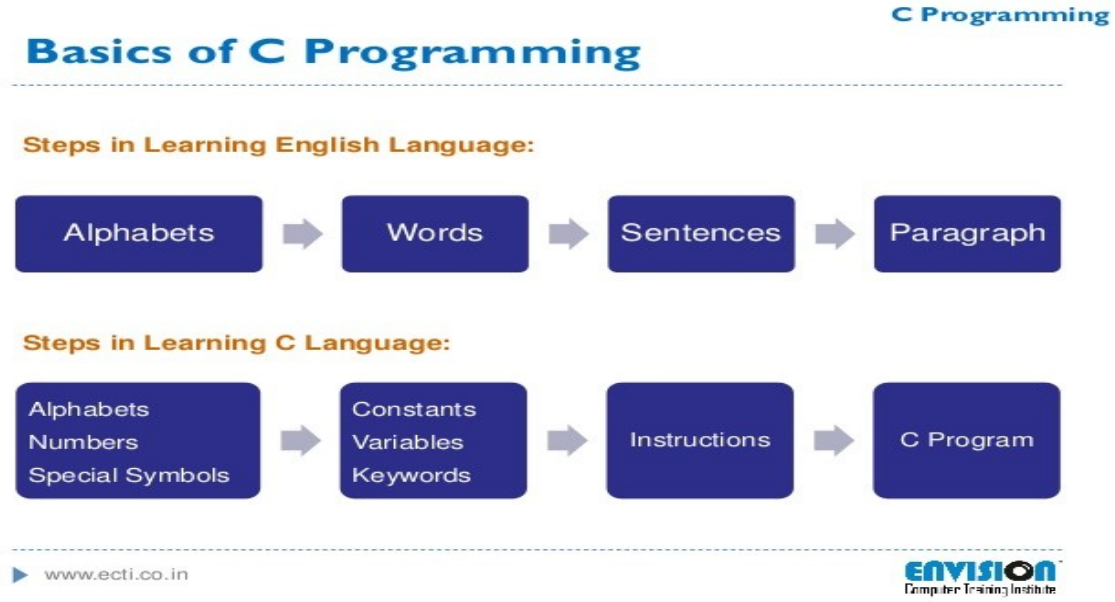
Syllabus

Sr. No	Details
1	Problem Definition
2	Algorithm – Developing Algorithm, Efficiency of Algorithms
3	Expressing Algorithms -Sequence
3.1	Expressions in C - Arithmetic and Boolean expressions
3.2	Use of Standard functions
3.3	Assignment statement, Input and output
4	Concept of Scalar Data-type
4.1	Scalar data types in C, Scope and life time, type conversion,
5	Expressing Algorithm – Selection
5.1	C Control structure for Selection
6	Expressing Algorithm – Iteration
6.1	Ordering a solution in a loop, C Control Structure for Iteration
7	Decomposition of a function
7.1	Defining functions in C, Function & types, Recursive functions
8	Additional C Data-types
8.1	Arrays, Strings, Structures, Files & Pointers

INDEX

1. C Fundamentals

Communicating with the computer involves speaking the language the computer understands which immediately rules out English as the language of communication with computer. So there is a close analogy between learning English and learning C language.



1.1. C Character Set

A character denotes any alphabets, digits or special symbols used to represent information.

Alphabets: A,B,C…..Y,Z

a,b,c……y,z

Digits: 0,1,2,…….9

Special Symbols: ~,!,@,#,$,%,^,&,*,(,),_,+,=,-,{,},[,],<,>,?,\, /

1.2. Constants, Variables and Keywords

The alphabets, numbers and special symbols when properly combined form constants, variables and keywords. A ‘constant’ is an entity that does not change where as variable is an entity that may change. So when we say x = 3 then x is the name of the variable and 3 is a constant.

1.2.1. Constants

Rules of constructing an Integer Constant:

1. At least one digit.
2. Must not have a decimal point.
3. Positive or negative but by default is taken as positive.
4. No commas and blanks are allowed with integer constant.
5. It ranges from -32767 to 32767 and unsigned int from 0 to 65535.

Example: 2,5,8,98,2000

Rules for constructing a Real Constant:

1. At least one digit.
2. Must have a decimal point.
3. Positive or negative but by default is taken as positive.
4. No commas and blanks are allowed with integer constant.
5. The mantissa and the exponent should be separated by e.
6. The mantissa should have a positive or negative sign but default is positive.
7. It ranges from -3.4e38 to 3.4e38

Example: 3.14,3.4e-6

Rules for constructing a Character Constant:

1. A character constant is a single alphabet, a single digit or a single specified symbol enclosed with single inverted commas. Both the inverted commas should point to the left.
2. The maximum length of a character constant can be one character.

Example: 'A','6','+'.

1.2.2. Variables:

An entity that may vary during program execution is called a variable. Variable names are given to locations in memory. These locations can contain integer, real or character constants. A particular type of variable can hold only the same type of constant.

Rules for constructing variable names:

1. A variable name is a combination of 1 to 31 alphabets, digits or underscores. Do not write unnecessary long names as it adds up to your typing effort.
2. The first character in a variable name should be an alphabet.
3. No commas or blanks are allowed in the variable name.
4. No special symbols other than underscores are allowed in a variable name.
5. Try giving some relevant variable names.

Example: si_int,m_hra,pop_e_89.

Next how do you declare variables?

int si_int;

float bas_sal;

char code;

1.2.3. Keywords:

Keywords are the words whose meaning has already been explained to the C compiler. The keywords cannot be used as variable names because if we do so then we are trying to assign new meaning to the keyword which is not allowed by the compiler. So the keywords are also called as Reserved words. There are 32 reserved keywords in C.

The first C program:

1. Each instruction in a C is written as a separate statement.

2. The statements must appear in the same order in which we wish them to be executed.

3. Blank spaces may be inserted between two words to improve readability of the statement. However no blank spaces are allowed within a variable, constant and keyword.

