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

Студентка гр. 0382 _____ Чегодаева 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

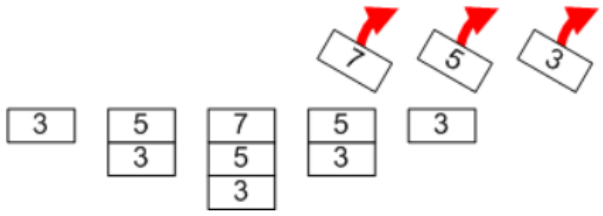
Исходная последовательность: 1 -10 - 2 *

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.



Последовательное выполнение операций push 3, push 5, push 7, pop, pop, pop

Выполнение работы.

Методы класс CustomStack:

- *void push(int val)* - добавляет новый элемент в стек. Элемент добавляется посредством создания нового узла списка, полям которого передаются значения, соответствующие новому элементу. Далее новый узел становится головным элементом списка.
- *void pop()* - удаляет из стека последний элемент. Реализована проверка на наличие элементов в списке. Если элементы в списке есть — последний элемент удаляется и головой списка становится след идущий.

- *int top()* - доступ к верхнему элементу. Реализована проверка на наличие элементов в списке. Если элементы в списке есть — возвращается значение поля *mData* элемента списка, стоящего первым на данный момент.
- size_t size() - возвращает количество элементов в стеке. Посредством цикла *while()*, выходом из которого является встреча значения *nullptr* (что означает конец списка) идёт подсчёт всех элементов в переменную *size*.
- *bool empty()* - проверяет отсутствие элементов в стеке. Если головной элемент списка имеет значение *nullptr* (что означает конец списка), то возвращается значение *TRUE*, иначе — *FALSE*.
- *void check()* - вспомогательная функция. В ней реализуется сообщение о случившейся ошибке для корректного завершения программы. Посредством функции *bool empty()* проверяет наличие элементов в списке, что необходимо для дальнейшей обработки.

Функция int main():

Посредством функции *cin.get()* поступает первый элемент. Далее реализован цикл *while()*, выходом из которого является встреча символа окончания строки. Внутри цикла на каждой итерации поступают новые значения, параллельно из входных данных извлекаются числа для обработки и арифметические операции.

Отдельно рассмотрен символ ‘- ‘ — так как он может быть частью числа или арифметической операцией. При встрече этого символа программа получает следующий элемент: если этот элемент является числом — в дополнительную переменную *m* (предназначена для хранения полной записи числа, состоящей из получаемых цифр и возможно символа “-“) кладётся “-“. Если полученное значение не является числом, то переменной *first* передаётся значение, лежащее в голове списка, далее этот элемент удаляется, аналогично и со следующим значением, которое передаётся переменной *next*. Далее

посредством функции *push()* в список кладется значение, являющееся результатом данной арифметической операции.

Затем оператором *else if* полученние значение вновь проверяется функцией *isdigit()* (в случае, если предыдущий шаг был выполнен, то это повторная проверка значения). Далее до тех пор, пока мы встречаем цифры они присоединяются к переменной *m*. После выхода из цикла посредством функции *push()* в список кладется значение переменной *m*, с преобразованием в числовой тип данных.

Затем реализована обработка остальных арифметических операций. Алгоритм повторяет действия, производимые с вычитанием, с изменениями самих операций.

По завершении реализована проверка на количество элементов в стеке — если более 1: выводится сообщение об ошибке, иначе: выводится результат всех вычислений.

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.

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

Название файла: lb4.cpp

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

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

using namespace std;

class CustomStack {
   public:
      CustomStack(){
         mHead = nullptr;
      }

      void push(int val){
         ListNode* elem = new ListNode{mHead, val};
         mHead = elem;
      }

      void check(){
            if(empty()){
            cout << "error" << endl;
            exit(0);
         }
      }

      void pop(){
         check();
         ListNode* temp = mHead;
         mHead = mHead->mNext;
         delete temp;
      }

      int top(){
         check();
         return mHead->mData;
      }

      size_t size(){
         ListNode* val = mHead;
```

```
        size_t size = 0;
        while (val!=nullptr){
            val = val->mNext;
            size=size+1;
        }
        return size;
    }

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

  protected:
    ListNode *mHead;
};

int main() {
  CustomStack* stack = new CustomStack();
  char val;
  int first;
  int next;
  string m;
  val=cin.get();
  while (val!='\n') {
    if (val=='-'){
        val=cin.get();
        if (isdigit(val)){
            m=m+'-';
        }
        else{
            first = stack->top();
            stack->pop();
            next = stack->top();
            stack->pop();
            stack->push(next-first);
        }

    }
    else if (isdigit(val)){
        while(isdigit(val)){
            m=m+val;
            val=cin.get();
        }
        stack->push(stoi(m));
```

```
        m="";
      }
      else {
        if (val== '+'){
          next = stack->top();
          stack->pop();
          first = stack->top();
          stack->pop();
          stack->push(first+next);
        }
        if (val=='*'){
          next = stack->top();
          stack->pop();
          first = stack->top();
          stack->pop();
          stack->push(first*next);
        }
        if (val=='/'){
          next = stack->top();
          stack->pop();
          first = stack->top();
          stack->pop();
          if (next== 0) {
            cout << "error" << endl;
            exit(0);
          }
          stack->push(first/next);
        }
        val=cin.get();
      }
    }
    if(stack->size()>1){
      cout << "error" << endl;
      exit(0);
    }
    cout << stack->top();
    return 0;
}
```