NAME:VISHAL CHAUHAN

PRN:2020BTECS00090

SUB:DS

TOPIC: Experiment -Developing ADT for stack and queue and their applications

1)ADT FOR STACK AND IT’S APPLICATION

#include <iostream>

using namespace std;

struct stack

{

    int size;

    int top;

    int \*arr;

};

int isEmpty(stack \*s)

{

    if (s->top == -1)

        return 1;

    else

        return 0;

}

int isFull(stack \*s)

{

    if (s->top == s->size - 1)

        return 1;

    else

        return 0;

}

void pushing(stack \*s, int val)

{

    if (isFull(s))

        cout << "satck is overflow\n";

    else

    {

        s->arr[++s->top] = val;

    }

}

void display(stack \*s)

{

    if (isEmpty(s))

        cout << "stack is underflow\n";

    else

    {

        int toop = s->top;

        int i = 0;

        while (i <= toop)

        {

            cout << s->arr[i] << " ";

            i++;

        }

        cout << "\n";

    }

}

int pop(stack \*\*s)

{

    int pop\_element = -1;

    if (isEmpty(\*s))

        cout << "stack is underflow\n";

    else

    {

        pop\_element = (\*s)->arr[(\*s)->top];

        (\*s)->top--;

    }

    return pop\_element;

}

int main()

{

    stack \*s = new stack;

    s->top = -1;

    s->size = 8;

    s->arr = new int(s->size);

    pushing(s, 1989);

    pushing(s, 2989);

    pushing(s, 3989);

    pushing(s, 4989);

    pushing(s, 5989);

    display(s);

    pushing(s, 6989);

    pushing(s, 7989);

    pushing(s, 8989);pushing(s, 9989);

    display(s);

    cout<<"pop element:"<<pop(&s) <<endl;

    cout<<"after deleting an element the stack:";

    display(s);

    pushing(s, 9989);

     cout<<"after pushing an element the stack:";

    display(s);

}

OUTPUT:

1989 2989 3989 4989 5989

satck is overflow

1989 2989 3989 4989 5989 6989 7989 8989

pop element:8989

after deleting an element the stack:1989 2989 3989 4989 5989 6989 7989

after pushing an element the stack:1989 2989 3989 4989 5989 6989 7989 9989

APPLICATION-1:PARENTHESIS MATCHING

#include <iostream>

#include<string>

using namespace std;

struct stack

{

    int top;

    int size;

    char \*str;

};

stack \*s = new stack;

int isFull(stack \*s)

{

    if (s->top == s->size - 1)

        return 1;

    else

        return 0;

}

void push(char val)

{

    if (isFull(s))

    {

        cout << "stack is underflow\n";

    }

    else

    {

        s->str[++s->top] = val;

    }

}

int isEmpty(stack \*s)

{

    if (s->top == -1)

        return 1;

    else

        return 0;

}

char pop()

{

    char remove\_char = '0';

    if (isEmpty(s))

        cout << "stack is underflow\n";

    else

    {

        remove\_char = s->str[s->top];

        s->top--;

    }

    return remove\_char;

}

int isOpening(char c)

{

    if(c=='(' || c== '[' || c=='{')

    return 1;

    else

    return 0;

}

int isClosing(char c)

{

    if(c==')' || c==']' || c=='}')

    return 1;

    else

    return 0;

}

int isParenthesisMatched(char a,char b)

{

    if(a=='(' && b==')')

    return 1;

    else if(a=='{' && b=='}')

    return 1;

    else if(a=='[' && b==']')

    return 1;

    else

    return 0;

}

int parenthesis\_match(string str)

{

    for(int i=0;str[i]!='\0';i++)

    {

        if(isOpening(str[i]))

        {

            push(str[i]);

            cout<<str[i];

        }

        else if(isClosing(str[i]))

        {

            cout<<str[i];

            if(isEmpty(s))

            {

                cout<<"excess closing bracket\n";

                return 0;

            }

            else

            {

                char last\_bracket=pop();

                if(!isParenthesisMatched(last\_bracket,str[i]))

                return 0;

            }

        }

    }

    if(isEmpty(s))

    return 1;

    else

    return 0;

}

int main()

