V2V EDTECH LLP

Online Coaching at an Affordable Price.

OUR SERVICES:

- Diploma in All Branches, All Subjects
- Degree in All Branches, All Subjects
- BSCIT / CS
- Professional Courses

📞 **+91 93260 50669**
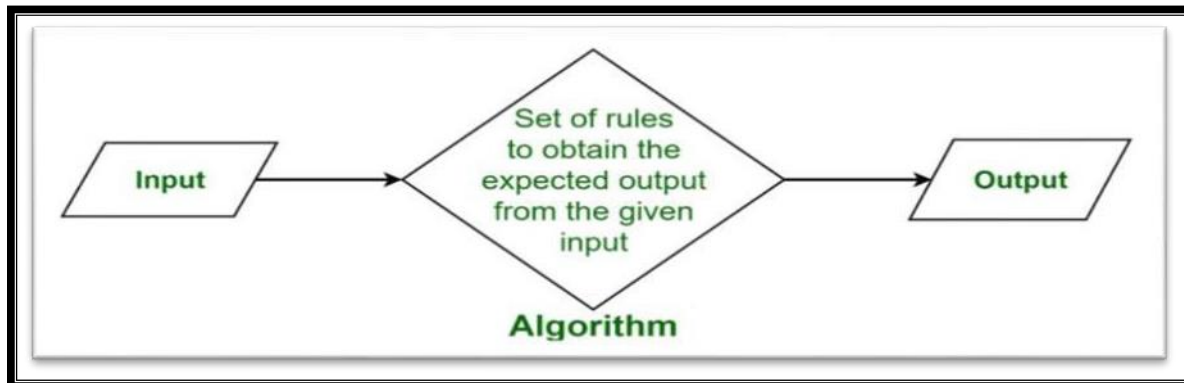
🌐 **v2vedtech.com**

▶️ **V2V EdTech LLP**

📷 **v2vedtech**

22226 PIC Notes | Programming in C Notes

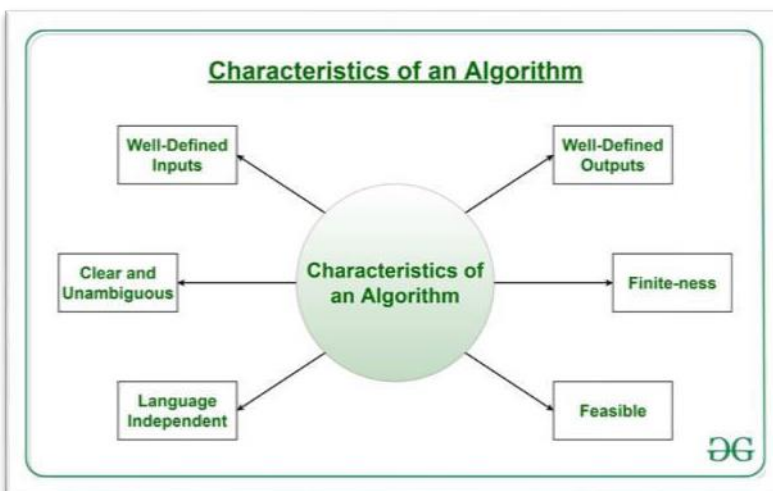| Sr. No	Unit Name
1	Program Logic Development
2	Basic of C Programming
3	Control Structures
4	Array and Structure
5	Functions
6	Pointers

Fundamentals of algorithms

Explain the term algorithm

Algorithm is a stepwise set of instructions written to perform a specific task..

What are the Characteristics of an Algorithm?

- **Clear and Unambiguous**: Algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- **Well-Defined Inputs**: If an algorithm says to take inputs, it should be well-defined inputs.
- **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well.
- **Finite-ness:** The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.
- **Feasible:** The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources. It must not contain some future technology, or anything.
- **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.

Example:

Write an algorithm to calculate the addition of two given numbers.
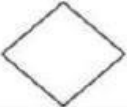
Step 1: Start

Step 2: Read two numbers A and B
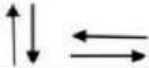
Step 3: Add A and B and store result in C

Step 4: Display C

Step 5: Stop

Explain the term Flowchart

Flowchart is a graphical representation of an algorithm.

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

Basic Symbols used in Flowchart Designs

1. **Terminal:** Terminal is the first and last symbols in the flowchart.
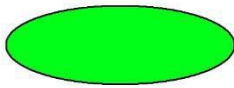
2. **Input/Output:** Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.

3. **Processing:** All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.

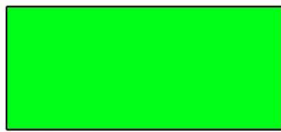4. **Decision** Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.
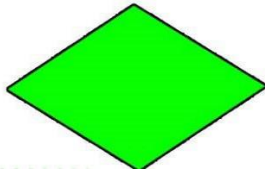
5. **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.
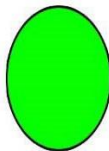
6. **Flow lines:** Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.
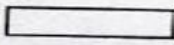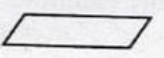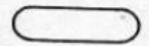
What are the advantages Of Using FLOWCHARTS?

• Communication: Flowcharts are better way of communicating the logic of a system to all concerned or involved.

• Effective analysis: With the help of flowchart, problem can be analysed in more effective way therefore reducing cost and wastage of time.

• Proper documentation: Program flowcharts serve as a good program documentation, which is needed for various purposes, making things more efficient.

• Efficient Coding: The flowcharts act as a guide or blueprint during the systems analysis and program development phase.

• Proper Debugging: The flowchart helps in debugging process.

• Efficient Program Maintenance: The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part

State the use of following symbols used for flowchart drawing:

(i) ▭   (ii) ◇

(iii) ▱   (iv) ⬭
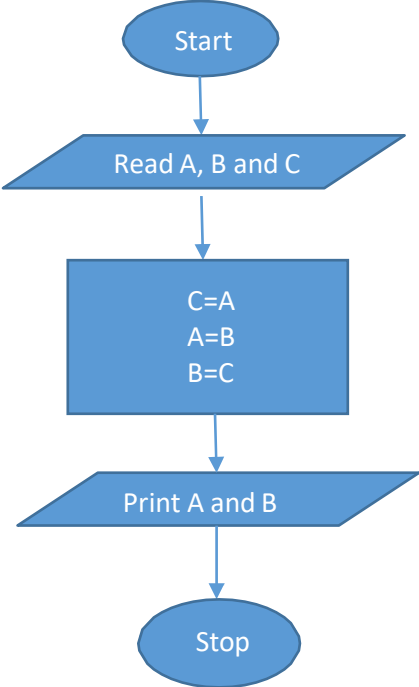
(i) ▭   General processing

(ii) ◇   Decision making

(iii) ▱   Input/ Output statements

(iv) ⬭   Start / Stop
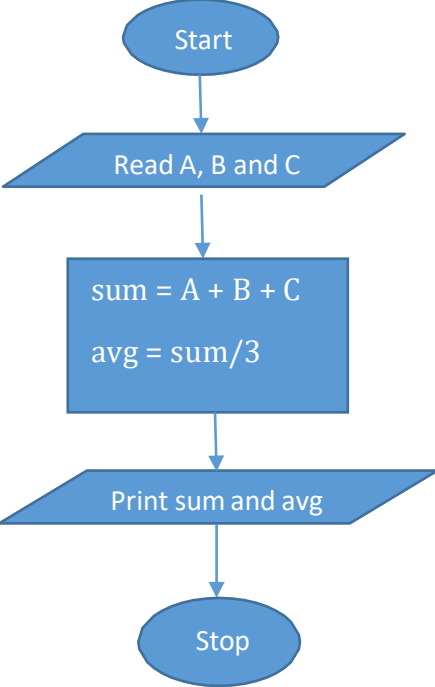
Write algorithm and flowchart of addition of two numbers

Algorithm	Flowchart
Step 1: Start Step 2: Read two numbers A and B Step 3: Add A and B and store result in C Step 4: Display C Step 5: Stop	

Write algorithm and flowchart for swapping of two numbers

Algorithm	Flowchart
Step 1: Start Step 2: Read three numbers A, B and C Step 3: C = A Step 4: A = B Step 5: B = C Step 6: Print A and B Step 7: Stop	Start Read A, B and C C=A A=B B=C Print A and B Stop

Write algorithm and flowchart to find sum and average of three numbers

Algorithm	Flowchart
Step 1: Start Step 2: Read three numbers A, B and C Step 3: sum = A + B + C Step 4: avg = sum/3 Step 5: Print sum and avg Step 6: Stop	Start Read A, B and C sum = A + B + C avg = sum/3 Print sum and avg Stop

Write an algorithm to determine whether a given number is divisible by 5 or not

Step 1- Start
Step 2- Read / input the number.
Step 3- if n%5==0 then goto step 5.
Step 4- else number is not divisible by 5 goto step 6.
Step 5- display the output number is divisible by 5.
Step 6- Stop

Write algorithm and draw flow-chart to print even numbers from 1 to 100.

Algorithm	Flowchart
Algorithm 1. Start 2. Initialize the variable i to 1. 3. while i<=100 4. if i%2==0 5. print the number 6. increment value of i 7. stop	

Algorithm & Flowchart to find the largest of two numbers

Algorithm	Flowchart
Algorithm Step-1 Start Step-2 Input two numbers say NUM1,NUM2 Step-3 IF NUM1 > NUM2 THEN print largest is NUM1 ELSE print largest is NUM2 ENDIF Step-4 Stop	

Algorithm & Flowchart to find the given number is odd or even.

Algorithm	Flowchart
Step 1: Start Step 2: [Take Input] Read: Number Step 3: Check: If Number%2 == 0 Then Print : N is an Even Number. Else Print : N is an Odd Number. Step 4: Exit	

Algorithm & Flowchart to find the largest of three numbers

Algorithm	Flowchart
Step-1 Start **Step-2** Read three numbers say num1,num2, num3 **Step-3** if num1>num2 then go to step-5 **Step-4** IF num2>num3 THEN print num2 is largest ELSE print num3 is largest ENDIF GO TO Step-6 **Step-5** IF num1>num3 THEN print num1 is largest ELSE print num3 is largest ENDIF **Step-6** Stop	

Algorithm & Flowchart to find Factorial of number n (n!=1x2x3x...n)

Algorithm	Flowchart
step 1. Start step 2. Read the number n step 3. [Initialize] i=1, fact=1 step 4. Repeat step 4 through 6 until i=n step 5. fact=fact*i step 6. i=i+1 step 7. Print fact step 8. Stop	

Algorithm & Flowchart to find if a number is prime or not

Algorithm	Flowchart
Step-1 Start Step-2 Input NUM Step-3 R=SQRT(NUM) Step-4 I=2 Step-5 IF (I > R) THEN Write NUM is Prime Number Stop ENDIF Step 6 IF (NUM % I ==0) THEN Write NUM is Not Prime Stop ENDIF Step-7 I = I + 1 Step-8 Go to Step-5	

What is Pseudocode

Definition: Pseudocode is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations. It is used for creating an outline or a rough draft of a program.

Example:

Write pseudo code that will perform the following.

a) Read in 5 separate numbers.

b) Calculate the average of the five numbers.

c) Find the smallest (minimum) and largest (maximum) of the five entered numbers.

a) Write "please enter 5 numbers"

Read n1,n2,n3,n4,n5

b) Write "The average is"

Set avg to (n1+n2+n3+n4+n5)/5

Write avg

c) If(n1 < n2)

Set max to n2

Else

 Set max to n1

If(n3 > max)

Set max to n3

If(n4 > max)

Set max to n4

If(n5 > max)

Set max to n5

Write "The max is"

Write max

C Program to Find Volume and Surface Area of Sphere

The formula used in this program are Surface_area = 4 * Pi * r2, Volume = 4/3 * Pi * r3
where r is the radius of the sphere, Pi= 22/7

```c
/*
 * C Program to Find Volume and Surface Area of Sphere
 */
#include <stdio.h>
#include <math.h>

int main()
{

    float radius;
    float surface_area, volume;

    printf("Enter radius of the sphere : \n");
    scanf("%f", &radius);
    surface_area = 4 * (22/7) * radius * radius;
    volume = (4.0/3) * (22/7) * radius * radius * radius;
    printf("Surface area of sphere is: %.3f", surface_area);
    printf("\n Volume of sphere is : %.3f", volume);
    return 0;
}
```

Output

Enter radius of the sphere :

40

Surface area of sphere is: 19200.000

Volume of sphere is : 256000.000

C program to swap two numbers (interchange the content of two variables)

```c
#include <stdio.h>
int main()
{
    int x, y, t;
    printf("Enter two integers\n");
    scanf("%d%d", &x, &y);
```

```
    printf("Before Swapping\nFirst integer = %d\nSecond integer = %d\n", x, y);
    t = x;
    x = y;
    y = t;
    printf("After Swapping\nFirst integer = %d\nSecond integer = %d\n", x, y);
    return 0;
}
```

The output of the program:

Enter two integers

23

45

Before Swapping

First integer = 23

Second integer = 45

After Swapping

First integer = 45

Second integer = 23

If a five-digit number is input through the keyboard, write a program to calculate the sum of its digits. (Hint: Use the modulus operator '%')

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

int main()
{
    long int Num;
    int Digit1, Digit2,Digit3, Digit4, Digit5 , Sum=0;
    Clrscr();
    Printf("Enter 5 digit Number : ");
    Scanf("%ld",&Num);
    Digit1 = Num % 10;
    Num = Num / 10;
    Digit2 = Num % 10;
```

Num = Num / 10;

Digit3 = Num % 10;

Num = Num / 10;

Digit4 = Num % 10;

Num = Num / 10;

Digit5 = Num % 10;

Num = Num / 10;

Sum = Digit1 + Digit2 + Digit3 + Digit4 + Digit5;

printf("Sum of Digits = %d", Sum);

getch();

Return 0;

}

Draw a flowchart for checking whether given number is prime or not.
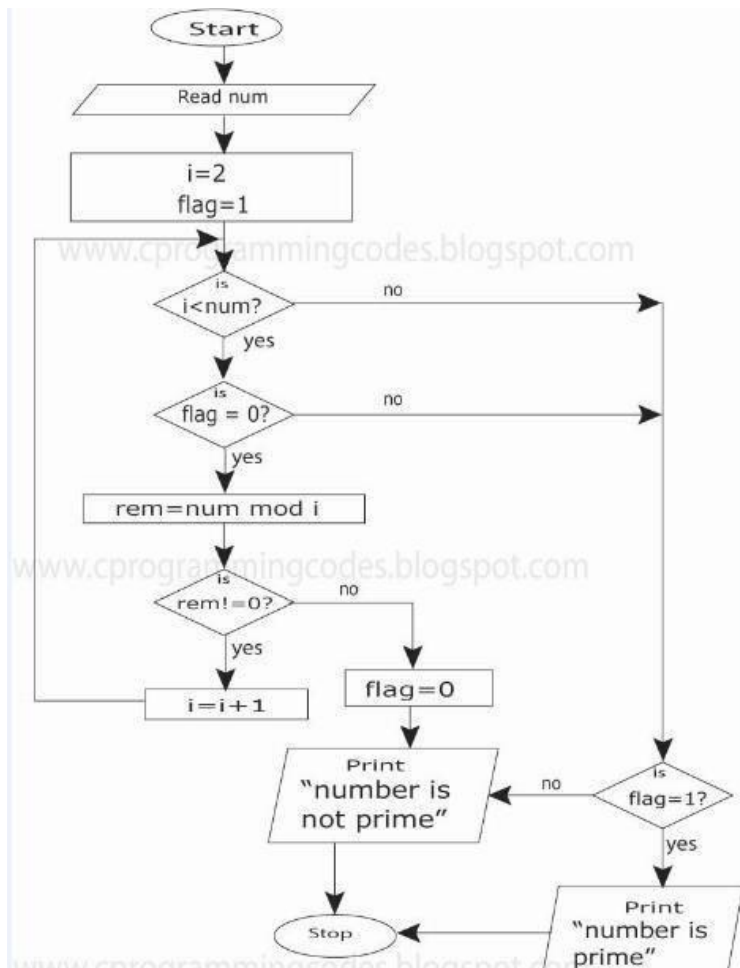
History of C Language

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

Dennis Ritchie is known as the **founder of the c language**.

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

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

Features of C Language

C is the widely used language. It provides many **features** that are given below.

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

```
              Portable           Mid-level

     Simple                                    Structured

   Memory                C Language            Rich library
  Mangement

     Faster                                     Extensible

              Pointers            Recursion
```
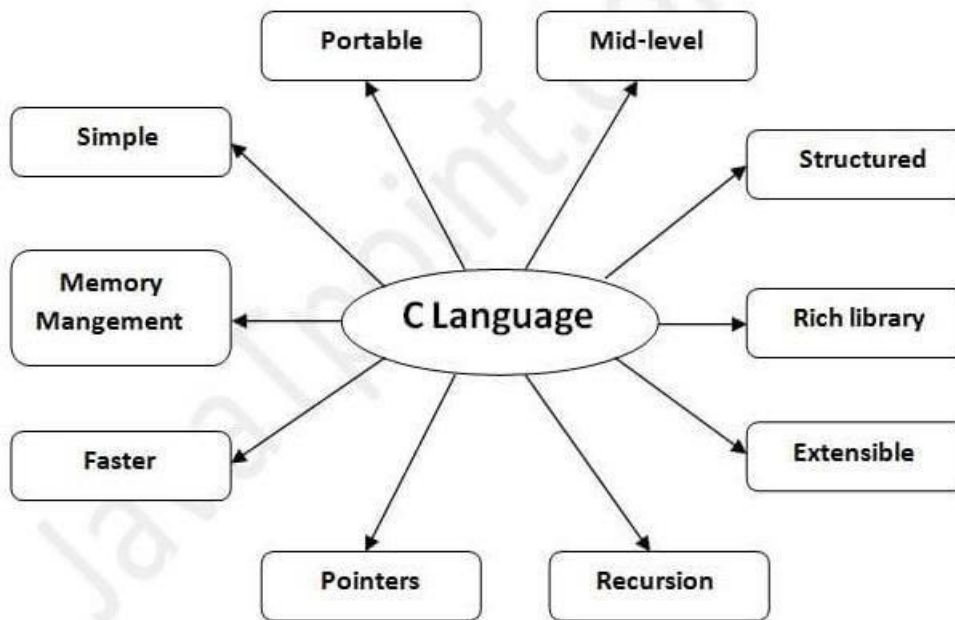
Basic Structure of C Program

Basic Structure of C Programs
Documentation Section
Link Section
Definition Section
Global Declaration Section

main() Function Section
{
     Declaration Part
     Executable Part
}

Subprogram Section
     Function 1
     Function 2
     Function 3
       -
       -
       -
     Function n

First C Program

```
#include <stdio.h>

int main()

{

printf("Hello C Language");

return 0;

}
```

Preprocessor directive: it includes information required to execute specific function from specified header file.

Save the file with name first.c

Output:

Hello C Language

Header Files

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

All the header file have a '.h' an extension that contains C function declaration and macro definitions.

In other words, the header files can be requested using the preprocessor directive *#include*. The default header file that comes with the C compiler is the *stdio.h*. (Standard input output)

Explain any four library functions under conio.h header file.

clrscr() -This function is used to clear the output screen.

getch() -It reads character from keyboard

getche()-It reads character from keyboard and echoes to o/p screen

putch - Writes a character directly to the console.

textcolor()-This function is used to change the text color

textbackground()-This function is used to change text background

Give the significance of <math.h> and <stdio.h> header files.

enumerated **Write syntax and use of pow ()function of <math.h> header file.**

pow()- compute the power of a input value

Syntax:

double pow (double x, double y);

main() function in C

main() function is the entry point of any C program. It is the point at which execution of program is started. When a C program is executed, the execution control goes directly to the main() function. Every C program have a main() function.

