PROGRAMMING IN C AND DATA STRUCTURES

June/July 2016 Question Paper Solution

MODULE 1

1 a. Define pseudocode. Write a pseudocode to find sum and average of given three numbers. [5M]

Answer

➔ Pseudocode is series of steps to solve a given problem written using a mixture of English and C language.

- It is the first step in writing a program.
- It is written using a mixture of English and C language.
- It is a series of steps to solve a given problem
- It acts as a problem solving tool.

<u>Advantage:</u>

- Easy to write and understand
- It is relatively easy to convert English description solution of small programs to C program.

<u>Disadvantage:</u>

- It is very difficult to translate the solution of lengthy and complex problem in English to C.

Sum and average of three numbers

```
➢ Get the numbers[a,b,c]
➢ Compute addition [Sum= a + b+c]
➢ Compute average[avg=sum/n]
➢ Print the results [Sum].
```

1b.what is an identifier? What are the rules to constructs identifier? Classify the following as valid and invalid identifiers. [6M]

Answer:

<u>Identifiers :</u>

- Identifiers are the names given to program elements such as variables, functions and arrays.
- These are user defined names and do not have fixed meaning.

Rules for identifiers/variables

- First character must be an alphabet or an underscore(_)
- First character is followed by any number of letters or digits.
- No extra symbols are allowed other than letters,digits and "_"
- Only first 31 characters are significant
- Keywords cannot be used as an identifier.
- Must not contain blank space/white space.
- Identifiers are case sensitive.
- Must contain only alphabets, digits and one special symbol underscore.

i)num2 : **valid**
ii)$num1 : **Invalid** (Identifier cannot start with symbol $).
iii)+add : **Invalid**(Identifier cannot start with symbol +).
iv)a_2: **Valid**.

1c.Write a C Program to find area of rectangle. [5M]

Answer:

Program	TRACING

/* program to find area of rectangle */
#include<stdio.h>
#include<conio.h>
void main()
{
float l,b,area;
clrscr();
printf("enter the length and breadth of rerctangle\n");
scanf("%f%f",&l,&b);
arera=l*b;
printf("the area of rectangle is %f\n",area);
}
	enter the length and breadth of rectangle
	4 5
	Computation
	
	The area of rectangle is 20.0

2a. Explain printf and scanf functions with example. (04 Marks)

Answer:

printf()

- The function printf means print the data in the specified memory locations after formatting the data.
- **The general syntax of printf() function is shown below:**

```
printf("format string ",list of variables);
printf("%d %f %c ",x,y,z);
```

- The format string also called control string is enclosed within two double quotes"…."
- The format string may contain:
- The sequence of characters to be displayed on the screen.
- It may also contain zero or more format specifiers.A format specifier starts with % sign and is followed by conversion code.

Scanf()

- scanf( ) is used to enter the input through the input devices like keyboard we make use of scanf statement.
- General Syntax:

```
scanf ("format string", list of address of variables);
```

Where:

✓ Format string consists of the access specifiers/format specifiers.

✓ List of addresses of variables consist of the variable name preceded with & symbol.

- Example:

```
int a;
float b;
scanf("%d %f",&a,&b);
```

2b. List all the operators used in C. Give example (8 Marks)

Answer:

The following operators are available in C:

1. **Arithmetic operators.**
2. **Relational operators**
3. **Logical operators**
4. **Assignment operators**
5. **Increment and decrement operators**
6. **Conditional operators**
7. **Bitwise operators**
8. **Special operators**

1) Arithmetic operators

C provides all basic arithmetic operators.

The operators are +, - ,* , / , %.

Description	Symbol	Example	Result	Priority
Addition	**+**	a+b = 10 + 5	15	2
Subtraction	**-**	a-b = 10 – 5	5	2
Multiplication	*****	a* b = 10 * 5	50	1
Division(Quotient)	**/**	a/b = 10/5	2	1
Modulus(Remainder)	**%**	a % b = 10 % 5	0	1

Let a=10, b=5. Here, **a and b are variables and are known as operands.**

2) Relational Operators

An expression such as **a<b or 1<20** Containing a relational operator is termed as a relational expression.

The value of relational expression is either one or zero.

It is one if the specified relation is **true** and zero if the relation is **false**.

Example: 10 < 20 is true.

20 < 10 is false.

C supports six relational operators in all. These operators are

Operator	**Meaning**	**Priority**

<	is less than	1
<=	is less than or equal to	1
>	is greater than	1
>=	is greater than or equal to	1
==	is Equal to	2
!=	is not equal to	2

3) Logical Operators

In addition to the relational operators, C has the following three logical operators.

Operator	*Meaning*
&&	**logical AND**
||	**logical OR**
!	**logical NOT**

❖ **logical AND (&&)**

The output of logical operation is true if and only if both the operands are evaluated to true.

operand 1	**AND**	**operand 2**	**result**
True (1)	**&&**	**True(1)**	**True(1)**
True (1)	**&&**	**False(0)**	**False(0)**
False(0)	**&&**	**True(1)**	**False(0)**
False(0)	**&&**	**False(0)**	**False(0)**

❖ **logical OR (||)**

The output of logical operation is false if and only if both the operands are evaluated to false.

operand 1	**OR**	**operand 2**	**result**
True (1)	**||**	**True(1)**	**True(1)**

True (1)	||	**False(0)**	**True(1)**
False(0)	||	**True(1)**	**True(1)**
False(0)	||	**False(0)**	**False(0)**

❖ **logical NOT**

It is a unary operator. The result is true if the operand is false and the result is false if the operand is true.

The logical operators && and | | are used when we want to test more than one condition and make decisions.

An example: a>b && x==10

The above logical expression is true only if a>b is true and x==10 is true. If either (or both) of them are false, the expression is false.

4) Assignment Operators

- Assignment operators are used to assign the result of an expression to a variable.
- assignment operator,"=".

Variable=expression;

Ex:

a= 10; //RHS is constant

a=b; // RHS is variable.

a=b+c; //RHS is an expression

5) Increment and Decrement Operators

- C allows two very useful operators not generally found in other languages. These are the increment and decrement operators:

++ and --

- The operator ++ adds 1 to the operand, while -- subtracts 1.
- Both are unary operators takes the following.

