that accepts two integer and an operator (+,-,*,/,%). And the program should calculate the value as directed by the operator and display the result (use switch case)

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        int a,b,c;
        char ch;
        printf("Enter two numbers");
        scanf("%d%d",&a,&b);
        printf("\n Enter your choice + - * / %");
        scanf("%c",&ch);
        printf("\n");
        switch(ch)
        {
                case '+':
                {
                        c=a+b;
                        break;
                }
                case '-':
                {
```

```
                    c=a-b;
                    break;
                }
                case '*':
                {
                    c=a*b;
                    break;
}
                case '/':
                {
                    c=a/b;
                    break;
                }
                case '%':
                {
                    c=a%b;
                    break;
                }
                default:
                printf("Invalid choice\n");
        }
        printf("Result %d",c);
        getch();
}
```

4.5.2 Looping (while, do while, for)

Looping

Loop is used to repeatedly execute set of statement. Structure that repeats a statement is known as iterative, repetitive or looping construct.

The computer has the ability to perform a set of instructions repeatedly. This involves repetition of some portion of a program either for a specified number of times or till a given condition is satisfied. This repetition operation is called looping.

Purpose:

- Execute statements for a specified number of times.
- To use a sequence of values.

Types of loop

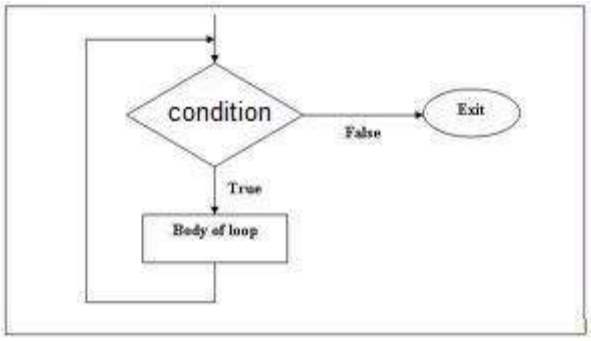
1. **while loop**
2. **do….. while loop**
3. **for loop**

a) while loop

It executes one or more statements while the given condition remains true. It is useful when number of iterations is unknown.The while loop is an entry control (pre test) loop.

Syntax

initialization;
while (condition)
{
statement;
increment/decrement;
}



Flow chart

Example

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

Example: Write a program to display number from 40 to 1 using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i=40;
        while(i>=1)
        {
                printf("%d\t",i);
                i--;
        }
        getch();
}
```

Q. Write a program to demonstrate the entry test nature of while loop.

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

```
{
	int x=25,y=55;
	while(x>y)
	{
		printf("See if this statement is executed ??");
	}
	getch();
}
```

Q. Write a program to find sum of first n natural number using while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int n,n1,s=0;
	printf("Enter the value of n ");
	scanf("%d",&n);
	n1=n;
	while(n>=1)
	{
		s=s+n;
		n=n-1;
	}
	printf("The sum upto %d = %d",n1,s);
	getch();
}
```

Q. Write a program to find the sum of odd and even numbers using while loop

Finding sum of odd and even number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
/*Sum of even and odd numbers from 1 to 50*/
	int n=1,s_evn=0,s_odd=0;
	while(n<=50)
	{
		if(n%2==0)
		s_evn+=n;
		else
		s_odd+=n;
		n++;
	}
```

```
        printf("\nThe sum even numbers upto 50 = %d",s_evn);
        printf("\nThe sum odd numbers upto 50 = %d",s_odd);
        getch();
}
```

Q. Write a program to Find factorial of a number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
/*Sum of even and odd numbers from 1 to 50*/
        int n,fact=1;
        printf("Enter any number ");
        scanf("%d",&n);
        while(n>=1)
        {
                fact*=n;
                n--;
        }
        printf("\nThe factorial of given number = %d",fact);
        getch();
}
```

Q. Display the given series using while loop.

Finding Fibonacci series

0 1 1 2 3 5 8 ….. upto 10 terms

```
#include<stdio.h>
#include<conio.h>
void main()
{
/*0 1 1 2 3 5 8 upto 10 terms*/
        int a=0,b=1,i=1,c;
printf(%d\t%d\t”,a,b);
        while(i<=8)
        {
c=a+ b;
        printf("%d\t",c);
        a=b;
        b=c;
        i++;
        }
        getch();
}
```

Q. Write a program to Reverse number which is input by the user.

```
#include<stdio.h>
#include<conio.h>
void main()
{
/*Reversing given number*/
        int n,r=0,c;
        printf("Enter any number ");
        scanf("%d",&n);
        while(n!=0)
        {
                c=n%10;
                r=r*10+c;
                n=n/10;
        }
        printf("\nThe reverse of given number = %d",r);
        getch();
}
```

Q. Write a program to check Armstrong number. The number is entered by the user.

```
#include<stdio.h>
#include<conio.h>
void main()
{
/*Checking for arm strong*/
        int n,s=0,c,n1;
        printf("Enter any number ");
        scanf("%d",&n);
        n1=n;
        while(n!=0)
        {
                c=n%10;
                s=s+c*c*c;
                n=n/10;
        }
        if(s==n1)
        printf("\n%d is Armstrong number.",n1);
        else
        printf("\n%d is not Armstrong number.",n1);
        getch();
}
```

ii) do….. while loop

Do while loops are useful where loop is to be executed at least once. In do while loop condition comes after the body of loop. This loop executes one or more statements while the given condition is true. It is an exit control (post test) loop.

Syntax:

initialization;

do
{
statement(s);
increment/decrement;
}
while (condition);

Example

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int a;
a=1;
clrscr();
do
{
printf("\n %d",a);
a++;
} while (a<=5);
getch();
}
```

Example: Write a program to display from 40 to 1 using do while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i=40;
        do
        {
                printf("%d\t",i);
                i--;
        } while(i>=1);
        getch();
}
```

Q. Write a program to find sum of first n natural number using do-while.

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

```
void main()
{
        int n,n1,s=0;
        printf("Enter the value of n ");
        scanf("%d",&n);
        n1=n;
        do
        {
                s=s+n;
                n=n-1;
        }while(n>=1);
        printf("The sum upto %d = %d",n1,s);
        getch();
}
```

iii) **for loop:**

Which is probably the most popular looping constant. The for allows us to specify three things about a loop in a single time. For loops are used when the number of iterations is known before entering the loop. It is also known as counter-controlled loop.

i) setting a loop counter to an initial values

ii)testing the loop counter to determine whether its value has reached the number of repetition desired.

ii)increasing,decreasing the values of loop counter each time the program statement within the loop has been executed.

Syntax:

```
for (initialization; condition; increment/decrement)
{
statement block;
}
```

Flowchart

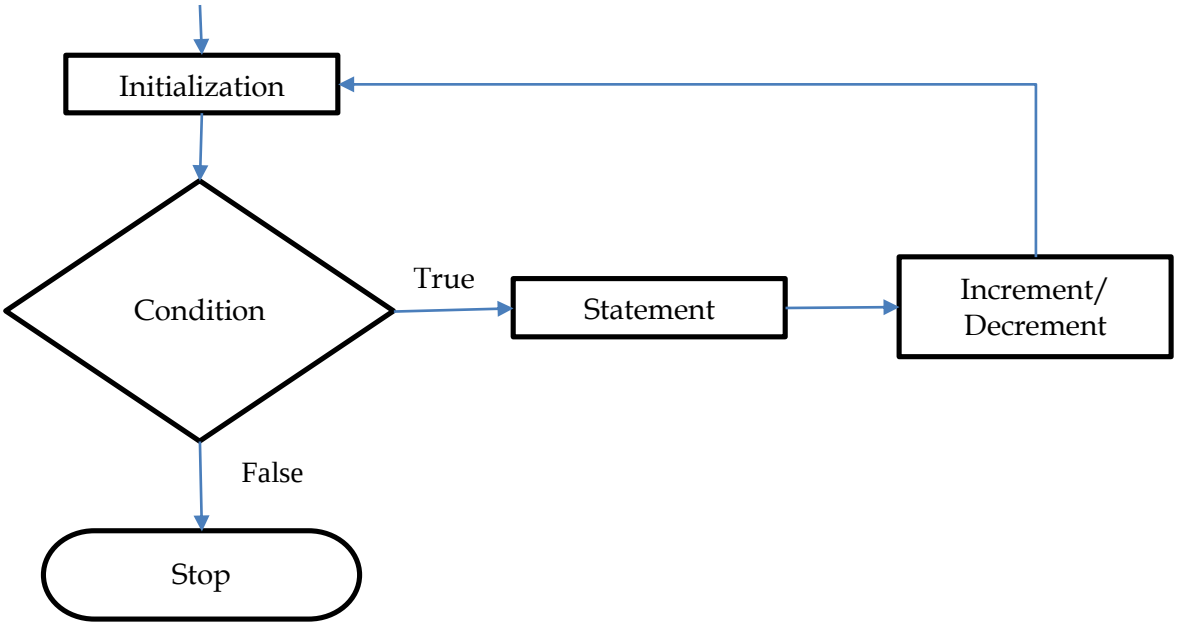


Fig.

Example

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

Q. Write a program to print 1 to 40 using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i;
        for(i=1;i<=40;i++)
        {
                printf("%d\t",i);
        }
        getch();
}
```

Q. Write a program to find sum of first n natural number using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int i,n,s=0;
	printf("Enter the value of n ");
	scanf("%d",&n);
	printf("\n");
	for(i=1;i<=n;i++)
	{
		s=s+i;
	}
	printf("The sum upto %d is %d.",n,s);
	getch();
}
```

Q. Write a program to find the sum of 1 to 100 number using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int i,n=100,s=0;
	for(i=1;i<=n;i++)
	{
		s=s+i;
	}
	printf("The sum upto %d is %d.",n,s);
	getch();
}
```