4. All statements are entered in small case letters.

5. Comments of the program should be enclosed /* this is a C program*/

Example:/* Calculation of Simple Interest */

```
#include<stdio.h>
void main()
{       int p,n;
        float si,r;
        p=1000;
        n=3;
        r=8.5;
        /* Formula for Simple Interest */
        si=(p*n*r)/100;
        printf("\nSimple Interest : %f",si);
}
```

```
/*Simple Interest : 255.000000*/
```

printf( ) function :

The general form of printf( ) function is:

printf("<format specifier>,<list of variables>");

<format string> can contain:

%f for printing real values

%d for printing integer values and %c for printing character values

Receiving Input:

scanf( ) is used as a function for receiving input.

Example: /* Calculation of Simple Interest */

```
#include<stdio.h>
void main()
{
        int p,n;
        float si,r;
        printf("\n Enter the value of p,n,r :");
        scanf("%d %d %f",&p,&n,&r);
        /* Formula for Simple Interest */
        si=(p*n*r)/100;
        printf("\nSimple Interest : %f",si);
}
```

```
/* Enter the value of p,n,r :2000 5 10.5
Simple Interest : 1050.000000*/
```

1.3. C Instructions:

There are basically three types of instructions in C:

1.3.1. Type Declaration Statements:

This instruction is used to declare the type of variables being used in the program. Any variable used in the program must be declared before using it in any statement. The type declaration statement is written at the beginning of the main( ) function.

Ex: int bas;

float rs, grossal;

char name, code;

a) While declaring the type of variable we can also initialize it as shown below:

int i = 10, j=25;

float a = 1.5, b=1.99+2.4 * 1.44;

b) The order in which we define the variables is sometimes important sometimes not. For example :

int i = 10, j=25; is same as

int j=25, i=10; However,

float a = 1.5, b= a+ 3.1; is alright but

float b = a + 3.1, a =1.5 is not. This is because here we are trying to use ‘a’ even before defining it.

1.3.2. Arithmetic Instructions:

A C instruction consists of a variable name on the left hand side of = and names and constants on the right hand side of = are connected by arithmetic operators like +, - , * and /.Example: int ad;

```
float kot, delta, alpha,beta,gamma;
 ad=3200;
kot = 0.0056;
delta = alpha  * beta / gamma + 3.2 * 2 / 5;
```

Here,

- *, /, -,+ are the arithmetic operators.
- = is the assignment operator
- 2, 5, 3200 are integer constants.
- 3.2 and 0.0056 are real constants.
- ad is an integer variable. kot, delta, alpha, beta, gamma are real variables

1.4. Hierarchy of Operators:

Priority	Operators	Description
1st	*, /, %	Multiplication, Division, Modular Division
2nd	+, -	Addition, Subtraction
3rd	=	Assignment

Example:

I = 2 * 3 / 4 + 4 / 4 + 8 – 2 + 5 / 8

Stepwise evaluation of this expression is shown below:

I = 2 * 3 / 4 + 4 / 4 + 8 – 2 + 5 / 8
I = 6 / 4 + 4 / 4 + 8 – 2 + 5 / 8
I = 1 + 4 / 4 + 8 – 2 + 5 / 8
I = 1 + 1 + 8 – 2 + 5 / 8
I = 1 + 1 + 8 – 2 + 0
I = 2 + 8 – 2 + 0
I = 10 – 2 + 0
I = 8 + 0
I = 8

Associativity of Operators:

If both the operators have got same priority then the left to right associativity should be taken.

1.5. Relational Operators:

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not Equal to

1.6. Logical Operators:

&&	And
||	Or

1.7. Exercise:

1. Which one of the following are invalid variable names:

Basicsalary	_basic	Basic-hra	#mean
Group.	422	Population in 2006	Over time
Meter	Float	Hello	Team'svictory

2. What will be the output of the following programs?

```
#include<stdio.h>
void main()
{
 int i=2,j=3,k,l;
 float a,b;
 k=i/j*j;
 l= j/i*i;
 a=i/j*j;
 b=j/i*i;
 printf("%d %d %f %f",k,l,a,b);
}
```

3. A C program contains the following declarations and initial assignments:

   int i = 8, j = 5;

   float x = 0.005, u = 0.01;

   char c= 'c', d= 'd';

   Determine the value of the following expressions. Use the values initially assigned to the variables for each expression.

   i) (3 * I – 2 * j) % (2 * d – c)

   ii) 2* x + (y = = 0)

   iii) –(i + j)

   iv) C > d

   v) (x >y) && ((i >0||(j<5))

4. What is the output of the following program?

i)
```
void  main()
{
 int x=10,y,z;
 z= y =x;
 y-=x--;
 z-=--x;
 x-=--x-x--;
 printf(" y=%d z = %d x = %d",y,z,x);
}
```

ii)
```
void main()
{
 float a =0.5, b= 0.9;
 if(a && b > 0.9)
          printf("if part");
 else
          printf("else part");
 }
```

iii)
```
void main()
{
 int a,b,c,d,e;
 a=b=c=d=e=40;
 printf("%d %d %d %d %d",a,b,c,d,e);
}
```

5. What is the output of the following expressions if a = 1, b= 2, c=3. Also give the contents of a, b and c at the end of expression:

   i) a = a + + + - - b

   ii) b = b + c - - * - - a

6. If x, y and z are integer variable then evaluate the following expressions and give the final values of x, y and z. x= 10, y= 6, z= 8.

   i) x = = y + z

   ii) x - - + y - - + - -z / (x * x)

   iii) - - x + y - - * - - z

7. Write the following programs in C:

a) If marks obtained by a student in 5 subjects are input through a keyboard find the aggregate marks and percentage obtained by the student.

b) Temperature of a city in Fahrenheit is input through the keyboard. Write a program to convert this temperature into centigrade degrees.

c) The length and breadth of a rectangle and radius of circle are input through the keyboard. Write a C program to calculate area and perimeter of the rectangle and the area and circumference of the circle.

Console Input / Output functions:

Formatted Functions				Unformatted Functions		
Type	Input	Output		Type	Input	Output
char	scanf( )	printf( )		char	getch( ) getche( ) getchar( )	putch( ) putchar( )
int	scanf( )	printf( )		int	-	-
float	scanf( )	printf( )		float	-	-
string	scanf( )	printf( )		string	gets( )	puts( )

Format Specifications:

Data type	Formate specifier
int	%d,%i
float	%f
char	%c
String	%s

Implementation:

```
#include<stdio.h>
void main()
{
        int i;
        float j;
        char c;
        printf("\nPress an integer,float and char:\n");
        scanf("%d%f%c",&i,&j,&c);
        printf("i=%d,j=%f,k=%c",i,j,k);
}
```

```
/*
Press an interger,float and char
12 5.5 A
i=12,j=5.5,k=A
*/
```

1.8. Decision Control Structure:

In the previous cases we have executed sequential instructions i.e. they are executed in the order in which they appear. But if we want the instructions to follow a sequence pattern depending on the condition then we use decision control structure. There are relational operators used to check the condition.

