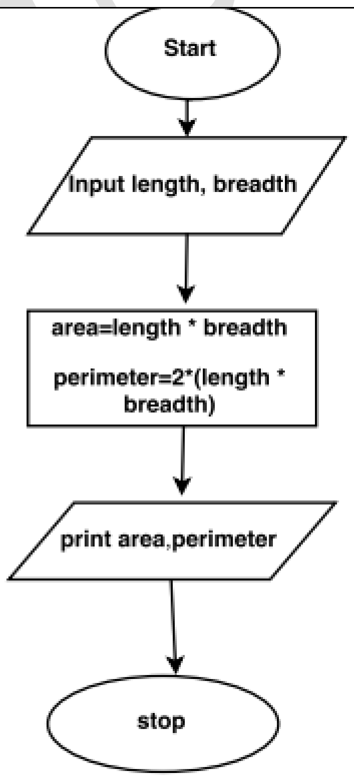
First Semester B.E Degree Examination Dec 2016/ Jan 2017 Programming in C and data structures Solution

Module 1

1a.Define an Algorithm. Write an algorithm to find area and perimeter of a rectangle.[06M]

Solution:

- It is a step by step procedure to solve a problem
- A sequential solution of any problem that is written in natural language.
- Using the algorithm, the programmer then writes the actual program.
- The main use of algorithm is to help us to translate English into C.
- It is an outline or basic structure or logic of the problem.

Algorithm	**flowchart**
Step 1: Start **Step 2:** [input the values of length and breadth] read length, breadth **Step 3:** [Compute 'area'] area=←length * breadth **Step 4:** [Compute 'perimeter'] perimeter=←2*(length * breadth) **Step 5:** display area **Step 6**: Stop	 Start Input length, breadth area=length * breadth perimeter=2*(length * breadth) print area,perimeter stop

1b. Write general structure of C explain with an example.[06 M]

Solution:

Structure of C program	Example.

Comments/Documentation Section
Preprocessor Directives
Global declaration section
void main( ) [Program Header]
{
Local Declaration part
Execution part
Statement 1
------------ 	
------------ 	
Statement n
}
User defined Functions
	/* program to print a message on screen*/
	#include<stdio.h>
	
	void main( )
	{
	printf("INDIA\");
	
	}

- ## Structure of C Program

- **Comments/Documentation Section**
- Comments are short explaining the purpose of the program or a statement.
- They are non executable statements which are ignored by the compiler.
- The comment begins **with /* and ends with */.**
- The symbols /* and */ are called **comment line delimiters.**
- Example: /* Program to compute Quadratic Equation */
- **Preprocessor Directives**
- Preprocessor Directives begins with a # symbol
- Provides instructions to the compiler to include some of the files before compilation starts.
- Some examples of header files are

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include<stdlib.h>        Etc…
```

- **Global Declaration Section**
  - There will be some variable which has to be used in anywhere in program and used in more than one function which will be declared as global.

- **The Program Header**
  - Every program must contain a main() function.
  - The execution always starts from main function.

- Every C program must have only one main function.

- **Body of the program**
  - Series of statements that performs specific task will be enclosed within braces { and }
  - It is present immediately after program header.
    It consists of two parts
    1. **Local Declaration part**: Describes variables of program
       Ex:**int sum = 0;**
       int a , b:
       float b;
    2. **Executable part**: Task done using statements.
       All statements should end with semicolon(;)

Subroutine Section: All user defined functions that are called in main function should be defined.

1c. Convert the following Mathematical Expressions in to C equivalent (04M)

Mathematical Expression	C Expression
Area=√s(s-a)(s-b)(s-c)	**Area=sqrt(s*(s-a)*(s-b)*(s-b))**
$x = \frac{-b+\sqrt{b^2}-4ac}{2a}$	**X= (-b+sqrt(b*b-4*a*c))/(2*a)**

OR

2a. Explain the different types of input output function in c with syntax and example.(06M)

Solution:

Displaying Output using printf

- printf is an output statement in C used to display the content on the screen.
- print: Print the data stored in the specified memory location or variable.
- Format: The data present in memory location is formatted in to appropriate data type.
- There are various forms of printf statements.

Syntax	Example
printf(" format string", variable list);	Ex1: **printf("INDIA");**

	Ex1:
	int a=10;
	float b=20;
	printf(“a=%d, b=%f”,a,b);
	
	output INDIA
	a=10,b=20

- Format string may be any character. The characters included within the double quotes will be displayed on the output screen
- Format string also called as control string.
- Format string consist of **format specifier** of particular data type
- Format specifiers starts with **%** sign followed by **conversion code.**
- variable list are the **variable names** separated by **comma.**
- Number of format specifiers must be equal to number of variables.
- While specifying the variables name make sure that it matches to the **format specifiers** with in the double quotes.
- It is also called ***conversion specifier or conversion code*.**
- There are many format specifiers defined in C.
- It specifies the type of data that is being processed
- Format specifiers are the character string with **% sign** followed with a character.

Input Function (scanf)

- To enter the input through the input devices like keyboard we make use of scanf statement.

Syntax	Example

scanf(“ format string”, address of list);	Ex1:
	int a;
	float b;
	scanf(“%d %f”,&a,&b);

- Format string may be any character. The characters included within the double quotes will be displayed on the output screen
- Format string also called as control string.
- Format string consist of **format specifier** of particular data type
- **List of addresses of variables consist of the variable name preceded with & symbol(address operator).**

Rules for scanf

- **No escape sequences or additional blank spaces** should be specified in the format specifiers.
- & symbol is must to read the values, if not the entered value will not be stored in the variable specified.

Symbols	Meaning
%d	Decimal signed integer number
%f	float point number
%c	Character
%o	octal number
%x	hexadecimal integer(Lower case letter x)
%X	hexadecimal integer(Upper case letter X)
%e	floating point value with exponent(Lower case letter e)
%E	floating point value with exponent (Upper case letter E)
%ld	long integer
%s	String
%lf	double

2b.Explain the following operators[06M]
Solution:

i. ***Unary Operators:*** An operator which acts on only **one operand** to produce the result is called unary operator. The Operators precede the operand.

Examples: -10
-a
--b

ii. **Binary operators**

An operator which acts on **two operands** to produce the result is called a binary operator. In an expression involving a binary operator, the operator is in between two operands.

Examples: a + b a*b

iii. **Conditional Operator (OR) Ternary Operator**

The operator which operates on 3 operands are called Ternary operator or conditional operator.

Syntax: (exp1)? exp2: exp3

where exp1,exp2, and exp3 are expressions and

- exp1 is an expression evaluated to true or false
- if exp1 is evaluated to true exp2 is executed
- if exp1 is evaluated false exp3 is executed

For example, consider the following statements.