Q. Write a program to print all the ASCII code for alphabet A to Z using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
	char ch;
	for(ch='A';ch<='Z';ch++)
	{
		printf("%c = %d\t",ch,ch);
	}
	getch();
}
```

Q. Write a program to read two number n1 and n2. Display all even numbers between n1 and n2.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,n1,n2;
        printf("Enter the value of n1 and n2 ");
        scanf("%d%d",&n1,&n2);
        for(i=n1;i<=n2;i++)
        {
                if(i%2==0)
                printf("%d\t",i);
        }
        getch();
}
```

Q. Write a program to find the greatest number among 'n' integer entered from the keyboard.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,max=0,no,n;
        printf("How many numbers? ");
        scanf("%d",&no);
        for(i=1;i<=no;i++)
        {
                printf("\nEnter any number");
                scanf("%d",&n);
                if (n>max)
                {
                        max=n;
                }
        }
        printf("The largest number = %d",max);
        getch();
}
```

Q. Write a program to display Multiplication Table using for loop.

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

```
	int i,n;
	printf("Enter number ");
	scanf("%d",&n);
	for(i=1;i<=10;i++)
	{
		printf("\n% d*%d \t = \t %d",n,i,n*i);
	}
	getch();
}
```

Q. Write a program to find the sum of odd and even number for 1 to 50.

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int i,s_odd=0,s_even=0;
	for(i=1;i<=50;i++)
	{
		if(i%2==0)
		s_even=s_even+i;
		else
		s_odd=s_odd+i;
	}
	printf("\n The sum of odd numbers = %d ",s_odd);
	printf("\n The sum of even numbers = %d ",s_even);
	getch();
}
```

Q. Write a program to find factorial of a given number

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int i,n,fact=1;
	printf("Enter any number ");
	scanf("%d",&n);
	for(i=n;i>=1;i--)
	{
		fact=fact*i;
	}
	printf("\n The factorial of %d = %d ",n,fact);
	getch();
}
```

Q. Write a program to display the following series using for loop.

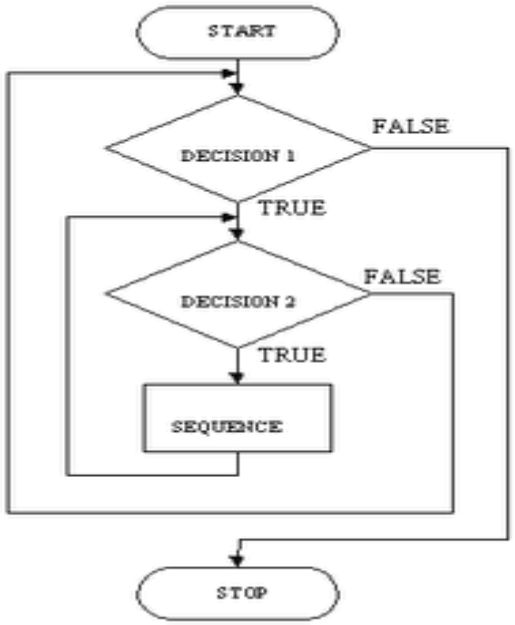
Write a program to display Fibonacci series upto 10th terms.

0 1 1 2 3 5 8 13 …. upto 10th terms

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a=0,b=1,c,i;
printf(“%d\t%d\t”,a,b);
        for(i=1;i<=8;i++)
        {
       c=a+b;
                printf("%d\t",c);
                a=b;
                b=c;
        }
        getch();
}
```

Q. Write a program to read ‘n’ number, find sum of positive and sum of negative numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,s_pos=0,s_neg=0,n,no;
        printf("How many numbers ");
        scanf("%d",&no);
        for(i=1;i<=no;i++)
        {
                printf("\n Enter number ");
                scanf("%d",&n);
                if(n>0)
                s_pos=s_pos+n;
                else
                s_neg=s_neg+n;
        }
        printf("\n The sum of positive numbers = %d ",s_pos);
        printf("\n The sum of negative numbers = %d ",s_neg);
        getch();
}
```



Nested loop

A loop within a loop is called nested loop. In this the outer loop is used for counting rows and the internal loop is used for counting columns.Any loop can be used as inner loop of another loop.

Syntax:

```
for (initialization;condition;increment/decrement)
{
  for(initialization;condition,increment/decrement)
  {
    statements(s);
    }
}
```

Flow chart

Example

```
#include<stdio.h>
#include<conio.h>
void main ()
{
clrscr();
for(int a=1;a<=3;a++)
{
for(int s=1;s<=3;s++)
{
printf("*");
} //end of internal loop
printf("\n");
} //end of external loop
getch();
} //end of main
```

Q. WAP to display Prime number display between1 to 100.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b;
        for (a=1;a<=100;a++)
        {
                int p=0;
                for (b=1;b<=a;b++)
                {
                        if (a%b==0)
                        p=p+1;
```

```
                }
                if (p==2)
                printf("%4d",a);
        }
        getch();
}
```

Q.A program to input number and check whether it is prime or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b;
        printf("Enter any number ");
        scanf("%d",&a);
        int p=0;
        for (b=1;b<=a;b++)
        {
        if (a%b==0)
        p=p+1;
        }
        if (p==2)
        printf("\n %d is prime number.",a);
        else
        printf("\n %d is not prime number.",a);
        getch();
}
```

Q. A program to find sum of cubes of first 10 numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,s=0;
        for(i=1;i<=10;i++)
        s=s+i*i*i;
        printf("The sum of first 10 cubes = %d",s);
        getch();
}
```

Infinite loop:

A loop is said to be infinite if it never terminates. To come out from loop we have to press reset.

Example:

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

Loop Interruption:

To interrupt the normal way of program, there are two statements for the loop interruption;

i) break statement ii) continue statement

i) Break Statement

When break statement is passed in a program, remaining of the program is terminated and control is passed to out of the loop.

Syntax:

break;

statement;

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i;
        for(i=0;i<5;i++)
        {
                if(i==2)
                break;
                printf("%d ",i);
        }
        getch();
}
```

Output: 0 1

When value of i==2, the break is passed and control is passed out of the loop. As a result program is terminated.

Continue Statement:

Unlike break, it doesn't pass control to out of the loop but continues to the next value of loop variable. The condition which is made, with continue, it not executed, all other are executed.

Syntax:

continue;
statement;

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
for(i=0;i<5;i++)
{
if(i==2)
continue;
printf("%d ",i);
}
getch();
}
```

Output: 0 1 3 4

Here 2 is not printed because of continue statement.

goto statement

A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

NOTE − Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

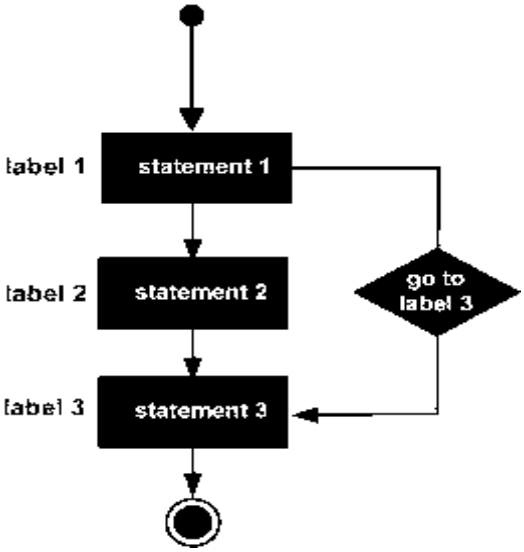


Fig: Flow Diagram

Syntax

The syntax for a **goto** statement in C is as follows -

goto label;

…..

…..

label: statement;

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement

Example:

```
#include  <stdio.h>
#include  <conio.h>
  void  main ()
      {
      int a = 10;
      LOOP:
```

```
      do
       {
        if( a == 15)
        {
           a = a + 1;
        goto LOOP;
        }
     printf("value of a: %d\n", a);
     a++;
      }
    while( a < 20 );
   getch();
}
```

Differences between for loop and while loop

for loop	**while loop**
It is definite loop.	It may not be definite.
The loop expression is within a block.	The loop expression is scattered throughout the program.
It uses keyword for.	It uses keyword while.
It contains three expressions.	It contains only one expression.

Syntax:
for
(initialization;condition;increment/decrement)
{
statement;
}
	Syntax
	Initialization;
	while (condition)
	{
	statement;
	increment/decrement;
	}

Example:
int n;
for (n=0;m<10;n++)
{
printf(“Nepal”);
}
	Example:
	int n=10;
	while(n!=0)
	{
	printf(“Nepal”);
	n--;
	}

Differences between while loop and do…while loop.

While	**do…..while**
entry control loop	exit control loop
condition is checked before loop execution	condition is checked at the end of loop
never execute loop if condition is false	executes false condition at least once since condition is checked later

there is no semicolon at the end of while statement	there is semicolon at the end of while statement.

Syntax
initialization;
while (condition)
{
statement;
increment/decrement;
}
	Syntax
	initialization;
	do
	{
	statement;
	increment/decrement;
	}
	while (condition);

Example:
int n=1;
while (n<10)
{
printf("Nepal");
n++;
}
	Example:
	int n=1;
	do
	{
	printf("Nepal");
	n++;
	}
	while (n<10);

Difference between break & continue

Break	Continue
Keyword is break	Keyword is continue
The loop expression is terminate after break.	The loop expression is not terminate after continue.
Syntax break; statement;	**Syntax** continue; statement;

Example:
int n;
for (n=0;m<10;n++)
{
if(n==2)
break;
printf("%d",n);
}
	Example:
	int n;
	for (n=0;m<10;n++)
	{
	if(n==2)
	continue;
	printf("%d",n);
	}

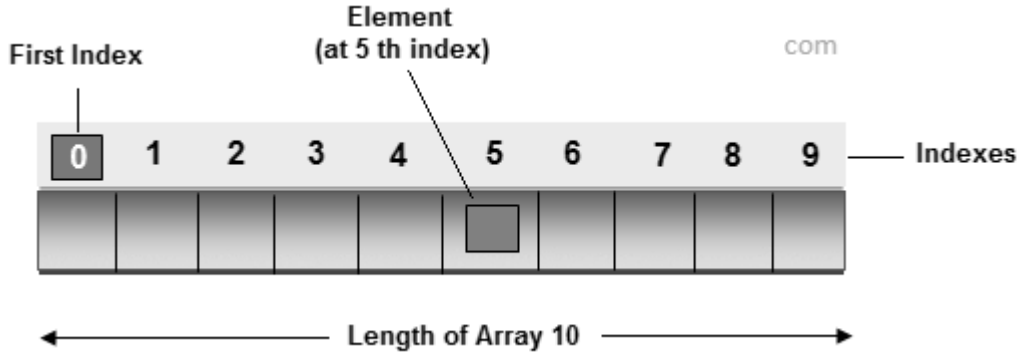
Output: 0 1	Output: 0 1 3 4 5 6 7 8 9

Array and String

Introduction to Array

An array is a collection of similar data type which is stored in consecutive memory location in a single variable. It is a derived data type in C, which is constructed from fundamental data type of C language. The individual values in an array are called elements. It contains int data type, all the data of the array must be int, if float then all the data must be float.

