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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: Лабораторная работа № 4: Динамические структуры данных

Студентка гр. 3343 _____ Синицкая Д. В.

Преподаватель _____ Государкин Я. С.

Санкт-Петербург

2024

Цель работы

Целью работы является изучение основных механизмов языка C++ путем разработки структур данных стека и очереди на основе динамической памяти.

Задание

Вариант 1.
1) Реализовать класс CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных int.
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 массив данных
                int* mData;
};
```

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
- extend(int n) - расширяет исходный массив на n ячеек

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.

Выполнение работы

В классе CustomStack в public были реализованы конструктор класса CustomStack(size_t initialCapacity) и следующие методы и функции:

void push(int val) – метод для добавления нового элемента в стек.

void pop() – метод для удаления из стека последнего элемента.

int top() – функция для доступа к верхнему элементу.

size_t size() const – константная функция для определения количества элементов в стеке.

bool empty() const – константная функция для определения заполненности стека.

void extend(int n) – метод для расширения исходного стека на базе массива на n ячеек.

В protected создается указатель на массив данных и описаны следующие методы и функции:

size_t getNewCapacity() const – константная функция для вычисления новой вместимости стека.

void ensureSpace() – метод проверяющий достижение максимального размера.

void resize(size_t NewCapacity) – метод создания стека на базе массива.

В int main() осуществляется чтение строки входных данных, ее разбиение на основе пробелов и запись в массив, проверка в соответствие с условиями задачи, обработка ошибок и вывод результата.

Разработанный программный код см. в приложении А.

Выводы

В ходе выполнения лабораторной работы были изучены и применены на практике структуры данных стека и очереди и базовые механизмы, необходимые для их реализации, на языке C++. Освоены навыки, необходимые для реализации стека в виде класса, а также методов для работы с ним, ввода и вывода данных программы и обработки возможных ошибок в процессе исполнения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
class CustomStack {
public:
    CustomStack(size_t initialCapacity){
         this->mCapacity = initialCapacity; // установка начальной вместимости стека
        this->mData = new int[initialCapacity]; // создание массива int
        this->mIndex = -1; // установка индекса вершины стека на -1 (стек изначально пуст)
    }

    CustomStack() : CustomStack(10){
        // 10 начальный размер стека, вызов другого конструктора
    }

    // освобождение памяти, выделенной под массив данных стека
    ~CustomStack(){
        delete[] this->mData;
    }

    // метод добавления нового элемента в стек
    void push(int val){
            this->ensureSpace(); // проверка, что размера массива достаточно для нового элемента
          this->mIndex++; // увеличение индекса вершины стека для последующей записи нового элемента
           this->mData[this->mIndex] = val; // запись переданного значения в массив данных стека
    }

    // метод удаления из стека последнего элемента
    void pop(){
        if(this->empty()){
            throw logic_error("pop () called on empty stack");
        }
        this->mIndex--; // "удаление" элемента
```

```
    }

    // функция доступа к верхнему элементу
    int top(){
        if(this->empty()){
            throw logic_error("top () called on empty stack");
        }
        return this->mData[this->mIndex];
    }

    // функция возвращения количества элементов в стеке
    size_t size() const{
        return this->mIndex+1;
    }

    // функция проверки на отсутствие элементов в стеке
    bool empty() const{
        return this->mIndex == -1;
    }

    // метод расширения исходного массива на n ячеек
    void extend(int n){
        if(n <= 0){
             throw logic_error("extend () called with a non-positive
argument");
        }
        this->resize(this->mCapacity + n);
    }

protected:
    size_t mCapacity; // вместимость стека
    size_t mIndex; // индекс вершины стека
    int* mData; // указатель на массив данных стека

    size_t getNewCapacity() const{
          // метод для вычисления новой вместимости массива данных
при необходимости расширения
        return this->mCapacity * 3 / 2 + 1;
```

```
    }

    void ensureSpace(){
        if(this->mIndex + 1 == mCapacity){
            // проверка на достижение максимального размера
            size_t NewCapacity = this->getNewCapacity();
            this->resize(NewCapacity);
        }
    }

    void resize(size_t NewCapacity){
        if(NewCapacity == mCapacity){
            return;
        }
        if(NewCapacity < mCapacity){
                throw logic_error("resize () called with a lower
capacity");
        }
        int *newData = new int[NewCapacity];
          // копирование данных при помощи функции из заголовочного
файла <algorithm>
        copy(this->mData, this->mData + this->mCapacity, newData);
           delete[] this->mData; // освобождение памяти, выделенной
для старого массива
            this->mData = newData; // обновление указателя на новый
массив данных
             this->mCapacity = NewCapacity; // обновление значения
вместимости

    }
};

int main() {
    CustomStack stack;
    string s;
    getline(cin, s); // считывание строки
    // создание объекта istringstream для работы со строкой s как с
потоком
```

```
    istringstream stream(s);
    vector<string> n; // вектор для хранения подстрок
    string token; // переменная для хранения каждой подстрки
    // разбиение строки
    while(stream >> token){
        n.push_back(token);
    }
    int size_n = n.size();
    for(int i=0;i<size_n;i++){
          if(n[i]=="+" or n[i]=="-" or n[i]=="*" or n[i]=="/"){ //
элемент операция
            if(stack.size() < 2){
                continue;
            }

            int b = stack.top();
            stack.pop();
            int a = stack.top();
            stack.pop();

            if(n[i]=="+"){
                stack.push(a + b);
            } else{
                if(n[i]=="-"){
                    stack.push(a - b);
                } else {
                    if(n[i]=="*"){
                        stack.push(a * b);
                    } else {
                         if(b!=0){
                           stack.push(a / b);
                         }
                      }
                  }
            }

        } else  { // элемент число
                stack.push(stoi(n[i]));
```

```
            }
        }

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

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

        return 0;
    }
```