```
a=10;
b=15;
Max = (a>b)? a:b;
```

In this example, Max will be assigned the value of b i.e b=15.

2c. Draw the flowchart and write a c program to compute simple interest.[4M}

```
/*program to compute simple interest*/
#include<stdio.h>
void main()
{
        float p,t,r,si;
        clrscr();
        printf("\nEnter values of p, t and r:");
        scanf("%f%f%f",&p,&t,&r);
        si=(p*t*r)/100;
        printf("\nSimple interest is:%f”,si);
}
```

Module 2

3a.List all the conditional control statements used in c.Explain any two with syntax and example.[6M]

Solution:

There are **5 Conditional** statements in C

1. **if control construct(simple if)**
2. **if else control construct**
3. **nested if else control construct**
4. **else if ladder or cascaded**
5. **switch control construct**

1.The if statement (Simple if/ One way selection)

- An if statement is a single selection statement.
- It is used to execute a set of statements if the condition is true, if the condition is false, it skips executing those set of statements. Hence it is called **one way selection**.

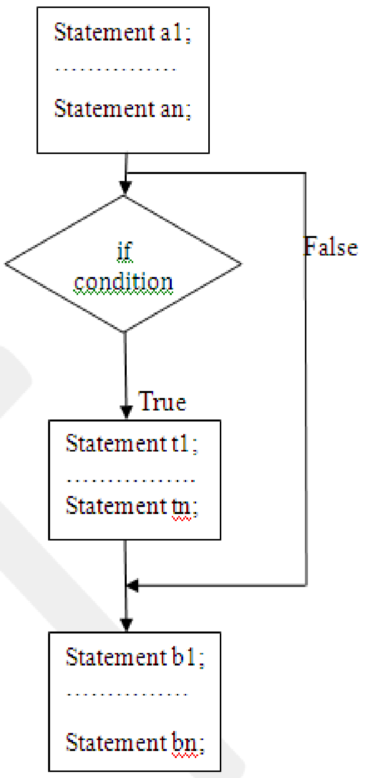
Explanation

- The keyword **if** must be followed by an **expression** and expression must be enclosed within parentheses.
- First statement1 is executed followed by statement2.
- Condition is checked

Syntax:

```
Statement a1;
     ........
Statement an;
if(condition)
{
     Statement t1;
      .............
     Statement tn;
}
     Statement b1;
     ........
     Statement bn;
```

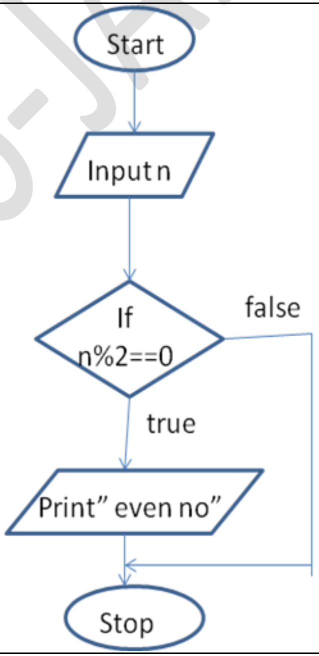
Flowchart:



Explanation

Example:

Algorithm	**Flowchart**	**Program**

Algorithm: Check even number
S1: Start
S2: Read N
S3: if(n % 2 == 0)
{
Print "even no"
}
S4: Stop
	 Start
	Input n
	If n%2==0
	false
	true
	Print" even no"
	Stop
		#include<stdio.h>
		void main()
		{
		int n;
		clrscr();
		printf(" Enter the number\n")
		scanf("%d",&n);
		if(n%2==0)
		{
		printf("Even no");
		}
		}

2. if else statement

Explanation

- The keyword if and else must be followed by an expression and expression must be enclosed within parentheses.
- First statement1 is executed followed by statement2.
- Condition is checked

Synatx:	Example :

```
    Statement 1;
    Statement 2;
if(condition)
{
    Statement 3;
    Statement 4;
}
else
{
    Statement 5;
    Statement 6;
}
    Statement 7;
```
	```
	#include<stdio.h>
	void main()
	{
	int n;
	printf(" Enter the number\n")
	scanf("%d",&n);
	     if(n%2==0)
	        {
	printf("Even no");
	        }
	else
	        {
	printf("odd no");
	        }
	}
	```

- ➢ if true control goes to statement3 , statement4 and automatically goes to statement7.
- ➢ If condition is false control goes to statement5 , statement6 and automatically goes to statement7.

3b.write a c program that reads from the user an arithmetic operator and two operands perform thje corresponding arithmetic operation on the operands using switch statement.
Solution:

```
/*program to perform arithmetic operation*/
#include<stdio.h>
void main( )
{
         int a,b,c,ch;
        printf("enter the two numbers");
        scanf("%d%d",&a,&b);
        printf("1. addition");
        printf("2. subtraction");
        printf("3. multiplication");
        printf("4. division");
        printf("enter the choice");
        scanf("%d",&ch);
        switc(ch)
        {
                case 1 :c=a+b;
                        printf("sum is %d",c);
                        break;
                case 2: c=a-b;
                        printf("difference is %d",c);
                        break;
                case 3: c=a*b;
                        printf("product is %d",c);
                        break;
                case 4: if(b==0)
                        {
                                printf("divide by zero error");
                                exit(0);
                        }
                        else
                        {
                                c=a/b;
                                printf("quotient is %d",c);
                                break;
                        }
                default: printf("invalid choice");
}
```

3C implement a c program to find the reverse of an integer number and check whether it is palindrome or not.

```
/*c program to find the reverse of an integer number and check whether it is
palindrome or not.*/
#include<stdio.h>
void main()
{
         int  n,rev=0,rem,temp;
        clrscr();
        print5f("enter the number\n"):
        scanf("%d",&num);
        temp=num;
        while(num!=0)
        {
                rem=num%10;
                rev=rev*10+rem;
                num=num/10;

        }

        printf ("The reversed number is %d",rev);

        if(temp==rev)
```

```
                    printf("%d is a PALINDROME\n",temp);
          else
                    printf("%d is not  a PALINDROME\n",temp);
}
```

OR

4a. what are unconditional control statements.explain any two with example.

Definition:

The statements that transfer the control from one part of the program to another part without checking for any conditions are called unconditional branch statements.

The 4 Unconditional branch statements are

1. **goto**
2. **break**
3. **continue**
4. **return**

1.The break statement:

- The break statement is an unconditional jump statement.
- The break statement can be used as the last statement in each case's statement list.
- A break statement causes control to transfer to the end of the switch statement.
- If a break statement is not used, the flow of control will continue into the next case.

"What is a use of break statement?"

The **break** statement in C programming language has the following two usages:

- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement.
- If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

switch (option)	for(…………)
{	{
case 'A':	……………….
aCount++;	if(…………)
break;	{
case 'B':	……
bCount++;	break;
break;	……………..
case 'C':	…………..
cCount++;	}
break;	…………

}	………….. }

	Example:
	#include<stdio.h>
	void main()
	{
	  int i,n=5;
	  for(i=0;i<=n;i++)
	  {
	    if(i==3)
	      break;
	      printf("%d\n",i);
	  }
	}
	
	Output: 1 2

The continue Statement

- The continue statement can be placed only in the body of a for loop, a while loop, or a do...while loop.
- The continue statement is used only in the loops to terminate the current iteration and continue with remaining iterations.
- In the **while** statement and **do-while** statement, whenever a **continue statement** is executed, the rest of the statements within the body of the loop are skipped and the **conditional** expression in the loop is executed.
- But, when **continue** is executed in the **for loop** ,control is transferred to **expression 3**.
- The pictorial representation is shown below:

Current iteration is skipped	Current iteration is skipped	Control is transferred exp3

while(expression)
{
    Action 1;
      continue;
   Action n;
}
	do
	{
	    Action 1;
	      continue;
	   Action n;
	
	} while(expression);
		for(exp1,exp2,exp3)
		{
		    Action 1;
		      continue;
		   Action n;
		}

Example:

```
#include<stdio.h>                                   Output: 1  2  4  5
void main()
{
   int i,n=5;
   for(i=1;i<=5;i++)
   {
            if(i==3)
           continue;
           printf("%d\n",i);
    }
}
```

Example:

```
#include<stdio.h>                                   OUTPUT: INDIA
void main()                                                 INDIA
{                                                           INDIA
        int i;
       for(i=1;i<=3;i++)
       {
              printf("INDIA\n");
            continue;
            printf("is our country\n");
        }
}
```

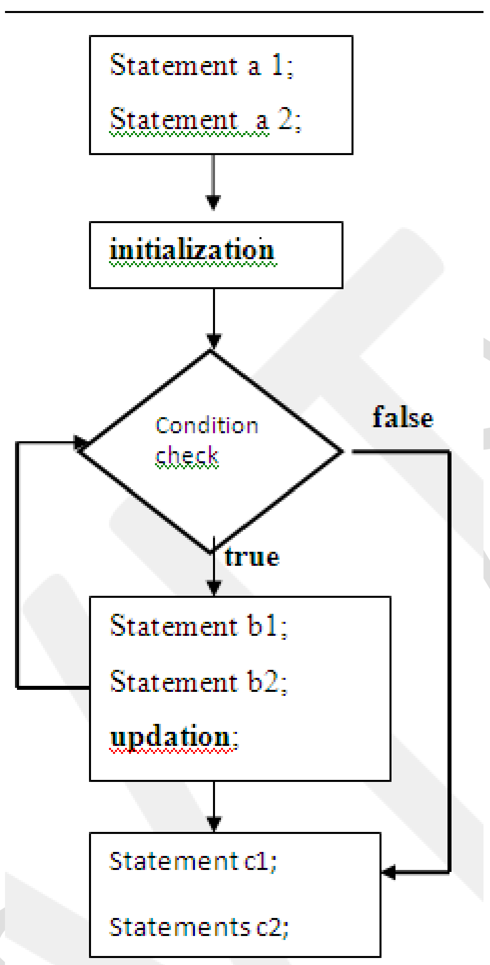
4b. List the types of Looping control statements.Explain any two with example.

Solution:

- The statements that help us to execute a set of statements repeatedly are called **Looping Constructs.**
- It is implemented using the following statements.
    - ➢ ***for***
    - ➢ ***while***
    - ➢ ***do-while***

1.The *while* loop

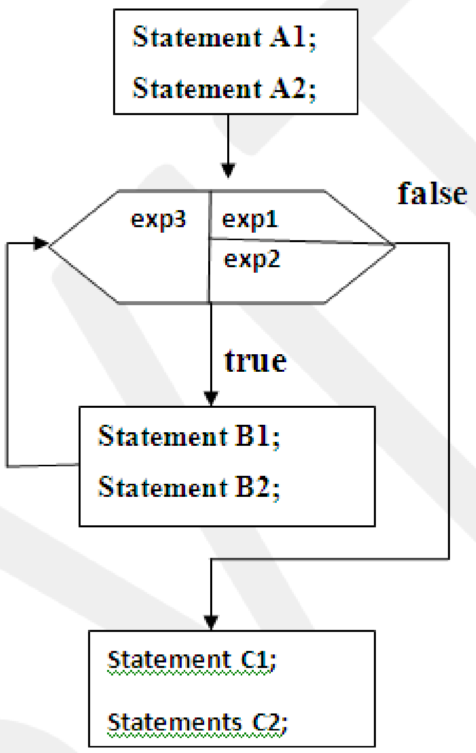
Syntax	Flowchart	Example

Statement a1;
Statement a2;
initialization;
***while*(Condition check)**
{
    Statement b1;
    Statement b2;
    updation;
}
Statement c1;
 	
Statements c2;
	 Statement a 1;
	Statement a 2;
	initialization
	Condition check
	false
	true
	Statement b1;
	Statement b2;
	updation;
	Statement c1;
	Statements c2;
		#include<stdio.h>
		void main( )
		{
		    int i;
		    i=1;
		    while(i<=5)
		    {
		printf("INDIA\n");
		        i++;
		    }
		
		}
		
		<u>Output:</u>
		INDIA
		INDIA
		INDIA
		INDIA
		INDIA

- A while loop is a looping statement which are used to execute a set of statements repeatedly.
- Working:
  - Statements a1 to an are executed in sequence.
  - Condition is checked ,if it is true ,body of while loop, ie statements b1 to bn are executed.
  - Thus the body of loop is repeatedly executed as long as the condition is true.
  - Once the condition/expression is evaluated to false ,the control comes out of loop and statements c1 to c n are executed.
- It is a **pre testing loop and top testing loop and entry controlled loop.**

. The *for* loop

Syntax	Flowchart	Example

Statement A1;
Statement A2;
***for* (exp1 ;exp2; exp3)**
{
Statement B1;
Statement B2;
}
Statement C1;
 	
Statements C2;
	 Statement A1;
	Statement A2;
	exp3
	exp1
	exp2
	false
	true
	Statement B1;
	Statement B2;
	Statement C1;
	Statements C2;
		#include<stdio.h>
		void main()
		{
		int i;
		for(i=1; i<=5; i++)
		{
		printf("INDIA\n");
		i++;
		}
		}
		
		<u>Output:</u>
		INDIA
		INDIA
		INDIA
		INDIA
		INDIA

Definition

- A for loop is a control statement using which the programmer can give instructions to the computer to execute a set of statements repeatedly for a specified number of times.
-

Explanation:

- First (exp1) is Initialization of the loop .
- Next (exp2) condition is checked. If condition is true then the control enters the body of the loop and statements b1 to b2 are executed.
- After executing statements b1 to bn , control goes back and expression 3 (updation) is evaluated.
  (exp3)updation of the loop is performed after which again the control goes back to the condition check, if condition is true, the body is executed again.
- This process repeats as long as the condition evaluates to be true.
- Once the condition evaluates to be false, the control comes outside the body of the loop.

4c. Develop a c program to read a year as an input and find whether it is leap year or not.

/*c program to read a year as an input and find whether it is leap year or not*/

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

     printf("enter a year:\n");
     scanf("%d",&year);

     if((year%4==0 && year%100!=0)||(year%400==0))
          printf("%d is a leap year",year);

     else
          printf("%d is not a leap year\n",year);

}
```

Module 3

5a.What is an array? How to declaration and Initialization of single and two dimensional arrays with example?

Solution:

- Array:An Array is a collection of elements of same datatype grouped under a single name.
- Array is the derived data type. Array is the homogeneous collection of data items. The elements of an array are of same data type and each item can be accessed using the same name.
- A single dimensional array is a linear list of related data items of same data type.
  - In memory, all the data items are stored in contiguous memory locations.

<u>Declaration of one-dimensional array(Single dimensional array)</u>

<u>Syntax:</u>

datatype array_name[size];

- **datatype** can be int,float,char,double.
- **array_name** is the name of the array and it should be an valid identifier.
- **Size** is the total number of elements in array.

For example:

int a[5];

- The above statement allocates 5*2=10 Bytes of memory for the array **a.**

a[0]	a[1]	a[2]	a[3]	a[4]

float b[5];

The above statement allocatests 5*4=20 Bytes of memory for the array **b.**

- ➢ Each element in the array is identified using integer number called as i**ndex.**
- ➢ If n is the size of array, the array index starts from **0** and ends at **n-1.**

Initialization of one-dimensional array

- ➢ **Assigning the required values to an array elements before processing is called initialization.**

data type array_name[expression]={v1,v2,v3…,vn};

Where

- ✓ datatype can be char,int,float,double
- ✓ array name is the valid identifier
- ✓ size is the number of elements in array
- ✓ v1,v2,v3……..vn are values to be assigned.

- ➢ Arrays can be initialized at declaration time.
  Example:
  int a[5]={2,4,34,3,4};

2	4	34	3	4
a[0]	a[1]	a[2]	a[3]	a[4]

Declaration of two dimensional arrays:

- ➢ Syntax:
  Data_type Array_name[row_size][column_size];
  - ✓ Data_type –general data types like - int, char, float, double.
  - ✓ Array_name – valid identifier.
  - ✓ row_size- indicates the subscript value for the maximum number of rows.
  - ✓ column_size - indicates the subscript value for the maximum number of columns.

