NRI INSTITUTE OF INFORMATION SCIENCE & TECHNOLOGY BHOPAL

DEPARTMENT OF MASTER OF COMPUTER APPLICATION

LAB MANUAL

C AND DS LAB (MCA-106)

MASTER OF COMPUTER APPLICATION (MCA)

EXPERIMENT #1

Objective(s):

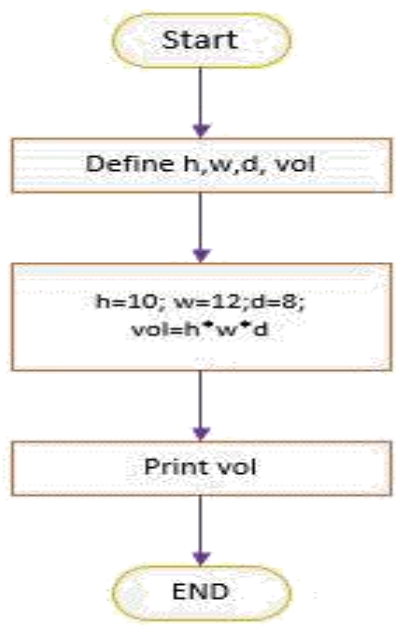
To be familiar with syntax and structure of C- programming.
To learn problem solving techniques using C

Program: Write a Program to calculate and display the volume of a CUBE having its height (h=10cm), width (w=12cm) and depth (8cm).

Algorithm:

1. Start
2. Define variables: h(int), w(int), d(int), vol(int)
3. Assign value to variables: h = 10, w=12, d=8
4. Calculate the volume as: vol = h*w*d
5. Display the volume (vol)
6. Stop

Flowchart:



Code: *(Use comments wherever applicable)*

```
//Following code is written and compiled in GCC

#include<stdio.h>
void main()
{
//start the program
int h,w,d,vol; //variables declaration
h=10;w=12;d=8; //assign value to variables
vol=h*w*d;     //calculation using mathematical formula
printf("The Volume of the cube is: %d",vol); //display the
volume
getch();
//end the main program
}
```

Output :

The Volume of the cube is: 960

EXPERIMENT #2

Objective(s):

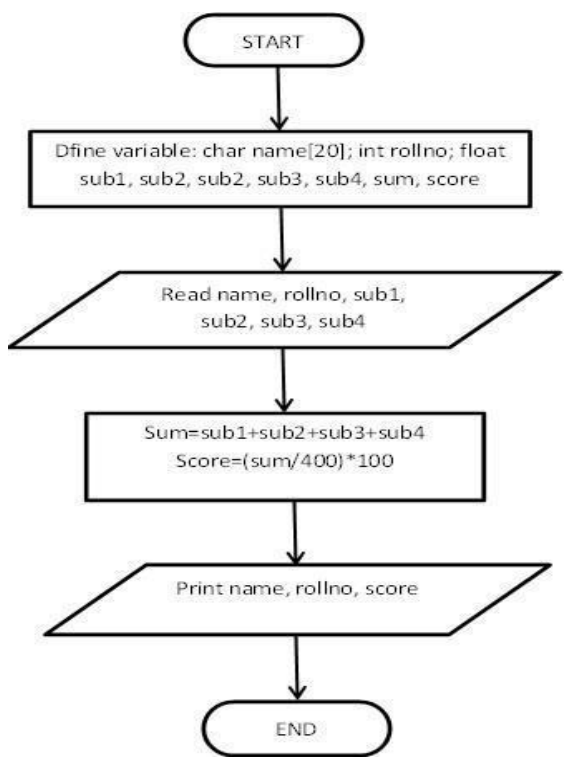
To be familiar with different data types, Operators and Expressions in C.

Program: Write a program to take input of name, rollno and marks obtained by a student in 4 subjects of 100 marks each and display the name, rollno with percentage score secured.

