**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3341 |  | Яковлева А.А. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Целью работы является изучение динамических структур данных.

Для достижения поставленной цели требуется решить следующие задачи:

* написать программу на языке C++, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе списка.
* реализовать следующие методы класса стека: *void push(int val), void pop(), int top(), size\_t size(), bool empty()*

## Задание

2 вариант.

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

Используемые переменные:

* макрос *MAX\_SEQUENCE\_LENGTH -* максимальная длина входной последовательности
* макрос *END\_STRING -* символ конца строки '\0'

Поля и методы класса стека *CustomStack*:

* *ListNode\* mHead* указатель на голову списка.
* *CustomStack() -* конструктор, указателю на голову списка *mHead* присваивает *NULL.*
* *bool empty() -* проверяет отсутствие элементов в стеке, если указатель на голову *NULL* возвращает *true*, иначе возвращает *false*.
* *void push(long long int val)* добавляет новый элемент в стек. С помощью оператора *new* выделяется память для нового узла списка *tempNode*, полю узла *mData* присваивается значение *val*, полю узла *mNext* присваивается *mHead*, полю *mHead* присваивается *tempNode,* т.е. элемент добавляется в голову списка.
* *void pop()* удаляет из стека последний элемент. Переменной *tempNode* присваивает *mHead, mHead* присваивает указатель на следующий элемент и с помощью оператора *delete* освобождает память, выделенную под удалённый элемент *tempNode*,т.е. элемент удаляется из головы списка.
* *long long int top()* возвращает верхний элемент стека, т.е. *mHead->mData*.
* *size\_t size()* возвращает количество элементов в стеке. Переменной *sizeStack* присваивает 0, циклом *while* проходит по всем элементам стека, на каждом шаге прибавляя 1 к *sizeStack,* после цикла возвращает *sizeStack.*
* *~CustomStack() -* деструктор, циклом *while* проходит по всем элементам стека и оператором *delete* освобождает память, выделенную под эти элементы*.*

Функции:

* *bool isArithmeticOperation(char currChar)* проверяет является ли текущий символ знаком операции +, -, \* или /.
* *void performArithmeticOperation(CustomStack &stack, bool &noError, char currChar)* принимает ссылку на стек, ссылку на логическую переменную *noError* и текущий символ. Методом *stack.top()* получает правый операнд выражения, методом *stack.pop()* удаляет верхний элемент стека, методом *stack.top()* получает левый операнд выражения, методом *stack.pop()* удаляет верхний элемент стека. С помощью оператора *switch* и метода *stack.push()* добавляет в стек результат соответствующей арифметической операции. Если при делении правый операнд равен 0, то *noError* присваивается *false*.
* *void processInputSequence(CustomStack &stack, char\* inputSequence, bool &noError)* принимает ссылку на стек, входную последовательность и ссылку на логическую переменную *noError*.С помощью функции *strtok* считывает из входной последовательности число или арифметическую операцию. Циклом *while* проходит до конца последовательности, если считанная строка состоит из 1 символа и этот символ является арифметической операцией, то, если в стеке есть хотя бы 2 элемента вызывается функция *performArithmeticOperation(stack, noError, NumberOrArithmeticOperation[0]),* которая выполняет арифметическую операцию и добавляет результат в стек; если в стеке меньше 2 элементов, то *noError* присваивается *false*. Если считалось число, то оно добавляется в стек.
* *main.* Создаёт новый экземпляр класса *CustomStack,* логической переменной *noError* присваивает *true* (если нет ошибок значение переменной *true,* иначе *false*), функцией *fgets* считывает входную последовательность максимальной длины *MAX\_SEQUENCE\_LENGTH* и последний символ '\n' заменяет на *END\_STRING* ('\0'), затем обрабатывает последовательность функцией *processInputSequence(stack, inputSequence, noError).* Если в стеке осталось больше одного элемента *noError* присваивает *false.* Если ошибок нет выводится число в стеке, иначе выводится "error".

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 | 1 + 2 = 3  3-4 = -1  -1\*5 = -5  3 + (-5) = -2 |
|  | 5 1 1 - / | error | 1 –1 = 0  5/0 - проверка деления на ноль |
|  | 5 1 + 4 | error | в стеке осталось большего 1 элемента |
|  | 7 + 7 | error | для сложения не хватает аргументов в стеке |

## Выводы

Были изучены динамические структуры данных.

Были решены следующие задачи:

* написана программа на языке C++, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе списка.
* реализованы следующие методы класса стека: *void push(int val), void pop(), int top(), size\_t size(), bool empty()*

# Приложение А Исходный код программы

Название файла: main.cpp

#define MAX\_SEQUENCE\_LENGTH 100

#define END\_STRING '\0'

class CustomStack

{

public:

CustomStack()

{

mHead = NULL;

}

bool empty()

{

return mHead == NULL ? true : false;

}

void push(long long int val)

{

ListNode\* tempNode = new ListNode;

tempNode->mNext = mHead;

tempNode->mData = val;

mHead = tempNode;

}

void pop()

{

ListNode\* tempNode = mHead;

mHead = mHead->mNext;

delete tempNode;

}

long long int top()

{

return mHead->mData;

}

size\_t size()

{

long long int sizeStack = 0;

ListNode\* tempNode = mHead;

while (tempNode)

{

sizeStack++;

tempNode = tempNode->mNext;

}

return sizeStack;

}

~CustomStack()

{

ListNode\* head = mHead;

ListNode\* tempNode;

while (head)

{

tempNode = head->mNext;

delete head;

head = tempNode;

}

};

protected:

ListNode\* mHead;

};

bool isArithmeticOperation(char currChar)

{

return (currChar == '+' || currChar == '-' || currChar == '\*' || currChar == '/');

}

void performArithmeticOperation(CustomStack &stack, bool &noError, char currChar)

{

long long int secondNumber = stack.top();

stack.pop();

long long int firstNumber = stack.top();

stack.pop();

switch(currChar)

{

case '+':

stack.push(firstNumber + secondNumber);

break;

case '-':

stack.push(firstNumber - secondNumber);

break;

case '\*':

stack.push(firstNumber \* secondNumber);

break;

case '/':

if (secondNumber == 0) noError = false;

else stack.push(firstNumber / secondNumber);

break;

}

}

void processInputSequence(CustomStack &stack, char\* inputSequence, bool &noError)

{

char \*NumberOrArithmeticOperation = strtok(inputSequence, " ");

while (NumberOrArithmeticOperation != NULL)

{

if (strlen(NumberOrArithmeticOperation) == 1 && isArithmeticOperation(NumberOrArithmeticOperation[0]))

{

if(stack.size() >= 2)

performArithmeticOperation(stack, noError, NumberOrArithmeticOperation[0]);

else noError = false;

}

else stack.push(stoi(NumberOrArithmeticOperation));

NumberOrArithmeticOperation = strtok(NULL, " ");

}

}

int main()

{

CustomStack stack = CustomStack();

bool noError = true;

char inputSequence[MAX\_SEQUENCE\_LENGTH + 1];

fgets(inputSequence, MAX\_SEQUENCE\_LENGTH, stdin);

inputSequence[strlen(inputSequence) - 1] = END\_STRING;

processInputSequence(stack, inputSequence, noError);

if (stack.size() != 1) noError = false;

noError ? cout << stack.top() : cout << "error";

return 0;

}