{

    s->top = -1;

    s->size = 20;

    s->str = new char(s->size);

    string str="(9+[6\*{5+6}])";

    int check=parenthesis\_match(str);

    if(check==1)

    cout<<"\n it is balance parenthesis\n";

    else

    cout<<"\n it is not balance parenthesis\n";

}

OUTPUT:

([{}])

it is balance parenthesis

PS E:\DSA\CWH-Lecture\stack> cd "e:\DSA\CWH-Lecture\stack\" ; if ($?) { g++ 3\_parenthesis\_match.cpp -o 3\_parenthesis\_match } ; if ($?) { .\3\_parenthesis\_match }

([{}]

it is not balance parenthesis

APPLICATION-2:INFIX TO POSTFIX

#include <iostream>

#include <string>

//correct version for all type of infixTopostfix

using namespace std;

struct stack

{

    int top;

    int size;

    char \*str;

};

stack \*s = new stack;

int isFull(stack \*s)

{

    if (s->top == s->size - 1)

        return 1;

    else

        return 0;

}

void push(char val)

{

    if (isFull(s))

    {

        cout << "stack is underflow\n";

    }

    else

    {

        s->str[++s->top] = val;

    }

}

int isEmpty(stack \*s)

{

    if (s->top == -1)

        return 1;

    else

        return 0;

}

char pop()

{

    char remove\_char = '0';

    if (isEmpty(s))

        cout << "stack is underflow\n";

    else

    {

        remove\_char = s->str[s->top];

        s->top--;

    }

    return remove\_char;

}

int isOperator(char c)

{

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

        return 1;

    else

        return 0;

}

char stack\_Top(stack \*s)

{

    if (isEmpty(s))

        return '\0';

    else

        return s->str[s->top];

}

int operator\_precedance(char c)

{

    if (c == '+' || c == '-')

        return 1;

    if (c == '\*' || c == '/')

        return 2;

    if (c == '^')

        return 3;

    else

        return 0;

}

void InfixToPrefix(string exp)

{

    char postfox\_exp[20];

    int j = 0;

    int operator\_count = 0;

    for (int i = 0; exp[i] != '\0'; i++)

    {

        if (exp[i] == '(')

        {

            push(exp[i]);

        }

        else if (!isOperator(exp[i]) && exp[i] != '('  && exp[i]!=')')

        {

            postfox\_exp[j++] = exp[i];

        }

        else if (isOperator(exp[i]))

        {

            if (operator\_count == 0)

            {

                push(exp[i]);

                operator\_count++;

                int op=0;

                cout<<"op:"<<++op<<endl;

            }

            else if (operator\_precedance(exp[i]) > operator\_precedance(stack\_Top(s)))

            {

                push(exp[i]);

            }

            else if (operator\_precedance(exp[i]) <= operator\_precedance(stack\_Top(s)))

            {

                postfox\_exp[j++] = pop();

                push(exp[i]);

            }

        }

         else if (exp[i] == ')')

        {

            int close\_brack=0;

            // cout<<"close\_bract:"<<++close\_brack<<endl;

            while (stack\_Top(s)!='(')

            {

                postfox\_exp[j++] = pop();

            }

            cout << "removed open bracket:" << pop() << endl;

        }

    }

    while (!isEmpty(s))

    {

        postfox\_exp[j++] = pop();

    }

    for (int k = 0; k < j; k++)

    {

        cout << postfox\_exp[k];

    }

}

int main()

{

    s->top = -1;

    s->size = 20;

    s->str = new char(s->size);

    string infix\_exp = "(a+b\*(c^d-e)^(f+g\*h))-i";

    cout << "infix expression:" << infix\_exp << endl;

    InfixToPrefix(infix\_exp);

}

OUTPUT:

close\_bract:1

removed open bracket:(

close\_bract:1

removed open bracket:(

close\_bract:1

removed open bracket:(

abcd^e-fgh\*+^\*+i-

APPLICATION-3:POSTFIX EXPRESSION EVOLUTION

#include <iostream>

using namespace std;

struct stack

{

    int top;

    int size;

    int \*arr;

};

stack \*s = new stack;

int isFull()

{

    if (s->top == s->size - 1)

        return 1;

    else

        return 0;

}

int isEmpty()

{

    if (s->top == -1)

        return 1;

    else

        return 0;

}

void push(int val)

{

    if (isFull())

        cout << "stack is overeflow\n";

    else

    {

        s->arr[++s->top] = val;

    }

}

int pop()

