TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Lab Report of Data Structures and Algorithms (CT552)

Submitted By

Sushovan Shakya
THA075BEI046
2020/12/12

Submitted To

Er. Rama Bastola
Department of Electronics
and Computer Engineering

TABLE OF CONTENTS

6. Lab 6: Trees
    6.1 Create a BST of N integer
    6.2 Traverse the BST in In-order, Pre-order and Post-order
    6.3 Search the BST for a given element (key) and print appropriate message
7. Lab 7: Sorting
    7.1 Implementation of Insertion Sort Algorithm
    7.2 Implementation of Merge Sort Algorithm

LAB 1
Stack

1.1 WAP for array implementation of stack.

Problem Analysis

Stack works in the principle of LIFO (Last in First Out). There are mainly three methods for working with Stacks.

1. push() : for inserting a data to the top of the stack.
2. pop() : for popping out or deletes a data from the top of the stack.
3. peek() : for traversing the stack items without deleting the contents.

Here, we have to implement stack using arrays, which means we have to insert, delete and traverse the stack data using arrays following LIFO principle.

Algorithm

For pushing data into stack

```
Step 1: Initialize variable ‘top’, ‘size’
Step 2: For checking whether the stack is full
        if top == size – 1:
            print “Stack Overflow”
            Go to step 4
Step 3: For storing data
        Initialize variable ‘data’
        stack[top] = data
        ++top
Step 4: Exit
```

For popping data from the stack

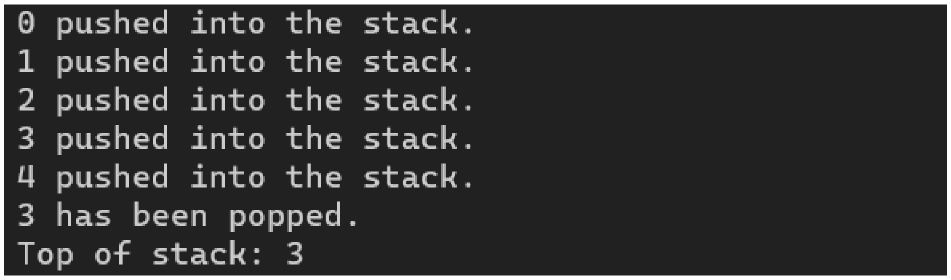
```
Step 1: Check whether the stack is empty (if top == -1)
Step 2: If it is empty:
    Print "Stack underflow"
    Go to step 3
    Else
        delete stack[top]
    --top
Step 3: Exit
```

Source code:

```
#include <iostream>

#define MAX 100

using namespace std;

class Stack {
	int top;
	int _stack_[MAX];

public:
	Stack() {
		top = -1;
	}

	bool isEmpty() {
		return (top == -1);
	}

	void push(int data) {
		if(top == MAX - 1) {
			cout << "Stack Overflow" << endl;
		}
		else {
			_stack_[++top] = data;
			cout << data << " pushed into the stack." << endl;
		}
	}

	void pop() {
		int pop_value;

		if(top < 0) {
			cout << "Stack underflow" << endl;
		}
```

```
        else {
            pop_value = _stack_[--top];
            cout << pop_value << " has been popped." << endl;
        }
    }

    void peek() {
        if(top < 0) {
            cout << "Stack underflow" << endl;
        }
        else {
            cout << "Top of stack: " << _stack_[top] << endl;
        }
    }
};

int main() {

    Stack stack_obj;

    // Pushing values to the stack
    for(int i = 0; i < 5; ++i) {
        stack_obj.push(i);
    }

    stack_obj.pop();
    stack_obj.peek();

    return 0;
}
```

Output



```
0 pushed into the stack.
1 pushed into the stack.
2 pushed into the stack.
3 pushed into the stack.
4 pushed into the stack.
3 has been popped.
Top of stack: 3
```

Conclusion

Stack data structure was thus implemented using array; however, this is only useful for storing a fixed size of data, and is not memory efficient. Thus, stack can be implemented using array.

1.2 WAP to reverse list using stack.

Problem Analysis

The problem here is to print the data of the stack in reverse. The stack implemented here is by using array, and since stack is a LIFO data structure, the data is popped from the top of the stack, which is in descending order. Thus, to print the data in reverse, data is shown from index 0, the first index of the stack.

Algorithm

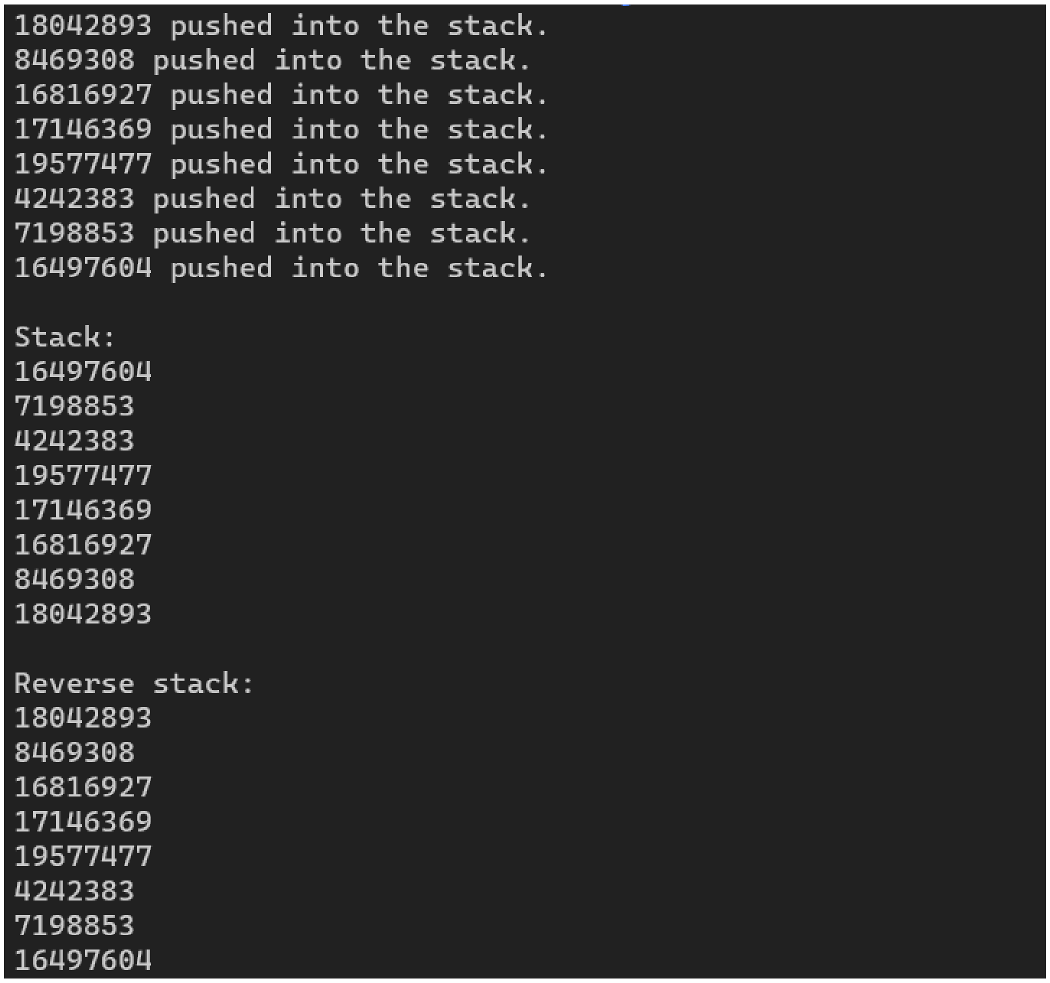
```
Step 1: Check if stack is empty.
        if top == -1:
                Go to step 4
Step 2: Initialize iterator 'i' with value 0
Step 3: while i <= top:
                Print stack[i]

Step 4: Stop
```

Source Code

```
#include <iostream>

#define MAX 100

using namespace std;

class Stack {
	int _stack_[MAX];
	int top;

public:
	Stack() { top = -1; }
	bool isEmpty { return (top == -1); }

	void push(int data) {
		if(top == MAX - 1) {
			cout << "Stack overflow" << endl;
		}
		else {
			_stack_[++top] = data;
			cout << "Pushed into the stack. " << endl;
```

```
                }
        }

        void pop() {
                int pop_value;

                if(top < 0) {
                        cout << "Stack underflow" << endl;
                }
                else {
                        pop_value = _stack_[--top];
                        return pop_value;
                }
        }

        void peek() {
                if(top < 0) {
                        cout << "Stack underflow" << endl;
                }
                else {
                        return _stack_[top];
                }
        }

    void display() {
        cout << "Stack: " << endl;

        for(int i = top; i > -1; --i)
        {
           cout << _stack_[i] << endl;
        }
    }

    void reverse()
    {
        cout << "Reverse stack: " << endl;

        for (int i = 0; i <= top; i++)
        {
            cout << _stack_[i] << endl;
        }
    }
}

int main() {
        Stack stack_obj;
        int rand_int;

        for(int i = 0; i <=7; ++i) {
                rand_int = rand() / 7;
                stack_obj.push(rand_int);
        }

        stack_obj.display();
        stack_obj.reverse();

        return 0;
}
```

Output



```
18042893 pushed into the stack.
8469308 pushed into the stack.
16816927 pushed into the stack.
17146369 pushed into the stack.
19577477 pushed into the stack.
4242383 pushed into the stack.
7198853 pushed into the stack.
16497604 pushed into the stack.

Stack:
16497604
7198853
4242383
19577477
17146369
16816927
8469308
18042893

Reverse stack:
18042893
8469308
16816927
17146369
19577477
4242383
7198853
16497604
```

Conclusion

Thus, we can use stack, implemented using array, and perform operations like reversing the stack.

1.3 WAP to check parentheses of algebraic expression using stack.

Problem Analysis

The problem here is to check if an expression has all the parentheses balanced. To check if the expression is balanced, the expression is traversed through. If the current character is a starting bracket, it is pushed into the stack. If the current character is a closing bracket, it is popped from the stack. If the popped character is the matching starting bracket then it is balanced else the expression is not balanced.

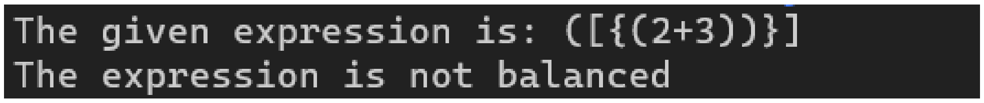
Algorithm

```
Step 1: Initialize a character stack
Step 2: Traverse through the expression.
        If character = '[' or '{' or '(':
                Push to the stack.
        Else if character = '[' or '}' or ')':
                Pop from the stack
        [END OF IF]
Step 3: If popped_char = matching_bracket:
                Print "Balanced
          Else
                Print "Unbalanced"
Step 4: EXIT
```

Source Code

```cpp
#include <iostream>
#include <stack>

using namespace std;

bool isBalanced(string exp) {
    stack <char> st;
    char c;

    // Traversing through the Expression
    for(int i=0; i< exp.length(); i++) {
        // Checking the input char
        if(exp[i] == '[' || exp[i] == '{'|| exp[i] == '(') {
            // Push the char in the stack
            st.push(exp[i]);
            continue;
        }
        else {
            continue;
        }

        // If the char is not the opening bracket
        // // it must be closing bracket, so stack is not empty
        // if(st.empty())
        //  return true;

        switch(exp[i]) {

            case ')': {
                c = st.top();
                st.pop();
                if(c == '{' || c == '[')
                    return false;

                break;
            }
            case '}': {
                c = st.top();
                st.pop();
                if(c == '(' || c == '[')
                    return false;
                break;
            }
            case ']': {
                c = st.top();
                st.top();
                if(c == '(' || c == '{')
                    return false;
                break;
            }

        }

    }

    // // check the status of the stack
    // if (st.empty())
        return true;
}

int main() {

    string exp = "([{(2+3))}]";

    // Checking the value return by the function
    if(isBalanced(exp))
        cout << "The expression is balanced" << endl;
    else if(!isBalanced(exp))
        cout << "The expression is not balanced" << endl;
    else
        cout << "I don't know";
    return 0;
}
```

Output



```
The given expression is: ([{(2+3))}]
The expression is not balanced
```

Conclusion

In this way, we can use the stack data structure to analyze an algebraic expression and check if it is a balanced expression or not.

1.4 WAP to convert infix to postfix using stack.

Problem Analysis

To convert infix to postfix, the given expression is first traversed and checked if the current char is an operand or an operator. If the character is an operand, the postfix expression is appended and the char is added. If the char is an opening bracket, it is pushed into the stack and if it is a closing parenthesis, it is popped from the stack. If the char is an operator, it is pushed to the top of the stack following the precedence of the operator.

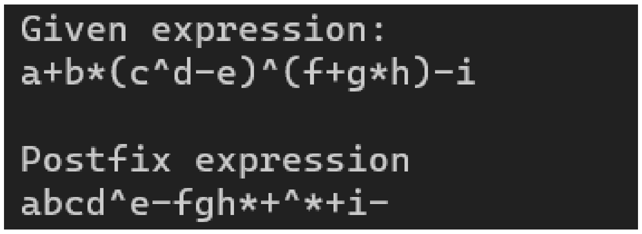
Algorithm

```
Step 1: Traverse through the expression
        If top = NULL:
            Print “Stack is empty”
Step 2: If character = operand:
                postfix = operand
            Else if character = operator:
                   If operator1.precedence > operator2.precedence:
                       push()
            Else if character = ‘)’:
                       pop()
Step 3:
          If char = ‘(’:
              push()
          Else if char = ‘)’:
                 While char != ‘(’:
          pop()
Step 4: Exit
```

Source Code

```
#include <iostream>
#include <stack>
#include <string>

// Function to return precedence of operators
int prec(char c)
{
       switch (c) {
              case '^':
                     return 3;
              case '*':
                     return 2;
              case '/':
                     return 2;
              case '+':
                     return 1;
              case '-':
                     return 1:
              default:
                     return -1;
       }
}

void infixToPostfix(string s)
{
    std::stack<char> st;
    st.push('N');
    int l = s.length();
    std::string ns;

    for(int i = 0; i < l; i++)
    {
        // If the scanned character is an operand, add it to output string.
        if((s[i] >= 'a' && s[i] <= 'z')||(s[i] >= 'A' && s[i] <= 'Z'))
        ns+=s[i];

        // If the scanned character is an Ô(Ô, push it to the stack.
        else if(s[i] == '(')

        st.push('(');

        // If the scanned character is an Ô)Õ, pop and to output string from the stack
        // until an Ô(Ô is encountered.
        else if(s[i] == ')')
        {
            while(st.top() != 'N' && st.top() != '(')
            {
                char c = st.top();
                st.pop();
               ns += c;
            }
            if(st.top() == '(')
            {
                char c = st.top();
                st.pop();
            }
        }

        //If an operator is scanned
        else{
            while(st.top() != 'N' && prec(s[i]) <= prec(st.top()))
            {
                char c = st.top();
                st.pop();
                ns += c;
            }
            st.push(s[i]);
        }

    }
    //Pop all the remaining elements from the stack
    while(st.top() != 'N')
    {
        char c = st.top();
        st.pop();
        ns += c;
    }

    std::cout << ns << std::endl;

}

//Driver program to test above functions
int main()
{
    std::string exp = "a+b*(c^d-e)^(f+g*h)-i";
    infixToPostfix(exp);
    return 0;
}
```

Output



```
Given expression:
a+b*(c^d-e)^(f+g*h)-i

Postfix expression
abcd^e-fgh*+^*+i-
```

Conclusion

In this way, we can apply the stack data structure to convert an infix algebraic expression into postfix expression.

1.5 WAP to convert infix to prefix using stack.

Problem Analysis

For converting infix to prefix, the infix expression is first converted into a postfix expression, and the obtained postfix expression is reversed to obtain the prefix expression.