Each array element is referred by specifying the array name followed by one or more subscript with each subscript enclosed in square brackets. In C subscript starts at zero rather than one and each subscript must be expressed as a non-negative integer. Subscript is used to denote the size of array.



Advantage of array

- **Code Optimization:** Less code is required, one variable can store numbers of value.
- **Easy to traverse data:** By using array easily retrieve the data of array.
- **Easy to sort data:** Easily short the data using swapping technique
- **Random Access:** With the help of array index you can randomly access any elements from array.

Disadvantage of array

Fixed Size: Whatever size, we define at the time of declaration of array, we can not change their size, if you need more memory in that time we can not increase memory size, and if you need less memory in that case also wastage of memory.

Declaring Array

To declare an array in C you need to declare datatype and size of an array.

Syntax
datatype arrayName[SIZE];

Example

int roll_no[10];

Initializing Array

Initializing is a process to initialize the value in array variable. This is happen in two ways, initialize array one by one or all elements are initializing once.

Initialization of array one by one

```
int arr[5];
arr[0]=10;
arr[1]=20;
arr[2]=30;
 arr[3]=40;
```

arr[4]=50;

Initialization of array at once

int arr[]={10,20,30,40,50};

Accessing Array Elements

We can access array elements with the help of index value of element.

Example

int arr[]={10,20,30,40,50};

arr[3] // here 3 is index value and it return 40

Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, marks[]={80, 62, 70, 90, 98}; //declaration and initialization of array
clrscr();
//traversal of array
for(i=0;i<5;i++)
{
printf("%d",marks[i]);
}
getch();
}
```

output

80 62 70 90 98

Types of Array

There are two types array:

1) One dimensional array:

It consists only one rows or columns.

Syntax:

We must define the type of array, name and dimension of the array.

data type array name[size];

Example: int marks[200];

char name[20];

marks is a 200 element integer array.

name is a 20 element character array.

2) Two-dimensional array:

- In 2-dimentional elements are arranged in row and column format.
- When we are working with 2-dimentional array we require to refer 2-subscript operator which indicates row and column sizes.
- The main memory of 2-dimentional array is rows and sub-memory is columns.

- On 2-dimentional array when we are referring one-subscript operator then it gives row address, 2-subscript operator will gives element.
- On 2-dimentional array, array Name always gives main memory that is 1st row base address, arrayName will gives next row base address.

Syntax:

datatype arrayName[SIZE][SIZE];

Examples:

int matrix[3][2];
char name[20][10];

Important points related to array

Always size of the array must be an unsigned integer value which is greater than '0' only. In declaration of the array size must be required to mention, if size is not mention then compiler will give an error.

Example

```
int arr[];          //error
```

In declaration of the array size must be assigned type which value is greater than 0. In initialization of the array if specific number of values are not initialized it then rest of all elements will be initialized with it '0'.

Example

```
int arr[5]={10,20};       // yes valid
arr[0]=10;
arr[1]=20;
arr[2], arr[3], arr[4];   // initialized with zero
```

In initialization of the array mentioning the size is optional, in this case how many elements are initialize it that many variable are created.

Example

```
int arr[]={10,20,30,40,50};       // valid
Size=5;
Sizeof(arr)=10 byte
```

Important points for Array

- In implementation when we required 'n' number of variables of same data type then go for an array.
- When we are working with arrays always memory will created in contigeous memory location, so randomly we can access the data.
- In arrays all elements will share same name with unique identification value called index.
- Always array index will start with '0' and end with 'size-1'.

- When we are working with array compile time memory management will occur that is static memory allocation.

Q. Differentiate between one dimensional and two dimensional array:

One dimensional array	**Two dimensional array**
Only one subscript is required .	Two subscript is required.
It contains only rows or columns	It contains both rows or columns
Syntax: data type array name[size];	Syntax: data-type array name [expression1] [expression2];
Example: int marks[200]; char name[20];	Examples: int matrix[3][2]; char name[20][10];
Only one loop is used to create or retrieve array elements	two loop is used to create or retrieve array elements.

Example:
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char name[22];
int roll;
printf("enter name and roll");
scanf("%s%d",name,&roll);
printf("\nname=%s\nroll=%d",name,roll);
getch();
}
	Example:
	#include<stdio.h>
	#include<conio.h>
	void main()
	{
	clrscr();
	int mat[2][3];
	int i,j;
	printf("making matrix\n");
	for(i=0;i<2;i++)
	{
	for(j=0;j<3;j++)
	{
	printf("enter elements");
	scanf("%d",&
	mat[i][j]);
	}
	}
	printf("\n displaying matrix 2 by 3\n");
	for(i=0;i<2;i++)
	{
	for(j=0;j<3;j++)
	{
	printf("%d\t",mat[i][j]);
	}
	printf("\n");
	}
	getch();
	}

WAP to calculate total and average salary of 40 persons using array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
float sal[40];
float sum=0,avg;
int i;
for(i=0;i<40;i++)
{
printf("enter salary");
scanf("%f",&sal[i]);
}
for(i=0;i<40;i++)
{
sum=sum+sal[i];
}
avg=sum/40;
printf("sum=%f\n",sum);
printf("avg=%f",avg);
getch();
}
```

WAP to input 'n' numbers and find out the greatest and smallest numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int num[100],i,n;
int large,small;
printf("enter how many number");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter number");
scanf("%d",&num[i]);
}
large=num[0];
small=num[0];
for(i=0;i<n;i++)
{
if(large<num[i])
```

```
large=num[i];
else if(small>num[i])
small=num[i];
}
printf("\n largest=%d\n",large);
printf("smalesl =%d",small);
getch();
}
```

Example: Write a program using C language to read age of 100 persons and count no. of persons in the age group between 20 and 30. Use for and continue statements.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int num[100],i,n;
int count=0;
printf("enter how many number");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter number");
scanf("%d",&num[i]);
}
for(i=0;i<n;i++)
{
if(num[i]>=20 &&num[i]<30)
{
count=count+1;
continue;
}
}
printf("age between 20 and 30 are =%d",count);
getch();
}
```

Write a program in C to do the following:

WAP to read 'n' students roll number and sort them in Ascending order.	**WAP to read 'n' students roll number And sort them in Descending order**
#include<stdio.h> #include<conio.h> void main() {	#include<stdio.h> #include<conio.h> void main() {

```
clrscr();
int num[100];
int temp,i,j,n;
printf("enter how many number");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter number");
scanf("%d",&num[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(num[i]>num[j])
{
temp=num[i];
num[i]=num[j];
num[j]=temp;
}
}
}
printf("the rollno isn ascending order\n");
for(i=0;i<n;i++)
{
printf("\n%d",num[i]);
}
getch();
}
```

```
clrscr();
int num[100];
int temp,i,j,n;
printf("enter how many number");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter number");
scanf("%d",&num[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(num[i]<num[j])
{
temp=num[i];
num[i]=num[j];
num[j]=temp;
}
}
}
printf("the rollno isn Descending order\n");
for(i=0;i<n;i++)
{
printf("\n%d",num[i]);
}
getch();
}
```

WAP to read the elements of the given two matrices of order 3X3 and to perform the matrix addition.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int A[3][3],B[3][3],C[3][3];
int i,j;
printf("entering the elements of matrices a\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("enter no of elemnts\n");
```

```
scanf("%d",&A[i][j]);
}
}
printf("entering the elements of matrices B \n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("enter no of elemnts\n");
scanf("%d",&B[i][j]);
}
}
printf("\n the sum of matrices are\n");

for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
C[i][j]=A[i][j]+B[i][j];
}
}
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",C[i][j]);
}
printf("\n");
}
getch();
}
```

Q. **WAP to input 3X2 matrices and display in TRANSPOSE form that is in 2X3 order.**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
clrscr();
int A[3][2];
int i,j;
printf("entering the elements of matrices A\n");
for(i=0;i<3;i++)
{
```

```
for(j=0;j<2;j++)
{
printf("enter no of elemnts\n");
scanf("%d",&A[i][j]);
}
}
printf("\n displaying in Transpose matrix \n");
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",A[j][i]);
}
printf("\n");
}
getch();
}
```

String Function

String *is a collection of character or group of character, it is achieve in C language by using array character.* The string in C language is one-dimensional array of character which is terminated by a null character '\0'. In other words string is a collection of character which is enclose between double quotes (" ").

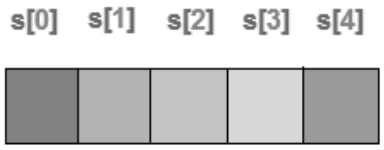
Note: Strings are always enclosed within double quotes. Whereas, character is enclosed within single quotes in C.

Declaration of string

Strings are declared in C in similar manner as arrays. Only difference is that, strings are of char type.

