

**FINAL LAB REPORT**

**OF**

**“DATA STRUCTURE AND ALGORITHMS (CT 552) ”**

**SUBMITTED BY:**

NAME: SHOVIT NEPAL

ROLL: THA075BEI040

**SUBMITTED TO:**

ER. RAMA BASTOLA

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

THAPATHALI CAMPUS

**Marks and Signature:**

**Submission Date: 12/10/2020**

Contents

[1. Lab Work 1: Stack 8](#_Toc58526540)

[Objective(s): 8](#_Toc58526541)

[1.1 WAP for array implementation of Stack 8](#_Toc58526542)

[Problem Analysis 8](#_Toc58526543)

[Algorithm 8](#_Toc58526544)

[Code 9](#_Toc58526545)

[Output 12](#_Toc58526546)

[Discussion and Conclusion 12](#_Toc58526547)

[1.2 WAP to reverse a list using stack 13](#_Toc58526548)

[Problem Analysis 13](#_Toc58526549)

[Algorithm 13](#_Toc58526550)

[Code 13](#_Toc58526551)

[Output 16](#_Toc58526552)

[Discussion and Conclusion 16](#_Toc58526553)

[1.3 WAP to check parenthesis of algebraic expression using stack 17](#_Toc58526554)

[Problem Analysis 17](#_Toc58526555)

[Algorithm 17](#_Toc58526556)

[Code 17](#_Toc58526557)

[Output 21](#_Toc58526558)

[Discussion and Conclusion 21](#_Toc58526559)

[1.4 WAP to convert infix to postfix using Stack 22](#_Toc58526560)

[Problem Analysis 22](#_Toc58526561)

[Algorithm 22](#_Toc58526562)

[Code 23](#_Toc58526563)

[Output 27](#_Toc58526564)

[Discussion and Conclusion 28](#_Toc58526565)

[1.5 WAP to evaluate postfix expression using Stack 29](#_Toc58526566)

[Problem Analysis 29](#_Toc58526567)

[Algorithm 29](#_Toc58526568)

[Code 29](#_Toc58526569)

[Output 33](#_Toc58526570)

[Discussion and Conclusion 33](#_Toc58526571)

[1.6 WAP to convert an infix expression into a prefix expression 34](#_Toc58526572)

[Problem Analysis 34](#_Toc58526573)

[Algorithm 34](#_Toc58526574)

[Code 34](#_Toc58526575)

[Output 39](#_Toc58526576)

[Discussion and Conclusion 39](#_Toc58526577)

[1.7 WAP to evaluate a prefix expression 40](#_Toc58526578)

[Problem Analysis 40](#_Toc58526579)

[Algorithm 40](#_Toc58526580)

[Code 40](#_Toc58526581)

[Output 44](#_Toc58526582)

[Discussion and Conclusion 44](#_Toc58526583)

[2. Lab Work 2: Queue 45](#_Toc58526584)

[2.1 WAP for array implementation of Linear Queue 45](#_Toc58526585)

[Problem Analysis 45](#_Toc58526586)

[Algorithm 45](#_Toc58526587)

[Code 46](#_Toc58526588)

[Output 50](#_Toc58526589)

[Discussion and Conclusion 50](#_Toc58526590)

[2.2 WAP for implementation of Circular Queue 51](#_Toc58526591)

[Problem Analysis: 51](#_Toc58526592)

[Algorithm 51](#_Toc58526593)

[Code 52](#_Toc58526594)

[Output 56](#_Toc58526595)

[Discussion and Conclusion 56](#_Toc58526596)

[3. Lab Work 3: List 57](#_Toc58526597)

[Objective(s) 57](#_Toc58526598)

[3.1 WAP to implement contiguous list using array 57](#_Toc58526599)

[Problem Analysis 57](#_Toc58526600)

[Algorithm 57](#_Toc58526601)

[Code 60](#_Toc58526602)

[Output 68](#_Toc58526603)

[Discussion and Conclusion 68](#_Toc58526604)

[3.2 WAP for list implementation of QUEUE 69](#_Toc58526605)

[Problem Analysis: 69](#_Toc58526606)

[Algorithm: 69](#_Toc58526607)

[Code 70](#_Toc58526608)

[Output 76](#_Toc58526609)

[Discussion and Conclusion 76](#_Toc58526610)

[4. Lab Work 4: Linked List 77](#_Toc58526611)

[Objective(s) 77](#_Toc58526612)

[4.1 WAP to implement singly linked list 77](#_Toc58526613)

[Problem Analysis 77](#_Toc58526614)

[Algorithm 77](#_Toc58526615)

[Code 82](#_Toc58526616)

[Output 97](#_Toc58526617)

[Discussion and Conclusion 97](#_Toc58526618)

[4.2 WAP to implement circular linked list 98](#_Toc58526619)

[Problem Analysis 98](#_Toc58526620)

[Algorithm 98](#_Toc58526621)

[Code 100](#_Toc58526622)

[Outputs 111](#_Toc58526623)

[Discussion and Conclusion 112](#_Toc58526624)

[4.3 WAP to implement doubly linked list 113](#_Toc58526625)

[Problem Analysis 113](#_Toc58526626)

[Algorithm 113](#_Toc58526627)

[Code 118](#_Toc58526628)

[Output 130](#_Toc58526629)

[Discussion and Conclusion 130](#_Toc58526630)

[4.4 WAP to implement priority queue using linked list 131](#_Toc58526631)

[Problem Analysis 131](#_Toc58526632)

[Algorithm 131](#_Toc58526633)

[Code 132](#_Toc58526634)

[Output 137](#_Toc58526635)

[Discussion and Conclusion 137](#_Toc58526636)

[4.5 WAP to implement STACK using linked list 138](#_Toc58526637)

[Problem analysis 138](#_Toc58526638)

[Algorithm 138](#_Toc58526639)

[Code 139](#_Toc58526640)

[Output 143](#_Toc58526641)

[Discussion and Conclusion 143](#_Toc58526642)

[4.6 WAP to implement QUEUE using linked list 144](#_Toc58526643)

[Problem Analysis 144](#_Toc58526644)

[Algorithm 144](#_Toc58526645)

[Code 145](#_Toc58526646)

[Output 149](#_Toc58526647)

[Discussion and conclusion 149](#_Toc58526648)

[4.7 Write a program to store a polynomial using linked list. Also, perform addition and subtraction on two polynomials 150](#_Toc58526649)

[Problem Analysis 150](#_Toc58526650)

[Algorithm 150](#_Toc58526651)

[Code 152](#_Toc58526652)

[Output 157](#_Toc58526653)

[Discussion and Conclusion 158](#_Toc58526654)

[5. Lab Work 5: Recursion 159](#_Toc58526655)

[5.1 Write a recursive program to find factorial of a given number 159](#_Toc58526656)

[Problem Analysis 159](#_Toc58526657)

[Algorithm 159](#_Toc58526658)

[Code 159](#_Toc58526659)

[Output 160](#_Toc58526660)

[Discussion and Conclusion 160](#_Toc58526661)

[5.2 Write a recursive program to find N terms Fibonacci series 161](#_Toc58526662)

[Problem Analysis 161](#_Toc58526663)

[Algorithm 161](#_Toc58526664)

[Code 161](#_Toc58526665)

[Output 162](#_Toc58526666)

[Discussion and Conclusion 162](#_Toc58526667)

[5.3 Write a recursive program to solve Tower of Hanoi. 163](#_Toc58526668)

[Problem Analysis 163](#_Toc58526669)

[Algorithm 163](#_Toc58526670)

[Code 164](#_Toc58526671)

[Output 165](#_Toc58526672)

[Discussion and conclusion 165](#_Toc58526673)

[5.4 Write a recursive program to find Greatest Common Divisor GCD of two numbers. 166](#_Toc58526674)

[Problem Analysis 166](#_Toc58526675)

[Algorithm 166](#_Toc58526676)

[Code 166](#_Toc58526677)

[Output 167](#_Toc58526678)

[Discussion and Conclusion 167](#_Toc58526679)

[6. Lab Work 6: Trees 168](#_Toc58526680)

[6.1 Write a menu driven Program for the following operations on Binary Search Tree (BST) of Integers 168](#_Toc58526681)

[Problem Analysis 168](#_Toc58526682)

[Algorithm 168](#_Toc58526683)

[Code 172](#_Toc58526684)

[Output 179](#_Toc58526685)

[Discussion and conclusion 180](#_Toc58526686)

[7. Lab Work 7: Sorting 181](#_Toc58526687)

[Objective(s) 181](#_Toc58526688)

[7.1 WAP to implement Insertion Sorting Algorithm 181](#_Toc58526689)

[Problem Analysis 181](#_Toc58526690)

[Algorithm 181](#_Toc58526691)

[Code 182](#_Toc58526692)

[Output 183](#_Toc58526693)

[Discussion and Conclusion 183](#_Toc58526694)

[7.2 WAP to implement Merge Sorting Algorithm 184](#_Toc58526695)

[Problem Analysis 184](#_Toc58526696)

[Algorithm 184](#_Toc58526697)

[Code 186](#_Toc58526698)

[Output 188](#_Toc58526699)

[Discussion and Conclusion 188](#_Toc58526700)

# Lab Work 1: Stack

### Objective(s):

1. To be familiar with stack and its functions.
2. To work with the use of stack in different programs.

## WAP for array implementation of Stack

### Problem Analysis

The purpose is to create a Stack using a fixed size array. The functions like push, pop and peek are to be created. i.e. 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 push operation

Step 1: Start

Step 2: If top = Max-1

Display “Overflow”

[ End of IF]

Go to step 4

Step 2: Set top = top +1

Step 3: Set Stack[top] = value

Step 4: End

* For pop operation

Step 1: Start

Step 1: If top = NULL

Display “Underflow”

[End of IF]

Go to step 4

Step 2: Set val = Stack[top]

Step 3: Set top = top – 1

Step 4: End

* For peek operation

Step 1: Start

Step 1: If top = NULL

Display “Stack is empty.”

Go to step 3

Step 2: Return Stack[top]

Step 3: End

### Code

//1.WAP for array implementation of Stack

#include <iostream>

using namespace std;

#define max 5

int array[max],stack=-1;

void push()

{

int val;

if (stack==max-1)

{

cout<<"stack overflow"<<endl;

}

else

{

cout<<"enter the element you want to push in stack "<<endl;

cin>>val;

stack++;

array[stack]=val;

cout<<"pushing element is :"<<array[stack]<<endl;

}

}

void pop()

{

if(stack==-1)

{

cout<<"stack underflow"<<endl;

}

else

{

cout<<"popping element is : "<<array[stack]<<endl;

stack--;

}

}

void display()

{

if(stack==-1)

{

cout<<"empty stack"<<endl;

}

else {

cout<<"element in stack are \t";

for(int i=0;i<=stack;i++)

{

cout<<array[i];

cout<<"\t";

}

}

cout<<endl;

}

int main (){

int a;

cout<<"1. PUSH "<<endl;

cout<<"2. POP "<<endl;

cout<<"3. DISPLAY STACK "<<endl;

cout<<"4. EXIT"<<endl;

do{

cout<<"choose the option : \t \t ";

cin>>a;

switch(a)

{

case 1:push();

break;

case 2:pop();

break;

case 3:display();

break;

case 4:break;

default: cout<<"invalid input"<<endl;

}

}while(a!=4);

return 0;

}

### Output

### 

Fig: Main Menu with operations

### Discussion and Conclusion

This program hence uses the array spaces to implement the properties of a stack. However, the program can

## WAP to reverse a list using stack

### Problem Analysis

The problem is to create a stack that accepts the elements of the list and pop them to reverse the given list.

### Algorithm

Step 1: Start

Step 2: Accept a list as List.

Step 3: Push all elements of List in stack.

Step 4: Pop all elements from stack and store in List1.

Step 5: Display List1.

Step 6: End

### Code

#include<iostream>

using namespace std;

template<class T>

class stack

{

T data[10];

int top = -1;

int i;

T list[10];

public:

void list\_create()

{

cout<<"\nEnter the data to the stack: \n";

for (i = 0; i < 10; i++ )

{

cout<<"\nEnter the list data: ";

cin>>list[i];

push(list[i]);

}

}

void push(T val)

{

top++;

data[top] = val;

}

T pop()

{

T tmp;

tmp = data[top];

top--;

return tmp;

}

T peek()

{

return data[top];

}

void display\_list()

{

int i;

cout<<"\nList: ";

for(i=0; i<10;i++)

{

cout<<" "<<list[i]<<" ";

}

}

void display\_rev\_list()

{

int i;

cout<<"\nReverse List: ";

for(i=0;i<10;i++)

{

cout<<" "<<pop()<<" ";

}

}

};

int main()

{

stack<int>stackobj;

stackobj.list\_create();

stackobj.display\_list();

stackobj.display\_rev\_list();

return 0;

}

### Output

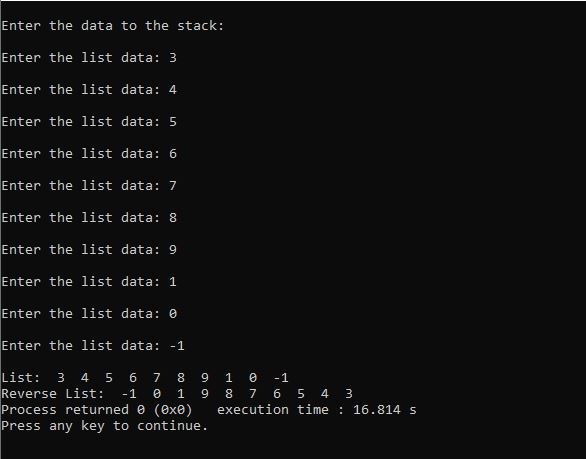


Fig: Output Screen

### Discussion and Conclusion

In this program, a list is filled with data first and the elements of the list are correspondingly pushed into the stack and pop at last. This process reverses the list.

## WAP to check parenthesis of algebraic expression using stack

### Problem Analysis

The problem is to determine the validity of given expression by checking its prantheses. The parantheses are to be pushed into and popped from the stack for the program.

### Algorithm

Step 1: Start

Step 2: Accept an algebraic expression as STR.

Step 3: Repeat step 3 while reading each char of STR

Step 4: IF opening parenthesis is found, push in the stack.

Else if closing parenthesis is found, go to step 4

Step 3.1: IF Stack is not empty, pop an element

Else Display “Use of parenthesis is incorrect.” and go to step 6

[End of IF]

[End of IF]

Step 5: Display “Use of parenthesis is correct.”

Step 6: End

### Code

#include <iostream>

#include <string>

using namespace std;

#define MAX 10

int check\_bracket(std::string expression);

class Stack

{

char bckt[MAX];

int top=-1;

public:

char push(char x)

{

if (top >= (MAX - 1))

{

cout << "\nOverflow!!";

return -1;

}

else

{

top += 1;

bckt[top] = x;

return x;

}

}

char pop()

{

if (top < 0)

{

cout<<"\nUnderflow!!";

return -1;

}

else

{

return bckt[top--];

}

}

};

bool check\_bckt(string alg\_exp)

{

Stack brackets;

for (int i = 0; i < alg\_exp.length(); i++)

{

char c = alg\_exp[i];

if (c == '(' or c == '{' or c == '[' )

{

brackets.push(c);

}

else if (c == ')' or c == '}' or c == ']' )

{

char bckt\_in\_stack;

switch (c)

{

case ')':

bckt\_in\_stack = brackets.pop();

if (bckt\_in\_stack != '(')

return false;

break;

case '}':

bckt\_in\_stack = brackets.pop();

if (bckt\_in\_stack != '{')

return false;

break;

case ']':

bckt\_in\_stack = brackets.pop();

if (bckt\_in\_stack != '[')

return false;

break;

default:

break;

}

}

else

continue;

}

return true;

}

int main()

{

string expression;

cout << "Enter an algebraic expression: ";

cin>>expression;

if (check\_bckt(expression))

cout << "\nExpression is valid.";

else

cout << "\nExpression is invalid.";

return 0;

}

### Output



Fig: Checking Parenthesis of expression

### Discussion and Conclusion

In this program, we have used the stack to check the parity of the parentheses for a given input algebraic expression.

## WAP to convert infix to postfix using Stack

### Problem Analysis

The problem is to convert a given regular infix expression to its postix form using the stack. The program requires stck to hold the operators and parantheses of the infix expression.