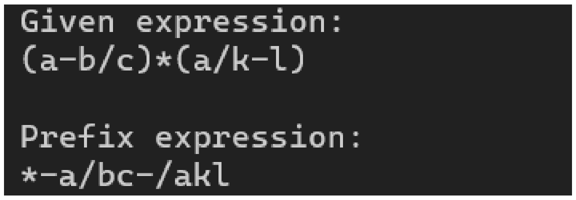
Algorithm

```
Step 1: Traverse through the expression
        If top = NULL:
            Print “Stack is empty”
Step 2: If character = operand:
                postfix = operand
            Else if character = operator:
                  If operator1.precedence > operator2.precedence:
                      push()
            Else if character = ‘)’:
                      pop()
Step 3:
          If char = ‘(’:
              push()
          Else if char = ‘)’:
                While char != ‘(’:
          pop()
Step 4:
         reverse(postfix.begin(), postfix.end())
         print result
Step 5: Exit
```

Source Code

```
#include <iostream>
#include <stack>
#include <string>
#include <algorithm>

bool isOperator(char c)
{
    return (!isalpha(c) && !isdigit(c));
}

int getPriority(char C)
{
    if (C == '-' || C == '+')
        return 1;
    else if (C == '*' || C == '/')
        return 2;
    else if (C == '^')
        return 3;
    return 0;
}

std::string infixToPostfix(std::string infix)
{
    infix = '(' + infix + ')';
    int l = infix.size();
    std::stack<char> char_stack;
    std::string output;

    for (int i = 0; i < l; i++) {

        // If the scanned character is an
        // operand, add it to output.
        if (isalpha(infix[i]) || isdigit(infix[i]))
            output += infix[i];

        // If the scanned character is an
        // '(', push it to the stack.
        else if (infix[i] == '(')
            char_stack.push('(');

        // If the scanned character is an
        // ')', pop and output from the stack
        // until an '(' is encountered.
        else if (infix[i] == ')') {

            while (char_stack.top() != '(') {
                output += char_stack.top();
                char_stack.pop();
            }

            // Remove '(' from the stack
            char_stack.pop();
        }

        // Operator found
        else {
            if (isOperator(char_stack.top())) {
                while (getPriority(infix[i])
                   <= getPriority(char_stack.top())) {
                    output += char_stack.top();
                    char_stack.pop();
                }

                // Push current Operator on stack
                char_stack.push(infix[i]);
            }
        }
    }
    return output;
}

std::string infixToPrefix(std::string infix)
{
    /* Reverse String
     * Replace ( with ) and vice versa
     * Get Postfix
     * Reverse Postfix  *  */
    int l = infix.size();

    // Reverse infix
    std::reverse(infix.begin(), infix.end());

    // Replace ( with ) and vice versa
    for (int i = 0; i < l; i++) {

        if (infix[i] == '(') {
            infix[i] = ')';
            i++;
        }
        else if (infix[i] == ')') {
            infix[i] = '(';
            i++;
        }
    }

    std::string prefix = infixToPostfix(infix);

    // Reverse postfix
    std::reverse(prefix.begin(), prefix.end());

    return prefix;
}

// Driver code
int main()
{
    std::string s = ("(a-b/c)*(a/k-l)");
       std::cout << "Given expression: " << std::endl << s << std::endl << std::endl;
    std::cout << "Prefix expression: " << std::endl << infixToPrefix(s) << std::endl;
    return 0;
}
```

Output



```
Given expression:
(a-b/c)*(a/k-l)

Prefix expression:
*-a/bc-/akl
```

Conclusion

Similar to the previous problem, stack can be applied to analyze an algebraic expression and convert an infix expression into prefix expression.

LAB 2
Queue

2.1 WAP for array implementation of linear queue

Problem Analysis

Queue is a data structure following First In, First Out (FIFO) principle. Here, we use array to implement a linear queue, where we insert, delete and traverse through the data structure. The data is inserted from the rear, a process called enqueue, while the data is retrieved from the first, a process called dequeue.

Algorithm

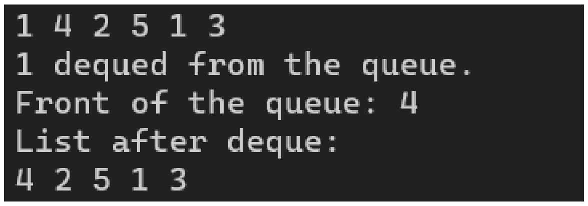
```
Step 1: Enqueue
        if rear == MAX – 1:
                print “Queue overflow”
Step 2:
        if front == -1 and rear == -1:
                front = rear = -1
                queue[rear] = -1
        else:
                rear += 1
                queue[rear] = data
        [END IF]
Step 3: Dequeue
        if rear = -1:
                print “Queue Empty”
        else if front == 0 and rear == 0:
                delete queue[front]
                front = rear = -1
        else:
                delete queue[front]
                front += 1
Step 4: Traversing the queue
        while index <= rear:
                print queue[index]
Step 5: Exit
```

Source Code

```
#include <iostream>
using namespace std;

int max_size = 100;

class Queue {
	int front, rear;
	int *queue;

public:
	Queue() {
		front = 0;
		rear = 0;
		queue = new int[max_size];
	}

	bool isEmpty() {
		return(front == rear);
	}

	bool isFull() {
		return(rear == max_size - 1);
	}

	// Enque method
	void enque(int data) {
		if(isFull()) {
			cout << "Overflow" << endl;
			return;
		}
		else{
			queue[rear] = data;
			++rear;
		}
	}

	// Deque method
	void deque() {
		int deque_value = 0;

		if(isEmpty()) {
			cout << "Underflow" << endl;
		}
		else {
			deque_value = queue[front];

			for(int i = 0; i < rear - 1; ++i) {
				queue[i] = queue[i+1];
			}

			--rear;
			cout << deque_value << " dequed from the queue." << endl;
		}
	}
```

```cpp
        // For returning front of the queue
        void frontOf() {
               if(isEmpty()) {
                      cout << "Queue empty." << endl;
                      return;
               }
               else {
                      cout << "Front of the queue: " << queue[front] << endl;
               }
        }

        void display() {
               if(isEmpty()) {
                      cout << "Queue empty." << endl;
                      return;
               }
               else {
                      for(int i = front; i < rear; ++i) {
                             cout << queue[i] << " ";
                      }
               }
        }

        ~Queue() {
               delete[] queue;
        }
};

int main() {
        Queue *queue_obj = new Queue();

        for(int i = 0; i < 6; ++i) {
               queue_obj->enque(rand() % 7);
        }

        queue_obj->display();

        cout << endl;
        queue_obj->deque();
        queue_obj->frontOf();
        cout << "List after deque: " << endl;
        queue_obj->display();
        cout << endl;

        delete queue_obj;
        return 0;
}
```

Output



```
1 4 2 5 1 3
1 dequed from the queue.
Front of the queue: 4
List after deque:
4 2 5 1 3
```

Conclusion

Similar to implementing stack, queue data structure can be implemented through the use of arrays. However, this data structure, like stack from array, is not memory efficient, and is only useful for storing fixed size of data.

2.2 WAP for implementation of Circular Queue

Problem Analysis

For creating a circular queue, the last position of the queue is connected to the first position of the queue, creating a circle of data, hence efficiently managing the memory and implementing a circular queue.

Algorithm

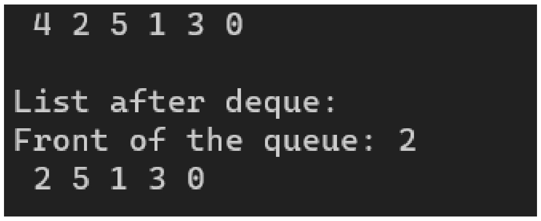
```
Step 1: For insertion
        if (front == 0 and rear == size – 1) or rear == (front – 1) % (size – 1):
            print “Queue full”
Step 2:
        if front == -1:
            front = rear = 0
            queue[rear] = data
        else:
            rear += 1
            queue[rear] = data
Step 3: For deleting data
        if front == -1:
            print “Queue empty”
Step 4:
        delete queue[front]
        queue[front] = -1
Step 5:
        if front == rear:
            front = rear = -1
        else if front == size – 1:
            front = 0
        else:
            front += 1
Step 6: Traversing data
        if rear >= front:
            while index <= rear:
                print queue[index]
Step 7: Exit
```

Source Code

```
#include <iostream>

using namespace std;

int max_size = 100;

class Queue {
	int rear, front;
	int *queue;
	int size;

public:
	Queue(int size) {
		front = -1;
		rear = -1;
		this->size = size;

		queue = new int[this->size];
	}

	bool isEmpty() { return(front == -1); }

	void enque(int data) {

		// We know that the next element in a circular queue is given by (rear + 1)
% size
		if(((rear+1) % size) == front) {
			cout << "Queue full" << endl;
			return;
		}
		else if (front == -1) {
			front = rear = 0;
			queue[rear] = data;
		}
		else {
			queue[rear] = data;
			rear = (rear + 1) % size;
		}
	}

	void deque() {
		int deque_item = 0;

		if(isEmpty()) {
			cout << "Empty queue" << endl;
		}
		else if (front == rear) {
			deque_item = queue[front];
			front = rear = -1;
		}
		else {
			deque_item = queue[front];
			front = (front + 1) % size;
		}
	}
```

```cpp
        void frontOf() {
                if(isEmpty()) {
                        cout << "Queue empty." << endl;
                        return;
                }
                else {
                        cout << "Front of the queue: " << queue[front] << endl;
                }
        }

        void display() {
                if(isEmpty()) {
                        cout << "Queue empty." << endl;
                        return;
                }
                else if(front <= rear) {
                        for(int i = front; i <= rear; ++i) {
                                cout << " " << queue[i];
                        }
                }
                else {
                        for(int i = front; i < size; ++i) {
                                cout << " " << queue[i];
                        }
                        for(int i = 0; i <= rear; ++i) {
                                cout << " " << queue[i];
                        }
                }
                cout << endl;
        }

        ~Queue() {
                delete[] queue;
        }
};

int main() {

        Queue *queue_obj = new Queue(max_size);

        for(int i = 0; i < 6; ++i) {
                queue_obj->enque(rand() % 7);
        }

        queue_obj->display();
        cout << endl;

        cout << "List after deque: " << endl;
        queue_obj->deque();
        queue_obj->frontOf();
        queue_obj->display();
        cout << endl;

        delete queue_obj;

        return 0;
}
```

Output



```
 4 2 5 1 3 0

List after deque:
Front of the queue: 2
 2 5 1 3 0
```

Conclusion

In this way, we can implement circular queue data structure through the use of arrays. However, unlike a linear queue, this is more memory efficient, but only can only store a fixed size of data.

LAB 3
List

3.1 WAP to create contiguous list using array

Problem Analysis

List is a collection of nodes. A pointer to a node is represented by array index whose value lies between 0 and max-1. Null pointer is represented by -1. There must be separate function to get the available nodes and free the nodes.

Algorithm

For getting a Node

```
Step 1. If avail = NULL
            Write overflow
            Go to step 5
Step 2. Set pointer ptr = avail
Step 3. Set avail = node[avail].next
Step 4. Return ptr
Step 5. Exit
```

For freeing a node

```
Step 1. Input a pointer ptr
Step 2. Set node[ptr].next = avail
Step 3. Set avail = ptr
Step 4. Stop
```

Delete node

```
Step 1. Input ptr
Step 2. node[ptr].info = 0
Step 3. If ptr = null or ptr > size – 1
            print “Invalid node”
        Goto step 6
Step 4. node[ptr-1].next = node[ptr].next
Step 5. Free node ptr
Step 6. Exit
```

Insert after a pointer

```
Step 1. Input a value val and pointer ptr
Step 2. if ptr = null
    Write “Invalid insertion”
    Goto step 7
Step 3. newptr = available node
Step 4. node[newptr].info = val
Step 5. node[newptr].next = node[ptr].next
Step 6. node[ptr].next = newptr
Step 7. Exit
```

Delete after a pointer

```
Step 1. Input a pointer ptr
Step 2. If ptr = null or node[ptr].next = null
        print “Invalid deletion”
        Goto step 7
Step 3. delptr = node[ptr].next
Step 4. delval = node[delptr].info
Step 5. node[ptr].next = node[delptr].next
Step 6. free node delptr
Step 7. Exit
```

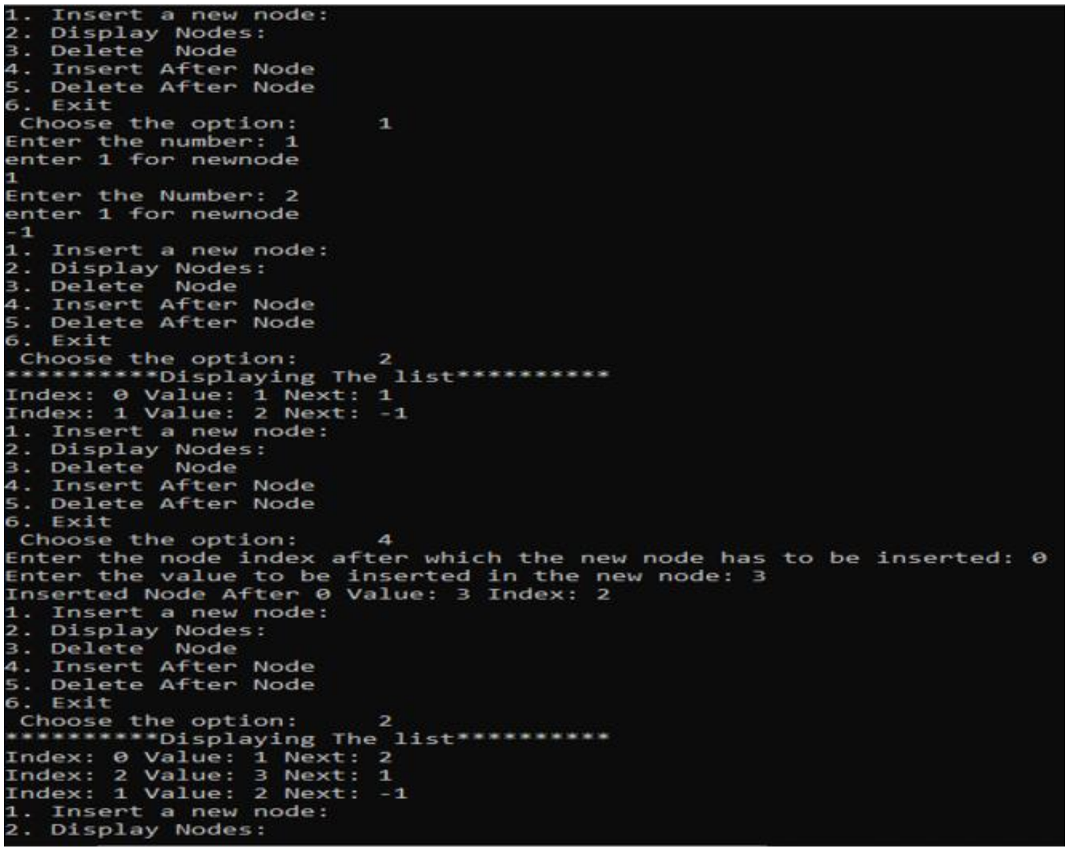
Source Code

```
#include<iostream>
#define max 15
using namespace std;
struct nodetype
{
    int info, next;
};
class list
{
    struct nodetype node[max];
    int avail = 0;
public:
    int intialize_availlist()
    {
        int i;
        for (i = 0;i < max - 1;i++)
        {
            node[i].next = i + 1;
        }
        node[max - 1].next = -1;
    }
    int get_node()
    {
        int p;
        if (avail == -1)
        {
            cout << "Overflow";
        }
        p = avail;
        avail = node[avail].next;
        return p;
    }
    int freenode(int p)
    {
        node[p].next = avail;
        avail = p;
    }
    void insertnode(int& list1)
    {
        int val, ptr, curptr, newnode = 1;
        while (newnode == 1)
        {
            if (list1 == -1)
            {
                ptr = get_node();
                list1 = ptr;
                cout << "Enter the number: ";
                cin >> val;
                node[ptr].info = val;
                node[ptr].next = -1;
            }
            else
            {
                curptr = 0;
                while (node[curptr].next != -1)
                {
                    curptr = node[curptr].next;
                }
                ptr = get_node();
                cout << "Enter the Number: ";
                cin >> val;
                node[curptr].next = ptr;
                node[ptr].info = val;
                node[ptr].next = -1;
            }
            cout << "enter 1 for newnode" << endl;
            cin >> newnode;
        }
    }
    int displaynode()
    {
        cout << "**********Displaying The list**********" << endl;
        int i;
        int ptr = 0;
        if (avail == 0)
        {
            cout << "List Underflow" << endl;
        }
        while (ptr != -1)
        {
            cout << "Index: " << ptr << " Value: " << node[ptr].info << " Next: " <<
node[ptr].next << endl;
            ptr = node[ptr].next;
        }
    }
    int deletenode(int& list1)
    {
        int val, curptr, preptr = -1;
        curptr = list1;
        while (node[curptr].next != -1)
        {
            preptr = curptr;
            curptr = node[curptr].next;
        }
        freenode(curptr);
        cout << endl;
        cout << "The deleted value is: " << node[curptr].info << endl;
        if (preptr == -1)
        {
            list1 = -1;
        }
        else
        {
            node[preptr].next = -1;
        }
    }
    int insert_after(int ptr, int val)
    {
        int newptr;
        if (ptr == -1)
        {
            cout << "Invalid Insertion";
```