Example: int arr[3][4];

- ✓ In the above declaration the data type is integer
- ✓ The name of the 2D array is "arr"
- ✓ The maximum number of rows are 3
- ✓ The maximum number of columns are 4
- ✓ The maximum numbers of elements can be stored is 3*4=12 elements.
- ✓

Complete 2D array Initialization:

- ✓ In this kind of initialization all the memory locations will be assigned with a value during declaration itself.

Syntax:

```
Data_type  Array_name [Exp1][Exp2]={
                                          {a1,a2,a3….an},
                                          {b1,b2,b3….bn},
                                          ……………. . .
                                          {z1,z2,z3…….zn}
                                      };
```

Example:

```
int arr[3][4]= {
                { 13,45,63,89},
                { 10,01,90,21},
                { 34,07,81,23}
            };
```

- ➢ **The pictorial representation of the above initialization is shown below**

Column index → 0 1 2 3

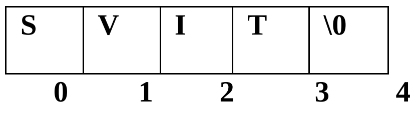
Row index ↓

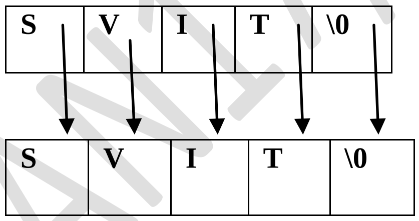
Row index	0	1	2	3
0	**13**	**45**	**63**	**89**
1	**10**	**1**	**90**	**21**
2	**34**	**7**	**81**	**23**

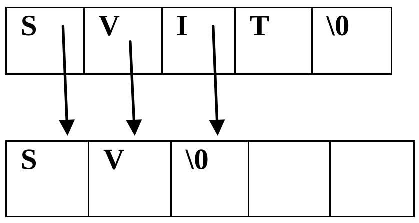
5b.Explain any four string manipulation library function with example.[4M]

Solution:

Name	Syntax	Example	Explanation

strlen	**intstrlen (char str[ ]);**	**char str[15]="SVIT";**
		int count;
		count=strlen(str);
		
		The example strvariable contains 4 characters S,V,I,T , hence count is 4
			-This function returns the length of the string str.
			-It counts all the characters until null character is encountered.

strcpy	**strcpy(char dest[ ] , char src[ ]);**	**char src[5] ="SVIT";**
		char dest[5];
		strcpy(dest ,src);
		src[0] src[1] src[2] src[3] src[4]
		
		dest[0] dest [1] dest [2] dest [3] dest [4]
			✓ **This function copies content from source string to destination string including \0.**
			✓ **Size of dest string should be greater or equal to the size of source string src to store the entire source string.**

strncpy	**strcpy(char dest[ ] , char src[ ],intn);**	**char src[5] ="SVIT";**
		char dest[5];
		strcpy(dest ,src,2);
		src[0] src[1] src[2] src[3] src[4]
		
		dest[0] dest [1] dest [2] dest [3] dest [4]
			✓ **This function copies n characters from source string to destination string .**
			✓ **In this example only 2 characters are copied from src to dest.**

strcat	**strcat(char s1[ ] , char s2[ ]);**	**char s1[5]=”SVIT”;** **char s2[5]=”ECE”;** **strcat(s1,s2);** **S V I T \0** **0 1 2 3 4** **E C E \0** **0 1 2 3 4** **S1 S V I T E C E \0** **0 1 2 3 4 4 6 7 8 9**	✓ **This function concatenates all characters of s2 string to the end of s1 string.** ✓ **The null character of s1 is replaced by first character of s2.** ✓ **Size of s1 string should be greater to store the contents of both the string(s1+s2)**

5c. write a c program to implement string copy operation STRCOPY(str1,str2) that copies string str1 to another string str2 without using library function.[6M]

```
#include<stdio.h>

void main()
{
       char src[20],dest[20];
       int i;

       printf("enter the source string\n");
       scanf("%s",src);
       i=0;
       while(src[i]!='\0')
       {
       dest[i]=src[i];
       i++;
       }
       dest[i]='\0';
       printf("\n source string is %s",src);
       printf("\n destination string is %s",dest);
```

}

OR

6a.what is a string ?write a c program that reads a sentence and prints the frequency of <u>each vowels</u> and total number of consonants.

Solution:

```
/*program to print frequency of  each vowels and
consonants*/
#include<stdio.h>

void main()
{
    char str[80],ch;
    int i,vca=vce=vci=vco=vcu=0,cc=0;

    printf("enter a sentence\n");
    scanf("%s",str);
    i=0;
    for(i=0;str[i]!='\0';i++)
    {
            ch=tolower(str[i]);

            if(isalpha(ch))
            {
                if(ch=='a')
                    vca++;

                else if(ch=='e')
                    vce++;

                else if(ch=='i')
                    vci++;
                else if(ch=='o')
                    vco++;
                else if(ch=='u')
                    vcu++;
                else
                    cc++;
            }

 }
    printf("No. of vowels a is %d\n",vca);
    printf("No. of vowels e is %d\n",vce);
    printf("No. of vowels i is %d\n",vci);
    printf("No. of vowels o is %d\n",vco);
    printf("No. of vowels u is %d\n",vcu);
    printf("No. of consonants in %d\n",cc);

}
```

6b.what is a function?explain the type of function based on parameters.

- A function as series of instructions or group of statements with one specific purpose.
- A function is a program segment that carries out some specific, well defined task.
- A function is a self contained block of code that performs a particular task.

1. **Function with no parameters and no return values**
2. **Function with no parameters and return values.**
3. **Function with parameters and no return values**
4. **Function with parameters and return values**

1. Function with no parameters and no return values

1. Function with no parameters and no return values **(void function without parameter)**	
Calling function	**Called function**

/*program to find sum of two numbers using function*/
#include<stdio.h>
void add( );
void main( )
{
    add( );
}
	void add ( )
	{
	    int sum;
	    printf("enter a and b values\n");
	    scanf("%d%d",&a,&b);
	    sum=a+b;
	    printf("\n The sum is %d", sum);
	    return;

	}
✓ In this category **no data** is transferred from **calling function to called function**, hence called function cannot receive any values. ✓ In the above example,no arguments are passed to user defined function **add( ).** ✓ Hence no parameter are defined in function header. ✓ When the control is transferred from calling function to called function a ,and b values are read,they are added,the result is printed on monitor. ✓ When return statement is executed ,control is transferred from called function/add to calling function/main.	

2. **Function with parameters and no return values (void function with parameter)**	
Calling function	**Called function**

/*program to find sum of two numbers using function*/
#include<stdio.h>
void add(int m, int n);
void main()
{
  int m,n;
  printf("enter values for m and n:");
  scanf("%d %d",&m,&n);
  add(m,n);
}
	void add(int a, int b)
	{
	  int sum;
	  sum = a+b;
	  printf("sum is:%d",sum);
	  return;
	}

✓ In this category, **there is data transfer** from the calling function to the called function using parameters. ✓ **But there is no data transfer from** called function to the calling function. ✓ The values of actual parameters m and n are copied into formal parameters a and b. ✓ The value of a and b are added and result stored in sum is displayed on the screen in called function itself.	

3. Function with no parameters and with return values	
Calling function	**Called function**

/*program to find sum of two numbers using function*/
#include<stdio.h>
int add();
void main()
{
int result;
result=add( );
printf("sum is:%d",result);
}
	int add( ) /* function header */
	{
	int a,b,sum;
	printf("enter values for a and b:");
	scanf("%d %d",&a,&b);
	sum= a+b;
	return sum;
	}

✓ In this category **there is no data transfer from the calling function to the called function.** ✓ But, there is data transfer from called function to the calling function. ✓ No arguments are passed to the function add( ). So, no parameters are defined in the function header ✓ When the function returns a value, the calling function receives one value from the called function and assigns to variable result. ✓ The result value is printed in calling function.	