++m; or m++;

--m; or m--;

++m is equivalent to m=m+1;

--m is equivalent to m=m-1;

6) **Conditional Operator (OR) Ternary Operator**

A ternary operator pair **“? :”** is available in C to construct conditional expressions of the form

exp1? exp2: exp3

where exp1,exp2, and exp3 are expressions.it operates on three operands.

7) **Bitwise Operators**

bitwise operatorsare used to manipulate the bits of given data.These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double.

Operator	*Meaning*
&	**Bitwise AND**
|	**Bitwise OR**
^	**Bitwise exclusive OR**
<<	**Shift left**
>>	**Shift right**
~	**Bitwise Negate**

8) **Special Operators**

C supports some special operators of interest such as comma operator, sizeof operator.

- ***The Comma Operator***
- ***The sizeof() Operator***

2c. Write the output of the following C code. **[04M]**

Answer:

i) **void main()**

```
{
        int a=5,b=2,res1;
        float f1=5.0,f2=2.0,res2;
        res1=5/2.0 + a/2 + a/2;
```

```
            res2 = f1/2 * f1 – f2;
            printf("res1 = %d res2 = %f",res1,res2);
    }
```

Output:

```
res1 = 6
res2 = 10.500000
```

ii)
```
    void main()
    {
            int i=5,j=6,m,n;
            m = ++i + j++;
            n = --i + j --;
            printf("m = %d  n =%d, m,n);
    }
```

Output:

```
m=12
n=12
```

MODULE 2

3a. List all the conditional control statement used in C. Write a C program to find the biggest of three numbers. [08M]

Answer:

The conditional control statements used in C are:

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

```
#include<stdio.h>
void main()
{
      int  a,b,c;
      printf(" enter the 3 numbers\n")
      scanf("%d%d%d",&a,&b,&c);
       if(a>b)
      {
          if(a>c)
          {
                printf("a largest");
          }
         else
         {
                printf("c largest");
          }
      }
     else
      {
            if(b>c)
           {
                printf("b largest");
            }
           else
            {
                printf("c largest");
            }
        }
}
```

3b. Write a C Program to find the reverse of an interger number NUM and check whether it is PALINDROME or NOT. [08M]

Answer:

```
#include<stdio.h>
void main()
{
           int  n,rev=0,rem,temp;
          printf("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);

}
```

<u>Output1:</u>

```
enter the number
2345
The reversed number is 5432
5432 is not  a PALINDROME
```

<u>Output2:</u>

```
enter the number
2112
The reversed number is 2112
2112 is not  a PALINDROME
```

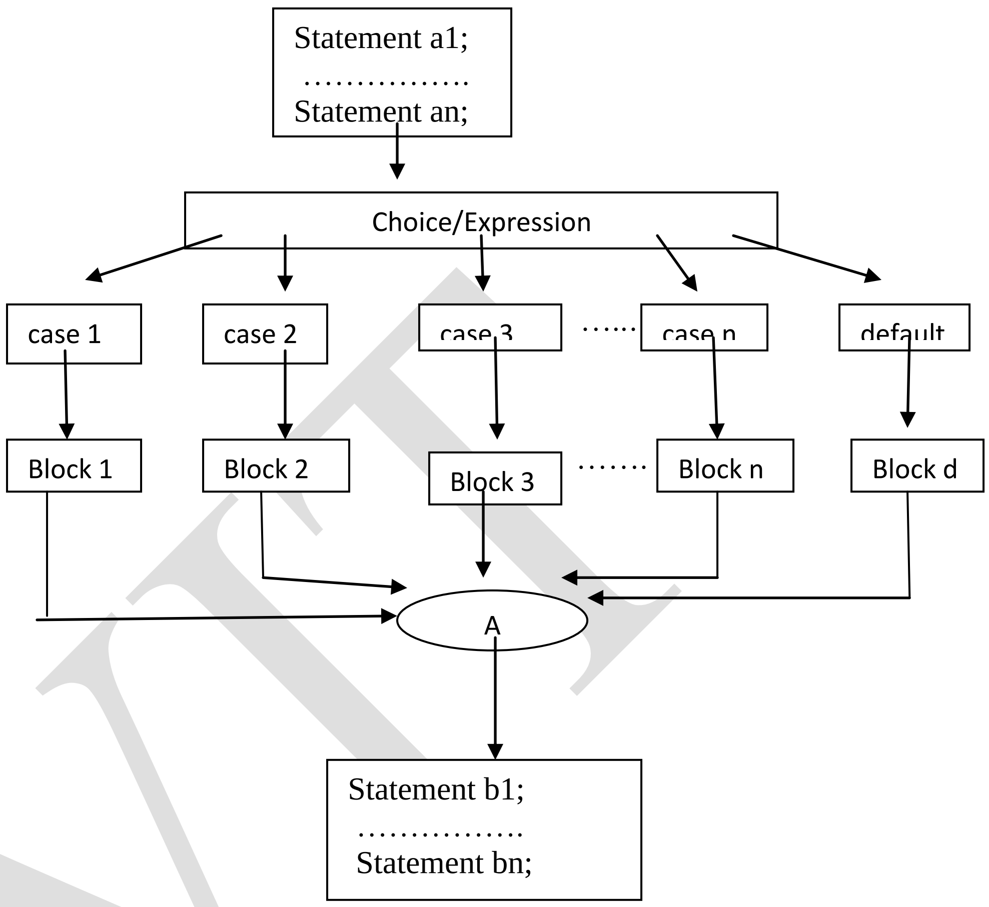
4a.Explain the switch statement with syntax and example. **[08M]**

Answer:
Switch statements are used in following scenarios

- When a decision has to be made between many alternatives
- When the selection condition reduces to an integer value.

<u>Definition</u>: The switch statement is a control statement used to make a selection between many alternatives. i.e multiple selection mechanism is implemented using *switch*.

- ***switch*** is alternative for two way selection multiple ***if-else-if***.

Statement a1;
…………….
Statement an;
 	
switch(choice)
{
case value1: Block1;
break;
case value2: Block 2;
break;
case value3: Block3;
break;
default: Block d;
}
 	
Statement b1;
…………….
Statement bn;
	 Statement a1;
	…………….
	Statement an;
	Choice/Expression
	case 1
	case 2
	case 3
	……
	case n
	default
	Block 1
	Block 2
	Block 3
	…….
	Block n
	Block d
	A
	Statement b1;
	…………….
	Statement bn;

Explanation

- The switch statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly.
- Each case is labeled by one or more integer-valued constants or constant expressions.
- If a case matches the expression value, execution starts at that case.
- All case expressions must be different.
- The case labeled default is executed if none of the other cases are satisfied.