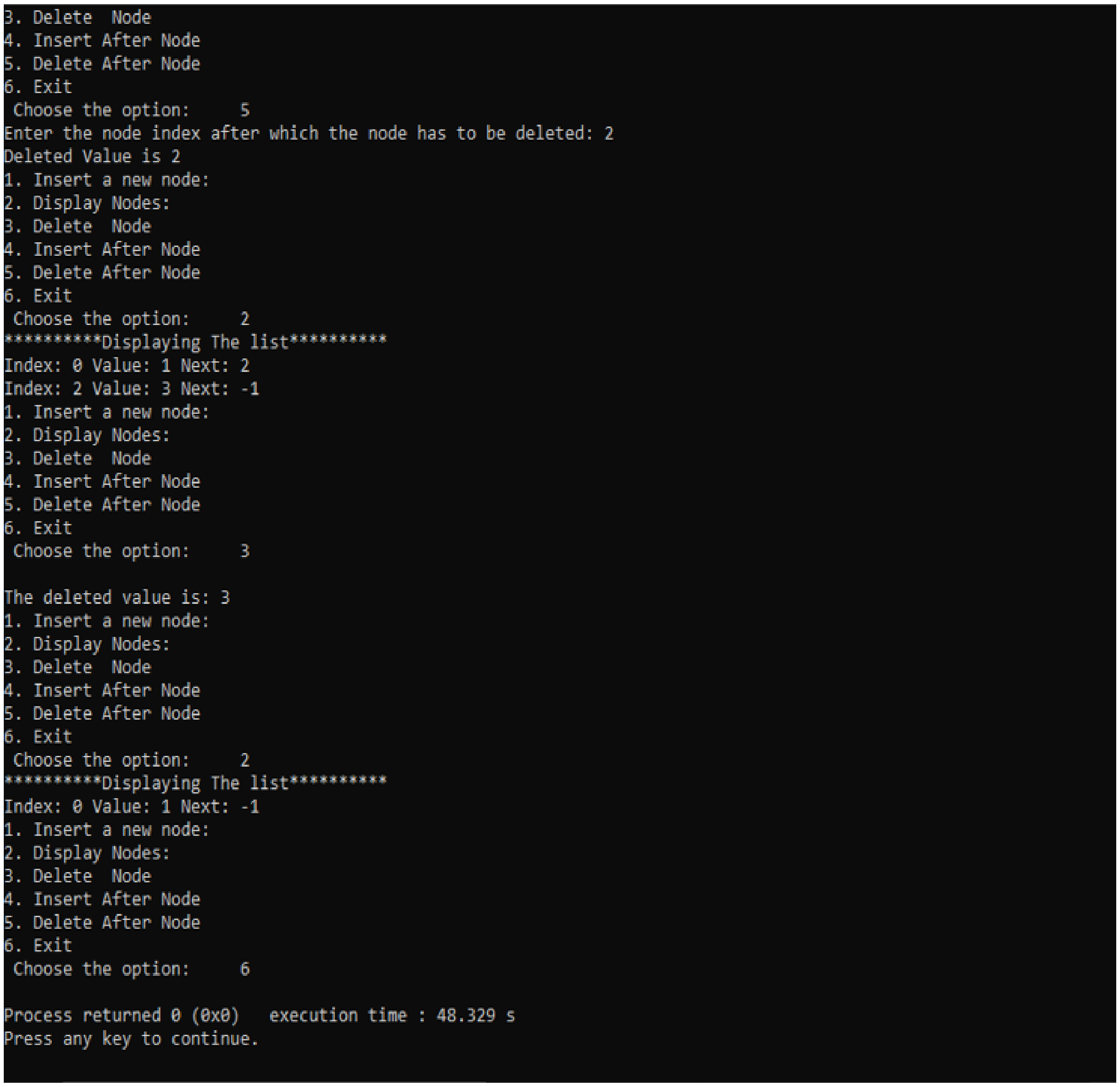
```cpp
        }
        else
        {
            newptr = get_node();
            node[newptr].info = val;
            node[newptr].next = node[ptr].next;
            node[ptr].next = newptr;
            cout << "Inserted Node After " << ptr << " Value: " << val << " Index: " <<
newptr << endl;
        }
    }
    int delete_after(int ptr)
    {
        int delptr, delval;
        if (ptr == -1 || node[ptr].next == -1)
        {
            cout << "Invalid deletion after given ptr" << endl;
        }
        else {
            delptr = node[ptr].next;
            delval = node[delptr].info;
            cout << "Deleted Value is " << delval << endl;
            node[ptr].next = node[delptr].next;
            freenode(delptr);
        }
    }
};
int main()
{
    list l;
    l.intialize_availlist();
    int ch;
    int list1 = -1;
    do
    {
        cout << "1. Insert a new node: " << endl;
        cout << "2. Display Nodes: " << endl;
        cout << "3. Delete  Node" << endl;
        cout << "4. Insert After Node" << endl;
        cout << "5. Delete After Node" << endl;
        cout << "6. Exit" << endl;
        cout << " Choose the option: \t";
        cin >> ch;
        switch (ch)
        {
        case 1:
            l.insertnode(list1);
            break;
        case 2:
            l.displaynode();
            break;
        case 3:
            l.deletenode(list1);
            break;
        case 4:
        {
            int ptr, val;
            cout << "Enter the node index after which the new node has to be inserted: ";
```

```
                cin >> ptr;
                cout << "Enter the value to be inserted in the new node: ";
                cin >> val;
                l.insert_after(ptr, val);
                break;
            }
            case 5:
            {
                int ptr;
                cout << "Enter the node index after which the node has to be deleted: ";
                cin >> ptr;
                l.delete_after(ptr);
                break;
            }
            case 6: break;
            default: cout << "invalid input" << endl;
            }
        } while (ch != 6);
    }
```

Output



```
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:      1
Enter the number: 1
enter 1 for newnode
1
Enter the Number: 2
enter 1 for newnode
-1
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:      2
**********Displaying The list**********
Index: 0 Value: 1 Next: 1
Index: 1 Value: 2 Next: -1
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:      4
Enter the node index after which the new node has to be inserted: 0
Enter the value to be inserted in the new node: 3
Inserted Node After 0 Value: 3 Index: 2
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:      2
**********Displaying The list**********
Index: 0 Value: 1 Next: 2
Index: 2 Value: 3 Next: 1
Index: 1 Value: 2 Next: -1
1. Insert a new node:
2. Display Nodes:
```



```
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:     5
Enter the node index after which the node has to be deleted: 2
Deleted Value is 2
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:     2
**********Displaying The list**********
Index: 0 Value: 1 Next: 2
Index: 2 Value: 3 Next: -1
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:     3

The deleted value is: 3
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:     2
**********Displaying The list**********
Index: 0 Value: 1 Next: -1
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Insert After Node
5. Delete After Node
6. Exit
 Choose the option:     6

Process returned 0 (0x0)   execution time : 48.329 s
Press any key to continue.
```

Conclusion

Thus, we can implement linked list data structure through the use of array, to store contiguous value. However, this list is not dynamic in nature, and is less memory efficient, since it uses arrays, and can only store a fixed size of data.

3.2 WAP to implement Queue using List

Problem Analysis

Queue can be implemented by using static list structure. Initially front and rear nodes can be made null. Operations like enqueue and dequeue can be implemented by getting the available node and making the node free as required.

Algorithm

Enqueue a data

```
Step 1: Initialize a variable for inputting data
Step 2: ptr = available node
Step 3: node[ptr].data = val
Step 4: node[ptr].next = null
Step 5: if rear == null:
                front = ptr
        else:
                node[rear].next = ptr
```

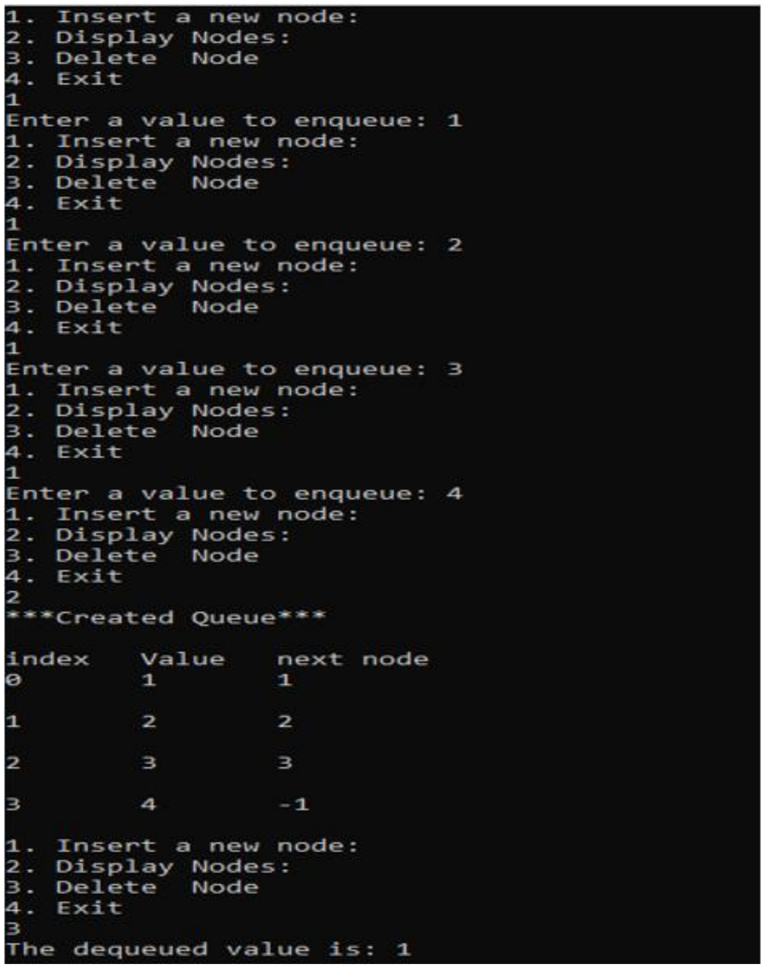
Dequeue a data

```
Step 1: if front == null or front > rear:
                print “Queue underflow”
Step 2: del_val = node[front].data
Step 3: ptr = front
Step 4: front = node[front].next
Step 5: if front == null:
                rear = null
Step 6: Free node ptr
Step 7: Return del_val
Step 8: Exit
```

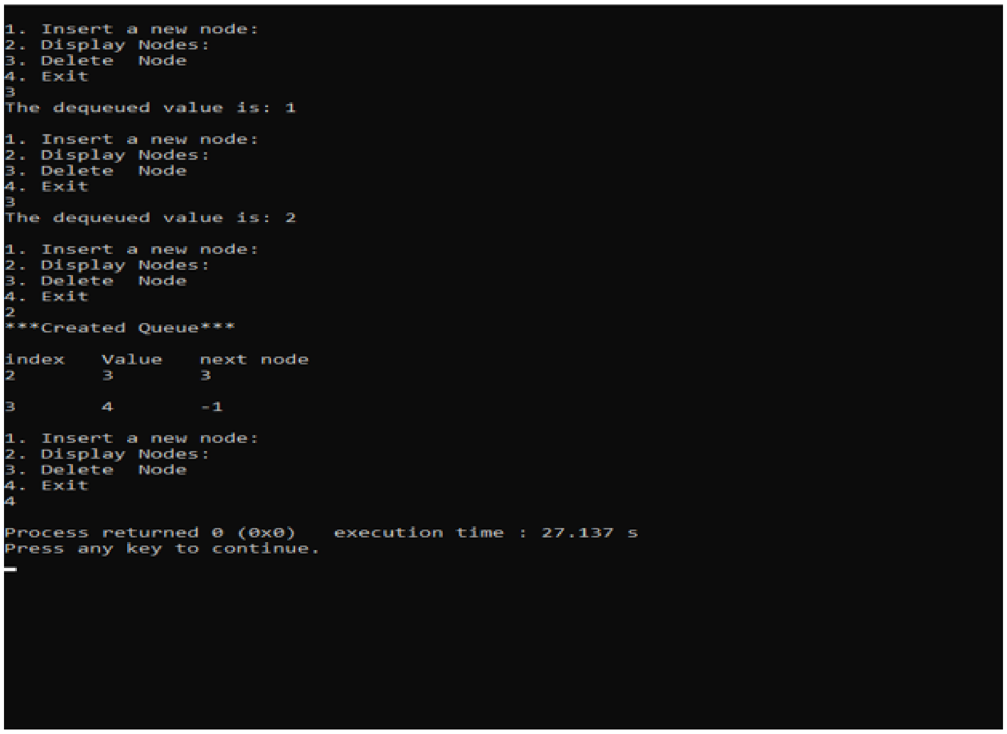
Source Code

```
#include<iostream>
using namespace std;

int max_size = 7;

struct nodeType {
    int data;
    int next;
};

int get_node() {
    if(avail == -1) {
        return -1;
    }
    else {
        int temp;
        temp = avail;
        avail = node[avail].next;
        return temp;
    }
}

void free_node(int n) {
    node[n].data = 0;
    node[n].next = avail;
    avail = n;
}

void display() {
    if(avail==0) {
        cout<<"List Underflow"<<endl;
    }

    do {
        cout << "Index: " << ptr << " Value: " << node[ptr].info << " Next: "<<
node[ptr].next<<endl;
        ptr = node[ptr].next;
    } while(ptr != -1);
}

class Queue{
    private:
        int front, rear;
    public:
        Queue() {
            front = -1;
            rear  = -1;
        }

        bool isempty(){
            return (front == -1);
        }

        void enqueue() {
            int ptr;
            ptr = getnode();

            cout << "Enter an integer:";
            cin >> node[ptr].info;
            if(rear == -1)
                front = ptr;
            else
                node[rear].next = ptr;
            rear = ptr;
        }

        int dequeue() {
            int del_val, ptr;

            if(isempty()) {
                cout << "Underflow" << endl;
                exit(1);
            }
            else{
                del_val = node[front].info;
                ptr   = front;
                front = node[front].next;

                if(front==-1)
                    rear=-1;

                freenode(ptr);
                return del_val;
            }
        }

        int getfront(){
            return front;
        }
};

int main(){
    Queue queue_obj;
    int option = 1;
    int val, pos;

    for(int i = 0; i < max;i++){
        if(i == max-1){
            node[i].next = -1;
        }
        else{
            node[i].next = i+1;
        }
    }

    while(option != 0){
        cout << "Menu:" << endl;
        cout << "1 .Enqueue" << endl;
        cout << "2 .Dequeue" << endl;
        cout << "0 .Exit" << endl;
        cout << "Enter your choice: ";
        cin >> option;
        switch(choose){
        case 1:
            queue_obj.enqueue();
            break;
        case 2:
            val = queue_obj.dequeue();
            cout << val << " is dequeued" << endl;
            break;
        case 0:
            break;
        }
    }
    return 0;
}
```

Output



```
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
1
Enter a value to enqueue: 1
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
1
Enter a value to enqueue: 2
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
1
Enter a value to enqueue: 3
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
1
Enter a value to enqueue: 4
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
2
***Created Queue***

index    Value    next node
0        1        1

1        2        2

2        3        3

3        4        -1

1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
3
The dequeued value is: 1
```



```
1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
3
The dequeued value is: 1

1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
3
The dequeued value is: 2

1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
2
***Created Queue***

index   Value   next node
2       3       3

3       4       -1

1. Insert a new node:
2. Display Nodes:
3. Delete  Node
4. Exit
4

Process returned 0 (0x0)   execution time : 27.137 s
Press any key to continue.
```

Conclusion

Thus, we can use contiguous list implemented through array to implement a queue.

LAB 4

Linked List

4.1 WAP to implement singly linked list

Problem Analysis

Here, a struct containing data and a pointer is created, and a class is further created, encapsulating the variables necessary for a linked list. When the first node is created, the node is null, and for further addition of nodes, the pointer points to the newly created node and hence a list is formed.