Algorithm:

1. Start
2. Define variables: name, rollno, sub1, sub2, sub3, sub4, sum, score
3. Take input from keyboard for all the input variables
4. Calculate the sum of marks of 4 subjects and also calculate the percentage score as:
   sum = sub1 + sub2 + sub3 + sub4;
   score = (sum/400) * 100
5. Display the name, roll number and percentage score.
6. Stop

Flowchart:



Code: *(Use comments wherever applicable)*

```
//Following code is written and compiled in TURBO C++

#include<stdio.h>
#include<conio.h>
void main()
{
char name[20];
int rollno;
float sub1, sub2, sub3, sub4, , sum, score;
printf("Enter name of student: ");
scanf("%s",&name[]);
printf ("\n Enter Roll Number: ");
scanf("%d", &rollno);
printf ("\n Enter Marks in 4 Subjects:\n");
scanf("%f%f%f%f", &sub1, &sub2, &sub3, &sub4);
sum=sub1+sub2+sub3+sub4;
score = (sum/500)*100;
printf("\n Name of student: %s", name[]);
printf("\n Roll Number: %d", rollno);
printf ("\nPercentage score secured: %2.2f%c", score,'%');
getch();
}
```

Output:

Enter name of student: Ajit Singh
Roll Number: 25
Enter Marks in 4 Subjects:
50
75
85
62
Name of student: Ajit Singh
Roll Number: 25
Percentage score secured: 68.00%

EXPERIMENT #3

Objective(s):

To understand the programming knowledge using Decision Statements (if, if-else, if-else-if ladder, switch and GOTO)

Program: Write a program to print whether a given number is even or odd.

Code: *(Use comments wherever applicable)*

```
//Following code is written and compiled in TURBO C++

#include<stdio.h>
#include<conio.h>
void main()
{
int num;
printf("Enter the number: ");
scanf("%d",&num);
if(num%2==0)
     printf("\n %d is even", num);
else
     printf("\n %d is odd", num);
getch();
}
```

Output:

Enter the number: 6
6 is even

EXPERIMENT #4

Objective(s):

To understand the programming using Loop & nested loop Statements (for, while, do-while)

Program: Write a program to print positive integers from 1 to 10.

Code:

```
//Following code is written and compiled in TURBO C++
//Using FOR LOOP

#include<stdio.h>
#include<conio.h>
void main()
{
int i;
for(i=1; i<=10;i++)
     printf("%d \n", i);
getch();
}


//Using WHILE LOOP

#include<stdio.h>
#include<conio.h>
void main()
{
int i=1;
while(i<=10)
     {
     printf("%d \n", i);
     }
     i++;
getch();
}
```

```
//Using DO-WHILE LOOP

#include<stdio.h>
#include<conio.h>
void main()
{
int i=1;
do
     {
     printf("%d \n", i);
     i++;
     }
while(i<=10);
getch();
}
```

Output:

1
2
3
4
5
6
7
8
9
10

EXPERIMENT #5

Objective(s):

To understand programming using different dimensions of Array.

Program: Write a program to insert 5 elements into an array and print the elements of the array.

Code: *(Use comments wherever applicable)*

```
#include<stdio.h>
#include<conio.h>  void
main()
{
int i, arr[5];
printf("Enter the elements into the array:");
for(i=0; i<=4;i++)
     scanf("%d",&arr[i]);
printf("The elements of the array are:");
for(i=0; i<=4;i++)
     printf("%d \t", arr[i]);
getch();
}
```

Output:

```
1
2
3
4
5
```

EXPERIMENT #6

Objective: To learn the concepts ofArray and its operations.

Problem statement: Implement Array operations – Insertion, Deletion, and Display.

Underlying concepts: Array.

An array is a collection of homogeneous elements i.e., all the elements are of an array are of the same type.

To define an array, you tell the compiler to set aside storage for a given number of data items of a specified type. You also tell the compiler the name of the array. Here's an example of an array definition that creates storage for four integers:

```
int age[4];
```

The int specifies the type of data to be stored, age is the name of the array, and 4 is the size of the array; that is, the maximum number of variables of type int that it will hold. Brackets []surround the size.

Each variable stored in an array is called an element. The elements are numbered. These numbers are called index numbers or indexes or subscripts. The index of the first array element is 0, the index of the second is 1, and so on. If the size of the array is n, the last element has the index n-1. For example, in the age array, which has a size of 4, the elements are numbered 0, 1, 2, and 3. This numbering can be the source of some confusion. Keep in mind that the last element in an array has an index one less than the size of the array.

You refer to individual array elements using their index numbers and the array name. The first element is called age [0], the second is age [1], and so on. You can make statements such as

```
age[2] = 23;
```

Insertion: To insert a data item in an array, we specify the location of the data item to be inserted.

Suppose we have an array whose maximum size is 10 and the current elements are:

12	23	14	25	14	45	35	23		
0	1	2	3	4	5	6	7	8	9

And it is required to insert '16' at index '3'. Since there is no space available at index three we need to move all the elements from index '3' to '7' upward '4' to '8' respectively.