### Algorithm

Step 1: Start

Step 1: Add “)” to the end of the infix expression

Step 2: Push “(” on the stack

Step 3: Repeat until each character in the infix notation is scanned

IF a “(” is encountered, push it on the stack

IF an operand (whether a digit or a character) is encountered, add it to the postfix expression.

IF a “)” is encountered, then

1. Repeatedly pop from stack and add it to the postfix expression until a “(” is encountered.
2. Discard the “(”. That is, remove the “(” from stack and do not add it to the postfix expression

IF an operator 0 is encountered, then

1. Repeatedly pop from stack and add each operator (popped from the stack) to the postfix expression which has the same precedence or a higher precedence than 0
2. Push the operator 0 to the stack

[END OF IF]

Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty

Step 5: End

### Code

#include<iostream>

#include<string>

#define MAX 20

using namespace std;

class stack

{

char data[MAX];

int top = -1;

public:

void push(char oprtr)

{

if(top == MAX-1)

{

cout<<"\nStack is full!!\n";

}

else

{

top++;

data[top]=oprtr;

}

}

char pop()

{

char oprtr;

if(top == -1)

{

cout<<"\nStack is empty!!\n";

return '\0';

}

else

{

oprtr = data[top];

data[top]='\0';

top--;

return oprtr;

}

}

int priority( char oprtr)

{

if(oprtr == '+' || oprtr == '-')

{

return 1;

}

if(oprtr == '\*' || oprtr == '/')

{

return 2;

}

if(oprtr == '$')

{

return 3;

}

return 0;

}

string convert(string infix)

{

int i=0;

int j;

string postfix = "";

while(infix[i]!='\0')

{

if(infix[i] >= 'a'&&infix[i] <= 'z' || infix[i] >= 'A' && infix[i]<='Z')

{

postfix += infix[i];

i++;

}

else if(infix[i] == '(')

{

push(infix[i]);

i++;

}

else if(infix[i] == ')')

{

while(data[top] != '(')

{

postfix += pop();

}

pop();

i++;

}

else

{

if(top == -1)

{

push(infix[i]);

i++;

}

else if(priority(infix[i]) <= priority(data[top]))

{

char tmp;

if(infix[i] == '$' && data[top] == '$')

{

push(infix[i]);

i++;

}

else

{

postfix += pop();

while(priority(infix[i]) == priority(data[top]))

{

tmp = data[top];

postfix += pop();

if(top < 0)

{

break;

}

}

if(tmp = '$' && data[top] == '$')

{

postfix += pop();

}

push(infix[i]);

i++;

}

}

else if(priority(infix[i]) > priority(data[top]))

{

push(infix[i]);

i++;

}

}

}

while(top != -1)

{

postfix += pop();

}

return postfix;

}

};

int main()

{

int trash;

stack stack1;

string infix, postfix;

cout<<"\nEnter the infix expression; ";

cin>>infix;

postfix = stack1.convert(infix);

cout<<"\n The corresponding postfix expression is: "<<postfix<<endl;

cin>>trash;

return 0;

}

### Output

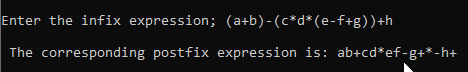


Fig: Infix to postfix

### Discussion and Conclusion

In this program, stack has been used to convert a given infix expression to its postfix form. The properties like push, pop and peek have been used to implement this program.

## WAP to evaluate postfix expression using Stack

### Problem Analysis

The problem is to evaluate the given postfix expression using the stack. In this program we user stack to store the operands until the operators encounters and store the final result.

### Algorithm

Step 1: Add a “)” at the end of the postfix expression

Step 2: Scan every character of the postfix expression and steps 3 and 4 until “)” is encountered

Step 3: IF an operand is encountered, push it on the stack

IF an operator 0 is encountered, then

1. Pop the top two elements from the stack as A and B
2. Evaluate B 0 A, where A is the topmost element and B is the element below A
3. Push the result of evaluation on the stack

[END OF IF]

Step 4: Set result equal to the topmost element of the stack

Step 5: EXIT

### Code

#include<iostream>

#include<cmath>

#include<cstring>

#define N 10

using namespace std;

template<class T>

class stack

{

T data[N];

int top= -1;

public:

void push(T x)

{

if (top >= N-1)

{

cout<<"Stack Overflow!!! Error in expression.\n";

exit(0);

}

else

{

top++;

data[top]=x;

}

}

T pop()

{

if (top <= -1)

{

cout<<"\nStack Underflow!! Error in expression";

exit(0);

}

else

{

return data[top--];

}

}

};

bool isOperator(char c)

{

if(c == '+' || c == '-' || c == '\*' || c == '/' || c == '$')

return true;

else

return false;

}

float calculate(float a, float b, char c)

{

if(c == '+')

return a+b;

else if(c == '-')

return b-a;

else if(c == '\*')

return a\*b;

else if(c == '/')

return b/a;

else if(c == '$')

return pow(b,a);

else

return -1;

}

float postfix\_evaluation(string& postfix)

{

int i;

int l= postfix.length();

float a,b;

stack<float>stackobj;

string holder = "";

for(i=0;i<=l-1;i++)

{

if(isOperator(postfix[i]))

{

a=stackobj.pop();

b=stackobj.pop();

float val = calculate(a,b,postfix[i]);

stackobj.push(val);

holder = "";

i++;

}

else

{

if(postfix[i] != '\_')

{

holder += postfix[i];

}

else

{

stackobj.push(stof(holder));

holder = "";

}

}

};

return stackobj.pop();

}

int main()

{

string postfix;

float answer;

cout<<"\nEnter the postfix expression(Use'\_'to separate term): \n";

cin>>postfix;

answer = postfix\_evaluation(postfix);

cout<<"\nThe result : "<<answer;

return 0;

}

### Output

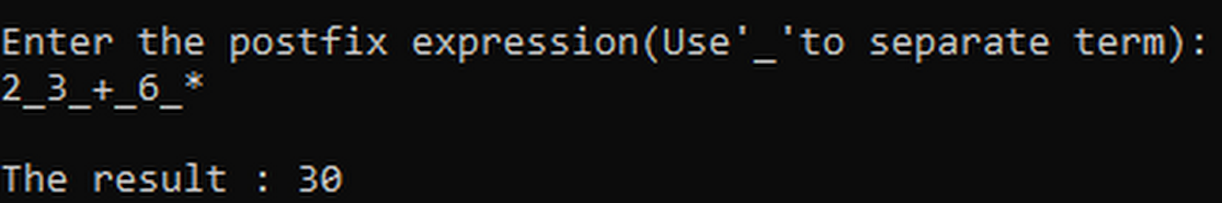


Fig: Postfix Evaluation

### Discussion and Conclusion

In this program, the stack has been used to store the postfix expressions’ operands and calculate the result for each operator occurrence in the stack itself and pop the output at end of expression.

## WAP to convert an infix expression into a prefix expression

### Problem Analysis

The problem is to convert an infix epression to a prefix using the stack. For this program, we need to derive a mirror prefix of the actual prefix and finally display the result.

### Algorithm

Step 1: Start

Step 2: Reverse the infix expression. Note that while reversing the string you must interchange left and right parenthesis.

Step 3: Obtain the postfix expression of the infix expression obtained.

Step 4: Reverse the postfix expression to get the prefix expression.

Step 5: Exit

### Code

#include<iostream>

#include<string>

#define MAX 20

using namespace std;

void reverseStr(string& str)

{

int n = str.length();

for(int i=0; i<n; i++)

{

if(str[i]=='(')

{

str[i] = ')';

}

else if(str[i] == ')')

{

str[i] ='(';

}

}

for (int i = 0; i < n / 2; i++)

{

swap(str[i], str[n - i - 1]);

}

}

class stack

{

char data[MAX];

int top = -1;

public:

void push(char oprtr)

{

if(top == MAX-1)

{

cout<<"\nStack is full!!\n";

}

else

{

top++;

data[top]=oprtr;

}

}

char pop()

{

char oprtr;

if(top == -1)

{

cout<<"\nStack is empty!!\n";

return '\0';

}

else

{

oprtr = data[top];

data[top]='\0';

top--;

return oprtr;

}

}

int priority( char oprtr)

{

if(oprtr == '+' || oprtr == '-')

{

return 1;

}

if(oprtr == '\*' || oprtr == '/')

{

return 2;

}

if(oprtr == '$')

{

return 3;

}

return 0;

}

string convert(string infix)

{

int i=0;

int j;

string prefix = "";

while(infix[i]!='\0')

{

if(infix[i] >= 'a'&&infix[i] <= 'z' || infix[i] >= 'A' && infix[i]<='Z')

{

prefix += infix[i];

i++;

}

else if(infix[i] == '(')

{

push(infix[i]);

i++;

}

else if(infix[i] == ')')

{

while(data[top] != '(')

{

prefix += pop();

}

pop();

i++;

}

else

{

if(top == -1)

{

push(infix[i]);

i++;

}

else if(priority(infix[i]) < priority(data[top]))

{

prefix += pop();

if(top <1)

{

break;

}

push(infix[i]);

i++;

}

else if(priority(infix[i]) >= priority(data[top]))

{

if(top < 0)

{

break;

}

if(infix[i] == '$' && data[top] == '$')

{

prefix += pop();

}

push(infix[i]);

i++;

}

}

}

while(top != -1)

{

prefix += pop();

}

return prefix;

}

};

int main()

{

stack stack1;

string infix, prefix;

cout<<"\nEnter the infix expression; ";

cin>>infix;

reverseStr(infix);

prefix = stack1.convert(infix);

reverseStr(prefix);

cout<<"\n The corresponding prefix expression is: "<<prefix<<endl;

return 0;

}

### Output

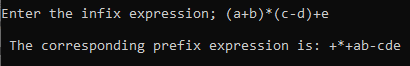


Fig: Infix to Prefix

### Discussion and Conclusion

In this program, the infix expression has been converted into its prefix expression using the stack and its properties.

## WAP to evaluate a prefix expression

### Problem Analysis

The problem is to calculate or evaluate a given prefix expression. We do so by using the stack and reversing the prefx into its mirror formand follow the similar method as postfix evaluation.

### Algorithm

Step 1: Start

Step 2: Accept the prefix expression

Step 3: Repeat until all characters in the prefix expression have been scanned

1. Scan the prefix expression from the right, one character at a time
2. If the scanned character is an operand, push it on the operand stack
3. If the scanned character is an operator, then
4. Pop two values from the operand stack
5. Apply the operator on the popped operands
6. Push the result on the operand stack

Step 4: END

### Code

#include<iostream>

#include<cmath>

#include<cstring>

#define N 10

using namespace std;

template<class T>

class stack

{

T data[N];

int top= -1;

public:

void push(T x)

{

if (top >= N-1)

{

cout<<"Stack Overflow!!! Error in expression.\n";

exit(0);

}

else

{

top++;

data[top]=x;

}

}

T pop()

{

if (top <= -1)

{

cout<<"\nStack Underflow!! Error in expression";

exit(0);

}

else

{

return data[top--];

}

}

};

bool isOperator(char c)

{

if(c == '+' || c == '-' || c == '\*' || c == '/' || c == '$')

return true;

else

return false;

}

float calculate(float a, float b, char c)

{

if(c == '+')

return a+b;

else if(c == '-')

return a-b;

else if(c == '\*')

return a\*b;

else if(c == '/')

return a/b;

else if(c == '$')

return pow(a,b);

else

return -1;

}

float prefix\_evaluation(string& prefix)

{

int i;

int l= prefix.length();

float a,b,c;

stack<float>stackobj;

string holder = "";

for(i = (l-1); i >= 0; i--)

{

if(isOperator(prefix[i]))

{

if(prefix[i] == '$' && prefix[i-1] == '$')

{

a = stackobj.pop();

b = stackobj.pop();

c = stackobj.pop();

stackobj.push(pow(pow(c,b),a));

i-=2;

holder = "";

}

else

{

a=stackobj.pop();

b=stackobj.pop();

float val = calculate(a,b,prefix[i]);

stackobj.push(val);

holder = "";

i--;

}

}

else

{

if(prefix[i] != '\_')

{

holder= prefix[i]+holder;

}

else

{

stackobj.push(stof(holder));

holder = "";

}

}

};

return stackobj.pop();

}

int main()

{

string prefix;

float answer;

cout<<"\nEnter the prefix expression(Use'\_'to separate term): \n";

cin>>prefix;;

answer = prefix\_evaluation(prefix);

cout<<"\nThe result : "<<answer;

return 0;

}

### Output

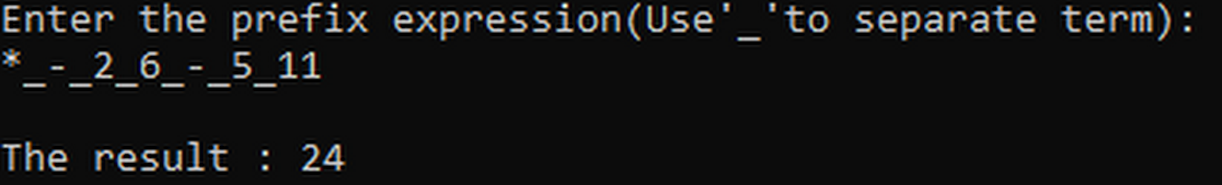


Fig: Prefix evaluation

### Discussion and Conclusion

In this program, we have used the stack to evaluate a given prefix expression. The program reverses the given expression and uses stack to hold the operands and calculate along the way to evaluate the values with each occurrence of operators.

# Lab Work 2: Queue

## WAP for array implementation of Linear Queue

### Problem Analysis

The problem is to make linear queue using array implementation. There should be basic queue operations such as enqueue and dequeue.

### Algorithm

* For Enqueue operation

Step 1: IF REAR = MAX – 1

Display “Overflow”

Go to step 4

[END OF IF]

Step 2: IF FRONT = -1 and REAR = -1

Set FRONT = REAR = 0

ELSE

Set REAR = REAR + 1

[END OF IF]

Step 3: Set QUEUE[REAR] = NUM

Step 4: EXIT

* For Dequeue operation

Step 1: IF FRONT = -1 OR FRONT > REAR

Display “Underflow”

ELSE

Set VAL = QUEUE[FRONT]

Set FRONT = FRONT + 1

[END OF IF]

Step 2: EXIT

### Code

#include<iostream>

#define MAX 10

using namespace std;

class queue

{

int data[MAX];

int FRONT, REAR;

int i;

public:

queue()

{

FRONT = -1;

REAR = -1;

}

bool IsFull()

{

if(REAR == MAX)

return true;

else

{

return false;

}

}

bool IsEmpty()

{

if(FRONT == -1 || REAR <= FRONT)

return true;

else

{

return false;

}

}

void enqueue(int a)

{

if(IsFull() == true)

{

cout<<"\nOverflow";

}

else

{

if(FRONT==-1)

{

FRONT++;

REAR++;

}

data[REAR] = a;

REAR++;

}

}

void dequeue()

{

if(IsEmpty()==true)

{

cout<<"\nUnderflow\n";

}

else

{

cout<<"\n\n\n\nThe dequeued data is:"<<data[FRONT]<<endl;

FRONT++;

}

}

void display\_queue()

{

cout<<"\nQueue: [";

for(int i = FRONT; i < REAR; i++)

{

cout<<" "<<data[i]<<" ";

}

cout<<"]\n";

}

};

int main()

{

queue queue1;

int choice, num;

do

{

cout<<"\n1. Enqueue ";

cout<<"\n2. Dequeue ";

cout<<"\n3. Exit ";

cout<<"\nEnter your choice: ";

cin>>choice;

switch (choice)

{

case (1):

cout<<"Enter the number to add to queue: ";

cin>>num;

queue1.enqueue(num);

break;

case(2):

queue1.dequeue();

break;

case(3):

break;

default:

break;

}

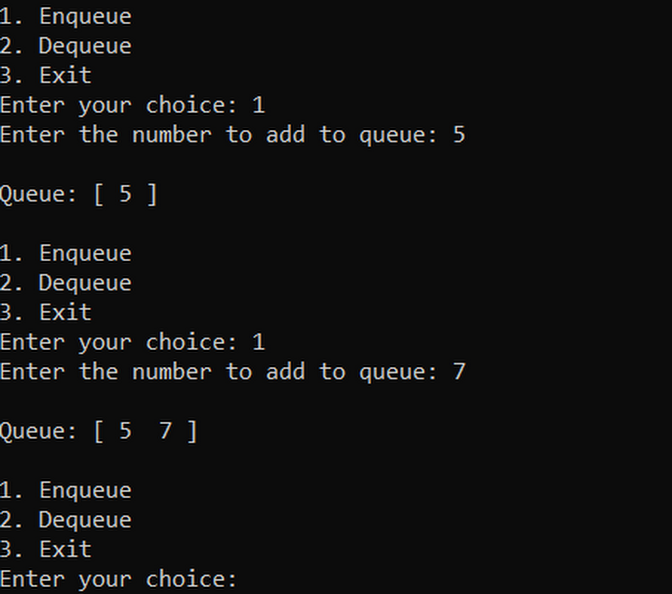
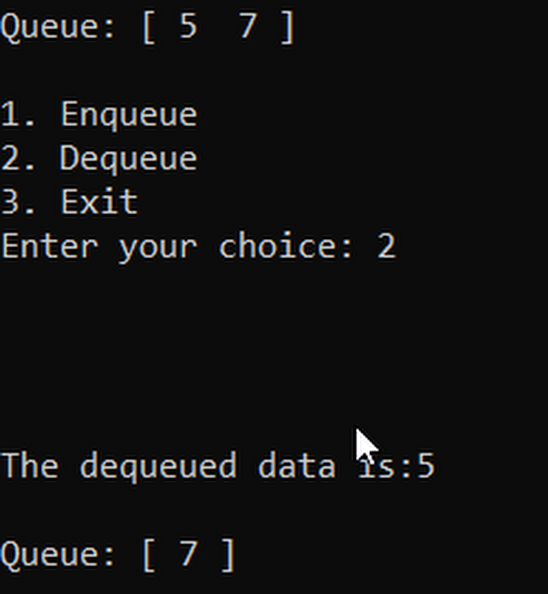
queue1.display\_queue();

} while (choice!=3);

return 0;

}

### Output

### Discussion and Conclusion

In this program, we have used the array to make a queue. The problem in this program is the a preoccupied space cannot be reused.

