МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: : **Динамические структуры данных**

Студент гр. 9304 Мохаммед А.А

Преподаватель Чайка К.В.

Санкт-Петербург
2020

Цель работы.

Изучить работу динамических структур данных. Реализовать стек с помощью языка программирования C++.

Задание.

Вариант 2

Стековая машина.

Требуется написать программу, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе **списка.**

1) Реализовать **класс** CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных ***int***

Структура класса узла списка:

```
struct ListNode {    ListNode* mNext;    int mData;};
```

Объявление класса стека:

```
class CustomStack {

public:

// методы push, pop, size, empty, top + конструкторы, деструктор

private:
```

```
// поля класса, к которым не должно быть доступа
извнеprotected:  // в этом блоке должен быть указатель на
голову    ListNode* mHead;};
```

Перечень методов класса стека, которые должны быть реализованы:

- **void push(int val)** - добавляет новый элемент в стек
- **void pop()** - удаляет из стека последний элемент
- **int top()** - доступ к верхнему элементу
- **size_t size()** - возвращает количество элементов в стеке
- **bool empty()** - проверяет отсутствие элементов в стеке

2) Обеспечить в программе считывание из потока ***stdin*** последовательности (не более 100 элементов) из чисел и арифметических операций (+, -, *, / (деление нацело)) разделенных пробелом, которые программа должна интерпретировать и выполнить по следующим правилам:

- Если очередной элемент входной последовательности - число, то положить его в стек
- Если очередной элемент - знак операции, то применить эту операцию над двумя верхними элементами стека, а результат положить обратно в стек (следует считать, что левый операнд выражения лежит в стеке глубже)
- Если входная последовательность закончилась, то вывести результат (число в стеке)

Если в процессе вычисления возникает ошибка:

- например вызов метода **pop** или **top** при пустом стеке (для операции в стеке не хватает аргументов)

- по завершении работы программы в стеке более одного элемента

программа должна вывести "**error**" и завершиться.

Примечания:

1. Указатель на голову должен быть protected.
2. Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено
3. Предполагается, что пространство имен std уже доступно
4. Использование ключевого слова using также не требуется
5. Структуру **ListNode** реализовывать самому не надо, она уже реализована

Результаты тестирования.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования.

№ п/п	Входные данные	Выходные данные
1	1 2 + 3 4 - 5 * +	-2
2	1 + 5 3 -	error
3	1 2 + 3 4	error
4	-12 -1 2 10 5 -14 17 17 * - - + - * +	304
5	7 -12 -11 -2 14 18 -12 13 19 * + - - - * - +	-2568

Выводы.

В процессе выполнения лабораторной работы были динамические структуры данных. Создана программа, производящая вычисления значения, опираясь на введенные в поток stdin данные и операции.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: solution.c

```
#define MAX_ELEMENTS 100

class CustomStack {
public:
// методы push, pop, size, empty, top + конструкторы, деструктор

  // Constructor 1
  CustomStack(int val){
    push(val);
  }

  // Constructor 2
  CustomStack(ListNode* new_head){
    mHead = new_head;
  }

  // Constructor 3
  CustomStack():CustomStack(nullptr){}

  // Desctructor
  ~CustomStack(){
    if (mHead!=nullptr){
      mHead->mNext = nullptr;
    }
    mHead = nullptr;
    delete mHead;
  }
```

```
  // empty method
  bool empty(){
    return (mHead==NULL);
  }

  int size(){
    int count = 0;
    ListNode* p = mHead;
    if (empty()) return 0;
    do{
      count++;
    }while((p = p->mNext)!=NULL);
    return count;
  }

  void push(int val){
    ListNode* new_node = new ListNode();
    new_node->mData = val;
    new_node->mNext = mHead;
    mHead = new_node;
  }

  void pop(){
    ListNode* p = mHead;
    if (empty()) {
        cout<<"error"<<endl;
        exit(EXIT_SUCCESS);
    }
    mHead = p->mNext;
    p->mNext = NULL;
    delete p;
  }

  int top(){
    if(empty()){
        cout<<"error\n";
        exit(EXIT_SUCCESS);
    }
    return mHead->mData;
  }
```

protected: // в этом блоке должен быть указатель на голову

```
    ListNode* mHead;
};

bool is_operator(char c){
  return (c=='+')||(c=='-')||(c=='/')||(c=='*');
}

bool is_operand(char token){
  return isdigit(token);
}

float calculate(char operation, int operand1, int operand2){
  switch(operation){
    case '+': return operand1+operand2;break;
    case '-': return operand1-operand2;break;
    case '*': return operand1*operand2;break;
    case '/': return operand1/operand2;break;
    default : cout<<"error\n"; exit(EXIT_SUCCESS);
  }
  return -1;
}

int evaluate_expression(string expression){
  char c;
  int elements = 0;
  bool neg = false, is_number = false;
  CustomStack* stack = new CustomStack();
  int result = 0;
  for (int i=0; i<(int)(expression.size()) && elements<MAX_ELEMENTS;
i++){
    c = expression[i];
    if (!neg && c=='-') {
        if (i+1 < (int)expression.size() && is_operand(expression[i+1])){
            is_number = true;
        }else{
            i--;
            is_number = false;
        }
        neg = true;
        continue;
    }

    if (is_operand(c)){
```

```
      int number = 0;
      i--;
      while(is_operand(c)){
        i++;
        number = number*10 + (c-'0');
        c = expression[i+1];
      }
      if (neg) number = -number;
      elements++;
      stack->push(number);
    }else if(is_operator(c)){
      int op2 = stack->top();stack->pop();
      int op1 = stack->top();stack->pop();
      result = calculate(c, op1, op2);
      elements++;
      stack->push(result);
    }
    if ((c==' ' || c==',' || c=='\t') && neg && !is_number) {
          i--;
    }
    if (neg) neg = false;
  }
  if (stack->size()!=1){
      cout<<"error\n";
      exit(EXIT_SUCCESS);
  }
  result = stack->top();
  delete stack;
  return result;
}

int main() {
    string expression;
    getline(cin, expression);
    int result = evaluate_expression(expression);
    cout<<result<<endl;
    return 0;
}
```