4. Function with parameters and with return values	
Calling function	**Called function**

/*program to find sum of two numbers using function*/
`#include<stdio.h>`
`int add();`
`void main()`
`{`
`    int result,m,n;`
`    printf("enter values for m and n:");`
`    scanf("%d %d",&m,&n);`
`    result=add(m,n);`
`    printf("sum is:%d",result);`
`}`
	`int add(int a, int b) /* function header */`
	`{`
	`    int sum;`
	`    sum= a+b;`
	`    return sum;`
	`}`

✓ In this category, there is data transfer between the calling function and called function. ✓ When Actual parameters values are passed, the formal parameters in called function can receive the values from the calling function. ✓ When the add function returns a value, the calling function receives a value from the called function. ✓ The values of actual parameters m and n are copied into formal parameters a and b. ✓ Sum is computed and returned back to calling function which is assigned to variable result.	

6c.what is recursion?write a c program to compute <u>polynomial co efficient</u> c using recursion.

- Recursion is a method of solving the problem where the solution to a problem depends on solutions to smaller instances of the same problem.
- Recursive function is a function that calls itself during the execution.
- The two types of recursion are
  1.Direct Recursion
  2. Indirect Recursion

Note:polynomial coefficient means ncr program

```
/*program to compute ncr binomial coefficient*/
#include<stdio.h>
int fact(int n);
void main()
{
     int n,r;
     float ncr;

     printf("enter the value of n and r\n");
     scanf("%d%d",&n,&r);
     ncr=(float)fact(n)/(fact(n-r)*fact(r));
     printf("Binomial Coefficient of %d C%d=%f\n",n,r,ncr);
}

int fact(int n)
{
     if(n==0)
          return 1;
     else
          return (n*fact(n-1));
}
```

MODULE 4

7a.what is structure? Explain the c syntax of structure declaration with example.[4M]

- A Structure is a collection of one or more variables of same datatype or dissimilar datatype, grouped together under a single name for convienient handling.

syntax
struct tag_name
{
type1 member1;
type2 member2;
-------------------- 	
-------------------- 	
};
	Example
	struct student
	{
	char name[20];
	int usn;
	float marks;
	};

Where

- struct is a keyword which informs the compiler that a structure is being defined.
- tagname name of a structure.
- members of structure are member1,member2
  type1,type2 can be int,float,char,double.

Structure declaration and initialization.
Three ways of declaring structure are:

1. Tagged structure
2. Structure without tag
3. Type defined structures

1.Tagged structure	

syntax
struct tag_name
{
type1 member1;
type2 member2;
-------------------- 	
-------------------- 	
};
	Example
	struct student
	{
	char name[20];
	int usn;
	float marks;
	};

declaring structure variables	
struct tagname v1,v2,v3…vn;	struct student s1,s2,s3;

2.structure without tagname	

syntax
struct
{
type1 member1;
type2 member2;
-------------------- 	
-------------------- 	
}v1,v2,v3;
	Example
	struct
	{
	char name[20];
	int usn;
	float marks;
	}s1,s2;

3.Type defined structure	

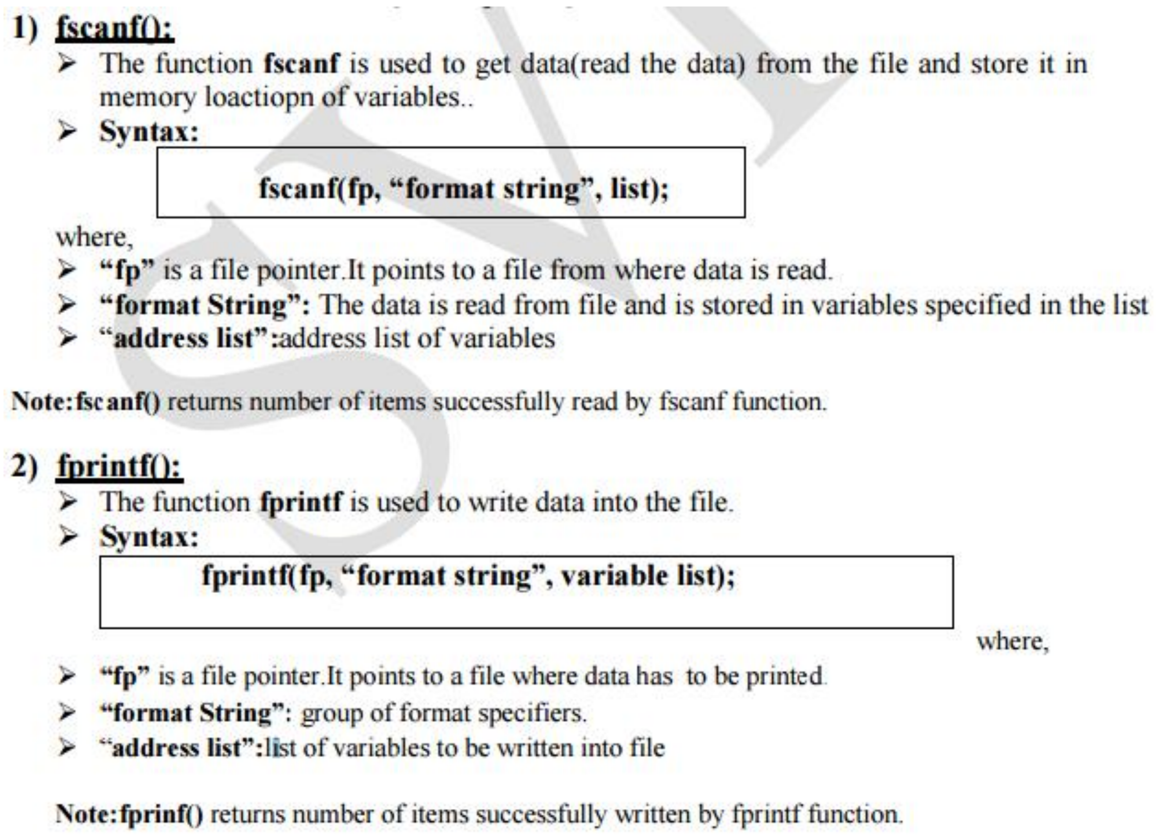
syntax
typedef struct
{
type1 member1;
type2 member2;
-------------------- 	
-------------------- 	
}**TYPE_ID;**
	Example
	typedef struct
	{
	char name[20];
	int usn;
	float marks;
	}**STUDENT;**

Declaring structure variables	
TYPE_ID v1,v2,v3…vn;	STUDENT s1,s2,s3;

7b. what is a FILE? Explain any five file manipulation functions with example.[6M]

Solution:

- **File:** A file is defined as a collection of data stored on the secondary device such as hard disk. **FILE** is type defined structure and it is defined in a header file "stdio.h".
- FILE is a derived data type.



1) fscanf():

- The function **fscanf** is used to get data(read the data) from the file and store it in memory loactiopn of variables..
- **Syntax:**

```
fscanf(fp, "format string", list);
```

where,

- **"fp"** is a file pointer.It points to a file from where data is read.
- **"format String":** The data is read from file and is stored in variables specified in the list
- "**address list**"**:**address list of variables

Note:fscanf() returns number of items successfully read by fscanf function.

2) fprintf():

- The function **fprintf** is used to write data into the file.
- **Syntax:**

```
fprintf(fp, "format string", variable list);
```

where,

- **"fp"** is a file pointer.It points to a file where data has to be printed.
- **"format String"**: group of format specifiers.
- "**address list**"**:**list of variables to be written into file

Note:fprinf() returns number of items successfully written by fprintf function.

3.fgets()

fgets() is used to read a string from file and store in memory.