```
void main()
{
 .........
 .........
}
```

In above syntax;

- **void:** is a keyword in C language, void means nothing, whenever we use void as a function return type then that function nothing return. here main() function no return any value.
- In place of void we can also use **int** return type of main() function, at that time main() return integer type value.
- **main:** is a name of function which is predefined function in C library.

Character set of C

C language also has a set of characters which include **alphabets, digits**, and **special symbols**. C language supports a total of 256 characters.

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

1. Alphabets
2. Digits
3. Special Symbols

Alphabets

C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.

lower case letters - **a to z**

UPPER CASE LETTERS - **A to Z**

Digits

C language supports 10 digits which are used to construct numerical values in C language.

Digits - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

Special Symbols

C language supports a rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, backspaces, and other special symbols.

Special Symbols - **~ @ # $ % ^ & * ( ) _ - + = { } [ ] ; : ' " / ? . > , < \ | tab newline space NULL bell backspace verticaltab etc.,**

Every character in C language has its equivalent ASCII (American Standard Code for Information Interchange) value.

C Tokens

Every C program is a collection of instructions and every instruction is a collection of some

individual units. Every smallest individual unit of a c program is called token. Every

instruction in a c program is a collection of tokens. Tokens are used to construct c programs

and they are said to the basic building blocks of a c program.

In a c program tokens may contain the following...

1. Keywords

2. Identifiers

3. Operators

4. Special Symbols

5. Constants

6. Strings

7. Data values

In a C program, a collection of all the keywords, identifiers, operators, special symbols, constants, strings, and data values are called tokens.

C Keywords

Keywords are the reserved words with predefined meaning which already known to the compiler

Whenever C compiler come across a keyword, automatically it understands its meaning.

Properties of Keywords

1. All the keywords in C programming language are defined as lowercase letters so they must be used only in lowercase letters
2. Every keyword has a specific meaning, users can not change that meaning.
3. Keywords can not be used as user-defined names like variable, functions, arrays, pointers, etc...
4. Every keyword in C programming language represents something or specifies some kind of action to be performed by the compiler.

5. C has 32 Keywords as follows:

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile

do	if	static	while

C Identifiers

The identifier is a user-defined name of an entity to identify it uniquely during the program execution

Example

int marks;
char studentName[30];

Here, **marks** and **studentName** are identifiers.

Rules for Creating Identifiers

1. An identifier can contain **letters** (UPPERCASE and lowercase), **numerics** & **underscore** symbol only.
2. An identifier should not start with a numerical value. It can start with a letter or an underscore.
3. We should not use any special symbols in between the identifier even whitespace. However, the only underscore symbol is allowed.
4. Keywords should not be used as identifiers.
5. There is no limit for the length of an identifier. However, the compiler considers the first 31 characters only.
6. An identifier must be unique in its scope.

C data types

The Data type is a set of value with predefined characteristics. data types are used to declare variable, constants, arrays, pointers, and functions.

In the c programming language, data types are classified as follows...

1. Primary data types (Basic data types OR Predefined data types)
2. Derived data types (Secondary data types OR User-defined data types)
3. Enumeration data types
4. Void data type

Primary data types

The primary data types in the C programming language are the basic data types. All the primary data types are already defined in the system. Primary data types are also called as Built-In data types. The following are the primary data types in c programming language...

1. Integer data type
2. Floating Point data type

3. Double data type
4. Character data type

Basic Datatypes (Primary Datatpyes)

Interger	Floating Point	Character
Signed Unsigned	float double long double	char signed char Unsigned Char
int short int long int	int short int long int	
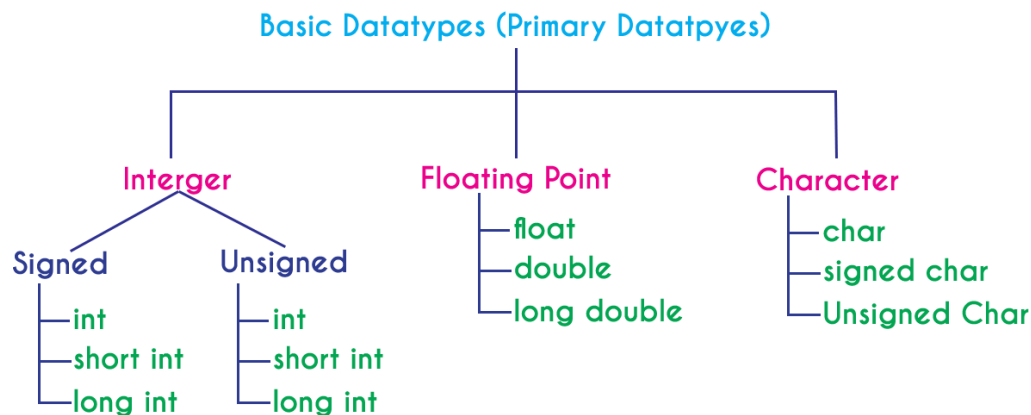
Integer Data type

The integer data type is a set of whole numbers. Every integer value does not have the decimal value. We use the keyword "**int**" to represent integer data type in c.

Type	Size (Bytes)	Range	Specifier
int (signed short int)	2	-32768 to +32767	%d
short int (signed short int)	2	-32768 to +32767	%d
long int (signed long int)	4	-2,147,483,648 to +2,147,483,647	%d
unsigned int (unsigned short int)	2	0 to 65535	%u
unsigned long int	4	0 to 4,294,967,295	%u

Floating Point data types

Floating-point data types are a set of numbers with the decimal value. Every floating-point value must contain the decimal value. The floating-point data type has two variants...

- float
- double

We use the keyword "**float**" to represent floating-point data type and "**double**" to represent double data type in c. Both float and double are similar but they differ in the number of decimal places. The float value contains 6 decimal places whereas double value contains 15 or 19 decimal places. The following table provides complete details about floating-point data types.

Type	Size (Bytes)	Range	Specifier
float	4	1.2E - 38 to 3.4E + 38	%f
double	8	2.3E-308 to 1.7E+308	%ld
long double	10	3.4E-4932 to 1.1E+4932	%ld

Character data type

The character data type is a set of characters enclosed in single quotations. The following table provides complete details about the character data type.

Type	Size (Bytes)	Range	Specifier
char (signed char)	1	-128 to +127	%c
unsigned char	1	0 to 255	%c

The following table provides complete information about all the data types in c programming language...

	Integer	Floating Point	Double	Character
What is it?	Numbers without decimal value	Numbers with decimal value	Numbers with decimal value	Any symbol enclosed in single quotation
Keyword	int	float	double	char
Memory Size	2 or 4 Bytes	4 Bytes	8 or 10 Bytes	1 Byte
Range	-32768 to +32767 (or) 0 to 65535 (Incase of 2 bytes only)	1.2E - 38 to 3.4E + 38	2.3E-308 to 1.7E+308	-128 to + 127 (or) 0 to 255
Type Specifier	%d or %i or %u	%f	%ld	%c or %s
Type Modifier	short, long signed, unsigned	No modifiers	long	signed, unsigned
Type Qualifier	const, volatile	const, volatile	const, volatil	const, volatile

void data type

The void data type means nothing or no value. Generally, the void is used to specify a function which does not return any value. We also use the void data type to specify empty parameters of a function.

Enumerated data type

An enumerated data type is a user-defined data type that consists of integer constants and each integer constant is given a name. The keyword "**enum**" is used to define the enumerated data type.

Here is the syntax of enum in C language,

enum enum_name{const1, const2, ... };

The enum keyword is also used to define the variables of enum type. There are two ways to define the variables of enum type as follows.

enum week{sunday, monday, tuesday, wednesday, thursday, friday, saturday};
enum week day;

Here is an example of enum in C language,

Example

```c
#include<stdio.h>

enum week{Mon=10, Tue, Wed, Thur, Fri=10, Sat=16, Sun};

enum day{Mond, Tues, Wedn, Thurs, Frid=18, Satu=11, Sund};

int main() {

  printf("The value of enum week: %d\t%d\t%d\t%d\t%d\t%d\t%d\n\n",Mon , Tue, Wed, Thur, Fri, Sat, Sun);

  printf("The default value of enum day: %d\t%d\t%d\t%d\t%d\t%d\t%d",Mond , Tues, Wedn, Thurs, Frid, Satu, Sund);

  return 0;

}
```

Output

| The value of enum week: 10 | 11 | 12 | 13 | 10 | 16 | 17 |
| The default value of enum day: 0 | 1 | 2 | 3 | 18 | 11 | 12 |

Derived data types

Derived data types are user-defined data types. The derived data types are also called as user-defined data types or secondary data types. In the c programming language, the derived data types are created using the following concepts...

- Arrays
- Structures
- Unions
- Enumeration

C Variables

Variable is a name given to a memory location where we can store different values of the same datatype during the program execution.

Every variable in c programming language must be declared in the declaration section before it is used. Every variable must have a datatype that determines the range and type of values be stored and the size of the memory to be allocated.

A variable name may contain letters, digits and underscore symbol. The following are the rules to specify a variable name...

1. Variable name should not start with a digit.
2. Keywords should not be used as variable names.
3. A variable name should not contain any special symbols except underscore(_).
4. A variable name can be of any length but compiler considers only the first 31 characters of the variable name.

Declaration of Variable

 Declaration Syntax:

datatype variableName;

   Example

        int number;

C Constants

A constant is a named memory location which holds only one value throughout the program execution.

In C programming language, a constant can be of any data type like integer, floating-point, character, string and double, etc.,

Floating Point constants

A floating-point constant must contain both integer and decimal parts. Some times it may also contain the exponent part. When a floating-point constant is represented in exponent form, the value must be suffixed with 'e' or 'E'.

Example

The floating-point value **3.14** is represented as **3E-14** in exponent form.

Character Constants

A character constant is a symbol enclosed in single quotation. A character constant has a maximum length of one character.

Example

'A'
'2'
'+'

String Constants

A string constant is a collection of characters, digits, special symbols and escape sequences that are enclosed in double quotations.

We define string constant in a single line as follows...
"This is btechsmartclass"

Creating constants in C

In a c programming language, constants can be created using two concepts...

1. **Using the 'const' keyword**
2. **Using '#define' preprocessor**

Using the 'const' keyword

We create a constant of any datatype using 'const' keyword. To create a constant, we prefix the variable declaration with 'const' keyword.
The general syntax for creating constant using 'const' keyword is as follows...

const datatype constantName ;

OR

const datatype constantName = value ;

Example

const int x = 10 ;

Here, 'x' is a integer constant with fixed value 10.

Example Program

#include<stdio.h>

#include<conio.h>

void main(){

   int i = 9 ;

   const int x = 10 ;

```
  i = 15 ;
  x = 100 ; // creates an error
  printf("i = %d\nx = %d", i, x ) ;


}
```

Using '#define' preprocessor
We can also create constants using '#define' preprocessor directive. When we create constant using this preprocessor directive it must be defined at the beginning of the program (because all the preprocessor directives must be written before the global declaration).
We use the following syntax to create constant using '#define' preprocessor directive...

#define CONSTANTNAME value

Example
#define PI 3.14
Here, **PI** is a constant with value **3.14**
Example Program

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

#defien PI  3.14

void main(){
  int r, area ;
    printf("Please enter the radius of circle : ") ;
  scanf("%d", &r) ;
    area = PI * (r * r) ;
    printf("Area of the circle = %d", area) ;
}
```

C Operators

An operator is a symbol used to perform arithmetic and logical operations in a program. That means an operator is a special symbol that tells the compiler to perform mathematical or logical operations. C programming language supports a rich set of operators that are classified as follows.

1. **Arithmetic Operators**
2. **Relational Operators**
3. **Logical Operators**

4. **Increment & Decrement Operators**
5. **Assignment Operators**
6. **Bitwise Operators**
7. **Conditional Operator**
8. **Special Operators**

Arithmetic Operators (+, -, *, /, %)

Operator	Meaning	Example
+	Addition	10 + 5 = 15
-	Subtraction	10 - 5 = 5
*	Multiplication	10 * 5 = 50
/	Division	10 / 5 = 2
%	Remainder of the Division	5 % 2 = 1

Relational Operators (<, >, <=, >=, ==, !=)

Operator	Meaning	Example
<	Returns TRUE if the first value is smaller than second value otherwise returns FALSE	10 < 5 is FALSE
>	Returns TRUE if the first value is larger than second value otherwise returns FALSE	10 > 5 is TRUE
<=	Returns TRUE if the first value is smaller than or equal to second value otherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if the first value is larger than or equal to second value otherwise returns FALSE	10 >= 5 is TRUE
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 is TRUE

Logical Operators (&&, ||, !)

Operator	Meaning	Example
&&	**Logical AND** - Returns TRUE if all conditions are TRUE otherwise returns FALSE	10 < 5 && 12 > 10 is FALSE
||	**Logical OR** - Returns FALSE if all conditions are FALSE otherwise returns TRUE	10 < 5 || 12 > 10 is TRUE
!	**Logical NOT** - Returns TRUE if condition	!(10 < 5 && 12 >

	is FLASE and returns FALSE if it is TRUE	10) is TRUE

☐ **Logical AND -** Returns TRUE only if all conditions are TRUE, if any of the conditions is FALSE then complete condition becomes FALSE.

☐ **Logical OR -** Returns FALSE only if all conditions are FALSE, if any of the conditions is TRUE then complete condition becomes TRUE.

Implement a program to demonstrate logical AND operator.

```
#include<stdio.h>

#include<conio.h>

void main()

{

int i;

int j;

clrscr();

printf("Enter the values of i and j");

scanf("%d%d",&i,&j);

if(i==5 && j==5) {

printf("Both i and j are equal to 5");

} else {

printf("Both the values are different and either or both are not

equal to 5");

}

getch();

}
```

Increment & Decrement Operators (++ & --)

Operator	Meaning	Example
++	**Increment** - Adds one to existing value	int a = 5; a++; ⇒ a = 6
--	**Decrement** - Subtracts one from existing value	int a = 5; a--; ⇒ a = 4

Explain increment and decrement operator.
Increment operator is used to increment or increase the value of a variable by one. It is equivalent to adding one to the value of the variable. The symbol used is ++. The decrement operator is used to decrement or decrease the value of variable by 1. It is equivalent to subtracting one from the value of the variable.

The symbol used is --.

Syntax: ++var or var++ for increment and --var or var—for decrement.

Example:
int m=5;
int n = ++m;
printf(%d%d",m,n);

When the increment operator is used prior to the variable name m, the value of the variable m is incremented first and then assigned to the variable n. The values of both the variable m and n here will be 6. But if the increment operator ++ is used after the variable name, then the value of the variable m is assigned to the variable n and then the value of m is increased. Therefore the values of m and n will be 6 and 5 respectively.

Example for decrement operator
int m=5;
int n=--m;

Pre Increment and Post Increment

Pre-increment unary operator:
Pre-increment unary operator is used to increment the value of variable by one before using in the expression. In the Pre-Increment operator concern value is first incremented and then it used inside the expression with final updated value.
Syntax of Pre-increment unary operator:
  - ++variable;
Example Pre-increment unary operator:
  - ++i; //it is equivalents to the i=i+1; or i+=1 ;

Programming Code 1:
```
#include <stdio.h>
int main()
{
int i=5;
printf("%d\n",++i);
return 0;
}
```
Output: 6

Post-increment unary operator:
In Post-increment first of all the loop executes then value of the concern variable is increments by 1 and return the value before present update state.

Syntax of Post-increment unary operator:

  - variable++;

Example Post-increment unary operator:

- ++i;

Programming Code 1:

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

Output: 5

Explanation: In the above example the value of 'i' is initialize by 5 after that applying the post increment operation on variable 'i' then it updated 6 but the output is print before update i.e. 5.

Decrement Operator in C Programming :

1. Decrement operator is used to decrease the current value of variable by subtracting integer 1.
2. Like Increment operator, decrement operator can be applied to only variables.
3. Decrement operator is denoted by –.

Different Types of Decrement Operation :

When decrement operator used in C Programming then it can be used as pre-decrement or post-decrement operator.

A. Pre Decrement Operator

Pre-decrement operator is used to decrement the value of variable before using in the expression. In the Pre-decrement value is first decremented and then used inside the expression.

```
b = --var;
```

Suppose the value of variable var is 10 then we can say that value of variable 'var' is firstly decremented then updated value will be used in the expression.

B. Post Decrement Operator

Post-decrement operator is used to decrement the value of variable immediatly after executing expression completely in which post decrement is used. In the Post-decrement old value is first used in a expression and then old value will be decrement by 1.

```
b = var--;
```

Value of variable 'var' is 5. Same value will be used in expression and after execution of expression new value will be 4.

CProgram

```c
#include<stdio.h>

void main()
{
int a,b,x=10,y=10;

a = x--;
b = --y;

printf("Value of a : %d",a);
printf("Value of b : %d",b);

}
```

Output :

```
Value of a : 10
Value of b : 9
```

Assignment Operators (=, +=, -=, *=, /=, %=)

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	A = 15
+=	Add both left and right-hand side values and store the result into left-hand side variable	A += 10 ⇒ A = A+10
-=	Subtract right-hand side value from left-hand side variable value and store the result into left-hand side variable	A -= B ⇒ A = A-B
*=	Multiply right-hand side value with left-hand side variable value and store the result into left-hand side variable	A *= B ⇒ A = A*B

/=	Divide left-hand side variable value with right-hand side variable value and store the result into the left-hand side variable	A /= B $\Rightarrow$ A = A/B
%=	Divide left-hand side variable value with right-hand side variable value and store the remainder into the left-hand side variable	A %= B $\Rightarrow$ A = A%B

Bitwise Operators (&, |, ^, ~, >>, <<)

Operator	**Meaning**	**Example**
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B $\Rightarrow$ 16 (10000)
		the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A | B $\Rightarrow$ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B $\Rightarrow$ 13 (01101)
~	the result of Bitwise once complement is negation of the bit (Flipping)	~A $\Rightarrow$ 6 (00110)
<<	the Bitwise left shift operator shifts all the bits to the left by the specified number of positions	A << 2 $\Rightarrow$ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specified number of positions	A >> 2 $\Rightarrow$ 6 (00110)

Conditional Operator (?:)

Condition ? TRUE Part : FALSE Part;

Example

A = (10<15)?100: 200; $\Rightarrow$ A value is 100

Explain conditional operator with example.

Conditional operators return one value if condition is true and returns another value is condition is false. This operator is also called as ternary operator as it takes three arguments.

Syntax :

(Condition? true_value: false_value);

Example:

#include<stdio.h>

#include<conio.h>

void main() {

```
int i;

clrscr();

printf("Enter a number:");

scanf("%d",&i);

i%2==0?printf("%d is even",i):printf("%d is odd",i) ;

getch();

}
```

Special Operators (sizeof, pointer, comma, dot, etc.)

sizeof operator

This operator is used to find the size of the memory (in bytes) allocated for a variable. This operator is used with the following syntax.

<div align="center">sizeof(variableName);</div>

Example

<div align="center">sizeof(A); ⇒ the result is 2 if A is an integer</div>

Pointer operator (*)

This operator is used to define pointer variables in c programming language.

Comma operator (,)

This operator is used to separate variables while they are declaring, separate the expressions in function calls, etc.

Dot operator (.)

