

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. 1304

Преподаватель

Шалагинов И.В.

Чайка К.В.

Санкт-Петербург

2022

Цель работы.

Изучить работу с динамическими структурами данных в языке C++.

Задание.

Моделирование стека.

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*.

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()** - 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

Для реализации класса CustomStack нам нужны:

- Конструктор CustomStack(), который при создании объекта класса CustomStack переменную mHead;
- Деструктор ~CustomStack().
- Функция void push(), которая принимает в качестве аргумента число и помещает её в стек;
- Функция void pop(), которая удаляет верхний элемент в стеке;
- Функция int top(), которая возвращает поле mData последнего элемента стека;
- Функция size_t size(), которая количество элементов в стеке;
- Функция bool empty(), которая проверяет, пуст ли стек.

В функции main() мы в цикле считываем строку, введенную в stdin, после чего сравниваем ее с одной из возможных команд, после чего вызывается соответствующий метод класса.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментари и
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	

Выводы.

Была изучена работа с динамическими структурами данных в языке C++.

Приложение А

Исходный код программы

Название файла: lb4.cpp

```
class CustomStack {
public:
    CustomStack(){
        mHead = NULL;
    };
    ~CustomStack(){
        while(mHead){
            ListNode* deleting = mHead;
            mHead = mHead->mNext;
            free(deleting);
        };
    };
    void push(int val){
        ListNode* newNode = (ListNode*)malloc(sizeof(ListNode));
        if(mHead == NULL){
            newNode->mNext = NULL;
        }else{
            newNode->mNext = mHead;
        }
        newNode->mData = val;
        mHead = newNode;
    };
    int pop(){
        if(mHead == NULL){
            printf("error");
            exit(0);
        }
        ListNode* temp = mHead;
        int tempD = temp->mData;
        mHead = mHead->mNext;
```

```

free (temp);
return tempD;
};

int top(){
if(this->empty()){
printf("error\n");
exit(0);
}
return mHead->mData;
};

size_t size() {
if (mHead == NULL) {
return 0;
} else {
int counter = 1;
struct ListNode *temp = mHead;
while (temp->mNext != NULL) {
counter++;
temp = temp->mNext;
}
return counter;
}
}

bool empty(){
return mHead==NULL;
};

protected:
ListNode* mHead = NULL;
};

int main() {
char option[10];
CustomStack Stack;
while (1) {
fgets(option, 10, stdin);
option[strcspn(option, "\n ") = 0;
if (!strcmp(option, "cmd_push")) {
int count;
scanf("%d", &count);
Stack.push(count);
cout<<"ok"<<endl;
} else if (!strcmp(option, "cmd_pop")) {
int poppedEl = Stack.pop();
cout<<poppedEl<<endl;
} else if (!strcmp(option, "cmd_top")) {
int topEl = Stack.top();
cout<<topEl<<endl;
}
}
}

```

```
} else if (!strcmp(option, "cmd_size")) {  
    cout<<Stack.size()<<endl;  
} else if (!strcmp(option, "cmd_exit")) {  
    cout<<"bye";  
    exit(0);  
}  
}  
}
```