Syntax:

ptr=fgets(str,n,fp);

where

fp ->file pointer which points to the file to be read

str ->string variable where read string will be stored

n ->number of characters to be read from file

ptr->If the operation is successful, it returns a pointer to the string read in. Otherwise it returns NULL.

The returned value is copied into ptr.

4.fputs()

fputs() is used to write a string into file.

Syntax:

fputs(str,fp);

where

fp ->file pointer which points to the file to be read

str ->string variable where read string will be stored

7c.what are Actual parameters and formal parameters?explain various storage classes available in c.[6M]

Solution:

Actual Parameters	**Formal Parameters**
Actual parameters are also called as **argument list.** Ex: add(m,n)	Formal parameters are also called as **dummy parameters.** **Ex:**int add(int a, int b)
The variables used in function call are called as actual parameters	The variables defined in function header are called formal parameters
Actual parameters are used in calling function when a function is called or invoked Ex: add(m,n) Here, m and n are called actual parameters	Formal parameters are used in the function header of a called function. Example: int add(int a, int b) { ……….. } Here, a and b are called formal parameters.
Actual parameters sends data to the formal parameters Example:	Formal parameters receive data from the actual parameters.

OR

8a.Explain array of structure and structure within a structure with example.[6M]

Solution:

Nested Structures

- A structure **which includes another structure** is called nested structure.

struct subject
{
int marks1;
int marks1;
int marks1;
};
	Structure definition with tagname subject

typedef struct
{
        char name[20];
        int usn;
        struct subject PCD;
} STUDENT;
STUDENT s1;
	//Structure definition with TYPEID **STUDENT**
	// include structure subject with a member name **PCD**

- Structure student has member called **PCD**, which inturn is a structure .
- Hence **STUDENT** structure is a nested structure.
- We can access various members as follows:

  S1.name;

  S1.usn;

  S1.PCD.marks1;

  S1.PCD.marks2;

  S1.PCD.marks3;

Array of structure

- As array of integers we can have array of structures
- Suppose we want to store information of 5 students consisting of name,usn,marks,structure definition is as follows:

```
typedef struct

{

        char name[20];

        int usn;

        float marks;

} STUDENT;

STUDENT  s[5];
```

If n=3

- We can access the first student information as follows:

  S[0].name

  S[0].usn

  S[0].marks

- We can access the second student information as follows
  S[1].name
  S[1].usn
  S[1].marks
- We can access the second student information as follows
  S[2].name
  S[2].usn
  S[2].marks

hence i ranges from 0 till 2 if n=3

```
for(i=0;i<n;i++)
{
            printf("enter the %d student details\n",i+1);
                printf("enter the roll number:\n");
                scanf("%d",&s[i].rollno);
                printf("enter the student name:\n");
                scanf("%s",s[i].name);
                printf("enter the marks:\n");
                scanf("%d",&s[i].marks);
                printf("enter the grade:\n");
                scanf("%s",s[i].grade);
}
```

8b.write a c program to maintain a record of n students ,using array of structures with four fields name,rollno,grade,marks of student using structure.print the marks of a student given the student name as input.[8M]

Solution:

```
/*program to print marks  given name of a students*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct student
{    int rollno,marks;
     char name[20];
     char grade[2];
};
void main()
{         int i,n;
          struct student s[10];
          char key[20];
          clrscr();
          printf("enter the number\n");
          scanf("%d",&n);
          for(i=0;i<n;i++)
          {
               printf("enter the %d student details\n",i+1);
```

```
                printf("enter the roll number:\n");
                scanf("%d",&s[i].rollno);
                printf("enter the student name:\n");
                scanf("%s",&s[i].name);
                printf("enter the marks:\n");
                scanf("%d",&s[i].marks);
                printf("enter the grade:\n");
                scanf("%s",&s[i].grade);
           }
     printf("enter the key student name:\n");
     scanf("%s",key);
     for(i=0;i<n;i++)
     {
           if(strcmp(s[i].name,key)==0)
           {    printf("\n   marks   of   the   student   is   %d
\n",s[i].marks);
                exit(0);
           }
     }
     printf("student name NOT FOUND\n");
}
```

8c. Explain various modes of a file with example.[4M]

Solution:

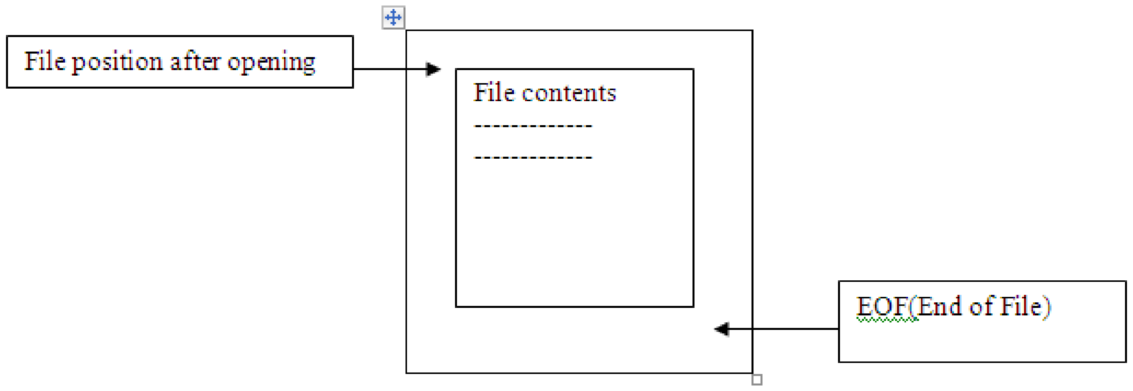
Modes of File

The various mode in which a file can be opened/created are:

Mode	**Meaning**
"r"	opens a text file for reading. The file must exist.
"w"	creates an text file for writing.
"a"	Append to a text file.

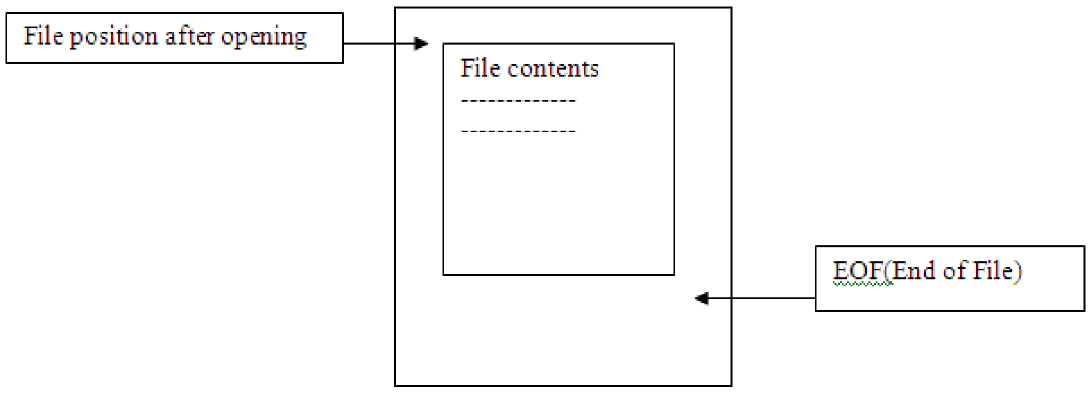
1.Read Mode("r")

- This mode is used for opening an existing file to perform read operation.
- The various features of this mode are
    - Used only for text file
    - If the file does not exist, an error is returned
    - The contents of the file are not lost
    - The file pointer points to beginning of the file.