## WAP for implementation of Circular Queue

### Problem Analysis:

The problem is to make circular queue. There should be basic queue operations such as enqueue and dequeue.

### Algorithm

* For Enqueue operation

Step 1: IF (FRONT = 0 and REAR = MAX -1) OR FRONT=REAR+1

Display “Overflow”

Go to step 4

[END OF IF]

Step 2: IF FRONT = 1 and REAR = -1

Set FRONT = REAR = 0

ELSE IF REAR = MAX – 1 and FRONT != 0

Set REAR = 0

ELSE

Set REAR = REAR + 1

[END OF IF]

Step 3: Set QUEUE[REAR] = VAL

Step 4: EXIT

For Dequeue operation

Step 1: IF FRONT = -1

Display “Underflow”

Go to step 4

[END OF IF]

Step 2: Set VAL = QUEUE[FRONT]

Step 3: IF FRONT = REAR

Set FRONT = REAR = -1

ELSE

IF FRONT = MAX – 1

Set FRONT = 0

ELSE

Set FRONT = FRONT + 1

[END OF IF]

Step 4: EXIT

### Code

#include<iostream>

#define MAX 5

using namespace std;

class queue

{

int data[MAX];

int FRONT, REAR;

int i;

public:

queue()

{

FRONT=-1;

REAR=-1;

for(i=0;i<MAX;i++)

{

data[i] = 0;

}

}

void enqueue(int a)

{

if(FRONT == 0 && REAR == MAX-1)

{

cout<<"\nOVERFLOW\n";

}

else if (FRONT == -1 && REAR == -1)

{

FRONT=REAR=0;

data[REAR] = a;

}

else if(REAR == MAX-1 && FRONT != 0)

{

REAR = 0;

data[REAR] = a;

}

else

{

REAR++;

data[REAR] = a;

}

}

void dequeue()

{

if(FRONT == -1)

{

cout<<"\nUNDERFLOW\n";

}

else

{

cout<<"\nThe dequeued data is: "<<data[FRONT]<<endl;

data[FRONT] = 0;

}

if(FRONT == REAR)

{

FRONT = -1;

REAR = -1;

}

else if(FRONT == MAX-1)

{

FRONT = 0;

}

else

{

FRONT++;

}

}

void display\_queue()

{

cout<<"\n\nQueue: [";

for(int i = 0; i <MAX; i++)

{

cout<<" "<<data[i]<<" ";

}

cout<<"]"<<endl<<"FRONT: "<<FRONT<<endl<<"REAR: "<<REAR<<endl;

}

};

int main()

{

queue queue1;

int choice,num;

do

{

cout<<"\n1. Enqueue";

cout<<"\n2. Dequeue";

cout<<"\n3. Exit";

cout<<"\n Enter your choice: ";

cin>>choice;

switch (choice)

{

case(1):

cout<<"Enter the number to add: ";

cin>>num;

queue1.enqueue(num);

break;

case(2):

queue1.dequeue();

break;

default:

break;

}

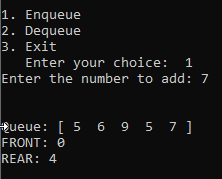
queue1.display\_queue();

} while (choice != 3);

return 0;

}

### Output



### Discussion and Conclusion

In this program a queue with a loop is created, when an element of a queue is de-queued, the space is allotted to the end for reuse hence creating an endless cycle.

# Lab Work 3: List

### Objective(s)

1. To be familiar with the properties and applications of array implementation of list.

## WAP to implement contiguous list using array

### Problem Analysis

The problem is to implement contiguous list using array implementation. There should be basic list operations such as create, insert and delete node and many others.

### Algorithm

* For getnode operation

Step 1: IF AVAIL = NULL

Display “Overflow”

Go to step 5

Step 2: Set pointer P = AVAIL

Step 3: Set AVAIL = NODE[AVAIL].NEXT

Step 4: Return P

Step 5: EXIT

* For freenode operation

Step 1: Input a pointer P

Step 2: Set NODE[P].NEXT = AVAIL

Step 3: Set AVAIL = P

Step 4: EXIT

* For creating list

Step 1: Set ROOT = getnode()

Step 2: Set NODE[ROOT] = VAL

Step 3: Set NODE[ROOT].NEXT = -1

Step 4: Insert new node at last for other given values

Step 5: End

* For inserting node at last

Step 1: IF ROOT = -1

Display “No existing list”

Go to step 8

Step 2: Set PTR = ROOT

Step3: While NODE[PTR].NEXT != -1

PTR = NODE[PTR].NEXT

Step 4: Set NODE[PTR].NEXT = getnode()

Step 5: Set PTR = NODE[PTR].NEXT

Step 6: Set NODE[PTR].INFO = VAL

Step 7: Set NODE[PTR] = -1

Step 8: End

* For deleting last node

Step 1: IF ROOT = -1

Display “No existing list”

Go to step 8

Else go to step 2

Step 2: IF NODE[ROOT].NEXT = -1

Display NODE[ROOT].INFO

Go to step 8

Else go to step 3

Step 3: Set POINT= PTR=ROOT

Step 4: While NODE[PTR].NEXT != -1

POINT=PTR

PTR= NODE[PTR].NEXT

Step 5: Set NODE[POINT].NEXT = -1

Step 6: Display NODE[PTR].INFO

Step 7: Free PTR

Step 8: End

* For inserting node after a pointer

Step 1: Get PTR and VAL

Step 2: IF PTR = -1

Display “Invalid value”

Go to step 7

Step 3: Set NEWPTR = getnode()

Step 4: Set NODE[NEWPTR].INFO = VAL

Step 5: Set NODE[NEWPTR].NEXT = NODE[PTR].NEXT

Step 6: Set NODE[PTR].NEXT = NEWPTR

Step 7: End

* For deleting node after a pointer

Step 1: Get PTR

Step 2: IF PTR = -1 OR NODE[PTR].NEXT = -1

Display “Invalid value”

Go to step 8

Step 3: Set DELPTR = NODE[PTR].NEXT

Step 4: Set DELVAL= NODE[DELPTR].INFO

Step 5: Display DELVAL

Step 6: NODE[PTR].NEXT = NODE[DELPTR].NEXT

Step 7: Free DELPTR

Step 8: End

### Code

//List using array

#include<iostream>

#define MAX 5

using namespace std;

struct nodeType

{

int data;

int next;

};

class List

{

int head = 0;

int avail =0;

nodeType node[MAX];

public:

List()

{

for(int i = 0; i < MAX; i++)

{

node[i].data = 0;

node[i].next = i+1;

}

node[MAX-1].next = -1;

}

int get\_node()

{

if (avail == -1)

{

return -1;

}

else

{

int tmp;

tmp = avail;

avail = node[avail].next;

return tmp;

}

}

void free\_node(int n)

{

node[n].data = 0;

node[n].next = avail;

avail = n;

}

void create\_list()

{

int val;

cout<<"Enter the data (-1 to stop): "<<endl;

do

{

if(avail == -1)

break;

cout<<endl<<"enter the data: ";

cin>>val;

if(val != -1)

{

insert\_node(val);

}

}while(val != -1);

}

void insert\_node(int val)

{

int p = get\_node();

if(p == -1)

{

cout<<"Overflow!!"<<endl;

}

else if(p == head)

{

node[p].data = val;

node[p].next = -1;

}

else

{

node[p].data = val;

node[p].next = -1;

int tmp = head;

while(true)

{

if(node[tmp].next == -1)

{

node[tmp].next = p;

break;

}

tmp = node[tmp].next;

}

}

}

void insert\_after(int n, int val)

{

int p = get\_node();

if(p == -1)

{

cout<<"Overflow!!"<<endl;

}

else

{

node[p].data = val;

node[p].next = node[n].next;

node[n].next = p;

}

}

void delete\_node(int n)

{

if(n == head)

{

head = node[n].next;

free\_node(n);

}

else

{

int tmp = head;

while(true)

{

if(node[tmp].next == n)

{

node[tmp].next = node[n].next;

free\_node(n);

break;

}

tmp = node[tmp].next;

}

}

}

void delete\_after(int n)

{

if(node[n].next == -1)

{

cout<<"Cannot delete!!"<<endl;

}

else if(n == head)

{

int tmp = node[head].next;

node[head].next = node[tmp].next;

free\_node(tmp);

}

else

{

int tmp =head;

while (true)

{

if(node[tmp].next == n)

{

tmp = node[n].next;

node[n].next = node[tmp].next;

free\_node(tmp);

break;

}

tmp = node[tmp].next;

}

}

}

void display()

{

cout<<endl<<"List"<<endl<<"Head = "<<head<<"\t\tAvailable = "<<avail<<endl<<endl<<"\tIndex\t\tData\t\tNext"<<endl<<endl;

for(int i=0; i<MAX; i++)

{

cout<<"\t"<<i<<"\t\t"<<node[i].data<<"\t\t"<<node[i].next<<endl;

}

}

};

int main()

{

List listobj;

int n, choice;

int val;

do

{

cout<<endl<<"1. Create List"<<endl;

cout<<"2. Insert a node"<<endl;

cout<<"3. Insert after a given node"<<endl;

cout<<"4. Delete a given node"<<endl;

cout<<"5. Delete after a given node"<<endl;

cout<<"6. Exit"<<endl;

cout<<" Enter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

{

listobj.create\_list();

break;

}

case 2:

{

cout<<endl<<"Enter the node value: ";

cin>>val;

listobj.insert\_node(val);

break;

}

case 3:

{

cout<<endl<<"Enter the node after which you want to add the node: ";

cin>>n;

cout<<"Enter the new node value: ";

cin>>val;

listobj.insert\_after(n, val);

break;

}

case 4:

{

cout<<endl<<"Enter the node to delete: ";

cin>>n;

listobj.delete\_node(n);

break;

}

case 5:

{

cout<<endl<<"Enter the node after which you want to delete a node ";

cin>>n;

listobj.delete\_after(n);

break;

}

case 6:

{

exit(0);

}

default:

{

break;

}

}

listobj.display();

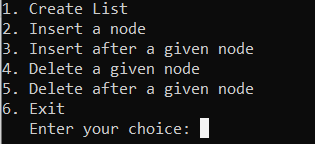
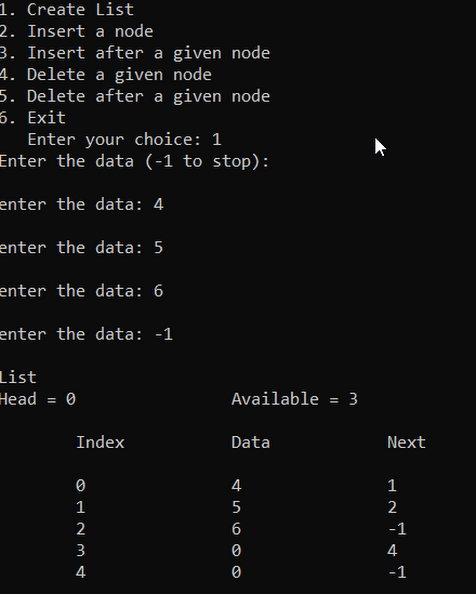
}

while(choice!=6);

return 0;

}

### Output

### Discussion and Conclusion

In this program, fixed array has been used to store the elements and replicate a list. The list stores its data along with the next index number it points to.

## WAP for list implementation of QUEUE

### Problem Analysis:

The problem is to implement a list as a queue. There should be modification of the list functions to make enqueue and dequeue functions.

### Algorithm:

* For **Enqueue** operation

Step 1: Set PTR = getnode()

Step 2: Get VAL

Step 3: Set NODE[PTR].NEXT = -1

Step 4: IF REAR = -1

FRONT = PTR

Else

NODE[REAR].NEXT = PTR

Step 5: REAR = PTR

Step 6: End

* For **Dequeue** operation

Step 1: IF REAR = -1

Display “Underflow”

Go to step 8

Step 2: Set DELVAL = NODE[FRONT].INFO

Step 3: Set PTR = FRONT

Step 4: Set FRONT = NODE[FRONT].NEXT

Step 5: IF FRONT = -1

REAR = -1

Step 6: Free PTR

Step 7: Display DELVAL

Step 8: End

### Code

#include<iostream>

#include<cstdlib>

#define MAX 5

using namespace std;

struct nodeType

{

int data;

int next;

};

class List

{

int head = 0;

int avail =0;

nodeType node[MAX];

public:

List()

{

for(int i = 0; i < MAX; i++)

{

node[i].data = 0;

node[i].next = i+1;

}

node[MAX-1].next = -1;

}

int get\_node()

{

if (avail == -1)

{

return -1;

}

else

{

int tmp;

tmp = avail;

avail = node[avail].next;

return tmp;

}

}

void create\_list()

{

int val;

cout<<endl<<"Enter the data (-1 to stop): "<<endl;

do

{

if(avail == -1)

break;

cout<<endl<<"Enter the data: ";

cin>>val;

if(val != -1)

{

insert\_node(val);

}

}while(val != -1);

}

void insert\_node(int val)

{

int p = get\_node();

if(p == -1)

{

cout<<endl<<endl<<"Overflow!!"<<endl;

}

else if(p == head)

{

node[p].data = val;

node[p].next = -1;

}

else

{

node[p].data = val;

node[p].next = -1;

int tmp = head;

while(true)

{

if(node[tmp].next == -1)

{

node[tmp].next = p;

break;

}

tmp = node[tmp].next;

}

}

}

void delete\_node()

{

int n = head;

if(node[head].data == 0 || head == -1)

{

cout<<endl<<endl<<"Underflow"<<endl;

}

else

{

head = node[head].next;

cout<<endl<<"Dequeued: "<<node[n].data<<endl;

node[n].data = 0;

node[n].next = avail;

avail = n;

}

}

void display()

{

int tmp = head;

cout<<endl<<endl<<"Head: "<<head+1<<endl;

cout<<endl<<"Queue: "<<endl<<"[";

if(node[head].data == 0 || head == -1)

{

cout<<"";

}

else

{

while(node[tmp].next != -1)

{

cout<<" "<<node[tmp].data<<" ";

tmp = node[tmp].next;

}

cout<<" "<<node[tmp].data<<" ";

}

cout<<"]"<<endl;

}

/\*void display()

{

cout<<endl<<"List"<<endl<<"Head = "<<head<<"\t\tAvailable = "<<avail<<endl<<endl<<"\tIndex\t\tData\t\tNext"<<endl<<endl;

for(int i=0; i<MAX; i++)

{

cout<<"\t"<<i<<"\t\t"<<node[i].data<<"\t\t"<<node[i].next<<endl;

}

}\*/

};

int main()

{

List listobj;

int n, choice;

int val;

do

{

cout<<endl<<"1. Create Queue"<<endl;

cout<<"2. Enqueue"<<endl;

cout<<"3. Dequeue"<<endl;

cout<<"4. Exit"<<endl;

cout<<" Enter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

{

//system("CLS");

listobj.create\_list();

break;

}

case 2:

{

//system("CLS");

cout<<endl<<"Enter the node value: ";

cin>>val;

listobj.insert\_node(val);

break;

}

case 3:

{

//system("CLS");

listobj.delete\_node();

break;

}

case 4:

{

exit(0);

}

default:

{

break;

}

}

listobj.display();

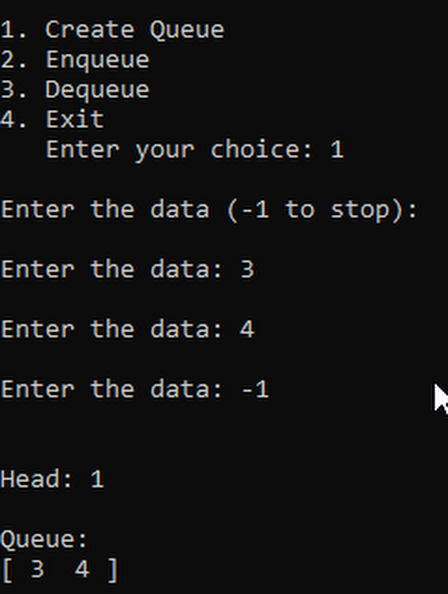
}

while(choice!=6);

return 0;

}

### Output



### Discussion and Conclusion

In this program the functions and properties of an array implementation of list are used to create a queue. The functions of list are modified to enqueue and de-queue elements.

# Lab Work 4: Linked List

### Objective(s)

1. To understand the concept of linked list, its various functions and properties.
2. To be familiar with the different types of linked lists and its modifications.

## WAP to implement singly linked list

### Problem Analysis

The problem is to make the singly linked list. The program is to be made using the pointer concepts to link the two nodes.

### Algorithm

* Traversing a linked list

Step 1: [INITIALIZE] SET PTR = START

Step 2: Repeat Steps 3 and 4 while PTR != NULL

Step 3: Apply process to PTRDATA

Step 4: SET PTR = PTR🡪NEXT

[END OF LOOP]

Step 5: EXIT

* Printing the number of nodes in a lined list

Step 1: [INTIALIZE] SET COUNT=0

Step 2: [INITIALIZE] SET PTR = START

Step 3: Repeat Steps 4 and 5 while PTR != NULL

Step 4: SET COUNT = COUNT+1

Step 5: SET PTR = PTR🡪NEXT

Step 6: write COUNT

Step 7: EXIT

* Insert a Node at the 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: IF START = NULL

SET NEW\_NODE NEXT = NULL

Otherwise,

SET NEW\_NODE NEXT = START

Step 6: SET START = NEW\_NODE

Step 7: Exit

* Insert a Node at the 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 = NULL

Step 6: IF START = NULL

SET START = NEW\_NODE

Go to step 9

Step 7: Otherwise, SET PTR = START

Step 7.1: Repeat Step 7.2 while PTR  NEXT!=NULL

Step 7.2: SET PTR = PTRNEXT

[END of While Loop]

Step 8: SET PTRNEXT = NEW\_NODE

Step 9: Exit

* Insert a new node after a node with value NUM

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 a new node before a node with value NUM

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

* Deleting the first node from linked list

Step 1: IF START = NULL

Write UNDERFLOW

Go to step 5

Step 2: SET PTR = START

Step 3: SET START = START🡪NEXT

Step 4: FREE PTR

Step 5: EXIT

* Deleting the last node from linked list

Step 1: IF START = NULL

Write UNDERFLOW

Go to step 8

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 = NULL

Step 7: FREE PTR

Step 8: EXIT

* Deleting a node after a given node

Step 1: INPUT node value in NUM

Step 2: IF START = NULL

Write UNDERFLOW

Go to step 10

Step 3: SET PTR = START

Step 4: SET PREPTR = PTR

Step 5: Repeat steps 6 and 7 while PREPTR🡪DATA != NUM

Step 6: SET PREPTR = PTR

Step 7: SET PTR = PTR🡪NEXT

Step 8: SET PREPTR🡪NEXT = PTR🡪NEXT

Step 9: FREE PTR

Step 10: EXIT

### Code

*#include<iostream>*

*#include<cstdlib>*

*using namespace std;*

*struct node*

*{*

*int data;*

*node\* next;*

*};*

*class linkedList*

*{*

*node \*head;*

*public:*

*linkedList()*

*{*

*head = NULL;*

*}*

*void create\_linkedlist()*

*{*

*int val = 0;*

*while(val != -1)*

*{*

*cout<<"\nEnter a value(-1 to stop): ";*

*cin>>val;*

*if(val != -1)*

*{*

*node \*newNode = new node;*

*newNode->data = val;*

*if(head == NULL)*

*{*

*head= newNode;*

*newNode->next = NULL;*

*}*

*else*

*{*

*node \*ptr = head;*

*while(ptr->next != NULL)*

*{*

*ptr = ptr->next;*

*}*

*ptr->next = newNode;*

*newNode->next = NULL;*

*}*

*}*

*}*

*}*

*void insert\_end\_linkedlist(int n)*

*{*

*node \*ptr = head;*

*node \*newNode = new node;*

*newNode->data = n;*

*newNode->next = NULL;*

*if(head == NULL)*

*{*

*head= newNode;*

*}*

*else*

*{*

*while(ptr->next != NULL)*

*{*

*ptr = ptr->next;*

*}*

*ptr->next = newNode;*

*}*

*}*

*void insert\_beg\_linkedlist(int n)*

*{*

*node\*tmp = new node;*

*tmp->data = n;*

*tmp-> next = head;*

*head = tmp;*

*}*

*void insert\_before\_linkedlist(int n, int val)*

*{*

*node \*newNode = new node;*

*newNode->data = val;*

*if(head->data == n)*

*{*

*insert\_beg\_linkedlist(val);*

*}*

*else*

*{*

*node \*ptr = head;*

*node \*preptr;*

*while(ptr->data != n)*

*{*

*preptr = ptr;*

*ptr = ptr->next;*

*}*

*preptr->next = newNode;*

*newNode->next = ptr;*

*}*

*}*

*void insert\_after\_linkedlist(int n, int val)*

*{*

*node \*newNode = new node;*

*newNode->data = val;*

*node \*ptr = head;*

*while (ptr->data != n)*

*{*

*ptr = ptr->next;*

*}*

*newNode->next=ptr->next;*

*ptr->next = newNode;*

*}*

*void delete\_beg\_linkedlist()*

*{*

*node \*ptr = head;*

*head = head->next;*

*delete ptr;*

*}*

*void delete\_end\_linkedlist()*

*{*

*node \*ptr =head;*

*node \*preptr = ptr;*

*while(ptr->next != NULL)*

*{*

*preptr = ptr;*

*ptr = ptr->next;*

*}*

*preptr->next = NULL;*

*delete ptr;*

*}*

*void delete\_node\_linkedlist(int n)*

*{*

*node \*ptr = head;*

*if(ptr->data == n)*

*{*

*delete\_beg\_linkedlist();*

*}*

*else*

*{*

*node\*preptr = ptr;*

*while(ptr->data != n)*

*{*

*preptr = ptr;*

*ptr = ptr->next;*

*}*

*preptr->next = ptr->next;*

*delete ptr;*

*}*

*}*

*void delete\_after\_linkedlist(int n)*

*{*

*node \*ptr= head;*

*while(ptr->data != n)*

*{*

*ptr = ptr->next;*

*}*

*if(ptr->next == NULL)*

*{*

*cout<<endl<<"No node to delete!!"<<endl;*

*}*

*else*

*{*

*node \*tmp = ptr->next;*

*ptr->next = tmp->next;*

*delete tmp;*

*}*

*}*

*void delete\_linkedlist()*

*{*

*while(head != NULL)*

*{*

*delete\_beg\_linkedlist();*

*}*

*}*

*void display\_linkedlist()*

*{*

*node \*ptr = head;*

*if(head == NULL)*

*{*

*cout<<"\nThe list is empty!!"<<endl;*

*}*

*cout<<endl<<endl;*

*while(ptr != NULL)*

*{*

*cout<<" "<<ptr->data<<" ";*

*ptr = ptr->next;*

*}*

*cout<<endl<<endl;*

*}*

*};*

*int main()*

*{*

*linkedList listobj;*

*int ch;*

*do*

*{*

*cout<<"\n\n\*\*\*\*\*\*\*LINKED LIST\*\*\*\*\*\*\*"<<endl;*

*cout<<"\n\n1. Create a linked list."<<endl;*

*cout<<"2. Insert at beginningg."<<endl;*

*cout<<"3. Insert at end."<<endl;*

*cout<<"4. Insert before a node in linked list."<<endl;*

*cout<<"5. Insert after a node in linked list."<<endl;*

*cout<<"6. Delete beginning of a linked list."<<endl;*

*cout<<"7. Delete end of a linked list."<<endl;*

*cout<<"8. Delete a node of a linked list."<<endl;*

*cout<<"9. Delete after a node of a linked list."<<endl;*

*cout<<"10. Delete a linked list."<<endl;*

*cout<<"11.Exit"<<endl;*

*cout<<"\n\nChoose option: ";*

*cin>>ch;*

*switch (ch)*

*{*

*case 1:*

*{*

*listobj.create\_linkedlist();*

*break;*

*}*

*case 2:*

*{*

*int val;*

*cout<<"\nenter the number to insert at the beginning: ";*

*cin>>val;*

*listobj.insert\_beg\_linkedlist(val);*

*break;*

*}*

*case 3:*

*{*

*int val;*

*cout<<"\nenter the number to insert at end: ";*

*cin>>val;*

*listobj.insert\_end\_linkedlist(val);*

*break;*

*}*

*case 4:*

*{*

*int n,val;*

*cout<<"\nEnter the the node value whose predecessor is to be added: ";*

*cin>>n;*

*cout<<"Enter the number to insert: ";*

*cin>>val;*

*listobj.insert\_before\_linkedlist(n,val);*

*break;*

*}*

*case 5:*

*{*

*int n,val;*

*cout<<"\nEnter the the node value whose successor is to be added: ";*

*cin>>n;*

*cout<<"Enter the number to insert: ";*

*cin>>val;*

*listobj.insert\_after\_linkedlist(n,val);*

*break;*

*}*

*case 6:*

*{*

*listobj.delete\_beg\_linkedlist();*

*break;*

*}*

*case 7:*

*{*

*listobj.delete\_end\_linkedlist();*

*break;*

*}*

*case 8:*

*{*

*int n;*

*cout<<"\nEnter the node value to delete: ";*

*cin>>n;*

*listobj.delete\_node\_linkedlist(n);*

*break;*

*}*

*case 9:*

*{*

*int n;*

*cout<<"\nEnter the node value whose succeeding value is to be deleted: ";*

*cin>>n;*

*listobj.delete\_after\_linkedlist(n);*

*break;*

*}*

*case 10:*

*{*

*listobj.delete\_linkedlist();*

*break;*

*}*

*case 11:*

*{*

*exit(1);*

*break;*

*}*

*default :*

*{*

*cout<<"Invalid input";*

*break;*

*}*

*}*

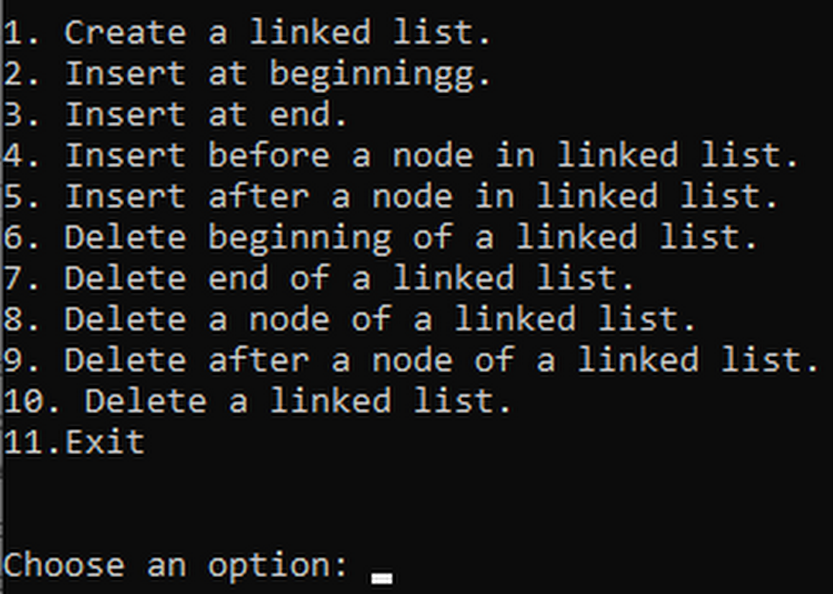
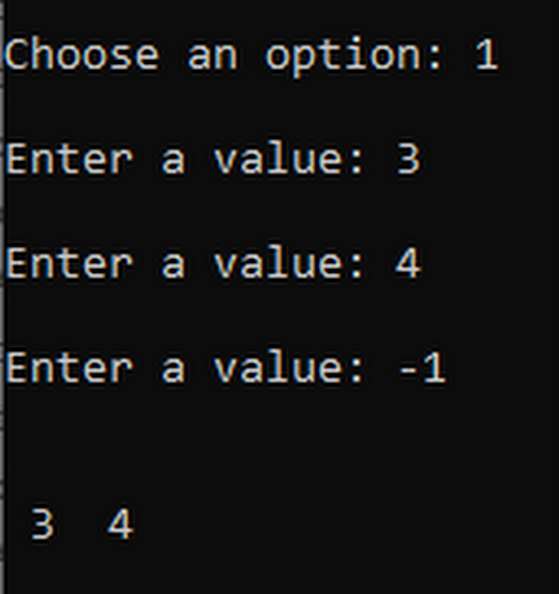
*listobj.display\_linkedlist();*

*} while (ch!=11);*

*return 0;*

*}*

### Output

*Fig: Main menu* *fig: Operation*

### Discussion and Conclusion

In this program, the list has been created with all of its properties like insertion, deletion with various ways. Each node holds its data and address of the coming node within its structure. The concept of pointers has been used in this program to connect the nodes.

## WAP to implement circular linked list

### Problem Analysis

The problem is to make a circular linked list. The circular linked list’s end node points to the starting or root node as its next node.