- A default is optional; if it isn't there and if none of the cases match, no action takes place. Cases and the default clause can occur in any order.
- The break statement causes an immediate exit from the switch.
- The expression of a switch statement must result in an *integral type*, meaning an integer (byte, short, int, long) or a char.
- The expression cannot be a Boolean value or a floating point value (float or double)
- No two case labels can have the same constant value.

An Example which illustrates switch control statement:

Program to simulate a simple calculator that performs arithmetic operations only on integers. Error message should be reported if any attempt is made to divide by 0.

```
#include<stdio.h>
void main()
{
    int a,b,res;
    char choice;
    printf(“ Enter your choice\n”);
    scanf(“%c”,&choicec);
    printf(“ Enter two operands\n”);
    scanf(“%d%d”,&a,&b,);
    switch(choice)
    {
         case ‘+’:   res = a+ b;
                     printf(“%d”, res);
                      break;
         case ‘-’:   res = a- b;
                     printf(“%d”, res);
                     break;
         case ‘*’:   res = a-*b;
                     printf(“%d”, res);
                     break;
          case ‘/’:  if(b==0)
                   {
                       printf(“error :Divided by   0”);
                     }
                    else
                    {
                       res = a/ b;
                       printf(“%d”, res);
                     }
```

```
                  break;
        default: printf( "invalid op");
          }
    }
```

Advantages:
1. Improves readability of a program.
2. More Structured way of writing program.

Disadvantages:
1. Used only if the expressions used for checking results in integer value.
2. Cannot be used when a decision is based on range of values.

4b.List the differences between the while loop and do while loop. Write a C program to find sum of natural numbers from 1 to N using for loop. (08M)

Answer

Differences between the while loop and do while loop

while loops ***pretest loop***	***do while loops*** ***post test loop***

Syntax
 	
 Statement a1;
 Statement a2;
 initialization;
 ***while*(Condition check**)
 {
         Statement b1;
         Statement b2;
         updation;
 }
 Statement c1;
 Statements c2;
	Syntax
	
	 Statement a 1;
	 Statement a2;
	 initialization;
	   do
	 {
	         Statement b1;
	         Statement b2;
	         updation;
	 } ***while*(Condition check);**
	 Statement c 1;
	 Statements c 2;

It is a **top testing/entry controlled loop** since the condition is checked in the beginning itself.	It is a bottom testing /exit controlled loop since the condition is checked in the bottom of the loop.
It is a **pre test loop**, so if the expression is false in the beginning itself, the statements within the body of the loop will not be	It is a **post test loop,**and the body of the loop will be executed atleast once.

executed.	
While loop is a Entry controlled loop ,because the condition is checked first and if that condition is true than the block of statement in the loop body will be executed	***Do while*** loop is a Exit controlled loop the body of loop will be executed first and at the end the test condition is checked, if condition is satisfied then body of loop will be executed again.

C program to find sum of natural numbers.

	Tracing

```
/* c program tp find sum of natural numbers.*/
#include<stdio.h>
void main()
{
        int i,n;
        printf("enter the value of n\n")
        scanf("%d",&n);
        sum=0;
        for(i=1;i<=n;i++)
        {
                sum=sum+i;
        }
        printf("sum=%d",sum);
}
```
	Enter the value of n
	4
	
	i= 1,2,3,4,5
	
	sum=1+2+3+4+5
	
	sum=15

Module 3

5a.what is an array ?Explain the declaration and initialization of single and double dimensional arrays withy example. [08M]

Answer:

Arrays: Array is a sequential collection of similar data items.

Pictorial representation of an array of 5 integers

10	20	30	40	50
A[0]	A[1]	A[2]	A[3]	A[4]

- An array is a collection of similar data items.
- All the elements of the array share a common name .
- Each element in the array can be accessed by the subscript(or index) and array name.

- The arrays are classified as:
  1. **Single dimensional array**
  2. **Multidimensional array.**

Single Dimensional Array.

- 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.

Declaration of one-dimensional array(Single dimensional array)

Syntax:

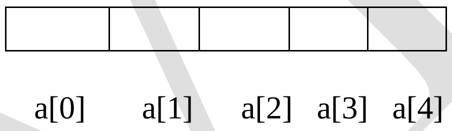
```
datatype  array_name[size];
```

- **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.**



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.**

Storing Values in Arrays

- Declaration of arrays only allocates memory space for array. But array elements are not initialized and hence values has to be stored.
- Therefore to store the values in array, there are 3 methods
  1. Initialization
  2. Assigning Values
  3. Input values from keyboard through **scanf()**

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]

- ➢ The various ways of initializing arrays are as follows:
  1. **Initializing all elements of array(Complete array initialization)**
  2. **Partial array initialization**
  3. **Initialization without size**
  4. **String initialization**

1. Initializing all elements of array:

- ➢ Arrays can be initialized at the time of declaration when their initial values are known in advance.
- ➢ In this type of array initialization, initialize all the elements of specified memory size.
- ➢ Example:
  int a[5]={10,20,30,40,50};

10	**20**	**30**	**40**	**50**

2. Partial array initialization

- ➢ If the number of values to be initialized is less than the size of array then it is called as partial array initialization**.**
- ➢ In such a case elements are initialized in the order from 0th element.
- ➢ The remaining elements will be initialized to **zero automatically by the compiler.**
- ➢ Example:
  int a[5]={10,20};

10	**20**	**0**	**0**	**0**

3. **Initialization without size**

- In the declaration the array size will be set to the total number of initial values specified.
- The compiler will set the size based on the number of initial values.
- Example:

  int a[ ]={10,20,30,40,50};
- **In the above example the size of an array is set to 5**

4. **String Initialization**

- Sequence of characters enclosed within double quotes is called as string.
- The string always ends with NULL character**(\0)**

```
char s[5]=”SVIT”;
```

We can observe that string length is 4,but size is 5 because to store NULL character we need one more location.

So pictorial representation of an array **s** is as follows:

S	V	I	T	\0
S[0]	S[1]	S[2]	S[3]	S[4]

<u>Two Dimensional arrays:</u>

- In two dimensional arrays, elements will be arranged in rows and columns.
- To identify two dimensional arrays we will use two indices(say i and j) where I index indicates row number and j index indicates column number.