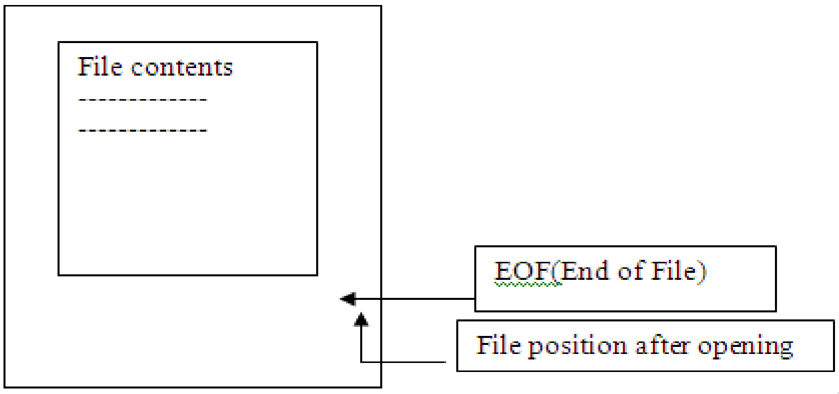
2.Write Mode("w")

- This mode is used to create a file for writing.
- The various features of this mode are
    - If file does not exist, a new file is created.
    - If a file already exists with the same name, the contents of the file are erased and the file is considered as a new empty file.
    - The file pointer points to the beginning of the file.



3.Append Mode("a")

- This mode is used to insert the data at the end of the existing file.
- The various features of this mode are
- Used only for text file.
- If the file already exist, the file pointer points to end of the file.
- If the file is does not exist. A new file is created.
- Existing data cannot be modified.



MODULE 5

9a. what is a pointer ?explain how the pointer variable is declared and initialized[4M]

Solution:

- **A pointer is a variable which contains the address of another variable.**
  <u>Advantages of pointer</u>
- Enables us to access a variable that is defined outside the function.
- Can be used to pass information back and forth between a function and its reference point.
- More efficient in handling data tables.
- Reduces the length and complexity of a program.
- Sometimes also increases the execution speed.

Declaring of a pointer variable

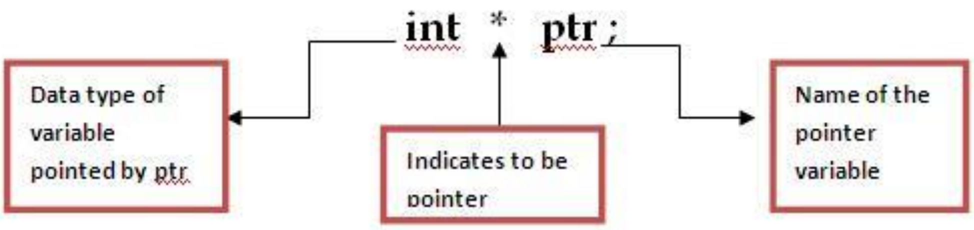
General syntax :

```
data_type *pointer_name;
```

where,

- The asterisk (*) indicates that the variable pointer_name is a pointer variable.
- Pointer_name is a identifier.
- pointer_name needs a memory location.
- pointer_name points to a variable of type data_typewhich may be int, float, double etc..

Example:


int * ptr ;
Data type of variable pointed by ptr
Indicates to be pointer
Name of the pointer variable

where

- ptr is not an integer variable but ptr can hold the address of theinteger variable
- i.e. it is a **pointerto an integer variable'** ,but thedeclaration of pointer variable does not make them point to any location .
- We can also declare the pointer variables of any other data types .

Example:

```
double  *dptr;
char * ch;
float *  fptr;
```

Dereference operator (*)

- **The unary operator (*) is the dereferencing pointer or indirection pointer , when applied to the pointer can access the value the pointer points to.**

Initialization of pointers

- Initializing a pointer variable is a important thing, it is done as follows:

Step 1: Declare a data variable
Step 2:Declare a Pointer variable
Step 3:Assign address of data variable to pointer variable using & operator and assignment operator.
Example:

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

9b. what is dynamic memory allocation?explain different dynamic memory allocation functions in c.[6M]

Dynamic Memory allocation

- Dynamic memory allocation is the process of allocating memory during run time(execution time).
- Data structures can grow and shrink to fit changing data requirements.

1. **malloc( ):**
   - Allocates requested number of bytes and returns a pointer tothe first byte of the allocated space.
   - **Syntax:**
     ptr=(datatype *)malloc(size);
     where,
     - ✓ ptr is a pointer variable of type datatype
     - ✓ datatpe can be any of the basic datatype or user define datatype
     - ✓ Size is number of bytes required.

Example:

int *p;

p=(int *)malloc(sizeof(int));

2. **calloc( ):**

- It allocates a contiguous block of memory large enough to contain an array of elements of specified size. So it requires two parameters as number of elements to be allocated and for size of each element. It returns pointer to first element of allocated array.
- **Syntax:**

  ptr=(datatype *)calloc(n,size);

  where,

  - ✓ ptr is a pointer variable of type datatype
  - ✓ datatype can be any of the basic datatype or user define datatype
  - ✓ n is number of blocks to be allocated
  - ✓ Size is number of bytes required.

Example:

int *p;

p=(int *)calloc(sizeof(5,int));

3.realloc( )

- realloc() changes the size of block by deleting or extending the memory at end of block.
- If memory is not available it gives complete new block.

Syntax:

ptr=(datatype *)realloc(ptr,size);

where,

- ✓ ptr is a pointer to a block previously allocated memory either using malloc() or calloc()
- ✓ Size is new size of the block.

Example:

int *p;
p=(int *)calloc(sizeof(5,int));
p=(int *)realloc(p,sizeof(int *8));

If enough space doesn't exist in memory of current block to extend, new block is allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

4.free()

This function is used to de-allocate(or free) the allocated block of memory which is allocated by using functions malloc(),calloc(),realloc().
It is the responsibility of a programmer to de-allocate memory whenever not required by the program and initialize **ptr**to **Null.**

Syntax:

free(ptr);

9.c.write a c program to find sum and mean of all elements in an array using pointers.[6M]

```
/*program to compute the sum, mean and sd */
#include<stdio.h>
#include<math.h>

void main()
{
    int n,i;
    float a[20],sum, mean, var, sd;

    printf("\n enter size of array");
    scanf("%d",&n);

    printf("\n Enter array elements");
    for(i=0;i<n;i++)
     {
         scanf("%f",(a+i));
      }
     sum=0;
    for(i=0;i<n;i++)
     {
         sum=sum+ *(a+i);
     }
     printf("\n sum=%f",sum);
    mean=sum/n;
    sum=0;
    for(i=0;i<n;i++)
     {
         sum=sum+(*(a+i)-mean)*(*(a+i)-mean);
     }
    var=sum/n;
    sd=sqrt(var);
    printf("\n Mean=%f\n" ,mean);
    printf("\n Variance =%f\n",var);
    printf("\n standard deviation is %f\n",sd);

}
```

OR

10a.explain the array of pointers with example.

5.7Pointer and arrays

When an array is declared,

- The compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.
- The base address is the location of the first element (index 0) of the array.
- The compiler also defines with **samearray name** as a **constant pointer .**
- **Assigns the base address /starting address to pointer variable**
- There is a strong relationship between the pointer and the arrays.
- Any operation which can be performed using array subscripting can also be performed using pointers .
- The pointer version will be generally faster than the array version of that program.
- To access the address of any element use **(a+i) instead of &a[i].**
- To access the element of array use *(a+i) instead of a[i].

Program to read and display array elements using pointers

```
    #include<stdio.h>
void main()
    {
          inta[100], i, n;
           printf("enter number of elements");
           scanf ("%d", &n);
           printf("enter array elements");
           for (i=0; i<n; i++)
                  scanf("%d",(a+i));
           printf("enter array elements are");
           for (i=0; i<n;i++)
                  printf("%d",*(a+i));
    }
```

10b.explain any two pre processor directives in c

Solution:

- The various types of preprocessor directives are:

1. Symbolic Names
2. Macros
3. File Inclusion

4. Conditional Compilation

symbolic constant:

These are the names which are used to define the names for a constant values. The "#define" directive is used to specify the constant hence it is also termed as defined constant.

