МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Информационные технологии»

Тема: Динамические структуры данных

Студент гр. 3343 _____ Какира У.Н.

Преподаватель _____ Государкин Я. С.

Санкт-Петербург

2024

Цель работы

Целью работы является изучение языка программирования C++ и использование его 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;
};
```

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:

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

CustomStack() - конструктор класса, инициализирует пустой стек (устанавливает указатель mHead в nullptr).

~CustomStack() - деструктор класса, освобождает память, выделенную для элементов стека. Вызывает метод pop() до тех пор, пока стек не станет пустым.

push(int val) - добавляет новый элемент со значением val на голову стека. Создается новый узел списка, и его указатель на следующий элемент устанавливается на текущую вершину стека. Затем указатель вершины стека обновляется, указывая на новый узел.

pop() - удаляет элемент с головы стека. Если стек пустой, генерируется исключение std::runtime_error. Указатель на голову стека обновляется, указывая на следующий элемент списка, а память, занятая удаляемым элементом, освобождается.

top() - возвращает значение элемента, находящегося в голове стека. Если стек пустой, генерируется исключение std::runtime_error.

size() - возвращает количество элементов в стеке. Подсчитывает количество элементов, перебирая узлы списка, начиная с головы стека.

empty() - проверяет, является ли стек пустым. Возвращает true, если стек пустой (голова указывает на nullptr), и false в противном случае.

В функции main() создается объект класса CustomStack с именем stack. Затем объявляется строковая переменная str, в которую считывается строка с помощью функции ввода std::getline. Далее с цикл for перебирает каждый символ строки str и происходит обработка символа по следующим правилам:

Если символ является пробелом, пропускается, цикл переходит на следующую итерацию

Если символ является цифрой или отрицательным знаком перед цифрой, происходит построение числа, состоящего из последовательности цифр. После сборки строка конвертируется в число и добавляется в стек. Если символ не является цифрой, выполняется арифметическая операция соответствующая символу (сложение, вычитание, умножение, деление), для этого извлекаются два

числа из стека с помощью методов top() и pop(), затем выполняется операция, а результат записывается обратно в стек. Если при извлечении чисел из стека произошла ошибка (стек пуст), отлавливается исключение std::runtime_error и выводится сообщение "error". Далее программа завершается с кодом 0. Если символ не соответствует ни одной из допустимых арифметических операций, выводится сообщение "error", и программа завершается с кодом 0.

Выводы

В ходе выполнения работы была разработана программа, которая вычисляет арифметическое выражение, введенное пользователем в виде строки. Для этого был реализован класс CustomStack, который имеет свои методы: push(int val) – добавляет новый элемент с тек, pop() – удаляет верхний элемент, top() – возвращает голову стека, size() – возвращает размер стека, empty() – проверка на пустоту стека. Были использованы следующие механики: классы, динамическое выделение памяти, указатели, обработка исключений, для создания структуры данных стека и обработки введенного выражения. В результате выполнения программы выводится значение выражения или сообщение об ошибке, если выражение некорректно.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class CustomStack {
protected:
    ListNode* mHead;

public:
    CustomStack() : mHead(nullptr) {}

    ~CustomStack() {
        while (!empty()) {
            pop();
        }
    }

    void push(int val) {
        ListNode* node = new ListNode;
        node->mData = val;
        node->mNext = mHead;
        mHead = node;
    }

    void pop() {
        if (empty()) {
            throw std::runtime_error("error");
        }

        ListNode* tmp = mHead;
        mHead = mHead->mNext;
        delete tmp;
    }

    int top() {
        if (empty()) {
            throw std::runtime_error("error");
        }
        return mHead->mData;
```

```
        }

        size_t size() {
            size_t size = 0;
            ListNode* node = mHead;
            while (node != nullptr) {
                node = node->mNext;
                size++;
            }
            return size;
        }

        bool empty() {
            return mHead == nullptr;
        }
    };

    int main() {
        CustomStack stack;
        std::string str;
        std::getline(std::cin, str);

        for (int i = 0; i < str.size(); i++) {
            if (str[i] == ' ') {
                continue;
            }
            else if (isdigit(str[i]) || (str[i] == '-' &&
isdigit(str[i+1]))) {
                int j = i+1;
                while (isdigit(str[j])) {
                    j++;
                }
                stack.push(std::stoi(str.substr(i, j-i)));
                i = j-1;
            }
            else {
                int num1, num2, res;
                try {
                    num1 = stack.top();
```

```
                stack.pop();
                num2 = stack.top();
                stack.pop();
            }
            catch (const std::runtime_error& error) {
                std::cout << "error" << std::endl;
                return 0;
            }

            switch(str[i]) {
                case '+': res = num1 + num2; break;
                case '-': res = num2 - num1; break;
                case '*': res = num1 * num2; break;
                case '/':
                    if(num1 == 0) {
                        std::cout << "error" << std::endl;
                        return 0;
                    }
                    res = num2 / num1;
                    break;
                default: std::cout << "error" << std::endl; return 0;
            }

            stack.push(res);
        }
    }

    if (stack.size() != 1) {
        std::cout << "error" << std::endl;
        return 0;
    }

    std::cout << stack.top() << std::endl;

    return 0;
}
```