<u>Declaration of two dimensional array:</u>

data_type array_name[exp1][exp2];

Or

data_type array_name[row_size][column_size];

- **data_type** can be int,float,char,double.
- **array_name** is the name of the array.
- **exp1 and exp2** indicates number of rows and columns

For example:

int a[2][3];

- ✓ The above statements allocates memory for 3*4=12 elements i.e 12*2=24 bytes.

Initialization of two dimensional array

Assigning or providing the required values to a variable before processing is called initialization.

```
Data_type array_name[exp1][exp2]={
                                        {a1,a2,….an},
                                        {b1,b2,   .bn},
                                        …………….
                                        ……………..
                                        {z1,z2,…zn}
                                   };
```

- Data type can be int,float etc.
- exp1 and exp2 are enclosed within square brackets .
- both exp1 and exp2 can be integer constants or constant integer expressions(number of rows and number of columns).
- a1 to an are the values assigned to 1st row ,
- b1 to bn are the values assigned to 2nd row and so on.

Example:

```
int a[3][3]={
               {10,20,30},
               {40,50,60},
               {70,80,90}
           };
```

10	**20**	**30**
40	**50**	**60**
70	**80**	**90**

Partial Array Initialization

- If the number of values to be initialized is less than the size of array, then the elements are initialized from left to right one after the other.
- The remaining locations initialized to zero automatically.
- Example:

```
int a[3][3]={
               {10,20},
               {40,50},
               {70,80}
```

};

10 ➢	**20**	**0**
40	**50**	**0**
70	**80**	**0**

5b. write a C program to search a name in a list of names using Binary searching technique. [08M]

* C program to search a name in a list of names using Binary searching technique*/

```
#include<stdio.h>
#include<string.h>
void main()
{
int i, n, low, high, mid;
char names[20][10] ,key;
printf("enter the number of names\n"):
scanf("%d",&n);
printf("enter the names\n");
for(i=0;i<n;i++)
{
        Scanf("%s",names[i]);
}
printf("enter the key name to be searched\n");
scanf("%s",key);
low=0;
high=n-1;
while(low<=high)
{
        mid=(low+high)/2;
        res=strcmp(key,name);
        if(res==0)
        {
                printf("successful search\n");
                exit(0);
        }
        if(res>0)
        {
                low=mid+1;
        }
        else
        {
```

```
                        high=mid-1;
                }
        }
        printf("unsuccesfull seasrch\n");

        }
```

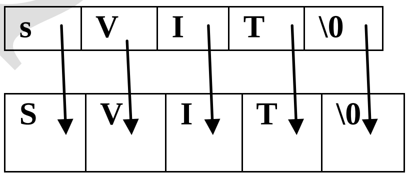
6a.Explain any five string handling functions with example. [8M]

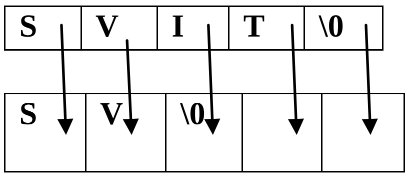
Answer: Write any five in below list with example

String handling Functions

S L . N o	Name	Syntax	Example	Explanation

1	**strlen**	**int strlen (char str[ ]);**	**char str[15]="SVIT";**
			int count;
			count=strlen(str);
			S V I T \0
			0 1 2 3 4
			The example str variable 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.

2	**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.**

3.	**strncpy**	**strcpy(char dest[ ] , char src[ ],int n);**	**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.**
4	**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 copies the all characters of s2 string to the end of s1 string.** ✓ **The delimiter of s1 is replaced by first character of s2.** ✓ **Size of s1 string should be greater or store the contents of both the string**
5	**strncat**	**strncat(char s1[ ] , char s2[ ],n);**	**char s1[5]=”SVIT”; char s2[5]=”ECE”; strncat(s1,s2,2);** **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 \0** **0 1 2 3 4 4 6 7 8 9** **In the above example only 2 characters(EC) from string s2 copies to**	✓ **This function copies the n characters of s2 string to the end of s1 string.** ✓ **The delimiter of s1 is replaced by first character of s2.**

			string s1.	
6	**strcmp**	**int strcmp(char s1[ ] , char s2[ ]);**	**1) Strings are equal** **S1[4]=”RAM”;** **S2[4]=”RAM”;** **Strcmp(S1,S2);** **S1[0] | R | == | S[0] | R** **S1[1] | A | == | S2[1] | A** **S1[2] | M | == | S3[2] | M** **S1[3] | \0 | == | S4[3] | \0** **S1[0]==S2[0]** **R==R(ASCII value of R is compared) similarly for other characters.** **S1[3]==S2[3]** **\0==\0(ASCII value of \0 is compared and it is 0.)** **2)String S1 is Lesser than String S2** **S1[4]=”ABC”;** **S2[4]=”BAC”;** **Strcmp(S1,S2);** **S1[0] | A | == | S2[0] | B** **S1[1] | B | == | S2[1] | A** **S1[2] | C | == | S3[2] | C** **S1[3] | \0 | == | S4[3] | \0** **S1[0]==S2[0]** **A==B(ASCII value of A is compared with ASCII value of B)** **i.e 65==66 returns S1<S2** **3)String S1 is Greater than String S2** **S1[4]=”BBC”;** **S2[4]=”ABC”;** **Strcmp(S1,S2);**	**where:** **s1 is first string** **s2 is second string** ✓ **This function used to compare two strings.** ✓ **The comparison starts with first character of each string.** ✓ **This comparison continues till the corresponding character differ or until the end of the character is reached.** ✓ **The strcmp Returns 3values** **Possibly:** **returns 0 if both strings are equal.** **returns positive value ,if s1>s2** **returns negative value if s1<s2**