### Algorithm

* Inserting the node at the beginning of a circular linked list

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 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

* Inserting a node at the end of a circular linked list

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 10

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: PTR🡪NEXT = NEW\_NODE

Step 10: EXIT

* Deleting the first node of circular linked list

Step 1: IF START = NULL

Write UNDERFLOW

Go to step 8

Step 2: SET PTR = START

Step 3: Repeat step 4 and 5 while PTR->NEXT != START

Step 4: SET PTR = PTR🡪NEXT

[END OF LOOP]

Step 5: SET PTR🡪NEXT = START🡪NEXT

Step 6: FREE START

Step 7: SET START = PTR🡪NEXT

Step 8: EXIT

* Deleting the last node of a circularly linked list

Step 1: IF START = NULL

Write UNDERFLOW

Go to step 8

Step 2: SET PTR = START

Step 3: Repeat steps 4 and 5 while PTR🡪NEXT != START

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

### Code

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

node\* next;

};

class clinkedlist

{

node \*head;

public:

clinkedlist()

{

head = NULL;

}

void create\_clinkedlist()

{

int val = 0;

while(val != -1)

{

cout<<"\nEnter a value: ";

cin>>val;

if(val != -1)

{

node \*newNode = new node;

newNode->data = val;

if(head == NULL)

{

head= newNode;

newNode->next = head;

}

else

{

node \*ptr = head;

while(ptr->next != head)

{

ptr = ptr->next;

}

ptr->next = newNode;

newNode->next = head;

}

}

}

}

void insert\_end\_clinkedlist(int n)

{

node \*ptr = head;

node \*newNode = new node;

newNode->data = n;

newNode->next = head;

if(head == NULL)

{

head= newNode;

}

else

{

while(ptr->next != head)

{

ptr = ptr->next;

}

ptr->next = newNode;

newNode->next = head;

}

}

void insert\_beg\_clinkedlist(int n)

{

node \*newNode = new node;

newNode->data = n;

newNode->next = head;

node \*ptr = head;

while(ptr->next != head)

{

ptr = ptr->next;

}

ptr->next = newNode;

head = newNode;

}

void insert\_before\_clinkedlist(int n, int val)

{

node \*newNode = new node;

newNode->data = val;

if(head->data == n)

{

insert\_beg\_clinkedlist(val);

}

else

{

node \*ptr = head;

node \*preptr;

while(ptr->data != n)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = newNode;

newNode->next = ptr;

}

}

void insert\_after\_clinkedlist(int n, int val)

{

node \*newNode = new node;

newNode->data = val;

node \*ptr = head;

while (ptr->data != n)

{

ptr = ptr->next;

}

if(ptr->next == head)

{

ptr->next = newNode;

newNode->next = head;

}

else

{

newNode->next=ptr->next;

ptr->next = newNode;

}

}

void delete\_beg\_clinkedlist()

{

if(head->next == head)

{

head = NULL;

}

else

{

node \*ptr = head;

node \*tmp = head;

while(ptr->next != head)

{

ptr = ptr->next;

}

ptr->next = head->next;

head = head->next;

delete tmp;

}

}

void delete\_end\_clinkedlist()

{

node \*ptr =head;

node \*preptr = ptr;

while(ptr->next != head)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = head;

delete ptr;

}

void delete\_node\_clinkedlist(int n)

{

node \*ptr = head;

if(ptr->data == n)

{

delete\_beg\_clinkedlist();

}

else

{

node\*preptr = ptr;

while(ptr->data != n)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = ptr->next;

delete ptr;

}

}

void delete\_after\_clinkedlist(int n)

{

node \*ptr= head;

while(ptr->data != n)

{

ptr = ptr->next;

}

if(ptr->next == head)

{

delete\_beg\_clinkedlist();

}

else

{

node \*tmp = ptr->next;

ptr->next = tmp->next;

delete tmp;

}

}

void delete\_clinkedlist()

{

while(head != NULL)

{

delete\_beg\_clinkedlist();

}

}

void display\_clinkedlist()

{

node \*ptr = head;

if(head == NULL)

{

cout<<"\nThe list is empty!!"<<endl;

}

else

{

cout<<endl<<endl<<"Head: "<<head->data<<endl;

while(ptr->next != head)

{

cout<<" "<<ptr->data<<" ";

ptr = ptr->next;

}

cout<<" "<<ptr->data<<" ";

cout<<endl<<endl;

}

}

};

int main()

{

clinkedlist listobj;

int choose;

do

{

cout<<"\n\n1. Create a linked list."<<endl;

cout<<"2. Insert at beginningg."<<endl;

cout<<"3. Insert at end."<<endl;

cout<<"4. Insert before a node in linked list."<<endl;

cout<<"5. Insert after a node in linked list."<<endl;

cout<<"6. Delete beginning of a linked list."<<endl;

cout<<"7. Delete end of a linked list."<<endl;

cout<<"8. Delete a node of a linked list."<<endl;

cout<<"9. Delete after a node of a linked list."<<endl;

cout<<"10. Delete a linked list."<<endl;

cout<<"11.Exit"<<endl;

cout<<"\n\nChoose an option: ";

cin>>choose;

switch (choose)

{

case 1:

{

listobj.create\_clinkedlist();

break;

}

case 2:

{

int val;

cout<<"\nenter the number to insert at the beginning: ";

cin>>val;

listobj.insert\_beg\_clinkedlist(val);

break;

}

case 3:

{

int val;

cout<<"\nenter the number to insert at end: ";

cin>>val;

listobj.insert\_end\_clinkedlist(val);

break;

}

case 4:

{

int n,val;

cout<<"\nEnter the the node value whose predecessor is to be added: ";

cin>>n;

cout<<"Enter the number to insert: ";

cin>>val;

listobj.insert\_before\_clinkedlist(n,val);

break;

}

case 5:

{

int n,val;

cout<<"\nEnter the the node value whose successor is to be added: ";

cin>>n;

cout<<"Enter the number to insert: ";

cin>>val;

listobj.insert\_after\_clinkedlist(n,val);

break;

}

case 6:

{

listobj.delete\_beg\_clinkedlist();

break;

}

case 7:

{

listobj.delete\_end\_clinkedlist();

break;

}

case 8:

{

int n;

cout<<"\nEnter the node value to delete: ";

cin>>n;

listobj.delete\_node\_clinkedlist(n);

break;

}

case 9:

{

int n;

cout<<"\nEnter the node value whose succeeding value is to be deleted: ";

cin>>n;

listobj.delete\_after\_clinkedlist(n);

break;

}

case 10:

{

listobj.delete\_clinkedlist();

break;

}

case 11:

{

exit(1);

break;

}

default :

{

cout<<"Invalid input";

break;

}

}

system("CLS");

listobj.display\_clinkedlist();

} while (choose!=11);

return 0;

}

### Outputs

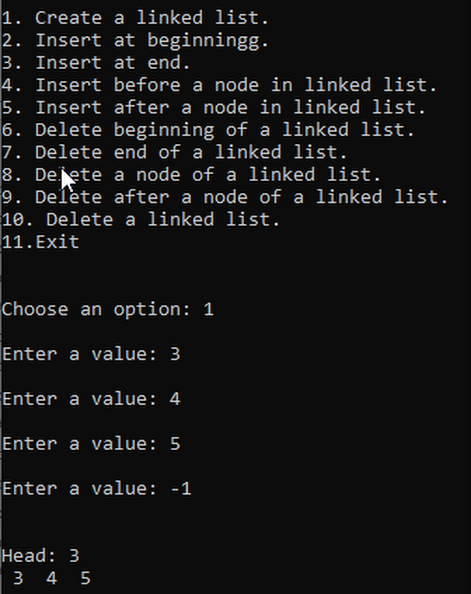


Fig: Circular linked List Menu

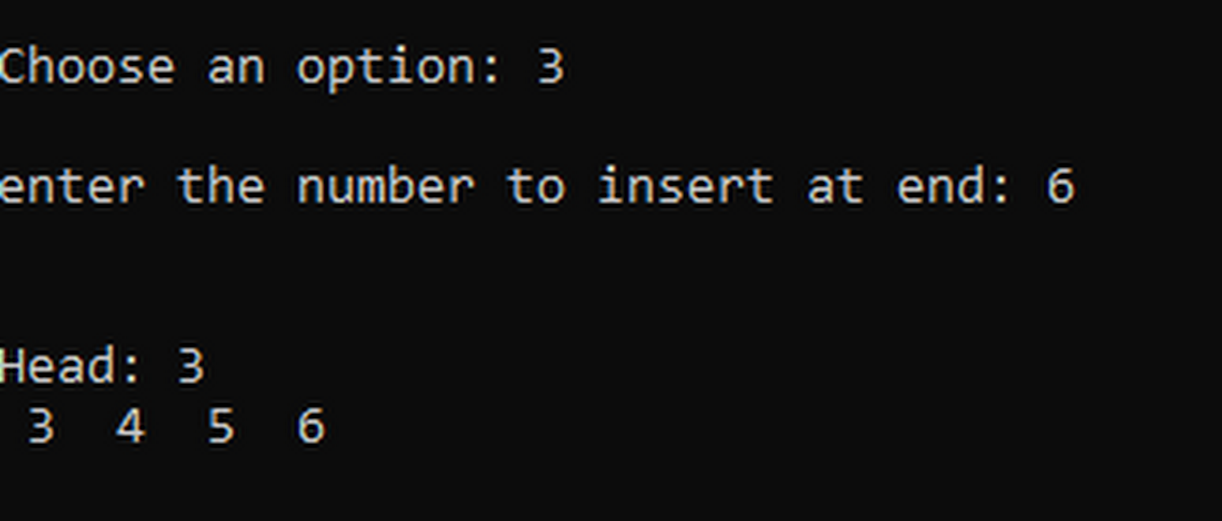


Fig: Insertion

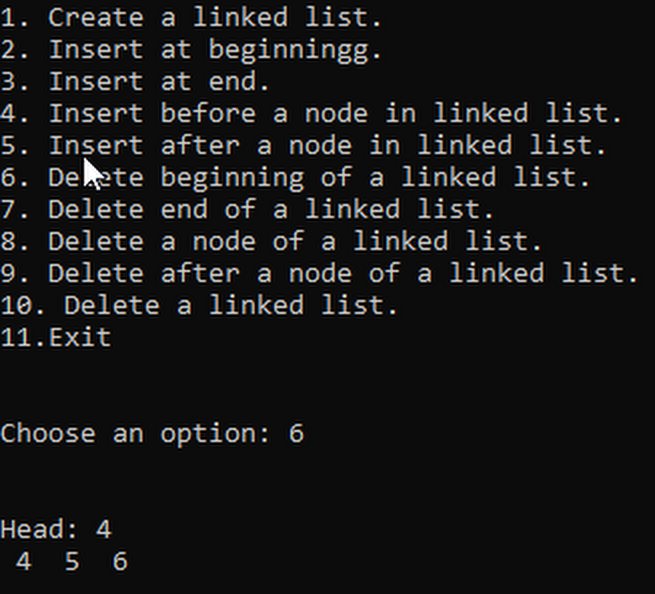


Fig: Deleting first node

### Discussion and Conclusion

In this program, linked list has been modeified to make it circular. The end node of the list points to the head or root node afterwards. As shown in the output figures, when the root node is deleted the root node changes along with the node pointed by the last node.

## WAP to implement doubly linked list

### Problem Analysis

The problem is to create a program to demonstrate a doubly linked list. Unlike singly linked list, adoubly linked list has a two way linking i.e. to both the next and the previous nodes.

### Algorithm

* Inserting a node 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 a node at the 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 a 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 a 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

Go to 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

Go to step 7

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 the node after a given node

Step 1: IF START = NULL

Write UNDERFLOW

Go to step 9

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 TMP = PTR🡪NEXT

Step 6: SET PTR🡪NEXT = TMP🡪NEXT

Step 7: SET TMP🡪NEXT🡪PREV = PTR

Step 8: FREE TMP

Step 9: EXIT

* Deleting a node before a given node

Step 1: IF START = NULL

Write UNDERFLOW

Go to step 9

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

### Code

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

node\* next;

node\* prev;

};

class dlinkedlist

{

node \*head;

public:

dlinkedlist()

{

head = NULL;

}

void create\_dlinkedlist()

{

int val = 0;

while(val != -1)

{

cout<<"\nEnter a value(-1 to stop): ";

cin>>val;

if(val != -1)

{

node \*newNode = new node;

newNode->data = val;

if(head == NULL)

{

head= newNode;

newNode->next = NULL;

newNode->prev = NULL;

}

else

{

node \*ptr = head;

while(ptr->next != NULL)

{

ptr = ptr->next;

}

ptr->next = newNode;

newNode->next = NULL;

newNode->prev = ptr;

}

}

}

}

void insert\_end\_dlinkedlist(int n)

{

node \*ptr = head;

node \*newNode = new node;

newNode->data = n;

newNode->next = NULL;

if(head == NULL)

{

head= newNode;

newNode->prev = NULL;

}

else

{

while(ptr->next != NULL)

{

ptr = ptr->next;

}

ptr->next = newNode;

newNode->prev = ptr;

}

}

void insert\_beg\_dlinkedlist(int n)

{

node\*tmp = new node;

tmp->data = n;

tmp-> next = head;

head->prev = tmp;

tmp->prev = NULL;

head = tmp;

}

void insert\_before\_dlinkedlist(int n, int val)

{

if(head->data == n)

{

insert\_beg\_dlinkedlist(val);

}

else

{

node \*newNode = new node;

newNode->data = val;

node \*ptr = head;

while(ptr->data != n)

{

ptr = ptr->next;

}

newNode->prev = ptr->prev;

ptr->prev->next = newNode;

newNode->next = ptr;

ptr->prev = newNode;

}

}

void insert\_after\_dlinkedlist(int n, int val)

{

node \*newNode = new node;

newNode->data = val;

node \*ptr = head;

while (ptr->data != n)

{

ptr = ptr->next;

}

if(ptr->next == NULL)

{

newNode->prev = ptr;

ptr->next = newNode;

newNode->next = NULL;

}

else

{

newNode->next = ptr->next;

ptr->next->prev = newNode;

newNode->prev = ptr;

ptr->next = newNode;

}

}

void delete\_beg\_dlinkedlist()

{

if(head == NULL)

{

cout<<"\nList is empty!!\n";

}

else

{

node \*ptr = head;

if(head->next == NULL)

{

head = NULL;

}

else

{

head = head->next;

head->prev = NULL;

delete ptr;

}

}

}

void delete\_end\_dlinkedlist()

{

node \*ptr =head;

while(ptr->next != NULL)

{

ptr = ptr->next;

}

if(ptr == head)

{

cout<<"\nList is empty!!\n";

}

else

{

ptr->prev->next = NULL;

delete ptr;

}

}

void delete\_node\_dlinkedlist(int n)

{

node \*ptr = head;

while(ptr->data != n)

{

ptr = ptr->next;

}

if(ptr == head)

{

delete\_beg\_dlinkedlist();

}

else if(ptr->next == NULL)

{

delete\_end\_dlinkedlist();

}

else

{

ptr->prev->next = ptr->next;

ptr->next->prev = ptr->prev;

delete ptr;

}

}

void delete\_before\_dlinkedlist(int n)

{

node \*ptr= head;

if(ptr->data == n)

{

cout<<"\nNo node to delete!\n";

}

else

{

while(ptr->next->data != n)

{

ptr = ptr->next;

}

if(ptr->data == head->data)

{

ptr->next->prev = NULL;

head = ptr->next;

delete ptr;

}

else

{

ptr->prev->next = ptr->next;

ptr->next->prev = ptr->prev;

delete ptr;

}

}

}

void delete\_after\_dlinkedlist(int n)

{

node \*ptr= head;

while(ptr->data != n)

{

ptr = ptr->next;

}

if(ptr->next == NULL)

{

cout<<"\nNO node to delete after\n";

}

else

{

ptr = ptr->next;

if(ptr->next == NULL)

{

delete\_end\_dlinkedlist();

}

else

{

ptr->prev->next = ptr->next;

ptr->next->prev = ptr->prev;

delete ptr;

}

}

}

void delete\_dlinkedlist()

{

while(head->next != NULL)

{

delete\_beg\_dlinkedlist();

}

delete\_beg\_dlinkedlist();

}

void display\_dlinkedlist()

{

node \*ptr = head;

if(head == NULL)

{

cout<<"\nThe list is empty!!"<<endl;

}

else

{

cout<<"List:\n";

while(ptr != NULL)

{

cout<<" "<<ptr->data<<" ";

ptr = ptr->next;

}

cout<<endl<<endl;

}

}

};

int main()

{

dlinkedlist listobj;

int choose;

do

{

cout<<"\n\n1. Create a doubly linked list."<<endl;

cout<<"2. Insert a node at beginningg."<<endl;

cout<<"3. Insert a node at end."<<endl;

cout<<"4. Insert a node before a given node."<<endl;

cout<<"5. Insert a node after a given node."<<endl;

cout<<"6. Delete a node at beginning."<<endl;

cout<<"7. Delete a node at last."<<endl;

cout<<"8. Delete a given node."<<endl;

cout<<"9. Delete a node before a given node."<<endl;

cout<<"10. Delete a node after a given node."<<endl;

cout<<"11. Delete the entire doubly linked list."<<endl;

cout<<"12. Exit"<<endl;

cout<<"\n\n\tChoose an option: ";

cin>>choose;

switch (choose)

{

case 1:

{

listobj.create\_dlinkedlist();

break;

}

case 2:

{

int val;

cout<<"\nenter the number to insert at the beginning: ";

cin>>val;

listobj.insert\_beg\_dlinkedlist(val);

break;

}

case 3:

{

int val;

cout<<"\nenter the number to insert at end: ";

cin>>val;

listobj.insert\_end\_dlinkedlist(val);

break;

}

case 4:

{

int n,val;

cout<<"\nEnter the the node value whose predecessor is to be added: ";

cin>>n;

cout<<"Enter the number to insert: ";

cin>>val;

listobj.insert\_before\_dlinkedlist(n,val);

break;

}

case 5:

{

int n,val;

cout<<"\nEnter the the node value whose successor is to be added: ";

cin>>n;

cout<<"Enter the number to insert: ";

cin>>val;

listobj.insert\_after\_dlinkedlist(n,val);

break;

}

case 6:

{

listobj.delete\_beg\_dlinkedlist();

break;

}

case 7:

{

listobj.delete\_end\_dlinkedlist();

break;

}

case 8:

{

int n;

cout<<"\nEnter the node value to delete: ";

cin>>n;

listobj.delete\_node\_dlinkedlist(n);

break;

}

case 9:

{

int n;

cout<<"\nEnter the node value whose preceeding value is to be deleted: ";

cin>>n;

listobj.delete\_before\_dlinkedlist(n);

break;

}

case 10:

{

int n;

cout<<"\nEnter the node value whose succeeding value is to be deleted: ";

cin>>n;

listobj.delete\_after\_dlinkedlist(n);

break;

}

case 11:

{

listobj.delete\_dlinkedlist();

break;

}

case 12:

{

exit(1);

break;

}

default :

{

cout<<"Invalid input";

break;

}

}

system("CLS");

listobj.display\_dlinkedlist();

} while (choose!=12);

return 0;

}

### Output

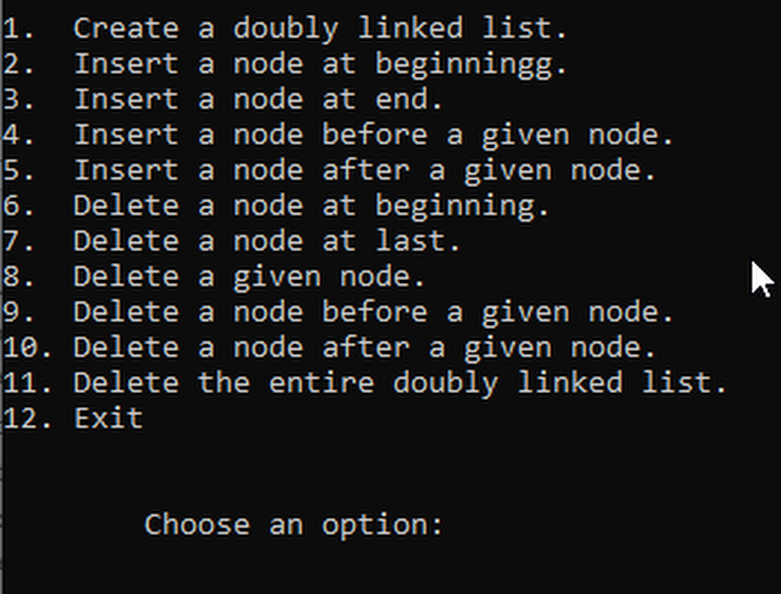


Fig: Doubly linked list and its functions

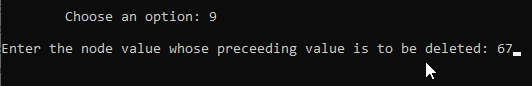


Fig: Deleting a node before given node



Fig: List elements after previous figure operation

### Discussion and Conclusion

In this program, unlike an ordinary linked list, a node contains the address of its next as well as the previous node. This property helps the faster implementationof insertion and deletion of the nodes.

## WAP to implement priority queue using linked list

### Problem Analysis

The problem is to crate a priority queue using the functions of linked list. In this program, priority queue’s functions are the modifications of the linked list.

### Algorithm

* Enqueue data

Step 1: Create new node NEW\_NODE with DATA and PRIORITY

Step 2: IF START🡪PRIORITY > PRIORITY

Step 2.1: NEW🡪NEXT = START

Step 2.2: START = NEW\_NODE

Step 2.3: Go to step

Step 5: SET TEMP = START

Step 6: Repeat step 7 while TEMP🡪NEXT != NULL and TEMP🡪NEXT🡪PRIORITY > PRIORITY

Step 7: TEMP = TEMP🡪NEXT

[END OF LOOP]

Step 8: NEW\_NODE🡪NEXT = TEMP🡪NEXT

Step 9: TEMP🡪NEXT = NEW\_NODE

Step 10: EXIT

* Dequeue a data

Step 1: IF START = NULL

Write UNDERFLOW

Go to step 6

Step 2: SET PTR = START

Step 3: START = START🡪NEXT.

Step 4: SET VAL = PTR🡪DATA

Step 5: FREE PTR

Step 6: END

### Code

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

int priority;

node\* next;

};

class prqueue

{

node \*head;

public:

prqueue()

{

head = NULL;

}

void enqueue(int n, int priority)

{

node \*newNode = new node;

newNode->data = n;

newNode->priority = priority;

if(head == NULL)

{

head = newNode;

head->next = NULL;

}

else

{

node \*ptr = head;

node \*preptr = NULL;

while(ptr->priority < priority)

{

preptr = ptr;

if(ptr->next == NULL)

{

break;

}

ptr = ptr->next;

}

if(preptr == NULL)

{

newNode->next = head;

head = newNode;

}

else if(priority <= ptr->priority)

{

preptr->next = newNode;

newNode->next = ptr;

}

else

{

newNode->next = ptr->next;

ptr->next = newNode;

}

}

}

void dequeue()

{

node \*ptr = head;

cout<<endl<<"The dequeued data is: "<<head->data<<endl;

head = head->next;

delete ptr;

}

void display\_prqueue()

{

if(head == NULL)

{

cout<<"\nThe list is empty!!"<<endl;

}

else

{

cout<<endl<<endl;

node \*ptr = head;

while(ptr != NULL)

{

cout<<" "<<ptr->data<<" ";

ptr = ptr->next;

}

cout<<endl<<endl;

}

}

};

int main()

{

prqueue queueobj;

int choose;

do

{

fflush(stdin);

cout<<"1. Enqueue."<<endl;

cout<<"2. Dequeue"<<endl;

cout<<"3. Exit"<<endl;

cout<<"\n\n\tChoose an option: ";

cin>>choose;

switch (choose)

{

case 1:

{

int val, priority;

char trash;

cout<<"\nEnter push val,priority: ";

cin>>val>>trash>>priority;

queueobj.enqueue(val,priority);

break;

}

case 2:

{

queueobj.dequeue();

break;

}

case 3:

{

exit(1);

break;

}

default :

{

cout<<"Invalid input";

break;

}

}

queueobj.display\_prqueue();

} while (choose != 3);

return 0;

}

### Output

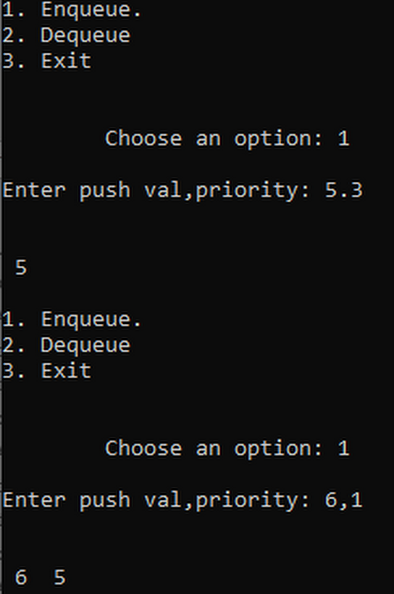


Fig: 2Priority queue

### Discussion and Conclusion

In this program, a linked list has been used to create a priority queue. The functions of the linked list are modified to replicate the functions of priority queue. Enquening and Dequeuing are done on a priority basis.

## WAP to implement STACK using linked list

### Problem analysis

The problem is to implement stack using the lik list. In this program, we need to modify the functions of linked list to replicate the functions of stack like push, pop and peek.

### Algorithm

* Push to stack

Step 1: Allocate memory for the new node and name it as NEW\_NODE

Step 2: SET NEW\_NODE🡪DATA = VAL

Step 3: IF TOP = NULL

Write 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 from stack

Step 1: IF TOP = NULL

Write UNDERFLOW

Go to step 5

[END OF IF]

Step 2: SET PTR = TOP

Step 3: SET TOP = TOP🡪NEXT

Step 4: FREE PTR

Step 5: END

* Peek from stack

Step 1: IF TOP = NULL

PRINT “UNDERFLOW”

Goto step 5

[ End of IF]

Step 2: SET PTR= TOP

Step 3 : VAL = PTR🡪DATA

Step 4: PRINT VAL

Step 5: END

### Code

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

node\* next;

};

class Stack

{

node \*head;

public:

Stack()

{

head = NULL;

}

void push()

{

int val;

cout<<endl<<endl<<"enter the value to push: ";

cin>>val;

node \*newNode = new node;

newNode->data = val;

if(head == NULL)

{

head = newNode;

head->next = NULL;

}

else

{

newNode->next = head;

head = newNode;

}

}

void pop()

{

if(head == NULL)

{

cout<<endl<<"No value to pop."<<endl;

}

else

{

node \*tmp = head;

head = head->next;

cout<<endl<<"the popped value: "<<tmp->data<<endl;

delete tmp;

}

}

void peek()

{

if(head == NULL)

{

cout<<endl<<"No value to peek"<<endl;

}

else

{

cout<<endl<<"The value at top: "<<head->data<<endl;

}

}

void display\_stack()

{

cout<<endl<<endl<<"Stack: "<<endl;

node \*ptr = head;

while(ptr != NULL)

{

cout<<" "<<ptr->data<<" ";

ptr = ptr->next;

}

cout<<endl<<endl<<endl;

}

};

int main()

{

Stack stackobj;

int choose;

do

{

cout<<"\n\n1. Push."<<endl;

cout<<"2. Pop."<<endl;

cout<<"3. Peek."<<endl;

cout<<"4.Exit"<<endl;

cout<<"\n\nChoose an option: ";

cin>>choose;

switch (choose)

{

case 1:

{

stackobj.push();

break;

}

case 2:

{

stackobj.pop();

break;

}

case 3:

{

stackobj.peek();

break;

}

case 4:

{

exit(1);

break;

}

default :

{

cout<<"Invalid input";

break;

}

}

system("CLS");

stackobj.display\_stack();

} while (choose!=4);

return 0;

}

### Output

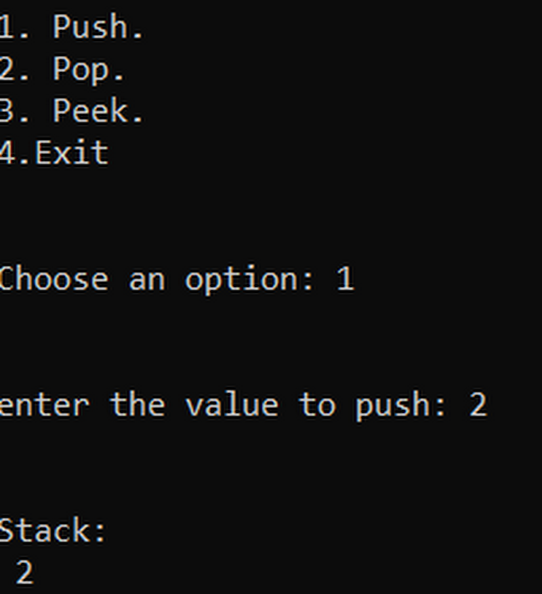


Fig: Stack Using Linked List

### Discussion and Conclusion

In this program, the functions of linked list are modifies to crates the push, pop and peek funtions of a stack.

## WAP to implement QUEUE using linked list

### Problem Analysis

The problem is to make a program for queue using the functions of the linked list. We needto modify the list’s functions to make the fucntions replicating a queue.

### Algorithm

* Enqueue an element

Step 1: Allocate memry 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: EXIT

* Dequeue an element

Step 1: IF FRONT = NULL

Write UNDERFLOW

Go to step 5

Step 2: SET PTR = FRONT

Step 3: SET FRONT = FRONT🡪NEXT

Step 4: FREE PTR

Step 5: EXIT

### Code

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

node\* next;

};

class prqueue

{

node \*head;

public:

prqueue()

{

head = NULL;

}

void enqueue(int n)

{

node \*ptr = head;

node \*newNode = new node;

newNode->data = n;

if(head == NULL)

{

head= newNode;

newNode->next = NULL;

}

else

{

while(ptr->next != NULL)

{

ptr = ptr->next;

}

newNode->next = ptr->next;

ptr->next = newNode;

}

}

void dequeue()

{

if(head == NULL)

{

cout<<endl<<endl<<"Underflow!!"<<endl;

}

else

{

node \*ptr = head;

cout<<endl<<"The dequeued data is: "<<head->data<<endl;

head = head->next;

delete ptr;

}

}

void display\_prqueue()

{

if(head == NULL)

{

cout<<"\nThe list is empty!!"<<endl;

}

else

{

cout<<endl<<endl;

node \*ptr = head;

while(ptr != NULL)

{

cout<<" "<<ptr->data<<" ";

ptr = ptr->next;

}

cout<<endl<<endl;

}

}

};

int main()

{

prqueue queueobj;

int choose;

do

{

fflush(stdin);

cout<<"1. Enqueue."<<endl;

cout<<"2. Dequeue"<<endl;

cout<<"3. Exit"<<endl;

cout<<"\n\n\tChoose an option: ";

cin>>choose;

switch (choose)

{

case 1:

{

int val;

cout<<"\nEnter push val: ";

cin>>val;

queueobj.enqueue(val);

break;

}

case 2:

{

queueobj.dequeue();

break;

}

case 3:

{

exit(1);

break;

}

default :

{

cout<<"Invalid input";

break;

}

}

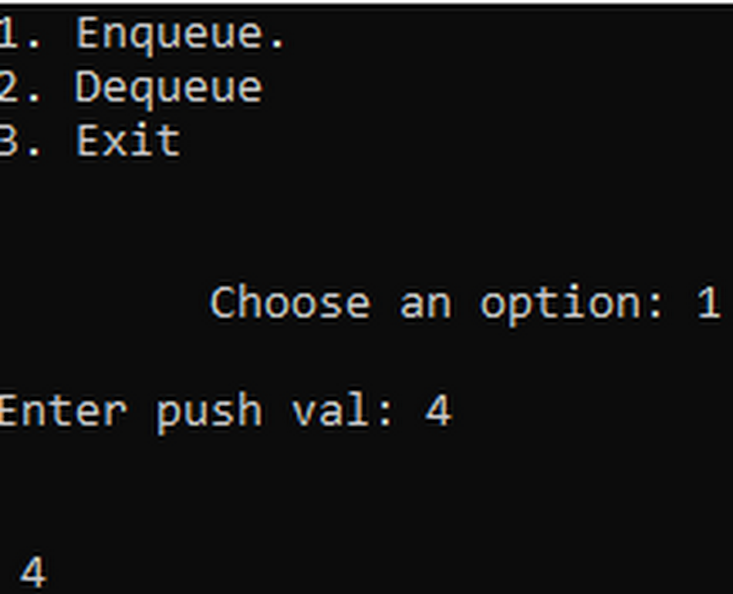
queueobj.display\_prqueue();

} while (choose != 3);

return 0;

}

### Output



### Discussion and conclusion

In this program, the functions of the linked list are modified to replicate the functions of a queue.This program has the functions enqueue, dequeuer like a regular QUEUE program.

## Write a program to store a polynomial using linked list. Also, perform addition and subtraction on two polynomials

### Problem Analysis

The problem is to make a program using the linked list to perform the addition and subtraction of the given algebraic expressions. We need to use link list to link two terms of a polynomial and a nose to store a term.

### Algorithm

* For adding terms in polynomial

Step 1: Accept the new term as VAL and POW

Step 2: Create NEW\_NODE

Step 3: IF NEW\_NODE = NULL

                 Display “Failed to initialize memory”

                 Go to step 11

Step 4: Set NEW\_NODE🡪INFO = VAL

Step 5: Set NEW\_NODE🡪POWER = POW

Step 6: Set NEW\_NODE🡪NEXT = NULL

Step 7: IF START = NULL

                 START = NEW\_NODE

                 Go to step 11

            ELSE , Go to step 8

Step 8: Set PTR = START

Step 9: While PTR🡪NEXT != NULL

                        PTR = PTR🡪NEXT

Step 10: Set PTR🡪NEXT = NEW\_NODE

Step 11: END

* For adding two polynomials

Step 1: Accept two polynomials POLY1 and POLY2

Step 2: Copy POLY1 to POLY

Step 3: Search the term with equal power for each term of

POLY in POLY2 and add them if found.

Step 4: Copy all the remaining terms of POLY2 to POLY

Step 5: Display POLY

Step 6: END

* For subtracting two polynomials

Step 1: Accept two polynomials POLY1 and POLY2

Step 2: Copy POLY1 to POLY

Step 3: Search the term with equal power for each term of

POLY in POLY2 and subtract them if found.

Step 4: Inverse the sign of all the remaining terms of

POLY2 and copy them to POLY

Step 5: Display POLY

Step 6: END

### Code

#include<iostream>

#include<cmath>

using namespace std;

struct Term

{

float coef;

float x\_index;

float y\_index;

float z\_index;

Term \*next;

};

class Polynomial

{

public:

Term \*head;

Polynomial()

{

head = NULL;

}

void assign\_polynomial(float c, float x, float y, float z)

{

Term \*newTerm= new Term;

newTerm->coef = c;

newTerm->x\_index = x;

newTerm->y\_index = y;

newTerm->z\_index = z;

if(head == NULL)

{

head = newTerm;

head->next = NULL;

}

else

{

Term \*ptr = head;

while(ptr->next != NULL)

{

ptr = ptr->next;

}

ptr->next = newTerm;

newTerm->next = NULL;

}

}

Polynomial& operator+(Polynomial& P2)

{

Polynomial P3;

Term \*ptr2 = P2.head;

while(ptr2 != NULL)

{

Term \*ptr1 = head;

while(ptr1 != NULL)

{

if(ptr1->x\_index == ptr2->x\_index && ptr1->y\_index == ptr2->y\_index && ptr1->z\_index == ptr2->z\_index )

{

ptr1->coef = ptr1->coef + ptr2->coef;

break;

}

ptr1 = ptr1->next;

}

if(ptr1 == NULL)

{

this->assign\_polynomial((ptr2->coef),ptr2->x\_index, ptr2->y\_index, ptr2->z\_index);

}

ptr2 = ptr2->next;

}

return \*this;

}

Polynomial& operator-(Polynomial& P2)

{

Polynomial P3;

Term \*ptr2 = P2.head;

while(ptr2 != NULL)

{

Term \*ptr1 = head;

while(ptr1 != NULL)

{

if(ptr1->x\_index == ptr2->x\_index && ptr1->y\_index == ptr2->y\_index && ptr1->z\_index == ptr2->z\_index )

{

ptr1->coef = ptr1->coef - ptr2->coef;

break;

}

ptr1 = ptr1->next;

}

if(ptr1 == NULL)

{

this->assign\_polynomial((ptr2->coef),ptr2->x\_index, ptr2->y\_index, ptr2->z\_index);

}

ptr2 = ptr2->next;

}

return \*this;

}

void display()

{

Term \*ptr = head;

while(ptr != NULL)

{

if(ptr->coef >= 0)

{

cout<<"+("<<ptr->coef<<"(x^"<<ptr->x\_index<<")(y^"<<ptr->y\_index<<")(z^"<<ptr->z\_index<<")) ";

}

else

{

cout<<"("<<ptr->coef<<"(x^"<<ptr->x\_index<<")(y^"<<ptr->y\_index<<")(z^"<<ptr->z\_index<<")) ";

}

ptr = ptr->next;

}

}

};

int main()

{

Polynomial POLY1,POLY2;

char choice ='y';

char trash;

float coef,x\_index,y\_index,z\_index;

cout<<endl<<endl<<"First polynomial"<<endl;

while(true)

{

cout<<"Enter coef, x\_index, y\_index, z\_index: "<<endl;

cin>>coef>>trash>>x\_index>>trash>>y\_index>>trash>>z\_index;

POLY1.assign\_polynomial(coef,x\_index,y\_index,z\_index);

cout<<endl<<"Add more?(y/n): ";

cin>>choice;

if(choice == 'n')

{

break;

}

}

choice = 'y';

cout<<endl<<endl<<"Second polynomial: "<<endl;

while(true)

{

cout<<endl<<"Enter coef, x\_index, y\_index, z\_index: "<<endl;

cin>>coef>>trash>>x\_index>>trash>>y\_index>>trash>>z\_index;

POLY2.assign\_polynomial(coef,x\_index,y\_index,z\_index);

cout<<endl<<"Add more?(y/n): ";

cin>>choice;

if(choice == 'n')

{

break;

}

}

cout<<"1. Add"<<endl;

cout<<"2. Subtract"<<endl;

cout<<"3. Exit"<<endl;

int option;

cout<<"Enter the option:";

cin>>option;

switch(option)

{

case 1:

POLY1 = POLY1 + POLY2;

cout<<endl<<"Sum= ";

POLY1.display();

break;

case 2:

POLY1 = POLY1 - POLY2;

cout<<endl<<"Difference= ";

POLY1.display();

break;

case 3:

exit(1);

break;

default:

cout<<endl<<"Error input"<<endl;

break;

}

return 0;

}

### Output

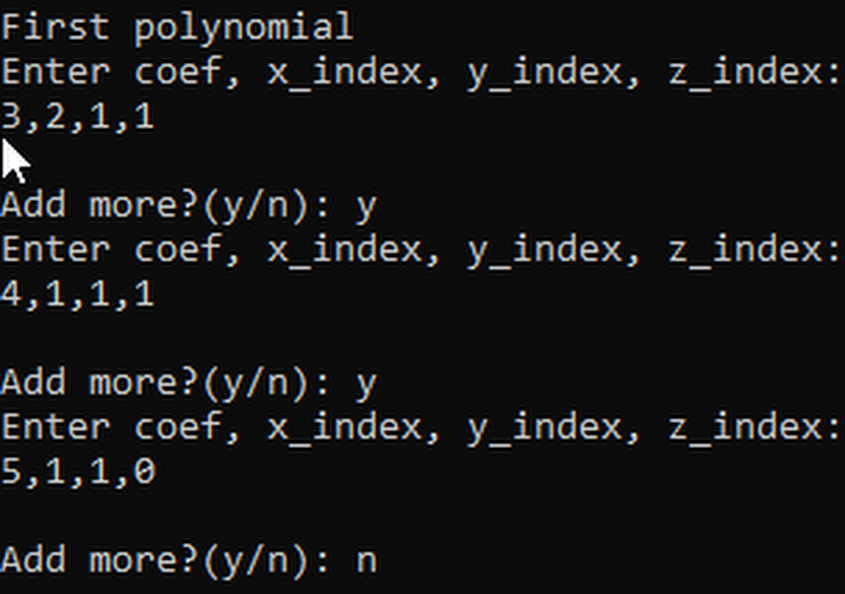


Fig: First Polynomial

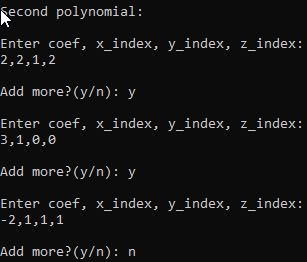


Fig: 3Second Polynomial

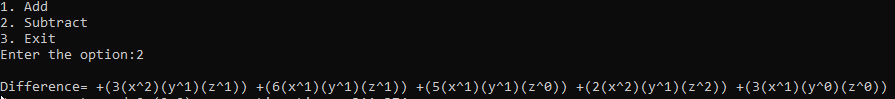


Fig: Difference of two polynomials

### Discussion and Conclusion

In this program we have used linked list to connect terms of a polynomial and used the operator overloading to operate the two polynomials.

# Lab Work 5: Recursion

## Write a recursive program to find factorial of a given number

### Problem Analysis

The problem is to create a program that calculates the factorial of the given number. The program should use the recursion to calculate the factorial.

### Algorithm

* FACT(NUM)

Step 1: IF NUM < 2

RETURN 1

ELSE,

RETURN NUM \* FACT( NUM – 1)

[END OF IF]

Step 2: EXIT

### Code

#include <iostream>

using namespace std;

int factorial(int num)

{

if(num == 0 || num == 1)

return 1;

else

return num \* factorial(num - 1);

}

int main()

{

int num;

cout<<"Enter the num to calculate its factorial: ";

cin>>num;

int result = factorial(num);

cout<<endl<<"Is factorial = "<<num<<"! = "<<result;

return 0;

}

### Output



Fig: Factorial Calculation

### Discussion and Conclusion

In this program, we have used the recursion to calculate the factorial of a given number.

## Write a recursive program to find N terms Fibonacci series

### Problem Analysis

The problem is to create a program that calculates the nth Fibonacci number for the input of n from the user. This program should use the recursion of the function to calculate the result.

### Algorithm

* FIBONACCI(NUM)

Step 1: IF NUM = 0

RETURN 0

ELSE IF NUM = 1

RETURN 1;

ELSE

RETURN FIBONACCI(NUM -1)+FIBONACCI(NUM-2)

[END OF IF]

Step 2: EXIT

### Code

#include <iostream>

using namespace std;

int fibonacci(int n\_th)

{

if(n\_th == 1 || n\_th == 0)

return n\_th;

else

{

return fibonacci(n\_th -1 ) + fibonacci(n\_th - 2);

}

}

int main()

{

int n\_th;

cout<<"Enter the position value: ";

cin>>n\_th;

int result = fibonacci(n\_th);

cout<<endl<<"The fibonacci at "<<n\_th<<" position is: "<<result;

return 0;

}

### Output

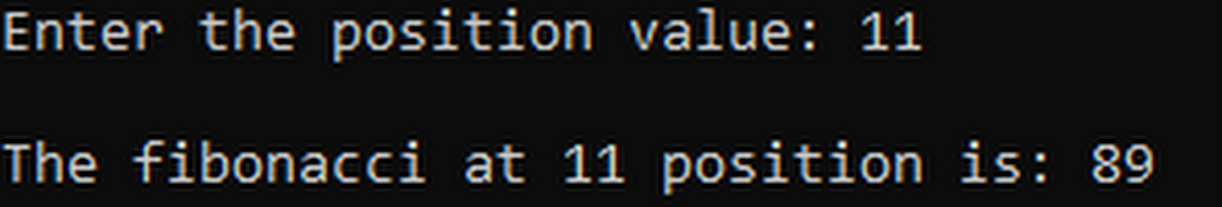


Fig: Fibonacci

### Discussion and Conclusion

In this program, tail recursion of a function has been used to calculate a fibonaccii number at a given position.

## Write a recursive program to solve Tower of Hanoi.

### Problem Analysis

The problem is to generate the necessary moves to solve the tower of Hanoi problems with a certain number of rings taken as input from the user. The program should use the recursive function to solve the problem.

### Algorithm

* TOH(NUM, SRC, DES, AUX )

Step 1: IF NUM = 1

Write Move disk from SRC to DES

ELSE

TOH(NUM-1,SRC,AUX,DES)

Write Move disk from SRC to DES

TOH(NUM-1,AUX,DES,SRC)

[END OF IF]

Step 2: EXIT

### Code

#include<iostream>

using namespace std;

void solve\_hanoi(int n, char &a, char &b, char &c)

{

if(n == 1)

{

cout<<"Move from "<<a<<" to "<<c<<endl;

}

else

{

solve\_hanoi(n-1, a, c, b);

cout<<"Move from "<<a<<" to "<<c<<endl;

solve\_hanoi(n-1,b, a, c);

}

}

int main()

{

int n;

cout<<"Enter the number of rings; ";

cin>>n;

char a='A';

char b='B';

char c='C';

solve\_hanoi(n,a,b,c);

return 0;

}

### Output

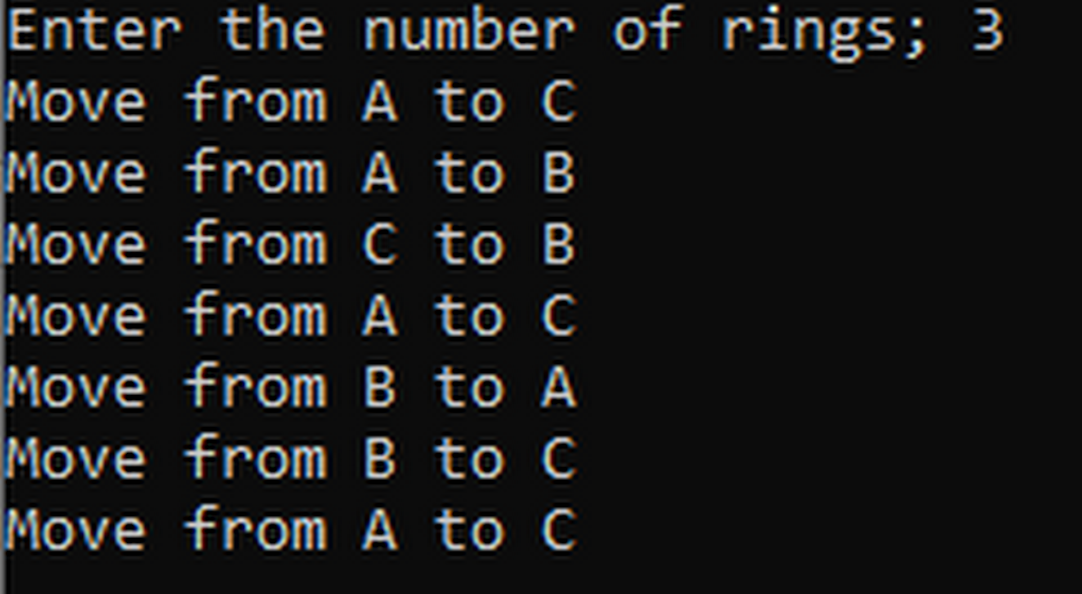


Fig: Tower of Hanoi

### Discussion and conclusion

In this program, recursion has been used to solve a tower of Hanoi. For an input of number of rings, the required moves to solve the pulzzes are displayed on the screen.

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

### Problem Analysis

The problem is to create a program that gives out the greatest common divisor of the two given numbers. The program needs to use the recursicve function to calculate the answer. We use Euclid’s algorithm to calculate the value.

### Algorithm

* GCD(NUM1, NUM2)

Step 1: IF NUM1 = 0

RETURN NUM2

ELSE

RETURN GCD(NUM2 % NUM1, NUM!)

[END OF IF]

Step 2: EXIT

### Code

#include <iostream>

using namespace std;

int gcdivisor(int num1, int num2)

{

if(num1 == 0)

return num2;

else

return gcdivisor(num2 % num1, num1);

}

int main()

{

int num1, num2;

char trash;

cout<<"enter the two numbers separated by comma: ";

cin>>num1>>trash>>num2;

int result = gcdivisor(num1, num2);

cout<<"The greatest common divisor is: "<<result;

return 0;

}

### Output



Fig: GCD of two numbers

### Discussion and Conclusion

In this program, for two given numbers, their greatest common divisor is displayed on the screen using the recursion for the calculation.

# Lab Work 6: Trees

## Write a menu driven Program for the following operations on Binary Search Tree (BST) of Integers

1. Create a BST of N Integers: 5, 10, 25, 2, 8, 15, 24, 14, 7, 8, 35, 2
2. Traverse the BST in In-order, Preorder and Post Order
3. Search the BST for a given element (KEY) and print the appropriate message
4. Exit

### Problem Analysis

The problem is to create the program for the binary search tree. The program needs to have the insert the given numbers into the binary tree and show the in-order, pre-order nad post-order traversal along with the search and exit options.