{

    int poped\_val = -1;

    if (isEmpty())

        cout << "stack is underflow\n";

    else

    {

        poped\_val = s->arr[s->top--];

    }

    return poped\_val;

}

int isOperator(char c)

{

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

        return 1;

    else

        return 0;

}

int stack\_Top(stack \*s)

{

    if (isEmpty())

        return '\0';

    else

        return s->arr[s->top];

}

void evalute\_posfixExpression(string str)

{

    int count\_operand = 0;

    for (int i = 0; str[i] != '\0'; i++)

    {

        if (!isOperator(str[i]))

        {

            count\_operand++;

            int x = 0;

            if (str[i] >= '0' && str[i] <= '9')

                x = str[i] - 48;

            push(x);

        }

        else if (isOperator(str[i]))

        {

            int operand1 = pop();

            int operand2 = pop();

            int operation = 0;

            switch (str[i])

            {

            case '+':

            {

                operation = operand2 + operand1;

                break;

            }

            case '-':

            {

                operation = operand2 - operand1;

                break;

            }

            case '\*':

            {

                operation = operand2 \* operand1;

                break;

            }

            case '/':

            {

                operation = operand2 / operand1;

                break;

            }

            case '^':

            {

                operation = operand2 ^ operand1;

                break;

            }

            }

            push(operation);

        }

    }

    cout << "evolution::" << stack\_Top(s) << endl;

    // return(stack\_Top(s));

}

int main()

{

    s->top = -1;

    s->size = 18;

    s->arr = new int(s->size);

    string str = "25+";

    cout << "current string:" << str << endl;

    evalute\_posfixExpression(str);

}

OUTPUT:

current string:25+

evolution::7

APPLICATION 4:PREFIX EVOLUTION

#include <iostream>

using namespace std;

struct stack

{

    int top;

    int size;

    int \*arr;

};

stack \*s = new stack;

int isFull()

{

    if (s->top == s->size - 1)

        return 1;

    else

        return 0;

}

int isEmpty()

{

    if (s->top == -1)

        return 1;

    else

        return 0;

}

void push(int val)

{

    if (isFull())

        cout << "stack is overeflow\n";

    else

    {

        s->arr[++s->top] = val;

    }

}

int pop()

{

    int poped\_val = -1;

    if (isEmpty())

        cout << "stack is underflow\n";

    else

    {

        poped\_val = s->arr[s->top--];

    }

    return poped\_val;

}

int isOperator(char c)

{

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

        return 1;

    else

        return 0;

}

int Operator\_precedance(char op)

{

    if (op == '(')

        return 4;

    else if (op == '^')

        return 3;

    else if (op == '\*' || op == '/' || op == '%')

        return 2;

    else if (op == '+' || op == '-')

        return 1;

    else

        return 0;

}

int stack\_Top(stack \*s)

{

    if (isEmpty())

        return '\0';

    else

        return s->arr[s->top];

}

int prefixEvalute(string str)

{

    int len = str.length();

    for (int i = len - 1; i >= 0; i--)

    {

        if (!isOperator(str[i]))

        {

            int x=0;

            if(str[i]>='0' && str[i]<='9')

                x=str[i]-48;

            push(x);

        }

        else if (isOperator(str[i]))

        {

            int operand1 = pop();

            int operand2 = pop();

            int operation = 0;

            switch (str[i])

            {

            case '+':

            {

                operation = operand1 + operand2;

                break;

            }

            case '-':

            {

                operation = operand1 - operand2;

                break;

            }

            case '\*':

            {

                operation = operand1 \* operand2;

                break;

            }

            case '/':

            {

                operation = operand1 / operand2;

                break;

            }

            case '^':

            {

                operation = operand1 ^ operand2;

                break;

            }

            default:

                break;

            }

            push(operation);

        }

    }

    cout<<"evalution:"<<stack\_Top(s)<<endl;

    return (pop());

}

int main()

{

        s->top = -1;

    s->size = 18;

    s->arr = new int(s->size);

    string str="+89";

    cout<<"current string:"<<str<<endl;

    cout<<prefixEvalute(str);

}

OUTPUT:

current string:+89

evalution:17

17

------------------------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-------------------------------------------------

II)ADT OF QUEUE

#include <iostream>

using namespace std;

class node

{

public:

    int data;

    node \*next;

