# 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 Тюленев Т.В.

Преподаватель Берленко Т. А.

Санкт-Петербург 2021

# Цель работы.

Написать программу, использующую динамические структуры данных.

# Задание.

**Стековая машина.**

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

1. 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

# Основные теоретические положения.

* ***class NameofClass{}*** – объявление класса в языке С++.
* ***public*** – спецификатор, за которым следуют те методы и поля класса,

которые будут доступны любым функциям, взаимодействующим с объектом данного класса.

* ***private*** – спецификатор, за которым следуют те методы и поля класса,

которые будут доступны только внутри данного класса.

* ***protected*** – спецификатор, за которым следуют те методы и поля класса,

которые будут доступны как внутри класса, так и в классах, наследующихся от данного.

* ***NameofClass()*** – конструктор, функция, которая будет вызвана сразу

после создания объекта класса. Данная функция может отсутствовать, в таком случае компилятор создаст конструктор неявно.

* ***~NameofClass()*** – деструктор, функция, которая будет вызвана после

удаления объекта класса. Эта функция, как и конструктор, может отсутствовать, и так же, как и в случае с конструктором, компилятор создаст эту функцию неявно.

* ***new type*** – конструкция для выделения памяти в языке С++. В языке С

этот код эквивалентен данному: ***malloc(sizeof(type))***.

* ***delete value*** – конструкция для освобождения памяти в языке С++. В языке С этот код эквивалентен следующему: ***free(value)***.
* ***exit(code)*** – функция для завершения работы программы с переданным кодом выхода.

# Выполнение работы.

В первой строке написанной программы объявляется макрос ***MAX***, содержащий длину максимальной строки, которая может быть передана на вход программе. Далее следует объявление класса ***CustomStack***. Первыми в нем объявляются методы со спецификатором ***public***. Метод ***is\_number*** позволяет определить, число или операция содержится в строке, метод ***empty*** проверяет, не пуст ли стэк, метод ***size*** возвращает количество элементов, содержащихся в стэке, метод ***top*** возвращает самый верхний элемент стэка, метод ***pop*** удаляет верхний элемент стэка, метод ***push*** добавляет новый элемент в стэк, метод ***check*** проверяет, содержится ли хотя бы один элемент в стэке с помощью метода ***empty***, и если нет, то вызывается метод ***throw\_error()***, который выводит сообщение об ошибке и завершает выполнение программы. Деструктор выполняет удаление всех элементов стэка с помощью метода ***pop***.

Далее следует спецификатор ***protected*** c единственным полем ***mHead*** –

указателем на голову списка.

В функции ***main*** сначала выделяется память для хранения объекта класса ***CustomStack***, затем, после объявление необходимых переменных, считывается первая строка. В цикле ***while***, условием которого является “не достижение” конца строки, проверяется, является ли текущая строка числом. Если да, то число добавляется в стэк. В противном случае извлекаются первые два сверху стэка элемента (если это возможно) и в соответствии с переданной операцией создается новый элемент, который добавляется в стэк. После этого производится считывание нового элемента. По окончании цикла производится проверка числа оставшихся в стэке элементов: если число элементов не равно 1, то выводится сообщение об ошибке, и программа завершается. В противном случае выводится значение элемента.

# Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | Входные данные | | | | | | | | | | Выходные данные | Комментарии | |
| #1 | 1 2 + 3 4 – 5 \* | | | | | | | | | | -2 | Программа корректно. | работает |
| #2 | 1 + 5 3 - | | | | | | | | | | error | Программа корректно. | работает |
| #3 | -12 | -1 | 2 | 10 | 5 | -14 | 17 | 17 | \* | - | 304 | Программа | работает |
|  | - + | - | \* + |  |  |  |  |  |  |  |  | корректно. |  |

# Выводы.

Была написана программа, применяющая динамическую структуру данных.

Для написания простого калькулятора был реализован стэк на базе связанного списка.

Создан класс, реализующий удобную модель такого стэка.

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp #define MAX 100

class CustomStack { public:

~CustomStack(){ while (mHead){

pop();}}

void throw\_error(){ printf("error\n"); delete this; exit(0);}

void check(){

if (empty()){

throw\_error();}}

void push(int val){

ListNode\* new\_elem = new ListNode{mHead, val}; mHead = new\_elem;}

void pop(){

check();

ListNode\* temp = mHead; mHead = mHead->mNext; delete temp;}

int top(){

check();

return mHead->mData;} size\_t size(){

ListNode\* cur = mHead; size\_t num;

for (num = 0; cur; num++){ cur = cur->mNext;}

return num;}

bool empty(){

return !((bool)mHead);}

int is\_number(char\* str){ int len = strlen(str);

if (len == 1 && !isdigit(str[0])){ return 0;}

return 1;}

protected:

ListNode\* mHead;};

int main(){

CustomStack\* stack = new CustomStack(); char cur[MAX], a;

int first, second; scanf("%s", cur); while (!feof(stdin)){

if (stack->is\_number(cur)){ stack->push(atoi(cur));

} else {

first = stack->top(); stack->pop();

second = stack->top(); stack->pop();

if (!strcmp(cur, "+")){

stack->push(first + second);

} else if (!strcmp(cur, "-")){ stack->push(second - first);

} else if (!strcmp(cur, "\*")){ stack->push(first \* second);

} else if (!strcmp(cur, "/")){ stack->push(second / first);}}

scanf("%s", cur);}

if (stack->size() != 1){ stack->throw\_error();}

first = stack->top(); printf("%d\n", first); return 0;}