**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. 3342 |  | Пушко К.Д. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Изучить динамические структуры данных на языке С++ и применить полученные знания на практике.

## 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;

};

Click and drag to move

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;

};

Click and drag to move

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

В данной работе был реализован стек на списке. Класс стека (CustomStack) имеет методы: push – добавление нового элемента в конец стека, pop – удаление верхнего элемента стека, top – возвращает верхний элемент стека, size – возвращает количество элементов в стеке, empty возвращает bool значение, которое зависит от того, пуст ли стек.

В конструкторе класса CustomStack инициализируются поля stackSize и mHead. В деструкторе идет последовательное очищение памяти, выделенной под элементы стека.

Программа начинается с инициализации экземпляра класса CustomStack и других переменных. Программа считывает строку и далее, с помощью функции split разделяет ее на строки, разделенные пробелом. Далее программа проходит по всем этим переменным и выполняет с ними действия, описанные в задании.

После прохода по всем элементам, программа выводит сообщение об ошибке или результат выполнения программы, после этого память, которая была выделена под стек очищается.

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарий |
| 1. | 1 2 + 3 4 - 5 \* + | -2 | Верный вывод |
| 2. | 1 + 5 3 - | error | Верный вывод |
| 3. | -12 -1 2 10 5 -14 17 17 \* - - + - \* + | 304 | Верный вывод |
| 4. | 1 -10 - 2 \* | 22 | Верный вывод |

## Выводы

В ходе выполнения лабораторной работы были изучены и применены на практике основы написания программы на языке C++ и динамические структуры данных на языке C++. Реализована программа со стеком на базе списка, выполняющая задание.

# Приложение А Исходный код программы

Название файла: main.cpp

class CustomStack{

public:

CustomStack()

{

this->stackSize = 0;

this->mHead = NULL;

}

void push(int val)

{

ListNode\* newElement = new ListNode();

if(newElement == nullptr)

{

cout << "Memory error\n";

exit(1);

}

newElement->mNext = NULL;

newElement->mData = val;

if (this->empty())

{

this->mHead = newElement;

} else

{

ListNode\* tempNode = mHead;

while(tempNode->mNext!=NULL)

{

tempNode = tempNode->mNext;

}

tempNode->mNext = newElement;

}

this->stackSize ++;

}

void pop()

{

if (this->empty())

{

cout << "error";

exit(0);

} else if (this->stackSize == 1)

{

delete this->mHead;

this->stackSize --;

}

else

{

ListNode\* tempNode = mHead;

while(tempNode->mNext->mNext!=NULL)

{

tempNode = tempNode->mNext;

}

delete tempNode->mNext;

tempNode->mNext = NULL;

this->stackSize--;

}

}

int top()

{

if (this->empty())

{

cout << "error";

exit(0);

}

ListNode\* tempNode = mHead;

while(tempNode->mNext!=NULL)

{

tempNode = tempNode->mNext;

}

return tempNode->mData;

}

size\_t size()

{

return this->stackSize;

}

bool empty()

{

return (this->stackSize == 0);

}

~CustomStack()

{

if (this->empty())

{

return;

} else

{

ListNode\* tempNode = mHead;

while(tempNode->mNext!=NULL)

{

ListNode\* removeNode = tempNode;

tempNode = tempNode->mNext;

delete removeNode;

}

}

}

private:

size\_t stackSize;

protected:

ListNode\* mHead;

};

vector<string> split(const string &str, char separator) {

vector<string> result;

stringstream ss(str);

string buffer;

while (getline(ss, buffer, separator)) {

result.push\_back(buffer);

}

return result;

}

int main() {

bool okFlag = true;

CustomStack \*stack = new CustomStack();

if(stack == nullptr)

{

cout << "Memory error\n";

exit(1);

}

string text;

getline(cin, text);

vector<string> arr = split(text, ' ');

for (int i = 0; i < arr.size(); ++i) {

if (arr[i] == "+" || arr[i] == "-" || arr[i] == "\*" || arr[i] == "/") {

if (stack->empty()) {

okFlag = false;

break;

}

int rightOperand = stack->top();

stack->pop();

int leftOperand = stack->top();

stack->pop();

int result;

if (arr[i] == "+") {

result = leftOperand + rightOperand;

} else if (arr[i] == "-") {

result = leftOperand - rightOperand;

} else if (arr[i] == "\*") {

result = leftOperand \* rightOperand;

} else {

result = leftOperand / rightOperand;

}

stack->push(result);

} else {

stack->push(stoi(arr[i]));

}

}

if (!okFlag || stack->size()!=1) {

cout << "error";

} else

{

cout << stack->top();

}

delete stack;

return 0;

}