};

node \*front=NULL;

node \*rear =NULL;

node \*head = NULL;

int isFull(node \*ptr)

{

    if (ptr == NULL)

        return 1;

    else

        return 0;

}

void enqueue(int val)

{

  node \*insert\_node=new node;

  if(isFull(insert\_node))

  cout<<"queue is overflow\n";

  else

  {

      insert\_node->data=val;

      insert\_node->next=NULL;

      if(front==NULL && rear==NULL)

      {

          front=rear=insert\_node;

      }

      else

      {

          rear->next=insert\_node;

          rear=insert\_node;

      }

  }

}

int isEmpty()

{

    if(front==NULL )

    return 1;

    else

    return 0;

}

int dequeue()

{

    int dequeue\_val=-1;

    if(isEmpty())

    {cout<<"queue is  underflow\n";}

    else

    {

        node \*del\_node=front;

        dequeue\_val=front->data;

        front=front->next;

        delete del\_node;

    }

    return dequeue\_val;

}

void display()

{

    node \*ptr = front;

    while (ptr != 0)

    {

        cout << ptr->data << "->";

        ptr=ptr->next;

    }

    cout << "NULL\n";

}

int peek()

{

    int peek\_val=-1;

    if(isEmpty())

    cout<<"queue is underflow\n";

    else

    {

        peek\_val=front->data;

    }

    return peek\_val;

}

int main()

{

    enqueue(16);

    enqueue(26);

    enqueue(36);

    enqueue(46);

    enqueue(56);enqueue(66);

    cout<<"peek value:"<<peek()<<endl;

    display();

    cout<<"dequeue val:"<< dequeue()<<endl;

    cout<<"dequeue val:"<< dequeue()<<endl;

    cout<<"dequeue val:"<< dequeue()<<endl;

    cout<<"dequeue val:"<< dequeue()<<endl;

    cout<<"dequeue val:"<< dequeue()<<endl;

    cout<<"dequeue val:"<< dequeue()<<endl;

    display();

}

OUTPUT:

peek value:16

16->26->36->46->56->66->NULL

dequeue val:16

dequeue val:26

dequeue val:36

dequeue val:46

dequeue val:56

dequeue val:66

NULL

CIRCULAR QUEUE:

#include <iostream>

#include <windows.h>

// #include<ios>

// #include<time.h>

// #include<Windows

using namespace std;

class queue

{

public:

    int data;

    queue \*next;

    // int  size;

} \* q;

queue \*head = NULL;

queue \*front = 0;

queue \*rear = 0;

int isFull(queue \*ptr)

{

    if (ptr == NULL)

        return 1;

    else

        return 0;

}

void enqueue(int val)

{

    queue \*insert\_node = new queue;

    if (isFull(insert\_node))

    {

        cout << "queue is overflow\n";

    }

    else

    {

        insert\_node->data = val;

        insert\_node->next = NULL;

        if (front == 0 && rear == 0)

        {

            front = rear = insert\_node;

        }

        else

        {

            rear->next = insert\_node;

            rear = insert\_node;

            rear->next=front;

        }

    }

}

int isEmpty()

{

    if(front==NULL && rear==NULL)

    return 1;

    else

    return 0;

}

int dequeue()

{

    int del\_data=-1;

    if(isEmpty())

    {

        cout<<"queue is underflow\n";

    }

    else

    {

        queue \*del\_ptr=front;

         del\_data=del\_ptr->data;

        front=front->next;

        delete del\_ptr;

    }

    return del\_data;

}

void display()

{

    queue \*ptr = front;

    int size=8;

    int i=0;

    while (i<size)

    {i++;

        cout << ptr->data << "->";

        Sleep(1000);

        ptr = ptr->next;

    }

    cout << "NULL\n";

}

int main()

{

    enqueue(1898);

    enqueue(2898);

    enqueue(3898);

    enqueue(4898);

    enqueue(5898);

    enqueue(6898);

    enqueue(7898);

    enqueue(8898);

    display();

    cout<<"dequeue val:" <<dequeue()<<endl;

    cout<<"dequeue val:" <<dequeue()<<endl;

    display();

}

OUTPUT:

1898->2898->3898->4898->5898->6898->7898->8898->NULL

dequeue val:1898

dequeue val:2898

3898->4898->5898->6898->7898->8898->14839968->14816816->NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*