			S1[0] B == S2[0] A **S1[1] B == S2[1] B** **S1[2] C == S3[2] C** **S1[3] \0 == S4[3] \0** **S1[0]==S2[0]** **A==B(ASCII value of A is compared with ASCII value of B)** **i.e 66==65 returns S1>S2**	
7	**strnc mp**	**int strcmp(char s1[ ] , char s2[ ], n);**	**1) Strings are equal** **S1[4]=”RAM”;** **S2[4]=”RAM”;** **strcmp(S1,S2,2);** **S1[0] R == S2[0] R** **S1[1] A == S2[1] A** **S1[2] M == S3[2] M** **S1[3] \0 == S4[3] \0** **Only 2 characters from each S1 and S2 is compared.** **Other function is similar to strcmp().**	**where:** **s1 is first string** **s2 is second string** ✓ **This function used to compare n number of characters two strings.** ✓ **The comparison starts with first character of each string.** ✓ **This comparison continues till the corresponding character differ or until the end of the character is reached or specified number of characters have been tested..** ✓ **The strcmp Returns 3values** **Possibly:** **returns 0 if both strings are equal.** **returns positive value ,if s1>s2**

				returns negative value if s1<s2
8	**strrev ( )**	**void strrev(char str[ ]);**	**Given string** **S1 S V I T E C E \0** **0 1 2 3 4 5 6 7 8 9** **strrev(s1)** **Reverse String** **S1 E C E T I V S \0** **0 1 2 3 4 5 6 7 8 9** **S1[6]==S1[0]** **S1[5]==S1[1]** **S1[4]==S1[2]** **S1[3]==S1[3]** **S1[2]==S1[4]** **S1[1]==S1[5]** **S1[0]==S1[6]**	✓ **This function reverse all characters in the S1 except Null character.** ✓ **The original string is lost.**

Example Programs:

strlen()	**strrev()**

#include<stdio.h>
#include<string.h>
void main()
{ char name[15];
int len;
printf("Enter the string\n");
gets(name);
len=strlen(name);
printf("\n The string length is %d",len);
}
 	
OUTPUT
Enter the string
COMPUTER
The string length is 8
	#include<stdio.h>
	#include<string.h>
	void main()
	{
	char str[]="INDIA";
	strrev(str);
	printf("string=%s",str);
	}
	
	OUTPUT

strcpy()	**strncpy()**

```
#include<stdio.h>
#include<string.h>
void main()
{
      char src[15],char dest[15];
     printf("Enter the source string\n");
     gets(src);
      strcpy(dest,src);
      printf("\n The copied string is \n");
     puts(dest);
}
```

OUTPUT

```
Enter the source string
COMPUTER
The copied string is
 COMPUTER
```

```
#include<stdio.h>
#include<string.h>
void main()
{
      char src[15],char dest[15];
     int n;
     printf("Enter the source string\n");
     gets(src);
     printf("Enter n");
     scanf("%d",&n);
      strncpy(dest,src);
      printf("\n The copied string is \n");
     puts(dest);
}
```

OUTPUT

```
Enter the source string
COMPUTER
Enter n
3
The copied string is
 COM
```

strcat()

```
#include<stdio.h>
#include<string.h>
void main()
{
      char S1[15],char S2[15];
     printf("Enter the string 1\n");
     gets(S1);
     printf("Enter the string 2\n");
     gets(S2);
      strcat(S1,S2);
      printf("\n The Concatenated string is \n");
     puts(S1);
```

strncat()

```
#include<stdio.h>
#include<string.h>
void main()
{
      char S1[15],char S2[15];
     int n;
     printf("Enter the string 1\n");
     gets(S1);
     printf("Enter the string 2\n");
     gets(S2);
     printf("Enter n");
     scanf("%d",&n);
      strncat(S1,S2,n);
```

```
        }
```

OUTPUT

```
Enter the string1
HELLO
Enter the string 2
ALL
The Concatenated string is
 HELLOALL
```

```
            printf("\n The Concatenated string is \n");
            puts(S1);
        }
```

OUTPUT

```
Enter the string1
HELLO
Enter the string 2
SVIT
Enter n
2
The Concatenated string is
 HELLOSV
```

strcmp()

```
        #include<stdio.h>
        #include<string.h>
        void main()
        {
           char S1[15],char S2[15];
          int res;
          printf("Enter the string 1\n");
          gets(S1);
          printf("Enter the string 2\n");
          gets(S2);
           res=strcmp(S1,S2);
           if(res==0)
              printf("Strings are same\n");
           else if(res>0)
               printf("String1 is greater than string2\n");
           else
              printf("String1 is lesser than string2\n");
        }
```

OUTPUT

```
Enter the string1
```

strcnmp()

```
        #include<stdio.h>
        #include<string.h>
        void main()
        {
           char S1[15],char S2[15];
          int res,n;
          printf("Enter the string 1\n");
          gets(S1);
          printf("Enter the string 2\n");
          gets(S2);
          printf("Enter n");
          scanf("%d",&n);
           res=strcmp(S1,S2,n);
           if(res==0)
              printf("Strings are same\n");
           else if(res>0)
               printf("String1 is greater than string2\n");
           else
              printf("String1 is lesser than string2\n");
        }
```

OUTPUT

```
Enter the string1
SVIT
```

HELLO Enter the string 2 HELLO The Strings are equal	Enter the string 2 SVCE Enter n 2 String1 is greater than String2

6bWrite a c program to read N elements and find biggest element in the array. [8M]

Answer:

```
/* C program to find the biggest element*/
#include<stdio.h>
void main ()
{
        int i,big,a[30];
        printf("enter the number of elements\n");
        scanf("%d",&n);
        printf("enter the elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        big=a[0];
        for(i=1;i<n;i++)
        {
                if(a[i]>big)
                {
                        big=a[i];
                }
        }
        printf("The biggest element is %d",big);
}
```

MODULE 4

7a.What is structure ?Explain the syntax of structure declaration and initialization with example. [5M]

Answer:

Structures

- **Definition:** A *structure* is defined as a collection of variables of same data type or dissimilar datatype grouped together under a single name.

Syntax	Example

**struct tagname
{
datatype member1;
datatype member2;
datatype member3;
}**
	struct student
	{
	char name[10];
	int usn;
	float marks;
	};

where

struct is a keyword which informs the compiler that a structure is being defined

tagname: name of the structure

member1,member2: members of structure:

type1,type 2 : int,float,char,double

<u>**Structure Declaration**</u>

As variables are declared ,structure are also declared before they are used:

Three ways of declaring structure are:

- Tagged structure
- Structure without tag
- Type defined structures

1.Tagged structure	

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

Declaration of structure variables	
struct tagname v1,v2,v3…vn;	**struct student s1,s2,s3;**

2.structure without tagname	

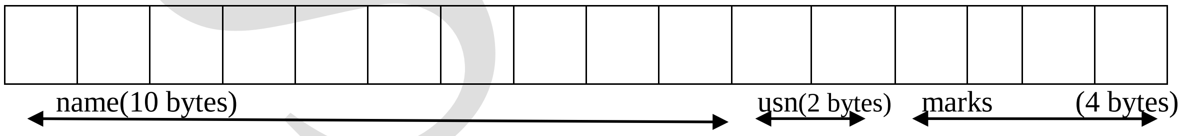
syntax
struct
{
  data type member1;
  data type member2;
;
  -------------------- 	
  -------------------- 	
}v1,v2,v3;
	Example
	struct
	{
	  char name[20];
	  int usn;
	  float marks;
	}s1,s2;

3.Type defined structure	

syntax
typedef struct
{
  data type member1;
  data type 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;**

Memory Allocation for structure variable s1:



memory allocated for a structure variable s1=memory allotted for name+usn+marks

10+2+4

16 bytes.