Example

```
char s[5];
```



Initializing Array string

String are initialize into various way in C language;

Example

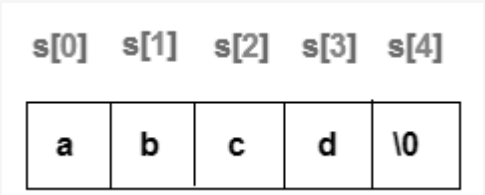
```
char str[]="abcd";
      OR
char str[5]="abcd";
      OR
char str[5]={'a','b','c','d','\0'};
```

OR

```
char str[]={'a','b','c','d','\0'};
```

OR

```
char str[5]={'a','b','c','d','\0'};
```



Reading String from user

Example

```
char str[5];
scanf("%s",str);
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[10];
printf("Enter name: ");
scanf("%s",name);
printf("Your name is: %s.",name);
getch();
}
```

Example of reading string

```
Enter name: Hitesh kumar
Your name is: Hitesh
```

Note: String variable str can only take only one word. It is because when white space is encountered, the scanf() function terminates. to over come this problem you can use gets() function.

Syntax

```
char str[5];
gets(str);
```

gets()

gets() are used to get input as a string from keyword, using gets() we can input more than one word at a time.

puts()

puts() are used to print output on screen, generally puts() function are used with gets() function.

Example of String program

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[10];
```

```
printf("Enter any string: ");
gets(str);
printf("String are: ");
puts(str);
getch();
}
```

Explanation: Here gets() function are used for input string and puts() function are used to show string on console or monitor.

Output

Enter String: hello word
String are: hello word

Q. What do you mean by character array(string)? What do you mean by null character?

We can represent a character easily with char data type. A group of character can be stored in a character array. A string is a group of any length. It can be used to manipulate text such as words and sentences.

Example: char s[ ]=''programming'';

Char s[ ]='p','r','o','g','r','a','m','m','i','n','g','\o';

Char s[13]=''programming'';

The end of character is marked by null character '\o'.

It looks two character, but it is actually only one character with the \ indicating that what follows it is something special. '\o' and '0' are not same.

ASCII value of '\o' is 0; and ASCII value of 0 is 48. In the declaration, '\o' is not necessary. C inserts the null character automatically.

String Manipulation Function

C has several built(library) function, such as :

strlrupr(),strcat(), strlwr(), strrev(), strlen(), strcmp(), strcpy() etc. for manipulating strings. To use this header file "string.h" must be included.

a)String length i.e. strlen():

The **strlen()** function returns the length of given string (including spaces).

Syntax: strlen(string);

strlen(s1) Returns the length of string s1.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string[30];
```

```
int l;
printf("enter word or sentences\n");
gets(string);
l=strlen(string);
printf("the length is %d",l);
getch();
}
```

b) String Concatenation i.e. strcat(): The string function **strcat()** is used to concatenate(join/combine) two strings resulting to a single string.

Syntax: strcat(string1,string2);

strcat(s1, s2); Concatenates string s2 onto the end of string s1.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string1[30],string2[22];
printf("enter first string\n");
gets(string1);
printf("entersecond string\n");
gets(string2);
strcat(string1,string2);
printf("joins atrings are %s \n",string1);
getch();
}
```

c) String Copy i.e. strcpy(): The string function **strcpy()** is used to copy content of a string to another string. it takes two arguments.

Syntax: strcpy(string1,string2);

strcpy(s1, s2); Copies string s2 into string s1.

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string1[30],string2[22];
printf("enter first string\n");
gets(string1);
```

```
strcpy(string2,string1);
printf("copied string is  %s",string2);
getch();
}
```

d) String Reverse i.e. strrev()

The string function **strrev()** is used to reverse a string. It takes only one string and returns in reverse order.

Syntax:strrev(string);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string1[30];
printf("enter first string\n");
gets(string1);
strrev(string1);
printf("reversestringis\n%s",string1);
getch();
}
```

e) String Uppercase i.e. strupr()

The string function **strupr()** is to convert the lower case into upper case. It takes only one string.

Syntax:strupr(string);

Examples:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string1[30];
printf("enter word\n");
gets(string1);
strupr(string1);
printf("uppercase is\n%s",string1);
getch();
}
```

e) String Lowercase i.e. strlwr()

The string function **strlwr()** is to convert the upper case into lower case. It takes only one string.

Syntax:strlwr(string);

Examples:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string1[30];
printf("enter word\n");
gets(string1);
strlwr(string1);
printf("lowercase is \n%s",string1);
getch();
}
```

g) String Comparision i.e. strcmp()

The string function **strcmp()** compares two string character by character. It accepts two string as parameters and returns an integer whose value is

< 0 if the first string is smaller than second string.
> 0 if the first string is greater than second string.
==0<if the first string equal to second string.

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string1[30],string2[22];
printf("enter first string\n");
gets(string1);
printf("enter second string\n");
gets(string2);
if(strcmp(string1,string2)>0)
printf("greater string %s",string1);
else if(strcmp(string1,string2)==0)
printf("both string are equal");
else
printf("greater string %s",string2);
```

```
getch();
}
```

Q. WAP to input a word and determine whether it is palindrome or not.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string1[30],string2[33];
printf("enter word\n");
gets(string1);
strcpy(string2,string1);
if(strcmp(string2,strrev(string1))==0)
printf("is pallindrome");
else
printf("is not pallindrome");
getch();
}
```

Q. Write a program to read a line and to convert it into uppercase.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char string1[30];
printf("enter word\n");
gets(string1);
strupr(string1);
printf("uppercaseis\n%s",string1);
getch();
}
```

Q. Write a program to print the 10 positive integers and their factorials.

```
 #include<stdio.h>
#include<conio.h>
void main()
{
int i,j,n[10],fact;
for(j=0;j<=9;j++)
{
```

```
printf("\nEnter any number ");
scanf("%d",&n[j]);
printf("\n");
fact=1;
for(i=n[j];i>=1;i--)
{
fact=fact*i;
}
printf("\n The factorial of %d = %d ",n[j],fact);
}
getch();
}
```

Q. Write a C program to read salaries of 200 employees and count the number of employees getting salary between 5000 to 10000.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int j,i[200],count=0;
for(j=0;j<=199;j++)
{
printf("\n Enter the salary of employee ");
scanf("%d",&i[j]);
if(i[j]>=5000 && i[j]<=10000)
count++;
}
printf("\n Total employee having salary between 5000 and 1000 are %d.",count);
getch();
}
```

Q. Write programs in C to do the following:

Question: WAP to input a 'n' name of students and sort them in Alphabetical (Ascending) order.
Ans:
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char name[25][20];
char temp[25];
	Question: WAP to input a 'n' name of students and sort them in Descending order.
	Ans:
	#include<stdio.h>
	#include<conio.h>
	#include<string.h>
	void main()
	{
	clrscr();
	char name[25][20];
	char temp[25];
	int i,j,n;

```
int i,j,n;
printf("enter how many students");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter name of students\n");
scanf("%s",name[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(strcmp(name[i],name[j])>0)
{
strcpy(temp,name[i]);
strcpy(name[i],name[j]);
strcpy(name[j],temp);
}
}
}
printf("sorted name in alphabetical \n");
for(i=0;i<n;i++)
{
printf("\n%s",name[i]);
}
getch();
}
```

```
printf("enter how many students");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter name of students\n");
scanf("%s",name[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(strcmp(name[i],name[j])<0)
{
strcpy(temp,name[i]);
strcpy(name[i],name[j]);
strcpy(name[j],temp);
}
}
}
printf("sorted name in descending \n");
for(i=0;i<n;i++)
{
printf("\n%s",name[i]);
}
getch();
}
```

Q. Write a program to store mark obtained by 'n' students and count the number of students who obtained mark greater than 70. Also count the number of students who failed (mark M<35).

```
#include<stdio.h>
#include<conio.h>
void main()
{
int count1=0,count2=0,i,n;
printf("How many students? ");
scanf("%d",&n);
float mark[n];
for(i=0;i<n;i++)
{
printf("\n Enter mark of student %d ",i+1);
scanf("%f",&mark[i]);
if(mark[i]>70)
count1++;
```

```
if(mark[i]<35)
count2++;
}
printf("\nNo. of students scoring above 70 = %d",count1);
printf("\nNo. of students who failed = %d",count2);
getch();
}
```

Functions

Concept of function, function definition, function prototype

A **function** is a group of statements that together perform a specific task. Function is a block or part of program. A function definition specifies the name of the function, the types and number of parameters, it expects to receive, and its return type, when any program is very long or same code is repeating many times then we try to cut the program in different parts(or blocks) so that whole program became more understandable, easier to debug(error checking) and size of code will be easier. A repeated group of instructions in a program can be organized as a function. Every C program has at least one function, which is main().

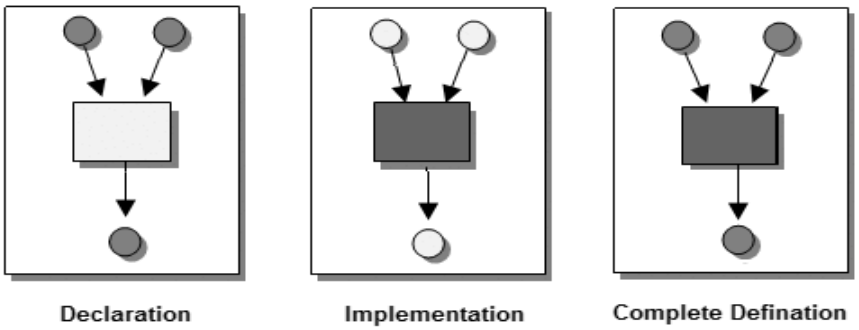


Fig: Function Declaration

Why use function?

Functions are used for divide a large code into module, due to this we can easily debug and maintain the code. For example if we write a calculator programs at that time we can write every logic in a separate function (For addition sum(), for subtraction sub()). Any function can be called many times.

Advantage of Function

- Code Re-usability.
- Develop an application in module format.
- Easily to debug the program.
- Code optimization: No need to write lot of code.
- Code is easier to read.
- Program testing becomes easy.
- Compiled executable is smaller.

- You don't have to make comments if function name make sense.
- Program can be developed in short period of time.
- There is no chance of code duplication.

Defining a function

Defining of function is nothing but give body of function that means write logic inside function body.

Syntax:

return_type_function(parameter)
{
function body;
}

Return type: A function may return a value. The return type is the data type of the value the function returns. Return type parameters and returns statement are optional.

Function name: Function name is the name of function it is decided by programmer or you.

Parameters: This is a value which is pass in function at the time of calling of function A parameter is like a placeholder. It is optional.

Function body: Function body is the collection of statements.

Function Declarations:

A function declaration is the process that tells the compiler about a function name. The actual body of the function can be defined separately.

Syntax:

return_type function_name();

Note: At the time of function declaration function must be terminated with **;**.

Calling a function

When we call any function control goes to function body and execute entire code. For call any function just write name of function and if any parameter is required then pass parameter.

Syntax:

function_name();
or
Variable=function_name(argument);

Note: At the time of function calling function must be terminated with ';'.

Example of Function