Algorithm

Creating a node

```
Step 1: Input data
Step 2: Create a node temp
Step 3: Set temp->data = data
Step 4: if head == null:
            head = temp
            tail   = temp
        else:
            tail->next = temp
            tail = temp
Step 5: temp->next = null
Step 6: Exit
```

Inserting node at beginning

```
Step 1: Input data
Step 2: Create new node temp
Step 3: if temp == null:
            print “Error in memory allocation”
            Goto step 7
Step 4: temp->data = data
Step 5: If head == null:
            temp->next = null
        else:
            temp->next = head
Step 6: head = temp
Step 7: Exit
```

Inserting node at the end

```
Step 1: Input data
Step 2: Create temp
Step 3: if temp == null:
            Goto step 7
Step 4: temp->data = data
        temp->next = null
Step 5: if head == null:
            head = temp
         else:
            ptr = head
            while ptr->next != null:
                    ptr = ptr->next
Step 6: ptr->next = temp
Step 7: Exit
```

Inserting node after a given node

```
Step 1: Input data
Step 2: Create temp
Step 3: if temp == null:
            Goto step 7
Step 4: temp->data = data
        temp->next = null
Step 5: ptr = head, pre_ptr = ptr
Step 6: while pre_ptr->data != num:
              pre_ptr = ptr
              ptr = ptr->next
Step 7: pre_ptr->next = temp
Step 8: temp->next = ptr
Step 9: Exit
```

Insert a node after given node

```
Step 1: Input data
Step 2: Create temp
Step 3: if temp == null:
            Goto step 7
Step 4: temp->data = data
        temp->next = null
Step 5: ptr = head, pre_ptr = ptr
Step 6: while pre_ptr->data != num:
            pre_ptr = ptr
            ptr = ptr->next
Step 7: pre_ptr->next = temp
Step 8: temp->next = ptr
Step 9: Exit
```

Delete first node

```
Step 1: if head == null
            print “Empty list”
Step 2: set ptr = head
Step 3: set head = head->next
Step 4: Delete ptr
Step 5: Exit
```

Delete last node

```
Step 1: if head == null:
            print “Empty list”
Step 2: ptr = head
Step 3: while ptr->next != null:
            pre_ptr = ptr
            ptr = ptr->next
Step 4: pre_ptr->next = null
Step 5: Delete ptr
Step 6: Exit
```

Delete given node

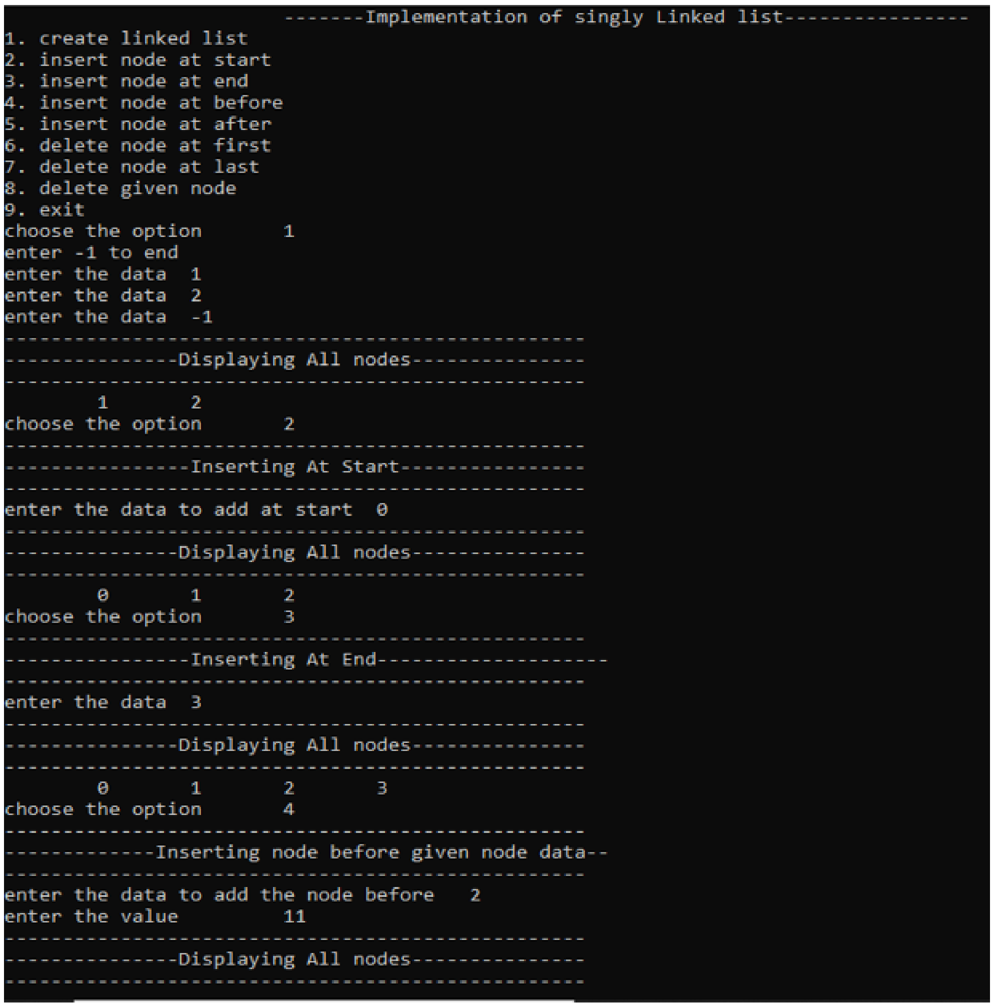
```
Step 1: if head == null:
            Print “Empty list”
Step 2: ptr = head
Step 3: while ptr->data != num:
            pre_ptr = ptr
            ptr = ptr->next
Step 4: temp = ptr
Step 5: pre_ptr->next = ptr->next
Step 6: Exit
```

Source Code

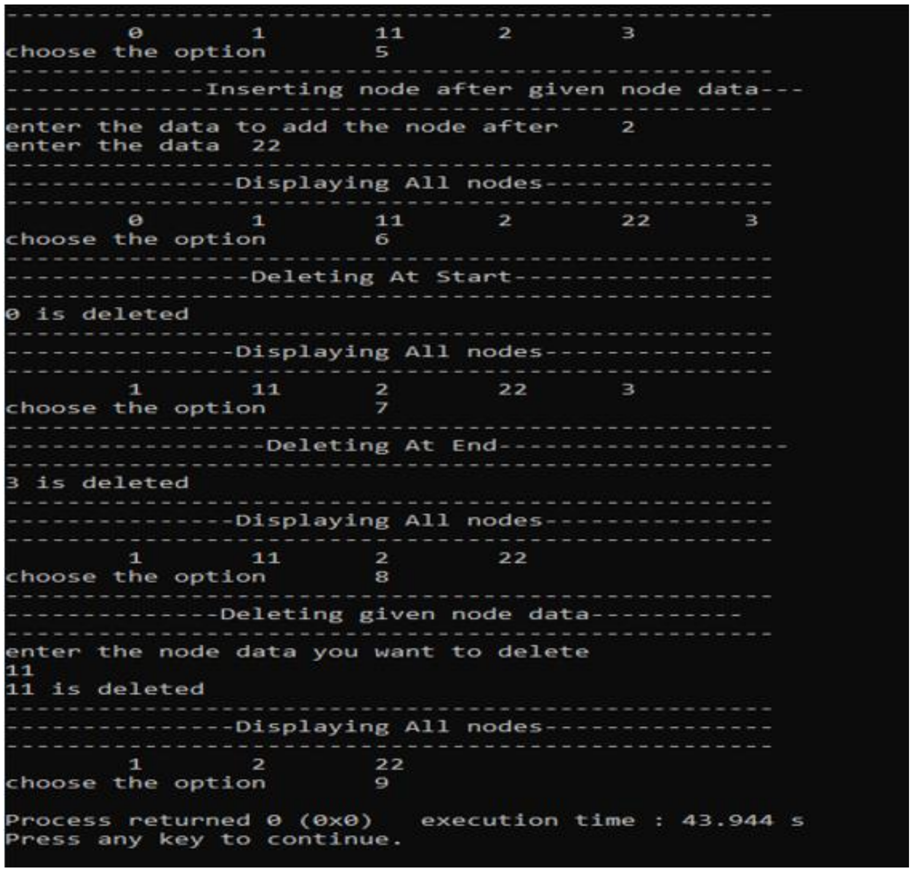
```
#include <iostream>

#define null NULL

using namespace std;

struct Node {
       int data;
       Node *next;
};

class List {

private:
       Node* head, *tail;

public:

       List() {
              head = null;
              tail = null;
       }

       bool isEmpty() { return (head == null); }

       void insert(int data) {
              Node *temp = new Node;

              temp->data = data;
              temp->next = null;

              if(isEmpty()) {
                     head = temp;
                     tail = temp;
              }
              else {
                     tail->next = temp;
                     tail = tail->next;
              }
       }

       void remove() {
       Node *temp = head;

           if(isEmpty()) {
               cerr << "Error, the list is empty" << endl;
           }
           else {
               while(temp->next->next != null)
               {
                   temp = temp->next;
               }
```

```
                temp->next = null;
            }
        }

        void display() {

            if(isEmpty()) {
                cerr << "Error, the list is empty!" << endl;
            }
            else {
                Node *temp = new Node;
                temp = head;

                do {
                    cout << temp->data << " ";
                    temp = temp->next;
                } while (temp->next != null);
            }
        }
};

int main() {

        List *list_obj = new List;
        char option, query;
        int some_data;

        cout << "Enter options below: " << endl;
        cout << "1. Insert data on list" << endl;
        cout << "2. Remove data from list" << endl;
        cout << "3. Display the list" << endl;

        cin >> option;

        if(option == '1') {

            do {
                cout << endl << "Enter data: ";
                cin  >> some_data;

                list_obj->insert(some_data);

                cout << "Do you want to add more elements? [Y/N] ";
                cin  >> query;
            } while(query != 'N' || query != 'n');
        }
        else if (option == '2') {
            list_obj->remove();
        }
        else if (option == '3') {
            list_obj->display();
        }
        else {
            cerr << "Error, incorrect option" << endl;
        }

        return 0;
}
```

Output



```
                        -------Implementation of singly Linked list----------------
1. create linked list
2. insert node at start
3. insert node at end
4. insert node at before
5. insert node at after
6. delete node at first
7. delete node at last
8. delete given node
9. exit
choose the option       1
enter -1 to end
enter the data  1
enter the data  2
enter the data  -1
--------------------------------------------------
---------------Displaying All nodes---------------
--------------------------------------------------
        1       2
choose the option       2
--------------------------------------------------
----------------Inserting At Start----------------
--------------------------------------------------
enter the data to add at start  0
--------------------------------------------------
---------------Displaying All nodes---------------
--------------------------------------------------
        0       1       2
choose the option       3
--------------------------------------------------
----------------Inserting At End--------------------
--------------------------------------------------
enter the data  3
--------------------------------------------------
---------------Displaying All nodes---------------
--------------------------------------------------
        0       1       2       3
choose the option       4
--------------------------------------------------
-------------Inserting node before given node data--
--------------------------------------------------
enter the data to add the node before   2
enter the value         11
--------------------------------------------------
---------------Displaying All nodes---------------
--------------------------------------------------
```



```
--------------------------------------------------
        0       1       11      2       3
choose the option       5
--------------------------------------------------
-------------Inserting node after given node data---
--------------------------------------------------
enter the data to add the node after    2
enter the data  22
--------------------------------------------------
---------------Displaying All nodes---------------
--------------------------------------------------
        0       1       11      2       22      3
choose the option       6
--------------------------------------------------
----------------Deleting At Start-----------------
--------------------------------------------------
0 is deleted
--------------------------------------------------
---------------Displaying All nodes---------------
--------------------------------------------------
        1       11      2       22      3
choose the option       7
--------------------------------------------------
------------------Deleting At End-------------------
--------------------------------------------------
3 is deleted
--------------------------------------------------
---------------Displaying All nodes---------------
--------------------------------------------------
        1       11      2       22
choose the option       8
--------------------------------------------------
--------------Deleting given node data----------
--------------------------------------------------
enter the node data you want to delete
11
11 is deleted
--------------------------------------------------
---------------Displaying All nodes---------------
--------------------------------------------------
        1       2       22
choose the option       9

Process returned 0 (0x0)   execution time : 43.944 s
Press any key to continue.
```

Conclusion

Thus, linked list can be implemented through the use of nodes, and the list will be of dynamic nature.

4.2 WAP to implement Circular Linked List

Problem Analysis

Here we have to create a node . Each node contains a pointer that points to the next node in the list. The last node's next pointer is pointed to the first node .

Algorithm

Create node

```
Step 1: Input data VAL
Step 2: Create a NEW_NODE
Step 3: SET NEW_NODE =>DATA = VAL
Step 4: IF START = NULL
                SET START=NEW_NODE
                SET END = NEW_NODE
        ELSE
                END=>NEXT=NEW_NODE
                END=NEW_NODE
Step 5: END=>NEXT = START
Step 6: Exit
```

Insert node at Beginning

```
Step 1: Input data VAL
Step 2: Create a NEW_NODE
Step 3: IF NEW_NODE = NULL
         write ERROR IN MEMORY ALLOCATION
        Go to Step 7
Step 4: SET NEW_NODE => DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7
```

```
        While PTR => NEXT != START
    Step 7: SET PTR=PTR=>NEXT
        [END OF While LOOP]
    Step 8: SET NEW_NODE =>NEXT = START
    Step 9: PTR=>NEXT = NEW_NODE
    Step 10: SET START = NEW_NODE
    Step 11: Exit
```

Insert node at End

```
    Step 1: Input data VAL
    Step 2: Create a NEW_NODE
    Step 3: IF NEW_NODE = NULL
        Step 3.1: write ERROR IN MEMORY ALLOCATION
        Step 3.2: Go to Step 9
     Step 4: SET NEW_NODE =>DATA = VAL
    Step 5: SET NEW_NODE =>NEXT = START
    Step 6: SET PTR=START
    Step 7: Repeat Step 8
        While PTR => NEXT!=START
    Step 8: SET PTR = PTR=>NEXT
        [END of While Loop]
     Step 9: SET PTR=>NEXT = NEW_NODE
    Step 10: Exit
```

Insert node after a given node

```
    Step 1: Input data VAL
    Step 2: Create a NEW_NODE
    Step 3: IF NEW_NODE = NULL
         Step 3.1: write ERROR IN MEMORY ALLOCATION
        Step 3.2: Go to Step 12
```

```
Step 4: SET NEW_NODE => DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR => DATA != NUM
Step 8: SET PREPTR = PTR
Step 9: SET PTR = PTR=>NEXT
        [End of while loop]
Step 10: PREPTR=> NEXT = NEW_NODE
Step 11: SET NEW_NODE =>NEXT = PTR
Step 12: Exit
```

Insert node before a given node

```
Step 1: Input data VAL
Step 2: Create a NEW_NODE
Step 3: IF NEW_NODE = NULL
        Step 3.1: write ERROR IN MEMORY ALLOCATION
        Step 3.2: Go to Step 12
Step 4: SET NEW_NODE =>DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR => DATA != NUM
Step 8: SET PREPTR = PTR
Step 9: SET PTR = PTR=>NEXT
        [End of while loop]
Step 10: PREPTR=>NEXT = NEW_NODE
Step 11: SET NEW_NODE =>NEXT = PTR
Step 12: Exit
```

Delete first node

```
Step 1: If START = NULL
        Write ‘underflow’
        Goto Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 While PTR => NEXT != START
Step 4: SET PTR=PTR=>NEXT
        [END OF While LOOP]
Step 5: SET PTR=>NEXT=START=>NEXT
Step 6: FREE START
Step 7 SET START=PTR=>NEXT
Step 8: Exit
```

Delete last node

```
Step 1: IF START = NULL
        Write ‘Underflow’
        Goto Step 8
        [END OF IF]
Step 2: SET PTR= START
Step 3: Repeat Steps 4 and 5 While PTR => NEXT ! = NULL
Step 4: SET PREPTR =PTR
Step 5: SET PTR = PTR=>NEXT
        [END OF LOOP]
Step 6:  SET PREPTR =>NEXT = START
Step 7: FREE PTR
Step 8: Exit
```

Delete a given node

```
Step 1: IF START = NULL
            Write ‘Underflow’
            Goto Step 9
            [END OF IF]
Step 2: SET PTR= START
Step 3: Repeat Steps 4 and 5 While PTR => DATA!=NUM
Step 4: SET PREPTR =PTR
Step 5: SET PTR = PTR=>NEXT
            [END OF LOOP]
Step 6: SET TEMP = PTR
Step 7: SET PREPTR =>NEXT = PTR => NEXT
Step 8: FREE TEMP
Step 9: Exit
```

Source Code

```
#include <iostream>

#define null NULL

using namespace std;

struct Node {
	int data;
	Node *next;
};

class List {

private:
	Node* head, *tail;

public:

	List() {
		head = null;
		tail = null;
	}

	bool isEmpty() { return (head == null); }

	void insert(int data) {
		Node *temp = new Node;
		Node *prev = head;

		temp->data = data;
		temp->next = head;

		if(isEmpty()) {
			temp->next = temp;
		}
		else {
			do {
				prev = prev->next;
			} while(prev->next != null);
			prev->next = temp;
		}

		head = temp;
	}

	void display() {

		if(isEmpty()) {
			cerr << "Error, the list is empty!" << endl;
		}
		else {
			Node *temp = head;
```

```
                        do {
                                cout << temp->data << " ";
                                temp = temp->next;
                        } while (temp->next != head);
                }
        }
};

int main() {

        List *list_obj = new List;

        for(int i = 0; i < 7; ++i) {
                list_obj->insert(rand() % 7);
        }
        list_obj->display();

        return 0;
}
```

4.3 WAP to implement Doubly Linked List

Problem Analysis

Here we have to create a node. Each node contains a pointer that points to the next node in the list. The last node's next pointer is pointed to the first node. Since it is doubly linked list or a two-way linked list which is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence. The problem here is to linked previous and current node. We have to create such type of linked list where not only forward but also backward traversing is possible.