This operator is used to access members of structure or union.

Type Conversion

The type conversion is the process of converting a data value from one data type to another data type automatically by the compiler. Sometimes type conversion is also called **implicit type conversion**.

```
int i = 10 ;
float x = 15.5 ;
char ch = 'A' ;
```

i = x ; =======> x value 15.5 is converted as 15 and assigned to variable i

x = i ; =======> Here i value 10 is converted as 10.000000 and assigned to variable x

i = ch ; =======> Here the ASCII value of A (65) is assigned to i
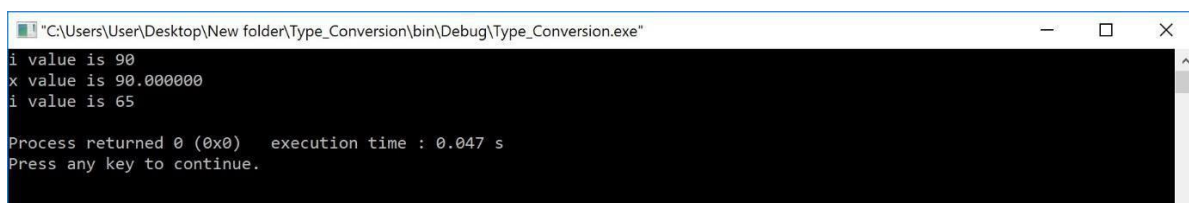
Example Program

```
#include<stdio.h>
#include<conio.h>
void main(){
  int i = 95 ;
  float x = 90.99 ;
  char ch = 'A' ;


  i = x ;
  printf("i value is %d\n",i);
  x = i ;
  printf("x value is %f\n",x);
  i = ch ;
  printf("i value is %d\n",i);


}
```

Output:


```
"C:\Users\User\Desktop\New folder\Type_Conversion\bin\Debug\Type_Conversion.exe"          —     □     ×
i value is 90
x value is 90.000000
i value is 65

Process returned 0 (0x0)    execution time : 0.047 s
Press any key to continue.
```

Typecasting

Typecasting is also called an **explicit type conversion**. Compiler converts data from one data type to another data type implicitly. When compiler converts implicitly, there may be a data loss. In such a case, we convert the data from one data type to another data type using explicit type conversion. To perform this we use the **unary cast operator**. The general syntax of typecasting is as follows.

(TargetDatatype) DataValue

Example

int totalMarks = 450, maxMarks = 600 ;
float average ;
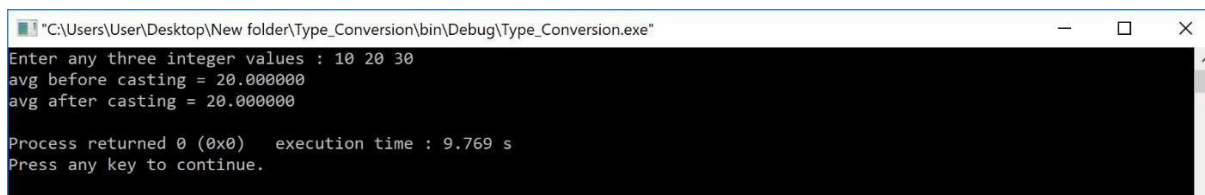
average = (float) totalMarks / maxMarks * 100 ;

In the above example code, both totalMarks and maxMarks are integer data values. When we perform totalMarks / maxMarks the result is a float value, but the destination (average) datatype is a float. So we use type casting to convert totalMarks and maxMarks into float data type.

Example Program

```c
#include<stdio.h>
#include<conio.h>
void main()
{
    int a, b, c ;
    float avg ;
    printf("Enter any three integer values : ") ;
    scanf("%d%d%d", &a, &b, &c) ;
        avg = (a + b + c) / 3 ;
    printf("avg before casting = %f\n",avg);


    avg = (float)(a + b + c) / 3 ;
    printf("avg after casting = %f\n",avg);
}
```

Output:

```
"C:\Users\User\Desktop\New folder\Type_Conversion\bin\Debug\Type_Conversion.exe"                    —    □    ×
Enter any three integer values : 10 20 30
avg before casting = 20.000000
avg after casting = 20.000000

Process returned 0 (0x0)    execution time : 9.769 s
Press any key to continue.
```

C Input Functions

C programming language provides built-in functions to perform input operations. The input operations are used to read user values (input) from the keyboard. The c programming language provides the following built-in input functions.

1. **scanf()**
2. **getchar()**
3. **getch()**
4. **gets()**
5. **fscanf()**

scanf() function

The scanf() function is used to read multiple data values of different data types from the keyboard. The scanf() function is built-in function defined in a header file called "**stdio.h**". When we want to use scanf() function in our program, we need to include the respective header file (stdio.h) using **#include** statement. The scanf() function has the following syntax...

Syntax:
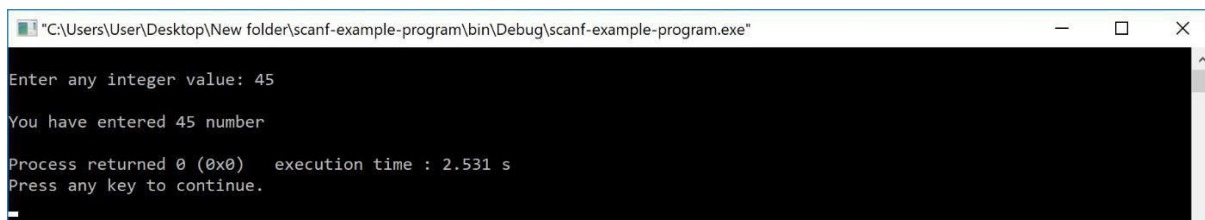
```
scanf("format strings",&variableNames);
```

Example Program

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

void main(){
   int i;
   printf("\nEnter any integer value: ");
   scanf("%d",&i);
   printf("\nYou have entered %d number",i);

}
```

Output:

In the above example program, we used the scanf() function to read an integer value from the keyboard and store it into variable 'i'.

The scanf function also used to read multiple data values of different or the same data types. Consider the following example program...

Example Program

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

void main(){
    int i;
    float x;
    printf("\nEnter one integer followed by one float value : ");
    scanf("%d%f",&i, &x);
    printf("\ninteger = %d, float = %f",i, x);

}
```

Output:

In the above example program, we used the scanf() function to read one integer value and one float value from the keyboard. Here **'i'** is an integer variable so we have used format string %d, and **'x'** is a float variable so we have used format string %f.

The scanf() function returns an integer value equal to the total number of input values read using scanf function.

Example Program

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

```
#include<conio.h>

void main(){
    int i,a,b;
    float x;
    printf("\nEnter two integers and one float : ");
    i = scanf("%d%d%f",&a, &b, &x);
    printf("\nTotal inputs read : %d",i);

}
```

Output:

getchar() function

The getchar() function is used to read a character from the keyboard and return it to the program. This function is used to read a single character. To read multiple characters we need to write multiple times or use a looping statement. Consider the following example program...
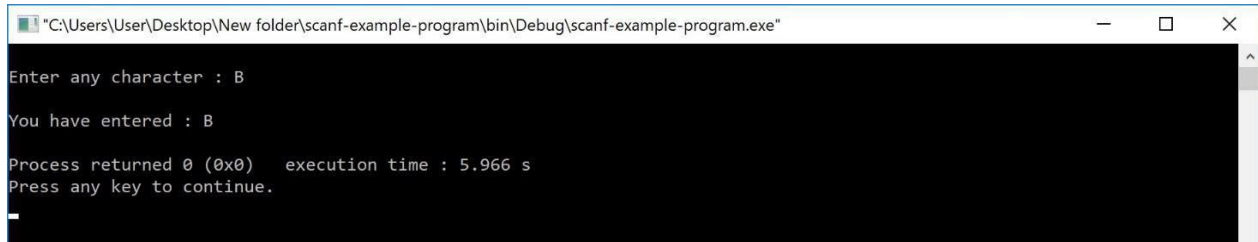
Example Program

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

void main(){
    char ch;
    printf("\nEnter any character : ");
    ch = getchar();
    printf("\nYou have entered : %c\n",ch);

}
```

Output:

getch() function

The getch() function is similar to getchar function. The getch() function is used to read a character from the keyboard and return it to the program. This function is used to read a single character. To read multiple characters we need to write multiple times or use a looping statement. Consider the following example program...
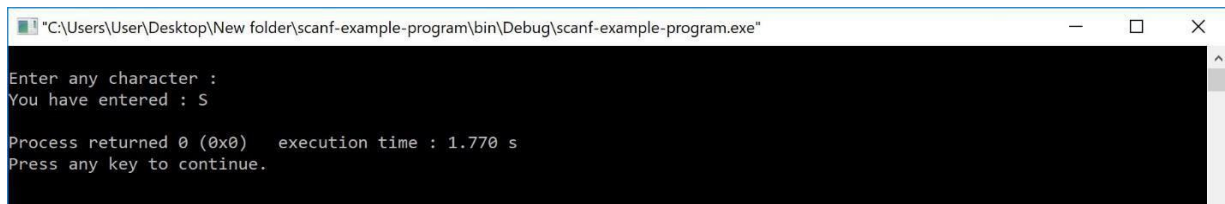
Example Program

#include<stdio.h>

#include<conio.h>

void main(){

   char ch;

   printf("\nEnter any character : ");

   ch = getch();

   printf("\nYou have entered : %c",ch);

}

Output:
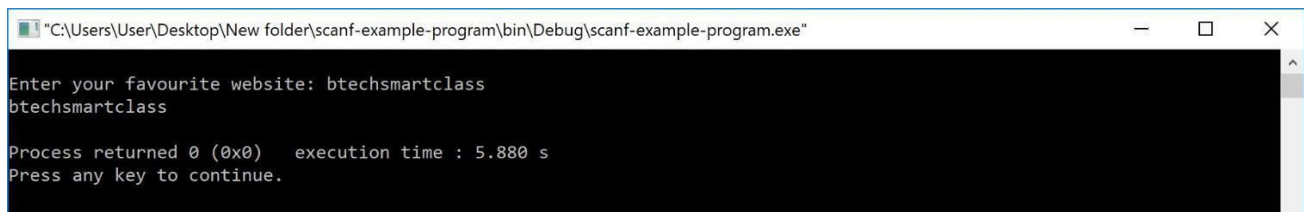
gets() function

The gets() function is used to read a line of string and stores it into a character array. The gets() function reads a line of string or sequence of characters till a newline symbol enters. Consider the following example program...

Example Program

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

void main(){
   char name[30];
   printf("\nEnter your favourite website: ");
   gets(name);
   printf("%s",name);
}
```

Output:

fscanf() function

The fscanf() function is used with the concept of files. The fscanf() function is used to read data values from a file. When you want to use fscanf() function the file must be opened in reading mode.

C Output Functions

C programming language provides built-in functions to perform output operation. The output operations are used to display data on user screen (output screen) or printer or any file. The c programming language provides the following built-in output functions...

1. **printf()**
2. **putchar()**
3. **puts()**
4. **fprintf()**

printf() function

The printf() function is used to print string or data values or a combination of string and data values on the output screen (User screen). The printf() function is built-in function defined in a header file called "**stdio.h**". When we want to use printf() function in our program we need to include the respective header file (stdio.h) using the **#include** statement. The printf() function has the following syntax...

Syntax:

```
printf("message to be display!!!");
```

Example Program

```
#include<stdio.h>
#include<conio.h>
void main(){
   printf("Hello! Welcome to btechsmartclass!!!");
}
```

Output:

In the above example program, we used the printf() function to print a string on to the output screen.

The printf() function is also used to display data values. When we want to display data values we use **format string** of the data value to be displayed.

Syntax:

```
printf("format string",variableName);
```

Example Program

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

```
  float x = 5.5;
  printf("%d %f",i, x);


}
```

Output:

```
 "C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"          —    □    ×
10 5.500000
Process returned 0 (0x0)     execution time : 0.047 s
Press any key to continue.
```

In the above example program, we used the printf() function to print data values of variables i and x on to the output screen. Here i is a an integer variable so we have used format string %d and x is a float variable so we have used format string %f.

The printf() function can also be used to display string along with data values.

Syntax:

```
printf("String format string",variableName);
```

Example Program

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

void main(){
  int i = 10;
  float x = 5.5;
  printf("Integer value = %d, float value = %f",i, x);


}
```

Output:

```
 "C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"          —    □    ×
Integer value = 10, float value = 5.500000
Process returned 0 (0x0)     execution time : 0.047 s
Press any key to continue.
```

In the above program, we are displaying string along with data values.

Every function in the C programming language must have a return value. The printf() function also have an integer as a return value. The printf() function returns an integer value equivalent to the total number of characters it has printed.

Example Program

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

void main(){
    int i;
    i = printf("btechsmartclass");
    printf(" is %d number of characters.",i);

}
```

Output:

In the above program, first printf() function printing "btechsmartclass" which is of 15 characters. So it returns integer value 15 to the variable "i". The value of "i" is printed in the second printf() function.

Formatted printf() function

Generally, when we write multiple printf() statements the result is displayed in a single line because the printf() function displays the output in a single line. Consider the following example program...

Example Program

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

void main(){
    printf("Welcome to ");
```

```
printf("btechsmartclass ");

printf("the perfect website for learning");
}
```

Output:


```
"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"                    —     □     ×
Welcome to btechsmartclass the perfect website for learning
Process returned 0 (0x0)    execution time : 0.031 s
Press any key to continue.
```

In the above program, there are 3 printf() statements written in different lines but the output is displayed in single line only.

To display the output in different lines or as we wish, we use some special characters called **escape sequences**. Escape sequences are special characters with special functionality used in printf() function to format the output according to the user requirement. In the C programming language, we have the following escape sequences...

Escape sequence	Meaning
\n	Moves the cursor to New Line
\t	Inserts Horizontal Tab (5 characters space)
\v	Inserts Vertical Tab (5 lines space)
\a	Beep sound
\b	Backspace (removes the previous character from its current position)
\\	Inserts Backward slash symbol
\?	Inserts Question mark symbol
\'	Inserts Single quotation mark symbol
\"	Inserts Double quotation mark symbol

Consider the following example program...

Example Program

```
#include<stdio.h>
#include<conio.h>
void main(){
  printf("Welcome to\n");
  printf("btechsmartclass\n");
  printf("the perfect website for learning");
}
```

Output:

```
"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"          —    □    ×
Welcome to
btechsmartclass
the perfect website for learning
Process returned 0 (0x0)    execution time : 0.047 s
Press any key to continue.
```

putchar() function

The putchar() function is used to display a single character on the output screen. The putchar() functions prints the character which is passed as a parameter to it and returns the same character as a return value. This function is used to print only a single character. To print multiple characters we need to write multiple times or use a looping statement. Consider the following example program...

Example Program

```c
#include<stdio.h>
#include<conio.h>
void main(){
   char ch = 'A';
   putchar(ch);
}
```

Output:

```
"C:\Users\User\Desktop\New folder\printf-Example\bin\Debug\printf-Example.exe"          —    □    ×
A
Process returned 0 (0x0)    execution time : 0.047 s
Press any key to continue.
```

puts() function

The puts() function is used to display a string on the output screen. The puts() functions prints a string or sequence of characters till the newline. Consider the following example program...

Example Program

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

```
    char name[30];

    printf("\nEnter your favourite website: ");

    gets(name);

    puts(name);

}
```

Output:

fprintf() function

The fprintf() function is used with the concept of files. The fprintf() function is used to print a line into the file. When you want to use fprintf() function the file must be opened in writting mode.

Comment

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.

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

/* Author: TechOnTheNet.com */

// Author: TechOnTheNet.com

State any four math functions with its use.
sqrt() - square root of an integer
abs() - absolute value of an integer
sin() - compute the sine value of an input value
cos()- compute the cosine value of an input value
pow()- compute the power of a input value
floor()- round down the input value
ceil()- round up the input value

Develop a simple 'C' program for addition and multiplication of two integer numbers.

```c
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,add,mul;
clrscr();
printf("Enter value for a and b:");
scanf("%d%d",&a,&b);
add=a+b;
mul=a*b;
printf("\nAddition of a and b=%d\n",add);
printf("\Multiplication of a and b=%d",mul);
getch();
}
```

Explain following functions:
getchar( )
putchar( )
getch( )
putch( )
with suitable examples.

getchar( ) -
It is function from stdio.h header file. This function is used to input a single character.
The enter key is pressed which is followed by the character that is typed. The character that is entered is echoed.
Syntax:
ch=getchar();
Example:
```c
void main()
{
char ch;
ch = getchar();
printf("Input Char Is :%c",ch);
}
```
During the program execution, a single character gets or read through the getchar(). The given value is displayed on the screen and the compiler waits for another character to be typed. If you press the enter key/any other characters and then only the given character is printed through the printf function.

putchar( ) -
It is used from standard input (stdio.h) header file. This function is the other side of getchar. A single character is displayed on the screen.
Syntax:
```c
putchar(ch);
void main()
{
char ch='a';
putchar(ch);
getch();
```

}
getch( ) -
It is used from the console (conio.h) header file. This function is used to input a single
character. The character is read instantly and it does not require an enter key to
be pressed. The character type is returned but it does not echo on the screen.
Syntax:
ch=getch();
Where, ch - assigned the character that is returned by getch().
void main()
{
char ch;
ch = getch();
printf("Input Char Is :%c",ch);
}
During the program execution, a single character gets or read through the getch(). The given
value is not displayed on the screen and the compiler does not wait for another character to
be typed. And then, the given character is printed through the printf function.
putch( )-
It is used from console input output header file (conio.h) This function is a counterpart of
getch(). Which means that it will display a single character on the screen.
The character that is displayed is returned.
Syntax:
putch(ch); Where, ch - the character that is to be printed.
void main()
{
char ch='a';
putch(ch)
}

State the use of printf( ) & scanf( ) with suitable example.
printf( ) & scanf( ):
printf() and scanf() functions are library functions in C programming language defined in
"stdio.h".

printf() function is used to print the character, string, float, integer, octal and hexadecimal
values onto the output screen.

scanf() function is used to read character, string, numeric data from keyboard. %d format
specifier is used in printf() and scanf() to specify the value of an integer variable.
%c is used to specify character, %f for float variable, %s for string variable, and %x for
hexadecimal variable.

Example:
#include<stdio.h>
#include<conio.h>
void main() {
int i;
clrscr();
printf("Enter a number");
scanf("%d",&i);
printf("Entered number is: %d",i);

```
getch();
}
```

Program to convert given number of days into months and days in c

```
main ()
  {
    int  months, days ;

    printf("Enter days\n") ;
    scanf("%d", &days) ;

    months = days / 30 ;
    days  = days % 30 ;

    printf("Months = %d  Days = %d", months, days) ;
  }
```

Output

```
Enter days
265
Months = 8  Days = 25

Enter days
364
Months = 12  Days = 4

Enter days
45
Months = 1  Days = 15
```

Program to calculate area of circle in c

```
#include<stdio.h>

int main() {
  float radius, area;

  printf("\nEnter the radius of Circle : ");
  scanf("%d", &radius);

  area = 3.14 * radius * radius;
  printf("\nArea of Circle : %f", area);

  return (0);

}
```
Output:
Enter the radius of Circle : 2.0

Area of Circle : 6.14

Program to find the maximum of three numbers

```c
#include <stdio.h>
int main()
{
   double n1, n2, n3;
   printf("Enter three different numbers: ");
   scanf("%lf %lf %lf", &n1, &n2, &n3);
   if( n1>=n2 && n1>=n3 )
      printf("%.2f is the largest number.", n1);
   if( n2>=n1 && n2>=n3 )
      printf("%.2f is the largest number.", n2);
   if( n3>=n1 && n3>=n2 )
      printf("%.2f is the largest number.", n3);
   return 0;
}
```

Program to check positive, negative or zero

```c
#include <stdio.h>

int main()
{
   int num;

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


   if(num > 0)
   {
      printf("Number is POSITIVE");
   }
   if(num < 0)
   {
      printf("Number is NEGATIVE");
   }
   if(num == 0)
   {
      printf("Number is ZERO");
   }

   return 0;
}
```

C program to convert temperature from degree fahrenheit to Celsius Temperature conversion formula

ormula to convert temperature from degree Fahrenheit to degree Celsius is given by -

$$°C = (°F - 32) * \frac{5}{9}$$

```c
/**
 * C program to convert temperature from degree fahrenheit to celsius
 */

#include <stdio.h>

int main()
{
    float celsius, fahrenheit;

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

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

    /* Print the value of celsius */
    printf("%.2f Fahrenheit = %.2f Celsius", fahrenheit, celsius);

    return 0;
}
```

Enter temperature in Fahrenheit: 205
205.00 Fahrenheit = 96.11 Celsius

C program to convert kilometers to miles, meters

```c
#include
#include
void main()
{
float km, m, miles;
clrscr();
printf("Enter the distance in kilometers : ");
scanf("%f",&km);
m = km * 1000;
printf("The equivalent distance in meters is : %f",m);

float miles = km / 1.6;
```

```
    printf("%f Miles", miles);

}
getch();
}
```

Program to convert temperature from degree Celsius to Kelvin

```
#include<stdio.h>

int main()
{
 float k, cel;
 printf("Enter the temperature in celsius: ");
 scanf("%f", &cel);

 k = (cel + 273.15 ) ; //temperature conversion formula
 printf("\nTemperature in Kelvin: %.f", k);

 return 0;}
```

Explain how formatted input can be obtain, give suitable example.
Formatted input:
When the input data is arranged in a specific format, it is called formatted input. scanf function is used for this purpose in C.

General syntax:
scanf("control string", arg1, arg2..);

Control string here refers to the format of the input data. It includes the conversion character %, a data type character and an optional number that specifies the field width. It also may contain new line character or tab. arg1, arg2 refers to the address of memory locations where the data should be stored.

Example:
```
scanf("%d",&num1);
Eg:
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
clrscr();
printf("Enter a number");
scanf("%d",&i);
printf("Entered number is: %d",i);
getch();
}
```

Decision making in C

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. C language handles decision-making by supporting the following statements,

- if statement
- switch statement
- conditional operator statement (? : operator)
- goto statement

State any four decision making statement.

Decision making statement:

1. if statement

2. if-else statement

3. if-else-if ladder

4. Nested if-else statement

5. switch statement

6. conditional operator statement (? : operator)

Decision making with if statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

1. Simple if statement

2. if…else statement

3. Nested if .. else statement

4. Using else if statement

Simple if statement

The general form of a simple if statement is,

```
if(expression)
{
    statement inside;
}
    statement outside;
```

Example:

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

```
{
    int x, y;
    x = 15;
    y = 13;
    if (x > y )
    {
        printf("x is greater than y");
    }
}
```
Output:
x is greater than y
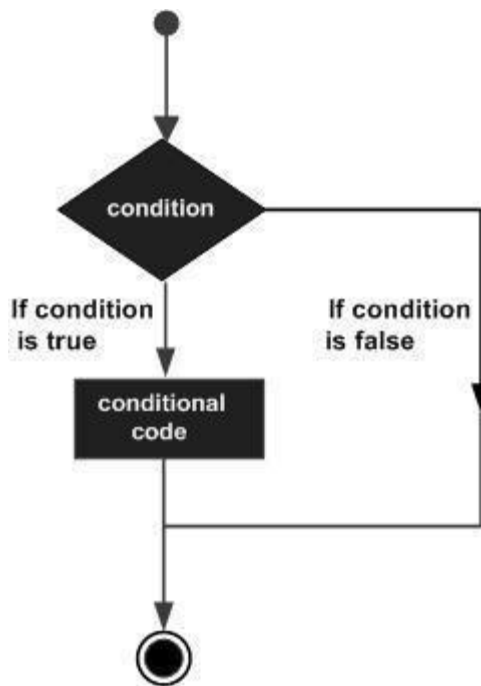
if...else statement

The general form of a simple if...else statement is,

```
if(expression)
{
    statement block1;
}
else
{
    statement block2;
}
```

If the expression is true, the statement-block1 is executed, else statement-block1 is skipped and statement-block2 is executed.

Example:

```c
#include <stdio.h>
void main( )
{
   int x, y;
   x = 15;
   y = 18;
   if (x > y )
   {
     printf("x is greater than y");
   }
   else
   {
     printf("y is greater than x");
   }
}
```
Output:
y is greater than x

Nested if... else statement
The general form of a nested if . else statement is,

```c
if( expression )
{
   if( expression1 )
   {
     statement block1;
   }
   else
   {
     statement block2;
   }
}
else
{
   statement block3;
}
```

Example:
```c
#include <stdio.h>

void main( )
{
   int a, b, c;
```

```
  printf("Enter 3 numbers...");
  scanf("%d%d%d",&a, &b, &c);
  if(a > b)
  {
    if(a > c)
    {
      printf("a is the greatest");
    }
    else
    {
      printf("c is the greatest");
    }
  }
  else
  {
    if(b > c)
    {
      printf("b is the greatest");
    }
    else
    {
      printf("c is the greatest");
    }
  }
}
```

C Program Enter the Student Marks and Find the Percentage and Grade

```
#include<stdio.h>
void main()
{
    int m1,m2,m3,total;
    float per;
    clrscr();
    printf("Enter 3 Nos.");
    scanf("%D%D%D",&m1,&m2,&m3);
    total=m1+m2+m3;
    per=total*100/300;
    if(per>=60&&per<=100)
        printf("You are 1st :");
    else if(per>=50&&per<=60)
        printf("You are 2nd");
    else if(per>=40&&per<=50)
        printf("You are 3rd");
    else
```

> *printf("You are Fail");*
> *getch();*
> *}*

```
Turbo C++ IDE
Enter 3 Nos. 80 50 75
You are 1st :_
```

Switch statement in C

The switch statement allows us to execute one code block among many alternatives.

Syntax of switch...case

```
switch (expression)
{
    case constant1:
     // statements
     break;

    case constant2:
     // statements
     break;
    .
    .
    .
    default:
     // default statements
}
```

switch Statement Flowchart

```c
// Program to create a simple calculator
#include <stdio.h>
int main()  {
    char operator;
    double n1, n2;
    printf("Enter an operator (+, -, *, /): ");
    scanf("%c", &operator);
    printf("Enter two operands: ");
    scanf("%lf %lf",&n1, &n2);
    switch(operator)
    {
        case '+':
            printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
            break;
```

```
        case '-':
            printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
            break;
        case '*':
            printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
            break;
        case '/':
            printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
            break;
        // operator doesn't match any case constant +, -, *, /
        default:
            printf("Error! operator is not correct");
    }
    return 0;
}
```

Output

Enter an operator (+, -, *,): -

Enter two operands: 32.5

12.4

32.5 - 12.4 = 20.1
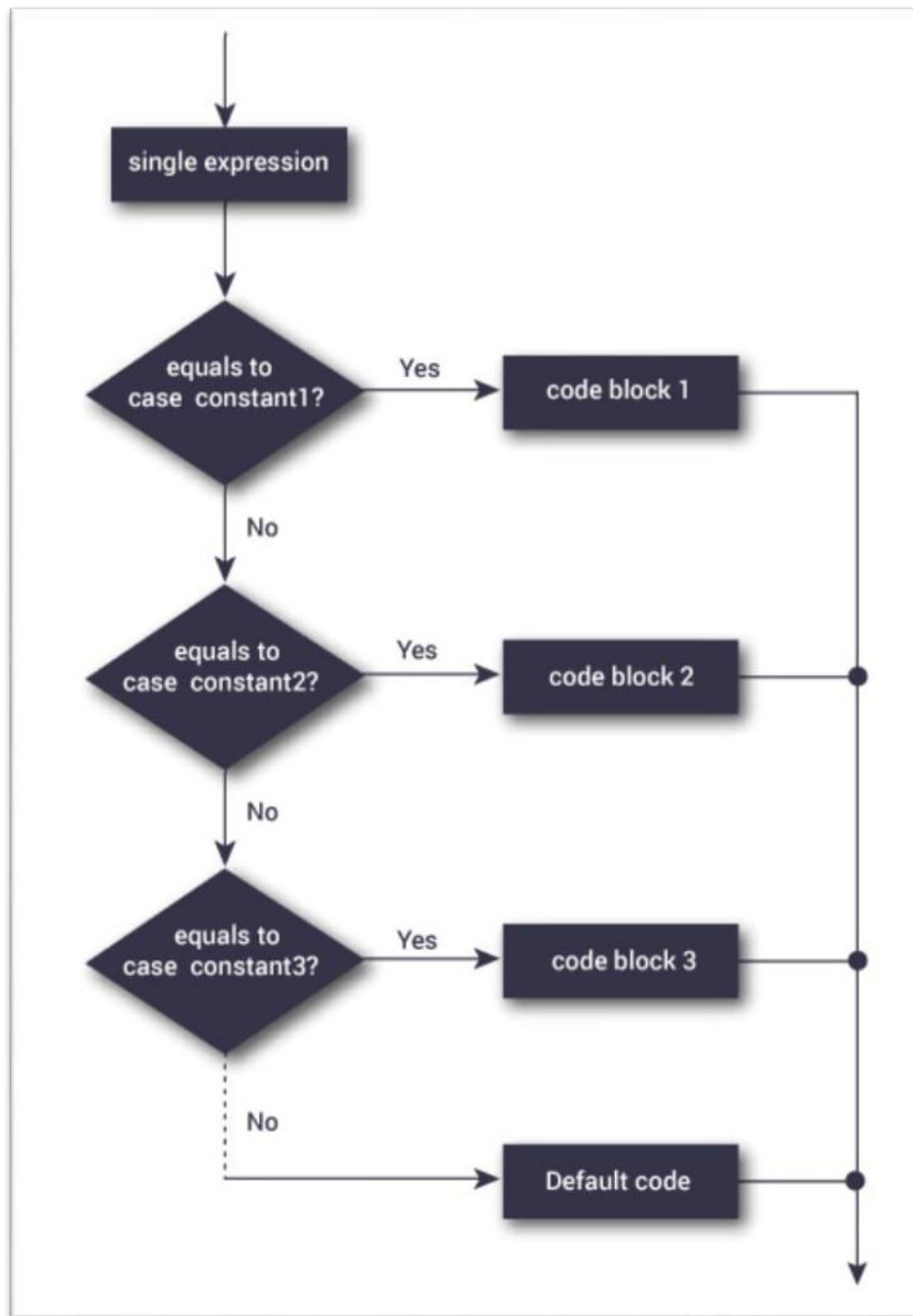
Example of switch statement
```c
#include<stdio.h>
void main( )
{
    int a, b, c, choice;
    while(choice != 3)
    {
        /* Printing the available options */
        printf("\n 1. Press 1 for addition");
```

```c
      printf("\n 2. Press 2 for subtraction");
      printf("\n Enter your choice");
      /* Taking users input */
      scanf("%d", &choice);

      switch(choice)
      {
        case 1:
          printf("Enter 2 numbers");
          scanf("%d%d", &a, &b);
          c = a + b;
          printf("%d", c);
          break;
        case 2:
          printf("Enter 2 numbers");
          scanf("%d%d", &a, &b);
          c = a - b;
          printf("%d", c);
          break;
        default:
          printf("you have passed a wrong key");
          printf("\n press any key to continue");
      }
   }
}
```

if...else Ladder
The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities. The if...else ladder allows you to check between multiple test expressions and execute different statements.

Syntax of nested if...else statement.
```c
   if (test expression1) {
     // statement(s)
   }
   else if(test expression2) {
     // statement(s)
   }
   else if (test expression3) {
     // statement(s)
   }
   .
   .
   else {
     // statement(s)
```

```
    }

    Example 3: C if...else Ladder
    // Program to relate two integers using =, > or < symbol

    #include <stdio.h>
    int main() {
      int number1, number2;
      printf("Enter two integers: ");
      scanf("%d %d", &number1, &number2);

      //checks if the two integers are equal.
      if(number1 == number2) {
        printf("Result: %d = %d",number1,number2);
      }

      //checks if number1 is greater than number2.
      else if (number1 > number2) {
        printf("Result: %d > %d", number1, number2);
      }

      //checks if both test expressions are false
      else {
        printf("Result: %d < %d",number1, number2);
      }

      return 0;
    }
```

Output
Enter two integers: 12
23
Result: 12 < 23

write a program to check whether given year is leap or not

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

  printf("Enter year: ");
  scanf("%d",&y);

  if(y % 4 == 0)
  {
      //Nested if else
```

```c
    if( y % 100 == 0)
    {
      if ( y % 400 == 0)
        printf("%d is a Leap Year", y);
      else
        printf("%d is not a Leap Year", y);
    }
    else
      printf("%d is a Leap Year", y );
  }
  else
    printf("%d is not a Leap Year", y);

  return 0;
}
```
Output:
Enter year: 1991
1991 is not a Leap Year

write a program to check whether the string is palindrome or not

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

    int main()
    {
     char a[100], b[100];

     printf("Enter a string to check if it is a palindrome\n");
     gets(a);

     strcpy(b, a); // Copying input string
     strrev(b); // Reversing the string

     if (strcmp(a, b) == 0) // Comparing input string with the reverse string
       printf("The string is a palindrome.\n");
     else
       printf("The string isn't a palindrome.\n");

     return 0;
    }
```
C program for palindrome without using string functions
```c
    #include <stdio.h>

    int main()
```

```c
{
 char text[100];
 int begin, middle, end, length = 0;

 gets(text);

 while (text[length] != '\0')
  length++;

 end = length - 1;
 middle = length/2;

 for (begin = 0; begin < middle; begin++)
 {
  if (text[begin] != text[end])
  {
   printf("Not a palindrome.\n");
   break;
  }
  end--;
 }
 if (begin == middle)
  printf("Palindrome.\n");

 return 0;
}
```

C program to find the largest of three numbers using Conditional Operator

```c
#include<stdio.h>
 int main()
   {
   int x, y, z, large;
   clrscr();
   printf(" Enter any three integer numbers for x, y, z : ") ;
   scanf("%d %d %d", &x, &y, &z) ;
   large = x > y ? ( x > z ? x : z) : (y > z ? y : z) ;
   printf("\n\n Largest  or biggest or greatest or maximum among 3 numbers using
Conditional ternary Operator : %d", large) ;
   getch();
   return 0;
   }
```

Sample Output:

Enter any three integer numbers for x, y, z : 536 234 782

Largest or biggest or greatest or maximum among 3 numbers using Conditional ternary
Operator : 782

C program to check whether a character is vowel or consonant

```
#include <stdio.h>

int main()
{
  char ch;

  printf("Enter a character\n");
  scanf("%c", &ch);

  // Checking both lower and upper case, || is the OR operator

  if (ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E' || ch == 'i' || ch == 'I' || ch =='o' || ch=='O' || ch
  == 'u' || ch == 'U')
    printf("%c is a vowel.\n", ch);
  else
    printf("%c isn't a vowel.\n", ch);

  return 0;
}
```

C program to print day of week name using switch case
```
/**
 * C program to print day of week using switch case
 */

#include <stdio.h>

int main()
{
  int week;

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

  switch(week)
```

```
   {
      case 1:
        printf("Monday");
        break;
      case 2:
        printf("Tuesday");
        break;
      case 3:
        printf("Wednesday");
        break;
      case 4:
        printf("Thursday");
        break;
      case 5:
        printf("Friday");
        break;
      case 6:
        printf("Saturday");
        break;
      case 7:
        printf("Sunday");
        break;
      default:
        printf("Invalid input! Please enter week number between 1-7.");
   }

   return 0;
}
```

Output
Enter week number(1-7): 1
Monday

C PROGRAM FOR GRADING SCHEME USING SWITCH CASE
Write a program to find grading scheme using (switch statement):

```
#include<stdio.h>
#include<conio.h>
int main()
{
int marks;
clrscr();
printf("Enter the marks of the students");
scanf("%d",&marks);
marks=marks/10;
switch(marks)
```

```
{
case 10:
case 9:
case 8:
    printf("The grade is Honors");
    break;
case 7:
case 6:
    printf("The grade is 1st Division");
    break;
case 5:
    printf("The grade is 2nd Division");
    break;
case 4:
    printf("The grade is 3rd Division");
    break;
default:
    printf("Student is failed");
}
getch();
return(0);
}
```

OUTPUT:
Enter the marks of the student
72
The grade is 1st Division

Find triangle is equilateral, isosceles or right angled Code in C Language

```
#include "stdio.h"
void main()
{
int i, j;
int temp;
int sides[3];

int type;
int isRightAngled;
clrscr();

printf("Please enter the three sides of triange\n");

for(i = 0; i < 3; i++)
{
```

```c
scanf("%d", &sides[i]);
}

for(i = 0; i < 3; i++)
{
for(j = i + 1; j < 3; j++)
{

if(sides[i] > sides[j])
{

temp = sides[i];
sides[i] = sides[j];
sides[j] = temp;
}
}
}

type = 3;
isRightAngled = 0;

if(sides[2] > sides[0] + sides[1]) /*check if triangle is valid */
type = 0;
else if(sides[0] == sides[2])
type = 1;
else
{
if(sides[0] == sides[1])
type = 2;

if(sides[2] * sides[2] == sides[0] * sides[0] + sides[1] * sides[1])
isRightAngled = 1;
}

switch(type)
{
case 0:
printf("The triangle is Invalid\n");
break;
case 1:
printf("The triangle is Equilateral Triangle\n");
break;
case 2:
if(isRightAngled == 1)
printf("The triangle is Isosceles and Right Angled Triangle\n");
else
```

```c
printf("The triangle is Isosceles Triangle\n");
break;
case 3:
if(isRightAngled == 1)
printf("The triangle is Scalene and Right Angled Triangle\n");
else
printf("The triangle is Scalene Triangle\n");
break;

}
getch();
}
```

Print the season name of the year based on the month number

```c
#include <stdio.h>

int main()
{
        int month;

        printf("Enter month: ");
        scanf("%d",&month);

        switch(month)
        {

                case 12:
                case  1:
                case  2:
                        printf("\n Winter");
                        break;
                case 3:
                case  4:
                case  5:
                        printf("\n Spring");
                        break;
                case 6:
                case 7:
                case 8:
                        printf("\n Summer");
                        break;
                case 9:
                case 10:
                case 11:
```

```
                printf("\n Autumn");
                break;

        default:
                printf("\n Invalid Month number");
                break;
    }

    return 0;


}
```

C For loop

Syntax of for loop:
for (initialization; condition test; increment or decrement)
{
    //Statements to be executed repeatedly
}
Flow Diagram of For loop
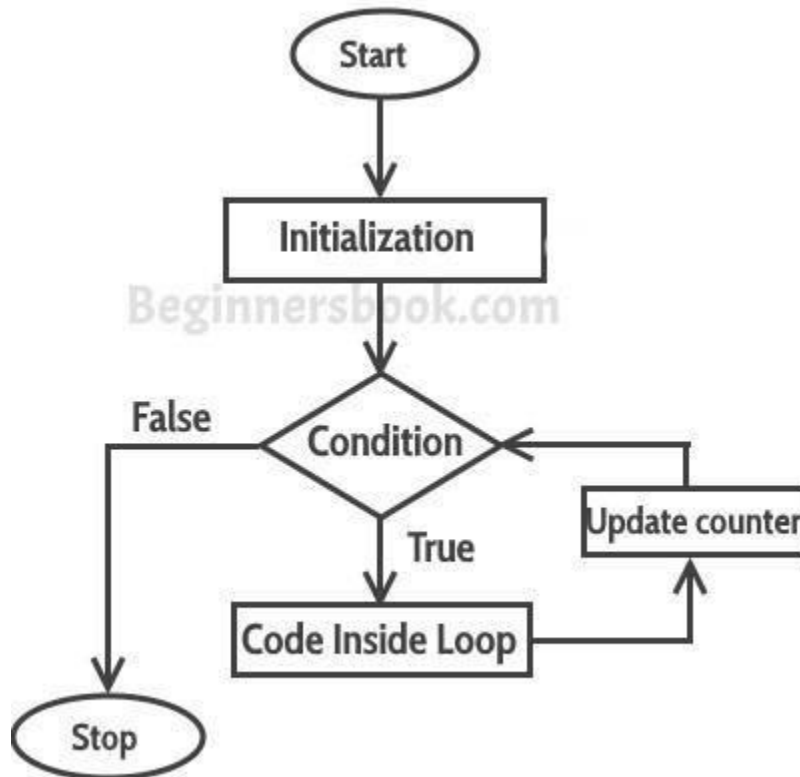
Step 1: First initialization happens and the counter variable gets initialized.
Step 2: In the second step the condition is checked, where the counter variable is tested for the given condition, if the condition returns true then the C statements inside the body of for loop gets executed, if the condition returns false then the for loop gets terminated and the control comes out of the loop.

Step 3: After successful execution of statements inside the body of loop, the counter variable is incremented or decremented, depending on the operation (++ or –).

Example of For loop

```c
#include <stdio.h>
int main()
{
  int i;
  for (i=1; i<=3; i++)
  {
     printf("%d\n", i);
  }
  return 0;
}
```

Output:

```
1
2
3
```

Design a program to print a message 10 times.

```c
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
clrscr();
for(i=0;i<10;i++)
{
printf("Welcome to C programming\n");
}
getch();
}
```

C – while loop

Syntax of while loop:

```c
while (condition test)
{
    //Statements to be executed repeatedly
    // Increment (++) or Decrement (--) Operation
}
```

Flow Diagram of while loop
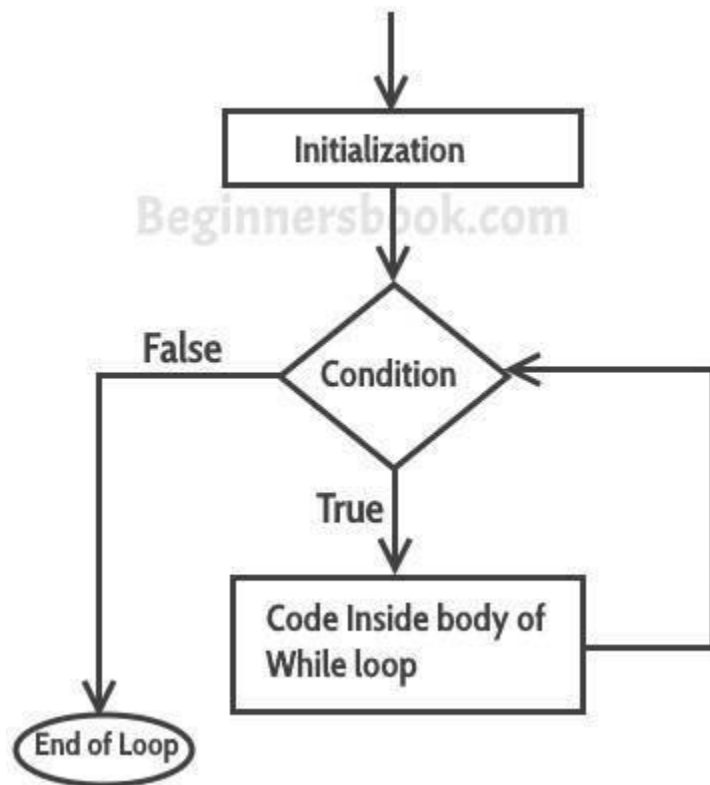
Example of while loop

```
// Print numbers from 1 to 5

#include <stdio.h>
int main()
{
   int i = 1;

   while (i <= 5)
   {
      printf("%d\n", i);
      ++i;
   }

   return 0;
}

Output
1
2
3
4
5
```

do...while loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated. The syntax of the do...while loop is:

```
do
{
   // statements inside the body of the loop
}
while (testExpression);
```

Flowchart of do...while Loop
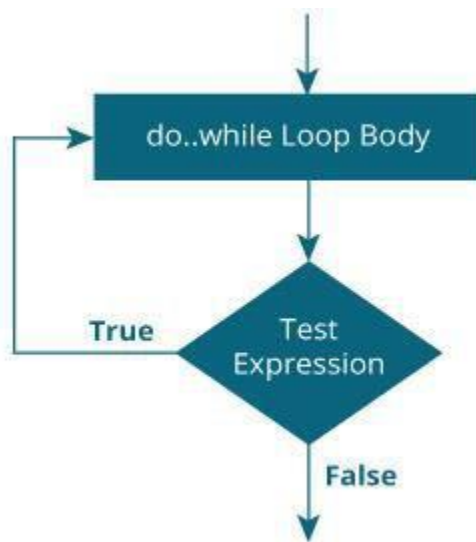
do..while loop

Syntax of do-while loop

```
do

{

   //Statements

}while(condition test);
```

Flow diagram of do while loop
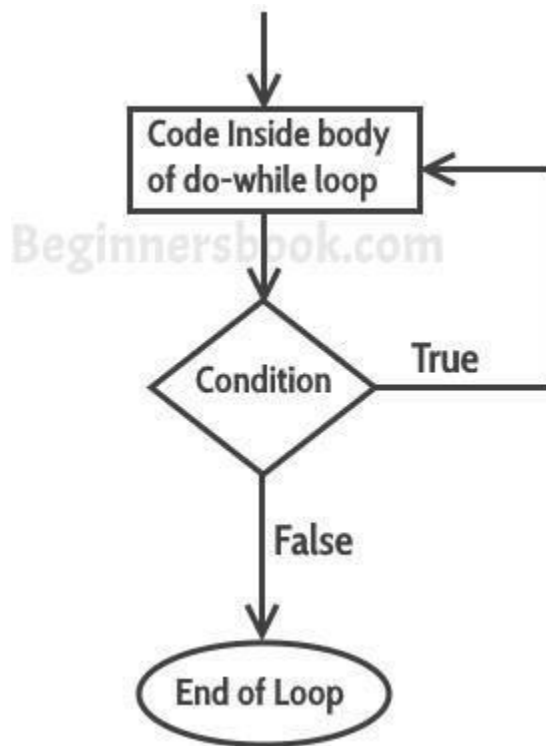
Example of do while loop

```
#include <stdio.h>
int main()
{
        int j=0;
        do
        {
                printf("Value of variable j is: %d\n", j);
                j++;
        }while (j<=3);
        return 0;
}
```
Output:
Value of variable j is: 0
Value of variable j is: 1
Value of variable j is: 2
Value of variable j is: 3

write a program to add numbers until user enters zero.
Program:-

```
#include<stdio.h>

#include<conio.h>

void main()
```

```
{

int no,sum=0;

clrscr();

do

{

printf("\n Enter a number:");

scanf("%d",&no);

sum=sum+no;

}while(no!=0);

printf("\n Sum of entered numbers =%d",sum);

getch();

}
```

C – goto statement

When a goto statement is encountered in a C program, the control jumps directly to the label
mentioned in the goto stateemnt
Syntax of goto statement in C
goto label_name;
..
..
label_name: C-statements
Flow Diagram of goto
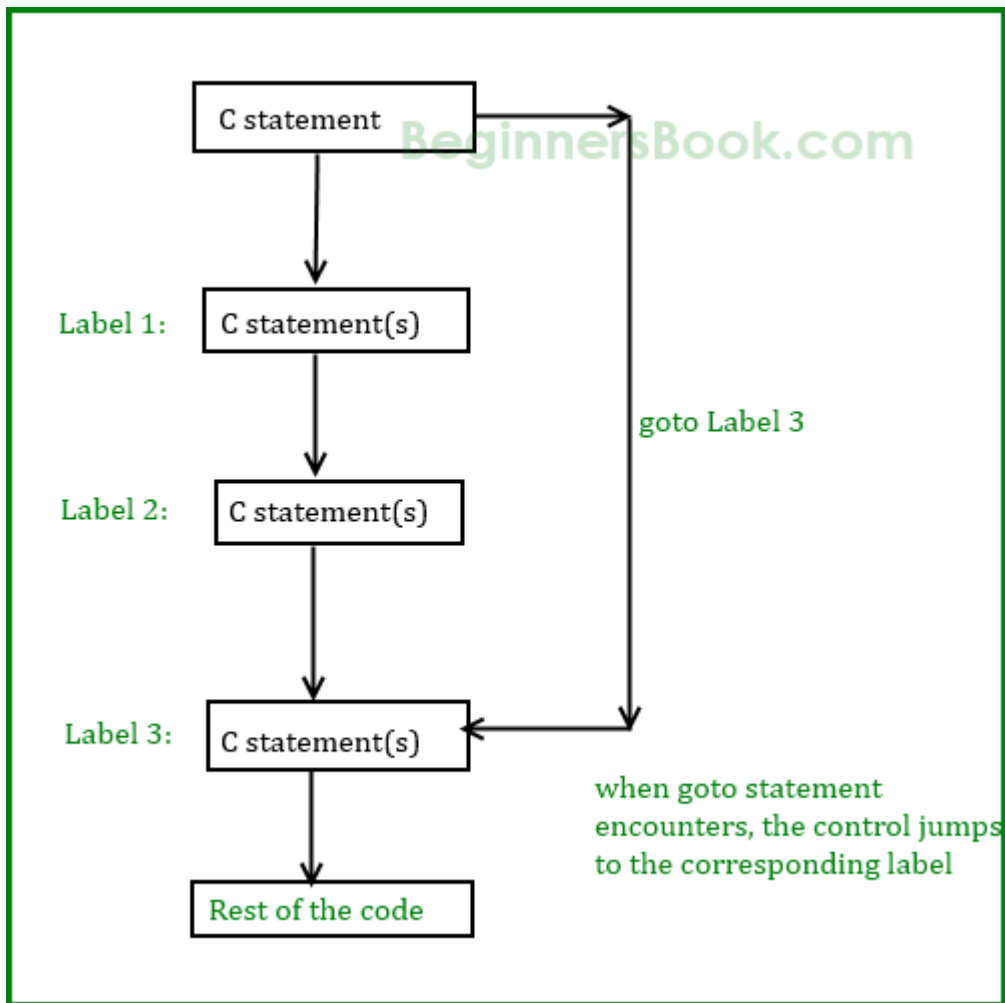
Example of goto statement
```
#include <stdio.h>
int main()
{
  int sum=0;
  for(int i = 0; i<=10; i++){
    sum = sum+i;
    if(i==5){
      goto addition;
    }
  }

  addition:
  printf("%d", sum);

  return 0;
}
```
Output:
15

C – Continue statement
Syntax:

continue;

Flow diagram of continue statement
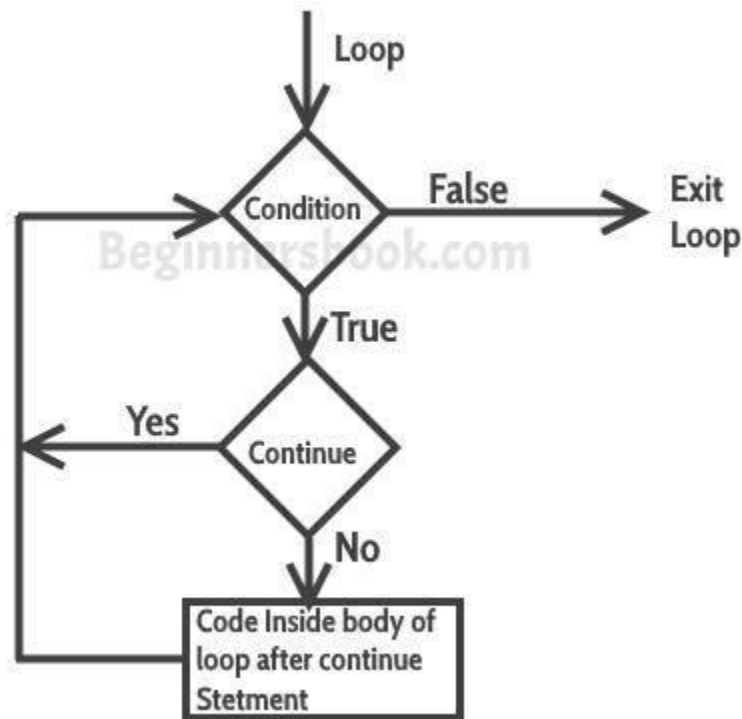
Example: continue statement inside for loop

```c
#include <stdio.h>
int main()
{
  for (int j=0; j<=8; j++)
  {
    if (j==4)
    {
        /* The continue statement is encountered when
         * the value of j is equal to 4.
         */
        continue;
    }

    /* This print statement would not execute for the
        * loop iteration where j ==4  because in that case
        * this statement would be skipped.
        */
    printf("%d ", j);
  }
  return 0;
```

}
Output:
0 1 2 3 5 6 7 8

C – break statement

1. It is used to come out of the loop instantly. When a break statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated. It is used with <u>if statement</u>, whenever used inside loop.
2. This can also be used in switch case control structure. Whenever it is encountered in switch-case block, the control comes out of the switch-case(see the example below).

Flow diagram of break statement
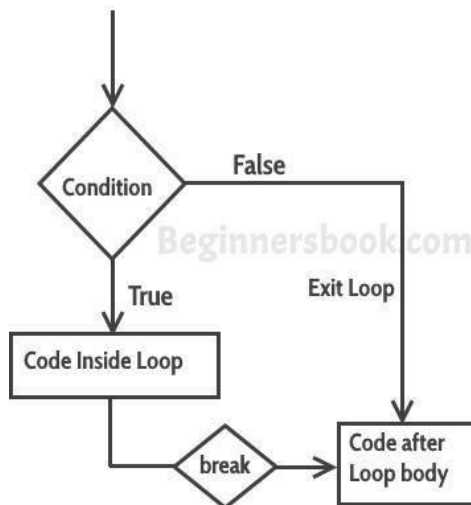
Syntax:
break;
Example – Use of break in a while loop
```
#include <stdio.h>
int main()
{
   int num =0;
   while(num<=100)
   {
     printf("value of variable num is: %d\n", num);
     if (num==2)
     {
        break;
     }
     num++;
   }
   printf("Out of while-loop");
   return 0;
}
```
Output:
value of variable num is: 0
value of variable num is: 1
value of variable num is: 2

Out of while-loop

c program to find sum of digits of a number

```c
#include <stdio.h>

int main()
{
  int n, t, sum = 0, remainder;

  printf("Enter an integer\n");
  scanf("%d", &n);

  t = n;

  while (t != 0)
  {
    remainder = t % 10;
    sum     = sum + remainder;
    t       = t / 10;
  }

  printf("Sum of digits of %d = %d\n", n, sum);

  return 0;
}
```



```
G:\C\add-digits.exe

Enter an integer
12345
Sum of digits of 12345 = 15
```
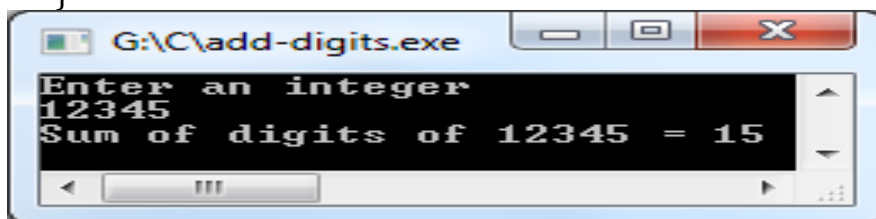
Write a program in C to display the multiplication table vertically from 1 to n.

```c
#include <stdio.h>
void main()
{
  int j,i,n=5;
  printf("Input upto the table number starting from 1 : ");
  scanf("%d",&n);
  printf("Multiplication table from 1 to %d \n",n);
  for(i=1;i<=10;i++)
  {
   for(j=1;j<=n;j++)
   {
    if (j<=n-1)
       printf("%dx%d = %d, ",j,i,i*j);
      else
          printf("%dx%d = %d",j,i,i*j);
```

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

Fibonacci Series in C without recursion

```c
#include<stdio.h>
int main()
{
 int n1=0,n2=1,n3,i,number;
 printf("Enter the number of elements:");
 scanf("%d",&number);
 printf("\n%d %d",n1,n2);//printing 0 and 1
 for(i=2;i<number;++i)//loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  printf(" %d",n3);
  n1=n2;
  n2=n3;
 }
 return 0;
}
```

Output:

Enter the number of elements:15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

     1

   2 3

  4 5 6

 7 8 9 10

11 12 13 14 15

#include<stdio.h>

int main()

{

 int i,j,k;

 k=1;

```c
for(i=1;i<=5;i++)
{
  for(j=5;j>=1;j--)
  {
    if(j > i)
    printf(" ");
    else
      printf("%d",k++);
  }
  printf("\n");
}
return 0; }
```

Define Array

Array is a collection of same data types.

They are declared by the given syntax:

Datatype array_name [dimensions] = {element1,element2,….,element}

The declaration form of one-dimensional array is

Data_type array_name [size];

e.g

int array[3] = {1,2,3);

Characteristics of Arrays in C

1) An array holds elements that have the same data type.

2) Array elements are stored in subsequent memory locations.

3) Two-dimensional array elements are stored row by row in subsequent memory locations.

4) Array name represents the address of the starting element.

5) Array size should be mentioned in the declaration. Array size must be a constant expression and not a variable.

Declaration of C Array

We can declare an array in the c language in the following way.

data_type array_name[array_size];

Now, let us see the example to declare the array.

int marks[5];

Here, int is the *data_type*, marks are the *array_name*, and 5 is the *array_size*.

Initialization of C Array

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;

80	60	70	85	75

marks[0]   marks[1]   marks[2]   marks[3]   marks[4]

Initialization of Array

Array example

```
#include<stdio.h>
int main(){
int i=0;
int marks[5];//declaration of array
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
//traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}//end of for loop
return 0;
}
```

Output

80

60

70

85

75

Array: Declaration with Initialization

We can initialize the c array at the time of declaration. Let's see the code.

   int marks[5]={20,30,40,50,60};

In such case, there is **no requirement to define the size**. So it may also be written as the following code.

   int marks[]={20,30,40,50,60};

Let's see the C program to declare and initialize the array in C.

#include<stdio.h>

int main(){

int i=0;

int marks[5]={20,30,40,50,60};//declaration and initialization of array

 //traversal of array

for(i=0;i<5;i++){

printf("%d \n",marks[i]);

}

return 0;

}

Output

20

30

40

50

60

Array Example: Sorting an array

In the following program, we are using bubble sort method to sort the array in ascending order.