Structure initialization

Syntax:

struct tagname variable={v1,v2….vn};

example

struct student s1={"sony",123,24};

7b.Write a C program to maintain a record of 'n' employee detail using an array of structures with three fields (id,name,salary) and print the details of employees whose salary is above 5000. [7M]

Answer:

```
 /*C program to print details of employee whose salary is above 5000*/
 #include<stdio.h>
typedef  struct
{
        char name[20];
        int id;
        float salary;
 }EMPLOYEE;

 void main()
 {
         int i,n;
        EMPLOYEE e[10];
        printf("enter the n value\n");
        scanf("%d",&n);
        printf("enter the student details\n");
        for(i=0;i<n;i++)
        {
         printf("enter the employee name:");
                scanf("%s",e[i].name);
                printf("enter the id\n");
                scanf("%d",&e[i].id);
                printf("enter the salary:");
                scanf("%f",&e[i].salary);
        }
        if(e[i].salary>5000)
        {
         printf("the employee is %s with %f salary\n"e[i].name,e[i].salary);
         exit(0);
        }
        printf("no records found whose salry is greater than 5000\n");
}
```

7c.Explain fprintf and fscanf withj syntax. [4M]

Answer

fscanf():

The function **fscanf** is used to get data from the file and store it in memory.

Syntax:

fscanf(fp, "format string", address 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 variable s specified in the list ,will take the values from the specified pointer fp by using the specification provided in format sting.

"**address list":**address list of variables

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

Example:

FILE *fp
fp=fopen("name.txt","r");
fscanf("fp,"%s",name);
	FILE *fp
	fp=fopen("marks.txt","r");
	fscanf("fp,"%d%d%d",&m1,&m2,&m3);

Note:

1.If the data is read from the keyboard then use **stdin** in place of **fp**

2.If the data is read from the file then use **fp**

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 to be print.

“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.

Example:

FILE *fp
fp=fopen(“name.txt”,”w”);
fscanf(“fp,”%s”,name);
	FILE *fp
	fp=fopen(“marks.txt”,”w”);
	fscanf(“fp,”%d%d%d”,m1,m2,m3);

Note:

1.If the data has to be printer on output screen then use **stdout** in place of **fp**

2.If the data has to be written to the file then use **fp**

Example Program

Write a C program to read the contents of two files called as name.txt and usn.text and merge the contents to another file called as output.txt and display the contents on console using fscanf() and fprintf()

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        FILE *fp1,*fp2,*fp3;
        char name[20];
        int usn;
        fp1=fopen("name.txt","r");
        fp2=fopen("usn.txt","r");
        fp3=fopen("output.txt","w");
        for(;;)
        {
           if(fscanf(fp1,"%s",name)>0)
           {
                 if(fscanf(fp2,"%d",&usn)>0)
                 {
                    fprintf(fp3,"%s %d\n",name,usn);
                 }
                 else break;
           }
           else break;
        }
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        fp3=fopen("output.txt","r");
        printf("NAME\tUSN\n");
        while(fscanf(fp3,"%s %d\n",name, &usn)>0)
        {
                printf("%s \t%d\n",name,usn);
        }
        fclose(fp3);
}
```

8a. Explain structure within a structure with an example. [7M]

Answer:

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;
<u>struct subject PCD;</u>
float avg;
} 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;
S1.avg;	

8b.What is a file? Explain fclose and fopen function. [5M]

- **Definition: A** file is defined as collection of data stored on the secondary device such as hard disk.

fopen()

- The file should be opened before reading a file or before writing into a file.
- The syntax to open a file is:
  FILE *fp;

```
fp=fopen(filename,mode);
```

- fp is a file pointer.
- fopen is a function to open a file.
- Mode can be r,w, or a
- fopen function will return the starting address of opened file and it is stored in file pointer
- If file is not opened then fopen function returns NULL.

```
        if(fp==null)
        {
                printf("error in opening file\n");
                exit(0);
        }
```

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.

fclose()

- Closing a file: fclose function()
- When we no longer need a file ,we should close the file .this is the last option to be performed on a file.
- A file can be closed using fclose() function.

- If a file is closed successfully,0 is returned, otherwise EOF is returned.

Syntax:

```
fclose(fp);
```

8c.Explain fgets and fputs function. **[4M]**

Answer:

<u>fgets()</u>

<u>fgets()</u> 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

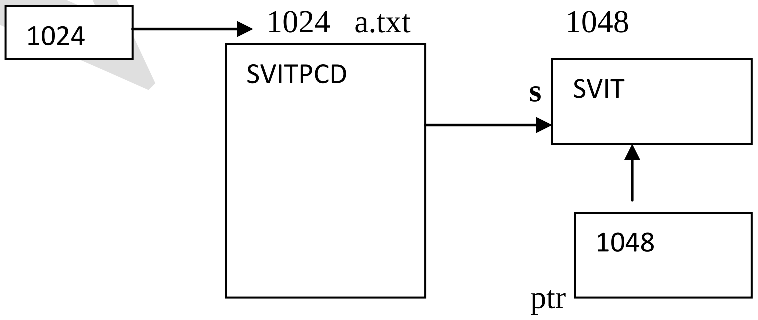
n ->number of characters to be read from file

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.

Example:



```
FILE *fp;

char s[10];

char *ptr;

fp=fopen("a.txt","r");


if(fp==NULL)