Algorithm

Inserting at beginning

```
Step 1: Input data VAL
Step 2: Create a NEW_NODE
Step 3: IF NEW_NODE = NULL
    Step 3.1: write ERROR IN MEMORY ALLOCATION
    Step 3.2: Go to Step 9
Step 4: SET NEW_NODE => DATA = VAL
Step 5: SET NEW_NODE =>PREV = NULL
Step 6: SET NEW_NODE => NEXT = START
Step 7: SET START=>PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

Inserting at End

```
Step 1: Input data VAL
Step 2: Create a NEW_NODE
Step 3: IF NEW_NODE = NULL
    Step 3.1: write ERROR IN MEMORY ALLOCATION
```

```
        Step 3.2: Go to Step 11
    Step 4: SET NEW_NODE => DATA = VAL
    Step 5: SET NEW_NODE =>NEXT = NULL
    Step 6: SET PTR = START
    Step 7: Repeat Step 8 While PTR=> NEXT != NULL
    Step 8: SET PTR = PTR=>NEXT
    Step 9: SET PTR => NEXT = NEW_NODE
     Step 10: Set NEW_NODE=>PREV = PTR
     Step 11: EXIT
```

Inserting node after a given node

```
    Step 1: Input data VAL
    Step 2: Create a NEW_NODE
    Step 3: IF NEW_NODE = NULL
        Step 3.1: write ERROR IN MEMORY ALLOCATION
        Step 3.2: Go to Step 12
     Step 4: SET NEW_NODE => DATA = VAL
     Step 5: SET PTR = START
     Step 6: Repeat Step 7 While PTR=>DATA != NUM
     Step 7: SET PTR = PTR=>NEXT
         [End of while Loop]
     Step 8: SET NEW_NODE =.>NEXT = PTR ◊ NEXT
     Step 9: SET NEW_NODE=>PREV = PTR
     Step 10 : SET PTR=>NEXT=>PREV = NEW_NODE
     Step 11: SET PTR=>NEXT = NEW_NODE
     Step 12: EXIT
```

Inserting node before a given node

```
Step 1: Input data VAL
Step 2: Create a NEW_NODE
Step 3: IF NEW_NODE = NULL
        Step 3.1: write ERROR IN MEMORY ALLOCATION
        Step 3.2: Go to Step 12
Step 4: SET NEW_NODE =>DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 While PTR=> DATA != NUM
Step 7: SET PTR = PTR=>NEXT [End of while Loop]
Step 8: SET NEW_NODE =>NEXT = PTR
Step 9: SET NEW_NODE=>PREV = PTR◊PREV
Step 10: SET PTR=>PREV=>NEXT = NEW_NODE
Step 11: SET PTR=>PREV = NEW_NODE
Step 12: EXIT
```

Deleting the first node

```
Step 1: IF START = NULL
        Write “underflow”
        Goto step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START=START=>NEXT
Step 4: SET START=>PREV=NULL
Step 5: FREE PTR
Step 6:  EXIT
```

Deleting the last node

```
Step 1: IF START = NULL
            Write “underflow”
            Goto step 7
            [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 While PTR=>NEXT!=NULL
Step 4: SET PTR = PTR=>NEXT
            [ END OF LOOP ]
Step 5: SET PTR=>PREV=>NEXT=NULL
Step 6: FREE PTR
Step 7: EXIT
```

Deleting node after a given node

```
Step 1: IF START=NULL
            Write “UNDERFLOW”
            Goto Step 9
            [ END OF IF ]
Step 2: SET PTR= START
Step 3: Repeat Step 4 While PTR=>DATA!=NUM
Step 4: SET PTR=PTR=>NEXT
            [END OF LOOP]
Step 5: SET TEMP = PTR=>NEXT
Step 6: SET PTR=>NEXT = TEMP=>NEXT
Step 7: SET TEMP=>NEXT=>PREV=PTR
Step 8: FREE TEMP
Step 9:  EXIT
```

Deleting node before a given node

```
Step 1: IF START=NULL
        Write “UNDERFLOW”
        Goto Step 9
        [ END OF IF ]
Step 2: SET PTR= START
Step 3: Repeat Step 4 While PTR=>DATA!=NUM
Step 4: SET PTR=PTR=>NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR=>PREV
Step 6: SET TEMP=>PREV=>NEXT = PTR
Step 7: SET PTR=>PREV = TEMP =>PREV
Step 8: FREE TEMP
Step 9: EXIT
```

Deleting a given node

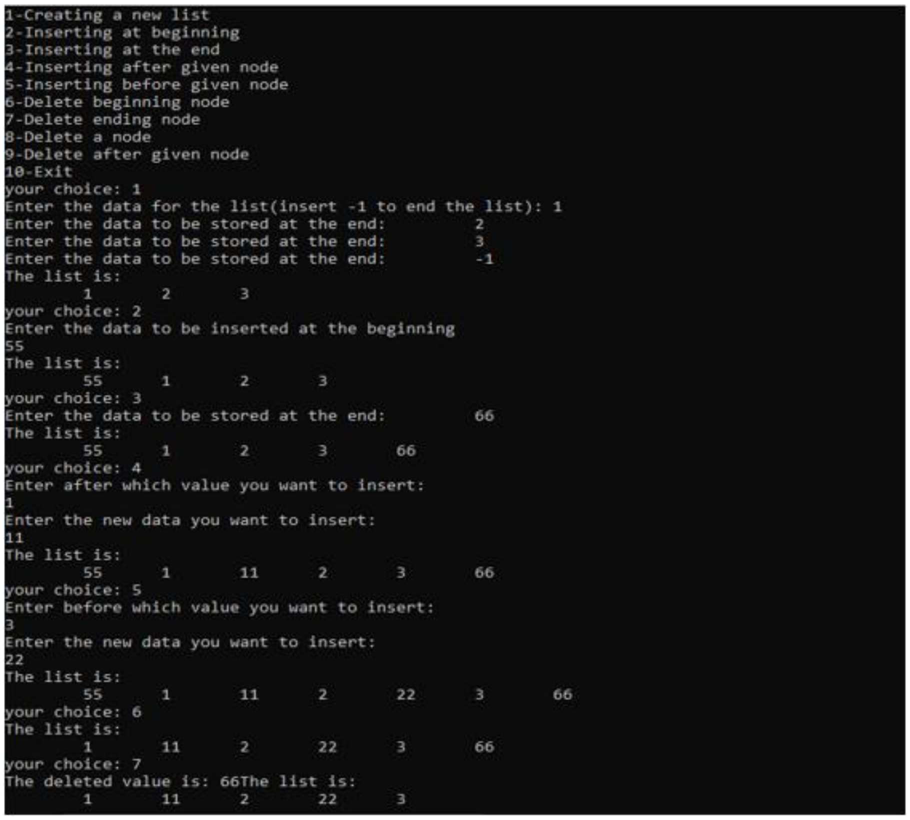
```
Step 1: IF START=NULL
        Write “UNDERFLOW”
        Goto Step 8
        [ END OF IF ]
Step 2: SET PTR= START
Step 3: Repeat Step 4 While PTR=>DATA!=NUM
Step 4: SET PTR=PTR=>NEXT
        [END OF LOOP]
Step 5: SET PTR=>PREV=>NEXT=PTR=>NEXT
Step 6: SET PTR=>NEXT=>PREV=PTR=>PREV
Step 7: FREE PTR
Step 8: EXIT
```

Source Code

```
#include <iostream>

#define null NULL

using namespace std;

struct Node {
	int data;
	Node *next;
	Node *prev;
};

class List {

private:
	Node* head;
	Node* tail;

public:

	List() {
		head = null;
		tail = null;
	}

	bool isEmpty() { return (head == null); }

	void insert(int data) {
		Node *temp = new Node;

		temp->data = data;

		if(isEmpty()) {
			head = temp;
			tail = temp;

			head->prev = null;
			tail->next = null;
		}
		else {
			tail->next = temp;
			temp->prev = tail;
			tail = temp;
			tail->next = null;
		}
	}

	void remove() {
	Node *temp = new Node;
	temp = head;

	head = head->next;
	head->next->prev = null;

	delete temp;
```

```
    }

       void display() {
              Node *temp = head;

              if(isEmpty()) {
                     cerr << "Error, the list is empty!" << endl;
              }
              else {
                     do {
                            cout << temp->data << " ";
                            temp = temp->next;
                     } while(temp->next != null);
              }
       }
};

int main() {
       List *list_obj = new List;
       char option, query;
       int some_data;

       cout << "Enter options below: " << endl;
       cout << "1. Insert data on list" << endl;
       cout << "2. Remove data from list" << endl;
       cout << "3. Display the list" << endl;

       cin >> option;

       if(option == '1') {

              do {
                     cout << endl << "Enter data: ";
                     cin  >> some_data;

                     list_obj->insert(some_data);

                     cout << "Do you want to add more elements? [Y/N] ";
                     cin  >> query;
              } while(query != 'N' || query != 'n');
       }
       else if (option == '2') {
              list_obj->remove();
       }
       else if (option == '3') {
              list_obj->display();
       }
       else {
              cerr << "Error, incorrect option" << endl;
       }

       return 0;
}
```

Output



```
1-Creating a new list
2-Inserting at beginning
3-Inserting at the end
4-Inserting after given node
5-Inserting before given node
6-Delete beginning node
7-Delete ending node
8-Delete a node
9-Delete after given node
10-Exit
your choice: 1
Enter the data for the list(insert -1 to end the list): 1
Enter the data to be stored at the end:         2
Enter the data to be stored at the end:         3
Enter the data to be stored at the end:         -1
The list is:
        1       2       3
your choice: 2
Enter the data to be inserted at the beginning
55
The list is:
        55      1       2       3
your choice: 3
Enter the data to be stored at the end:         66
The list is:
        55      1       2       3       66
your choice: 4
Enter after which value you want to insert:
1
Enter the new data you want to insert:
11
The list is:
        55      1       11      2       3       66
your choice: 5
Enter before which value you want to insert:
3
Enter the new data you want to insert:
22
The list is:
        55      1       11      2       22      3       66
your choice: 6
The list is:
        1       11      2       22      3       66
your choice: 7
The deleted value is: 66The list is:
        1       11      2       22      3
```

Conclusion

Thus, the linked list implemented is a doubly linked list, which can be traversed both ways.

4.4 WAP to implement priority queue using list

Problem Analysis

The problem here is to implement priority queue using linked list. Where we have to create a queue according to its priority and dequeue as FIFO principle according to its priority order.

Algorithm

Enqueue

```
Step 1: Allocate memory for the new node
Step 2: SET NEW_NODE=>DATA=VAL
Step 3: SET NEW_NODE=>NEXT=NULL
Step 4: PTR= START
Step 5: Repeat Step while NEW_NODE=>DATA!=NULL
    Step 5.1: IF NEW_NODE=>PRIORITY<START=>PRIORITY
                SET NEW_NODE=>START
                SET START=NEW_NODE
            ELSE
                Repeat  While PTR=>NEXT!=NULL AND
                PTR=>NEXT=>PRIORITY<NEW_NODE=>PRIORITY
                    PTR=PTR=>NEXT
                [END OF WHILE LOOP]
                SET NEW_NODE=>NEXT=PTR=>NEXT
                SET PTR=>NEXT = NEW_NODE
Step 6: EXIT
```

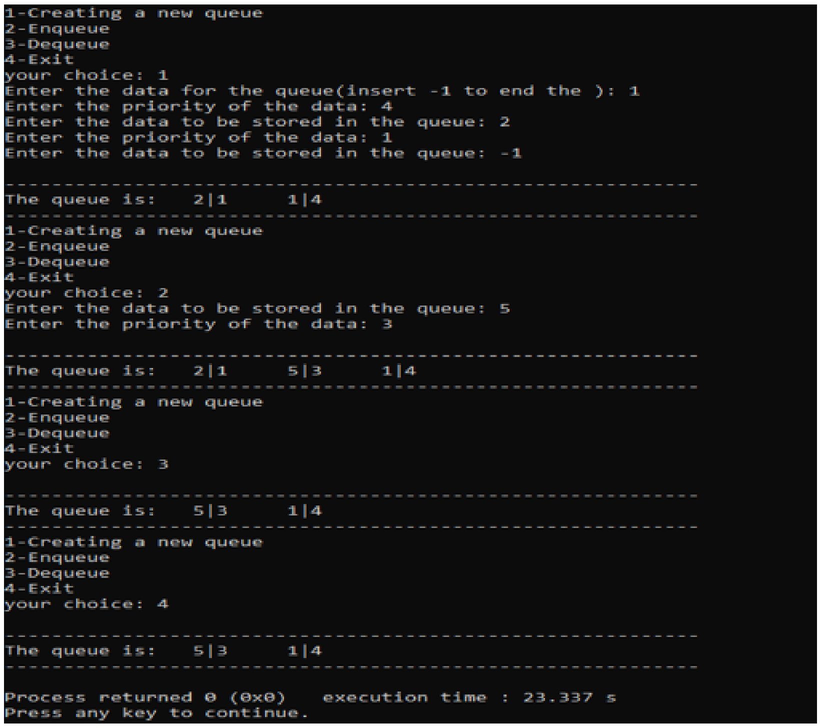
Dequeue

```
Step 1: SET PTR=START=>NEXT
Step 2: FREE START
Step 3: SET START = PTR
Step 4: EXIT
```

Source Code

```
#include <iostream>
using namespace std;

#define null NULL

struct Node {
       Node *next;
       int data;
       int priority;
};

// Queue class
class Queue {

       Node *head;

public:

       Queue() {
              head = null;
       }

       bool isEmpty() { return(front == null); }

       void enque(int data, int prior) {
              Node *new_node = new Node;
              Node *temp = head;
              Node *prev = null;

              if(isEmpty()) {
                     new_node->data = data;
                     new_node->priority = prior;
                     new_node->next = head;
                     head = new_node;
              }
              else {
            while(temp->priority < prior){
                prev = temp;
                if(temp->next == NULL){
                    break;
                }
                temp = temp->next;
            }
            if(temp == NULL){
                new_node = new Node;
                new_node->next = prev;
                new_node->data = data;
                temp->priority = prior;
                head = temp;
            }
            else if(temp->priority >= prior){
                     new_node = new Node;
                prev->next = new_node;
                new_node->next = temp;
                new_node->data = data;
                new_node->priority = prior;
            }
            else{
                new_node = new Node;
                temp->next = new_node;
                new_node->data = data;
                new_node->priority = prior;
                new_node->next =NULL;
            }

              }
        }


        void deque() {
         Node *temp = head;

         if(isEmpty()) {
               cerr << "Error, queue empty" << endl;
         }
         else {
               Node *temp = head;
               head = temp->next;

               cout << temp->data << " dequed" << endl;
               delete temp;
         }
        }

        // Traversing and displaying the queue
        void display() {
               if(isEmpty()) {
                      cout << "Queue empty...." << endl;
               }
               else {
                      Node *temp = front;

                      do {
                             cout << temp->data << " ";
                             temp = temp->next;
                      } while(temp->next != null);

                      cout << endl;
               }
        }
};

int main() {

        Queue *queue_obj = new Queue;

        for(int i = 0; i < 6; ++i) {
               queue_obj->enque(rand() % 7);
        }

        cout << "Queue: ";
        queue_obj->display();
        queue_obj->deque();
        cout << "Queue after deque: ";
        queue_obj->display();
        cout << endl;

        delete queue_obj;
        return 0;
}
```

Output



```
1-Creating a new queue
2-Enqueue
3-Dequeue
4-Exit
your choice: 1
Enter the data for the queue(insert -1 to end the ): 1
Enter the priority of the data: 4
Enter the data to be stored in the queue: 2
Enter the priority of the data: 1
Enter the data to be stored in the queue: -1

----------------------------------------------------------------
The queue is:   2|1     1|4
----------------------------------------------------------------
1-Creating a new queue
2-Enqueue
3-Dequeue
4-Exit
your choice: 2
Enter the data to be stored in the queue: 5
Enter the priority of the data: 3

----------------------------------------------------------------
The queue is:   2|1     5|3     1|4
----------------------------------------------------------------
1-Creating a new queue
2-Enqueue
3-Dequeue
4-Exit
your choice: 3

----------------------------------------------------------------
The queue is:   5|3     1|4
----------------------------------------------------------------
1-Creating a new queue
2-Enqueue
3-Dequeue
4-Exit
your choice: 4

----------------------------------------------------------------
The queue is:   5|3     1|4
----------------------------------------------------------------

Process returned 0 (0x0)   execution time : 23.337 s
Press any key to continue.
```

Conclusion

Thus, priority queue was implemented through the use of dynamic linked list data structure.

4.5 WAP to implement Stack using Linked List

Problem Analysis

The problem here is to use linked list to create a stack. Using linked list, we can insert and delete data, so we use it to use it as a stack by implementing LIFO principle. We push data to the linked list and pop data from the linked list.

Algorithm

Push

```
Step 1: Allocate memory for the new_node and named it as NEW_NODE
Step 2: SET NEW_NODE=>DATA = VAL
Step 3: IF TOP =NULL
                SET NEW_NODE=>NEXT=NULL
                SET TOP=NEW_NODE
        ELSE
                SET NEW_NODE=>NEXT = TOP
                SET TOP = NEW_NODE
        [ END OF IF ]
Step 4: END
```

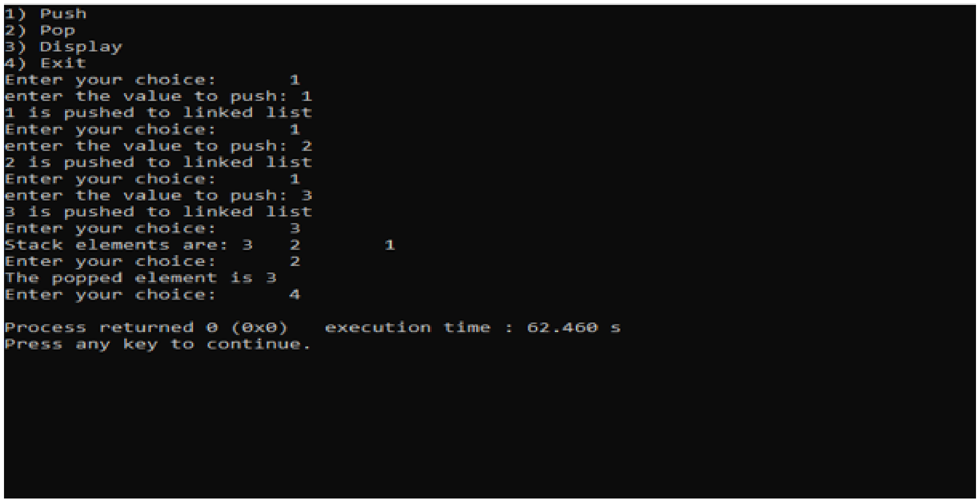
Pop

```
Step 1: IF TOP=NULL
                PRINT “UNDERFLOW”
                Goto Step 5
        [END OF IF ]
Step 2: SET PTR=TOP
Step 3: SET TOP = TOP=>NEXT
Step 4: FREE PTR
Step 5: END
```

Source Code:

```
#include <iostream>
using namespace std;

#define null NULL

// Struct for storing Node of the linked list
struct Node {
	Node *next;
	int data;
};

// Stack class
class Stack {
private:
	Node *head;

public:

	Stack() {
		head = null;
	}

	bool isEmpty() { return(head == null); }

	void push(int data) {
		Node *temp = new Node;

		temp->data = data;

		if(!isEmpty()) {
			temp->next = head;
		}
		else {
			temp->next = null;
		}

		head = temp;
	}

	void pop() {

		if(isEmpty()) {
			cerr << "Error, stack underflow!" << endl;
		}
		else {
			Node *temp = new Node;
			temp = head;

			head = head->next;

			delete temp;
		}
	}

	void peek() {
		cout << head->data << endl;
	}

	void display() {
		Node *temp = new Node;
		temp = head;

		do {
			cout << temp->data << " ";
			temp = temp->next;
		} while(temp->next != null);

		cout << endl;
	}
};

int main() {

	Stack *stack_obj = new Stack;

	for(int i = 0; i < 6; ++i) {
		stack_obj->push(rand() % 7);
	}

	stack_obj->peek();
	cout << "StackL ";
	stack_obj->display();
	stack_obj->pop();
	cout << "Stack after popping: ";
	stack_obj->display();
	cout << endl;

	delete stack_obj;
	return 0;
}
```

Output



```
1) Push
2) Pop
3) Display
4) Exit
Enter your choice:      1
enter the value to push: 1
1 is pushed to linked list
Enter your choice:      1
enter the value to push: 2
2 is pushed to linked list
Enter your choice:      1
enter the value to push: 3
3 is pushed to linked list
Enter your choice:      3
Stack elements are: 3   2       1
Enter your choice:      2
The popped element is 3
Enter your choice:      4

Process returned 0 (0x0)   execution time : 62.460 s
Press any key to continue.
```

Conclusion

Thus, stack was implemented through the use of dynamic linked list, and operations, like push, pop, peek and traversal can be done on the stack.

4.6 WAP to implement Queue using Linked List

Problem Analysis

The problem here is to implement the principle of FIFO principle to implement queue using linked list data structure. For this, we insert and delete the data from the rear end and the front respectively. We define two node pointers, *front and *rear which points to front end and rear end respectively. To insert data we create a new node and initialize the next pointer of rear to the newly created node and initialize rear end with the new node. For deleting data, we first store the front pointer to a temporary variable and initialize front with the value of next pointer and then delete the temp variable.

Algorithm

```
Enqueue:
        Step 1 : Allocate memory for the new node and name it as PTR
        Step 2 : SET PTR=>DATA =VAL
        Step 3 : IF FRONT = NULL
                        SET FRONT = REAR=PTR
                        SET FRONT => NEXT = REAR => NEXT = NULL
                ELSE
                        SET REAR=>NEXT = PTR
                        SET REAR=PTR
                        SET REAR=>NEXT = NULL
                [END OF IF]
        Step 4: END
Dequeue :
        Step 1 : IF FRONT = NULL
                        Write “UNDERFLOW”
                        Goto Step 5
                [END OF IF]
```

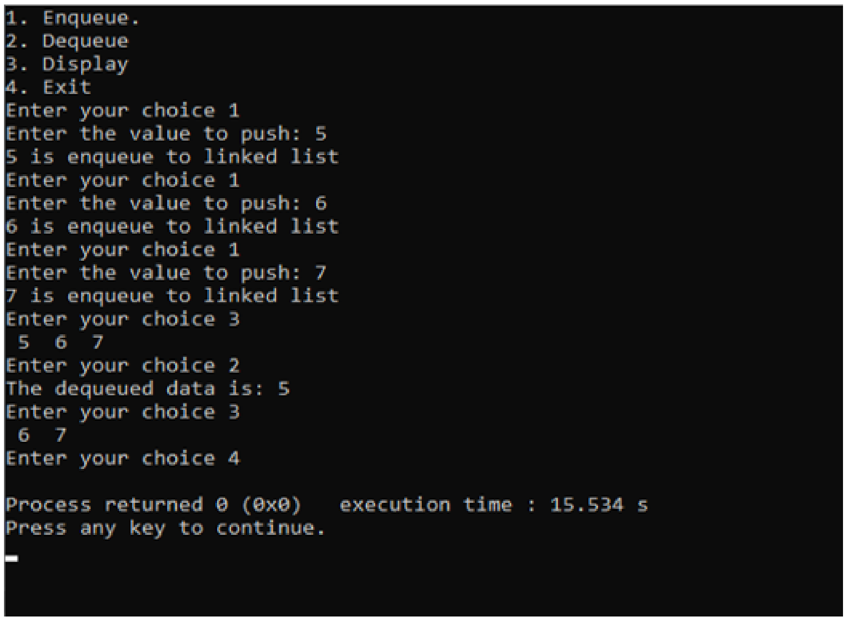
```
Step 2 : SET PTR = FRONT
Step 3 : SET FRONT = FRONT=>NEXT
Step 4 : FREE PTR
Step 5 : END
```

Source Code

```cpp
#include <iostream>
using namespace std;

#define null NULL

// Struct for storing Node of the linked list
struct Node {
	Node *next;
	int data;
};

// Queue class
class Queue {

	Node *front, *rear;                // Two pointers that point to the front and rear
of the Queue

public:

	Queue() {
		front = null;
		rear  = null;
	}

	bool isEmpty() { return(front == null); }

	// Method for pushing data into the queue
	void enque(int data) {
		Node *temp = new Node;
		temp->data = data;

		if(isEmpty()) {
			front = temp;
			rear  = temp;
		}
		else {
			rear->next = temp;
			rear = temp;
		}
	}

	// Method for removing data from front of the queue
	void deque() {
		if(isEmpty()) {
			cout << "Queue empty...." << endl;
		}
		else {
```

```
                        Node *temp = new Node;

                        temp  = front;
                        front = front->next;
                        if(front == null) {
                                rear = null;
                        }
                        delete temp;
                }
        }

        // Returns the front of the queue
        void frontof() {
                if(isEmpty()) {
                        cout << "Queue empty...." << endl;
                }
                else {
                        cout << "Front of the queue: " << front->data << endl;
                }
        }

        // Traversing and displaying the queue
        void display() {
                if(isEmpty()) {
                        cout << "Queue empty...." << endl;
                }
                else {
                        Node *temp = front;

                        do {
                                cout << temp->data << " ";
                                temp = temp->next;
                        } while(temp->next != null);

                        cout << endl;
                }
        }
};

int main() {

        Queue *queue_obj = new Queue;

        for(int i = 0; i < 6; ++i) {
                queue_obj->enque(rand() % 7);
        }

        queue_obj->frontof();
        cout << "Queue: ";
        queue_obj->display();
        queue_obj->deque();
        cout << "Queue after deque: ";
        queue_obj->display();
        cout << endl;

        delete queue_obj;
        return 0;
}
```

Output



```
1. Enqueue.
2. Dequeue
3. Display
4. Exit
Enter your choice 1
Enter the value to push: 5
5 is enqueue to linked list
Enter your choice 1
Enter the value to push: 6
6 is enqueue to linked list
Enter your choice 1
Enter the value to push: 7
7 is enqueue to linked list
Enter your choice 3
 5  6  7
Enter your choice 2
The dequeued data is: 5
Enter your choice 3
 6  7
Enter your choice 4

Process returned 0 (0x0)   execution time : 15.534 s
Press any key to continue.
```

Conclusion

Thus, stack was implemented through dynamic linked list, and different operations, like enqueue, dequeue and traversal, was performed in it.

4.7 WAP to store a polynomial using linked list, and perform addition and subtraction on two polynomials

Problem Analysis

In this program we perform an addition and subtraction of two polynomials by storing a polynomial using linked list.

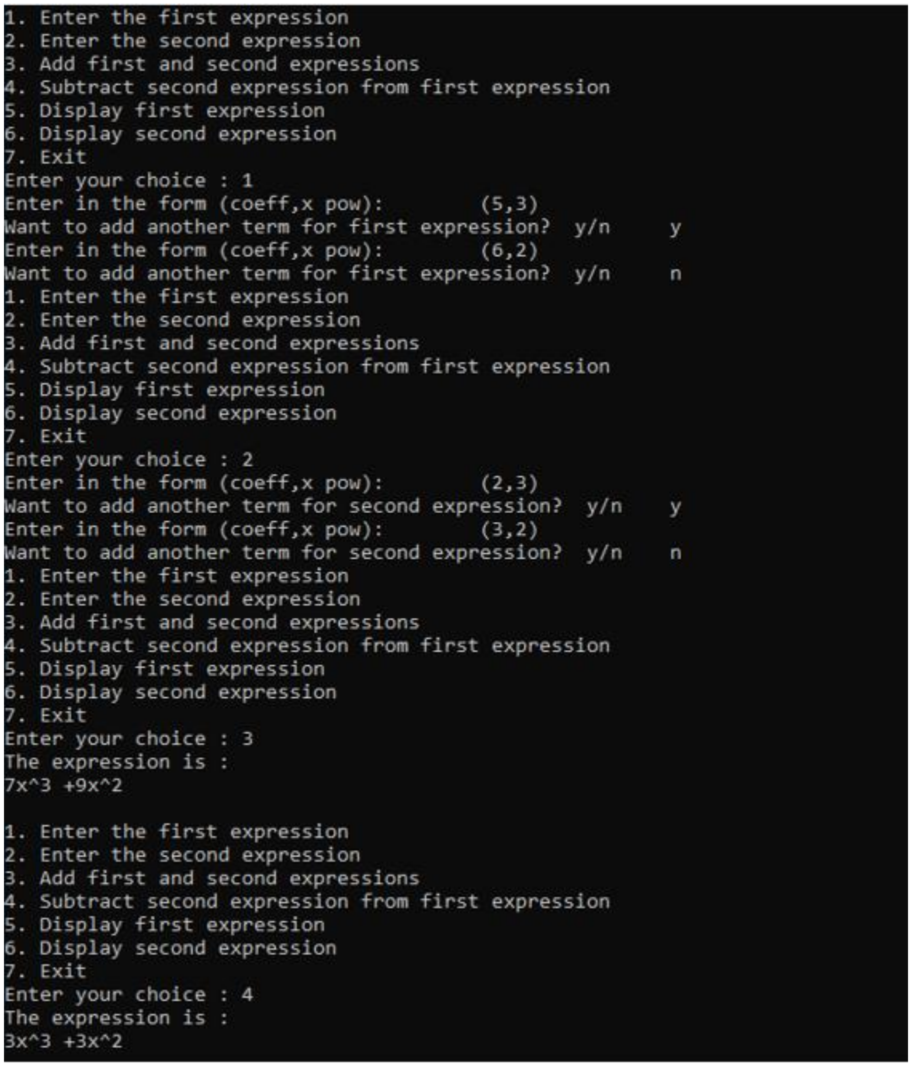
Algorithm

```
Step 1: Create a structure for coefficient , exponents and next for
            new node.
Step 2: Call a function to create a polynomial, add a polynomial,
            Subtract a polynomial, and display a polynomial.
Step 3: Scan two polynomials node by node comparing for the
            exponents.
Step 4: If the exponents are equal, add/subtract their coefficients.
Step 5: If the exponents are not equal, simply add/subtract a node
            in the new list.
```

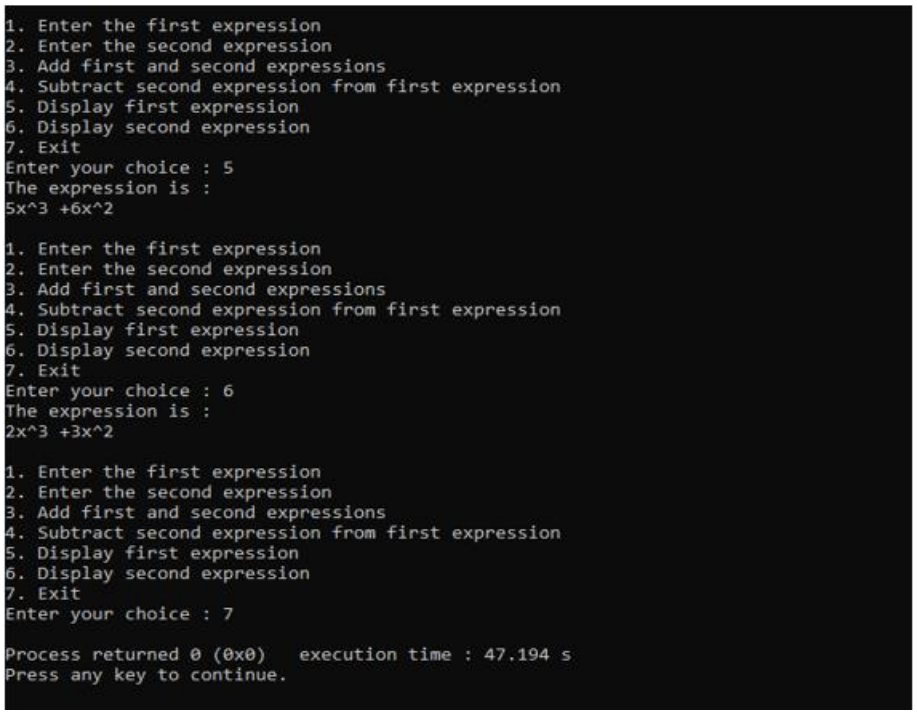
Source Code