This expression	Is true if
x = = y	X is equal to y
x ! = y	X is not equal to y
x < y	X is less than y
x > y	X is greater than y
x < = y	X is less than or equal to y
x > = y	X is greater than or equal to y

1.8.1. if statement:

The general form of if statement looks like this:

```
if (this condition is true)
        Execute this statement
```

Example:

```
#include<stdio.h>
void main()
{
        int num;
        printf("Enter the number :");
        scanf("%d",&num);
        if(num<10)
                printf("Good data is correct");
}
```

```
/*Enter the number :12
Enter the number :4
Good data is correct*/
```

1.8.2. If-Else Statement:

The expression following if evaluates to true and performs else part if it evaluates to false.

Syntax:

```
if (condition)
        true statements;
else
        false statements;
```

Example 1: If an employee's basic salary is less than Rs.1500 then his HRA = 10% of basic salary and DA = 90% of the basic salary. If the salary is either equal to or above Rs. 1500 then HRA = Rs. 500 and DA = 98% of the basic salary. If the employee's salary is input through keyboard then write a program to find gross salary.

```
#include<stdio.h>
void main()
{
        float bs,gs,da,hra;
        float total;
        printf("\nEnter basic salary :");
        scanf("%f",&bs);
        if(bs<1500)
        {
                hra=bs*10/100;
                da=bs*90/100;
        }
        else
        {
                hra=500;
                da=bs*98/100;
        }
        gs=bs+hra+da;
        printf("\nBasic salary : %f",bs);
        printf("\nHRA         : %f",hra);
        printf("\nDA          : %f",da);
        printf("\nGross salary : %f",gs);
}
```

```
/*Enter basic salary :2000
Basic salary : 2000.000000
HRA         : 500.000000
DA          : 1960.000000
Gross salary : 4460.000000
Enter basic salary :1400
Basic salary : 1400.000000
HRA         : 140.000000
DA          : 1260.000000
Gross salary : 2800.000000*/
```

Example 2:

Write a program to find greatest of three numbers:

```
#include<stdio.h>
void main()
{
        int a,b,c;
        printf("\nEnter the three numbers :");
        scanf("%d %d %d",&a,&b,&c);
        if(a>b && a>c)
                printf("%d is greatest",a);
        else if(b>a && b>c)
                printf("%d is greatest",b);
        else if(c>b && c>a)
                printf("%d is greatest",c);}
```

```
/*Enter the three numbers :56 78 34                    78 is greatest*/
```

1.9. Loop Control Structure:

The versatility of the computer lies in its ability to perform a set of instructions repeatedly. This involves repeating some portion of the program either specified number of times or until a particular condition is specified. This repetitive operation is done through a loop control instruction.

There are three methods by way of which we can repeat a part of the program. They are:

a) Using a **while** statement

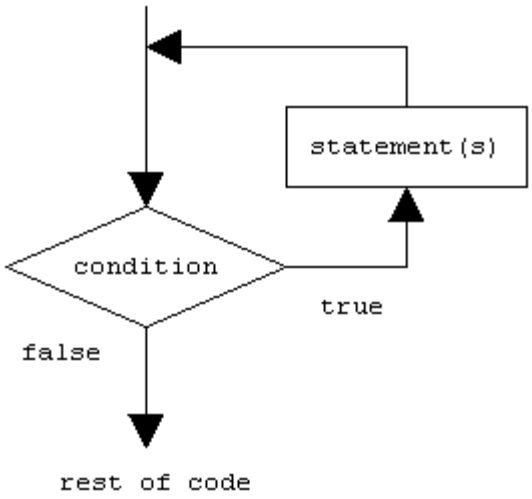
b) Using a **do-while** statement

c) Using a **for** statement

1.9.1. While Loop:

Structure of while loop:

```
while(condition)
{
some code;
}
```



example: WAP to display number from 1 to n, taking value of n from the user.

```
#include<stdio.h>
void main()
{
        int i=1,n;
        printf("\nEnter n :");
        scanf("%d",&n);
        while(i<=n)
        {       printf("%d ",i);
                i++;
        }
}
/*Enter n :7
1 2 3 4 5 6 7*/
```

1.9.2. Do-While Loop:

Structure of do-while:

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

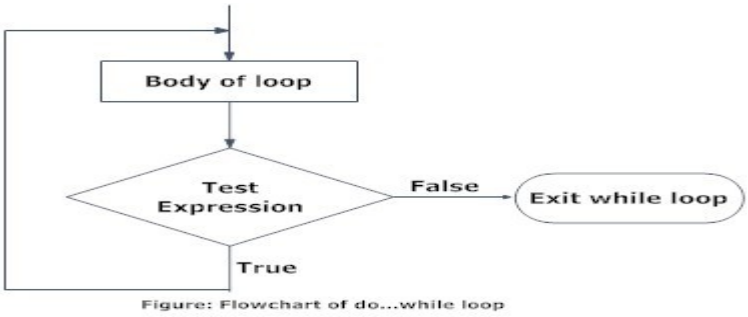


Figure: Flowchart of do...while loop

```
#include<stdio.h>
void main()
{
        int n,i=1;
        printf("\nEnter the n:");
        scanf("%d",&n);

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

```
/*Input 1: Enter the n:0
0
Input 2:Enter the n:5
12345*/
```

- **Difference between while and do-while:**

While	Do-while
entry control loop	exit control loop
condition is checked before loop execution	condition is checked at the end of loop
never execute loop if condition is false	executes false condition at least once since condition is checked later
there is no semicolon at the end of while statement	there is semicolon at the end of while statement.

while(condition){
some code;
}
	do{
	}while(condition);

1.9.3. For Loop:

Structure of for:

```
for(initialization ; condition ; update )
{
some code;
}
```

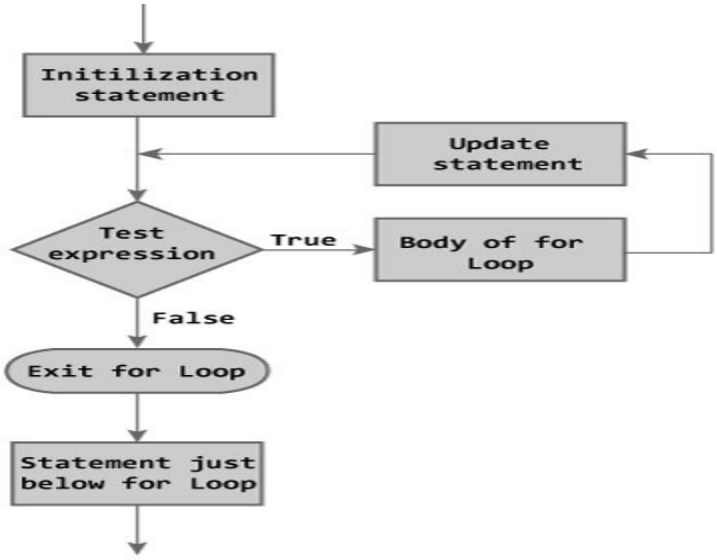


Figure: Flowchart of for Loop

Ex:WAP to find the factorial of a given number.

```
#include<stdio.h>
void main()
{
        int i,n,fact=1;
        printf("\nEnter n :");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                fact=fact*i;
        }
        printf("\n%d factorial is %d",n,fact);
}
```

```
/* Enter n :5
5 factorial is 120*/
```

1.10. The goto statement:

Write a program to read an integer from keyboard and if it is an even number divide by 2 and if odd multiply by 3 add 1 to it. Display all intermediate values until the number converges to 1. Also count and display the number of iterations required for converge.

```
#include<stdio.h>
void main()
{
        int x;
        printf("\n Enter the value of x :");
        scanf("%d",&x);
        l: if(x%2==0)
        {
                x=x/2;
                printf("%d  ",x);
        }
        else
        {
                x=x*3+1;
        }
         if(x>1)
                goto l;
}
```

```
/*Enter the value of x :7
11  17  26  13  20  10  5  8  4  2  1
```

1.11. The break statement:

We often come across situations where we have to jump out of loop instantly without waiting to get back to the conditional test. The keyword break allows us to do this. When the break is encountered inside any loop, control automatically passes to the first statement after the loop. A break is usually associated with an 'if'.

Ex: Write a program to determine whether a number is prime or not.

```
#include<stdio.h>
void main()
{       int i,n;
        printf("\n Enter the value :");
        scanf("%d",&n);
        for(i=2;i<n;i++)
        {
                if(n%i==0)
                        break;
        }
        if(i==n)
                printf("\n %d is a prime no",n);
        else    printf("\n %d is not a prime no",n);}
```

```
/*output: Enter the value :17                    17 is a prime no*/
```

1.12. The continue statement:

The continue statement is used to force an early iteration of a loop. This means we may continue running the loop, but stop processing the remainder of the code in its body for this particular iteration. The continue statement performs such an action. In the while and do-while loops a continue statement cause control to be transferred directly to the conditional expression that controls the loop. In a ***for*** loop control goes first to the iteration portion of the for statement and then to the conditional expression. For all three loops any intermediate code is bypassed.

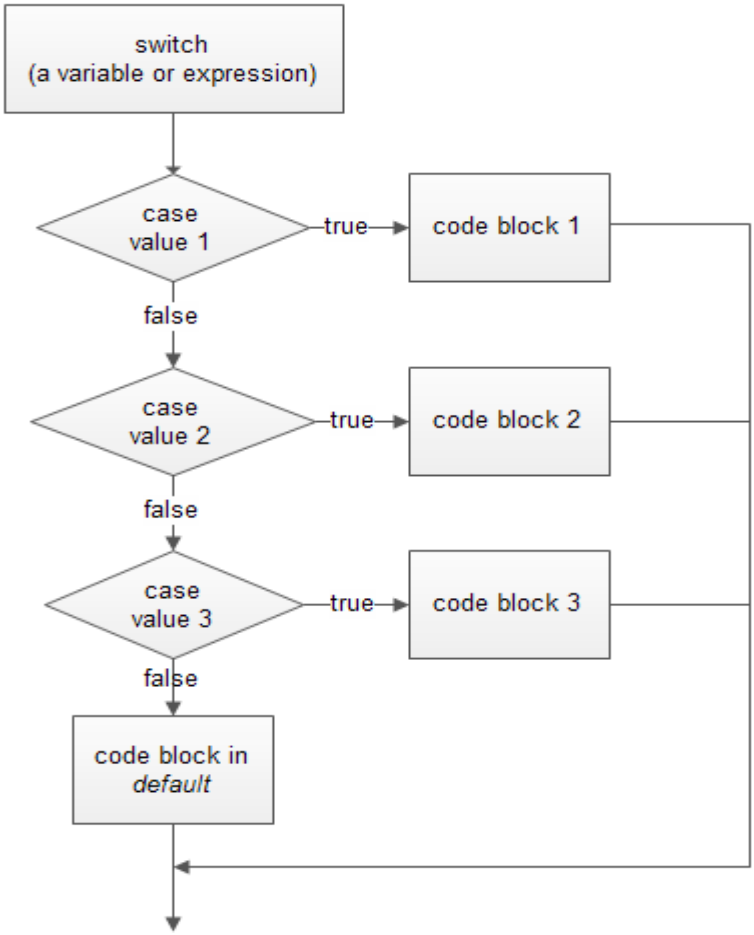
Example:

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

```
/*
output:
0
1
2
3
4
5
*/
```

In above program every time when for loop gets executed it will print the value of i then next statement is "continue", this will transfer the control to for loop skipping the remaining statement of the loop in this case **“the printf(''Hello'');”**.

1.13 Switch case:



switch case is a powerful way for decision making. It allows you to take different action based on different values/cases. Switch statement has multiple code blocks called cases and each case has a value associated to it.

Switch contains an expression, who's value is compared with the value of cases starting from the first case, whichever case has matching value will get executed.

If value of expression does not match with any case, then **"default"** case is executed.

However it is not compulsory to provide default case, but if it is not provided and none of the case is true then switch statement will simply terminated.

It is important to write break statement after every case because once any case is true then all the remaining cases wont be checked and all the remaining statement will be executed so once the case is true, break should be placed as the last statement of the case.

Example :WAP to print single digit number into words.