```
#include<stdio.h>
#include<conio.h>
void sum();// declaring a function
void  main()
{
```

```
clrscr();
int a=10,b=20, c;
sum();// calling function
getch();
}
void sum()  // defining function
{
c=a+b;
printf("Sum: %d", c);
}
```

Output

Sum: 30

Return and Void statement of a function

Function return: return type is a type of value returned by function. It has two type:

i) function that does not have a return type i.e. void.
ii) function that does have a return value.i.e. return().

Example:

```
/* addition of two numbers*/
#include<stdio.h>
#include<conio.h>
int add(int,int);
void main()
{
clrscr();
int a,b,c;
printf("eneter two numbers");
 scanf("%d%d",&a,&b);
 c=add(a,b);
 printf("\n the sum is %d",c);
 getch();
 }
 int add(int a, int b)
 {
 int y;
 y=a+b;
 return(y);
 }
```

Q. **/*write a program to find the sum of 'n' integers numbers using function.*/**

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

```
void main()
{
clrscr();
int n,c;
printf("enter any number");
scanf("%d",&n);
c=sum(n);
printf("sum=%d",c);
getch();
}
int sum(int n)
{
int sum=0;
for(int i=1;i<=n;i++)
{
sum=sum+i;
}
return(sum);
}
```

Q. Write a program in C to find the factorial of a number using function.

```
#include<stdio.h>
#include<conio.h>
int fun(int n);
void main()
{
clrscr();
int n,x;
printf("Enter any number");
scanf("%d",&n);
x=fun(n);
printf("The factorial of %d is %d.",n,x);
getch();
}
 int fun(int n)
{
int fact=1;
while(n!=0)
{
fact=fact*n;
n=n-1;
}
return (fact);
}
```

Q. write a program using user defined function to calculate y raise to power x.

```
#include<stdio.h>
#include<conio.h>
int power(int,int);
void main()
{
clrscr();
int a,b,value;
printf("enter a and b");
scanf("%d%d",&a,&b);
value=power(a,b);
printf("the value of a=%d raised to b=%d is %d",a,b,value);
getch();
}
int power(int a, int b)
{
int i, cal=a;
for(i=1;i<b;i++)
{
cal=cal*a;
}
return(cal);
}
```

Q. Write a program for Printing even numbers from 2 to 40.

```
#include<stdio.h>
#include<conio.h>
void printeven();
void main()
{
clrscr();
printf("even number from 2 to 40 are\n");
printeven();
getch();
}
void printeven()
{
int a;
for(a=2;a<=40;a=a+2)
{
printf("%d\n",a);
}
}
```

Q. A positive digit integers is entered through the keyboard:

I) WAP to calculate the sum of digit.

II) WAP to calculate the product of digit.

```
 #include<stdio.h>
#include<conio.h>
int sum(int);
void main()
{
clrscr();
int s,n;
printf("enter number\n");
scanf("%d",&n);
s=sum(n);
printf("the sum is %d",s);
getch();
}
int sum(int n)
{
int a,r=0;
while(n>0)
{
a=n%10;
r=r+a;
n=n/10;
}
return r;
}
```

For product in second module put

```
r =1;
r=r*a;
```

Q.Write a program to find the sum of squares of first 10 natural numbers using function.

```
#include<stdio.h>
#include<conio.h>
void sum();
void main()
{
clrscr();
printf("sum of first 10 natural number is \n");
sum();
getch();
}
void sum()
{
int a,sum=0;
for(a=1;a<=10;a++)
```

```
{
sum=sum+a*a;
}
printf("the sumis %d",sum);
}
```

Q. Write a program to count the positive and negative number from 'n' entered using function.

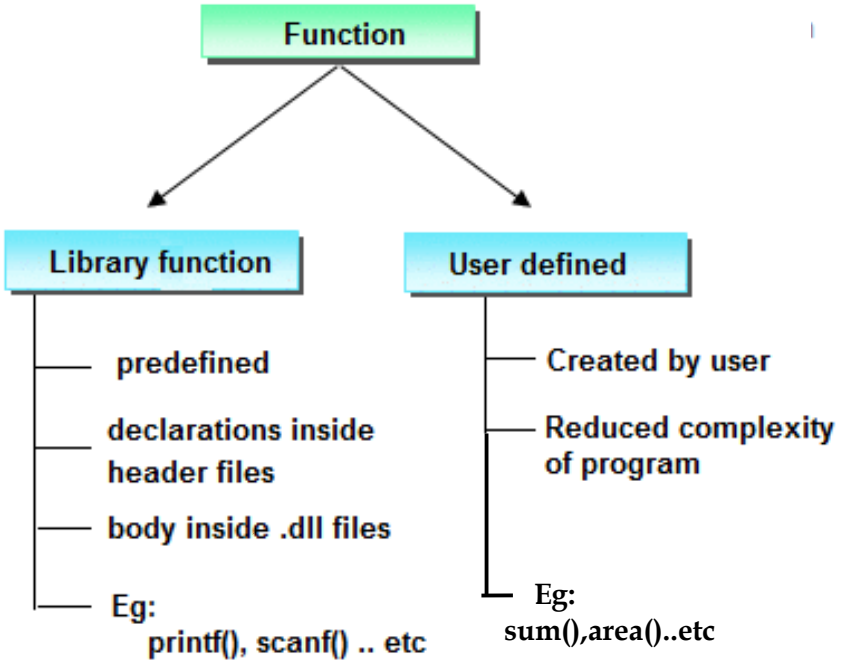
```
 #include<stdio.h>
#include<conio.h>
void number(int no[100], int n);
void main()
{
clrscr();
int n ,no[100],i;
printf("enter the value of n\n");
scanf("%d",&n);
for(i=1;i<n;i++)
{
printf("enter number=\n");
scanf("%d",&no[i]);
}
number(no,n);
getch();
}
void number(int no[100],int n)
{
int pco=0,nco=0;
for(int i=1;i<n;i++)
{
if(no[i]>0)
{
pco=pco+1;
}
else
{
nco=nco+1;
}
}
printf("\n the positive number is %d",pco);
printf("\n the negative number is %d",nco);
}
```

Type of Function

There are two type of function in C Language. They are;

- Library function or pre-define function.

- User defined function.



Library function

Library functions are those which are predefined in C compiler. The implementation part of pre-defined functions is available in library files that are .lib/.obj files. .lib or .obj files are contained pre-compiled code.printf(), scanf(), clrscr(), pow() etc. are pre-defined functions.

Limitations of Library function

- All predefined function are contained limited task only that is for what purpose function is designed for same purpose it should be used.
- As a programmer we do not having any controls on predefined function implementation part is there in machine readable format.
- In implementation whenever a predefined function is not supporting user requirement then go for user defined function.

User defined function

These functions are created by programmer according to their requirement for example suppose we want to create a function for add two number then you create a function with name **sum()** this type of function is called user defined function.

Q. Differentiate between library functions and user-defined functions.

Library function	**User-defined function**
It is predefined or built in function	It is defined or created by the user according to his need.
It is already created	We should have to create.
It is simple to use	It is usually complex to use.

Program development time will be faster	Program development time will be slower
It requires header file to use it	It requires function prototype to use it
Eg: printf(), scanf(), getch(), clrscsr()	Eg: sum(),area(),factorial(), fibonacci()

Classification of user-defined function

1) Function returning value and passing arguments.

```
#include<stdio.h>
#include<conio.h>
int add(int,int);
void main()
{
clrscr();
int a,b,c;
printf("enter two numbers");
scanf("%d%d",&a,&b);
c=add(a,b);
printf("the sum is %d",c);
getch();
}
int add(int a ,int b)
{
int sum;
sum=a+b;
return(sum);
}
```

2. Function returning no value but passing arguments.

```
#include<stdio.h>
#include<conio.h>
void add(int,int);
void main()
{
clrscr();
int a,b;
printf("enter two numbers");
scanf("%d%d",&a,&b);
add(a,b);
getch();
}
void add(int a ,int b)
{
int sum;
sum=a+b;
printf("the sum is %d",sum);
}
```

3. Function returning value and passing no arguments.

Ans:

```
#include<stdio.h>
#include<conio.h>
int add(void);
void main()
{
clrscr();
int c;
c=add();
printf("the sum is %d",c);
getch();
}
int add(void)
{
int sum,a,b;
printf("enter two numbers");
scanf("%d%d",&a,&b);
sum=a+b;
return(sum);
}
```

4. Function returning no value and passing no arguments.