### Algorithm

* For Pre-Order Transversal

Step 1: Repeat steps 2 to 4 while TREE != NULL

Step 2: Display TREE🡪DATA

Step 3: PREORDER(TREE🡪LEFT)

Step 4: PREORDER(TREE🡪RIGHT)

Step 5: END

* For In-Order Transversal

Step 1: Repeat steps 2 to 4 while TREE != NULL

Step 2: INORDER(TREE🡪LEFT)

Step 3: Display TREE🡪DATA

Step 4: INORDER(TREE🡪RIGHT)

Step 5: END

* For Post-Order Transversal

Step 1: Repeat steps 2 to 4 while TREE != NULL

Step 2: POSTORDER(TREE🡪LEFT)

Step 3: POSTORDER(TREE🡪RIGHT)

Step 4: Display TREE🡪DATA

Step 5: END

* For searching a node in a Binary Search Tree

searchElement (TREE, VAL)

Step 1: IF TREE🡪DATA = VAL OR TREE = NULL

Return TREE

ELSE

IF VAL < TREE🡪DATA

Return searchElement(TREE🡪LEFT,VAL)

ELSE

Return searchElemen(TREE🡪RIGHT,VAL)

[END OF IF]

[END OF IF]

Step 2: END

* For inserting a new node in a Binary Search Tree

Insert (TREE, VAL)

Step 1: IF TREE = NULL

Allocate memory for TREE

Set TREE🡪DATA = VAL

Set TREE🡪LEFT = TREE🡪RIGHT = NULL

ELSE

IF VAL < TREE🡪DATA

Insert(TREE🡪LEFT, VAL)

ELSE

Insert(TREE🡪RIGHT, VAL)

[END OF IF]

[END OF IF]

Step 2: END

* Find largest node

findLargestNode (TREE)

Step 1: IF TREE = NULL

Write “Tree is empty”

Go to step 5

Step 2: Repeat Step 3 while TREE🡪RIGHT != NULL

Step 3: SET TREE = TREE🡪RIGHT

[END OF LOOP]

Step 4: RETURN TREE🡪DATA

Step 5: EXIT

* Deleting a node in a Binary Search Tree

Delete (TREE, VAL)

Step 1: IF TREE = NULL

Write “ Val not found”

ELSE IF VAL < TREE DATA

Delete(TREELEFT, VAL)

ELSE IF VAL > TREE DATA

Delete(TREERIGHT, VAL)

ELSE IF TREE LEFT AND TREE RIGHT

SET TEMP = findLargestNode(TREELEFT)

SET TREEDATA = TEMPDATA

Delete(TREELEFT, TEMPDATA)

ELSE

SET TEMP = TREE

IF TREE LEFT=NULL AND TREERIGHT=NULL

SET TREE=NULL

ELSE IF TREELEFT !=NULL

SET TREE = TREELEFT

ELSE

SET TREE = TREERIGHT

[END OF If]

FREE TEMP

[END OF IF]

Step 2: END

### Code

#include <iostream>

#include <string>

using namespace std;

static string path = "";

struct node

{

int data;

node \*left;

node \*right;

};

class binTree

{

public:

node \*root;

binTree()

{

root = NULL;

}

void create\_binTree()

{

int val;

do

{

cout<<" Enter the value: ";

cin>>val;

if(val == -1)

{

break;

}

insert\_data(val, root);

} while (val != -1);

}

void insert\_data(int val, node \*ptr)

{

if(root == NULL)

{

node \*newNode = new node;

newNode->data = val;

newNode->left = NULL;

newNode->right = NULL;

root = newNode;

}

else if(ptr->data <= val)

{

if(ptr->right == NULL)

{

node \*newNode = new node;

newNode->data = val;

newNode->left = NULL;

newNode->right = NULL;

ptr->right = newNode;

}

else

{

insert\_data(val, ptr->right);

}

}

else

{

if(ptr->left == NULL)

{

node \*newNode = new node;

newNode->data = val;

newNode->left = NULL;

newNode->right = NULL;

ptr->left = newNode;

}

else

{

insert\_data(val, ptr->left);

}

}

}

void search\_btree(int val, node \*ptr)

{

if(ptr->data == val)

{

cout<<"\n The number "<<val<<" is in the tree."<<endl;

cout<<" Path : "<<path<<endl;

}

else if(ptr == NULL)

{

cout<<" The number doesn't exist."<<endl;

}

else if(val < ptr->data)

{

path += "Left ";

search\_btree(val, ptr->left);

}

else

{

path += "Right ";

search\_btree(val, ptr->right);

}

}

void inOrderTrav(node \*ptr)

{

if(ptr != NULL)

{

inOrderTrav(ptr->left);

cout<<ptr->data<<" ";

inOrderTrav(ptr->right);

}

}

void postOrderTrav(node \*ptr)

{

if(ptr != NULL)

{

postOrderTrav(ptr->left);

postOrderTrav(ptr->right);

cout<<ptr->data<<" ";

}

}

void preOrderTrav(node \*ptr)

{

if(ptr != NULL)

{

cout<<ptr->data<<" ";

preOrderTrav(ptr->left);

preOrderTrav(ptr->right);

}

}

void delete\_btree(node\* &r,int &val)

{

if(r==NULL)

cout<<"\n\n"<<val<<" is not present in the BST.\n";

else if(val<r->data)

delete\_btree(r->left,val);

else if(val>r->data)

delete\_btree(r->right,val);

else if(r->left && r->right)

{

node \*t;

t=LargestNode(r->left);

r->data=t->data;

delete\_btree(r->left,t->data);

}

else

{

node \*t;

t=r;

if(r->left==NULL &&r->right==NULL)

r=NULL;

else if(r->left!=NULL)

r=r->left;

else

r=r->right;

delete t;

}

}

node \* LargestNode(node\* &l)

{

if(l->right==NULL)

return l;

else

return LargestNode(l->right);

}

};

int main()

{

binTree tree1;

int choice;

do

{

cout<<"\n\n1. Create Binary Tree.\n";

cout<<"2. Insert a number.\n";

cout<<"3. Search a number.\n";

cout<<"4. In-order display.\n";

cout<<"5. Post-order display.\n";

cout<<"6. Pre-order display.\n";

cout<<"7. Delete a number.\n";

cout<<"8. Exit\n";

cout<<" Enter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

{

tree1.create\_binTree();

break;

}

case 2:

{

int val;

cout<<endl<<" Enter the number to insert: ";

cin>>val;

tree1.insert\_data(val,tree1.root);

break;

}

case 3:

{

int val;

cout<<endl<<" Enter the number to search: ";

cin>>val;

path = "";

tree1.search\_btree(val, tree1.root);

break;

}

case 4:

{

tree1.inOrderTrav(tree1.root);

cout<<endl;

break;

}

case 5:

{

tree1.postOrderTrav(tree1.root);

cout<<endl;

break;

}

case 6:

{

tree1.preOrderTrav(tree1.root);

cout<<endl;

break;

}

case 7:

{

int val;

cout<<" Enter the number to delete: ";

cin>>val;

tree1.delete\_btree(tree1.root, val);

}

case 8:

{

break;

}

default:

{

cout<<endl<<" !!!Invalid input!!!\n\n";

break;

}

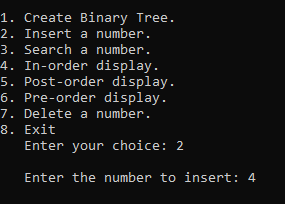
}

}while(choice != 8);

return 0;

}

### Output



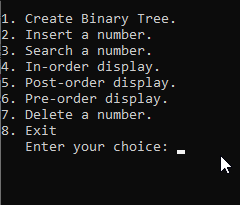


Fig: Main Menu Fig: Inserting a number

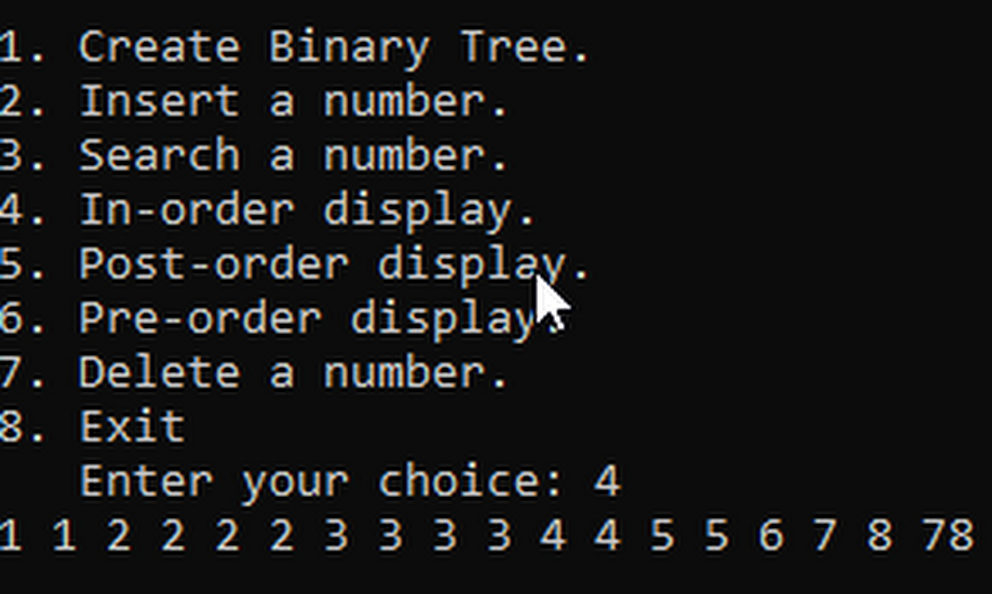
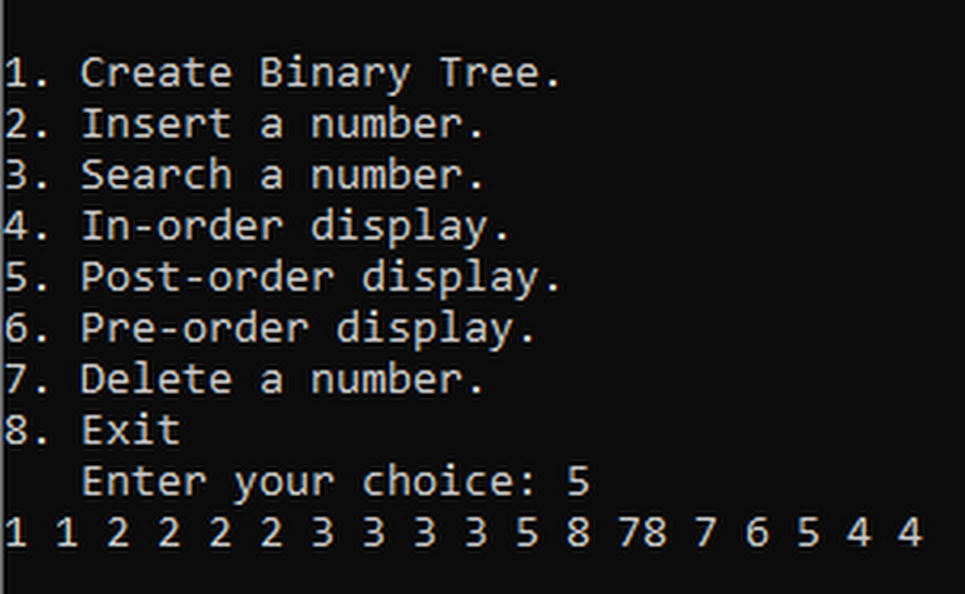
 

Fig: In-Order Display Fig: Post-Order Display

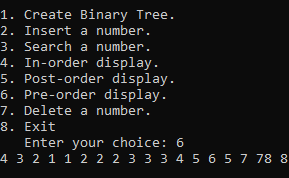


Fig: Pre-Order Display

### Discussion and conclusion

In this program, a binary search tree has been created with the specified functions. The binary search tree sorts the input numbers and makes easir searching and deleting. The concepts of pointers to link the baranches to the root has been followed here.

# Lab Work 7: Sorting

### Objective(s)

1. To understand the concepts of sorting

## WAP to implement Insertion Sorting Algorithm

### Problem Analysis

The problem is to create a program that sorts the elements of an array using the insertion sort algorithm.

### Algorithm

INSERTION\_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

Step 6: Exit

### Code

#include <iostream>

using namespace std;

void insertion\_sort(int arr[], int n)

{

int i, j, key;

for(i = 1; i < n; i++)

{

key = arr[i];

j = i - 1;

while(arr[j] > key )

{

arr[j+1] = arr[j];

j = j-1;

}

arr[j+1] = key;

}

}

int main()

{

int arr\_size;

cout<<"\tInsertion Sort"<<endl;

cout<<"\t\tenter the size of array: ";

cin>>arr\_size;

int arr[arr\_size];

cout<<endl<<"\tenter the array elements:"<<endl;

cout<<"\t";

for(int i =0; i < arr\_size; i++)

{

cin>>arr[i];

}

insertion\_sort(arr, arr\_size);

cout<<endl<<"\tThe sorted array is:\n\t";

for(int i = 0; i < arr\_size; i++)

{

cout<<arr[i]<<" ";

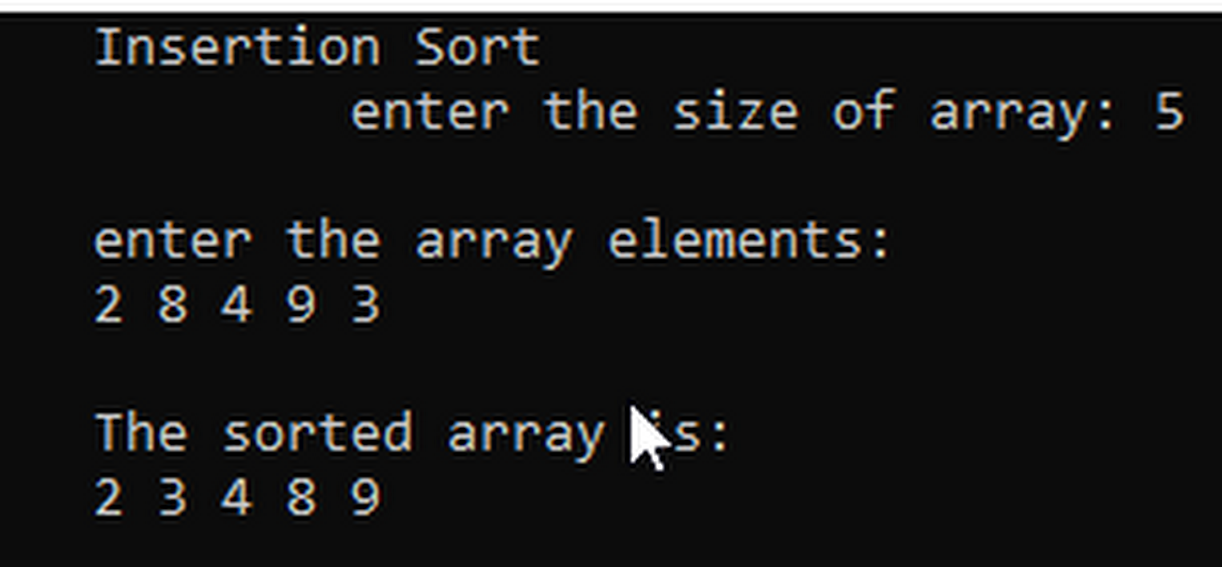
}

cout<<endl;

return 0;

}

### Output



### Discussion and Conclusion

The program uses the simple insertion sorting algorithm to sort the numbers taken as input from the user into the array.

## WAP to implement Merge Sorting Algorithm

### Problem Analysis

The problem is to create a program that uses merge sorting algorithm to sort the given numbers into the array.

### Algorithm

* MERGE\_SORT(ARR, BEG, END)

Step 1: IF BEG < END

SET MID = ( BEG + END)/2

CALL MERGE\_SORT(ARR, BEG, MID)

CALL MERGE\_SORT(ARR, MID + 1, END)

MERGE(ARR,BEG,MID,END)

[END OF IF]

Step 2: END

* MERGE(ARR, BEG, MID, END)

Step 1: [INITILIZE] 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 remining elements of right sub-array, if any]

IF I > MID

Repeat while J <=END

SET TEMP[INDEX] = ARR[J]

SET INDEX = INDEX +1, J = J + 1

[END OF LOOP]

[Copy the remining elements of right sub-array, if any]

ELSE,

Repeat while I <=MID

SET TEMP[INDEX] = ARR[I]

SET INDEX = INDEX +1, 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

### Code

#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<<"\tMerge Sorting\n";

cout<<"\tEnter no ofthe array inputs: ";

cin>>n;

int arr[n];

cout<<"\tEnter elements:\n\t";

for(int i=0;i<n;i++)

cin>>arr[i];

merge\_sort(arr,0,n-1);

cout<<"\tThe sorted items are: \n\t";

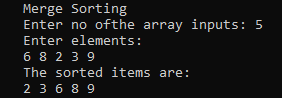
for(int i=0;i<n;i++)

cout<<arr[i]<<" ";

return 0;

}

### Output



### Discussion and Conclusion

This program takes inputs of numbers as the elements of an array following its size input. Then the merge sorting algorithms was used to sort these elements in the array.

**THANK YOU!**