MAY 2022

Evaluation of Arithmetic Expression

Data Structure - CS 214

PREPARED BY || WESAM ALJURAISH

NOTES

- ## Stack Using Array vs linked List

Singly-linked list: costs to push or pop into a linked-list-backed stack is O(1) worst-case.

Stack using a dynamic array: pushing onto the stack can be implemented by appending a new element to the dynamic array, which takes amortized O(1) time and worst-case O(n) time. Popping from the stack can be implemented by just removing the last element, which runs in worst-case O(1).

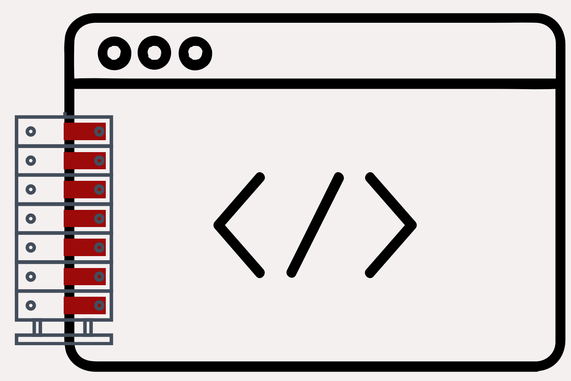
So I Use Singly-Linked List To Implemented The Stack In My Code.

- ## You Can Try The Code

On Online Editor (As Single Program)

Or

Download Files From GitHub (As Separated Programs)



CONVERT INFIX TO POSTFIX

```
#include <string>
#include<cstring> // for size_t
#include <iostream>

using namespace std;

struct node{
    double info;
    node* next;
};

class Stack {
private:
    node *top;
public:
    Stack(){top = NULL;}
    bool isEmpty();
    void push(double);
    double pop();
    double peak();
};

//Class Functions
bool Stack::isEmpty(){return (top == NULL);}

void Stack::push(double item){
    node*p = new node;
    p->info = item;
    p->next = top;
    top = p;
}

double Stack::pop(){
    if(isEmpty()){
        cout<<"Underflow\n";
        return -1;
    }else {
        node *temp = top;
        double x = top->info;
        top = top->next;
        delete temp;
        return x;
        }
}

double Stack::peak(){
    if(isEmpty()){
        cout<<"is Empty \n";
        return -1;
    }
    else
        return top->info;
}

// End Of Class's Functions

int Priority(char x);                                    // Operation 1: Check Priority
bool prcd(char symb1, char symb2);                      //  Operation 2: Decided Precedence Of
Operations
void infix_to_postfix();                                //   Operation 3: Convert Infix To
Postfix

int main(){
    cout<<"This Program Will Convert Infix To Postfix \n\n";
    infix_to_postfix();

    return 0;
}
```

CONVERT INFIX TO POSTFIX

```
int Priority(char x){
    int n = 0;
    if (x == '^')
        n = 3;
    if(x == '*' || x == '/' || x == '%' )
        n = 2;
    if(x == '+' || x == '-' )
        n = 1;
    return n;
}

bool prcd(char symb1, char symb2){
    if(symb1 == '^' && symb2 == '^') // 2^2^3
        return false;
    if(symb1 == '(' || symb2 == '('// open
        return false;
    else if(symb1 == ')')//to pop operations
        return true;
    else{
        if(Priority(symb1) >= Priority(symb2))
            return true;
        else
            return false;
    }
}
```

CONVERT INFIX TO POSTFIX

```
void infix_to_postfix(){
    Stack s;
    string infix;
    char symbol = '\0';
    int count1 = 0;// to accessing infix characters
    int count2 = 0;// to accessing postfix elements

    cout<<"Enter the infix expression:"<<endl;
    getline (cin,infix);

    size_t size = infix.length();

    string *postfix = new string[size];

    while (count1 < size) {
        symbol = infix[count1];
        if(isalpha(symbol) || isdigit(symbol)){
            postfix[count2] = symbol;
            count2++;
        }
        else{
            while (!s.isEmpty() && prcd(s.peak(), symbol)) {
                postfix[count2] = s.pop();
                count2++;
            }
            if(s.isEmpty() || symbol != ')')
                s.push(symbol);
            else
                s.pop();

        }
        count1++;
    }

    while(!s.isEmpty()){
        if( symbol == '(')
            s.pop();
        else{
            postfix[count2] = s.pop();
            count2++;
        }
    }

    for(int i=0; i<count2; i++)
        cout<< postfix[i];
    cout<<endl;

    delete [] postfix;
    postfix = NULL;
}
```