```
#include<stdio.h>
void main()
{
int no;
printf(" Enter number\n");
scanf("%d",&no);
switch(no)
{
case 0:printf("Zero ");
break;
case 1:printf("one ");
break;
case 2:printf("two ");
break;
case 3:printf("three ");
break;
case 4:printf("four  ");
break;
case 5:printf("five ");
break;
case 6:printf("six ");
break;
case 7:printf("seven ");
break;
case 8:printf("eight ");
break;
case 9:printf("Nine ");
break;
default:printf("Invalid number");
}
```

```
/*
Output:
Enter number
5
five
*/
```

1.14 Implementation of nested loops:

1. Write a program to generate the following pattern:

a)

*	*	*	*
*	*	*	*
*	*	*	*
*	*	*	*

```
#include<stdio.h>
void main()
{
int r,c;
for(r=1;r<=5;r++)                    //handles rows
{
for(c=1;c<=5;c++)                    //handles column
{
printf("* ");
}
printf("\n");
}
}
```

b)

*				
*	*			
*	*	*		
*	*	*	*	
*	*	*	*	*

```
#include<stdio.h>
void main()
{
        int r,c;
        for(r=1;r<=5;r++)                    //number of rows
        {
                for(c=1;c<=r;c++)            //number of coumns
                {
                        printf("* ");
                }
                printf("\n");
        }
}
```

c)

1				
1	2			
1	2	3		
1	2	3	4	
1	2	3	4	5

```
#include<stdio.h>
main()
{
        int r,c,sum,i;
        for(r=1;r<=5;r++)
        {
                for(c=1;c<=r;c++)
                {
                        printf("%d ",c);
                }
                printf("\n");
        }
```

d)

1			
2	2		
3	3	3	
4	4	4	4

```
#include<stdio.h>
main()
{
        int r,c,sum,i;
        for(r=1;r<=4;r++)
        {
                for(c=1;c<=r;c++)
                {
                        printf("%d ",r);
                }
                printf("\n");
        }
}
```

Important questions:

Operators:

1.WAP to find greatest of three number using conditional operator.

2.Explain Bitwise operator.

For loop:

3.WAP to display first n natural numbers.

4.WAP to find the factorial of a given number.

5.WAP to implement Fibonacci series upto n terms.

6.WAP to check whether the given number is prime or not.

7.WAP to calculate the value of series:1+1/2+1/3.....+1/n

8. WAP to calculate the value of series:1+1/2!+1/3!+....+1/n!

9. WAP to calculate the value of series:sinx=$x-x^3/3!+x^5/5!.....x^n/n!$

10.WAP to calculate the value of series:1/2-3/4+5/6.....n

11.WAP to check if the entered character is digit,alphabet or any other character.

if-else:

12.WAP to check whether a given number is divisible by 5 or not.

13.WAP to check if entered number is even or odd.

14.WAP to display result(first,second,third class) taking marks from user using nested if-else.

15.WAP to check if the entered year is leap year or not.

Switch control:

16.WAP to take month number from user and display in words using switch case.

17.WAP to develop menu based calculator for performing basic maths operations.

While loop:

18.WAP to find the sum and product of all the digits of a user entered number.

19.WAP to count the number of digits in a user entered number.

20.WAP to find whether a given number is Armstrong or not.

21.WAP to check whether a given number is palindrome or not.

22.WAP to display Armstrong numbers between 1-1000

23.WAP to display prime numbers between 1-100.

24.WAP to find LCM and GCD of two numbers.

25.WAP to display user entered number into words.

Nested for loop:

26.	27.	28.	29.
* ** *** *****	1 12 123 1234	1 22 333 4444	1 23 456 78910
30.	31.	32.	33.
A AB ABC ABCD	A BB CCC DDDD	A BC DEF GHIJ	4 43 432 4321
34.	35.	36.	37.
4444 333 22 1	1234 123 12 1	ABCD EFG HI j	10987 654 32 1
38.	39.	40.	41.
* ** *** ****	A B C D A B C A B A	**** *** ** *	1 1 1 1 1 1 1 1 1 1
42.	43.	44.	45.
1 1 2 1 1 2 3 2 1 1 2 3 4 3 2 1	1 1 2 A 1 2 3 A B 1 2 3 4 A B C	* * * * * * * * *	A ABA ABCBA ABCDCBA

2. Functions

A function is self contained block of statements that perform a coherent task of some kind. Every C program could be thought of as a collection of these functions. The function main ( ) is executed first and it calls the other function directly or indirectly. It is necessary that every program must have the main ( ). We may have user defined functions which can be invoked from other parts of the program. Functions are building blocks of C programs where all the program activity occurs. As we have noted earlier, using a function is something like hiring a person to do a specific job for you.

There are three categories of functions:

2.1 Functions with no arguments and no return values:

Example:

```
#include<stdio.h>
void message()
{
        printf("\nSmile the world smile's back at you");
}
void  main()
{
        message();
        printf("\nCry and you stop the monotony");
}
```

```
/*Smile the world smile's back at you
Cry and you stop the monotony*/
```

Explanation: The execution starts from main() function, the first statement in main is call to message() function so message() will be executed then printf() statement of main(), The message() function is neither taking any argument nor returning any value.

2.2 Function having arguments and no return value:

Example:

```
#include<stdio.h>
void power(int x,int n)
{
int i,res=1;
for(i=0;i<n;i++)
res=res*x;
printf("The result is : %d",res);}
```

```
void main()
{
        int x,n;
        printf("\nEnter the value of x and n :");
        scanf("%d %d",&x,&n);
        power(x,n);
}
```

```
/*Enter the value of x and n :3 7
The result is : 2187*/
```

Explanation: The power() function is taking two integers as arguments and not returning anything.Program will start from main() function where the value of two integers(x and n) are taken from the user and passed to power() function.

2.3 Function having arguments and return value:

Example:

```
#include<stdio.h>
int fact(int no)
{
int i,prod=1;
for(i=1;i<=no;i++)
{
prod=prod*i;
}
return prod;
}
void main()
{
int num;
printf("Enter the number\n");
scanf("%d",&num);
int facto=fact(num);
printf("factorial of %d =%d",num,facto)
}
```

```
/*
Output:
Enter the number
5
factorial of 5=120
*/
```

Explanation:
The fact() function is taking one integer as argument and returning an integer, a number is taken from the user from the main() function and that number is send to fact() function by calling it, fact() calcualte factorial and return the result to main() where it was called, in above case result will be stored i facto variable.

2.4 Recursion:

In C a function can call itself this is called as recursion. A function is said to be recursive if there exists a statement in its body for the function call itself. For example the Fibonacci terms are 0,1,1,2….In this sequence each term except the first two are obtained by adding its two immediate predecessor terms. The recursive definition of the sequence is:

fib(n) = 0 if n = 1

= 1 if n = 2

fib(n-1) + fib(n -2) if n > 2

While coding recursive function, we must take care that there exists a reachable termination condition inside the function so that function may not be invoked endlessly. The following program illustrates the use of recursion.

Example 1:WAP to generate fibo series upto n terms using recursion.

```
#include<stdio.h>
int fib(int n)
{
        if(n==0 || n==1)
                return(1);
        else
                return(fib(n-1)+fib(n-2));
}
void main()
{
        int x,i;
        printf("\nEnter the number of terms :");
        scanf("%d",&x);
        for(i=0;i<x;i++)
                printf("%d ",fib(i));
}
```

```
/*Enter the number of terms :10
1 1 2 3 5 8 13 21 34 55*/
```

2.5 Exercise:

1. Write a program which has a function that checks whether a number is prime or not.
2. Write a program to include a function which finds G.C.D and L.C.M of two variables using recursive and non-recursive functions.
3. Write a program to convert a number into binary, octal and hexadecimal using functions.
4. Write a function swap to interchange value of two variables with and without using a third variable. Call this function in main ( ).
5. Write a function to find reverse a number using recursive functions.
6. Write a recursive function to compute x^n where x and n are taken as integers.
7. Write a program to find the value of BIO = n! /r! * (n-r)!
8. Write a recursive function to implement Fibonacci series for n=10.

3. Array

The C language provides a capability that enables the user to design a set of similar data types called array. We define Array as a collective name given to a group of 'similar quantities'. These similar quantities could be percentage of marks salaries of 30 employees and age of 50 employees etc. Each member in the group is referred to by the position in the group. For example assume the following group numbers which represent the percentage of marks obtained by 5 students, per= {48, 88, 34, 23, 96}.

If we want to refer to the second number then we will say per[1] because an array by default starts from the position 0. So the general notion would be per[i] where I can take value from 0, 1,2,3,4 depending upon the position being referred. Here per is the subscripted variable and 'i' is its subscript. An array is a collection of all ints, floats and chars; usually the array of characters is called a string whereas an array of ints and floats is called as a simple array.

Let us try to **write a program to find the average of marks obtained by student:**

```
#include<stdio.h>
void main()
{
        int sum=0,n;
        int i,marks[10];
        printf("\nEnter the number of students :";
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter marks :");
                scanf("%d",&marks[i]);
                sum=sum+marks[i];
        }
        printf("\nSum of marks is :%d"),sum);
        printf("\nAverage of marks is :%d",sum/n);
}
```

```
/*Enter the number of students :5
Enter marks :56
Enter marks :78
Enter marks :89
Enter marks :45
Enter marks :56
Sum of marks is :324
Average of marks is :64*/
```

3.1 Array Initialization:

In the previous cases we have used an array that did not have any values in them to begin with. We managed to store values in them during program execution. Let us now see how we initialize an array while declaring it. Following are the few examples that demonstrate this:

```
int num[6] = {2,4,12,6,45,5};
int n[]={2,4,12,6,45,5};
float press[]={2.3,34.2,-23.4,-11.3};
```

3.2 Passing array elements to a function:

Array elements can be passed to a function by calling the function by value or by reference. In the call by value we pass the value of array elements and in call by reference we pass addresses of array elements to the function. Let us consider a program:

```
#include<stdio.h>
int fun(int x[],int n)
{
int s=0;
for(int i=0;i<n;i++)
{
 s=s+x[i];
}
return(s);
}
void main()
{
int sum=0,n;
int i,marks[10];
printf("\nEnter the number of students :";
scanf(“%d”,&n);
for(i=0;i<n;i++)
{
printf("\nEnter marks :";
scanf(“%d”,&marks[i]);
}
sum=fun(marks,n);
printf("\nSum of marks is :%d",sum);
printf("\tAverage of marks is :%d",sum/n);}
```

```
/*Enter the number of students :5
Enter marks :56
Enter marks :67
Enter marks :78
Enter marks :23
Enter marks :45
Sum of marks is :269      Average of marks is :53*/
```

3.3 Strings:

A string is a collection of characters enclosed within quotes. In C/C++ a string is defined as a character array terminated by NULL character '\0'. Each element of a string is used as one element in the array housing it. So the character arrays must be declared one character longer than the size of the string we wish to store. The last bytes store the string terminator '\0'.There is no data type available as string so we have to declare it as an array of characters. There are few inbuild string functions provided by the C.

- strlen(string): It return the length of the string.
- strcpy(string 1,string 2):It copies the second string into first string, but the previous value of first string gets override.
- strcat(first,second):It concatenates(append) the second string to first and result is stored in first string.
- Strcmp(s1,s2):It compares first and second string and if they are equal it returns 0 else return

  the difference in ASCII values of the first non-matching characters in the strings.

  Thus strcmp(s1,s2) =0 , if s1=s2

  <0 , if s1<s2

  >0 ,if s1>s2

Implementing of string functions:

```
#include<stdio.h>
#include<string.h>
void main(){
char x[20],y[10];
printf("\nEnter the string x :");
scanf("%s",&x);
printf("\nEnter the string y :");
scanf("%s",&y);
int a =strlen(x);
printf("\nThe string is :%s",x);
printf("\nThe length of an array :%s",a);
strcat(x,y);
printf("\nThe concatenated string is :%s",x);
strcpy(x,y);                                    // y=x=syed
printf("\nThe y string copied to x gives :%s",x);
a=strcmp(x,y);                                  //both x ans y contains 'syed' so they are
equal
if(a==0)
    printf("\nStrings are x and y equal");
else
    printf("\nStrings x and y are not equal");}
```