```
#include<stdio.h>
void main ()
{
   int i, j,temp;
   int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
```

```
        for(i = 0; i<10; i++)
        {
          for(j = i+1; j<10; j++)
          {
            if(a[j] > a[i])
            {
              temp = a[i];
              a[i] = a[j];
              a[j] = temp;
            }
          }
        }
        printf("Printing Sorted Element List ...\n");
        for(i = 0; i<10; i++)
        {
          printf("%d\n",a[i]);
        }
      }
```

Two Dimensional Array in C

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array in C

The syntax to declare the 2D array is given below.

```
data_type array_name[rows][columns];
```

Consider the following example.

```
int twodimen[4][3];
```

Here, 4 is the number of rows, and 3 is the number of columns.

Initialization of 2D Array in C

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

```
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
```

Two-dimensional array example in C

```c
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
for(j=0;j<3;j++){
  printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

Output

arr[0][0] = 1

arr[0][1] = 2

arr[0][2] = 3

arr[1][0] = 2

arr[1][1] = 3

arr[1][2] = 4

arr[2][0] = 3

arr[2][1] = 4

arr[2][2] = 5

arr[3][0] = 4

arr[3][1] = 5

arr[3][2] = 6

2D array example: Storing elements in a matrix and printing it.

```c
#include <stdio.h>

void main ()
{
  int arr[3][3],i,j;
  for (i=0;i<3;i++)
  {
    for (j=0;j<3;j++)
    {
      printf("Enter a[%d][%d]: ",i,j);
      scanf("%d",&arr[i][j]);
    }
  }
  printf("\n printing the elements....\n");
  for(i=0;i<3;i++)
  {
    printf("\n");
    for (j=0;j<3;j++)
    {
      printf("%d\t",arr[i][j]);
    }
  }
}
```

Output

Enter a[0][0]: 56

Enter a[0][1]: 10

Enter a[0][2]: 30

Enter a[1][0]: 34

Enter a[1][1]: 21

Enter a[1][2]: 34

Enter a[2][0]: 45

Enter a[2][1]: 56

Enter a[2][2]: 78

 printing the elements ....

56   10   30

34   21   34

45   56   78

Addition of two matrix in C

```
#include <stdio.h>
 int main()
 {
   int m, n, c, d, first[10][10], second[10][10], sum[10][10];


   printf("Enter the number of rows and columns of matrix\n");
   scanf("%d%d", &m, &n);
   printf("Enter the elements of first matrix\n");


   for (c = 0; c < m; c++)
     for (d = 0; d < n; d++)
       scanf("%d", &first[c][d]);


   printf("Enter the elements of second matrix\n");


   for (c = 0; c < m; c++)
     for (d = 0 ; d < n; d++)
       scanf("%d", &second[c][d]);
```
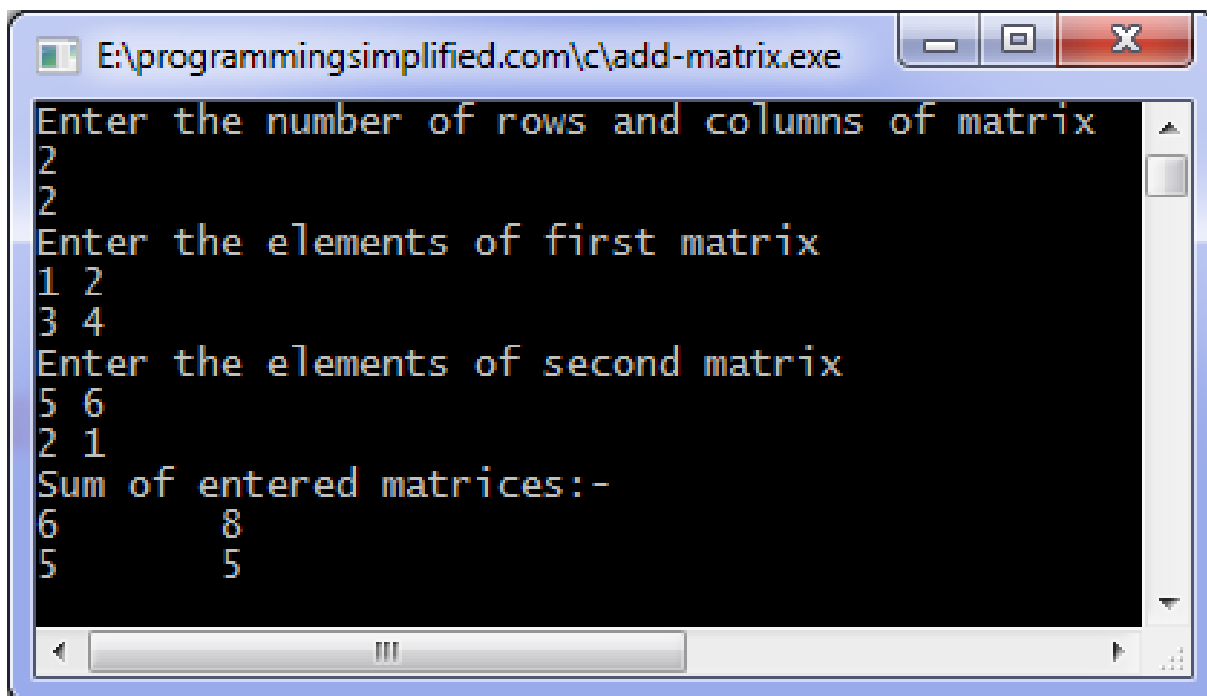
```
        printf("Sum of entered matrices:-\n");


    for (c = 0; c < m; c++) {

        for (d = 0 ; d < n; d++) {

            sum[c][d] = first[c][d] + second[c][d];

            printf("%d\t", sum[c][d]);

        }

        printf("\n");

    }

    return 0;

}
```


Enter the number of rows and columns of matrix
2
2
Enter the elements of first matrix
1 2
3 4
Enter the elements of second matrix
5 6
2 1
Sum of entered matrices:-
6   8
5   5

Advantages of Arrays

- Arrays represent multiple data items of the same type using a single name.
- In arrays, the elements can be accessed randomly by using the index number.
- Arrays allocate memory in contiguous memory locations for all its elements.
  Hence there is no chance of extra memory being allocated in case of arrays. This
  avoids memory overflow or shortage of memory in arrays.

Memory location (contiguous memory locations are allocated)

	100	101	102	103
Array	F	A	C	E
	0	1	2	3

→ Index
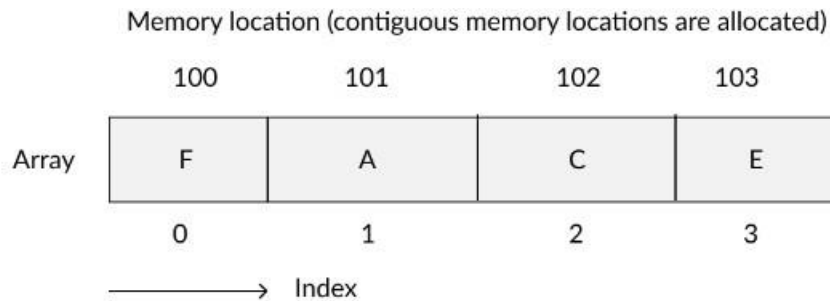
- Using arrays, other data structures like linked lists, stacks, queues, trees, graphs etc can be implemented.
- Two-dimensional arrays are used to represent matrices.

Disadvantages of Arrays

- The number of elements to be stored in an array should be known in advance.
- An array is a static structure (which means the array is of fixed size). Once declared the size of the array cannot be modified. The memory which is allocated to it cannot be increased or decreased.
- Insertion and deletion are quite difficult in an array as the elements are stored in consecutive memory locations and the shifting operation is costly.
- Allocating more memory than the requirement leads to wastage of memory space and less allocation of memory also leads to a problem.

Applications of Arrays

1) Array stores data elements ofthe same data type.

2) Maintains multiple variable names using a single name. Arrays help to maintain large data under a single variable name. This avoid the confusion of using multiple variables.

3) Arrays can be used for sorting data elements. Different sorting techniques like Bubble sort, Insertion sort, Selection sort etc use arrays to store and sort elements easily.

4) Arrays can be used for performing matrix operations. Many databases, small and large, consist of one-dimensional and two-dimensional arrays whose elements are records.

5) Arrays can be used for CPU scheduling.

6) Lastly, arrays are also used to implement other data structures like Stacks, Queues, Heaps, Hash tables etc.

Following operations can be performed on arrays:

1. Traversing

2. Searching

3. Insertion

4. Deletion

5. Sorting

6. Merging

1. Traversing: It is used to access each data item exactly once so that it can be processed.
E.g.
We have linear array A as below:

1	2	3	4	5
10	20	30	40	50

Here we will start from beginning and will go till last element and during this process we will access value of each element exactly once as below:

A [1] = 10
A [2] = 20
A [3] = 30
A [4] = 40
A [5] = 50

2. Searching: It is used to find out the location of the data item if it exists in the given collection of data items.
E.g.
We have linear array A as below:

1	2	3	4	5
15	50	35	20	25

Suppose item to be searched is 20. We will start from beginning and will compare 20 with each element. This process will continue until element is found or array is finished. Here:

1) Compare 20 with 15
20 # 15, go to next element.

2) Compare 20 with 50

20 # 50, go to next element.

3) Compare 20 with 35
20 #35, go to next element.

4) Compare 20 with 20
20 = 20, so 20 is found and its location is 4.

3. Insertion: It is used to add a new data item in the given collection of data items.
E.g.
We have linear array A as below:

1	2	3	4	5
10	20	50	30	15

New element to be inserted is 100 and location for insertion is 3. So shift the elements from 5th location to 3rd location downwards by 1 place. And then insert 100 at 3rd location. It is shown below:

1	2	3	4	5	6
10	20	50	30	15	15

1	2	3	4	5	6
10	20	50	30	30	15

1	2	3	4	5	6
10	20	50	50	30	15

1	2	3	4	5	6
10	20	100	50	30	15
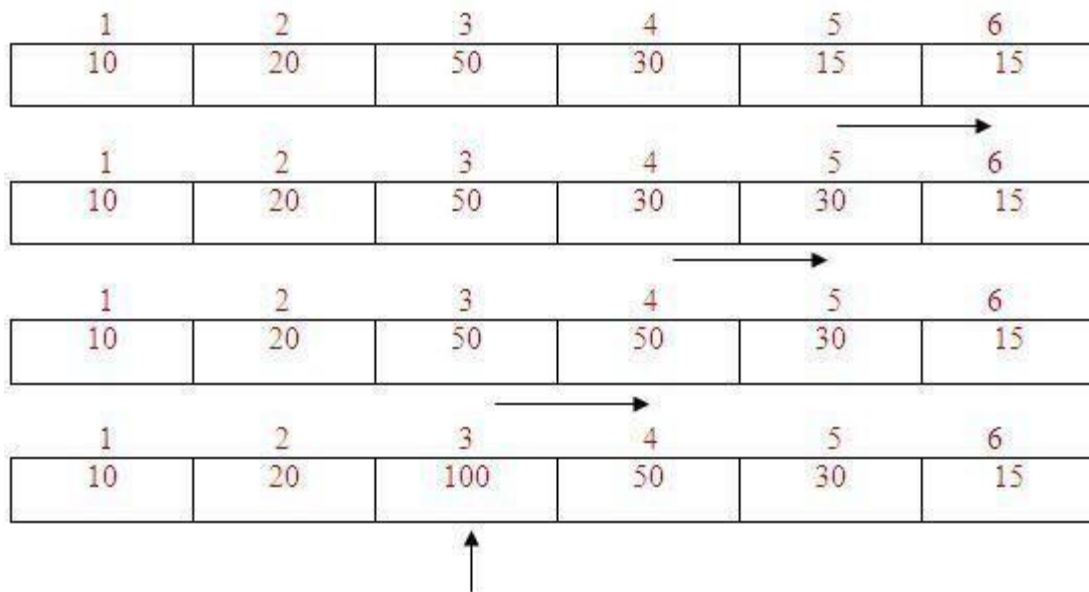
4. Deletion: It is used to delete an existing data item from the given collection of data items.
E.g.
We have linear array A as below:

1	2	3	4	5	
10	20	50	40	25	60

The element to be deleted is 50 which is at 3rd location. So shift the elements from 4th to 6th location upwards by 1 place. It is shown below:

1	2	3	4	5	6
10	20	40	40	25	60

1	2	3	4	5	6
10	20	40	25	25	60

1	2	3	4	5	6
10	20	40	25	60	60
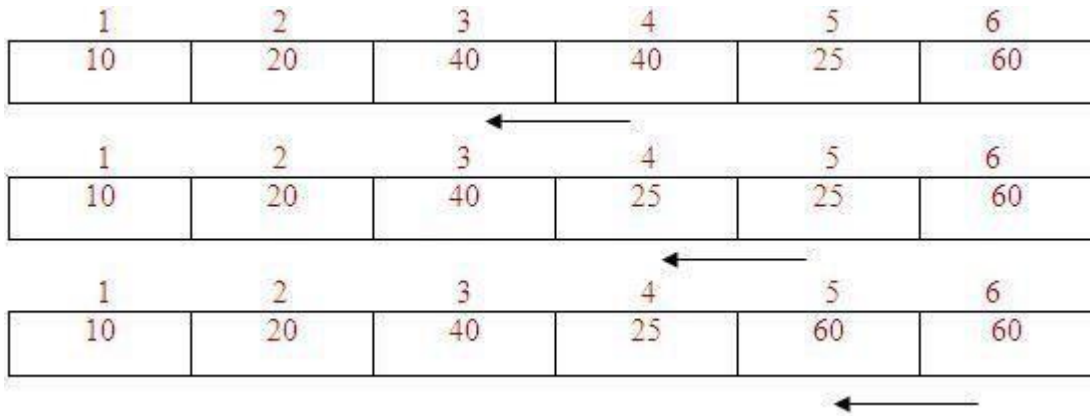
After deletion the array will be:

1	2	3	4	5	6
10	20	40	25	60	

5. Sorting: It is used to arrange the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.
E.g.
We have linear array A as below:

1	2	3	4	5
10	50	40	20	30

After arranging the elements in increasing order by using a sorting technique, the array will be:

1	2	3	4	5
10	20	30	40	50

6. Merging: It is used to combine the data items of two sorted files into single file in the sorted form
We have sorted linear array A as below:

1	2	3	4	5	6

10	40	50	80	95	100

And sorted linear array B as below:

1	2	3	4
20	35	45	90

After merging merged array C is as below:

1	2	3	4	5	6	7	8	9	10
10	20	35	40	45	50	80	90	95	100

Character array or String

A string is a collection of characters, stored in an array followed by null ('\0') character. Null character represents the end of string.
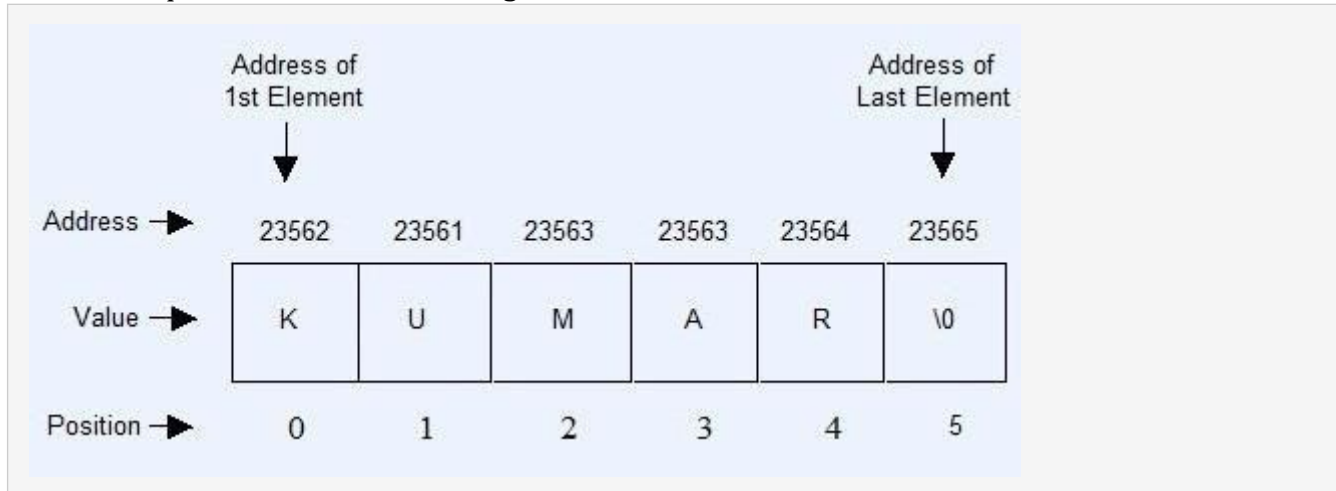
Address of first element is random, address of next element depend upon the type of array. Here, the type is character and character takes one byte in memory, therefore every next address will increment by one.

Index of array will always starts with zero.

Declaration of String or Character array

Declaration of array means creating sequential bolcks of memory to hold fixed number of values.

Syntax for string declaration :

```
                    char String-name[size of String ];
```

Example for string declaration :

```
            char String [25];                              //Statement 1
```

In the above example we have declared a character array which can hold twenty-five characters at a time.

Initialization of String

Initialization of string means assigning value to declared character array or string.
Examples 1 :Using sequence of characters.

```
    char String [ ] = {'H','e','l','l','o',' ','W','o','r','l','d','\0'};
```

In the above example, we are declaring and initializing a string at same time. When we declare and initialize a string at same time, giving the size of array is optional and **it is programmer's job to specify the null character at the end to terminate the string.**
Example 2 : Using assignment operator

```
    char String [ ] = "Hello World";
```

In this approch, programmer doesn't need to provide null character, compiler will automatically add null character at the end of string.

Input string using getche() function

The getche() read characters from console (output window) one by one and put these characters into string using loop.
Examples for input string with getche() function

```c
    #include<stdio.h>

    void main()
    {
        char String [50];
        char ch;
        int i;

        printf("\n\n\tEnter your name : ");
        for(i=0;i<50;i++)
        {
            ch = getche();       //Statement  1

            if(ch==13)           //Statement  2
            break;

            String [i] = ch;
        }
```

```
        String [i] = '\0';              //Statement  3

        printf("\n\n\tHello, ");
        for(i=0; String [i]!='\0';i++)
        printf("%c", String [i]);

    }
```

Output :

    Enter your name : Kumar
    Hello Kumar

In the above example, A for loop will execute upto 50 time and getche() which inside for loop will read single character from console and put the character into ch. If user press the enter key, condition given in statement 2 will satisfy and terminate the loop otherwise every character will assign to String. Statement 4 is assigning null character to String.

The scanf() can read the entire word at a time. The format specifier for a string is "%s". The scanf() ignores the any occurrence of leading whitespace characters and terminate the string at the first occurrence of whitespace after any input.

Examples for input string with scanf() function

```
    #include<stdio.h>

    void main()

    {

        char String [50];

        printf("\n\n\tEnter your name : ");

        scanf("%s", String );

        printf("\n\n\tHello %s", String );

    }
```

Output :

    Enter your name :   Kumar Wadhwa

    Hello Kumar

In the output of above example there is five leading whitespaces in the input which is ignored by the compiler.

Input string using gets() function

The gets() can read the entire line at a time and the string will not terminate until user press the enter key. The gets() will put all the leading and trailing whitespaces into str.