```
#include<iostream>
#include<cstdlib>
#include<cmath>

using namespace std;
struct node
{
    int check;
    int info, xp;
    node* next;
};
class POLY
{
    node* START;
public:
    POLY() :START(NULL) {}
    void AddExpression(int, int);
    POLY operator + (POLY&);
    POLY operator - (POLY&);
    bool DisplayExpression();
};
void POLY::AddExpression(int num, int x)
{
    node* temp = new node;
    if (temp == NULL)
        cout << "Failed to initialize the memory for new block.\n";
    else
    {
        temp->info = num;
        temp->xp = x;
        temp->next = NULL;
        if (START == NULL)
            START = temp;
        else
        {
            node* ptr;
            ptr = START;
            while (ptr->next != NULL)
                ptr = ptr->next;
            ptr->next = temp;
        }
    }
}
POLY POLY::operator+(POLY& second)
{
    POLY t;
    if (START == NULL)
    {
        cout << "There is no first polynomial expression.\n";
        return t;
    }
    else if (second.START == NULL)
    {
        cout << "There is no second polynomial expression.\n";
        return t;
    }
    else
    {
        int c;
        node* p1, * p2;
        p1 = START;
        while (p1 != NULL)
        {
            c = 0;
            p2 = second.START;
            while (p2 != NULL)
            {
                if (p1->xp == p2->xp)
                {
                    c = 1;
                    p2->check = 1;
                    t.AddExpression((p1->info + p2->info), p1->xp);
                    break;
                }
                p2 = p2->next;
            }
            if (c == 0)
                t.AddExpression(p1->info, p1->xp);
            p1 = p1->next;

        }
        p2 = second.START;
        while (p2 != NULL)
        {
            if (p2->check != 1)
                t.AddExpression(p2->info, p2->xp);
            p2 = p2->next;
        }
        return t;
    }
}
POLY POLY::operator-(POLY& second)
{
    POLY t;
    if (START == NULL)
    {
        cout << "There is no first polynomial expression.\n";
        return t;
    }
    else if (second.START == NULL)
    {
        cout << "There is no second polynomial expression.\n";
        return t;
    }
    else
    {
        int c;
        node* p1, * p2;
        p1 = START;
        while (p1 != NULL)
        {
            c = 0;
            p2 = second.START;
            while (p2 != NULL)
            {
                if (p1->xp == p2->xp)
                {
                    c = 1;
                    p2->check = 1;
                    t.AddExpression((p1->info - p2->info), p1->xp);
                    break;
                }
                p2 = p2->next;
            }
            if (c == 0)
                t.AddExpression(p1->info, p1->xp);
            p1 = p1->next;

        }
        p2 = second.START;
        while (p2 != NULL)
        {
            if (p2->check != 1)
                t.AddExpression(-p2->info, p2->xp);
            p2 = p2->next;
        }
        return t;
    }
}
bool POLY::DisplayExpression()
{
    if (START == NULL)
    {
        cout << "No expression\n";
        return false;
    }
    else
    {
        node* ptr;
        ptr = START;
        cout << "The expression is :\n";
        while (ptr != NULL)
        {
            if (ptr == START && ptr->info >= 0)
                cout << ptr->info << "x^" << ptr->xp << " ";
            else if (ptr->info >= 0)
                cout << "+" << ptr->info << "x^" << ptr->xp << " ";
            else
                cout << ptr->info << "x^" << ptr->xp << " ";
            ptr = ptr->next;
        }
        cout << "\n\n";
        return true;
    }
}
int main()
{
    POLY e1, e2, e3;
    int choice, info, x, y, z;
    char ch;
    while (1)
    {
        cout << "1. Enter the first expression\n2. Enter the second expression\n3. Add
first and second expressions\n4. Subtract second expression from first expression\n5.
Display first expression\n6. Display second expression\n7. Exit\nEnter your choice : ";
        cin >> choice;
        switch (choice)
        {
        case 1:
        {
            char c = 'y';
            while (c == 'y' || c == 'Y')
            {
                cout << "Enter in the form (coeff,x pow):   \t";
                cin >> ch >> info >> ch >> x >> ch;
                e1.AddExpression(info, x);
                cout << "Want to add another term for first expression?  y/n\t";
                cin >> c;
            }
            break;
        }
        case 2:
        {
            char c = 'y';
            while (c == 'y' || c == 'Y')
            {
                cout << "Enter in the form (coeff,x pow):   \t";
                cin >> ch >> info >> ch >> x >> ch;
                e2.AddExpression(info, x);
                cout << "Want to add another term for second expression?  y/n\t";
                cin >> c;
            }
            break;
        }
        case 3:
        {

            e3 = e1 + e2;
            bool b = e3.DisplayExpression();
            break;
        }
        case 4:
        {

            e3 = e1 - e2;
            bool b = e3.DisplayExpression();
            break;
        }
        case 5:
        {
            bool b = e1.DisplayExpression();
            break;
        }
        case 6:
        {
            bool b = e2.DisplayExpression();
            break;
        }
        default:exit(0);

        }

    }
    return 0;
}
```

Output



```
1. Enter the first expression
2. Enter the second expression
3. Add first and second expressions
4. Subtract second expression from first expression
5. Display first expression
6. Display second expression
7. Exit
Enter your choice : 1
Enter in the form (coeff,x pow):        (5,3)
Want to add another term for first expression?  y/n     y
Enter in the form (coeff,x pow):        (6,2)
Want to add another term for first expression?  y/n     n
1. Enter the first expression
2. Enter the second expression
3. Add first and second expressions
4. Subtract second expression from first expression
5. Display first expression
6. Display second expression
7. Exit
Enter your choice : 2
Enter in the form (coeff,x pow):        (2,3)
Want to add another term for second expression?  y/n    y
Enter in the form (coeff,x pow):        (3,2)
Want to add another term for second expression?  y/n    n
1. Enter the first expression
2. Enter the second expression
3. Add first and second expressions
4. Subtract second expression from first expression
5. Display first expression
6. Display second expression
7. Exit
Enter your choice : 3
The expression is :
7x^3 +9x^2

1. Enter the first expression
2. Enter the second expression
3. Add first and second expressions
4. Subtract second expression from first expression
5. Display first expression
6. Display second expression
7. Exit
Enter your choice : 4
The expression is :
3x^3 +3x^2
```


```
1. Enter the first expression
2. Enter the second expression
3. Add first and second expressions
4. Subtract second expression from first expression
5. Display first expression
6. Display second expression
7. Exit
Enter your choice : 5
The expression is :
5x^3 +6x^2

1. Enter the first expression
2. Enter the second expression
3. Add first and second expressions
4. Subtract second expression from first expression
5. Display first expression
6. Display second expression
7. Exit
Enter your choice : 6
The expression is :
2x^3 +3x^2

1. Enter the first expression
2. Enter the second expression
3. Add first and second expressions
4. Subtract second expression from first expression
5. Display first expression
6. Display second expression
7. Exit
Enter your choice : 7

Process returned 0 (0x0)   execution time : 47.194 s
Press any key to continue.
```

Conclusion

Thus, we can apply linked list data structure to analyze polynomial expression and operate on any two polynomial expressions.

LAB 5
Recursion

5.1 Write a recursive program to find factorial of a number

Problem Analysis

A factorial of a number is a number obtained by multiplying all the preceding numbers, provided it is not less than 1. Factorial is only applied for non-zero positive integers. For finding out the factorial, recursion can be used. A recursive function is a self-calling function; and for finding out factorial, the number is multiplied with a function calling a number less than the given number.

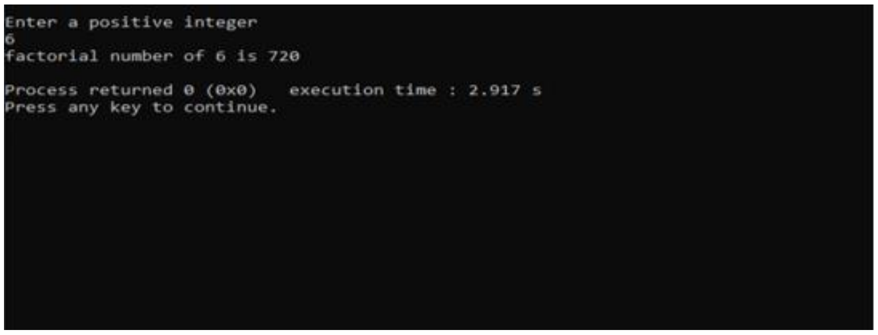
Algorithm

```
Step 1: Create a function returning unsigned int value, taking one parameter.
Step 2: if num == 0 or num == 1:
                return 1
        else:
                return num * factorial(num – 1)
Step 3: Stop.
```

Source Code

```c
#include <stdio.h>

unsigned long long int factorial(int n) {

	if(n < 0) {
		return -1;
	}
	else {
		if(n == 0 || n == 1)
			return 1;
		else {
			return (n * factorial(n - 1));
		}
	}
}

int main() {
	int input_num;
	unsigned long int fact;

	printf("Enter any number: ");
	scanf("%d", &input_num);

	fact = factorial(input_num);
	printf("Factorial of %d : %llu\n", input_num, fact);

	return 0;
}
```

Output



```
Enter a positive integer
6
factorial number of 6 is 720

Process returned 0 (0x0)   execution time : 2.917 s
Press any key to continue.
```

Conclusion

Thus, recursion was used to find factorial of a positive non-zero integer.

5.2 Write a recursive program to find N terms of a Fibonacci series.

Problem Analysis

A Fibonacci series is a series where the next term is a sum of previous two terms. For example, we take two positive integers, 0, 1. The next term is 0+1=1, and thus we have 0, 1, 1. The next term after this is 1+1=2, thus the sequence is 0, 1, 1, 2. Similarly, 1+2=3, 2+3=5, 3+5=8, and so on. This can be implemented through recursion, and the Nth term of the series can be deduced.

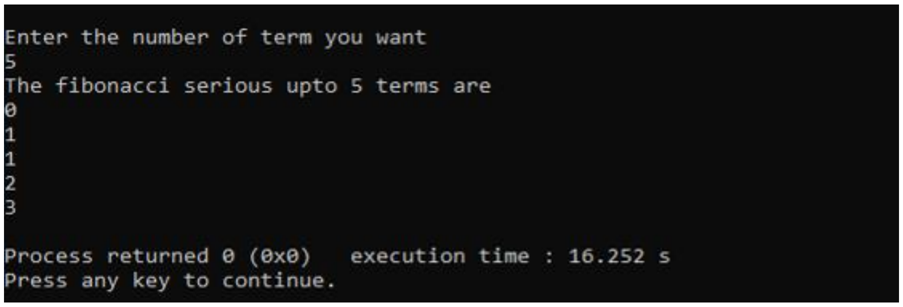
Algorithm

```
Step 1: Create a function taking one positive integer parameter.
Step 2: if num == 1:
                return 0
        else if num == 2:
                return 1
        else
                return Fibonacci(num – 2) + Fibonacci (num – 1)
Step 3: Stop
```

Source Code

```
#include <stdio.h>

unsigned long long int Fibonacci(int N) {
    if(N == 0)
        return 0;
    else if(N == 1)
        return 1;
    else
        return Fibonacci(N - 1) + Fibonacci(N - 2);
}

int main() {
    int input_num;
    unsigned long long int fibo;
    printf("Enter any number to find nth Fibonacci term: ");
    scanf("%d", &input_num);
    fibo = Fibonacci(input_num);
    printf("Fibonacci term at %d is %llu\n", input_num, fibo);
    return 0;
}
```

Output



```
Enter the number of term you want
5
The fibonacci serious upto 5 terms are
0
1
1
2
3

Process returned 0 (0x0)   execution time : 16.252 s
Press any key to continue.
```

Conclusion

Thus, we can deduce the nth term of a Fibonacci series through the use of recursion.

5.3 Write a recursive program to solve Tower of Hanoi.

Problem Analysis

Tower of Hanoi is a mathematical puzzle where we have three rods and n number of discs. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1) Only one disk can be moved at a time.

2) Only a smaller disk can be placed above a bigger disk, the opposite is strictly forbidden.

The problem here is that, we have to solve that puzzle using recursion though we cannot solve it by looping. So, this problem is typically recursion based.

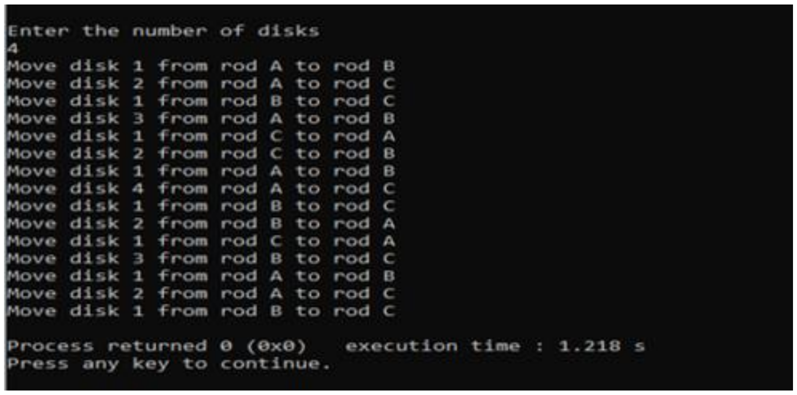
Algorithm

```
Step 1: Create a recursive function having four parameters.
                void TOH(int n, int source, int dest, int aux)
Step 2: if n == 1:
                Move disc 1 from source to dest
        else
                TOH(n-1, source, dest, aux)
                Move n from source to dest
                TOH(n-1, aux, dest, source)
Step 3: Exit
```

Source Code

```
#include <stdio.h>
#include <unistd.h>

void TOH(int N, char source, char dest, char aux) {

       if(N < 0) {
              printf("Error, invalid disc number");
              return;
       }

       // If it has only one disc
       if(N == 1) {
              printf("Move disc 1 from %c to %c \n", source, dest);
              sleep(1);
       }
       else {
          // Making C auxiliary and B destination
              TOH(N - 1, source, aux, dest);
              sleep(1);
              printf("Move disc %d from %c to %c \n", N, source, dest);
              sleep(1);

        // make  B the source and A an auxiliary position.
              TOH(N - 1, aux, dest, source);
       }
}

int main() {
    int disc_num;
    printf("Enter the number of disks: ");
    scanf("%d", &disc_num);
       TOH(disc_num, '1', '3', '2');            // 1 = source, 2 = auxillary, 3 =
destination
       return 0;
}
```

Output



```
Enter the number of disks
4
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C

Process returned 0 (0x0)   execution time : 1.218 s
Press any key to continue.
```

Conclusion

Thus, the Tower of Hanoi problem can be solved through the use of recursion.

5.4 Write a recursive program to find Greatest Common Divisor of two numbers.

Problem Analysis

The greatest common divisor of two numbers is the largest number that divides both the numbers. The GCD of two numbers can be deduced with a recursive function. First of all, whether one of the numbers is 0 is checked; if it is, then the second number is the GCD of the two numbers. The remainder is and the quotient of two numbers after division is calculated. The recursive function is then called and the second number and the remainder is passed as function parameters which will recursively check the conditions until one of them is zero.

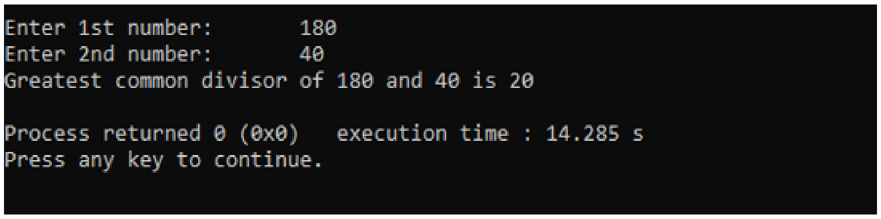
Algorithm

```
Step 1: Create a recursive function taking two numbers as parameters.
                int GCD(int n1, int n2)
Step 2: if n2 == 0:
                gcd = n1
        else:
                quotient = n1 / n2
                remainder = n1 % n2
Step 3: GCD(n2, remainder)
Step 4: Exit
```

Source Code

```c
#include <stdio.h>

int gcd(int A, int B) {
    if(B != 0)
        return gcd(B, A % B);
    else
        return A;
}

int main() {
    int in_num_1, in_num_2, hcf;

    printf("Enter any two numbers: \n");
    scanf("%d %d", &in_num_1, &in_num_2);

    hcf = gcd(in_num_1, in_num_2);
    printf("GCD of %d and %d is %d\n", in_num_1, in_num_2, hcf);

    return 0;
}
```

Output



```
Enter 1st number:       180
Enter 2nd number:       40
Greatest common divisor of 180 and 40 is 20