```
/*Enter the string x :amer
Enter the string y :syed
The string is :amer  The length of an array :4  The concatenated string is :amersyed
The y string copied to x gives :syed
Strings are x and y equal*/
```

Finding the length of string without using predefined functions:

```
#include<stdio.h>
void main()
{
 char x[10];
 int i;
 printf("\nEnter the string :");
 scanf(“%s”,&x);
 printf("\nThe string is :";
 for(i=0;x[i]!='\0';i++)
 printf(x[i]);
 printf("\nLength of string is :%d",i);
 }
```

3.4 Two dimensional arrays:

We have seen one dimensional array up till now but it is also possible to have arrays with more than one dimension. The two dimensional arrays is called a matrix. So let us write a sample **program that stores a roll number and marks obtained by student side by side in a matrix.**

```
#include<stdio.h>
void main()
{
    int stud[4][2];
    int i;
    for(i=0;i<4;i++)
    {
        printf("\nEnter Roll NO and marks :");
        scanf(“%d %d”,&stud[i][0],&stud[i][1]);
    }
    for(i=0;i<4;i++)
        printf("\nRoll No :%d    Marks : %d",stud[i][0],stud[i][1]);
}
```

```
/*Enter Roll NO and marks :1 45
Enter Roll NO and marks :2 56
Enter Roll NO and marks :3 67
Enter Roll NO and marks :4 78
Roll No :1  Marks :45
Roll No :2  Marks :56
Roll No :3  Marks :67
Roll No :4  Marks :78*/
```

3.5 Exercise:

1. Write a program to enter 10 numbers in an array and then search for the element in the array. Display appropriate messages if found or not found.
2. Write a program to delete duplicate numbers in an array.
3. Write a program to find the largest & the second largest in the list entered by the user.
4. Write a program to input data in an array carrying 25 elements. Compute the sum and average.
5. Write a program to display and count no. of vowels in a given string.
6. Accept the string and calculate length without using string functions.
7. Write a program to reverse a string without using predefined string functions.
8. Write a program to sort the array in ascending order.
9. Write a program to find string s2 in string s1 where s2 is substring & s1 is main string.
10. WAP to check if the entred string is palindrom or not.
11. Write a program which contains the function to check whether matrix is symmetric or not. A matrix is symmetric if its transpose equals the matrix itself.
12. Write a menu driven program to perform the following operations on matrices.

i) Addition ii) Subtraction iii) Transpose iv) Multiplication

4.Pointers

Pointers are the special type of variables which stores the address of other variable.
Let us consider the notation:

int i =4;

This declaration tells the compiler to:

1. Reserve space in memory to hold the integer values.
2. Associate the name 'i' with the memory location.
3. Store the value 4 at this location.

In above case we can create a pointer which will store the memory address of variable i.

Syntax: datatype *pointerName;

example: int *p //p is the pointer which can store the memory address of any integer variable

4.1 Operators: There are two operators which are used in pointer, Reference and de-reference.

4.1.1 Reference/address operator(&):

The adress of operator is unary operator which gives the address of its operand.

Suppose**,int x** is a variable then the expression &x gives the memory address of x.

Now consider **int *p,** here we defined a pointer which can store address of any integer variable.

p=&x, this statement stores the memory address of x into p.

4.1.2 De-referencing/value-of operator(*):

consider the example:

```
int x=7;
int *p;
p=&x;
```

The value of x i.e 7 can be accessed in two ways, one is direclty write x, and the second method is to write expression *p, where * is a unary operator, when this operator is used with the pointer it gives the value of the variable which the pointer is pointing to.

Programming Example:

```
#include<stdio.h>
void main()
{
int x, *p                    //x is interger variable and p is pointer
x=13;
p=&x;
printf("%d\t%p\t%d",x,p,*p);    //%p to print the address stored at p
}
```

//Output: 13 0x7fff64848b14 13 //here middle value is address of x

4.2 Pointer to array:

An array is an ordered collection of consecutive memory locations or objects. It means consecutive memory location means locations with consecutive memory addresses. Say x is an array of 10 elements then addresses of element x[1] can be obtained from x[0] by adding 1 to it. The address of the elements of an array is ordered in incremental increasing order

```
#include<stdio.h>
void main()
{       int arr[]={11,-20,30,41,15};
        int *ptr;
        ptr=&arr[0];
        printf("\nThe array elements are :\n");
        while(ptr<ptr+5)
        {
                printf("%d ",*ptr);
                ptr++;
        }
}
```

```
/*The array elements are :
11 -20 30 41 15*/
```

So in the above program the pointer points to the first element of an array initially. When it is incremented it points to the next element in the array. This is called as the pointer to an array.

4.3 Call by value and Call by reference:

In C language there are two ways of passing arguments to the function one is call by value which we used till now to call the functions and other way is call by reference.
Lets see both the ways one by one.

4.3.1 Call by value:

consider following program,

```
#include<stdio.h>
void interchange(int number1,int number2)
{
   int temp;
   temp = number1;
   number1 = number2;
   number2 = temp;
}
void main()
{
   int num1=50,num2=70;
   printf("Number 1 : %d",num1);
   printf("\tNumber 2 : %d",num2);

}
```

```
//Output:Number 1 : 50        Number 2 : 70
```

Explanation:

While Passing Parameters using call by value , **xerox copy of original parameter is created** and passed to the called function.

1. Any update made inside method will not affect the **original value of variable in calling function**.
2. In the above example num1 and num2 are the original values and xerox copy of these values is passed to the function and these values are copied into number1,number2 variable of sum function respectively.
3. As their scope is limited to only function so they **cannot alter the values inside main function**.

4.3.2 Call by reference:

```
#include<stdio.h>

void interchange(int *num1,int *num2)
{
   int temp;
   temp  = *num1;
   *num1 = *num2;
   *num2 = temp;
}
```