```
#include<stdio.h>
#include<conio.h>
void add(void);
void main()
{
clrscr();
add();
getch();
}
void add(void)
{
int sum,a,b;
printf("enter two numbers");
scanf("%d%d",&a,&b);
sum=a+b;
printf("the sum is %d",sum);
}
```

Concept of Recursion

When Function is call within same function is called ***Recursion***. The function which call same function is called **recursive function**. In other word when a function call itself then that function is called **Recursive function**.

Recursive function are very useful to solve many mathematical problems like to calculate factorial of a number, generating Fibonacci series, etc.

Advantage of Recursion

- Function calling related information will be maintained by recursion.
- Stack evaluation will be take place by using recursion.
- In fix prefix, post-fix notation will be evaluated by using recursion.

Disadvantage of Recursion

- It is a very slow process due to stack overlapping.
- Recursive programs can create stack overflow.
- Recursive functions can create as loops.

Q. Calculation of factorial number using recursion:

```
#include<stdio.h>
#include<conio.h>
int factorial(int);
void main()
{
clrscr();
int n,c;
printf("enter any number");
scanf("%d",&n);
c=factorial(n);
printf("the factorial is %d",c);
getch();
}
int factorial(int n)
{
if(n<=1)
return(1);
else
return(n*factorial(n-1));
}
```

Q. WAP to find the sum of given non-negative integer number using recursive function.

Or

WAP to find sum of 'n' natural numbers using recursive function.

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

```
int n,result;
printf("enter how many number");
scanf("%d",&n);
result=sum(n);
printf("the sum of negative number is =%d",result);
getch();
}
int sum(int n)
{
int sum1=0;
if(n==0)
return sum1;
else
return(n+sum(n-1));
}
```

Structures and Unions

Definition and Difference between Structure and Union

Structure *is a user defined data type which hold or store homogeneous data item or element in a single variable*. It is a Combination of primitive and derived data type.

Why Use Structure in C

In C language array is also a user defined data type but array hold or store only similar type of data, If we want to store different-different type of data in, then we need to defined separate variable for each type of data.

Example: Suppose we want to store Student record, then we need to store....

- Student Name
- Roll number
- Class
- Address

For store Student name and Address we need character data type, for Roll number and class we need integer data type.

If we are using Array then we need to defined separate variable.

Example

```
char student_name[10], address[20];
int roll_no[5], class[5];
```

If we use Structure then we use single variable for all data.

Syntax

```
struct stu
{
char student_name[10];
char address[20];
```

int roll_no[5];
int class[5];
};

Note: Minimum size of Structure is one byte and Maximum size of Structure is sum of all members variable size.

Note: Empty Structure is not possible in C Language.

Defining a Structure

Syntax:

The general syntax of declaring a structure is:

struct tagname
{
Datatype1 member1;
Datatype2 member2;
Datatype3 member3;
.....
};

In this declaration struct is required keyword. It is important to note that these member are enclosed within the braces. At end of the structure creation (;) must be required because it indicates that an entity is constructed.

Example:

```
struct emp
{
int id;
char name[36];
int sal;
};
sizeof(struct emp)      // --> 40 byte (2byte+36byte+2byte)
```

Once the composition of the structure has been specified, the individual structure type variables can be declared as

struct user defined-name variable1,variable2,……variablen;

struct keyword, user-defined name→ structure name, var1,var2,var3 …. var n are structure variables of type user defined name.

so combined structure declaration is

struct user-defined –name
{
data-type member1;
data-type member2;
……..
……..
data-type member n;
};

struct user defined var1, var2,…… var n;

Example:

```
struct student
{
char name[20];
char sex;
int roll_no;
float marks;
};
struct student s1,s2,s3;
```

or

```
struct student
{
char name[20];
char sex;
int roll_no;
float marks;
} a,b;
```

In this case name of the structure i.e. student is optional.

Structure initialization:

The member of a structure variable can be assigned initial values is same as array.

struct user-defined name variable={value-1,value-2,value-3,…….,value n};
where value-1 refers to the first member.
value 2 refers to second member and so on.
The example given below:

```
struct student
{
char name[10];
char sex;
int roll_no;
float marks;
{;
struct student stud={''sanu'','f',2566.67};
```

The stud is structure variable of type student whose member are assigned initial values. name[20] is assigned raj sex m roll 25 marks 66.67

Processing or Accessing structure:

Member of structure can be accessed through the use of dot(.) operator.
i.e. variable.member;

For Example:

stud.roll_no=27;
and the statement is

```
printf(''%d'',stud.roll_no);
```

Example showing that name and roll number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
struct student
{
char name[20];
int roll_no;
};
struct student s1;
printf("enter name");
scanf("%s",s1.name);
printf("enter roll number");
scanf("%d",&s1.roll_no);
printf("\n displaying data");
printf("\n name=%s",s1.name);
printf("\n roll=%d",s1.roll_no);
getch();
}
```

Example of Structure

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
struct emp
{
int id;
char name[36];
float sal;
};
struct emp e;
printf("Enter employee Id, Name, Salary: ");
scanf("%d",&e.id);
scanf("%s",&e.name);
scanf("%f",&e.sal);
printf("Id: %d",e.id);
printf("\nName: %s",e.name);
printf("\nSalary: %f",e.sal);
```

```
getch();
}
```

Output

Output: Enter employee Id, Name, Salary: 5 Spidy 45000 Id : 05 Name: Spidy Salary: 45000.00

Array of structure:

A structure type of structure placed in a common variable name is called array of structure.
For example:

```
struct student
{
char name[205];
char sex;
int roll;
float marks;
};
struct student stu[50];
```

Here stu[50] is a structure variable;
it can be accommodate the structure of a student upto 50.Each record may be accessed and processed separately like individual elements of an array.
For example to print rollno of student 3

```
Printf(''%d'',stu[2].roll);
```

Because array starts at 0.

Initialization of array of structure:

A structure can be initialized in the same way as that of array data in c.

For example:

```
struct school
{
int roll_no;
char sex;
float weight;
};
struct school stu[2]=}
                {22,'m',33.33},
                {33,'f',55.55},
                };
```

In this example, stu[2] is an array of structure whose dimension is 2.

```
stu[0].roll_no=22;          stu[1].roll_no=33;
stu[0].sex='m';             stu[1].sex='f';
str[0].weight=33.33;        str[1].weight=55.55;
```

Q. A program to enter name, price and pages of a book and to display them.

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

```
#include<conio.h>
void main()
{
clrscr();
struct book
{
char name[20];
int pag;
float price;
};
struct book b[3];
int i;
for(i=0;i<3;i++)
{
printf("enter name pages and price of book");
scanf("%s%d%f",b[i].name,&b[i].pag,b[i].price);
}
printf("\n dispalying data");
for(i=0;i<3;i++)
{
printf("\nname=%s pages=%d",b[i].name,b[i].pag,b[i].price);
}
getch();
}
```

structure within structure(nested structure)

When a structure is declared as the member of another structure then it is called as a nested structure or structure within structure.

For Example:

```
struct date
{
int day;
int month;
int year;
};
struct student
{
char name[30];
int roll;
struct date dob;
};
struct student s;
```

First we have to write the name of the outer structure variable followed by a dot(.) operator and name of inner structure followed by dot(.) operator and the name of the field name of outer structure.

The same process goes on if there are more than two levels of nested structure.

Q: Write a program to print name, roll and date of birth by nested structure.

```
#include<stdio.h>
#include<conio.h>
void main()
{
struct date
{
int day;
int month;
int year;
};
struct student
{
char name[30];
int roll;
struct date d;
};
struct student s;
printf("enter name");
scanf("%s",s.name);
printf("enter roll");
scanf("%d",s.roll);
printf("enter date month year");
scanf("%d%d%d",&s.d.day,&s.d.month,&s.d.year);
printf("\n name=%s\nroll=%d\n",s.name,s.roll);
printf("date=%d\n month=%d\n year=%d\n",s.d.day,s.d.month,s.d.year);
getch();
}
```

Difference Between Array and Structure

	Array	**Structure**
1	Array is collection of homogeneous data.	Structure is the collection of heterogeneous data.
2	Array data are access using index.	Structure elements are access using dot(.)operator.
3	Array allocates static memory.	Structures allocate dynamic memory.

4	Array element access takes less time than structures.	Structure elements takes more time than Array.
5.	Each data item is called elements.	Each data item is called member
6.	We cannot have array of array	We can have array of structure.
7.	Syntax: data-type array name[size];	Syntax: struct user-definedname { Data-type member1 Data-type member2 }var;

Union

A union is quite similar to the structures in C. It also store different data types in the same memory location. It is also a user defined data type same like structure. It is more near to structure. Union can be defined in same manner as structures, for defining union use **union** keyword where as for defining structure use struct keyword.

union and structure are almost same

Structure	Union
struct student { **int** roll; **char** name[10]; **float** marks; }u;	**union** student { **int** roll; **char** name[10]; **float** marks; }u;

Syntax

union tagname
{
datatype member1;
datatype member2;
.....
.....
};

Example of Union

union emp
{
int ID;
char name[10];
double salary;

}u;

Accessing members of an union

The member of unions can be accessed in similar manner as Structure with union reference. Suppose, we you want to access name variable in above example, it can be accessed as u.name.

Advantage of union over structure

It occupies less memory because it occupies the memory of largest member only.

Disadvantage of union over structure

It can store data in one member only.

Example of Union in C

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
union emp
{
int ID;
char name[10];
double salary;
}u; //  reference of union
printf("Enter emp Id: ");
scanf("%d",&u.ID);
printf("Emp ID: %d",u.ID);
printf("Enter emp Name: ");
scanf("%s",&u.name);
printf("Emp Name: %s",u.name);
printf("Enter emp Salary: ");
scanf("%f",&u.salary);
printf("Emp Salary: %f",u.salary);
getch();
}
```

Output:

Emp ID: 100
Emp Name: Porter
Emp Salary: 20000

Memory allocation : size of structure and union

The amount of memory required to store a structure variable is the sum of the size of all member.

