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 _____ Сергеев Д.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучение основных принципов работы с динамическими структурами данных на языке C++. Реализация класса стека на базе списка.

Задание.

Требуется написать программу, моделирующую работу стека на базе **списка**. Для этого необходимо:

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

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()** - возвращает верхний элемент
- **size_t size()** - возвращает количество элементов в стеке
- **bool empty()** - проверяет отсутствие элементов в стеке

2) Обеспечить в программе считывание из потока ***stdin*** последовательности команд (каждая команда с новой строки), в зависимости от которых программа выполняет ту или иную операцию и выводит результат ее выполнения с новой строки.

Перечень команд, которые подаются на вход программе в ***stdin***:

- **cmd_push n** - добавляет целое число n в стек. Программа должна вывести **"ok"**
- **cmd_pop** - удаляет из стека последний элемент и выводит его значение на экран
- **cmd_top** - программа должна вывести верхний элемент стека на экран не удаляя его из стека
- **cmd_size** - программа должна вывести количество элементов в стеке

- **cmd_exit** - программа должна вывести "**bye**" и завершить работу

Если в процессе вычисления возникает ошибка (например вызов метода **pop** или **top** при пустом стеке), 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.

1. Реализация класса CustomStack.

Конструктор класса представляет из себя присвоение полю mHead объекта класса значение nullptr.

Деконструктор класса представляет из себя освобождение памяти от элементов стека.

Метод void push(int val) добавляет в стек новый элемент со значением val, он становится головным элементов, а предыдущий головной элемент становится следующим для созданного элемента.

Метод void pop() в случае, если стек не пуст, вывод значение его головного элемента и удаляет его, делая следующий за удаленным элементов головным. В случае, если стек пуст, выводит сообщение об ошибке и завершает работу программы.

Метод int top() в случае, если стек не пуст, возвращает значение головного элемента. В случае, если стек пуст, выводит сообщение об ошибке и завершает работу программы.

Метод size_t size() возвращает количество элементов в стеке.

Метод bool empty() проверяет, пуст ли стек.

2. Реализация считывания и обработки

Считывание происходит построчно пока не поступит команда на конец работы, в зависимости от команды выполняется определенный метод.

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.	cmd_push 1 cmd_top cmd_push 2 cmd_top cmd_pop cmd_size cmd_pop cmd_size cmd_exit	ok 1 ok 2 2 1 1 0 bye	Ответ верный.

Выводы.

Были изучены динамические структуры данных языке C++. Разработана программа, в которой реализован стек на базе линейного списка. Программа выполняет считывание команд и работу со стеком.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb2.c

```
#include <iostream>
#include <cstring>

using namespace std;

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

class CustomStack {

public:

  CustomStack()
  {
    mHead= nullptr;
  }

  ~CustomStack()
  {
    auto* cur=mHead;
    while (cur!= nullptr)
    {
      mHead=mHead->mNext;
      delete(cur);
      cur=mHead;
    }
  }
```

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

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

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

size_t size()
{
```

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

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

private:

// поля класса, к которым не должно быть доступа извне

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

  ListNode* mHead;
};

int main()
{
  CustomStack head;
```

```
char comList[5][10]={"cmd_push","cmd_pop","cmd_top","cmd_size", "cmd_exit"};
char com[15];
while (true)
{
  cin >> com;
  if (strcmp(com,comList[0])==0)
  {
    int val;
    cin >> val;
    head.push(val);
    cout << "ok" << endl;
    continue;
  }
  if (strcmp(com,comList[1])==0)
  {
    head.pop();
    continue;
  }
  if (strcmp(com,comList[2])==0)
  {
    cout << head.top() << endl;
    continue;
  }
  if (strcmp(com,comList[3])==0)
  {
    cout << head.size() << endl;
    continue;
  }
  if (strcmp(com,comList[4])==0)
  {
    cout << "bye" << endl;
    exit(0);
  }
```

```
    }
    return o;
}
```