



Varuwan Vadivelan

Institute of Technology

Dharmapuri – 636 703

LAB MANUAL

ACADEMIC YEAR 2017-18 (ODD SEMESTER)

Regulation : 2013

Branch : B.E. – ECE

Year & Semester : II Year / III Semester

EC6312–OOPS AND DATA STRUCTURES LABORATORY



ANNA UNIVERSITY: CHENNAI**REGULATION: 2013****SYLLABUS****LIST OF EXPERIMENTS:****IMPLEMENTATION IN THE FOLLOWING TOPICS:**

1. Basic Programs for C++ Concepts
2. Array implementation of List Abstract Data Type (ADT)
3. Linked list implementation of List ADT
4. Cursor implementation of List ADT
5. Stack ADT - Array and linked list implementations
6. The next two exercises are to be done by implementing the following source files
 - i. Program source files for Stack Application 1
 - ii. Array implementation of Stack ADT
 - iii. Linked list implementation of Stack ADT
 - iv. Program source files for Stack Application 2
 - v. An appropriate header file for the Stack ADT should be included in (i) and (iv)
7. Implement any Stack Application using array implementation of Stack ADT (by implementing files (i) and (ii) given above) and then using linked list
8. Implementation of Stack ADT (by using files (i) and implementing file (iii))
9. Implement another Stack Application using array and linked list implementations of Stack ADT (by implementing files (iv) and using file (ii), and then by using files (iv) and (iii))
10. Queue ADT – Array and linked list implementations
11. Search Tree ADT - Binary Search Tree
12. Implement an interesting application as separate source files and using any of the Searchable ADT files developed earlier. Replace the ADT file alone with other appropriate ADT files. Compare the performance.
13. Quick Sort

TOTAL: 45 PERIODS

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

Standalone desktops with C++ compiler 30 Nos.

(or)

Server with C++ compiler supporting 30 terminals or more.

INDEX

EX. NO.	NAME OF THE EXERCISE	PAGE NO.	STAFF SIGNATURE	REMARKS
1	STUDENTS RECORD (CLASS AND OBJECT)			
2	UNARY OPERATORS OVERLOADING			
3	INHERITANCE			
4	FUNCTION OVERLOADING			
5	ARRAY IMPLEMENTATION OF LIST ADT			
6	LINKED LIST IMPLEMENTATION OF LIST ADT			
7	CURSOR IMPLEMENTATION OF LIST ADT			
8	ARRAY IMPLEMENTATION OF STACK ADT			
9	STACK ADT USING LINKED LIST			
10	PROGRAM SOURCE FILES FOR STACK APPLICATION1			
11	PROGRAM SOURCE FILES FOR STACK APPLICATION2			
12	QUEUE ADT USING LINKED LIST			
13	QUEUE ADT USING ARRAY			
14	BINARY SEARCH TREE			
15	HEAP SORT			
16	QUICK SORT			

EX.NO:1**STUDENTS RECORD (CLASS AND OBJECT)****AIM:**

To write a program in C++ to prepare a student Record using class and object.

DESCRIPTION:

The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming and are often called user-defined types.

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called members of the class.

C++ Class Definitions

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword `class` followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the `Box` data type using the keyword `class` as follows:

```
class Box {  
    public:  
        double length; // Length of a box  
        double breadth; // Breadth of a box  
        double height; // Height of a box  
};
```

The keyword `public` determines the access attributes of the members of the class that follow it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as private or protected which we will discuss in a sub-section.

Define C++ Objects

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class `Box`:

```
Box Box1;    // Declare Box1 of type Box  
Box Box2;    // Declare Box2 of type Box
```

Both of the objects `Box1` and `Box2` will have their own copy of data members.

Accessing the Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.). Let us try the following example to make the things clear:

ALGORITHM:

1. Create a class record.
2. Read the name, Regno, mark1, mark2, mark3 of the student.
3. Calculate the average of mark as $Avg = \frac{mark1 + mark2 + mark3}{3}$
4. Display the student record.
5. Stop the program.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class record
{
    public:
    char name[20];
    int regno;
    int marks,m1,m2,m3;
    float avg;
    void getdata()
    {
        cout<<"\nenter the name: " ;
        cin>>name;
        cout<<"enter the regno: ";
        cin>>regno;
        cout<<"enter the m1,m2,m3: \n";
        cin>>m1>>m2>>m3;
    }
    void calculate()
    {
        avg= (m1+m2+m3) /3;
    }
    void display()
    {
        cout<<"*****\n";
        cout<<"\nName: "<<name;
        cout<<"\nRegno: "<<regno;
        cout<<"\nMark1: "<<m1;
        cout<<"\nMark2: "<<m2;
        cout<<"\nMark3: "<<m3;
        cout<<"\nAvg: "<<avg;
        cout<<"*****\n";
    }
};

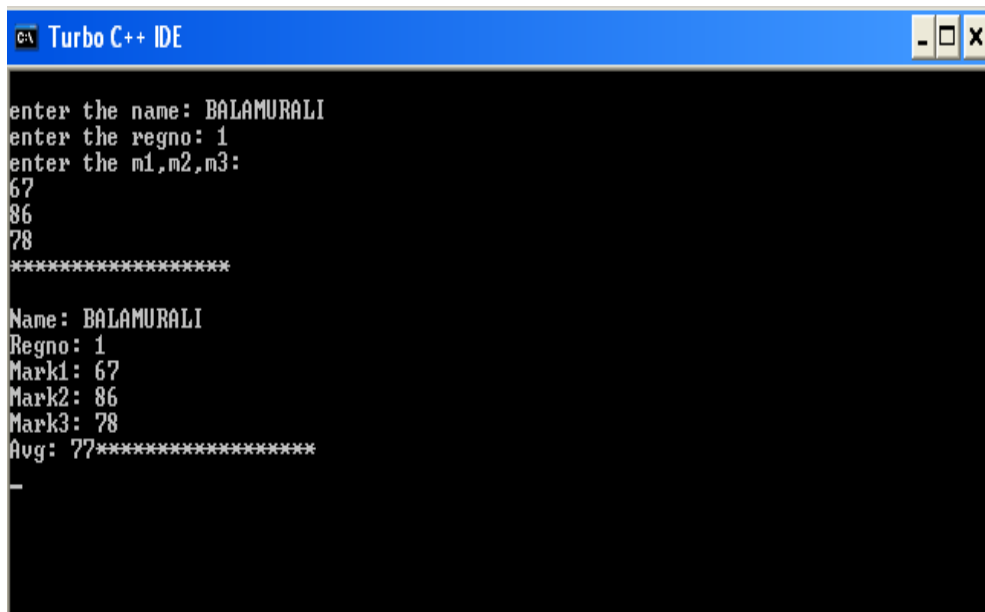
void main()
{
    record r;
```

```

        clrscr();
        r.getdata();
        r.calculate();
        r.display();
        getch();
    }

```

OUT PUT



The screenshot shows the Turbo C++ IDE window with a black background and white text. The output of the program is as follows:

```

enter the name: BALAMURALI
enter the regno: 1
enter the m1,m2,m3:
67
86
78
*****
Name: BALAMURALI
Regno: 1
Mark1: 67
Mark2: 86
Mark3: 78
Avg: 77*****
-

```

RESULT:

Thus the C++ program for implementation of classes and objects was created, executed and output was verified successfully.

EX.NO:2

UNARY OPERATORS OVERLOADING

AIM:

To write a C++ program for implementing the unary operator overloading.

DESCRIPTION:

The unary operators operate on a single operand and following are the examples of Unary operators:

- The increment (++) and decrement (--) operators.
- The unary minus (-) operator.
- The logical not (!) operator.

The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--.

ALGORITHM:

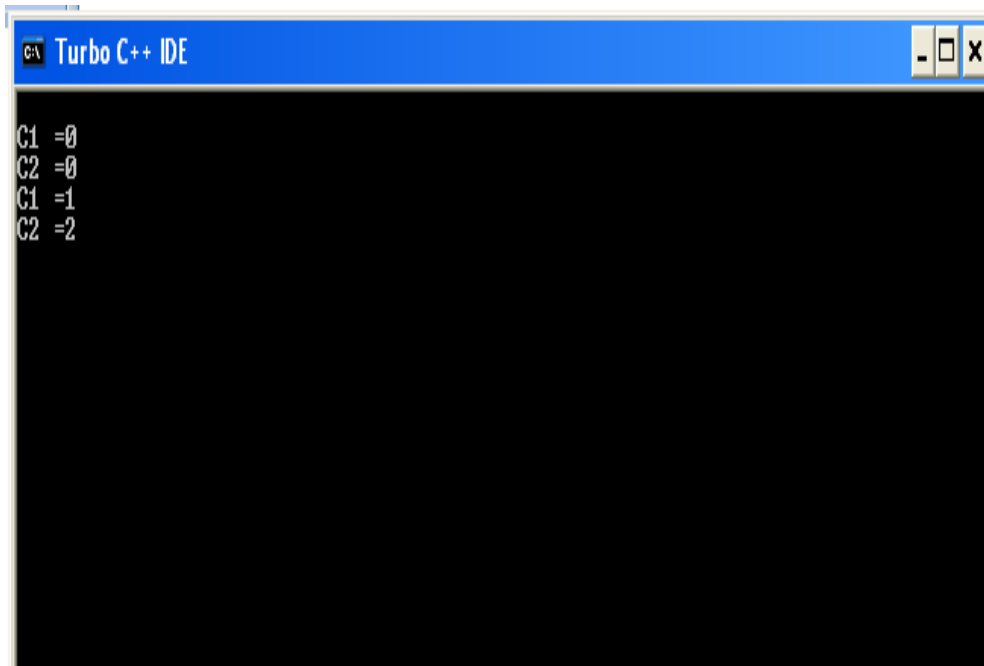
1. Start the process.
2. Invoke the class counter.
3. Create two objects c1 and c2 and assign values to them.
4. Call c1.get_count() and then Call c2.get_count()
5. Increment the values C1++ , C2++ and ++c2
6. Print c1 and c2.
7. Stop the process

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class counter
{
    int count;
public:
    counter()
    {
        count=0;
    }
    int get_count()
    {
        return count;
    }
    void operator++()
    {
        count++;
    }
};
void main()
```

```
{  
    counter c1,c2;  
    cout<<"\nC1 ="<<c1.get_count();  
    cout<<"\nC2 ="<<c2.get_count();  
  
    c1++;      //Using overloaded ++ operator.  
    c2++;  
    ++c2;  
    cout<<"\nC1 ="<<c1.get_count();  
    cout<<"\nC2 ="<<c2.get_count();  
    getch();  
}
```

OUTPUT



RESULT:

Thus the C++ program for Unary Operator Overloading was created, executed and output was verified successfully.

EX.NO:3**INHERITANCE****AIM:**

To write a C++ program for implementing the inheritance.

DESCRIPTION:

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

The idea of inheritance implements the is a relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

Base & Derived Classes

A class can be derived from more than one class, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class (es). A class derivation list names one or more base classes and has the form:

Class derived-class: access-specifier base-class

Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

Access Control and Inheritance

A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

A derived class inherits all base class methods with the following exceptions:

- Constructors, destructors and copy constructors of the base class.
- Overloaded operators of the base class.
- The friend functions of the base class.

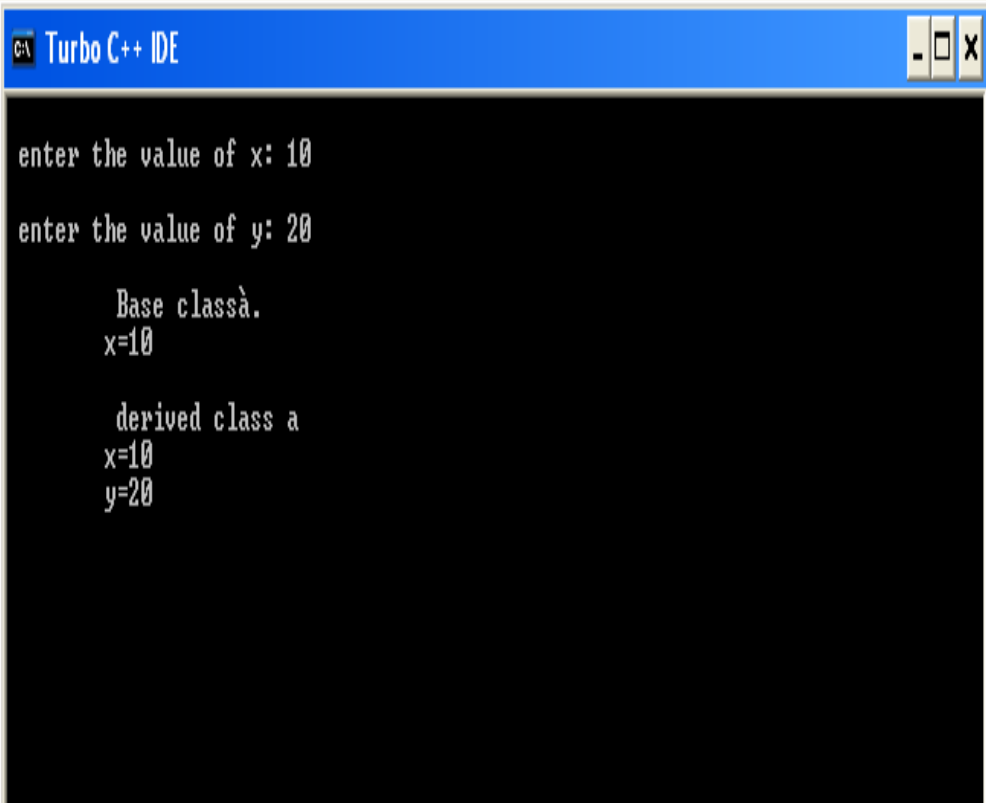
ALGORITHM:

1. Start the process.
2. Define the base class with variables and functions.
3. Define the derived class with variables and functions.
4. Get two values in main function.
5. Define the object for derived class in main function.
6. Access member of derived class and base class using the derived class object.
7. Stop the process.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class base
{
    public:
    int x;
    void set_x(int n)
    {
        x=n;
    }
    void show_x()
    {
        cout<<"\n\t Base class...";
        cout<<"\n\t x="<<x;
    }
};
class derived:public base
{
    int y;
    public:
    void set_y(int n)
    {
        y=n;
    }
    void show_xy()
    {
        cout<<"\n\n\t derived class...";
        cout<<"\n\t x="<<x;
        cout<<"\n\t y="<<y;
    }
};
void main()
{
    derived obj;
    int x,y;
    clrscr();
    cout<<"\n enter the value of x: ";
    cin>>x;
    cout<<"\n enter the value of y: ";
    cin>>y;
    obj.set_x(x); //inherits base class
```

```
obj.set_y(y); //access member of derived class
obj.show_x(); //inherits base class
obj.show_xy(); //access member of derived class
getch();
}
```

OUTPUT:A screenshot of the Turbo C++ IDE's output window. The window has a blue title bar with the text "Turbo C++ IDE" and standard window control buttons (minimize, maximize, close). The main area is black with white text. The text shows the program's execution: it prompts for the value of x (10) and y (20), then displays the state of the base class (x=10) and the derived class a (x=10, y=20).

```
enter the value of x: 10
enter the value of y: 20

Base class a.
x=10

derived class a
x=10
y=20
```

RESULT:

Thus the C++ program for inheritance was created, executed and output was verified successfully.

EX.NO:4**FUNCTION OVERLOADING****AIM:**

To write a C++ program for implementing the function overloading.

DESCRIPTION:

Function overloading in C++ You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

When you call an overloaded function or operator, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution.

Function overloading in C++

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

ALGORITHM:

1. Define a class test.
2. Define the function sum with different arguments with different data types.
3. Define the object for the class test in main function.
4. Call the function using the argument type to perform different operations.
5. Print the output.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class test
{
```

```

        public:
        int sum(int,int);
        float sum(float,float);
        double sum(double,double);
};
int test::sum(int a, int b)
{
    return(a+b);
}
float test::sum(float a ,float b)
{
    return(a+b);
}
double test::sum(double a, double b)
{
    return(a+b);
}
void main()
{
    test obj;
    int choice,ans;
    int a,b;
    float x,y;
    double m,n;
    double result=0;
    clrscr();
    cout<<"\n\t\t main menu";
    cout<<"\n\t1. Addition of two integer numbers";
    cout<<"\n\t2. Addition of two float numbers";
    cout<<"\n\t3. Addition of two double numbers"<<endl;
    do
    {
        cout<<"\n enter your choice:";
        cin>>choice;
        switch(choice)
        {
            case 1: cout<<"\n enter 2 numbers";
                    cin>>a>>b;
                    result=obj.sum(a,b);
                    break;
            case 2:
                    cout<<"\n enter 2 number";
                    cin>>x>>y;
                    result=obj.sum(x,y);
                    break;
            case 3:
                    cout<<"\n enter 2 number";
                    cin>>m>>n;
                    result=obj.sum(m,n);
                    break;
            default:
                    cout<<"wrong choice";
                    break;
        }
        cout<<"\n\n result"<<result<<endl;
    }
}

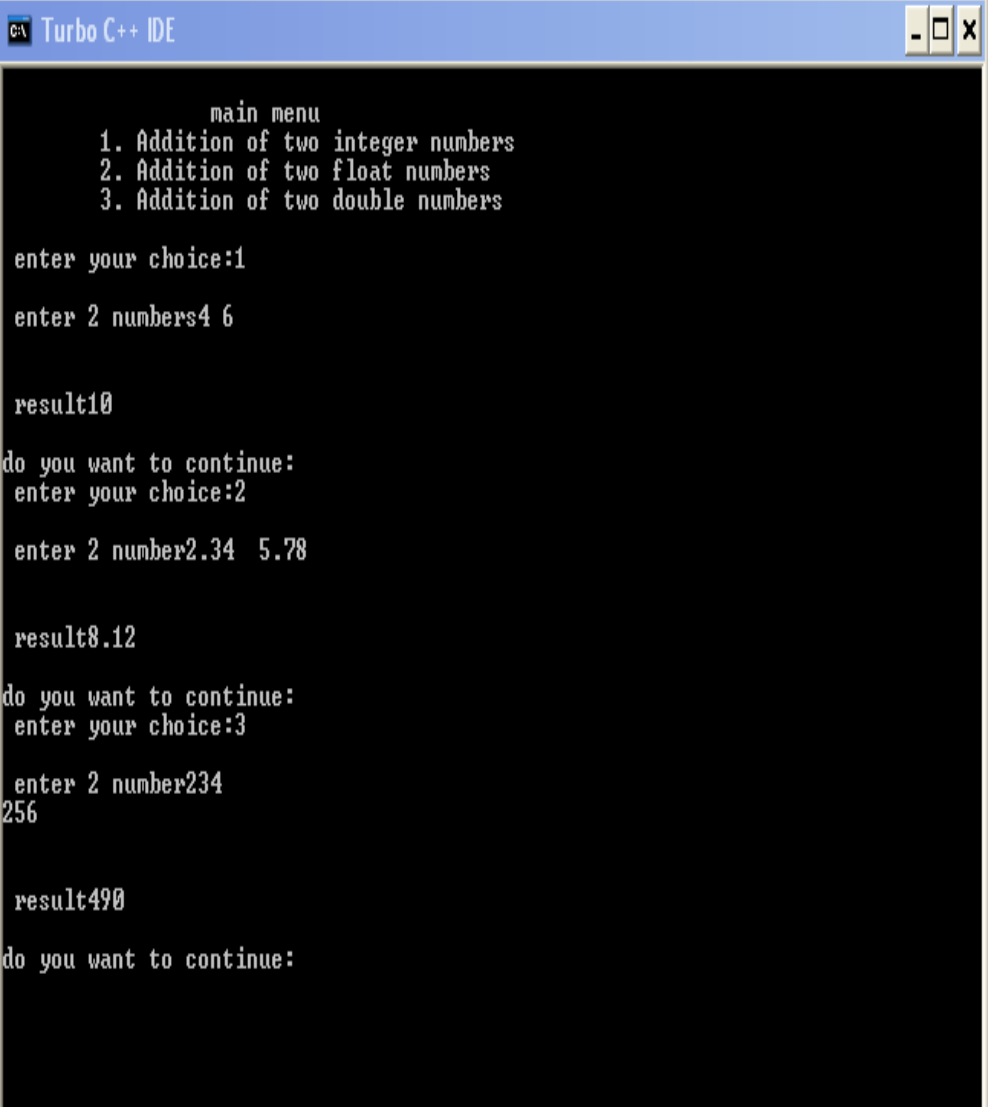
```

```

        cout<<"\ndo you want to continue:";
        ans=getch();
    }
    while(ans=='y' || ans=='Y');
    getch();
}

```

OUTPUT :



```

main menu
1. Addition of two integer numbers
2. Addition of two float numbers
3. Addition of two double numbers

enter your choice:1
enter 2 numbers4 6

result10
do you want to continue:
enter your choice:2
enter 2 number2.34 5.78

result8.12
do you want to continue:
enter your choice:3
enter 2 number234
256

result490
do you want to continue:

```

RESULT:

Thus the C++ program for function Overloading was created, executed and output was verified successfully.

EX.NO:5**ARRAY IMPLEMENTATION OF LIST ADT****AIM:**

To write a C++ program for array implementation of List ADT.

DESCRIPTION:

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Following are the important terms to understand the concept of Linked List.

- Link – each link of a linked list can store a data called an element.
- Next – each link of a linked list contains a link to the next link called Next.
- Linked List – A Linked List contains the connection link to the first link called First.

ALGORITHM:

Step1: Create nodes first, last; next, prev and cur then set the value as NULL.

Step 2: Read the list operation type.

Step 3: If operation type is create then process the following steps.

1. Allocate memory for node cur.
2. Read data in cur's data area.
3. Assign cur node as NULL.
4. Assign first=last=cur.

Step 4: If operation type is Insert then process the following steps.

1. Allocate memory for node cur.
2. Read data in cur's data area.
3. Read the position the Data to be insert.
4. Availability of the position is true then assing cur's node as first and first=cur.
5. If availability of position is false then do following steps.
 1. Assign next as cur and count as zero.
 2. Repeat the following steps until count less than postion.
 - 1 .Assign prev as next
 2. Next as prev of node.
 3. Add count by one.

4. If prev as NULL then display the message INVALID POSITION.

5. If prev not qual to NULL then do the following steps.

1. Assign cur's node as prev's node.
2. Assign prev's node as cur.

Step5: If operation type is delete then do the following steps.

1. Read the position .
2. Check list is Empty .If it is true display the message List empty.
3. If position is first.
 1. Assign cur as first.
 2. Assign First as first of node.
 3. Reallocate the cur from memory.
4. If position is last.
 1. Move the current node to prev.
 2. cur's node as Null.
 3. Reallocate the Last from memory.
 4. Assign last as cur.
5. If position is enter Mediate.
 1. Move the cur to required postion.
 2. Move the Previous to cur's previous position
 3. Move the Next to cur's Next position.
 4. Now Assign previous of node as next.
 5. Reallocate the cur from memory.

step 6: If operation is traverse.

1. Assign current as first.
2. Repeat the following steps untill cur becomes NULL.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
void create();
void insert();
void deletion();
void search();
void display();
int a,b[20],n,d,e,f,i;
void main()
{
    int c;
    clrscr();
```

```

cout<<"\n Main Menu";
cout<<"\n 1.Create \n 2.Delete \n 3.Search \n 4.insert \n
5.Display \n 6.Exit";
do
{
    cout<<"\n enter your choice:";
    cin>>c;
    switch(c)
    {
        case 1: create(); break;
        case 2: deletion(); break;
        case 3: search(); break;
        case 4: insert(); break;
        case 5: display(); break;
        case 6: exit(0); break;
        default:
            cout<<"The given number is not between 1-5\n";
    }
}
while(c<=6);
getch();
}
void create()
{
    cout<<"\n Enter the number of elements you want to
create: "; cin>>n;
    cout<<"\n enter the elements\n";
    for(i=0;i<n;i++)
    {
        cin>>b[i];
    }
}
void deletion()
{
    cout<<"Enter the number u want to delete \n";
    cin>>d;
    for(i=0;i<n;i++)
    {
        if(b[i]==d)
        {
            b[i]=0;
            cout<<d<<"  deleted";
        }
    }
}
void search()
{
    cout<<"Enter the number \n";
    cin>>e;
    for(i=0;i<n;i++)
    {
        if(b[i]==e)
        {
            cout<<"Value found the position\n"<<i+1;
        }
    }
}

```

```
    }  
}  
void insert()  
{  
    cout<<"\nenter how many number u want to insert: ";  
    cin>>f;  
    cout<<"\nEnter the elements\n";  
    for(i=0;i<f;i++)  
    {  
        cin>>b[n++];  
    }  
}  
void display()  
{  
    for(i=0;i<n;i++)  
    {  
        cout<<"\n"<<b[i];  
    }  
}
```

OUTPUT:

```

Turbo C++ IDE
3.Search
4.insert
5.Display
6.Exit
enter your choice:1

Enter the number of elements you want to create: 4

enter the elements
12
34
55
67

enter your choice:5

12
34
55
67
enter your choice:2
Enter the number u want to delete
34
34 deleted
enter your choice:5

12
0
55
67
enter your choice:3
Enter the number
55
Value found the position
3
enter your choice:4

enter how many number u want to insert: 1

Enter the elements
68

enter your choice:5

12
0
55
67
68
enter your choice:6

```

RESULT:

Thus the C++ program for array implementation of list ADT was created, executed and output was verified successfully

EX.NO:6**LINKED LIST IMPLEMENTATION OF LIST ADT**

AIM:

To write a C++ program to implement a linked list and do all operations on it.

DESCRIPTION:

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Following are the important terms to understand the concept of Linked List.

- Link – each link of a linked list can store a data called an element.
- Next – each link of a linked list contains a link to the next link called Next.
- Linked List – A Linked List contains the connection link to the first link called First.

Linked list can be visualized as a chain of nodes, where every node points to the next node.

- Linked List contains a link element called first.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Types of Linked List

Following are the various types of linked list.

- Simple Linked List – Item navigation is forward only.
- Doubly Linked List – Items can be navigated forward and backward.
- Circular Linked List – Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

Following are the basic operations supported by a list.

- Insertion – Adds an element at the beginning of the list.
- Deletion – Deletes an element at the beginning of the list.
- Display – Displays the complete list.
- Search – Searches an element using the given key.
- Delete – Deletes an element using the given key.

ALGORITHM:

1. Start the process.

2. Initialize and declare variables.
3. Enter the choice.
4. If choice is INSERT then
 - a. Enter the element to be inserted.
 - b. Get a new node and set DATA[NEUNODE] = ITEM.
 - c. Find the node after which the new node is to be inserted.
 - d. Adjust the link fields.
 - e. Print the linked list after insertion.
5. If choice is DELETE then
 - a. Enter the element to be deleted.
 - b. Find the node containing the element (LOC) and its preceding node (PAR).
 - c. Set ITEM = DATA[LOC] and delete the node LOC.
 - d. Adjust the link fields so that PAR points to the next element. ie LINK[PAR] = LINK [LOC].
 - e. Print the linked list after deletion.
6. Stop the process.

PROGRAM:

```

#include<iostream.h>
#include<conio.h>
#include<lib.h>
#define TRUE 1
#define FALSE 0
class sll
{
    private:
    struct node
    {
        int data;
        struct node *next;
    }*head;
    public:
        sll();
        void create();
        void display();
        void search(int key);
        void insert_head();
        void insert_after();
        void insert_last();
        void dele();
        ~sll();
};

sll::sll()
{
    head=NULL;
}

```

```

sll::~sll()
{
    node *temp,*temp1;
    temp=head->next;
    delete head;
    while(temp!=NULL)
    {
        temp1=temp->next;
        delete temp;
        temp=temp1;
    }
}

void sll::create()
{
    node *temp,*New;
    int val,flag;
    char ans='y';
    flag=TRUE;
    do
    {
        cout<<"\n\nEnter the data:";
        cin>>val;
        New=new node;
        if(New==NULL)
            cout<<"unable to allocate memory\n";
        New->data=val;
        New->next=NULL;
        if(flag==TRUE)//executed only for the first time
        {
            head=New;
            temp=head;
            flag=FALSE;
        }
        else
        {
            temp->next=New;
            temp=New;
        }
        cout<<"\n do you want to enter more elements?(y/n) ";
        ans=getch();
    }
    while(ans=='y' || ans=='Y');
    cout<<"\n the singly linked list is created\n";
    getch();
}

void sll::display()
{
    node *temp;
    temp=head;
    if(temp==NULL)
    {
        cout<<"\n the list is empty\n";
        getch();
        clrscr();
        return;
    }
}

```

```

    }
    while(temp!=NULL)
    {
        cout<<temp->data<<" ";
        temp=temp->next;
    }
    getch();
}
void sll::search(int key)
{
    node *temp;
    int found;
    temp=head;
    if(temp==NULL)
    {
        cout<<"linked list is empty\n";
        getch();
        clrscr();
    }
    found=FALSE;
    while(temp!=NULL&&found==FALSE)
    {
        if(temp->data!=key)
            temp=temp->next;
        else
            found=TRUE;
    }
    if(found==TRUE)
    {
        cout<<"\n the element is present in the list\n";
        getch();
    }
    else
    {
        cout<<"\nThe element is not present in the list\n";
        getch();
    }
}
void sll::dele()
{
    node *temp,*prev;
    int key;
    temp=head;
    cout<<"\n Enter the data of the node you want to delete:
    ";
    cin>>key;
    while(temp!=NULL)
    {
        if (temp->data==key)//traverse till required node to
            delete
            break; //is found
        prev=temp;
        temp=temp->next;
    }
    if(temp==NULL)

```

```

        cout<<"\n node not found";
    else
    {
        if(temp==head)//first node
        head=temp->next;
        else
        prev->next=temp->next; //intermediate or end node
        delete temp;
        cout<<"\n the element is deleted\n";
    }
    getch();
}
void sll::insert_last()
{
    node *New,*temp;
    cout<<"\n Enter the element which you want to insert: ";
    cin>>New->data;
    if(head==NULL)
    head=New;
    else
    {
        temp=head;
        while(temp->next!=NULL)
        temp=temp->next;
        temp->next=New;
        New->next=NULL;
    }
}
void sll::insert_after()
{
    int key;
    node *temp,*New;
    New=new node;
    cout<<"\n Enter the element which you want to insert: ";
    cin>>New->data;
    if(head==NULL)
    {
        head=New;
    }
    else
    {
        cout<<"\n Enter the element after which you want to
        insert the node: ";
        cin>>key;
        temp=head;
        do
        {
            if(temp->data==key)
            {
                New->next=temp->next;
                temp->next=New;
                break;
            }
            else
                temp=temp->next;
        }
    }
}

```

```

        }
        while (temp!=NULL) ;
    }
}
void sll::insert_head()
{
    node *New,*temp;
    New=new node;
    cout<<"\n Enter the element which you want to insert: ";
    cin>>New->data;
    if(head==NULL)
        head=New;
    else
    {
        temp=head;
        New->next=temp;
        head=New;
    }
}
void main()
{
    sll s;
    int choice,val,
    chl;
    char ans='y';
    clrscr();
    cout<<"\n\n\tProgram to perform various operations on
    linked list";
    cout<<"\n1.Create";
    cout<<"\n2.Display";
    cout<<"\n3.Search";
    cout<<"\n4.Insert an element in a list";
    cout<<"\n5.Delete an element from list";
    cout<<"\n6.Quit";
    do
    {
        cout<<"\n\nEnter your choice(1-6): ";
        cin>>choice;
        switch(choice)
        {
            case 1: s.create();
            break;
            case 2:s.display();
            break;
            case 3:cout<<"enter the element you want to
            search";
            cin>>val;
            s.search(val);
            break;
            case 4:
            clrscr();
            cout<<"\n the list is:\n";
            s.display();
            cout<<"\n menu";
            cout<<"\n1.insert at beginning ";

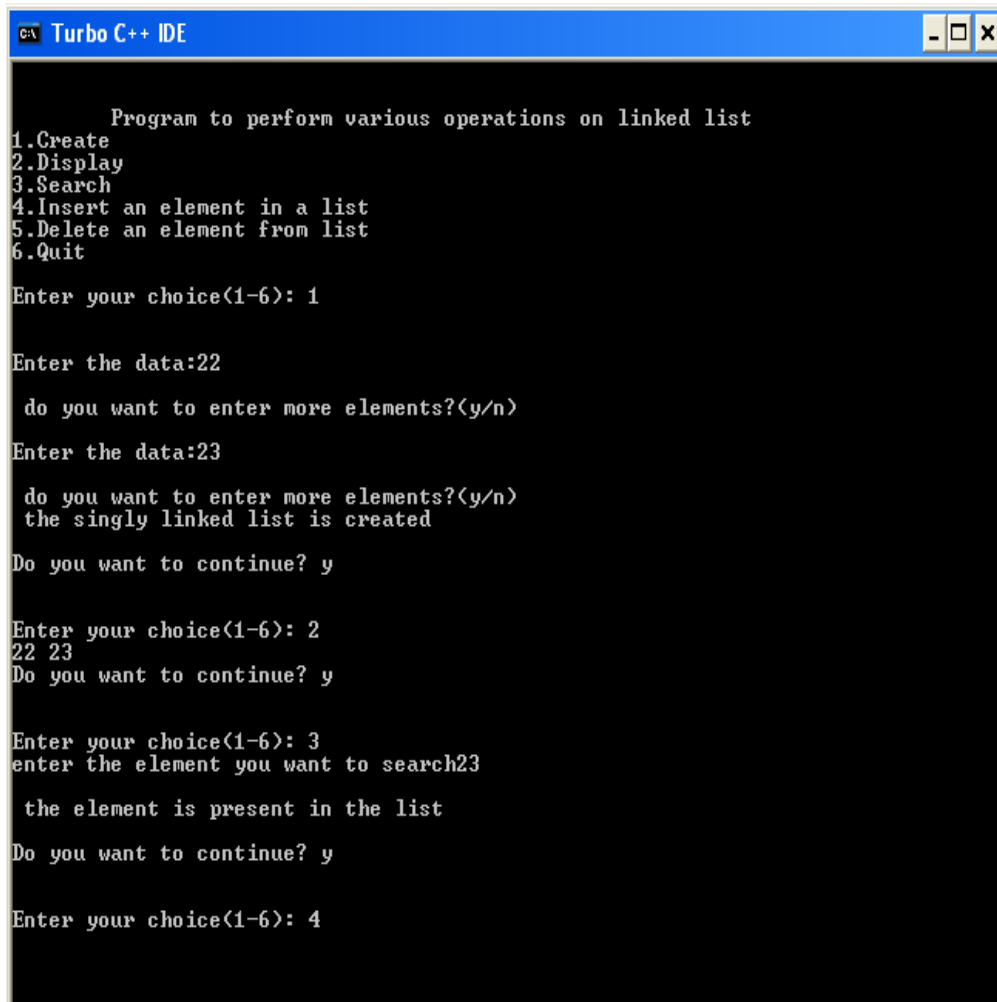
```

```

        cout<<"\n2.insert after";
        cout<<"\n3.insert at end";
        cout<<"\n enter your choice";
        cin>>ch1;
        switch(ch1)
        {
            case 1:s.insert_head();
            break;
            case 2:s.insert_after();
            break;
            case 3:s.insert_last();
            break;
            default:cout<<"\n invalid choice";
        }
        break;
        case 5:s.dele();
        break;
        case 6:exit(0);
        default:cout<<"\n invalid choice";
    }
    cout<<"\nDo you want to continue? ";
    cin>>ans;
}
while(ans=='y' || ans=='Y');
return;
}

```

OUTPUT:

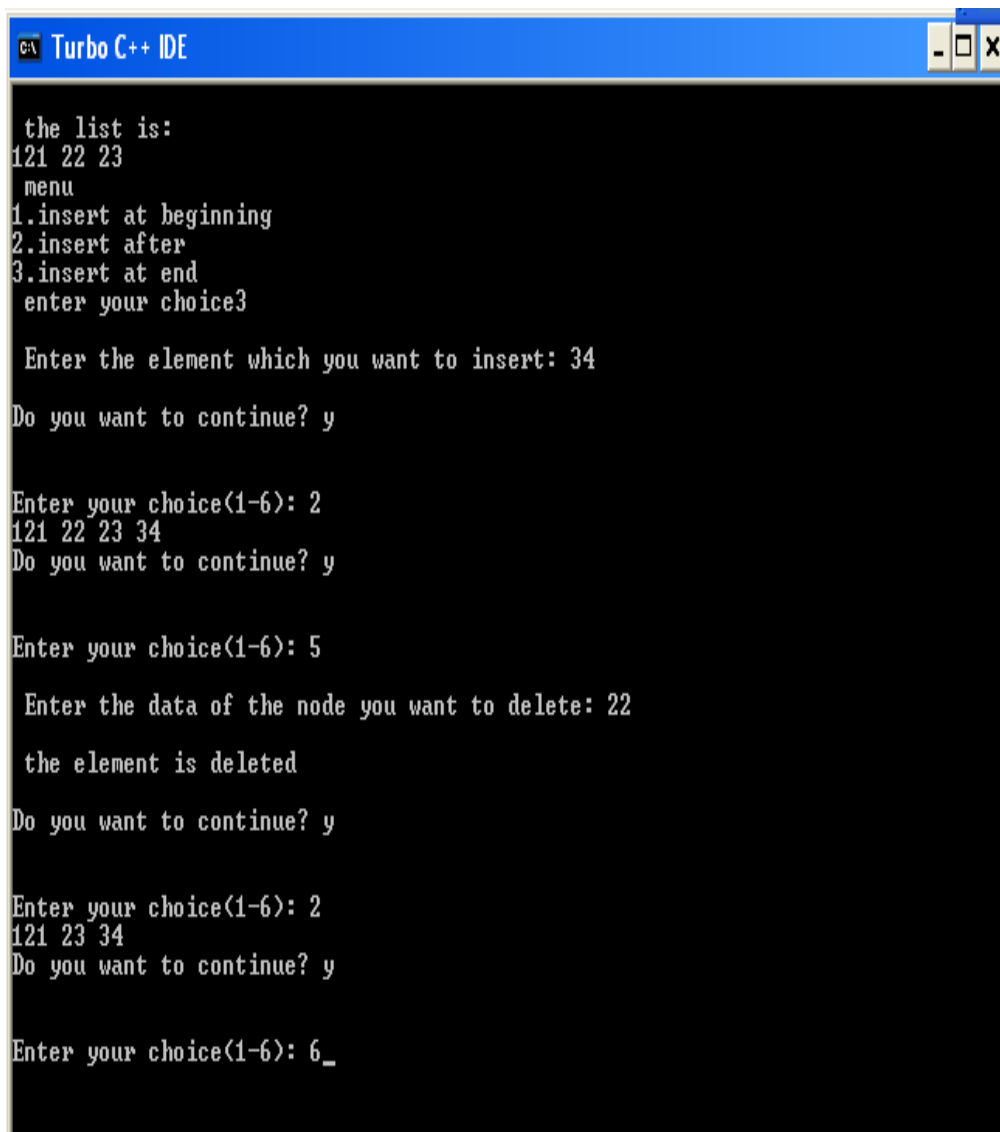


The screenshot shows the Turbo C++ IDE window with a black background and white text. The program is titled "Program to perform various operations on linked list". It lists six menu options: 1.Create, 2.Display, 3.Search, 4.Insert an element in a list, 5.Delete an element from list, and 6.Quit. The user enters choice 1, then provides data 22 and 23, and answers 'y' to the question "do you want to enter more elements?(y/n)". The program outputs "the singly linked list is created". The user then enters choice 3, provides the search element 23, and the program outputs "the element is present in the list". Finally, the user enters choice 4.

```

Turbo C++ IDE
Program to perform various operations on linked list
1.Create
2.Display
3.Search
4.Insert an element in a list
5.Delete an element from list
6.Quit
Enter your choice<1-6>: 1
Enter the data:22
do you want to enter more elements?(y/n)
Enter the data:23
do you want to enter more elements?(y/n)
the singly linked list is created
Do you want to continue? y
Enter your choice<1-6>: 2
22 23
Do you want to continue? y
Enter your choice<1-6>: 3
enter the element you want to search23
the element is present in the list
Do you want to continue? y
Enter your choice<1-6>: 4

```



```
the list is:
121 22 23
menu
1.insert at beginning
2.insert after
3.insert at end
enter your choice3

Enter the element which you want to insert: 34

Do you want to continue? y

Enter your choice(1-6): 2
121 22 23 34
Do you want to continue? y

Enter your choice(1-6): 5

Enter the data of the node you want to delete: 22

the element is deleted

Do you want to continue? y

Enter your choice(1-6): 2
121 23 34
Do you want to continue? y

Enter your choice(1-6): 6_
```

RESULT:

Thus the C++ program for linked list implementation of list ADT was created, executed and output was verified successfully.

EX.NO:7**CURSOR IMPLEMENTATION OF LIST ADT****AIM:**

To write a C++ program for cursor implementation of list ADT.

DESCRIPTION:

If linked lists are required and pointers are not available, then an alternate implementation must be used. The alternate method we will describe is known as a cursor implementation. The two important items present in a pointer implementation of linked lists are

1. The data is stored in a collection of structures. Each structure contains the data and a pointer to the next structure.
2. A new structure can be obtained from the system's global memory by a call to malloc and released by a call to free.

Algorithm:

1. Start the program.
2. Create a node with two fields' data and link field. O Allocate space for the node dynamically.
3. Create link between the created nodes and let the last node be with NULL Link
4. Insert the input data in the data field and press -1 to stop the same.
5. Get the choice of operations either insertion or deletion.
6. For insertion get the position in which insertion is to be done and the element to be inserted. Check for the start, middle or end position of insertion. Insert the node and change its link accordingly.
7. For deletion get the position in which deletion is to be done. Delete the node and then link it to the next node. Before deletion check whether there is data in the list to be deleted.
8. Using display option lists the elements of the list.
9. Stop the program.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 20
class LIST
{
```

```

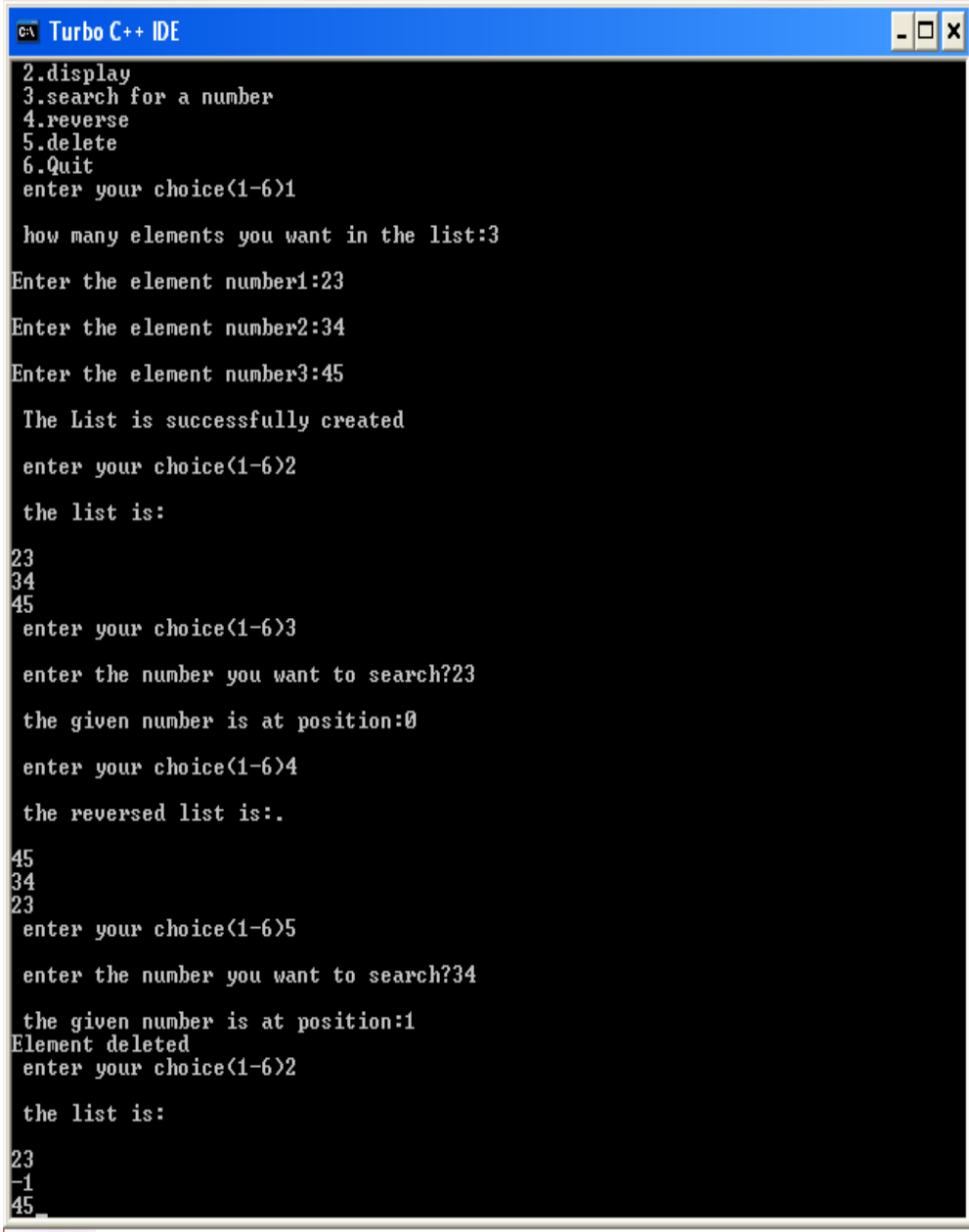
        private:
        int List[MAX];
        public:
        int create();
        void display(int);
        void reverse(int);
        int search(int);
        void delet(int);
};
int LIST::create()
{
    int n,i;
    cout<<"\n how many elements you want in the list:";
    cin>>n;
    if(n>MAX)
    cout<<"\n Error: Number of elements exceeds the limit";
    for(i=0;i<n;i++)
    {
        cout<<"\nEnter the element number"<<i+1<<":";
        cin>>List[i];
    }
    cout<<"\n The List is successfully created\n";
    return(n);
}
void LIST::display(int n)
{
    int i;
    cout<<"\n the list is:\n";
    for(i=0;i<n;i++)
    cout<<"\n"<<List[i];
}
void LIST::reverse(int n)
{
    int i;
    cout<<"\n the reversed list is:.\n";
    for(i=n-1;i>=0;i--)
    cout<<"\n"<<List[i];
}
int LIST::search(int n)
{
    int i,key;
    cout<<"\n enter the number you want to search?";
    cin>>key;
    for (i=0;i<n;i++)
    {
        if(List[i]==key)
        {
            cout<<"\n the given number is at
            position:"<<i<<"\n";
            return i;
        }
    }
    cout<<"\n the given number is not in the list\n";
    return -1;
}

```

```

void LIST::delet(int n)
{
    int i;
    i=search(n);
    if(i!=NULL)
    {
        List[i]=-1;
        cout<<"Element deleted";
    }
}
void main()
{
    LIST obj;
    int choice,len,position;
    char ans;
    clrscr();
    cout<<"\n\t program to perform operations on ordered
list"; cout<<"\n 1.create";
    cout<<"\n 2.display";
    cout<<"\n 3.search for a number";
    cout<<"\n 4.reverse";
    cout<<"\n 5.delete";
    cout<<"\n 6.Quit";
    do
    {
        cout<<"\n enter your choice(1-6) ";
        cin>>choice;
        switch(choice)
        {
            case 1:
                len=obj.create();
                break;
            case 2:
                obj.display(len);
                break;
            case 3:
                position=obj.search(len);
                break;
            case 4:
                obj.reverse(len);
                break;
            case 5:
                obj.delet(len);
                break;
            case 6:
                exit(0);
                break;
            default:
                cout<<"\n invalid choice, try again";
                break;
        }
        getch();
    }while(choice!=6);
}

```

OUTPUT :


```

Turbo C++ IDE
2.display
3.search for a number
4.reverse
5.delete
6.Quit
enter your choice(1-6)1

how many elements you want in the list:3
Enter the element number1:23
Enter the element number2:34
Enter the element number3:45

The List is successfully created

enter your choice(1-6)2

the list is:
23
34
45
enter your choice(1-6)3

enter the number you want to search?23

the given number is at position:0

enter your choice(1-6)4

the reversed list is:.
45
34
23
enter your choice(1-6)5

enter the number you want to search?34

the given number is at position:1
Element deleted
enter your choice(1-6)2

the list is:
23
-1
45

```

RESULT:

Thus the C++ program for cursor implementation of list ADT was created, executed and output was verified successfully.

EX.NO:8**ARRAY IMPLEMENTATION OF STACK ADT****AIM:**

To write a C++ program for stack using array implementation.

DESCRIPTION:

A stack data structure can be implemented using one dimensional array. But stack implemented using array, can store only fixed number of data values. This implementation is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable 'top'. Initially top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

ALGORITHM:

1. Define a array which stores stack elements..
2. The operations on the stack are
 - a. PUSH data into the stack
 - b. POP data out of stack
3. PUSH DATA INTO STACK
 - a. Enter the data to be inserted into stack.
 - b. If TOP is NULL
 - i. The input data is the first node in stack.
 - ii. The link of the node is NULL.
 - iii. TOP points to that node.
 - c. If TOP is NOT NULL
 - i. The link of TOP points to the new node.
 - ii. TOP points to that node.
4. POP DATA FROM STACK
 - a. If TOP is NULL
 - i. the stack is empty
 - b. If TOP is NOT NULL
 - i. The link of TOP is the current TOP.
 - ii. The pervious TOP is popped from stack.
5. The stack represented by linked list is traversed to display its content.

PROGRAM:

```

#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class stack
{
    int stk[5];
    int top;
public:
    stack()
    {
        top=-1;
    }
    void push(int x)
    {
        if(top > 4)
        {
            cout <<"stack over flow";
            return;
        }
        stk[++top]=x;
        cout <<"inserted" <<x;
    }
    void pop()
    {
        if(top <0)
        {
            cout <<"stack under flow";
            return;
        }
        cout <<"deleted" <<stk[top--];
    }
    void display()
    {
        if(top<0)
        {
            cout <<" stack empty";
            return;
        }
        for(int i=top;i>=0;i--)
            cout <<stk[i] <<" ";
    }
};

main()
{
    int ch;
    stack st;
    clrscr();
    cout <<"\n1.push  \n2.pop  \n3.display  \n4.exit";
    while(1)
    {
        cout<<"\nEnter ur choice";
        cin >> ch;
    }
}

```

```

        switch(ch)
        {
            case 1:  cout <<"enter the element";
                    cin >> ch;
                    st.push(ch);
                    break;
            case 2:  st.pop();  break;
            case 3:  st.display();break;
            case 4:  exit(0);
        }
    }
    return (0);
}

```

OUTPUT:

```

Turbo C++ IDE
1.push
2.pop
3.display
4.exit
Enter ur choice1
enter the element23
inserted23
Enter ur choice1
enter the element23
inserted23
Enter ur choice1
enter the element345
inserted345
Enter ur choice1
enter the element342
inserted342
Enter ur choice3
342 345 23 23
Enter ur choice2
deleted342
Enter ur choice3
345 23 23
Enter ur choice4_

```

RESULT:

Thus the C++ program for array implementation of stack ADT was created, executed and output was verified successfully.

EX. NO: 9**STACK ADT USING LINKED LIST****AIM:**

To write a C++ program for stack ADT using linked list implementation.

DESCRIPTION:

The major problem with the stack implemented using array is, it works only for fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using linked list can work for unlimited number of values. That means, stack implemented using linked list works for variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its next node in the list. The next field of the first element must be always NULL.

ALGORITHM:

1. Define a struct for each node in the stack. Each node in the stack contains data and link to the next node. TOP pointer points to last node inserted in the stack.
2. The operations on the stack are
 - a. PUSH data into the stack
 - b. POP data out of stack
3. PUSH DATA INTO STACK
 - a. Enter the data to be inserted into stack.
 - b. If TOP is NULL
 - i. The input data is the first node in stack.
 - ii. The link of the node is NULL.
 - iii. TOP points to that node.
 - c. If TOP is NOT NULL
 - i. The link of TOP points to the new node.
 - ii. TOP points to that node.
4. POP DATA FROM STACK

- a. 4a.If TOP is NULL
 - i. the stack is empty
 - b. 4b.If TOP is NOT NULL
 - i. The link of TOP is the current TOP.
 - ii. The pervious TOP is popped from stack.
5. The stack represented by linked list is traversed to display its content.

PROGRAM:

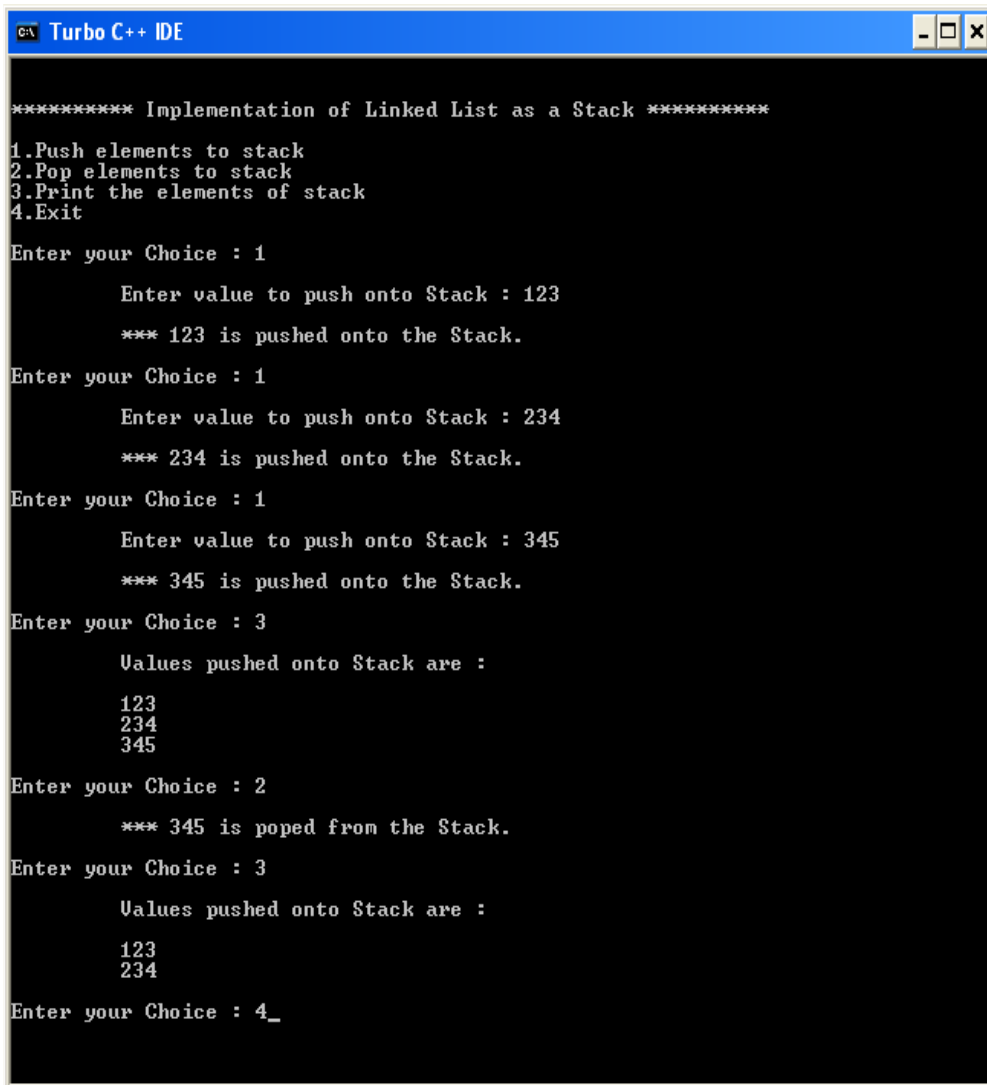
```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class Linked_list_Stack
{
    private:
    struct node
    {
        int data;
        node *next;
    };
    node *top;
    node *entry;
    node *print;
    node *bottom;
    node *last_entry;
    node *second_last_entry;
    public:
    Linked_list_Stack( );
    void pop( );
    void push( );
    void print_list( );
    void show_working( );
};
Linked_list_Stack::Linked_list_Stack ( )
{
    top=NULL;
    bottom=NULL;
}
void Linked_list_Stack::push( )
{
    int num;
    cout<<"\n\t Enter value to push onto Stack : ";
    cin>>num;
    entry=new node;
    if (bottom==NULL)
    {
        entry->data=num;
        entry->next=NULL;
        bottom=entry;
        top=entry;
    }
    else
```

```

        {
            entry->data=num;
            entry->next=NULL;
            top->next=entry;
            top=entry;
        }
        cout<<"\n\t *** "<<num<<" is pushed onto the
        Stack."<<endl;
    }
void Linked_list_Stack::pop( )
{
    if(bottom==NULL)
        cout<<"\n\t ***   Error : Stack is empty. \n"<<endl;
    else
    {
        for(last_entry=bottom; last_entry->next!=NULL;
            last_entry=last_entry->next)
            second_last_entry=last_entry;
        if(top==bottom)
            bottom=NULL;
        int popped_element=top->data;
        delete top;
        top=second_last_entry;
        top->next=NULL;
        cout<<"\n\t *** "<<popped_element<<" is popped from
        the Stack."<<endl;
    }
}
void Linked_list_Stack::print_list( )
{
    print=bottom;
    if(print!=NULL)
        cout<<"\n\t Values pushed onto Stack are : \n"<<endl;
    else
        cout<<"\n\t *** Nothing to show. "<<endl;
    while(print!=NULL)
    {
        cout<<"\t "<<print->data<<endl;
        print=print->next;
    }
}
void Linked_list_Stack::show_working( )
{
    int choice;
    clrscr( );
    cout<<"\n\n***** Implementation of Linked List as a
    Stack *****"<<endl;
    cout<<"\n1.Push elements to stack"<<endl;
    cout<<"2.Pop elements to stack"<<endl;
    cout<<"3.Print the elements of stack"<<endl;
    cout<<"4.Exit"<<endl;
    do
    {
        cout<<"\nEnter your Choice : ";
        cin>>choice;
    }
}

```

```
        switch(choice)
        {
            case 1: push();
            break;
            case 2: pop();
            break;
            case 3: print_list( );
            break;
            case 4: exit(0);
            break;
            default:
                cout<<"enter the valid choice";
        }
    }while(1);
}
int main( )
{
    Linked_list_Stack obj;
    obj.show_working( );
    return 0;
}
```

OUTPUT:


```

Turbo C++ IDE

***** Implementation of Linked List as a Stack *****
1.Push elements to stack
2.Pop elements to stack
3.Print the elements of stack
4.Exit
Enter your Choice : 1
    Enter value to push onto Stack : 123
    *** 123 is pushed onto the Stack.
Enter your Choice : 1
    Enter value to push onto Stack : 234
    *** 234 is pushed onto the Stack.
Enter your Choice : 1
    Enter value to push onto Stack : 345
    *** 345 is pushed onto the Stack.
Enter your Choice : 3
    Values pushed onto Stack are :
    123
    234
    345
Enter your Choice : 2
    *** 345 is popped from the Stack.
Enter your Choice : 3
    Values pushed onto Stack are :
    123
    234
Enter your Choice : 4_
  
```

RESULT:

Thus the C++ program for stack ADT using linked list was created, executed and output was verified successfully.

EX. NO: 10 PROGRAM SOURCE FILES FOR STACK APPLICATION1**Application 1: Checking well formedness of parenthesis.**

STACK IMPLEMENTED AS ARRAYS (USING THE HEADER FILE OF STACK OPERATIONS)

Step 1: Create a header file named stack.h. In this file we will declare the class and all the stack operations. The implementation of stack is using arrays.

```
#define size 10
class stk_class
{
    private:
        /*stack structure*/
        struct stack
        {
            char s[size];
            int top;
        }st;
    public:
        stk_class();
        void push(char item);
        int stempty();
        char pop();
};

stk_class::stk_class()
{
    st.top=-1;
}

/* the push function
input:item which is to be pished
output:none -simply pushes the item onto the stack
called by: main
calls: one
*/
void stk_class::push(char item)
{
    st.top++;
    st.s[st.top]=item;
}

/*
the stempty function
input:none
output:returns 1 or 0 for stack empty or not
called by:none
*/
int stk_class::stempty()
{
    inr(st.top== -1)
    return 1;
}
```

```

        else
            return 0;
    }
    /*
    the stfull function
    input:none
    coutput:returns 1 or 0 for stack full or not
    called by :main
    calls:none
    */
    int stk_class::stfull()
    {
        if(st.top==size)
            return 1;
        else
            return 0;
    }
    /*
    the pop function
    input:none
    output:returns the item which is popped from the stack
    called by: main
    calls:none
    */
    char stk_class::pop()
    {
        char item;
        item=st.s[st.top];
        st.top--;
        return(item);
    }

```

Step 2: Now we will create a stack application program. We have chosen an application as “checking well formedness of parenthesis” for this application we will use stack operations. These operations are used from the external file stack.h. Hence we will include this file in the include file section. Here is an application program.

```

/*****

```

program for checking the well formedness of the parenthesis using stack as arrays.

```

*****/

```

```

#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include "C:\TC\INCLUDE\stack.h"
#define size 10
/*
the main function
input:none
ouput:none
called by:O.S
calls:push,pop,stempty
*/

```

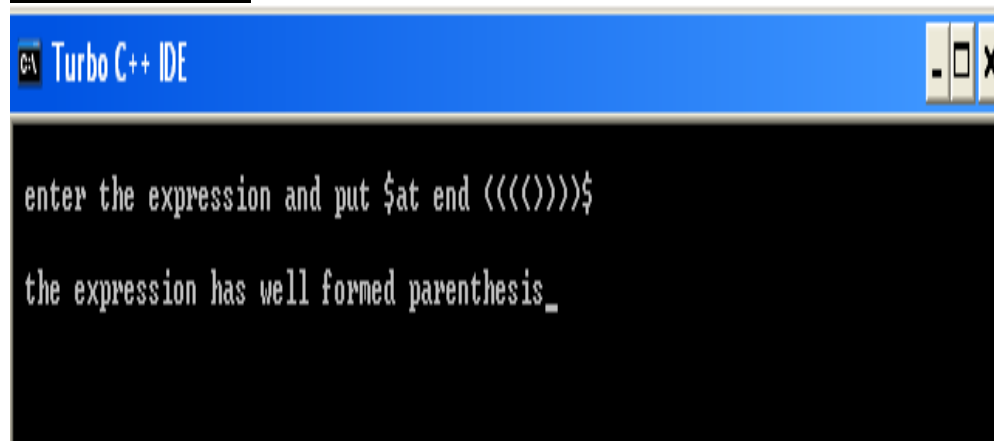
```

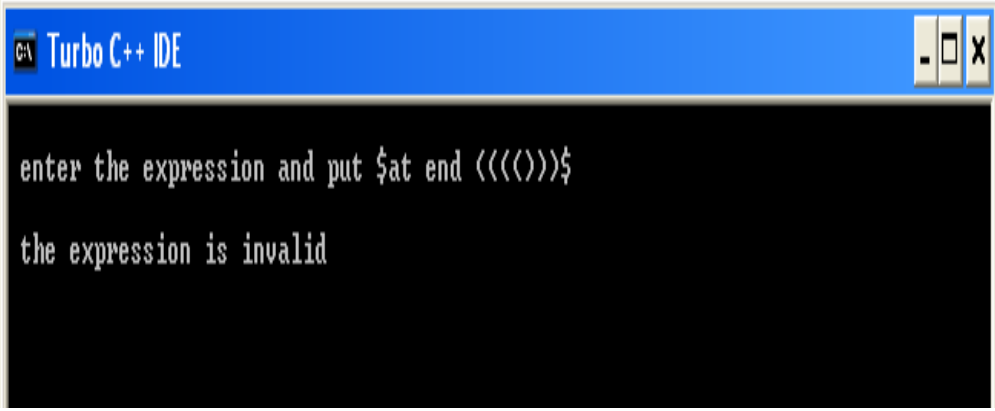
void main(void)
{
    char item;
    char ans,bracket[10];
    stk_class obj;
    int i;
    clrscr();
    cout<<"\n\t\t program for stack application using separate
header file"; cout<<"\n enter the expression and put $at
end ";
    cin>>bracket;
    i=0;
    if(bracket[i]=='')
    cout<<"\n the expressin is invalid";
    else
    {
        do
        {
            ile(bracket[i]=='(')
            {
                obj.push(bracket[i]);
                i++;
            }
            while(bracket[i]=='')
            {
                item=obj.pop();
                i++;
            }
        }
        while(bracktk[i]!='$')
        if(!obj.stempty())
        cout<<"\n the expression is invalid";
        else
        cout<<"\n the expression has well formed
parenthesis";
    }
    getch();
}

```

Step 3: Execute above program and following output can be obtained.

OUTPUT (RUN 1)



OUTPUT (RUN 2)A screenshot of the Turbo C++ IDE window. The title bar is blue and contains the text "Turbo C++ IDE" and standard window control buttons (minimize, maximize, close). The main area is black with white text. The first line of text is "enter the expression and put \$at end (((())))\$", and the second line is "the expression is invalid".

```
Turbo C++ IDE  
enter the expression and put $at end (((())))$  
the expression is invalid
```

RESULT:

Thus the C++ program for balanced parentheses checking using array implementation of stack ADT was created, executed and output was verified successfully.

B) STACK IMPLEMENTED AS LINKED LIST(USING THE HEADER FILE OF STACK OPERATION)

Step 1: create a header file named stack. In this file we declare the class and all the stack operations.

The implementation of stack using linked list.

```

/*****
Stack1.h file
*****/

class stk_class
{
private:
/*data structure for the linked stack*/
typedef struct stack
{
    char data;
    struct stack * next;
}node;
node *top;
public:
stk_class();
void push(char item);
int sempty();
void pop();
};
stk_class ::stk_class()
{
    top=NULL;
}
/* functionality performed on linked stack*/
void stk_class::push(char item)
{
    node * New;
    New=new node;
    if(New==NULL)
        cout<<"\n memory cannot be allocated \n";
    else
    {
        New->data=item;
        new->next=top;
        top=New;
    }
}
/* the sempty function
input: any node for checking the empty condition
output:1 or 0 for empty or not condition
called by: main
calls: none
*/
int stk_class::sempty()
{

```

```

        if(top==NULL)
            return 1;
        else
            return 0;
    }
    /*.....
the pop function
input: the top of the stack
called by: main
calls:none
.....*/
void stk_class::pop()
{
    node *temp;
    temp=top;
    top=top->next;
    delete temp;
}

```

Step 2: Now we will create a stack application program. We have chosen an application as “checking well formedness of parenthesis” for this application we will use stack operations. These operations are used from the external file stack.h. Hence we will include this file in the include file section. Here is an application program.

```

/*.....
Program for checking the well – formedness of the parenthesis using linked stack.
.....*/

/*include file section*/
#include<iostream.h>
#include<conio.h>
#include<process.h>
#include<stdlib.h>
/* Included linked stack as a header file*/
#include<"C:\TC\INCLUDE\stack1.h"
/*
the main function
input:none
output:none
called by:O.S
calls:push ,pop,empty
*/
void main(void)
{
    char ans,bracket[10];
    Char data ,item;
    stk_class obj;
    int choice;
    int I;
    clrscr();
    cout<<"\n\t\t enter the expression and put $ at the end
";

```

```

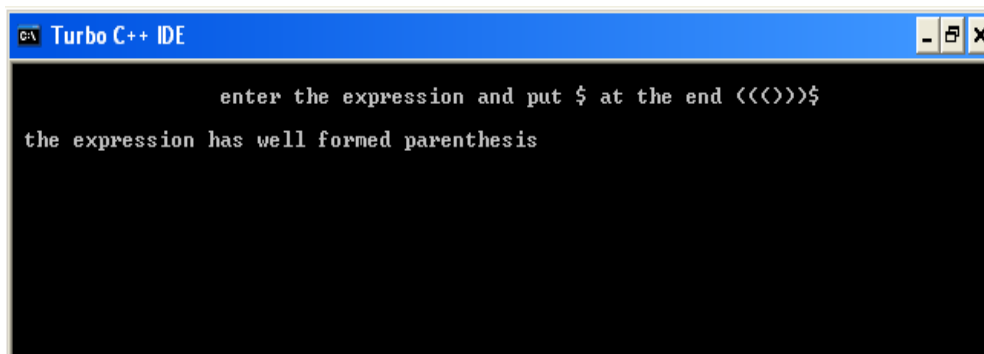
cin>>bracket;
i=0;
if(bracket[i]=='(')
cout<<"\n the expression is invalid";
else
{
    do
    {
        if(bracket[i]=='(')
        {
            obj.push(bracket[i]);
        }
        else if(bracket[i]==')')
        {
            if(obj.empty())
            {
                cout<<"\n the expression is invalid";
                getch();
                exit(0);
            }
            obj.pop();
        }
        i++;
    }
    while(bracket[i]!='$');
    if(obj.empty())
    cout<<"\n the expression is invalid";
    else
    cout<<"\n the expression has well formed
    parenthesis";
}
getch();
}

```

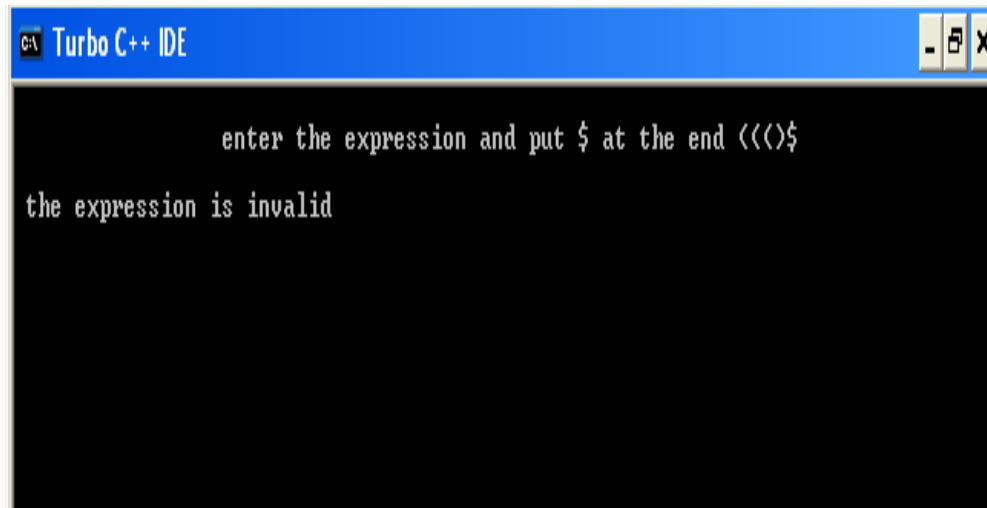
Step 3:

The above program will be executed to get output as follows

OUTPUT (RUN1)



The screenshot shows a Turbo C++ IDE window with a blue title bar. The main window has a black background with white text. The text displayed is: "enter the expression and put \$ at the end <<<>>>\$" followed by "the expression has well formed parenthesis".

OUTPUT (RUN 2)A screenshot of the Turbo C++ IDE window. The title bar is blue and contains the text "Turbo C++ IDE" and standard window control buttons (minimize, maximize, close). The main area is black with white text. The text displayed is: "enter the expression and put \$ at the end <<>\$" on the first line, and "the expression is invalid" on the second line.

```
enter the expression and put $ at the end <<>$  
the expression is invalid
```

RESULT:

Thus the C++ program for balanced parentheses checking using linked list implementation of stack ADT was created, executed and output was verified successfully.

EX. NO: 11 PROGRAM SOURCE FILES FOR STACK APPLICATION2**Application 2: Evaluation of postfix expression.****A.USING STACK (IMPLEMENTATION AS ARRAYS)**

STEP 1: create a header file named stack. In this file we declare the class and all the stack operations.

```

*****
stack.h
*****/
#define Max 10
class stk_class
{
/*declaration of stack data structure*/
    struct stack
    {
        double s[MAX];
        int top;
    }st;
public:
    stk_class();
    void push(double val);
    double pop();
};
stk_class::stk_class()
{
    st.top=0;
}
void stk_class::push(double val)
{
    if(st.top+1>=MAX)
        cout<<"\n stack is full";
    st.top++;
    st.s[st.top]=val;
}
double stk_class::pop()
{
    double val;
    if(st.top==0)
        cout<<"\n stack is empty\n";
    st.top--;
    return(val);
}

```

STEP 2: Program to evaluate a given postfix expression .her the stack using arrays is implemented in separate file named stack.h

```

#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
/*included the header file as stack using array*/
#define size 80
void main()
{
char exp[size];
int len;
double result;
double post(char exp[]);
clrscr();
cout<<"enter the postfix expression\n";
cin>>exp;
len=strlen(exp);
exp[len]='$';/*append $ at the end a end marker*/
result=post(exp);
cout<<"the value of the expression is"<<result;
getch();
exit(0);
}
double post(char exp[])
{
stk_class obj;
char ch,*type;
double result ,val,op1,op2;
int i;
i=0;
ch=exp[i];
while(ch!='$')
{
if(ch>='0' && ch<='9')
type="operand";
else if(ch=='+' || ch=='-' || ch=='^')
type="operator";
if(strcmp(type,"operand")==0)/* if the character is
operator*/
{
val=ch-48;
obj.push(val);
}
else
if(strcmp(type,"operator")==0)/* if it is operator*/
{
op2=obj.pop();
op1=obj.pop();
switch(ch)
{
case '+':result=op1+op2;
break;
case '-':result=op1-op2;

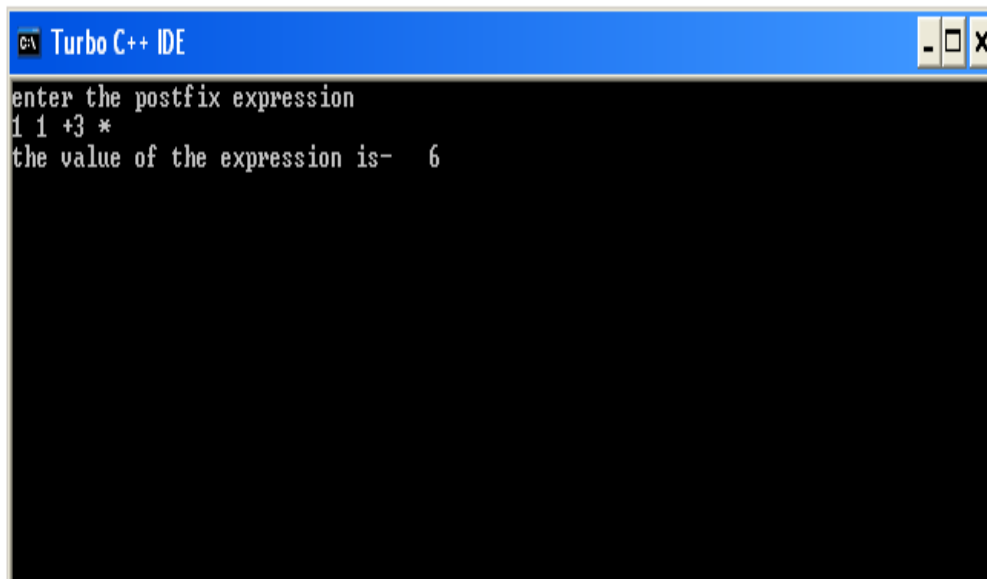
```

```

        break;
        case '*': result=op1*op2;
        break;
        case '/': result=op1/op2;
        break;
        case '^': result=pow(op1,op2)
        break;
    }/*switch*/
    obj.push(result);
}
i++;
ch=exp[i];
}/*while*/
result=obj.pop();/*pop the result*/
return(result);
}

```

OUTPUT:



RESULT:

Thus the given program Evaluation of postfix expression stack implemented as arrays was executed successfully.

B. STACK IMPLANTED AS LINKED LIST (USE OF SEPARATE HEADER FILE FOR STACK OPERATIONS)

STEP 1: create a header file named stack. In this file we declare the class and all the stack operations.

```

/*****
the stack.h file
*****/
class stk_class
{
/* data structure for the linked stack*/
typedef struct stack
{
    char data;
    struct stack *next;
}node;
public:
node *top;
stk_class();
void push(char item);
char pop();
};

stk_class::stk_class()
{
top=NULL;
}
/* functionality performed on linked linked stack*/
void stk_class::push(char item)
{
    node *New;
    New=new node;
    New->data =item;
    New->next=top;
    top=New;
}
char stk_class::pop()
{
    char item;
    node *temp;
    item=top->data;
    temp=top;
    top=top->next;
    delete temp;
    return item;
}

```

STEP 2: Program to evaluate a given postfix expression using linked stack. Here stack.h is a user defined header file created for linked stack

```

#include<iostream.h>
#include<stdlib.h>
#include<string.h>

```

```

#include<math.h>
#include<stdlib.h>
#include<conio.h>
/* included the stack.h file (given below) for linked stack*/
#include"C:\TC\INCLUDE\thystack.h "
#define size 80
void main()
{
    char exp[size];
    int len;
    double result;
    double post(char exp[])
    clrscr();
    cout<<"enter the postfix expression\n";
    cin>>exp;
    len=strlen(exp);
    cin>>exp;
    len=strlen(exp);
    exp[len]='$';/*append $at the end as a endmarker*/
    result=post(exp);
    cout<<"the value of the expression is"<<result;
    getch();
    exit(0);
}
double post(char exp[])
{
    char ch,*type;
    double result ,val,oip1,op2;
    int i;
    stk_class obj;
    i=0;
    ch=exp[i];
    while(ch!='$')
    {
        if(ch>='0' && ch<='9')
            type="operand";
        else if(ch=='+' || ch=='-' || ch=='*' || ch=='/' || ch=='^')
            type="operator";
        if(strcmp(type,"operator")==0) /* if the character
        is operand*/
        {
            val=ch-48;
            obj.push(val);
        }
        else
            if(strcmp(type,"operator")==0)/*if it is operator*/
            {
                op2=obj.pop();
                op1=obj.pop();
                switch(ch)
                {
                    case '+':result=op1+op2;
                    break;
                    case '-':result=op1-op2;
                    break;

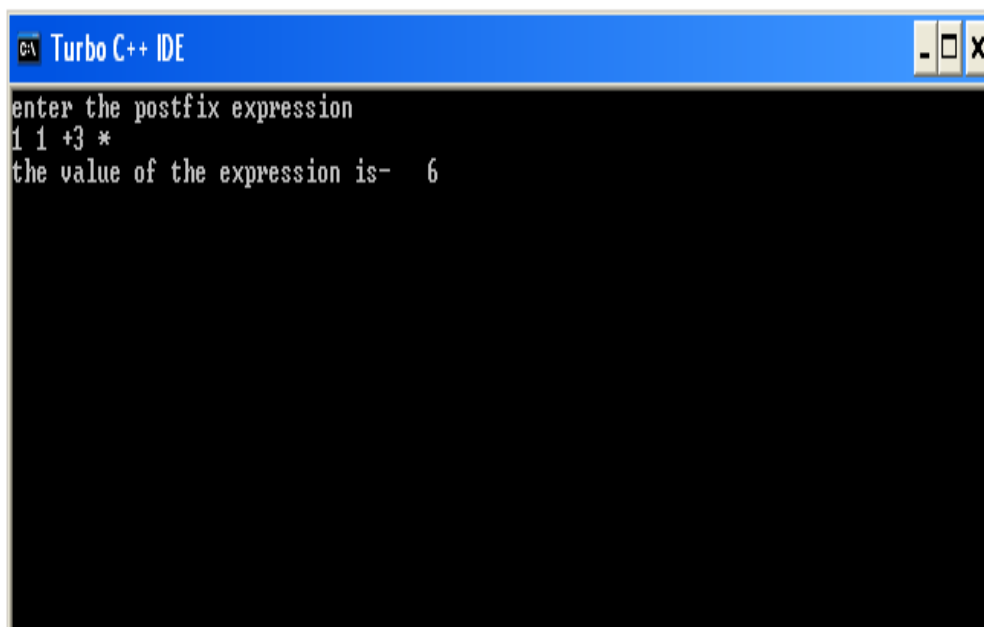
```

```

        case '*': result=op1*op2;
        break;
        case '/': result=op1/op2;
        break;
        case '^': result=pow(op1,op2)
        break;
    }/*switch*/
    obj.push(result);
}
i++;
ch=exp[i];
}/*while*/
result=obj.pop();/*pop the result*/
return(result);
}

```

OUTPUT:



```

Turbo C++ IDE
enter the postfix expression
1 1 +3 *
the value of the expression is- 6

```

RESULT:

Thus the given program Evaluation of postfix expression stack implemented as linked list was executed successfully.

EX. NO :12**QUEUE ADT USING LINKED LIST****AIM:**

To write a C++ program for Queue using Linked implementation.

DESCRIPTION:

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data(enqueue) and the other is used to remove data(dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues.

- enqueue () – add (store) an item to the queue.
- dequeue () – remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are

- peek() – Gets the element at the front of the queue without removing it.
- isfull() – Checks if the queue is full.
- isempty() – Checks if the queue is empty.

ALGORITHM:

1. Define a struct for each node in the queue. Each node in the queue contains data and link to the next node. Front and rear pointer points to first and last node inserted in the queue.
2. The operations on the queue are
 - a. INSERT data into the queue
 - b. DELETE data out of queue
3. INSERT DATA INTO queue
 - a. Enter the data to be inserted into queue.

- b. If TOP is NULL
 - i. The input data is the first node in queue.
 - ii. The link of the node is NULL.
 - iii. TOP points to that node.
 - c. If TOP is NOT NULL
 - i. The link of TOP points to the new node.
 - ii. TOP points to that node.
4. DELETE DATA FROM queue
- a. If TOP is NULL
 - i. the queue is empty
 - b. If TOP is NOT NULL
 - i. The link of TOP is the current TOP.
 - ii. The pervious TOP is popped from queue.
5. The queue represented by linked list is traversed to display its content.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class node
{
    public:
    class node *next;
    int data;
};
class queue : public node
{
    node *head;
    int front,rare;
    public:
    queue()
    {
        front=-1;
        rare=-1;
    }
    void enqueue(int x)
    {
        if (rare < 0 )
        {
            head =new node;
            head->next=NULL;
            head->data=x;
            rare ++;
        }
        else
        {
```

```

        node *temp,*temp1;
        temp=head;
        if(rare >= 4)
        {
            cout <<"queue over flow";
            return;
        }
        rare++;
        while(temp->next != NULL)
        temp=temp->next;
        temp1=new node;
        temp->next=temp1;
        temp1->next=NULL;
        temp1->data=x;
    }
}
void display()
{
    node *temp;
    temp=head;
    if (rare < 0)
    {
        cout <<" queue under flow";
        return;
    }
    while(temp != NULL)
    {
        cout <<temp->data<< " ";
        temp=temp->next;
    }
}
void dequeue()
{
    node *temp;
    temp=head;
    if( rare < 0)
    {
        cout <<"queue under flow";
        return;
    }
    if(front == rare)
    {
        front = rare =-1;
        head=NULL;
        return;
    }
    front++;
    head=head->next;
}
};
main()
{
    queue s1;
    int ch;
    clrscr();

```

```

cout<<"\n\n\tQUEUE USING LINKED LIST";
cout <<"\n1.enqueue\n2.dequeue\n3.DISPLAY\n4.EXIT";
while(1)
{
    cout<<"\n enter your choice:";
    cin >> ch;
    switch(ch)
    {
        case 1:
            cout <<"\n enter a element";
            cin >> ch;
            s1.enqueue(ch);
            break;
        case 2: s1.dequeue();
            break;
        case 3: s1.display();
            break;
        case 4: exit(0);
    }
}
return (0);
}

```

OUTPUT:

```

Turbo C++ IDE
QUEUE USING LINKED LIST
1.enqueue
2.dequeue
3.DISPLAY
4.EXIT
enter your choice:1
enter a element12
enter your choice:1
enter a element13
enter your choice:1
enter a element14
enter your choice:3
12 13 14
enter your choice:2
enter your choice:3
13 14
enter your choice:4

```

RESULT:

Thus the C++ program for queue ADT using linked list implementation was created, executed and output was verified successfully.

EX. NO: 13**QUEUE ADT USING ARRAY****AIM:**

To write a program for Queue using array implementation.

DESCRIPTION:

A queue data structure can be implemented using one dimensional array. But, queue implemented using array can store only fixed number of data values. The implementation of queue data structure using array is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using FIFO (First In First Out) principle with the help of variables 'front' and 'rear'. Initially both 'front' and 'rear' are set to -1. Whenever, we want to insert a new value into the queue, increment 'rear' value by one and then insert at that position. Whenever we want to delete a value from the queue, then increment 'front' value by one and then display the value at 'front' position as deleted element.

ALGORITHM:

1. Define a array which stores queue elements..
2. The operations on the queue are
 - a. a)INSERT data into the queue
 - b. b)DELETE data out of queue
3. INSERT DATA INTO queue
 - a. Enter the data to be inserted into queue.
 - b. If TOP is NULL
 - i. The input data is the first node in queue.
 - ii. The link of the node is NULL.
 - iii. TOP points to that node.
 - c. If TOP is NOT NULL
 - i. The link of TOP points to the new node.
 - ii. TOP points to that node.
4. DELETE DATA FROM queue
 - a. If TOP is NULL
 - i. the queue is empty
 - b. If TOP is NOT NULL
 - i. The link of TOP is the current TOP.
 - ii. The pervious TOP is popped from queue.

5. The queue represented by linked list is traversed to display its content.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class queue
{
    int queue1[5];
    int rear,front;
public:
    queue()
    {
        rear=-1;
        front=-1;
    }
    void insert(int x)
    {
        if(rear > 4)
        {
            cout <<"queue over flow";
            front=rear=-1;
            return;
        }
        queue1[++rear]=x;
        cout <<"inserted" <<x;
    }
    void delet()
    {
        if(front==rear)
        {
            cout <<"queue under flow";
            return;
        }
        cout <<"deleted" <<queue1[++front];
    }
    void display()
    {
        if(rear==front)
        {
            cout <<" queue empty";
            return;
        }
        for(int i=front+1;i<=rear;i++)
            cout <<queue1[i]<<" ";
    }
};
main()
{
    int ch;
    queue qu;
    cout <<"\n1.insert  2.delete  3.display  4.exit";
    while(1)
    {
```

```

        cout<<"\nEnter ur choice";
        cin >> ch;
        switch(ch)
        {
            case 1: cout <<"enter the element";
                    cin >> ch;
                    qu.insert(ch);
                    break;
            case 2:  qu.delet();
                    break;
            case 3:  qu.display();
                    break;
            case 4:  exit(0);
        }
    }
    return (0);
}

```

OUTPUT:

```

Turbo C++ IDE
1.insert 2.delete 3.display 4.exit
Enter ur choice1
enter the element123
inserted123
Enter ur choice1
enter the element23
inserted23
Enter ur choice1
enter the element456
inserted456
Enter ur choice3
123 23 456
Enter ur choice2
deleted123
Enter ur choice3
23 456
Enter ur choice4_

```

RESULT:

Thus the C++ program for queue ADT using array implementation was created, executed and output was verified successfully.

EX. NO: 14**BINARY SEARCH TREE****AIM:**

To write a C++ program for binary search tree.

DESCRIPTION:

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties

The left sub-tree of a node has a key less than or equal to its parent node's key.

The right sub-tree of a node has a key greater than to its parent node's key.

Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as

$$\text{left_subtree (keys)} \leq \text{node (key)} \leq \text{right_subtree (keys)}$$

Representation

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

Following is a pictorial representation of BST

Basic Operations

Following are the basic operations of a tree

- Search – Searches an element in a tree.
- Insert – Inserts an element in a tree.
- Pre-order Traversal – Traverses a tree in a pre-order manner.
- In-order Traversal – Traverses a tree in an in-order manner.
- Post-order Traversal – Traverses a tree in a post-order manner.

ALGORITHM:

1. Declare function create (), search (), delete (), Display ().
2. Create a structure for a tree contains left pointer and right pointer.
3. Insert an element is by checking the top node and the leaf node and the operation will be performed.
4. Deleting an element contains searching the tree and deleting the item.
5. Display the Tree elements.

PROGRAM:

```

#include<iostream.h>
#include<conio.h>
class bintree
{
    typedef struct bst
    {
        int data;
        struct bst *left,*right;
    }
    node;
    node *root,*New,*temp,*parent;
public:
    bintree()
    {
        root=NULL;
    }
    void create();
    void display();
    void delet();
    void find();
    void insert(node *,node*);
    void inorder(node *);
    void search(node**,int,node **);
    void del(node *,int);
};
void bintree::create()
{
    New=new node;
    New->left=NULL;
    New->right=NULL;
    cout<<"\n enter the element";
    cin>>New->data;
    if(root==NULL)
        root=New;
    else
        insert(root,New);
}
void bintree::insert(node *root,node *New)
{
    if(New->data<root->data)
    {
        if(root->left==NULL)
            root->left=New;
        else
            insert(root->left,New);
    }
    if(New->data>root->data)
    {
        if(root->right==NULL) root->right=New;
        else
            insert(root->right,New);
    }
}
}

```

```

void bintree::display()
{
    if(root==NULL)
        cout<<"tree is not created";
    else
    {
        cout<<"\n the tree is: ";
        inorder(root);
    }
}
void bintree::inorder(node *temp)
{
    if(temp!=NULL)
    {
        inorder(temp->left);
        cout<<" "<<temp->data;
        inorder(temp->right);
    }
}
void bintree::find()
{
    int key;
    cout <<"\n enter the element which you want to search";
    cin>>key;
    temp=root;
    search(&temp, key, &parent);
    if(temp==NULL)
        cout<<"\n element is not present";
    else
        cout<<"\n parent of node "<<temp->data<<" is"<<parent->data;
}
void bintree::search(node **temp,int key,node ** parent)
{
    if(*temp==NULL)
        cout<<endl<<" tree is not created"<<endl;
    else
    {
        while(*temp!=NULL)
        {
            if((*temp)->data==key)
            {
                cout<<"\n the "<<(*temp)->data<<" element
                is present";
                break;
            }
            *parent=*temp;//stores the parent value
            if((*temp)->data>key)
                *temp=(*temp)->left;
            else
                *temp=(*temp)->right;
        }
    }
    return;
}

```

```

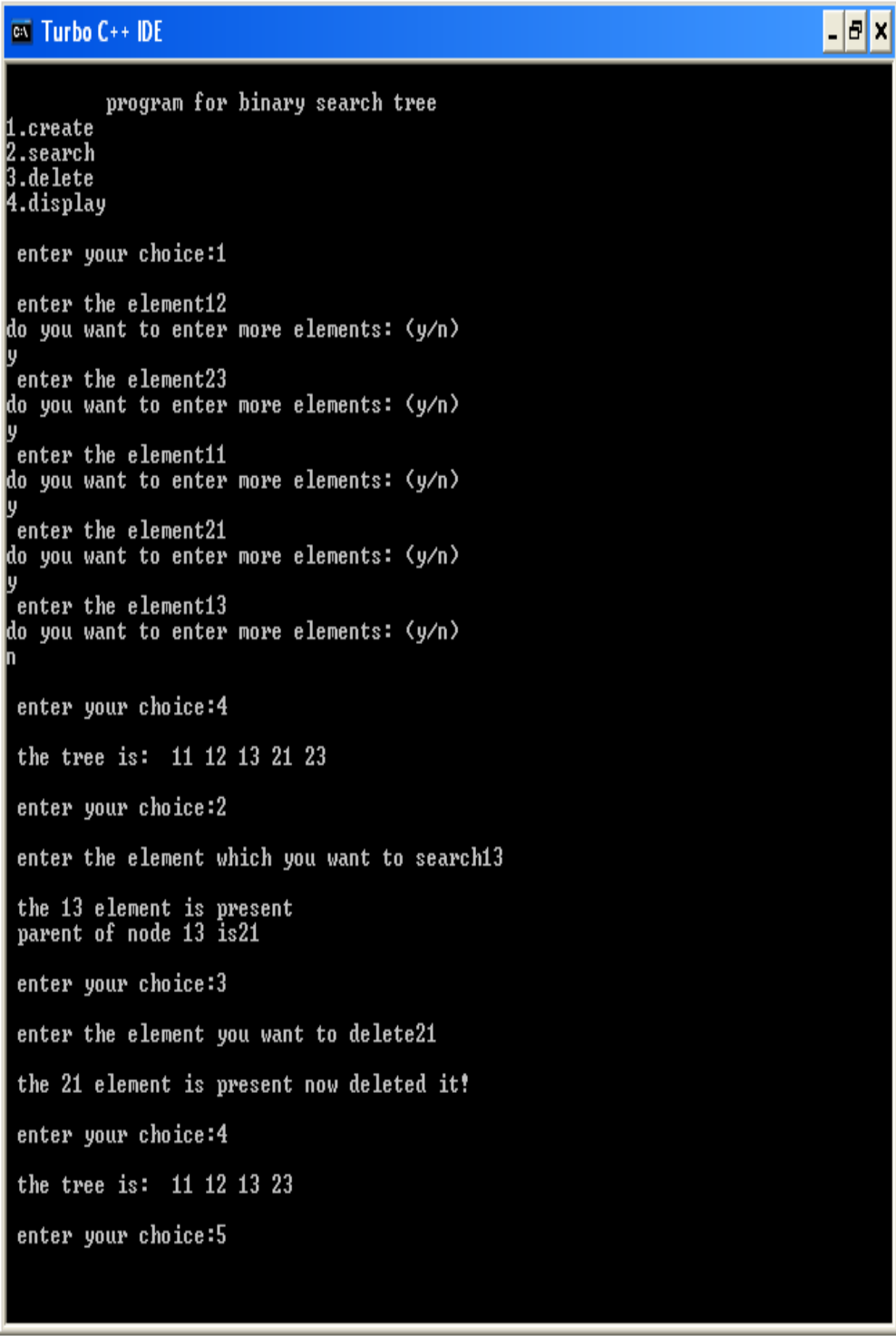
void bintree::delet()
{
    int key;
    cout<<"\n enter the element you want to delete";
    cin>>key;
    if(key==root->data)
    {
        bintree();// assigning a value NULL to root
    }
    else
        del(root, key); }
void bintree::del(node *root, int key)
{
    node *temp_succ;
    if(root==NULL)
        cout<<" tree is not created";
    else
    {
        temp=root;
        search(&temp, key, &parent);
        if (temp->left!=NULL&&temp->right!=NULL)
        {
            parent=temp;
            temp_succ=temp->left;
            while (temp_succ->left!=NULL)
            {
                parent=temp_succ;
                temp_succ=temp_succ->left;
            }
            temp->data=temp_succ->data;
            temp->right=NULL;
            cout<<" now deleted it!";
            return;
        }
        if (temp->left!=NULL&&temp->right==NULL)
        {
            if (parent->left==temp)
                parent->left=temp->left;
            else
                parent->right=temp->left;
            temp=NULL;
            delete temp;
            cout<<" now deleted it!";
            return;
        }
        if (temp->left==NULL&&temp->right!=NULL)
        {
            if (parent->left==temp)
                parent->left=temp->right;
            else
                parent->right=temp->right;
            temp=NULL;
            delete temp;
            cout<<" now deleted it!";
            return;
        }
    }
}

```

```

    }
    /*deleting a node which is having no child*/
    if(temp->left==NULL&&temp->right==NULL)
    {
        if(parent->left==temp)
            parent->left=NULL;
        else
            parent->right=NULL;
        cout<<" now deleted it!";
        return;
    }
}
}
void main()
{
    int choice;
    char ans='N';
    bintree tr;
    clrscr();
    cout<<"\n\t program for binary search tree";
    cout<<"\n1.create\n2.search\n3.delete\n4.display";
    do
    {
        cout<<"\n\n enter your choice:";
        cin>>choice;
        switch(choice)
        {
            case 1:do
            {
                tr.create();
                cout<<"do you want to enter more elements:
                (y/n)"<<endl;
                ans=getche();
            }
            while(ans=='y');
            break;
            case 2:tr.find();
            break;
            case 3:
            tr.delet();
            break;
            case 4:
            tr.display();
            break;
        }
    }
    while(choice!=5);
}

```

OUTPUT:


```

program for binary search tree
1.create
2.search
3.delete
4.display

enter your choice:1

enter the element12
do you want to enter more elements: (y/n)
y
enter the element23
do you want to enter more elements: (y/n)
y
enter the element11
do you want to enter more elements: (y/n)
y
enter the element21
do you want to enter more elements: (y/n)
y
enter the element13
do you want to enter more elements: (y/n)
n

enter your choice:4

the tree is: 11 12 13 21 23

enter your choice:2

enter the element which you want to search13

the 13 element is present
parent of node 13 is21

enter your choice:3

enter the element you want to delete21

the 21 element is present now deleted it!

enter your choice:4

the tree is: 11 12 13 23

enter your choice:5

```

RESULT:

Thus the C++ program for binary search tree was created, executed and output was verified successfully.

EX. NO: 15**HEAP SORT****AIM:**

To write a c program to perform heap sort.

DESCRIPTION:

Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If α has child node β then $\text{key}(\alpha) \geq \text{key}(\beta)$

As the value of parent is greater than that of child, this property generates Max Heap. Based on these criteria, a heap can be of two types.

Min-Heap – Where the value of the root node is less than or equal to either of its children.

Max-Heap – Where the value of the root node is greater than or equal to either of its children.

ALGORITHM:

1. Get the size of the array from the user.
2. Get the elements to be sorted.
3. Build a heap.
4. Sort the heap in ascending order.
5. Now the array is contained with sorted elements.
6. Display the sorted elements.

PROGRAM:

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    int a[50],size; int p,c;
    cout<<"Enter the Size of an Array :"; cin>>size;
    cout<<"Enter Value of A[1] :"; cin>>a[1];
    for(int i=2;i<=size;i++)
    {
        cout<<"Enter Value of A["<<i<<" ] :"; cin>>a[i];
        p=i/2;
        c=i;
        while(1)
        {
            if( a[c] > a[p])
            {
                int t=a[c]; a[c]=a[p]; a[p]=t;
            }
        }
    }
}
```

```

        c=p;
        p=p/2;
        if (p<1)
        {
            break;
        }
    }
}
cout<<endl<<"Heap ..."<<endl;
for(i=1;i<=size;i++)
{
    cout<<endl;
    cout<<"Arr["<<i<<" ] : "<<a[i];
}
int j=size;
int lc,rc;
while(j>1)
{
    if(a[1] > a[j])
    {
        int t=a[1];
        a[1]=a[j];
        a[j]=t;
        j--;
    }
    else
    {
        j--;
        continue;
    }
    p=1;
    while(p < j)
    {
        lc=p*2;
        rc=p*2 + 1;
        if(lc>=j || rc >=j)
        {
            break;
        }
        if(a[p] < a[lc] && a[lc] > a[rc])
        {
            int temp=a[lc];
            a[lc]=a[p];
            a[p]=temp;
            p=lc;
        }
        else if (a[p] < a[rc] && a[rc] > a[lc])
        {
            int temp=a[rc];
            a[rc]=a[p];
            a[p]=temp;
            p=rc;
        }
        else
        {

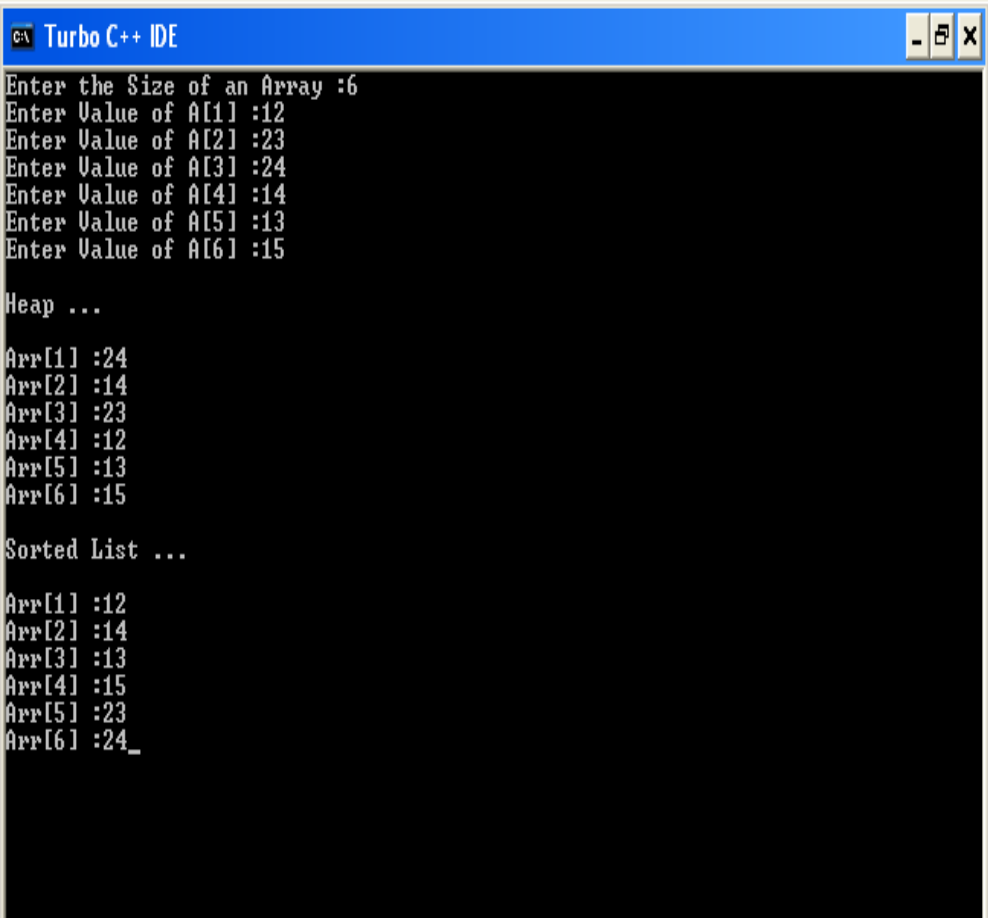
```

```

        break;
    }
}
cout<<endl<<"\nSorted List ..."<<endl;
for(i=1;i<=size;i++)
{
    cout<<endl;
    cout<<"Arr["<<i<<"] : "<<a[i];
}
getch();
}

```

OUTPUT:



```

Turbo C++ IDE
Enter the Size of an Array :6
Enter Value of A[1] :12
Enter Value of A[2] :23
Enter Value of A[3] :24
Enter Value of A[4] :14
Enter Value of A[5] :13
Enter Value of A[6] :15

Heap ...

Arr[1] :24
Arr[2] :14
Arr[3] :23
Arr[4] :12
Arr[5] :13
Arr[6] :15

Sorted List ...

Arr[1] :12
Arr[2] :14
Arr[3] :13
Arr[4] :15
Arr[5] :23
Arr[6] :24_

```

RESULT:

Thus the C++ program for heapsort was created, executed and output was verified successfully.

EX.NO: 16**QUICK SORT****AIM:**

To write a c program to perform quick sort.

DESCRIPTION:

Quick Sort, as the name suggests, sorts any list very quickly. Quick sort is not stable search, but it is very fast and requires very less additional space. It is based on the rule of Divide and Conquer (also called partition-exchange sort). This algorithm divides the list into three main parts:

- Elements less than the Pivot element
- Pivot element
- Elements greater than the pivot element

ALGORITHM:

1. Get the value of how many no. to be sorted.
2. Get the elements from the user.
3. Two function quicksort() and swap(). Quick sort to perform sorting.
4. Swap() is just to rearrange the values.
5. Quick sort algorithm works by partitioning the array to be sorted, then recursively sorting each partition.
6. Display the sorted value.

PROGRAM:

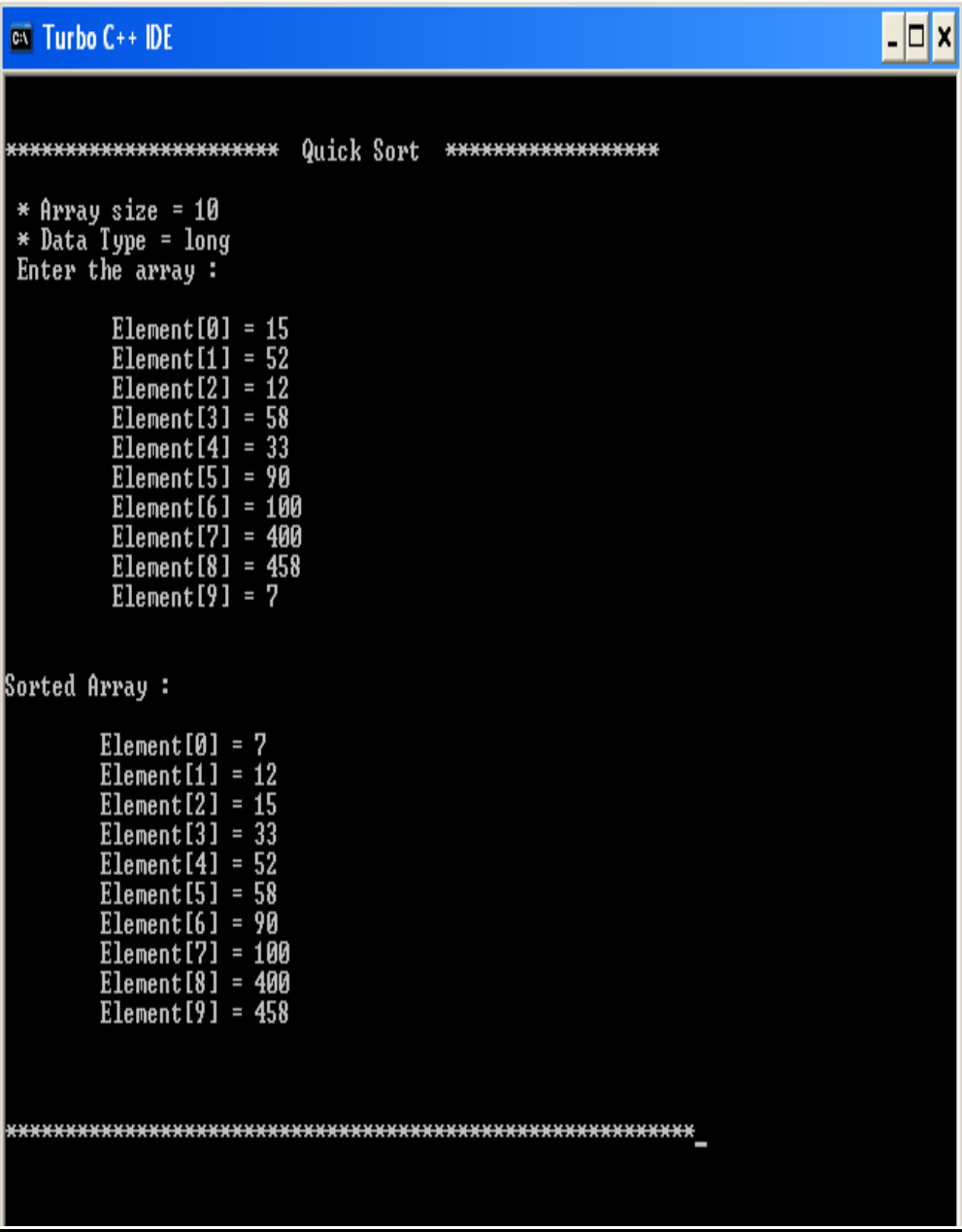
```
#include <iostream.h>
#include <conio.h>
void swap(long &, long &);
void quick_sort(long [], int, int);
main( )
{
    clrscr( );
    const int array_size=10;
    long array[array_size]={0};
    cout<<" \n\n***** Quick Sort
*****"<<endl;
    cout<<"\n * Array size = 10"<<endl;
    cout<<" * Data Type = long"<<endl;
    cout<<" Enter the array : "<<endl<<endl;
    for(int count_1=0;count_1<array_size;count_1++)
    {
        cout<<"\t Element["<<count_1<<"] = ";
        cin>>array[count_1];
    }
    quick_sort(array, 0, array_size-1);
    cout<<" \n\nSorted Array : \n\n";
    for(int count_2=0;count_2<array_size;count_2++)
```

```

    {
        cout<<"\tElement ["<<count_2<<" ] =
        "<<array[count_2]<<endl;
    }

    cout<<" \n\n\n*****";
    getch( );
    return 0;
}
void swap(long &element_1,long &element_2)
{
    long temp=element_1;
    element_1=element_2;
    element_2=temp;
}
void quick_sort(long array[],int first,int last)
{
    if(first>=last)
    {
    }
    else
    {
        int middle=array[last];
        int count_1=first-1;
        int count_2=last;
        while(count_1<count_2)
        {
            do
            {
                count_1++;
            }
            while(array[count_1]<middle);
            do
            {
                count_2--;
            }
            while(count_2>=0 && array[count_2]>middle);
            if(count_1<count_2)
                swap(array[count_1],array[count_2]);
        }
        swap(array[count_1],array[last]);
        quick_sort(array,first,count_1-1);
        quick_sort(array,count_1+1,last);
    }
}

```

OUTPUT:

```
***** Quick Sort *****

* Array size = 10
* Data Type = long
Enter the array :

Element[0] = 15
Element[1] = 52
Element[2] = 12
Element[3] = 58
Element[4] = 33
Element[5] = 90
Element[6] = 100
Element[7] = 400
Element[8] = 458
Element[9] = 7

Sorted Array :

Element[0] = 7
Element[1] = 12
Element[2] = 15
Element[3] = 33
Element[4] = 52
Element[5] = 58
Element[6] = 90
Element[7] = 100
Element[8] = 400
Element[9] = 458

*****
```

RESULT:

Thus the C++ program for quick sort was created, executed and output was verified successfully.