OUTPUT

CASE1:

```
This Program Will Convert Infix To Postfix

Enter the infix expression:
9^2-(5+5)*2+(3*2)
92^55+2*-32*+
Program ended with exit code: 0
```

CASE2:

```
This Program Will Convert Infix To Postfix

Enter the infix expression:
2^2^3
223^^
Program ended with exit code: 0
```

EVALUATE POSTFIX EXPRESSION

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

struct node{
    double info;
    node* next;
};

class Stack {
private:
    node *top;
public:
    Stack(){top = NULL;}
    bool isEmpty();
    void push(double);
    double pop();
    double peak();
};

//Class Functions
bool Stack::isEmpty(){return (top == NULL);}

void Stack::push(double item){
    node*p = new node;
    p->info = item;
    p->next = top;
    top = p;
}

double Stack::pop(){
    if(isEmpty()){
        cout<<"Underflow\n";
        return -1;
    }else {
        node *temp = top;
        double x = top->info;
        top = top->next;
        delete temp;
        return x;
        }
}

double Stack::peak(){
    if(isEmpty()){
        cout<<"is Empty \n";
        return -1;
    }
    else
        return top->info;
}

// End Of Class's Functions


double evaluation(double op1, double op2, char Operator);    // Operation 1: Calculate Two Values
void expression();                                          // Operation 2:  Evaluate  Postfix
Expression


int main(){
    cout<<"This Program Will Evaluate  Postfix Expression \n\n";
    expression();
    return 0;
}
```

EVALUATE POSTFIX EXPRESSION

```
double evaluation(double op1, double op2, char Operator){
    double value = 0;

    switch(Operator){
        case '^':
            value = (double)pow(op1, op2); break;
        case '*':
            value = op1 * op2; break;
        case '/':
            value = op2 != 0 ? (op1 / op2) : throw runtime_error("Math error: Attempted to divide by
Zero\n"); break;// to avoid devided by zero which is comman logic error
        case '%':
            value = (int)op1 % (int)op2; break;
        case '+':
            value = op1 + op2; break;
        case '-':
            value = op1 - op2; break;
    }
    return value;
}

void expression(){
    Stack s;
    char symbol ;
    string postfix;
    double value, opnd1 , opnd2;
    int count = 0 ;

    cout<<"Enter postfix expression : ";
    cin>> postfix;

    while( count < postfix.length() ){
        symbol = postfix[count];