Example:

#define max 30

where,

#define specifies the directive
max specifies the name of the constant
30 indicates the constant value assigned to max

consider the following program

```
#include<stdio.h>
#define pi 3.142
void main()
{
        printf("value is %f",pi);
}
```

Defining macros:

Macro is the name given to the group of statements. When a macro is included in the program , the program replaces the set of instructions defined. The #define directive is used to define macro.

Syntax:

#define <macro> (set of instructions)

for example

1. maximum of two variables can be written as
   #define max(a,b) ((a) > (b) ? (a):(b))

2. converting degrees into radians
   #define deg_to_rad(x) (x*m_pi/180.0)

Include preprocessor directive

#include specifies to insert the content of the specified files to the program.

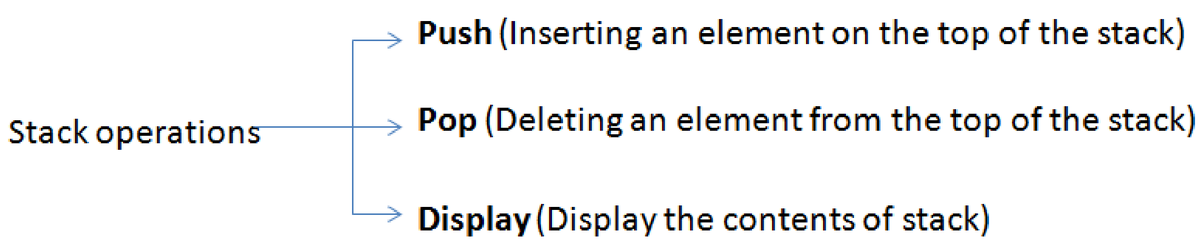
➔ This directive includes a file into code.
➔ It has two possible forms:
   #include <file>

10 c. what is stack? Explain its operation with examples.[4M]

Solution:

Stacks

➔ A stack is a linear data structure in which an element may be inserted or deleted only at one end called the top end of the stack i.e. the elements are removed from a stack in the reverse order of that in which they were inserted into the stack.
➔ A stack follows the principle of last-in-first-out (LIFO) system. The various operations that can be performed on stacks are shown below:


Stack operations
Push (Inserting an element on the top of the stack)
Pop (Deleting an element from the top of the stack)
Display (Display the contents of stack)

- **Representation of Stacks**

A stack may be represented by means of a one way list or a linear array. We need to define two more variables the size of stack i.e, MAX and the entry or exit of stack i.e., top.

Push: inserting the element to the stack

- Here we need to check the stack overflow condition which means whether the stack is full.
- To insert an element two activities has to be done
  1. Increment the top by 1
  2. Insert the element to the stack at the position top.

Pop: deleting an element from the stack

- Here we need to check the stack underflow condition which means whether the stack is empty or not.
- To delete the element we need to perform following operations.
  1. Access the top element from the stack.
  2. Decrement top by 1

Display: printing the elements of the stack

- Here we need to check the stack underflow condition which means whether the stack is empty or not. If empty there will be no elements to display.
- Display the elements from the 0th position of stack till the stack top.

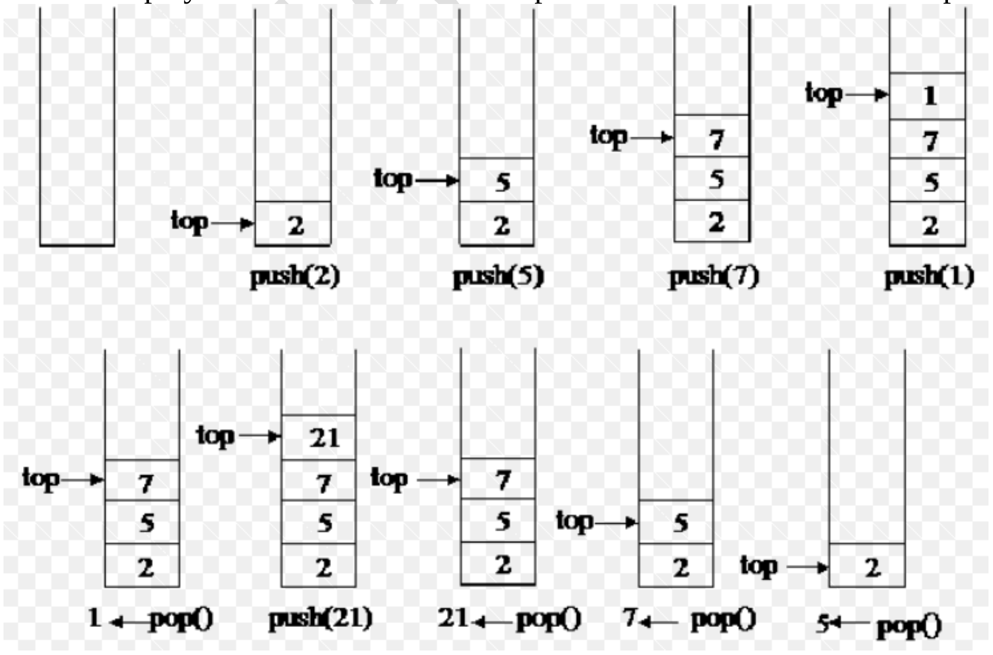
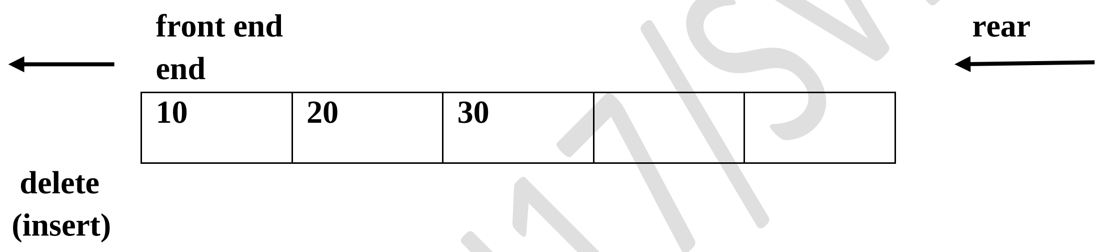


Fig: Representation of push and pop operation.

10d.what is a queue?explain its applications.[4M]

- Queue is a linear data structure in which insertion can take place at only one end called **rear end** and deletion can take place at other end called **top end**.
- The front and rear are two terms used to represent the two ends of the list when it is implemented as queue.
- Queue is also called First In First Out (FIFO) system since the first element in queue will be the first element out of the queue.
- Different Types of queues are
  1. **Queue (ordinary queue)**
  2. **Circular queue**
  3. **Double ended queue**
  4. **Priority queue**



1. **Queue**

- Here the elements are inserted from one end and deleted from other end. The inserting end is the rear end and the deleting end is the front end.
- The operations that can be performed on queue are.
  a) **Insert an element at rear end**
  b) **Delete an element from the front end**
  c) **Display elements**

Applications of Queues:

a) Number of print jobs waiting in a queue, when we use network printer. The print jobs are stored in the order in which they arrive. Here the job which is at the front of the queue, gets the services of the network printer.
b) Call center phone system will use a queue to hold people in line until a service representative is free.
c) Buffers on MP3 players and portable CD player, iPod playlist are all implemented using the concept of a queue.
d) Movie ticket counter system, it maintains a queue to take tickets based on first come first serve means who is standing first in a queue, that person will get the ticket first than second person, and so on.

ALL THE BEST

****