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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: Динамические структуры данных

Студент гр. 0382 _____ Осинкин Е.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучение динамических структур данных на языке 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 извне
protected:  // 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

Пример

Исходная последовательность: 1 -10 - 2 *

Результат: 22

Основные теоретические положения.

- Класс - это абстрактный тип данных, который может включать в себя данные и программный код в виде функций. Функции реализуют в себе оба принципа,описанных следующим образом:
  1. В классе могут размещаться как данные (их называют **полями**), так и функции (их называют **методы**) для обработки этих данных.
  2. Любой метод или поле класса имеет свой спецификатор доступа: ***public, private*** или ***protected*** .

- Стек – это структура данных, в которой элементы поддерживают принцип LIFO (“Last in – first out”): последним зашёл – первым вышел.

Выполнение работы.

1) Реализация класса стека *«Custom Stack»*.

Конструктор класса представляет из себя присваивание полю *mHead* объекта класса значение *nullptr*.

Метод *void push(int value)* — это функция добавления нового элемента со значением *value* в стек. В данной функции создается новый узел списка, полю *mData* этого узла присваивается значение *value*, после чего этот узел ставитсяна место головы списка.

Метод *void pop()* - это функция удаления верхнего элемента в стеке. В данной функции производится проверка стека на пустоту. Если проверка пройдена, то головой списка становится второй по счёту элемент, а первый элемент удаляется.

Метод *int top()* - это функция получения значения первого элемента в списке. В ней также производится проверка стека на пустоту, если проверка пройдена, то функция возвращает значения поля *mData* головы списка.

Метод *size_t size()* - это функция подсчёта элементов в стеке. В данной

функции производится подсчёт всех элементов списка, то есть количества элементов до того, у которого поле *mNext* имеет значение *nullptr* включительно.

Метод *bool empty()* - это функция, позволяющая проверить список на пустоту. Возвращает единицу, если *mHead* имеет значение не *nullptr* и ноль в противном случае.

2) Реализация считывания и обработки

Считывание происходит посимвольно при помощи *cin* с манипулятором *noskipws* до тех пор, пока очередной символ не будет равен переводу строки. Если очередной символ — это знак операции, то эта операция применяется к двум верхним элементам списка по следующему алгоритму:

```
int right = stack.top();
stack.pop();
int left = stack.top();
stack.pop();
stack.push(left * right);
```

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 2 + 3 4 - 5 * +	-2	Ответ верный.
2.	1 + 5 3	error	Ответ верный.
3.	-12 -1 2 10 5 -14 17 17 * - - + - * +	304	Ответ верный.
4.	1 -10 - 2 *	22	Ответ верный.
5.	1 2 + 3 4 – 5 * -	8	Ответ верный.
6.	2 2 + 3 4 – 5 * -	9	Ответ верный.

Выводы.

Изучены динамические структуры данных на языке C++.

Разработана программа, считывающая числа и арифметические операции. Программа последовательно выполняет подаваемые ей арифметические операции над двумя числами с помощью стека на базе списка

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

1. Название файла: main.cpp

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cctype>

using namespace std;

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

class CustomStack {

public:

    CustomStack() {
          mHead = nullptr;
    }

    void push(int value) {
          auto *node = new ListNode;
          node->mNext = mHead;
          node->mData = value;
          mHead = node;
    }

    void pop() {
          if (this->empty()) {
                cout << "error" << endl;
                exit(0);
          }
          auto tmp = mHead;
          mHead = mHead->mNext;
          delete tmp;
    }

    int top() {
          if (this->empty()) {
                cout << "error" << endl;
                exit(0);
          }
          return mHead->mData;
    }
```

```
    size_t size() {
        size_t size = 0;
        ListNode *curr = mHead;
        while (curr != nullptr) {
            size += 1;
            curr = curr->mNext;
        }
        return size;
    }

    bool empty() {
        if (mHead == nullptr) {
            return true;
        }
        else {
            return false;
        }
    }

protected:
    ListNode *mHead;
};

int main() {

    char chr;
    string number;
    cin >> noskipws >> chr;
    CustomStack stack;
    while (chr != '\n') {
        if (chr == '-') {
            cin >> noskipws >> chr;
            if (isdigit(chr)) {
                number = number + '-';
                while (isdigit(chr)) {
                    number = number + chr;
                    cin >> noskipws >> chr;
                }
                stack.push(stoi(number));
                number = "";
            }
            else {
                int right = stack.top();
                stack.pop();
                int left = stack.top();
                stack.pop();
                stack.push(left - right);
            }
        }
```

```
        else if (isdigit(chr)) {
            while (isdigit(chr)) {
                number = number + chr;
                cin >> noskipws >> chr;
            }
            stack.push(stoi(number));
            number = "";
        }
        else {
            if (chr == '+') {
                int right = stack.top();
                stack.pop();
                int left = stack.top();
                stack.pop();
                stack.push(left + right);
            }
            if (chr == '*') {
                int right = stack.top();
                stack.pop();
                int left = stack.top();
                stack.pop();
                stack.push(left * right);
            }
            if (chr == '/') {
                int right = stack.top();
                if (right == 0) {
                    cout << "error" << endl;
                    exit(0);
                }
                stack.pop();
                int left = stack.top();
                stack.pop();
                stack.push(left / right);
            }
            cin >> noskipws >> chr;
        }
    }
    if (stack.size() > 1) {
        cout << "error" << endl;
    }
    else {
        cout << stack.top();
    }

}
```