In union the amount of memory required is equivalent to that required by the largest member.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
struct smem
{
char name[33];
int age;
float marks;
}sm;
union umem
{
char name[33];
int age;
float marks;
}um;
printf("the size of structure is %d",sizeof(sm));
printf("\nthe size of union is %d",sizeof(um));
getch();
}
```

```
the size of structure is 39
the size of union is 33_
```

Output is

When you use Structure and Union?

When you need to manipulate the data for all member variables then use structure. When need to manipulate only one member then use union.

Note:

- All the properties of the structure are same for union, except initialization process.
- In case of structure initialize all data members at a time because memory location are different but in case of union only one member need to be initialize.
- In case of union if we initializing multiple member then compiler will gives an error.

Difference between Structure and Union

	Structure	**Union**
1	For defining structure use struct keyword.	For defining union we use union keyword

2	Structure occupies more memory space than union.	Union occupies less memory space than Structure.
3	In Structure we can access all members of structure at a time.	In union we can access only one member of union at a time.
4	Structure allocates separate storage space for its every members.	Union allocates one common storage space for its all members. Union find which member need more memory than other member, then it allocate that much space
5	Can take part in complex data structure	Cannot take part in complex data structure
6	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
7	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

8	Syntax:
	struct user-definedname
	{
	Data-type member1
	Data-type member2
	}var;
		Syntax:
		Union user-defined name
		{
		Data-type member1;
		Data-type member2;
		};

Pointers

Defination of pointer

*A pointer is a variable which contains or hold the address of another variable. It's value is the address of another variable, i.e. direct address of memory location.*Like any variable or constants, we must declare a pointer before we use it to store any variable address. We can create pointer variable of any type of variable for example integer type pointer is 'int *ptr'.

Pointer Operator

In pointer following symbols are use;

Symbol	Nme	Description
& (ampersand sign)	Address of operator	Give the address of a variable
* (asterisk sign)	Indirection operator	Gives the contents of an object pointed to by a

		pointer.

Advantage of pointer

- Pointer reduces the code and improves the performance, because it direct access the address of variable.
- Using pointer concept we can return multiple value from any function.
- Using pointer we can access any memory location from pointer.

Address Of Operator

The address of operator **&** gives the address of a variable. For display address of variable, we need %**u**

Declaring a pointer

In C language for declared pointer we can use * (asterisk symbol).

Syntax:

```
int *p;  //pointer to integer
char *ch; //pointer to character
```

Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=50;
printf("\nValue of a is: %d",n);
printf("\Address of &n is: %u",&n);
getch();
}
```

Output

Value of a is: 50
Address of a is: 1002

Example of pointer

In below image pointer variable stores the address of num variable i.e. EEE3. The value of num is 50 and address of pointer prt is CCC4.

```
#include<stdio.h>
int main ()
{
 int num=50;
 int  *ptr; // pointer variable
 ptr = # // store address of variable in pointer
 printf("Address of num variable: %x\n", &num);
 /* address stored in pointer variable */
```

```
 printf("Address stored in ptr variable: %x\n", ptr );
 /* access the value using the pointer */
 printf("Value of *ptr variable: %d\n", *ptr );
 return 0;
}
```

Output

Address of num variable: EEE3

Address stored in ptr variable: CCC4

Value of *ptr variable: 50

Example: a program to display the address and the content of a variable

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int x=8;
printf("\n address of x=%u",&x);
printf("\n value of x=%d",x);
printf("\n the value of x=%d",*(&x));
getch();
}
```

The output is

Address of x=65524

Value of x=8

The value of x=8

Note that value of *(&x) is same as value of x.

Example 2:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int x=8;
int *y;
y=&x;
printf("\n address of x=%u",y);
printf("\nvalue of x=%d",*y);
getch();
}
```

Output is

address of x=65524

value of x=8

Examples:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int u=3;
int v;
int *pu;
int *pv;
pu=&u;
v=*pu;
pv=&v;
printf("\n u=%d &u=%u pu=%u *pu=%d",u,&u,pu,*pu);
printf("\n v=%d &v=%u pv=%u *pv=%d",v,&v,pv,*pv);
getch();
}
```

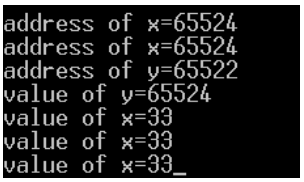
Output

u=3 &u=65524 pu=65224 *pu=3

v=3 &v=65522 pv=65522 *pv=3

Question:a program to display the address and the content of a pointer variable and ordinary variable.

```
#include<stdio.h>
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int x=33;
int *y;
y=&x;
printf("\n address of x=%u",&x);
printf("\n address of x=%u",y);
printf("\n address of y=%u",&y);
printf("\n value of y=%u",y);
printf("\n value of x=%d",x);
printf("\n value of x=%d",*(&x));
printf("\n value of x=%d",*y);
getch();
}
```



Output is

Q. Differentiate between **array and pointers**.

Array	Pointer
1.An array is a single, pre allocated chunk of contiguous elements (all of the same type), fixed in size and location.	1. A pointer is a place in memory that keeps address of another place inside
2. Array can be initialized at definition. Example int num[] = { 2, 4, 5}	2. Pointer can't be initialized at definition.
3. They are static in nature. Once memory is allocated , it cannot be resized or freed dynamically.	3. Pointer is dynamic in nature. The memory allocation can be resized or freed later.
4. The assembly code of Array is different than Pointer.	4. The assembly code of Pointer is different than Array
5.Array consists of contiguous memory location.	5.Address is location of another object (typically a variable) in memory.
6.Syntax: datatype name[size];	6.syntax: data type *variable

Pointer Expression and Assignment

A program to display the memory address of a variable using pointer before increment and after increment.

ptr++; ptr=ptr+size of (data type) use right value of ptr and then ptr is incremented after statement execution.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int x;
int *ptr;
x=32;
ptr=&x;
printf("\n the memory address before increment =%u",ptr);
ptr++;
printf("\n the memory address after increment =%u",ptr);
getch();
}
```

Suppose memory address =65524

Output is

```
the memory address before increment =65524
the memory address after increment =65526_
```

Question:

Find output

Main()

```
{
int x,*y,**z;
x=5;
y=&x;
z=&y;
printf(''%d'',**z);
}
```

Output is 5

Q. Explain the meaning

int *p; =a point that will point to an integer data.
int *p[10]; =p is a 10 element array of pointer to integer data.
int (*p)[10]; =p is a function that returns a pointer to an integer data.
int (*p)[void]; =p is a function that returns a pointer to an integer data.
int *p(char *a);=p is a function that accepts an argument which is a pointer to a character and return a pointer to an integer data.

Question:

Given below answer the question:

Variable name		**address**	**value**
P		20116	5
Q		21536	20116
R		22256	21536
&p=20116	&q=20116	**R=21536	q=20116

Pointer and function

A function call is a method to access a function. There are two types of function calls.

Call by value

```
#include<stdio.h>
#include<conio.h>
void swap(int , int );
int main()
{
int a=5, b=10;
printf("before swapping, values of a = %d, b = %d", a, b);
swap(a,b);
printf("\nafter swapping, values of a = %d, b = %d", a, b);
getch();
return 0;
}
void swap(int x, int y)
{
int temp;
temp = x;
x = y;
```

```
y = temp;
}
```
Output:

before swapping, values of a = 5, b = 10
after swapping, values of a = 5, b = 10

Call by reference

```
#include<stdio.h>
#include<conio.h>
void swap(int *, int *);
int main()
{
int a=5, b=10;
printf("before swapping, values of a = %d, b = %d", a, b);
swap(&a,&b);
printf("\nafter swapping, values of a = %d, b = %d", a, b);
getch();
return 0;
}
void swap(int *x, int *y)
{
int temp;
temp = *x;
*x = *y;
*y = temp;
}
```

Output:
before swapping, values of a = 5, b = 10
after swapping, values of a = 10, b = 5

Working with Files

A **file** represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a ready made structure.FILE is a data structure where we can permanently store data on the secondary storage devices such as harddisk, magnetic tapes and so on.

In C language, we use a structure **pointer of file type** to declare a file.

FILE ***fp**;

Where FILE must be in capital letter and 'fp' is a pointer of the FILE type.

Everything that is done with FILE is to be done with file pointer called file handler.

C provides a number of functions that helps to perform basic file operations. Following are the functions:

Function	Description

fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

File Handling Functions in detail

File handling functions are as follows:

i) Closing a File

The fclose() function is used to close an already opened file.

General Syntax :

fclose(FILE pointer);

Example: fclose(fp);

Here fclose() function closes the file and returns zero on success, or EOF. if there is an error in closing the file. This EOF is a constant defined in the header file stdio.h.

ii) rewind()

The rewind() function places the offset pointer to the beginning of the file, irrespective of current offset position.

iii) fseek()

It is used to move the reading control to different positions using fseek function.

Syntax:

fseek(filepointer,offset, offset_initial);

Example:

fseek(fp,5,SEEK_SET);

iv) ftell()

It tells the byte location of current position of cursor in file pointer.

Example:

Position=ftell(fp);

vi) fopen()

We can use the fopen( ) function to create a new file or to open an existing file, this call will initialize an object of the type FILE, which contains all the information necessary to control the stream.

FILE *fopen (const char * filename, const char * mode);

Example:

FILE *fp;
fp=fopen("C:\\manoj.txt","w+");

vi) fgetc()

The fgetc() function reads a character from the input file referenced by fp. The return value is the character read, or in case of any error it returns EOF. The following functions allow you to read a string from a stream.

Syntax:

int fgetc(FILE * fp)
Example:
char c;
c= fgetc(fp);

vii)fscanf()

this function is used to read data from a file.
Syntax:fscanf(file pointer,"format specifier",&v1,&v2,……);
Example: fscanf(fp,"%d%d%d", &v1,&v2,……);

viii)fprintf()

this function is used to write data into a file.
Syntax:fprintf(file pointer,"format specifier",v1,v2,……);
Example: fprinf(fp,"%d%d%d\n", v1,v2,……);

Opening a File or Creating a File

1.The fopen() function is used to create a new file or to open an existing file.

General Syntax :
FILE file_pointer

Example: FILE *fp;

***fp** is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file.

2.The file must be opened before using it.

Syntax:
File_pointer=fopen("file name","mode");

Here **filename** is the name of the file to be opened and **mode** specifies the purpose of opening the file.

Example: fp=fopen("Nepal.txt","r");

Mode can be of following types,

Mode	Description
R	opens a text file in reading mode.
W	opens or create a text file in writing mode.
A	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode

Difference between Append and Write Mode

Write (w) mode and Append (a) mode, while opening a file are almost the same. Both are used to write in a file. In both the modes, new file is created if it doesn't exist already.

The only difference they have is, when you open a file in the write mode, the file is reset, resulting in deletion of any data already present in the file. While in append mode this will not happen. Append mode is used to append or add data to the existing data of file (if any). Hence, when we open a file in Append (a) mode, the cursor is positioned at the end of the present data in the file.

