**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 |  | Лихацкий В.Р. |
| 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:

structListNode {

ListNode\* mNext;

int mData;

};

Объявление класса стека:

classCustomStack {

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:

* **voidpush(int val)** -  добавляет новый элемент в стек
* **voidpop()** - удаляет из стека последний элемент
* **int top()** - доступ к верхнему элементу
* **size\_tsize()**- возвращает количество элементов в стеке
* **boolempty()** - 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(intvalue)* – добавляет новый элемент в стек.
* *pop()*– удаляет элемент из стека, если это возможно.
* *change(stringvalue)* – удаляет два элемента из стека и в зависимости от полученного значения *value* добавляет сумму, разность, произведение или частное от деления удалённых элементов в стек.
* *~CustomStack()* – деконструктор класса, очищающий стек.

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

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

Описание основной части:

Сначала происходит считывание элементов и добавление в вектор. Для отслеживания символа перехода к новой строке, заканчивающего ввод, используется *cin.peek()*, который смотрит следующий символ из потока ввода, не удаляя его. Затем идёт обработка полученных элементов: числа добавляются в стек, а для операций вызывается метод *change*. По итогу должен остаться только один элемент в стеке, который выводится.

## 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

class CustomStack

{

public:

CustomStack() {

this->mHead = nullptr;

this->mSize = 0;

}

void push(int value) {

ListNode \*newNode = new ListNode;

newNode->mData = value;

newNode->mNext = this->mHead;

this->mHead = newNode;

this->mSize++;

}

void pop() {

if(this->empty()) {

throw range\_error("Stack is empty");

return;

}

this->mHead = this->mHead->mNext;

this->mSize--;

}

int top() {

if(this->empty()) {

throw range\_error("Stack is empty");

return -1;

}

return this->mHead->mData;

}

size\_t size() {

return this->mSize;

}

bool empty() {

return this->mHead == nullptr;

}

protected:

ListNode \*mHead;

int mSize;

};

vector<string> split(string src, string delimiter) {

size\_t pos;

string token;

auto result = vector<string>();

while((pos = src.find(delimiter)) != string::npos) {

token = src.substr(0, pos);

result.push\_back(token);

src.erase(0, pos + 1);

}

result.push\_back(src);

return result;

}

int calcRPNString(string expression) {

CustomStack \*stack = new CustomStack();

auto tokens = split(expression, " ");

for(auto token : tokens) {

try {

stack->push(stoi(token));

continue;

} catch(invalid\_argument &e) {}

char operation = token[0];

int a, b;

try {

b = stack->top();

stack->pop();

a = stack->top();

stack->pop();

} catch(range\_error &e) {

cout << "error" << endl;

exit(0);

}

switch(operation) {

case '+':

stack->push(a + b);

break;

case '-':

stack->push(a - b);

break;

case '\*':

stack->push(a \* b);

break;

case '/':

stack->push(a / b);

break;

}

}

if(stack->size() > 1) {

cout << "error" << endl;

exit(0);

}

return stack->top();

}

int main()

{

string expression;

getline(cin, expression);

int result = calcRPNString(expression);

cout << result << endl;

return 0;

}