{
printf("file cnnnot be opened);
exit(0);
}
ptr=fgets(s,4,fp);
fclose(fp);
```

fputs()

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

Syntax:

fputs(str,fp);

where

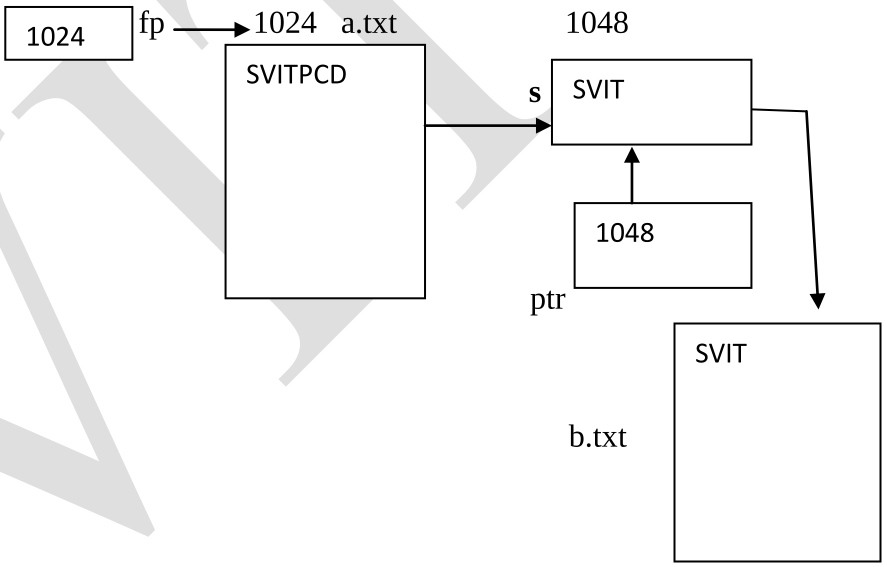
fp ->file pointer which points to the file to be read

str ->string variable where read string will be stored

Example:

```
FILE *fp,*fp1;
char s[10];
char *ptr;
fp=fopen("a.txt","r");
fp1=fopen("b.txt","w");
if(fp==NULL)

printf("file cnnnot be opened);
exit(0);
}
ptr=fgets(s,4,fp);
fputs(s,fp1);
fclose(fp);
fclose(fp1);
```



MODULE 5

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

Answer:

Definition:

- ***A pointer is a variable which contains the address of another variable.*** A pointer is a derived data type. This is the one which stores the address of data in memory.

Declaring of a pointer variable

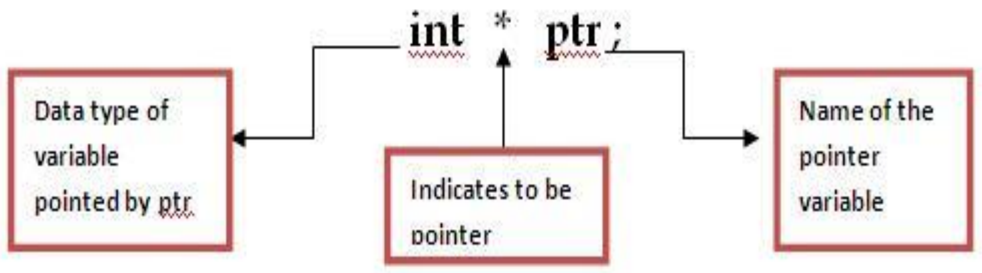
General syntax :

```
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_type which 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 the integer variable
- i.e. it is a **pointer to an integer variable'** ,but the declaration 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;

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:

```
int   x;

int   *p

p = &x;
```

9b.Explain any two preprocessor directive in c with example? [6M]

1. **<u>Symbolic Constant:</u>**

- 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);
```

```
    }
```

2. <u>defining macros:</u>

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)

consider the programming example

```
#include<stdio.h>
#define square(x)   ((x)*(x))
void main()
{
        int m=5;
        printf(" the square of m is %d",square(m));
}
```

9c.write a c program to swap two numbers using call by pointers (address) method? [6M]

Answer

```
    #include<stdio.h>
    void swap(int m,int n);
    void main()
    {
        int a,b;
```