12	23	14		25	14	45	35	23	
0	1	2	3	4	5	6	7	8	9

Now the element can be inserted at index '3'.

Deletion: To delete a data item from an array, we specify the location of the data item to be deleted.

Suppose we have an array whose maximum size is 10 and the current elements are:

12	23	14	25	14	45	35	23		
0	1	2	3	4	5	6	7	8	9

And it is required to delete element at index ‘3’. Since after deletion there will be a space available at index ‘3’ we need to move all the elements from index ‘4’ to ‘7’ downward ‘4’ to ‘8’ respectively.

12	23	25	14	45	35	23			
0	1	2	3	4	5	6	7	8	9

Display:

Arrays are useful primarily because the individual variables stored in an array can be accessed using an index number. This makes it easy to cycle through the array, accessing one variable after another. We can use a for loop to do this.

Code:

```
#include<iostream.h>
#include<conio.h> const
int max=10; void main()
{
	int choice, arr[max],i,cur=0, x, pos; do
	{
		cout<<"1 - INSERTION\n2 - DELETION\n3 - DISPLAY\n4 - EXIT\n";
		cin>>choice;
		switch(choice)
		{
			case 1:
				if(cur==max) // Check whether array is full.
					cout<<"Array is full\n";
				else
				{
					cout<<"Enter the element: ";
					cin>>x;
					cout<<"Enter the index ("<<0<<"-"<<cur<<") : ";
					cin>>pos;
					// move elements upward.
					for(i=cur-1;i>=pos;i--)
						arr[i+1]=arr[i];
					arr[pos]=x;//insert the elemtn. cur++;
				}
				break;
			case 2:
				if(cur==0) // Check whether array is empty.
					cout<<"Array is empty\n";
				else
				{
					cout<<"Enter the index: ";
					cin>>pos;
					x=arr[pos];
					// move elements upward.
					for(i=pos;i<cur;i++)
						arr[i-1]=arr[i];
```

```
                    cur--;
                }
                break;
            case 3:
                for(i=0;i<cur;i++)
                    cout<<arr[i]<<" ";
                cout<<endl;
                break;
            default:
                break;
        }
    }while(choice!=4);
}
```

Analysis:

Conclusion:

EXPERIMENT #7

Objective: To learn the concepts of linear search. **Problem**

statement: Implement linear search.

Underlying concepts: Linear search.

Linear search is an algorithm for locating the position of an element in an array. It inspects the elements of the list starting from first element of the array: if equal to the sought value, then the position has been found; otherwise, the next element is inspected for further searching and so on. It finds the sought value if it exists in the list or if not determines "not present", in linear time. The process continues till the end of the array is reached.

Code:

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

int linsearch(int*, int, int); void

main()
{
        int a[10], x, loc;
        cout<<"Enter 10 elemets of the array:\n"; for(int
        i=0; i<10;i++)
                cin>>a[i];
        cout<<"Enter the element to be searched : "; cin>>x;
        loc=linsearch(a, 10, x);
        if(loc==-1)
                cout<<"Element not in the list\n";
        else
                cout<<"Element found at "<<loc<<endl;
}

int linsearch(int*a, int n, int x)
{
        int i=0;


        while( i<n &&(a[i]!=x))
                i++;
        if(i<n)
                return i;
        else
                return -1;
}
```

Analysis:

Conclusion

EXPERIMENT #8

Objective: To learn the concepts of binarysearch.

Problem statement: Implement binary search.

Underlying concepts: Binary search.

Binary search is an algorithm for locating the position of an element in a sorted list. It inspects the middle element of the sorted list: if equal to the sought value, then the position has been found; otherwise, the upper half or lower half is chosen for further searching based on whether the sought value is greater than or less than the middle element. The method reduces the number of elements needed to be checked by a factor of two each time, and finds the sought value if it exists in the list or if not determines "not present", in logarithmic time.

Code:

```
#include<iostream.h>
#include<conio.h>
int binsearch(int*, int, int, int); void main()
{
        int a[10], x, loc, low, high; cout<<"Enter 10
        elemets of the array:\n"; for(int i=0; i<10;i++)
                cin>>a[i];
        cout<<"Enter the element to be searched : "; cin>>x;
        loc=binsearch(a, 0, 9, x);
        if(loc==-1)
                cout<<"Element not in the list\n";
        else
                cout<<"Element found at "<<loc<<endl;
}

int binsearch(int*a, int low, int high, int x)
{
        int mid;

        mid=(low+high)/2;

        while(a[mid]!=x)
        {
                if(a[mid]<x)
                        low=mid+1;
                else
                        high=mid-1;
                mid=(low+high)/2;
        }

        if(a[mid]==x)
                return mid;
        else
                return -1;
}
```

Analysis:

Conclusion:

EXPERIMENT #9

Objective: To learn the concepts stack and its operations.
Problem statement: Implement Stack.
Underlying concepts: Stack.

A stack is a data structure in which all the insertions and deletions are made at one end. This end is called the TOP of the stack. The insertion operation is called PUSH and the deletion operation is called POP. Stack has LIFO property i.e., Last In First Out. The element inserted last will be the first one to be removed.

Stack can be implemented as an array, where initially the top is set -1 to indicate empty stack. When we push an element into the stack we check whether the stack is full. If it is not full, we increment the value of top and store the element at the top position.

```
Top++;
Stack [top] =x;
```

When we pop an element from the stack we check whether the stack is empty. If it is not empty, we store the element at the top position in a variable and decrement the value of top.

```
X=  Stack  [top];
Top--;
```

Code:

```
#include<iostream.h>
#include<conio.h> const
int max=10; void main()
{
    int choice, stack[max],i,top=-1, x; do
    {
        cout<<"1 - PUSH\n2 - POP\n3 - DISPLAY\n4 - EXIT\n";
        cin>>choice;
        switch(choice)
        {
            case 1:
                if(top==max-1) // Check whether array is full.
                    cout<<"Stack is full\n";
                else
                {
                    cout<<"Enter the element: ";
                    cin>>x;
                    stack[++top]=x;//insert the element.
                }
                break;
            case 2:
                if(top==-1) // Check whether array is empty.
```

```
                    cout<<"Stack is empty\n";
                else
                    x=stack[top--];
                break;
            case 3:
                for(i=0;i<=top;i++)
                    cout<<stack[i]<<" ";
                cout<<endl;
                break;
            default:
                break;
        }
    }while(choice!=4);
}
```

Analysis:

Conclusion:

EXPERIMENT #10

Objective: To learn the concepts queue and its operations.

Problem statement: Implement Queue.

Underlying concepts: Queue.

A queue is a data structure in which all the insertions are made at one end (called rear or tail) and all deletions are made at the other end(called front or head). The insertion operation is called *Enqueue* and the deletion operation is called *Dequeue*. Queue has FIFO property i.e., First In First Out. The element inserted first will be the first one to be removed.

Queue can be implemented as an array, where initially the front and rear both are set 0 to indicate empty queue.

When we insert an element into the queue we check whether the queue is full. If it is not full, we increment the value of rear and store the element at the top position.

```
rear = (rear+1) % max; Queue
[rear] =x;
```

When we pop an element from the queue we check whether the queue is empty. If it is not empty, we store the element at the top position in a variable and decrement the value of top.

```
X=          Queue          [front];
Front=(front+1) % max;
```

We maintain a counter *size* to indicate the number of elements in the queue. Initially *size*=0. When we insert an element we increment *size* and when we delete an element we decrement *size*.

Code:

```
#include<iostream.h>
#include<conio.h> const int
max=10; void main()
{
        int choice, queue[max], i, front=0,rear=0, size=0,x; do
        {
                cout<<"1 - INSERT\n2 - DELETE\n3 - DISPLAY\n4 - EXIT\n";
                cin>>choice;
                switch(choice)
                {
                        case 1:
                                if(size==max) // Check whether queue is full. cout<<"Queue is
                                        full\n";
                                else
                                {
                                        cout<<"Enter the element: "; cin>>x;
```

```
                                        rear = (rear+1) % max; queue[rear]=x;//insert the
                                        element. size++;
                                        if (size==1)//front and rear are the same. front=rear;
                                }
                                break;
                        case 2:
                                if(size == 0) // Check whether queue is empty. cout<<"Queue is
                                        empty\n";
                                else
                                {
                                x= queue [front]; front=(front+1)
                                % max; size--;
                                }
                                break;
                        case 3:
                                for(i=front;i<=front+size-1;i=(i+1)%max)
                                        cout<<queue[i]<<" ";
                                cout<<endl;
                                break;
                        default:
                                break;
                }
        }while(choice!=4);
}
```

Analysis:

Conclusion: