**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. 3341 |  | Кузнецова С.Е. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Целью работы является изучение основ языка C++, а также разработки стека с выделением памяти динамически на данном языке.

Для достижения цели требуется решить следующие задачи:

1. Ознакомиться со понятиями динамических структур данных, таких как стек и очередь, а также особенностями их реализации.
2. Изучить базовые механизмы языка C++, необходимые для реализации стека и очереди.
3. Написать программу на языке C++, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе списка.

## Задание

**Вариант 2**

**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

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:

* Указатель на голову должен быть protected.
* Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено.
* Предполагается, что пространство имен std уже доступно.
* Использование ключевого слова using также не требуется.
* Структуру ListNode реализовывать самому не надо, она уже реализована.

## Выполнение работы

1. Создается класс CustomStack, который содержит методы для работы со стеком на базе списка. Стек имеет возможность хранить и работать с типом данных int. Методы класса в блоке public:

- void push(int val): добавляет элемент типа int в стек.

- void pop(): удаляет из стека последний элемент.

- int top(): возвращает верхний элемент стека.

- size\_t size(): возвращает количество элементов в стеке.

- bool empty(): проверяет отсутствие элементов в стеке.

- void print(): выводит верхний элемент, если он в стеке единственный.

2. Методы класса в блоке private:

- void error(): выводит сообщение об ошибке “error” и завершает программу.

1. Основные функции программы:

- Функция int operate(int a, int b, string sign): В зависимости от полученного арифметического оператора выводит сумму, разность, произведение или частное двух чисел.

- Функция string input(): Используется для осуществления пользователем ввода обрабатываемой строки, возвращает данную строку.

- Функция void process (CustomStack& stack): обрабатывает ввод пользователя и выполняет арифметические операции с введенными числами и арифметическими операторами, результат также записывает в стек, а потом выводит его, используя метод. Реализованный внутри класса.

- Функция int main(): создает экземпляр объекта CustomStack, после чего вызывается функция process(), где происходит обработка вводимой пользователем строки согласно логике, описанной в коде.

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 2 + 3 4 - 5 \* + | -2 | Стандартный тест на общую работоспособность программы |
|  | 1 + 5 3 - | error | Тест на ситуацию, вызывающую сообщение об ошибке |
|  | -12 -1 2 10 5 -14 17 17 \* - - + - \* + | 304 | Тест с нетипичным вводом данных |
|  | 1 2 3 4 5 | error | Тест на ситуацию, вызывающую сообщение об ошибке |
|  | Who am I | error | Тест с вводом неправильного типа данных |

## Выводы

В ходе выполнения работы были изучены основы языка C++. Также были достигнуты поставленные задачи:

* 1. Были изучены такие динамические структуры данных, как стек и очередь, их основные особенности.
  2. Для реализации стека в виде класса на языке C++ были изучены базовые механизмы языка C++.
  3. Была написана программа, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе списка.

# Приложение А Исходный код программы

Название файла: main.cpp

class CustomStack {

public:

CustomStack() {

this->mHead = nullptr;

}

~CustomStack() {

delete this->mHead;

}

void push(int data) {

ListNode\* element = new ListNode();

element->mData = data;

element->mNext = this->mHead;

this->mHead = element;

}

void pop() {

if (this->empty()) {

this->error();

}

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

}

int top() {

if (this->empty()) {

this->error();

}

return this->mHead->mData;

}

size\_t size() {

ListNode\* curr = mHead;

size\_t size = 0;

while (curr != nullptr) {

curr = curr->mNext;

size++;

}

return size;

}

bool empty() {

return (this->mHead == nullptr);

}

void print() {

if (this->size()==1) {

cout << this->top() << endl;

}

else {

this->error();

}

}

private:

void error() {

cout << "error" << endl;

exit(0);

}

protected:

ListNode\* mHead;

};

int operate(int a, int b, string sign) {

if (sign=="+") {

return b+a;

}

else if (sign=="-") {

return b-a;

}

else if (sign=="\*") {

return b\*a;

}

else {

return b/a;

}

}

string input(){

string inp;

getline(cin, inp);

return inp;

}

void process(CustomStack& stack) {

string inp = input();

istringstream stream(inp);

string symbol;

while (stream >> symbol) {

try {

int result = stoi(symbol);

stack.push(result);

}

catch (invalid\_argument e) {

string sign = symbol;

int a = stack.top();

stack.pop();

int b = stack.top();

stack.pop();

int result = operate(a, b, sign);

stack.push(result);

}

}

stack.print();

}

int main()

{

CustomStack stack;

process(stack);

return 0;

}