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

ОТЧЕТ

по лабораторной работе № 4
по дисциплине «Программирование»
Тема: «Динамические структуры данных»

Студент гр. 3343 _____ Гребнев Е. Д.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург
2024

Цель работы

Изучить особенности реализации классов на языке C++ и освоить работу с ними. Реализовать на основе списка динамическую структуру данных стек, с использованием ООП.

Задание

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.

Выполнение работы

Описание класса *CustomStack*:

public методы:

- *CustomStack()* – конструктор класса, заполняющий поля нулевыми данными.
- *empty()* – проверка наличия элементов в стеке.
- *top()* – возвращает данные в верхнем элементе стека, если это возможно.
- *size()* – возвращает размер стека.
- *push(int value)* – добавляет новый элемент в стек.
- *pop()* – удаляет элемент из стека и возвращает его значение, если это возможно.
- *change(string value)* – удаляет два элемента из стека и в зависимости от полученного значения *value* добавляет сумму, разность, произведение или частное от деления удалённых элементов в стек.

В области *private* находится размер стека *mSize*.

В области *protected* находится ссылка на голову стека *mHead*.

Описание основной части:

В string mDataBuffer происходит считывание строк через пробел до символа '\n'. Если строка является числом, то с помощью функции stoi() строка приводится к типу int, иначе над двумя верхними элементами стека производится операция, введённая пользователем. В конце, если в стеке больше одного элемента, выводится “error”, иначе значение в голове стека.

Тестирование

Результаты тестирования содержатся в таблице 1.

Таблица 1.

№	Входные данные	Выходные данные	Комментарии
1.	`1 2 + 3 4 - 5 * +`	-2	Вывод соответствует ожиданиям.
2.	`1 + 5 3 -`	error	
3.	`-12 -1 2 10 5 -14 17 17 * - - + - * +`	304	

Выводы

Во время выполнения лабораторной работы мы ознакомились с синтаксисом языка C++ по работе с классами, а также написали программу с использованием стека на основе списка.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <iostream>
#include <sstream>
#include <string>

class CustomStack {
public:
  CustomStack()
      :              mData(new              int[INITIAL_CAPACITY]),
mCapacity(INITIAL_CAPACITY),
       mSize(0) {}

  ~CustomStack() { delete[] mData; }

  void push(int val) {
    if (mSize >= mCapacity) {
      std::cout << "error" << std::endl;
      return;
    }
    mData[mSize++] = val;
  }

  void pop() {
    if (empty()) {
      std::cout << "error" << std::endl;
      return;
    }
    --mSize;
  }

  int top() const {
    if (empty()) {
      std::cout << "error" << std::endl;
      return -1;
    }
    return mData[mSize - 1];
  }

  size_t size() const { return mSize; }

  bool empty() const { return mSize == 0; }

  void extend(int n) {
    int newCapacity = mCapacity + n;
    int *newData = new int[newCapacity];
    for (size_t i = 0; i < mSize; ++i) {
      newData[i] = mData[i];
    }
    delete[] mData;
    mData = newData;
    mCapacity = newCapacity;
  }
```

```
protected:
  int *mData;

private:
  static const int INITIAL_CAPACITY = 100;
  static const int EXTENSION_SIZE = 10;
  size_t mCapacity;
  size_t mSize;
};

bool is_operator(const std::string &str) {
  return str == "+" || str == "-" || str == "*" || str == "/";
}

int main() {
  CustomStack stack;

  std::string input;
  std::getline(std::cin, input);
  std::istringstream iss(input);
  std::string word;

  while (iss >> word) {
    if (is_operator(word)) {
      if (stack.size() < 2) {
        std::cout << "error" << std::endl;
        return 1;
      }
      int operand2 = stack.top();
      stack.pop();
      int operand1 = stack.top();
      stack.pop();

      int result;
      if (word == "+")
        result = operand1 + operand2;
      else if (word == "-")
        result = operand1 - operand2;
      else if (word == "*")
        result = operand1 * operand2;
      else if (word == "/") {
        if (operand2 == 0) {
          std::cout << "error" << std::endl;
          return 1;
        }
        result = operand1 / operand2;
      }

      stack.push(result);
    } else {
      if (word == "error") {
        std::cout << "error" << std::endl;
        return 1;
      }
      stack.push(std::stoi(word));
    }
  }
```

```
  if (stack.size() != 1) {
    std::cout << "error" << std::endl;
    return 1;
  }

  std::cout << stack.top() << std::endl;

  return 0;
}
```