**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. 3344 |  | Бубякина Ю.В. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Изучение динамических структур данных. Моделирование динамических структур данных в языке С++.

## Задание

Вариант 4

**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***:

* **сmd\_push n** -  добавляет целое число n в стек. Программа должна вывести **"ok"**
* **сmd\_pop** - удаляет из стека последний элемент и выводит его значение на экран
* **сmd\_top** - программа должна вывести верхний элемент стека на экран не удаляя его из стека
* **сmd\_size** - программа должна вывести количество элементов в стеке
* **сmd\_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

Был реализован *class CustomStack*, содержащий в себе следующие поля и методы:

* public:
  + *CustomStack()* – конструктор, создающий голову стека.
  + *~CustomStack()* – деструктор, очищающий память, выделенную под стек.
  + ***bool*** *empty()* – возвращает *true*, если стек пуст и *false* в противном случае.
  + ***void*** *push(int val)* – добавляет новый элемент со значением *val* в стек и выводит *“ok”* .
  + ***void*** *pop()* – удаляет из стека последний элемент и выводит его значение на экран.
  + ***int*** *top()* – выводит на экран верхний элемент стека.
  + ***size\_t*** *size()* – возвращает количество элементов в стеке.
* private:
  + ***size\_t*** *count* – хранит количество элементов в стеке.
* protected:
  + ***ListNode\**** *mHead* – указатель на голову.

В функции *main()* объявляется объект класса *CustomStack* и переменная *command* типа *string* для записи в нее вводимых команд. Далее из стандартного потока ввода считываются команды, в зависимости от которых вызываются соответствующие методы для объявленного ранее стека. Также учитываются возможные ошибки и, при встрече таковых, на экран выводится *“error”*, а программа завершает работу с помощью *exit(0).* По завершении работы программы корректным образом, на экран выводится *“bye”*.

**Тестирование**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 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 | Корректно |

## Выводы

Были изучены динамические структуры данных. Был реализован стек на базе списка в языке С++ и написан API к нему. Также был реализован ввод команд из стандартного потока ввода.

# Приложение А Исходный код программы

Название файла: Bubyakina\_Yuliya\_lb4.cpp

#include <iostream>

#include <string>

class CustomStack

{

public:

CustomStack() : mHead(nullptr) {}

~CustomStack()

{

while (mHead)

{

auto temp = mHead;

mHead = mHead->mNext;

delete temp;

}

}

void push(int val)

{

auto temp = make\_ListNode(val);

auto end = get\_end\_ListNode();

if (!end)

mHead = temp;

else

end->mNext = temp;

std::cout << "ok\n";

}

void pop()

{

auto end = get\_end\_ListNode(true);

if (!end)

exitExec();

else if (end == mHead && !mHead->mNext)

{

delete end;

mHead = nullptr;

}

else

{

delete (end->mNext);

end->mNext = nullptr;

}

}

int top()

{

auto end = get\_end\_ListNode();

if (!end)

exitExec();

return end->mData;

}

size\_t size()

{

auto temp = mHead;

if (!temp)

return 0;

int count = 1;

for (; temp->mNext; count++)

temp = temp->mNext;

return count;

}

bool empty()

{

return !mHead;

}

private:

ListNode \*make\_ListNode(int val)

{

ListNode \*ln = new ListNode;

if (!ln)

exitExec();

ln->mNext = nullptr;

ln->mData = val;

return ln;

}

ListNode \*get\_end\_ListNode(bool returnOneBeforeLast = false)

{

auto temp = mHead;

if (!temp)

return nullptr;

auto prev = temp;

while (temp->mNext)

{

prev = temp;

temp = temp->mNext;

}

return returnOneBeforeLast ? prev : temp;

}

void exitExec()

{

std::cout << "error\n";

exit(0);

}

protected:

ListNode \*mHead;

};

int main()

{

{

auto stack = CustomStack();

std::string cmd = "";

do

{

std::cin >> cmd;

if (!cmd.compare("cmd\_push"))

{

int val = 0;

std::cin >> val;

stack.push(val);

}

else if (!cmd.compare("cmd\_pop"))

{

std::cout << stack.top() << '\n';

stack.pop();

}

else if (!cmd.compare("cmd\_top"))

{

std::cout << stack.top() << '\n';

}

else if (!cmd.compare("cmd\_size"))

{

std::cout << stack.size() << '\n';

}

else if (!cmd.compare("cmd\_exit"))

{

std::cout << "bye";

exit(0);

}

std::cin.ignore();

} while (cmd.compare("cmd\_exit") != 0);

}

}