        if( isdigit(symbol))
            s.push(symbol - '0'); // to convert char to int
        else{
            opnd2 = s.pop();
            opnd1 = s.pop();
            value = evaluation(opnd1 , opnd2, symbol);
            s.push(value);
        }
        count++;
    }
      cout<<"Evaluating a postfix expression : "<<s.pop()<<endl;
}
```

OUTPUT

CASE1:

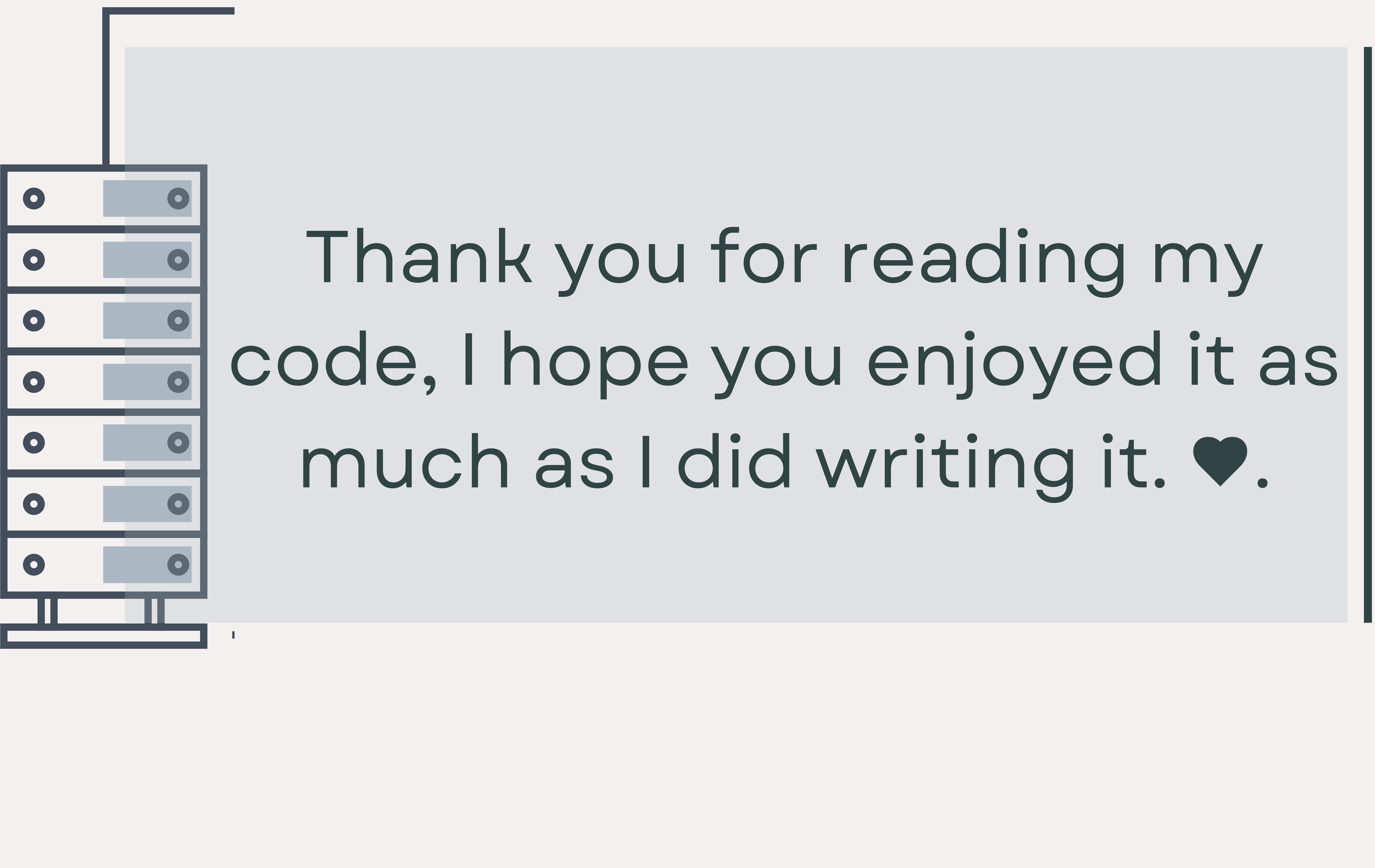
```
This Program Will Evaluate  Postfix Expression

Enter postfix expression : 92^55+2*-32*+
Evaluating a postfix expression : 67
Program ended with exit code: 0
```

CASE2:

```
This Program Will Evaluate  Postfix Expression

Enter postfix expression : 223^^
Evaluating a postfix expression : 256
Program ended with exit code: 0
```

SPECIAL CASE:

```
This Program Will Evaluate  Postfix Expression

Enter postfix expression : 95-5*55-/9*1+
libc++abi: terminating with uncaught exception of type std::runtime_error: Math error: Attempted to
divide by Zero

terminating with uncaught exception of type std::runtime_error: Math error: Attempted to divide by
Zero
Program ended with exit code: 9
```


Thank you for reading my
code, I hope you enjoyed it as
much as I did writing it. ♥.