Process returned 0 (0x0)   execution time : 14.285 s
Press any key to continue.
```

Conclusion

Thus, we can find the greatest common divisor of any two positive integers through the use of recursion.

LAB 6
Trees

6.1 Write a menu driven program for the following operations on Binary Search Tree (BST) of integers

i. Create a BST of N integers: 5, 10, 25, 2, 8, 15, 24, 14, 7, 8, 35, 2
ii. Traverse the BST in in-order, pre-order and post-order.
iii. Search the BST for a given element (key) and print the appropriate message.
iv. Exit.

Program Analysis

In this program, we create a binary search tree by inserting a given integer and perform the different operations like traversing in pre-order , in-order and post-order and searching the BST element.

Algorithm

Pre-order

```
Step 1: while tree != null:
            tree->data = data
            preorder(tree->left)
            preorder(tree->right)
    [ END OF LOOP ]
Step 2:  Exit
```

In-order

```
Step 1: while tree != null
            inorder(tree->left)
            tree->data = data
            inorder(tree->right)
    [ END OF LOOP ]
Step 2: Exit
```

Post-order:

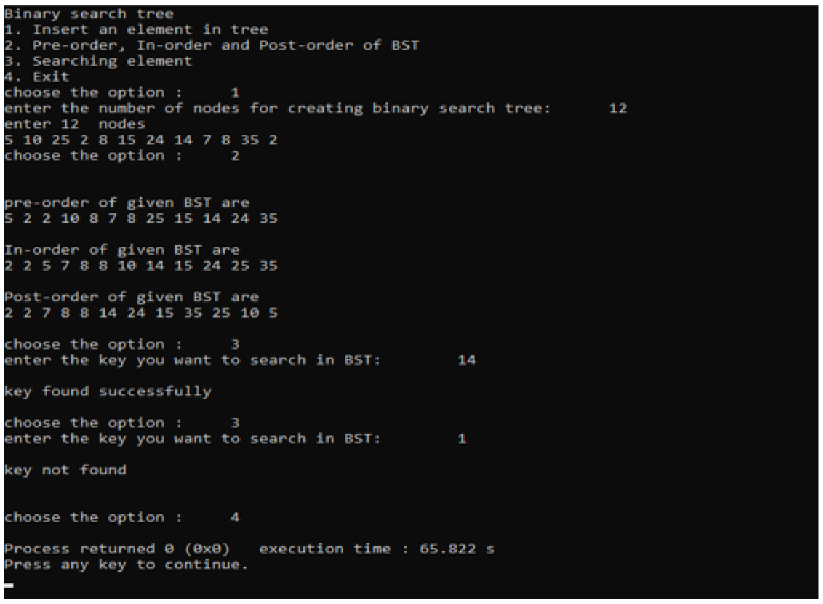
```
Step 1: while tree != null
            postorder(tree->left)
            postorder(tree->right)
            tree->data = data
      [ END OF LOOP ]

Step 2: Exit
```

Source Code:

```
#include <iostream>

struct Node {
    int data;
    struct Node *left, *right;

    Node() {
        data  = 0;
        left  = NULL;
        right = NULL;
    }
};

class BST {
    public:
        Node *rootPtr;

        BST() {
            rootPtr = NULL;
        }

        /*
            Here:
            insert() => inserting new data
            search() => searching specified data
            largest() => finding the largest node
            purge() => deleting specified data
            display_pre() => pre-order display
            display_post() => post-order display
            display_in() => in-order display
        */

        void insert(Node *rootPtr, int data);
        void search(Node *rootPtr, int data);
        Node* largest(Node *rootPtr);
        void purge(Node *rootPtr, int data);
        void display_pre(Node *rootPtr);
        void display_post(Node *rootPtr);
        void display_in(Node *rootPtr);
};
```

```cpp
// For insertion of data
void BST::insert(Node *rootPtr, int data) {
    Node *newNode;

    if(rootPtr == NULL) {
        newNode = new Node;
        newNode->data  = data;
        newNode->left  = NULL;
        newNode->right = NULL;
    }
    else {
        if(data < rootPtr->data) {
            insert(rootPtr->left, data);
        }
        else {
            insert(rootPtr->right, data);
        }
    }
}

// For searching the element
void BST::search(Node *rootPtr, int data) {
    if(rootPtr == NULL || data == rootPtr->data) {
        std::cout << "\nData not found!" << std::endl;
    }
    else {
        std::cout << rootPtr->data << " has been found" << std::endl;
    }
}

// For finding the largest
Node* BST::largest(Node *rootPtr) {
    if(rootPtr->right == NULL) {
        return rootPtr;
    }
    else {
        return largest(rootPtr->right);
    }
}

// For deleting the specified data
void BST::purge(Node *rootPtr, int data) {
    if(rootPtr == NULL) {
        std::cout << "Error, value not found" << std::endl;
    }
    else if(data < rootPtr->data) {
        purge(rootPtr->left, data);
    }
    else if(data > rootPtr->data) {
        purge(rootPtr->right, data);
    }
    else if(rootPtr->left && rootPtr->right) {
        Node *newNode = largest(rootPtr->left);
        rootPtr->data = newNode->data;
        purge(rootPtr->left, newNode->data);
    }
    else {
        Node *newNode = rootPtr;

        if(rootPtr->left == NULL && rootPtr->right == NULL) {
            rootPtr = NULL;
        }
        else if(rootPtr->left != NULL) {
            rootPtr = rootPtr->left;
        }
        else {
            rootPtr = rootPtr->right;
        }

        delete newNode;
    }
}

// Pre-order display
void BST::display_pre(Node *rootPtr) {
    if(rootPtr != NULL) {
        std::cout << " -> " << rootPtr->data;
        display_pre(rootPtr->left);
        display_pre(rootPtr->right);
    }
}

// Post-order display
void BST::display_post(Node *rootPtr) {
    if(rootPtr != NULL) {
        display_post(rootPtr->left);
        display_post(rootPtr->right);
        std::cout << "->" << rootPtr->data;
    }
}

// In-order display
void BST::display_in(Node *rootPtr) {
    if(rootPtr != NULL) {
        display_in(rootPtr->left);
        std::cout << "->" << rootPtr->data;
        display_in(rootPtr->right);
    }
}

int main() {
    BST *tree = new BST;

    int some_data, choice = 1;

    while (choice != 7) {
        std::cout << "Binary Search Tree" << std::endl << std::endl;
        std::cout << "1: Insert data " << std::endl;
        std::cout << "2: Search data" << std::endl;
        std::cout << "3: Delete data" << std::endl;
        std::cout << "4: Display pre-order" << std::endl;
        std::cout << "5: Display post-order" << std::endl;
        std::cout << "6: Display in-order" << std::endl;
        std::cout << "7: Exit" << std::endl << std::endl;
        std::cin >> choice;

        switch(choice) {
            case 1:
                std::cout << "Enter data to insert: ";
                std::cin >> some_data;
                tree->insert(tree->rootPtr, some_data);
            case 2:
                std::cout << "Enter data to search: ";
                std::cin >> some_data;
                tree->search(tree->rootPtr, some_data);
            case 3:
                std::cout << "Enter data to delete: ";
                std::cin >> some_data;
                tree->purge(tree->rootPtr, some_data);
            case 4:
                std::cout << "Pre-order display" << std::endl;
                tree->display_pre(tree->rootPtr);
            case 5:
                std::cout << "Post-order display" << std::endl;
                tree->display_post(tree->rootPtr);
            case 6:
                std::cout << "In-order display" << std::endl;
                tree->display_in(tree->rootPtr);
            case 7:
                break;
        }
    }

    delete tree;
    return 0;
}
```

Output



```
Binary search tree
1. Insert an element in tree
2. Pre-order, In-order and Post-order of BST
3. Searching element
4. Exit
choose the option :     1
enter the number of nodes for creating binary search tree:      12
enter 12  nodes
5 10 25 2 8 15 24 14 7 8 35 2
choose the option :     2


pre-order of given BST are
5 2 2 10 8 7 8 25 15 14 24 35

In-order of given BST are
2 2 5 7 8 8 10 14 15 24 25 35

Post-order of given BST are
2 2 7 8 8 14 24 15 35 25 10 5

choose the option :     3
enter the key you want to search in BST:        14

key found successfully

choose the option :     3
enter the key you want to search in BST:        1

key not found


choose the option :     4

Process returned 0 (0x0)   execution time : 65.822 s
Press any key to continue.
```

Conclusion

In this way a BST of given n integers can be created and traversed in pre-order, in-order and post-order. Search operation can also be implemented in the BST; if the element is in BST, the key is found and the appropriate message is displayed.

LAB 7
Sorting

7.1 WAP to implement Insertion Sorting Algorithm

Problem Analysis

The problem here is to sort the array values using insertion sorting algorithm. First of all, the array of values to be sorted is divided into two sets. One that stores sorted values and another that contains unsorted values. The sorting algorithm will proceed until there are elements in the unsorted set.

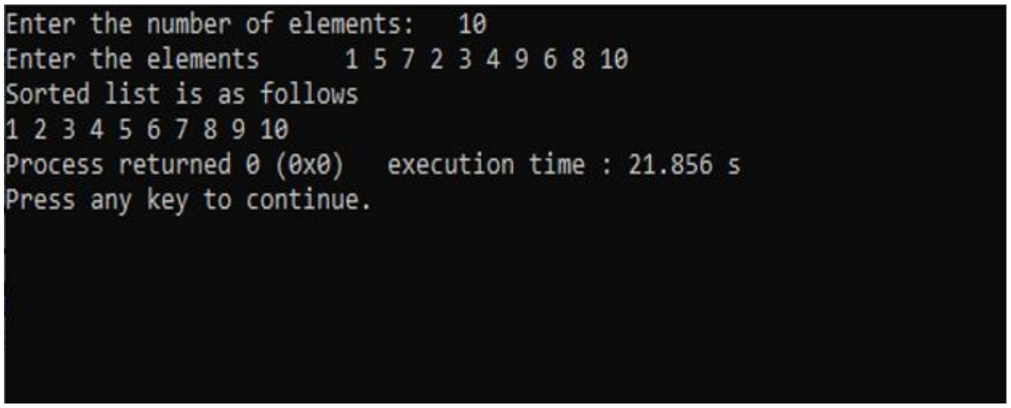
Algorithm

```
INSERTING-SORT (ARR, N)
Step 1: Repeat Steps 2 to 5 for k = 1 to N-1
Step 2: SET TEMP=ARR[K]
Step 3: SET J=K-1
Step 4: Repeat while TEMP <= ARR[J]
                SET ARR[J+1] = ARR[J]
                SET J = J-1
        [END OF INNER LOOP]
Step 5: SET ARR[J+1] = TEMP
        [END OF LOOP]
Step 6: EXIT
```

Source Code

```
#include<iostream>
using namespace std;
int main()
{
    int i, j, n, temp, a[30];
    cout << "Enter the number of elements:";
    cin >> n;
    cout << "Enter the elements\t";
    for (i = 0;i < n;i++)

    {
        cin >> a[i];
    }
    for (i = 1;i <= n - 1;i++)
    {
        temp = a[i];
        j = i - 1;

        while ((temp < a[j]) && (j >= 0))
        {
            a[j + 1] = a[j];    //moves element forward
            j = j - 1;
        }
        a[j + 1] = temp;    //insert element in proper place
    }
    cout << "Sorted list is as follows\n";
    for (i = 0;i < n;i++)
    {
        cout << a[i] << " ";
    }
    return 0;
}
```

Output



```
Enter the number of elements:   10
Enter the elements      1 5 7 2 3 4 9 6 8 10
Sorted list is as follows
1 2 3 4 5 6 7 8 9 10
Process returned 0 (0x0)   execution time : 21.856 s
Press any key to continue.
```

Conclusion

Thus, an array can be sorted through the use of insertion sort algorithm.

7.2 WAP to implement Merge Sorting Algorithm

Problem Analysis

The problem here is to sort the value of array using merge sorting algorithm. If the array is of length 0 or 1, then it is already sorted. Otherwise, divide the unsorted array into two sub-arrays of about half the size. Use merge sort algorithm recursively to sort each sub-array and Merge the two sub-arrays to form a single sorted list.

Algorithm

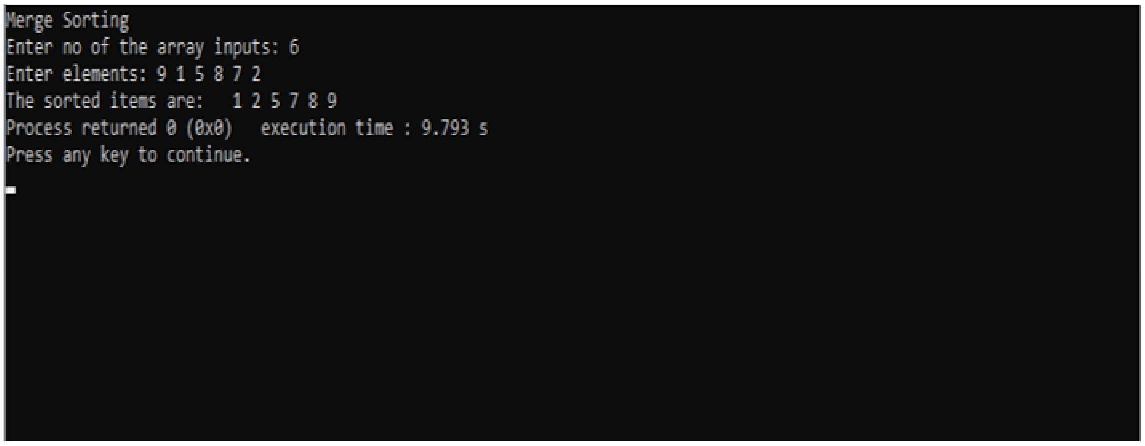
```
MERGE (ARR, BEG, MID, END)
Step 1: [INITIALIZE] SET I = BEG, J=MID+1, INDEX=0
Step 2: Repeat while (I<=MID) AND (J<=END)
                IF ARR[I] < ARR[J]
                        SET TEMP[INDEX]=ARR[I]
                        SET I = I+1
                ELSE
                        SET TEMP[INDEX]=ARR[J]
                        SET J = J+1
                [END OF IF]
                SET INDEX=INDEX+1
        [END OF LOOP]
Step 3: [Copy the remaining elements of right sub-array, if any]
                IF I > MID
                        Repeat while J<=END
                                SET TEMP[INDEX]=ARR[J]
                                SET INDEX=INDEX+1, SET J = J+1
                        [END OF LOOP]
                [Copy the remaining elements of left sub-array, if any]
                ELSE
                        Repeat while I<=END
                                SET TEMP[INDEX]=ARR[I]
                                SET INDEX=INDEX+1, SET I=I+1
                        [END OF LOOP]
                [END OF IF]
        Step 4: [Copy the contents of TEMP back to ARR] SET K=0
        Step 5: Repeat while K<INDEX
                        SET ARR[K]=TEMP[K]
                        SET K=K+1
                [END OF LOOP]
        Step 6: END

        MERGE_SORT (ARR, BEG, END)
        Step 1: IF BEG<END
                        SET MID = (BEG+END)/2
                        CALL MERGE_SORT (ARR, BIG, MID)
                        CALL MERGE_SORT (ARR, MID+1, END)
                        MERGE (ARR, BEG, MID, END)
                [END OF IF]
        Step 2: END
```

Source Code

```cpp
#include<iostream>
using namespace std;
void merge(int A[], int beg, int mid, int end) {
    int i = beg;
    int j = mid + 1;
    int index = beg;
    int temp[end + 1], k;
    while (i <= mid && j <= end) {
        if (A[i] < A[j]) {
            temp[index] = A[i];
            i++;
        }
        else {
            temp[index] = A[j];
            j++;
        }
        index++;
    }
    if (i > mid) {
        while (j <= end) {
            temp[index] = A[j];
            index++;
            j++;
        }
    }
    else {
        while (i <= mid) {
            temp[index] = A[i];
            index++;
            i++;
        }
    }
    k = beg;
    while (k < index) {
        A[k] = temp[k];
        k++;
    }
}
void merge_sort(int A[], int beg, int end) {
    int mid;
    if (beg < end) {
        mid = (beg + end) / 2;
        merge_sort(A, beg, mid);
        merge_sort(A, mid + 1, end);
        merge(A, beg, mid, end);
    }
}
int main() {
    int n;
    cout << "Merge Sorting\n";
    cout << "Enter no of the array inputs: ";
    cin >> n;
    int arr[n];
    cout << "Enter elements:\t";
    for (int i = 0;i < n;i++)
```

```
        cin >> arr[i];
    merge_sort(arr, 0, n - 1);
    cout << "The sorted items are: \t";
    for (int i = 0;i < n;i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output



```
Merge Sorting
Enter no of the array inputs: 6
Enter elements: 9 1 5 8 7 2
The sorted items are:   1 2 5 7 8 9
Process returned 0 (0x0)   execution time : 9.793 s
Press any key to continue.
```

Conclusion

Thus, array can be sorted using merge sort algorithm.