Examples for input string with gets() function

```
#include<stdio.h>

void main()

{

    char String [50];


    printf("\n\n\tEnter your name : ");

    gets( String );


    printf("\n\n\tHello %s", String );


}
```

Output :

Enter your name :   Kumar Wadhwa

Hello   Kumar Wadhwa

Differences between Strings and Character Arrays:

STRINGS	CHARACTER ARRAYS
String refers to a sequence of characters represented as a single data type.	Character Array is a sequential collection of data type char.
Strings are immutable.	Character Arrays are mutable.
Built in functions like substring(), charAt() etc can be used on Strings.	No built in functions are provided in Java for operations on Character Arrays.
'+' can be used to appended strings together to form a new string.	'+' cannot be used to append two Character Arrays.
The charAt() method can be used to access	The characters in a Character Array can

characters at a particular index in a String.	be accessed normally like in any other language by using [].
Strings can be stored in any any manner in the memory.	Elements in Character Array are stored contiguously in increasing memory locations.
All Strings are stored in the **String Constant Pool**.	All Character Arrays are stored in the **Heap**.
Not preferred for storing passwords in Java.	Preferred for storing passwords in Java.
A String can be converted into Character Array by using the toCharArray() method of String class. Eg: String s = "GEEKS"; char [] ch = s.toCharArray();	A Character Array can be converted into String by passing it into a **String Constructor**. Eg: char[] a = {'G', 'E', 'E', 'K', 'S'}; String A = new String(a);

C String Functions

There are many important string functions defined in "string.h" library.

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string, second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5)	strrev(string)	returns reverse string.
6)	strlwr(string)	returns string characters in lowercase.
7)	strupr(string)	returns string characters in uppercase.

Function	**C String Length: strlen() function**
Description	The strlen() function returns the length of the given string. It doesn't count null character '\0'.
| Input | ```c
#include<stdio.h>
#include <string.h>
int main(){
char ch[20]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
  printf("Length of string is: %d",strlen(ch));
 return 0;
}
``` |
| Output | Length of string is: 10 |

Function	**C Copy String: strcpy()**
Description	The strcpy(destination, source) function copies the source string in destination.
| Input | ```c
#include<stdio.h>
#include <string.h>
int main(){
 char ch[20]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
  char ch2[20];
  strcpy(ch2,ch);
  printf("Value of second string is: %s",ch2);
 return 0;
}
``` |
| Output | Value of second string is: javatpoint |

Function	**C String Concatenation: strcat()**
Description	The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.
| Input | ```c
#include<stdio.h>
#include <string.h>
int main(){
  char ch[10]={'h', 'e', 'l', 'l', 'o', '\0'};
  char ch2[10]={'c', '\0'};
  strcat(ch,ch2);
  printf("Value of first string is: %s",ch);
 return 0;
}
``` |
| Output | Value of first string is: helloc |

Function	**C Compare String: strcmp()**
Description	The strcmp(first_string, second_string) function compares two string and returns 0 if both strings are equal. Here, we are using *gets()* function which reads string from the console.
| Input | ```c
#include<stdio.h>
#include <string.h>
int main(){
  char str1[20],str2[20];
  printf("Enter 1st string: ");
  gets(str1);//reads string from console
  printf("Enter 2nd string:  ");
  gets(str2);
  if(strcmp(str1,str2)==0)
     printf("Strings are equal");
  else
     printf("Strings are not equal");
 return 0;

}
``` |
| Output | Enter 1st string: hello
Enter 2nd string: hello
Strings are equal |

Function	**C Reverse String: strrev()**
Description	The strrev(string) function returns reverse of the given string. Let's see a simple example of strrev() function.
| Input | ```c
#include<stdio.h>
#include <string.h>
int main(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nReverse String is: %s",strrev(str));
 return 0;
}
``` |
| Output | Enter string: javatpoint
String is: javatpoint
Reverse String is: tnioptavaj |

Function	**C String Lowercase: strlwr()**
Description	The strlwr(string) function returns string characters in lowercase. Let's see a simple example of strlwr() function.
| Input | ```c
#include<stdio.h>
#include <string.h>
int main(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nLower String is: %s",strlwr(str));
  return 0;
}
``` |
| Output | Enter string: JAVATpoint
String is: JAVATpoint
Lower String is: javatpoint |

Function	**C String Uppercase: strupr()**
Description	The strupr(string) function returns string characters in uppercase. Let's see a simple example of strupr() function.
| Input | ```c
#include<stdio.h>
#include <string.h>
int main(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nUpper String is: %s",strupr(str));
  return 0;
}
``` |
| Output | Enter string: javatpoint
String is: javatpoint
Upper String is: JAVATPOINT |

C Program to Find the Length of a String without using the Built-in Function

```c
#include<stdio.h>

#include<conio.h>

void main(){

char str[50];

int i, len=0;

clrscr();
```

```c
printf("Enter a string");

scanf("%s",&str);

for(i=0; str[i]!='\0'; i++){

len++;

}

printf("The length of string is %d",len);

getch();

}
```

C Program to Compare Two Strings Without Using Library Function

```c
#include<stdio.h>
 int main() {
   char str1[30], str2[30];
   int i;

   printf("\nEnter two strings :");
   gets(str1);
   gets(str2);

  i = 0;
  while (str1[i] == str2[i] && str1[i] != '\0')
    i++;
  if (str1[i] > str2[i])
    printf("str1 > str2");
  else if (str1[i] < str2[i])
    printf("str1 < str2");
  else
    printf("str1 = str2");
   return (0);
}
```

C Program to Copy One String into Other Without Using Library Function.

```c
#include<stdio.h>

int main() {
  char s1[100], s2[100];
  int i;

  printf("\nEnter the string :");
  gets(s1);

  i = 0;
  while (s1[i] != '\0') {
    s2[i] = s1[i];
    i++;
  }
  s2[i] = '\0';
  printf("\nCopied String is %s ", s2);
  return (0);
}
```

Output:

Enter the string : c4learn.com

Copied String is c4learn.com

C Program to Concat Two Strings without Using Library Function

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

void concat(char[], char[]);

int main() {
  char s1[50], s2[30];

  printf("\nEnter String 1 :");
  gets(s1);
  printf("\nEnter String 2 :");
  gets(s2);

  concat(s1, s2);
  printf("nConcated string is :%s", s1);

  return (0);
}
 void concat(char s1[], char s2[]) {
  int i, j;
  i = strlen(s1);
  for (j = 0; s2[j] != '\0'; i++, j++) {
    s1[i] = s2[j];
  }
  s1[i] = '\0';
}
```

Enter String 1 : Pritesh

Enter String 2 : Taral

Concated string is : PriteshTaral

Reverse String Without Using Library Function [Strrev]

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

int main() {
  char str[100], temp;
  int i, j = 0;

  printf("\nEnter the string :");
  gets(str);

  i = 0;
  j = strlen(str) - 1;

  while (i < j) {
    temp = str[i];
    str[i] = str[j];
    str[j] = temp;
    i++;
    j--;
  }

  printf("\nReverse string is :%s", str);
  return (0);
}
```

Output :

Enter the string : Pritesh

Reverse string is : hsetirP

What is Structure

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures ca; simulate the use of classes and templates as it can store various information

The **,struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memeberN;
};
```

Let's see the example to define a structure for an entity employee in c.

```
struct employee
{ int id;
    char name[20];
    float salary;
};
```

Declaring structure variable

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

1st way:

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

```
struct employee
{ int id;
    char name[50];
    float salary;
```

```
    };
```

Now write given code inside the main() function.

```
        struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

2nd way:

Let's see another way to declare variable at the time of defining the structure.

```
    struct employee
    {  int id;
        char name[50];
        float salary;
    }e1,e2;
```

Accessing members of the structure

There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)

Let's see the code to access the *id* member of *p1* variable by . (member) operator.

1. p1.id

C Structure example

Let's see a simple example of structure in C language.

```
    #include<stdio.h>
    #include <string.h>
    struct employee
    {  int id;
        char name[50];
    }e1;  //declaring e1 variable for structure
    int main( )
    {
        //store first employee information
        e1.id=101;
```

```
    strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array

    //printing first employee information

    printf( "employee 1 id : %d\n", e1.id);

    printf( "employee 1 name : %s\n", e1.name);

return 0;

}
```

Output:

employee 1 id : 101

employee 1 name : Sonoo Jaiswal

Let's see another example of the structure in C language to store many employees information.

```
#include<stdio.h>

#include <string.h>

struct employee

{  int id;

   char name[50];

   float salary;

}e1,e2; //declaring e1 and e2 variables for structure

int main( )

{

  //store first employee information

  e1.id=101;

  strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array

  e1.salary=56000;


  //store second employee information

  e2.id=102;

  strcpy(e2.name, "James Bond");

  e2.salary=126000;


  //printing first employee information
```

```
        printf( "employee 1 id : %d\n", e1.id);

        printf( "employee 1 name : %s\n", e1.name);

        printf( "employee 1 salary : %f\n", e1.salary);


        //printing second employee information

        printf( "employee 2 id : %d\n", e2.id);

        printf( "employee 2 name : %s\n", e2.name);

        printf( "employee 2 salary : %f\n", e2.salary);

        return 0;

    }
```

Output:

employee 1 id : 101

employee 1 name : Sonoo Jaiswal

employee 1 salary : 56000.000000

employee 2 id : 102

employee 2 name : James Bond

employee 2 salary : 126000.000000

Give a method to create, declare and initialize structure also develop a program to demonstrate nested structure.

Declaration of structure:-

```
struct structure_name
{
data_type member 1;
data_type member 2;
.
.
.
data_type member n;
} structure variable 1, structure variable 2,..., structure variable n;
```

Example:-

```
struct student
{
```

int rollno;

char name[10];

}s1;

Initialization:-

struct student s={1,"abc"};

structure variable contains two members as rollno and name. the above example initializes rollno to 1 and name to "abc".

Program:-

```
#include<stdio.h>
#include<conio.h>
struct college
{
int collegeid;
char collegename[20];
};
struct student
{
int rollno;
char studentname[10];
struct college c;
};
void main()
{
struct student s={1,"ABC",123,"Polytechnic"};
clrscr();
printf("\n Roll number=%d",s.rollno);
printf("\n Student Name=%s",s.studentname);
printf("\n College id=%d",s.c.collegeid);
printf("\n College name=%s",s.c.collegename);
getch();
}
```

Develop a program using structure to print data of three students having data members name, class, percentage.

```
#include<stdio.h>
#include<conio.h>
void main() {
struct student
{
char name[20];
char c[20];
int per;
```

```
} s[3];
int i;
clrscr();
for(i=0;i<3;i++)
{
printf("Enter name, class, percentage");
scanf("%s%s%d",&s[i].name,&s[i].c,&s[i].per);
}
for(i=0;i<3;i++)
{
printf("%s %s %d\n",s[i].name,s[i].c,s[i].per);
}
getch();
}
```

Write a program to declare structure employee having data member name, age, street and city. Accept data for two employees and display it.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
char name[10],street[10],city[10];
int age;
};
void main()
{
int i;
struct employee e[2];
clrscr();
for(i=0;i<2;i++)
{
printf("\n Enter name:");
scanf("%s",&e[i].name);
printf("\n Enter age:");
scanf("%d",&e[i].age);
printf("\n Enter street:");
scanf("%s",&e[i].street);
printf("\n Enter city:");
scanf("%s",&e[i].city);
}
for(i=0;i<2;i++)
{
printf("\n Name=%s",e[i].name);
printf("\n Age=%d",e[i].age);
printf("\n Street=%s",e[i].street);
printf("\n City=%s",e[i].city);
```

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

C – Typedef

- Typedef is a keyword that is used to give a new symbolic name for the existing name in a C program. This is same like defining alias for the commands.

- Consider the below structure.

```
struct student
{
     int mark [2];
     char name [10];
     float average;
}
```

- Variable for the above structure can be declared in two ways.

1st way :

```
struct student record;      /* for normal variable */
struct student *record;     /* for pointer variable */
```

2nd way :

```
typedef struct student status;
```

- When we use "typedef" keyword before struct <tag_name> like above, after that we can simply use type definition "status" in the C program to declare structure variable.

- Now, structure variable declaration will be, "status record".

- This is equal to "struct student record". Type definition for "struct student" is status. i.e. status = "struct student"

AN ALTERNATIVE WAY FOR STRUCTURE DECLARATION USING TYPEDEF IN C:

```
typedef struct student
{
     int mark [2];
     char name [10];
     float average;
} status;
```

// Structure using typedef:

```c
#include <stdio.h>

#include <string.h>


typedef struct student

{

  int id;

  char name[20];

  float percentage;

} status;


int main()

{

  status record;

  record.id=1;

  strcpy(record.name, "Raju");

  record.percentage = 86.5;

  printf(" Id is: %d \n", record.id);

  printf(" Name is: %s \n", record.name);

  printf(" Percentage is: %f \n", record.percentage);

  return 0;

}
```

OUTPUT:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

ANOTHER EXAMPLE PROGRAM FOR C TYPEDEF:

```c
#include <stdio.h>

#include <limits.h>
```

```
int main()

{

typedef long long int LLI;


printf("Storage size for long long int data " \

"type : %ld \n", sizeof(LLI));


return 0;

}
```

OUTPUT:

Storage size for long long int data type : 8

C enums

Enumerated Type Declaration

When you define an enum type, the blueprint for the variable is created. Here's how you can create variables of enum types.

enum boolean {false, true};

enum boolean check; // declaring an enum variable

Here, a variable check of the type enum boolean is created.

You can also declare enum variables like this.

enum boolean {false, true} check;

```
#include <stdio.h>

enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

int main()
{
  // creating today variable of enum week type
  enum week today;
  today = Wednesday;
  printf("Day %d",today+1);
  return 0;
}
```

Output

Day 4

C Program To Search An Element In An Array Using Standard Method

```
#include <conio.h>
 int main()
{
  int a[10000],i,n,key;

  printf("Enter size of the  array : ");
  scanf("%d", &n);
  printf("Enter elements in array : ");
  for(i=0; i<n; i++)
  {
    scanf("%d",&a[i]);
  }
   printf("Enter the key : ");
  scanf("%d", &key);

  for(i=0; i<n; i++)
  {
    if(a[i]==key)
    {
                    printf("element found ");
      return 0;
    }

  }

      printf("element not  found");
}
```

Enter size of the array: 5
Enter elements in array: 4
6
2
1
3
Enter the key: 2
element found

Write a program to declare the structure student, having data members as rollno, name and percentage. Accept data for five students and display the same.

```
#include <stdio.h>
struct student
{
  char name[50];
  int roll;
  float marks;
} s[10];

int main()
{
  int i;

  printf("Enter information of students:\n");

  // storing information
  for(i=0; i<10; ++i)
  {
    s[i].roll = i+1;

    printf("\nFor roll number%d,\n",s[i].roll);

    printf("Enter name: ");
    scanf("%s",s[i].name);

    printf("Enter marks: ");
    scanf("%f",&s[i].marks);

    printf("\n");
  }

  printf("Displaying Information:\n\n");
  // displaying information
  for(i=0; i<10; ++i)
```

```
        {
          printf("\nRoll number: %d\n",i+1);
          printf("Name: ");
          puts(s[i].name);
          printf("Marks: %.1f",s[i].marks);
          printf("\n");
        }
      return 0;
    }
```

Output
Enter information of students:

For roll number1,
Enter name: Tom
Enter marks: 98

For roll number2,
Enter name: Jerry
Enter marks: 89
.
Displaying Information:

Roll number: 1
Name: Tom
Marks: 98

.

Concept and need of function

WHAT IS C FUNCTION?

A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

```c
#include<stdio.h>

/*Function prototypes*/

myfunc();

 main()

{

   myfunc();

}

/*Function Defination*/

myfunc()

{

   printf("Hello, this is a test\n");

}
```

Need of functions in C

There are the following advantages of C functions.

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

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

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

- Reusability is the main achievement of C functions.

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

Library Functions: Math functions

C – Library functions

- Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.

- Each library function in C performs specific operation.

- We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.

- These library functions are created by the persons who designed and created C compilers.

- All C standard library functions are declared in many header files which are saved as file_name.h.

- Actually, function declaration, definition for macros are given in all header files.

- We are including these header files in our C program using "#include<file_name.h>" command to make use of the functions those are declared in the header files.

- When we include header files in our C program using "#include<filename.h>" command, all C code of the header files are included in C program. Then, this C program is compiled by compiler and executed.

C Math Functions

There are various methods in math.h header file. The commonly used functions of math.h header file are given below.

No.	Function	Description
1)	ceil(number)	rounds up the given number. It returns the integer value which is greater than or equal to given number.
2)	floor(number)	rounds down the given number. It returns the integer value which is less than or equal to given number.
3)	sqrt(number)	returns the square root of given number.
4)	pow(base, exponent)	returns the power of given number.
5)	abs(number)	returns the absolute value of given number.

C Math Example

Let's see a simple example of math functions found in math.h header file

```
#include<stdio.h>
#include <math.h>
int main(){
printf("\n%f",ceil(3.6));
printf("\n%f",ceil(3.3));
printf("\n%f",floor(3.6));
printf("\n%f",floor(3.2));
printf("\n%f",sqrt(16));
printf("\n%f",sqrt(7));
printf("\n%f",pow(2,4));
printf("\n%f",pow(3,3));
printf("\n%d",abs(-12));
 return 0;
}


Output:
4.000000
4.000000
3.000000
3.000000
4.000000
2.645751
16.000000
27.000000
12
```

Explain any four library functions under conio.h header file.

clrscr() -This function is used to clear the output screen.

getch() -It reads character from keyboard

getche()-It reads character from keyboard and echoes to o/p screen

putch - Writes a character directly to the console.

textcolor()-This function is used to change the text color

textbackground()-This function is used to change text background

C User-defined functions

C allows you to define functions according to your need. These functions are known as user-defined functions. For example:

Example: User-defined function

Here is an example to add two integers. To perform this task, we have created an user-defined addNumbers().


```
#include <stdio.h>
int addNumbers(int a, int b);        // function prototype
int main()
{
   int n1,n2,sum;
   printf("Enters two numbers: ");
   scanf("%d %d",&n1,&n2);
   sum = addNumbers(n1, n2);      // function call
   printf("sum = %d",sum);
   return 0;
}
int addNumbers(int a, int b)       // function definition
{
   int result;
   result = a+b;
   return result;            // return statement
```

}

C FUNCTION DECLARATION, FUNCTION CALL AND FUNCTION DEFINITION:
There are 3 aspects in each C function. They are,

Function declaration or prototype – This informs compiler about the function name, function parameters and return value's data type.

Function call – This calls the actual function

Function definition – This contains all the statements to be executed.

C functions aspects	syntax
function definition	Return_type function_name (arguments list) { Body of function; }
function call	function_name (arguments list);
function declaration	return_type function_name (argument list);

SIMPLE EXAMPLE PROGRAM FOR C FUNCTION:

- As you know, functions should be declared and defined before calling in a C program.
- In the below program, function "square" is called from main function.
- The value of "m" is passed as argument to the function "square". This value is multiplied by itself in this function and multiplied value "p" is returned to main function from function "square".

#include<stdio.h>

// function prototype, also called function declaration

float square (float x);

int main( )

{

   float m, n ;

   printf ("\nEnter some number for finding square \n");

   scanf ("%f", &m) ;

   // function call

   n = square (m) ;

   printf ("\nSquare of the given number %f is %f",m,n);

}

```
float square ( float x )  // function definition

{

    float p ;

    p = x * x ;

    return ( p ) ;

}
```

Scope of Variable in C

The scope of a variable decides the portion of a program in which the variable can be accessed. The scope of the variable is defined as follows...