```
            printf("enter values for a and b:");
            scanf("%d %d",&a,&b);
            printf("the values before swapping are a=%d b=%d \n",a,b);
            swap(&a,&b);
            printf("the values after swapping are a=%d b=%d \n",a,b);
 }
 void swap(int *m, int *n)
{
            int temp;
            temp=*m;
            *m=*n;
            *n=temp;
 }
```

10a.What is dynamic memory allocation ?write and explain different dynamic memory allocation functions in C? [6M]

Answer:

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.
- Additional storage can be allocated whenever needed.
- We can de-allocate the dynamic space whenever we done with them.
- To implementing dynamic memory the following 4 functions are used.

1. **malloc( ):** Allocates requested number of bytes and returns a pointer to the first byte of the allocated space.
2. **calloc( ):** Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.
3. **realloc( ):** Modifies the size of previously allocated space.
4. **free( ):** Frees previously allocated space.

1. **malloc( ):**
- Allocates requested number of bytes and returns a pointer to the 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. **<u>calloc( ):</u>**

- ➢ 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));

<u>3.realloc( )</u>

- ➢ 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:

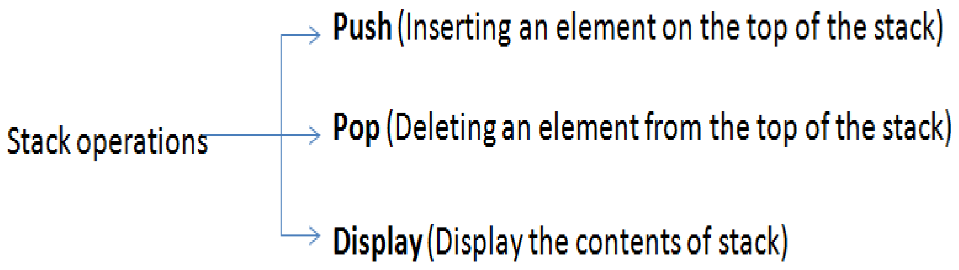
```
free(ptr);
```

10b.Explain stacks and queue data structures along with their applications. [4M]

Answer

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 .
- 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)

- **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.

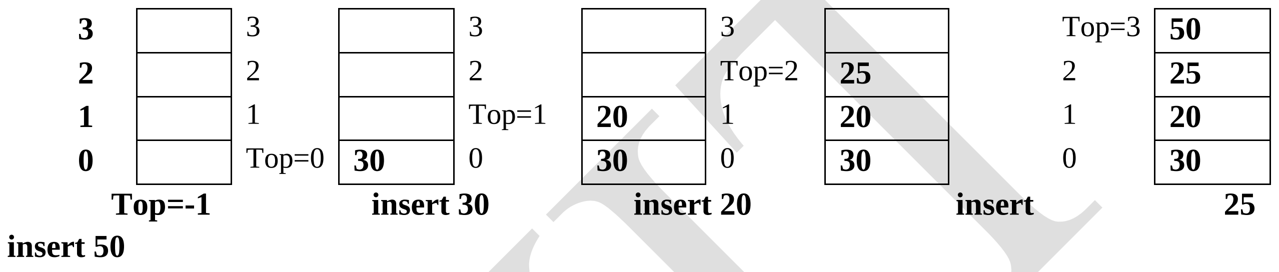


Figure stack push operation

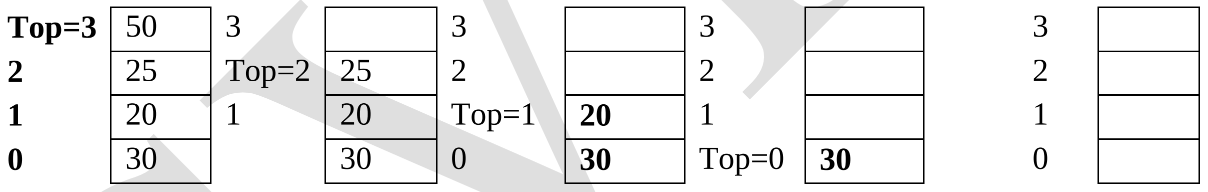


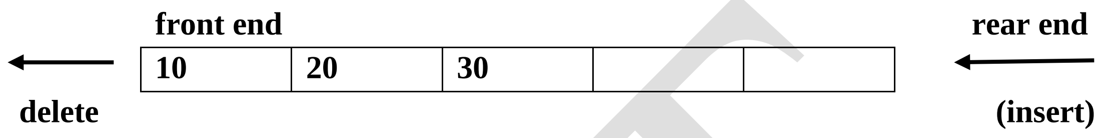
Figure stack pop operation

- **Application of Stacks**
  - ➢ There are two applications of stacks.
  - ➢ Recursion: A recursion function is a function which calls itself. The problems like towers of Hanoi, tree manipulation problems etc can be solved using recursion.
  - ➢ Arithmetic/Evaluation of Expression: The conversion of an expression in the form of either postfix or prefix can be easily evaluated.
  - ➢ Conversions of expressions: Evaluation of infix expressions will be very difficult and hence it needs to be converted to prefix or postfix expression which needs the use of stack.

Queue

- ➢ 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**

2. **Circular Queue**

- In circular queue the elements of queue can be stored efficiently in an array so as to wrap around so that the end of the queue is followed by front of queue.
- 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**

3. **Double Ended Queue**

- A Double Ended Queue is in short called as Deque (pronounced as Deck or dequeue). A deque is a linear queue in which insertion and deletion can take place at either ends but not in the middle. The operations that can be performed are
    - **a) Insert an item from the front end**
    - **b) Insert an item from rear end.**
    - **c) Delete element from front end**
    - **d) Delete an element from rear end**
    - **e) Display the elements**

4. **Priority Queue**

- A priority queue is a collection of elements such that each element has been assigned a priority value such that the order in which elements are deleted and processed based on the assigned priority.
- There are two different types of priority queue

1. Ascending priority queue: Elements can be inserted in any order but deletion is done in the ascending order of values.
2. Descending priority queue: Elements can be inserted in any order but deletion is done in the descending order of values.

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.

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.

10C Explain how pointers and arrays are related with example. [4M]

<u>Pointer and arrays</u>

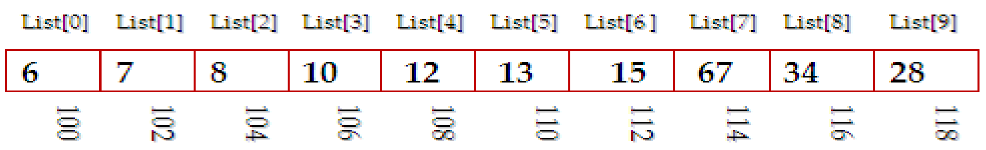
- When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.
- The base address is the location of the first element (index 0) of the array.
- The compiler also defines the array name as a constant pointer to the first element.

Consider the following declaration:

int list[10];

This defines an array of 10 memory locations which can store 10 integer numbers ,which can be referred as list[0] ……………list[9].

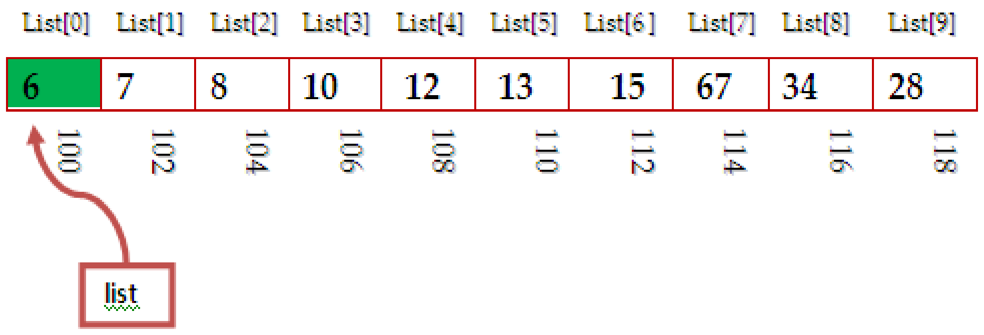
List array can be represented as :



The notation a[i] refers to the ith element of the array.

The 'list' contains the address of the first element of the array . Since a pointer is a variable which contains the address of another variable., it is evident that list is also a pointer variable because it contain the address of the first element of the array.

List is pointing to the first element of the array.



Reference to list[i] can also be referred as *(list + i) where list if the name of the array. Both the two forms are equivalent .

Also &list[i] and list+i are identical, as list +i is the address of the (i+1)th element And &list[i] is also refers to the address of the (i+1)th element of the array.

```
/*Program to read and display array elements using pointers*/
#include<stdio.h>
 void main()
{
        int a[100], i, n;
        printf("enter size of array");
        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));
}
```