Opening, Reading, Writing and Appending on/from Data File

Openeing a file for writing

a) create a file pointer
b) open file in write (w or a mode)
c) perform I/O operation
d) save the I/O results.

Example WAP which asks name, roll number and marks of students and store it into data file ''student.dat''.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char name[22];
int roll;
float marks;
FILE *fp;
fp=fopen("student.dat","w");
printf("enter name");
scanf("%s",name);
printf("enter roll number");
scanf("%d",&roll);
printf("enter marks");
scanf("%f",&marks);
fprintf(fp,"%s%d%f",name,roll,marks);
fclose(fp);
```

```
getch();
}
```

Opening a file for reading

a. create a FILE pointer
b. open file in read mode(r mode)
c. read data from file using fscanf syntax
d. display the data as required

Examle:WAP to display contents from the file ''student.dat''

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char name[22];
int roll;
float marks;
FILE *fp;
fp=fopen("student.dat","r");
fscanf(fp,"%s%d%f",name,&roll,&marks);
printf("name %s",name);
printf("roll %d",roll);
printf("marks %f",marks);
fclose(fp);
getch();
}
```

Q. WAP that stores rollno, name, age, percentage of a students to a data file until pressing Y yes.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char name[50];
int rollno;
int age;
float per;
char c;
FILE *fp;
fp=fopen("ss.txt","w");
do
{
```

```
printf("enter name");
scanf("%s",name);
printf("enter roll no");
scanf("%d",&rollno);
printf("enter age");
scanf("%d",&age);
printf("enter percentage");
scanf("%f",&per);
fprintf(fp,"\n%s\t%d\t%d\t%f",name,rollno,age,per);
printf("do you want to continue yes or no");
c = getche();    //scanf("%c",&choice);
}while(c=='Y'||c=='y');
fclose(fp);
getch();
}
```

Q. WAP to delete and rename the data file using remove and rename command.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char name[44];
int roll;
FILE *fp;
fp=fopen("oldf.txt","w");
printf("enter name");
scanf("%s",name);
printf("enter roll");
scanf("%d",&roll);
fprintf(fp,"%s%d",name,roll);
fclose(fp);
rename("oldf.txt","newf.txt");
remove("newf.txt");
getch();
}
```

Q. WAP to read and write name, address, rollno, and percentage of a student from/to data file using **fscanf()** and **fprintf().**

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

```
char name[55];
int roll;
float per;
FILE *fp;
fp=fopen("sad.txt","w");
printf("enter name");
scanf("%s",name);
printf("enter roll");
scanf("%d",&roll);
printf("enter percentage");
scanf("%f",&per);
fprintf(fp,"\n%s\n%d\n%f",name,roll,per);
fclose(fp);
fp=fopen("sad.txt","r");
printf("reading data\n");
fscanf(fp,"%s%d%f",name,roll,per);
printf("\n%s\n%d\n%f",name,roll,per);
fclose(fp);
getch();
}
```

Q. WAP to enter person's name, address, rollno, and marks of 'n' students.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char name[55];
char address[44];
int roll;
float me;
FILE *fp;
int i,n;
fp=fopen("sad.txt","a");
printf("enter how many records");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter name");
scanf("%s",name);
printf("enter address");
scanf("%s",address);
printf("enter roll");
```

```
scanf("%d",&roll);
printf("enter marks in english");
scanf("%f",&me);
fprintf(fp,"\n%s\n\n%s%d\n%f",name,address,roll,me);
}
fclose(fp);
getch();
}
```

Q.WAP to enter employee name, emp_id, and emp_salary of 'n' employee using structure and display the result in proper format.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
struct employee
{
char emp_name[55];
int emp_id;
float emp_salary;
}e1;
FILE *fp;
int i,n;
fp=fopen("emp.txt","w");
printf("enter how many records");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter employee name");
scanf("%s",e1.emp_name);
printf("enter employee id");
scanf("%d",&e1.emp_id);
printf("enter employee salary");
scanf("%f",&e1.emp_salary);
fprintf(fp,"\n%s\t%d\t%f",e1.emp_name,e1.emp_id,e1.emp_salary);
}
fclose(fp);
fp = fopen("emp.txt","r");
printf("\n\ndisplay the records of employee");
printf("\nemployee_name\t\temploye_id\t\temployee_salary");
while(fscanf(fp,"%s%d%f",e1.emp_name,&e1.emp_id,&e1.emp_salary) != EOF)
{
```

```
printf(“\n%s\t\t%d\t\t%f",e1.emp_name,e1.emp_id,e1.emp_salary);
}
getch();
}
```

Q. Write a program to enter name, roll-number and marks of 10 students and store them in the file.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char name[22];
int roll,i;
float marks;
FILE *fp;
fp=fopen("student.dat","w");
for(i=1;i<=10;i++)
{
printf("enter name");
scanf("%s",name);
printf("enter roll number");
scanf("%d",&roll);
printf("enter marks");
scanf("%f",&marks);
fprintf(fp,"%s%d%f",name,roll,marks);
}
fclose(fp);
getch();
}
```

Exercise

1. Define operator. Explain different types of operator used in C.
2. Explain precedence and associatively of different operators on C.
3. Explain about ternary operator and comma operator with example.
4. What is type conversion? Explain.
5. Differentiate between prefix and postfix increment and decrement operator.
6. Define symbolic constant with example.
7. Define library function with example.
8. What is an array?
9. How do you declare an array in C?
10. What are the types of arrays? Explain with example
11. How do you load elements of one dimensional and two dimensional array? Explain with examples.
12. Compare single dimensional and multidimensional array with example.
13. What is initialization of an array? Explain the initialization of one and two dimensional array with examples.
14. Write a program that reads the elements of an array and display the elements in reverse order.
15. What is searching? Write a program to search an item in an array.
16. Write a program to find the second smallest and second largest elements of an array of 10 integer.
17. Write a menu-driven program to perform the following operations
    a. Read two arrays a[M] and b[N] of size M and N respectively
    b. Sort both arrays in ascending order
    c. Find minimum element in array a.
    d. Find maximum element in array b.
    e. Exit
18. Write a program in C that reads the elements of an array from the user and display the elements and their sum.
19. Write a program to find the largest and smallest numbers from a set of 10 numbers entered by the user.
20. Write a program to find the addition and subtraction, and multiplication of two matrices by asking size (the rows and columns) by users. Perform the operations if possible.
21. Write a program to search an item in the array. Read elements and key item from user.
22. Write a program to perform the sorting of the elements of the array in ascending and descending order. Define array of size 100 in advance and ask size with user and load data from user.
23. Write a program to accept 'n' numbers' then increment each elements of the array by constant– c where c is also input by user.
24. Write a program to accept 'n' numbers from use and count positive, negative and zeros in the array.
25. Write a program to find the sum of the diagonal elements of a rectangular matrix.
26. Write a program that reads the elements of 3x4 matrix and find the sum of the elements of each row, column and total sum of all the elements.

27. What is a structure? How do you declare it in C? Explain with example.
28. What is structure variable? Write a program in C to demonstrates the initialization of structure variable.
29. What do you mean by array of structure? Write a program in C to read the roll number, name, sex, and total mark of a10 students and display the record of the students.
30. How do you pass the structure variable to function? Define a structure time sec, min, hrs fields. Define a function to add two time t1 and t2 and returns time structure variable.
31. How do you pass pointer to the function? Write a program to illustrate the passing of structure pointer to function.
32. Define a structure employee with id, name, address, salary, post . Read the record of 20 employee using read(struct employee) function . Display the record of all employee whose salary is greater than 20000.
33. What is a union? Write a program to demonstrate the uses of union.
34. Differentiate between structure and union with examples.
35. What is meant by the following terms? Give examples.
    a. Nested structures
    b. Array of structures.
36. Write a program that reads the information of a person with pid, name, address, age using structure and display the record.
37. Write a program in C to read the record of a students with fields roll_no, name, address, phone, section of 10 students and display the records like as given below

a.	Roll No	Name	Address	Phone	Section
b.	101	Ravi Dangol	Kathmandu	4242424	A
c.	102	Sabin Thapa	Pokhara	6969693	B

38. Write a program that uses a structure to input two complex numbers, pass them to a function to multiply them and return the result to main function and display.
39. Write a program that uses structure to read employee ID , name, age and salary of N employee. Sort them on the basis of name in alphabetical order.
40. Write a program to read account number, name and balance of 'n' number of customers from users and do the following by making use of structure variable.
41. List the name of customer whose balance is below minimum balance.
42. Ask user for account number and display the information corresponding to that account number. Display the name of customer having highest and lowest balance
43. What is a pointer? Explain the applications of pointer.
44. What are the features of a pointers in C? Explain
45. What do you mean by pointer arithmetic? Explain with a program.
46. What are the relationship between arrays and pointers? Explain with a program.
47. Write a program to perform simple arithmetic operation using pointers.
48. Write a program to reverse an array using pointer.
49. Write a program to display the address and content of an integer variable using pointer.(declare int x=100; int *ptr; ptr=&x; display x, ptr, and *ptr)

50. Write a program to display the memory address of a variable before and after pointer increment.
51. Write a program to demonstrate the use of call by reference by displaying the content of the variables before and after swapping.
52. Write a program to access the element of an array using pointer.(Hint: assign initial address to a pointer variable and display the pointer and increment it int *ptr=&a[0]; display *ptr and ptr++ within a for loop)
53. Write a program to demonstrate the following expressions x[i],*(x+i),*(i+x), i[x] are equivalent.213213213213213213213213213213213
54. What is a file? What are the file operations? Explain
55. What are the strand streams? Explain
56. What are file streams? Explain
57. What is a stream? Explain
58. What is file processing? Write the steps to process the file.
59. What is use of fopen()? List the file open modes.
60. What are the character file I/O functions? List with the syntax.
61. What are the formatted file I/O functions? List with the syntax.
62. What are the record file I/O functions? List with the syntax.
63. Write short notes on Block I/O Functions
64. Distinguish between text mode and binary mode operation of a file.
65. What is random access? Explain