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

# ОТЧЕТ

**по лабораторной работе №4**
**по дисциплине «Программирование»**
**Тема: ВВЕДЕНИЕ В ЯЗЫК C++**

Студент гр. 3341 _____ Шуменков А.П.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

**Цель работы**

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

Для достижения поставленной цели требуется решить следующие задачи:

– ознакомиться со структурами данных стека и очереди, особенностями их реализации;

– изучить и использовать базовые механизмы языка C++, необходимые для реализации стека и очереди;

– реализовать индивидуальный вариант стека в виде C++ класса, его операции в виде функций этого класса, ввод и вывод данных программы.

**Задание**

Требуется написать программу, получающую на вход строку, (без кириллических символов и не более 3000 символов) представляющую собой код "простой" html-страницы и проверяющую ее на валидность. Программа должна вывести correct если страница валидна или wrong.

html-страница, состоит из тегов и их содержимого, заключенного в эти теги. Теги представляют собой некоторые ключевые слова, заданные в треугольных скобках. Например, <tag> (где tag - имя тега). Область действия данного тега распространяется до соответствующего закрывающего тега </tag> который отличается символом /. Теги могут иметь вложенный характер, но не могут пересекаться.

<tag1><tag2></tag2></tag1> - верно

<tag1><tag2></tag1></tag2> - не верно

Существуют теги, не требующие закрывающего тега.

Валидной является html-страница, в коде которой всякому открывающему тегу соответствует закрывающий (за исключением тегов, которым закрывающий тег не требуется).

Во входной строке могут встречаться любые парные теги, но гарантируется, что в тексте, кроме обозначения тегов, символы < и > не встречаются. аттрибутов у тегов также нет.

Теги, которые не требуют закрывающего тега: <br>, <hr>.

Стек (который потребуется для алгоритма проверки парности тегов) требуется реализовать самостоятельно на базе массива. Для этого необходимо:

Реализовать класс CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных char*

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
   char** mData;
};
```

Перечень методов класса стека, которые должны быть реализованы:

void push(const char* val) -  добавляет новый элемент в стек

void pop() - удаляет из стека последний элемент

char* 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

Примечания:

Указатель на массив должен быть protected.

Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено(<cstring> и <iostream>).

Предполагается, что пространство имен std уже доступно.

Использование ключевого слова using также не требуется.

**Выполнение работы**

Конструктор CustomStack() инициализирует начальные значения переменных mSize, mData, mCapacity. В данном случае, стек начинает с размера 0, выделяется память под 10 указателей на строки, а емкость стека устанавливается равной 10.

Деструктор ~CustomStack() удаляет каждую строку в массиве mData, освобождает память для массива mData и его содержимого.

Метод push(const char *val) добавляет новый элемент в стек. Если размер стека достиг максимальной емкости, вызывается метод extend(int n), который увеличивает емкость стека на n элементов. Затем выделяется память под новую строку с содержимым val, копируется val в новую строку и указатель на неё добавляется в mData.

Метод pop() удаляет элемент из вершины стека путем освобождения памяти для строки и уменьшает размер стека на 1.

Метод top() возвращает указатель на вершину стека, то есть последний элемент.

Метод size() возвращает текущий размер стека.

Метод empty() проверяет, пуст ли стек и возвращает соответствующее значение.

Метод extend(int n) увеличивает емкость стека на n элементов, создавая временный массив tmp, копируя содержимое mData в tmp, удаляя старый массив mData и заменяя его новым массивом tmp.

Функция input() считывает строку ввода с помощью getline(cin, data), где cin - стандартный ввод, и возвращает считанную строку данных.

Функция checkingInput(const string data, CustomStack& open_tags) принимает строку data и ссылку на объект CustomStack. Она проверяет правильность введенных тегов HTML разметки в строке data.

- Проходит по каждому символу строки data и анализирует его содержимое.

- Если символ - открывающий тег "<", начиная с следующего символа до ">" (не включительно) считывает имя текущего тега в массив res.

- Если тег имеет вид закрывающего "</tag>", он извлекается из стека open_tags и проверяется на соответствие с закрывающим тегом res. Если соответствия нет, функция возвращает false.

- Если тег не является "<br>" или "<hr>", он помещается в стек open_tags.

- После обработки тега, индекс i устанавливается на символ после ">", чтобы пропустить обработку текущего тега.

- В конце функции возвращается true, если все теги были корректно закрыты.

Функция outPut(bool value) выводит сообщение "correct" если value равно true, и "wrong" если value равно false.

В функции main(), программа сначала считывает строку data с помощью input(), затем создает объект CustomStack open_tags, с которым вызывается функция checkingInput(data, open_tags), и результат передается в функцию outPut(), которая выводит соответствующее сообщение.

Код программы – см. Приложение А.

**Тестирование**

Результаты тестирования представлены в табл. 1

Табл. 1 — Результаты тестирования

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1 | <html><head><title>HTML Document</title></head><body> This text is bold,<br>this is bold and italics </body></html> | correct | Тест с moevm |

**Выводы**

В ходе работы были изучены и применены основные механизмы языка C++ для разработки структур данных стека и очереди на основе динамической памяти. Реализован класс CustomStack для работы со стеком, включающий операции push, pop, top, size, empty и метод extend. Был также разработан алгоритм проверки валидности HTML-страницы с помощью стека. Этот опыт позволил глубже понять принципы работы стека, освоить базовые механизмы C++ и их применение для создания сложных структур данных.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {
public:
    CustomStack() {
        mSize = 0;
        mData = new char *[10];
        mCapacity = 10;
    }

    ~CustomStack() {
        for (int i = 0; i < mSize; i++){
            delete[] mData[i];
        }
        delete[] mData;
    }

    void
    push(const char *val) {
        if (mSize >= mCapacity) {
            extend(10);
        }

        mData[mSize] = new char[strlen(val) + 1];
        strcpy(mData[mSize], val);
        mSize++;
    }

    void
    pop() {
        if (mSize > 0) {
            delete[] mData[mSize - 1];
            mSize--;
        }
    }

    char *
    top() {
        if (mSize > 0) {
            return mData[mSize - 1];
        }
        return nullptr;
    }

    size_t
    size(){
        return mSize;
    }

    bool
    empty(){
```

```
        return (mSize == 0);
    }

    void
    extend(int n){
        mCapacity += n;
        char** tmp = new char*[mCapacity];
        for (int i = 0; i < mSize; i++){
            tmp[i] = mData[i];
        }
        delete[] mData;
        mData = tmp;
    }

private:
    int mSize;
    int mCapacity;

protected:
    char** mData;
};

string input(){
    string data;
    getline(cin, data);
    return data;
}

bool checkingInput(const string data, CustomStack& open_tags){
    for (int i = 0; i < data.size(); i++) {
        char res[10];
        res[0] = '\0';
        if (data[i] == '<') {
            int j = i + 1, n = 0;
            while (data[j] != '>'){
                res[n] = data[j];
                n++; j++;
            }
            res[n] = '\0';
            if (res[0] == '/'){
                char* check = open_tags.top();
                for (int k = 1; res[k]; k++)
                    if (check[k - 1] != res[k]) return false;
                open_tags.pop();
            }
            else if (strcmp(res, "br") != 0 && strcmp(res, "hr") !=
0) open_tags.push(res);
            i = j;
        }
    }
    return true;
}

void outPut(bool value){
    if (value) cout << "correct";
    else cout << "wrong";
}
```

```
int main(){
    string data = input();
    CustomStack open_tags;
    outPut(checkingInput(data, open_tags));
    return 0;
}
```