```
void main() {

   int num1=50,num2=70;
   interchange(&num1,&num2);

   printf("\tNumber 1 : %d",num1);
   printf("\tNumber 2 : %d",num2);


}
```

Output: Number 1 : 70 Number 2 : 50

Explanation:While passing parameter using call by address scheme , we are passing the actual address of the variable to the called function.Any updates made inside the called function will modify the original copy since we are directly modifying the content of the exact memory location.

4.4 Exercise:

1. Explain the following functions: i) malloc ( ) ii) calloc( ) iii) free( )
2. The below program contains the following statements:

```
float  a =0.001, b= 0.003;
float c, *pa, *pb;
pa = &a;
*pa = 2*a;
Pb=&b;
c=3*(*pb - *pa);
```

The address of a as 1130, b= 1134 and c=1138 then what values are assigned to-
i) &a ii) *pa iii) c iv) pb v) *pa vi) &c

3. A program contains the following declarations : int a[5] = {10,40,60,75,85}

Assume the starting memory address is 65535. What is the meaning of the following?

i) sizeof(a) ii) a iii)*a iv) *(a+2) v) *a+2 vi) a+2

5. Structures and Unions

We can create and use the data types other than the fundamental data types. These are known as user defined data types. Data types using the keyword **struct** are known as structures. As seen earlier arrays have similar data type elements. A structure is a collection of mixed data types referenced by a single name. it is a group of related data items i.e. structural elements of arbitrary types. The general structure of declaring the syntax is:

```
struct <name>
{
        <type > <member1>;
        <type> <member2>;
        ……..
        <type> <memberN>;
} <structural variables>;
```

Where <name > is the name of the structure i.e. name of the new data type. The keyword **struct** and <name> are used to declare structured variables.

For example:

```
struct employee
{
        int empno;
        char name[20];
        char designation[20];
        char dept[20];
} emp;
```

The structure variable can be declared as:

```
struct employee
{
        int empno;
        char name[20];
        char designation[20];
        char dept[20];
};
struct employee emp;
```

5.1 Accessing structure elements:

Structure use a **dot(.)** operator the below program illustrate the use of structure

```
#include<stdio.h>
struct employee
{       int empno;
        char name[20];
        float salary;
};
void main()
{
        struct employee emp;
        printf("\nEnter the data :");
        printf("\nEnter the employee no :");
        scanf("%d ",&emp.empno);
        printf("\nEnter the name :";
        scanf("%s ",&emp.name);
        printf("\nEnter the salary :");
        scanf("%d",&emp.salary);
        printf("\nDisplay the details :");
        printf("\nEmployee Number : %d",emp.empno);
        printf("\nEmployee Name   : %s",emp.name);
        printf("\nEmployee Salary : %f",emp.salary);
}
```

```
/*Enter the data :
Enter the employee no :1
Enter the name :amer
Enter the salary :50000
Display the details :
Employee Number : 1
Employee Name   : amer
Employee Salary : 50000 */
```

5.2 Structure within a structure(Nested structure):

The following program illustrates the use of structure within a structure with the help of above example:

```
#include<stdio.h>
struct employee
{
        int empno;
        char name[20];
        float salary;

struct date
{
        int dd,mm,yy;
} d;
};
```

```
void main()
{
        struct employee emp;
        printf("\nEnter the data for employee :\n");
        printf("\nEnter the employee no :");
        scanf("%d",&emp.empno);
        printf("\nEnter the name :");
        scanf("%d ",&emp.name);
        printf("\nEnter the salary :");
        scanf("%d ",&emp.salary);
        printf("\nEnter Date of Birth (dd/mm/yy :");
        scanf("%d %d %d ",&emp.d.dd,&emp.d.mm,&emp.d.yy);
        printf("\nDisplay the details :");
        printf("\nEmployee Number       : %d",emp.empno);
        printf("\nEmployee Name         : %s",emp.name);
        printf("\nEmployee Salary       : %f",emp.salary);
        printf("\nEmployee Date of birth : %d/%d/%d", emp.d.dd,emp.d.mm,emp.d.yy);
        }
```

```
/*Enter the data for employee :
Enter the employee no :101
Enter the name :abc
Enter the salary :2000
Enter Date of Birth (dd/mm/yy) :01 03 1977
Display the details :
Employee Number       : 101
Employee Name         : abc
Employee Salary       : 2000
Employee Date of birth : 1/3/1977*/
```

5.3 Array of structures:

```
#include<stdio.h>
struct employee
{
        int empno;
        char name[20];
        float salary;
};
void main()
{
        struct employee emp[5];
        printf("\nEnter the data for 3 employees :\n");
        for(int i=0;i<3;i++)
        {
                printf("\nEnter the employee no :");
                scanf("%d",&emp[i].empno);
                printf("\nEnter the name :");
                scanf("%s",emp[i].name);
                printf("\nEnter the salary :");
                scanf("%f",&emp[i].salary);
        }
```

```
        printf("\nDisplay the details :");
        for(i=0;i<3;i++)
        {
                printf("\nEmployee Number :%d ",emp[i].empno);
                printf("\nEmployee Name   : %s ",emp[i].name);
                printf("\nEmployee Salary : %f ",emp[i].salary);
        }
}
```

```
/*Enter the data for 3 employees :
Enter the employee no :1
Enter the name :amer
Enter the salary :3000
Enter the employee no :2
Enter the name :junaid
Enter the salary :5000
Enter the employee no :3
Enter the name :sandip
Enter the salary :6000
Display the details :
Employee Number : 1
Employee Name   : amer
Employee Salary : 3000
Employee Number : 2
Employee Name   : junaid
Employee Salary : 5000
Employee Number : 3
Employee Name   :
Employee Salary : 6000*/
```

5.4 Exercise:

1. What is union? How does it differ from a structure? Explain with suitable example how is a union member accessed? How can a union member be processed? When is a union required in programming?
2. Define a structure called cricket that will describe the following information-
   i) Player name ii) Country Name iii) No. of matches played iv) Batting average
   Develop a program that will store information of 50 cricket players around the world using structure. Also display names of players in descending order of batting average.
3. Define a structure ‘st’ to contain the name, date of birth and total marks obtained. Define the structure ‘dob’ to represent date of birth. WAP to read data for n students in a class and sort them in descending order of batting average.
4. State the difference between structure and union?