> **Scope of a variable is the portion of the program where a defined variable can be accessed.**

The variable scope defines the visibility of variable in the program. Scope of a variable depends on the position of variable declaration.

In C programming language, a variable can be declared in three different positions and they are as follows...

- **Before the function definition (Global Declaration)**
- **Inside the function or block (Local Declaration)**
- **In the function definition parameters (Formal Parameters)**

Before the function definition (Global Declaration)

Declaring a variable before the function definition (outside the function definition) is called **global declaration**. That means the global variable can be accessed any where in the program after its declaration. The global variable scope is said to be **file scope**.

Example Program

```
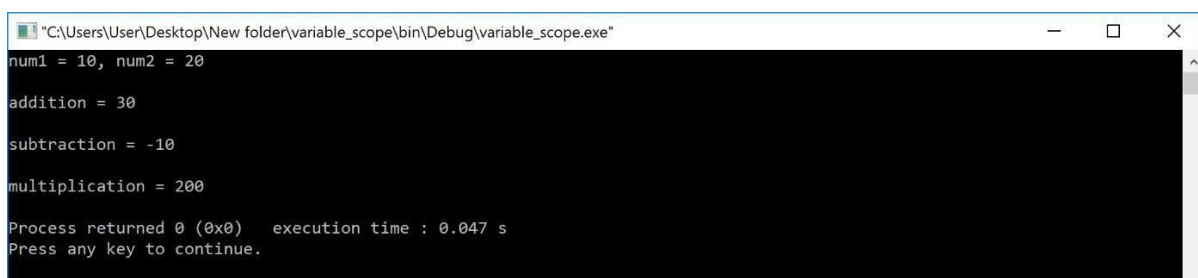#include<stdio.h>

#include<conio.h>


int num1, num2 ;
void main(){
    void addition() ;
    void subtraction() ;
    void multiplication() ;
    clrscr() ;
```

```c
  num1 = 10 ;

  num2 = 20 ;

  printf("num1 = %d, num2 = %d", num1, num2) ;

  addition() ;

  subtraction() ;

  multiplication() ;

  getch() ;

}

void addition()

{

  int result ;

  result = num1 + num2 ;

  printf("\naddition = %d", result) ;

}

void subtraction()

{

  int result ;

  result = num1 - num2 ;

  printf("\nsubtraction = %d", result) ;

}

void multiplication()

{

  int result ;

  result = num1 * num2 ;

  printf("\nmultiplication = %d", result) ;

}
```

Output:

In the above example program, the variables **num1** and **num2** are declared as global variables. They are declared before the main() function. So, they can be accessed by function main() and other functions that are defined after main(). In the above example, the functions main(), addition(), subtraction() and multiplication() can access the variables num1 and num2.

Inside the function or block (Local Declaration)

Declaring a variable inside the function or block is called **local declaration**. The variable declared using local declaration is called **local variable**. The local variable can be accessed only by the function or block in which it is declared.

Example Program

#include<stdio.h>

#include<conio.h>

void main(){

  void addition() ;

  int num1, num2 ;

  clrscr() ;

  num1 = 10 ;

  num2 = 20 ;

  printf("num1 = %d, num2 = %d", num1, num2) ;

  addition() ;

  getch() ;

}

void addition()

{

  int sumResult ;

  sumResult = num1 + num2 ;

  printf("\naddition = %d", sumResult) ;

}

Output:


```
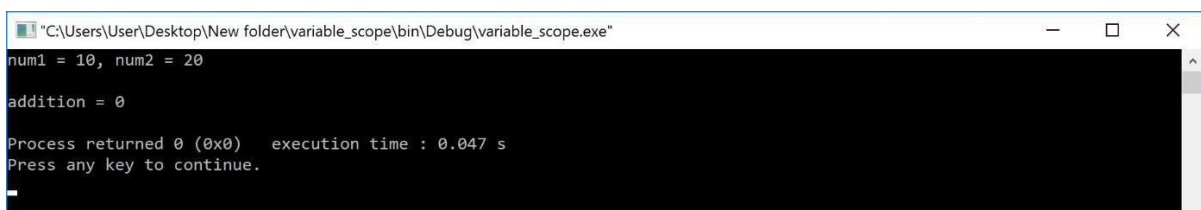"C:\Users\User\Desktop\New folder\variable_scope\bin\Debug\variable_scope.exe"
num1 = 10, num2 = 20

addition = 0

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

The above example program shows an **error** because, the variables num1 and num2 are declared inside the function main(). So, they can be used only inside main() function and not in addition() function.

In the function definition parameters (Formal Parameters)

The variables declared in function definition as parameters have a local variable scope. These variables behave like local variables in the function. They can be accessed inside the function but not outside the function.

Example Program

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

void main(){
   void addition(int, int) ;
   int num1, num2 ;
   clrscr() ;
   num1 = 10 ;
   num2 = 20 ;
   addition(num1, num2) ;
   getch() ;
}
void addition(int a, int b)
{
   int sumResult ;
   sumResult = a + b ;
   printf("\naddition = %d", sumResult) ;
}
```

Output:

In the above example program, the variables a and b are declared in function definition as parameters. So, they can be used only inside the addition() function.

Parameter Passing in C

When a function gets executed in the program, the execution control is transferred from calling-function to called function and executes function definition, and finally comes back to the calling function. When the execution control is transferred from calling-function to called- function it may carry one or number of data values. These data values are called as **parameters**.

Parameters are the data values that are passed from calling function to called function.

In C, there are two types of parameters and they are as follows...

- **Actual Parameters**
- **Formal Parameters**

In C Programming Language, there are two methods to pass parameters from calling function to called function and they are as follows...

Call by Value

Call by Reference

Call by Value

In call by value parameter passing method, the copy of actual parameter values are copied to formal parameters and these formal parameters are used in called function. The changes made on the formal parameters does not effect the values of actual parameters. That means, after the execution control comes back to the calling function, the actual parameter values remains same. For example consider the following program...

```
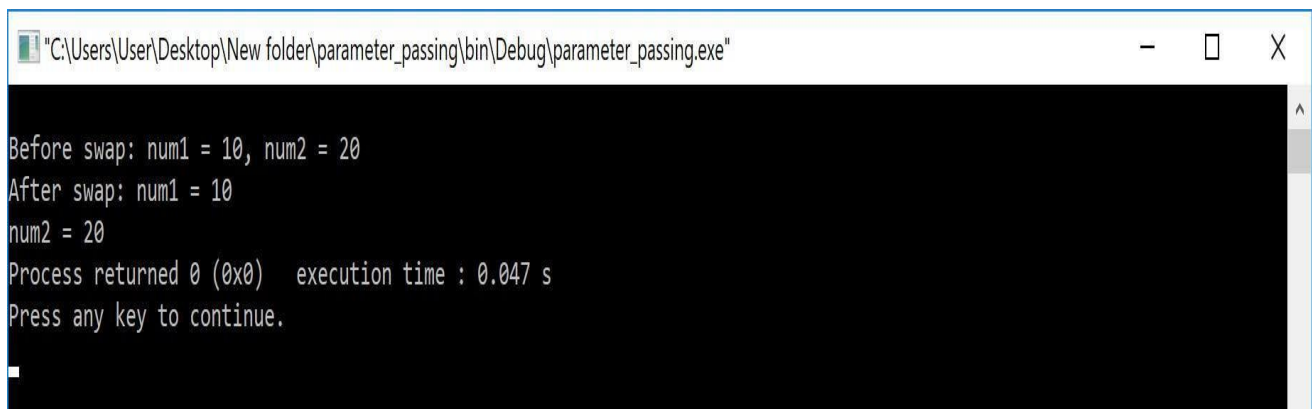#include<stdio.h>

#include<conio.h>

void main(){

  int num1, num2 ;

  void swap(int,int) ; // function declaration

  clrscr() ;

  num1 = 10 ;

  num2 = 20 ;


  printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
```

```
    swap(num1, num2) ; // calling function

      printf("\nAfter swap: num1 = %d\nnum2 = %d", num1, num2);

    getch() ;

}

void swap(int a, int b) // called function

{

    int temp ;

    temp = a ;

    a = b ;

    b = temp ;

}
```

Output:

Call by Reference

In Call by Reference parameter passing method, the memory location address of the actual parameters is copied to formal parameters. This address is used to access the memory locations of the actual parameters in called function. In this method of parameter passing, the formal parameters must be pointer variables.

```
#include<stdio.h>

#include<conio.h>

void main(){

    int num1, num2 ;

    void swap(int *,int *) ; // function declaration

    clrscr() ;
```

```
  num1 = 10 ;

  num2 = 20 ;

    printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;

  swap(&num1, &num2) ; // calling function

    printf("\nAfter swap: num1 = %d, num2 = %d", num1, num2);

  getch() ;

}

void swap(int *a, int *b)  // called function

{

  int temp ;

  temp = *a ;

  *a = *b ;

  *b = temp ;

}
```

Output:


```
"C:\Users\User\Desktop\New folder\parameter_passing\bin\Debug\parameter_passing.exe"

Before swap: num1 = 10, num2 = 20
After swap: num1 = 20, num2 = 10
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Difference between call by value and call by reference in c

No.	Call by value	Call by reference
1	A copy of the value is passed into the function	An address of value is passed into the function
2	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

Recursion in C

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls.

In the following example, recursion is used to calculate the factorial of a number.

```c
#include <stdio.h>
int fact (int);
int main()
{
    int n,f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("factorial = %d",f);
}
int fact(int n)
{
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}
```
Output
Enter the number whose factorial you want to calculate?5
factorial = 120

Let's see an example to find the nth term of the Fibonacci series.

```c
#include<stdio.h>
int  fibonacci(int);
void main ()
{
    int n,f;
    printf("Enter the value of n?");
    scanf("%d",&n);
    f = fibonacci(n);
    printf("%d",f);
```

```
}
int fibonacci (int n)
{
  if (n==0)
  {
  return 0;
  }
  else if (n == 1)
  {
    return 1;
  }
  else
  {
    return fibonacci(n-1)+fibonacci(n-2);
  }
}
```
Output
Enter the value of n?12
144

Storage Classes in C

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

- o Automatic
- o External
- o Static
- o Register

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function

C Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

int n = 10;
int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type intege
r.

Declaring a pointer

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

int *a;//pointer to int
char *c;//pointer to char

Pointer Example

An example of using pointers to print the address and value is given below.

As you can see in the above figure, pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of * (**indirection operator**), we can print the value of pointer variable p.

Let's see the pointer example as explained for the above figure.

```
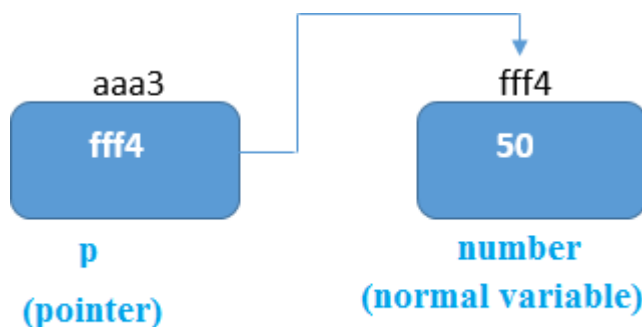#include<stdio.h>
int main(){
int number=50;
int *p;
p=&number;//stores the address of number variable
printf("Address of p variable is %x \n",p); // p contains the address of the number therefore prin
ting p gives the address of number.
printf("Value of p variable is %d \n",*p); // As we know that * is used to dereference a pointer th
erefore if we print *p, we will get the value stored at the address contained by p.
return 0;
}
```

Output

Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50

Advantage of pointer

1. Pointers reduce the length and complexity of a program.

2. They increase execution speed.

3. A pointer enables us to access a variable that is defined outside the function.

4. Pointers are more efficient in handling the data tables.

5. The use of a pointer array of character strings results in saving of data storage space in memory.

6. It supports dynamic memory management.

Usage of pointer

There are many applications of pointers in c language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where the pointer is used.

2) Arrays, Functions, and Structures

Pointers in c language are widely used in arrays, functions, and structures. It reduces the code and improves the performance.

NULL Pointer

A pointer that is not assigned any value but NULL is known as the NULL pointer. If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value. It will provide a better approach.

```
int *p=NULL;
```

In the most libraries, the value of the pointer is 0 (zero).

Declaration of C Pointer variable

General syntax of pointer declaration is,

```
datatype *pointer_name;
```

Data type of a pointer must be same as the data type of the variable to which the pointer variable is pointing. void type pointer works with all data types, but is not often used.

Here are a few examples:

```
int *ip      // pointer to integer variable

float *fp;      // pointer to float variable

double *dp;      // pointer to double variable

char *cp;      // pointer to char variable
```

Initialization of C Pointer variable

Pointer Initialization is the process of assigning address of a variable to a pointer variable. Pointer variable can only contain address of a variable of the same data type. In C language address operator & is used to determine the address of a variable. The & (immediately preceding a variable name) returns the address of the variable associated with it.

```
#include<stdio.h>

void main()
{
    int a = 10;
    int *ptr;      //pointer declaration
```

```
    ptr = &a;        //pointer initialization
}
```

Pointer Arithmetic in C

We can perform arithmetic operations on the pointers like addition, subtraction, etc.

Increment

Decrement

Addition

Subtraction

Comparison

Incrementing Pointer in C

If we increment a pointer by 1, the pointer will start pointing to the immediate next location. This is somewhat different from the general arithmetic since the value of the pointer will get increased by the size of the data type to which the pointer is pointing.

Let's see the example of incrementing pointer variable on 64-bit architecture.

```
#include<stdio.h>

int main(){

int number=50;

int *p;//pointer to int

p=&number;//stores the address of number variable

printf("Address of p variable is %u \n",p);

p=p+1;

printf("After increment: Address of p variable is %u \n",p); // in our case, p will get
incremented by 4 bytes.

return 0;

}
```

Output

Address of p variable is 3214864300

After increment: Address of p variable is 3214864304

Traversing an array by using pointer

```
#include<stdio.h>

void main ()

{

   int arr[5] = {1, 2, 3, 4, 5};

   int *p = arr;

   int i;

   printf("printing array elements...\n");

   for(i = 0; i< 5; i++)

   {

     printf("%d ",*(p+i));

   }

}
```

Output

printing array elements...

1 2 3 4 5

Decrementing Pointer in C

Like increment, we can decrement a pointer variable. If we decrement a pointer, it will start pointing to the previous location. The formula of decrementing the pointer is given below:

new_address= current_address - i * size_of(data type)

Example

```
#include <stdio.h>

void main(){

int number=50;

int *p;//pointer to int

p=&number;//stores the address of number variable

printf("Address of p variable is %u \n",p);

p=p-1;

printf("After decrement: Address of p variable is %u \n",p); // P will now point to the
immidiate previous location.
```

}

Output

Address of p variable is 3214864300

After decrement: Address of p variable is 3214864296

C Pointer Addition

We can add a value to the pointer variable. The formula of adding value to pointer is given below:

*new_address= current_address + (number * size_of(data type))*

#include<stdio.h>

int main(){

int number=50;

int *p;//pointer to int

p=&number;//stores the address of number variable

printf("Address of p variable is %u \n",p);

p=p+3; //adding 3 to pointer variable

printf("After adding 3: Address of p variable is %u \n",p);

return 0;

}

Output

Address of p variable is 3214864300

After adding 3: Address of p variable is 3214864312

C Pointer Subtraction

Like pointer addition, we can subtract a value from the pointer variable. Subtracting any number from a pointer will give an address. The formula of subtracting value from the pointer variable is given below:

*new_address= current_address - (number * size_of(data type))*

#include<stdio.h>

int main(){

int number=50;

int *p;//pointer to int

p=&number;//stores the address of number variable

printf("Address of p variable is %u \n",p);

p=p-3; //subtracting 3 from pointer variable

printf("After subtracting 3: Address of p variable is %u \n",p);

return 0;

}

Output

Address of p variable is 3214864300

After subtracting 3: Address of p variable is 3214864288

C Program to Print Elements of Array Using Pointers

include<stdio.h>

main()

{

  int a[5]= {5,4,6,8,9};

  int *p=&a[0];

  int i;

  //clrscr();

  for(i=0; i<5; i++)

    printf("\nArray[%d] is %d ",i,*(p+i));

  for(i=0; i<5; i++)

    printf("\n %d at %u ",*(p+i),(p+i));

  getch();

}

OUTPUT

```
Array[0] is 5
Array[1] is 4
Array[2] is 6
Array[3] is 8
Array[4] is 9
 5 at 2686708
 4 at 2686712
 6 at 2686716
 8 at 2686720
```

 9 at 2686724

Pointers as Function Argument in C

Pointer as a function parameter is used to hold addresses of arguments passed during function call. This is also known as call by reference. When a function is called by reference any change made to the reference variable will effect the original variable.

```c
#include <stdio.h>

void swap(int *a, int *b);

int main()
{
    int m = 10, n = 20;
    printf("m = %d\n", m);
    printf("n = %d\n\n", n);

    swap(&m, &n);    //passing address of m and n to the swap function
    printf("After Swapping:\n\n");
    printf("m = %d\n", m);
    printf("n = %d", n);
    return 0;
}

/*
    pointer 'a' and 'b' holds and
    points to the address of 'm' and 'n'
*/
void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

m = 10
n = 20
After Swapping:
m = 20
n = 10

Functions returning Pointer variables
A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function. They have scope only inside the function. Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.

```c
#include <stdio.h>

int* larger(int*, int*);

void main()
{
   int a = 15;
   int b = 92;
   int *p;
   p   =   larger(&a,   &b);
   printf("%d is larger",*p);
}

int* larger(int *x, int *y)
{
   if(*x > *y)
      return x;
   else
      return y;
}
```

92 is larger

Pointer to functions
It is possible to declare a pointer pointing to a function which can then be used as an argument in another function. A pointer to a function is declared as follows,

type (*pointer-name)(parameter);

Example:
```c
#include <stdio.h>

int sum(int x, int y)
{
   return x+y;
}

int main( )
{
```

```
    int (*fp)(int, int);
    fp = sum;
    int s = fp(10, 15);
    printf("Sum is %d", s);

    return 0;
}
```

25

C – Structure using Pointer

C structure can be accessed in 2 ways in a C program. They are,

Using normal structure variable
Using pointer variable

Dot(.) operator is used to access the data using normal structure variable and arrow (->) is used to access the data using pointer variable. You have learnt how to access structure data using normal variable in C – Structure topic. So, we are showing here how to access structure data using pointer variable in below C program.

EXAMPLE PROGRAM FOR C STRUCTURE USING POINTER:
In this program, "record1" is normal structure variable and "ptr" is pointer structure variable. As you know, Dot(.) operator is used to access the data using normal structure variable and arrow(->) is used to access data using pointer variable.

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

struct student
{
    int id;
    char name[30];
    float percentage;
};

int main()
{
    int i;
    struct student record1 = {1, "Raju", 90.5};
    struct student *ptr;

    ptr = &record1;

      printf("Records of STUDENT1: \n");
      printf("  Id is: %d \n", ptr->id);
```

```
    printf("  Name is: %s \n", ptr->name);
    printf("  Percentage is: %f \n\n", ptr->percentage);

  return 0;
}
```

OUTPUT:
Records of STUDENT1:
Id is: 1
Name is: Raju
Percentage is: 90.500000

Implement a program to demonstrate concept of pointers to function.

Pointer to function:
```
include<stdio.h>
int sum(int x, int y)
{
return x+y;
}
int main()
{
int s;
int(*fp)(int, int);
fp = sum;
s = fp(10,12);
printf("Sum = %d",s);
return 0;
}
```