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. 0382 _____ Злобин А. С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

Цель работы.

Научиться работать с динамическими структурами данных с помощью функций языка C++.

Задание.

Вариант 6

Расстановка тегов.

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:
, <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*

Основные теоретические положения.

Язык C++ реализует объектно-ориентированную парадигму программирования, которая включает в себя реализацию механизма инкапсуляции данных . Инкапсуляция в C++ подразумевает, что: в одной языковой конструкции размещаются как данные, так и функции для обработки этих данных.

Доступ к данным извне этой конструкции ограничен, иными словами, напрямую редактировать данные как в структурах C нельзя. Пользователю предоставляется интерфейс из методов (API) с помощью которого он может влиять на состояние данных.

Класс - это шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В классе могут размещаться как данные (их называют полями), так и функции (их называют методы) для обработки этих данных. Любой метод или поле класса имеет свой спецификатор доступа: public, private или protected .

В языке C память можно выделять с помощью библиотечной функции malloc(). Язык C++ предоставляет альтернативный способ - оператор new. Он обеспечивает выделение динамической памяти в куче. Для освобождения выделенной памяти используется оператор delete

Выполнение работы.

В классе CustomStack реализован стек. Поле char ** mData содержит массив строк, в которых хранятся теги. поле mData_size содержит текущий размер стека (количество тегов, хранящихся в стеке). Для класса реализован конструктор, в котором выделяется начальный объём памяти стека, и деконструктор, в котором освобождается память, выделенная под массив. Также реализованы методы согласно заданию: push, pop, top, size, empty, extend.

Метод push копирует строку, хранящуюся по переданной ссылке в стек, затем размер стека увеличивается на 1.

Метод top возвращает указатель на верхний элемент в стеке.

Метод size возвращает текущее количество элементов в стеке.

Метод empty проверяет, существует ли указатель или равно ли количество элементов в стеке 0.

Метод extend увеличивает количество строк, хранящих название тегов, а затем выделяет память под сами строки.

В функции main считывается строка, затем в цикле while перебираются все символы строки. Если встречается символ ‘<’, то последующие символы являются тегом, и происходит их копирование вплоть до символа ‘>’. Далее если тег не является закрывающим или одиночным, то он добавляется в стек. Если тег является закрывающим, то проверяется, равен ли он последнему элементу в стеке и если да, то последний элемент из стека удаляется и происходит считывание следующего тега, а если нет, то это значит, что тэги перепутаны местами и выводится сообщение “wrong”, а затем программа завершается.

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.	<html><head><title> HTML Document</title></head><body> This text is bold, this is bold and italics </body></html>	correct	Программа работает верно

Выводы.

Была разработана программа на языке C++ для проверки валидности html страницы.

Приложение А

Исходный код программы

Название файла: main.cpp

```
#include <iostream>
#include <cstring>

using namespace std;
class CustomStack {
public:

    CustomStack(){
        mData_size = 0;
        mData = (char **) malloc(3000 * sizeof (char * ));
        for (int i = 0; i < 3000; i++) {
            mData[i] = (char *)malloc (100 * sizeof(char));
            mData[i][0] = '\0';
        }
    }

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

    void push(const char* val) { // добавляет новый элемент в стек
        strcpy((mData)[mData_size], val);
        mData_size++;
        (mData)[mData_size][0] = '\0';
    }
    void pop() { // удаляет из стека последний элемент
        mData_size--;
        mData[mData_size][0] = '\0';
    }
    char* top() {// доступ к верхнему элементу
        return mData[mData_size-1];
    }
    size_t size() { // возвращает количество элементов в стеке
        return mData_size;
    }
    bool empty() { // проверяет отсутствие элементов в стеке
        if ((mData) == NULL || mData_size == 0 || mData[0][0] == '\0')
            return true;
        return false;
    }
    void extend(int n) { // расширяет исходный массив на n ячеек
            mData =(char **) realloc(mData, n * sizeof (char * ));
         for (int i = mData_size; i < n; i++) {
                mData[i] = (char *)malloc (100 * sizeof(char));
            mData[i][0] = '\0';
            mData_size += n;
            mData[mData_size][0] = '\0';
    }
```

```
private:
    unsigned long mData_size;
protected:
    char** mData;
};

int main() {
    string inp;
    CustomStack * theBestStackEver = new CustomStack();
    getline(cin, inp);
    char * tag = new char[100];
    int posCount = 0;
    int i = 0, j = 0;
    while (i < inp.size()) {
        if (inp[i] == '<') {
            i++;
            while (inp[i] != '>') {
                tag[j] = inp[i];
                i++;
                j++;
            }
            tag[j] = '\0';
            /*theBestStackEver->push(tag);
            cout << theBestStackEver->top() << endl;
            theBestStackEver->pop(); */
             if (tag[0] != '/'  && strcmp(tag, "br") != 0 && strcmp(tag,
"hr") != 0) {
                theBestStackEver->push(tag);
            } else {
                if (tag[0] == '/') {
                    if (strcmp(theBestStackEver->top(), tag+1)){
                        //cout << theBestStackEver->top() << ' ' << tag
<< endl;
                        cout << "wrong" << endl;
                        return 0;
                    } else {
                        theBestStackEver->pop();
                    }
                }
            }
            j = 0;

        }
        i++;
    }
    cout << "correct" << endl;
    //cout << theBestStackEver->size() << endl;
    delete theBestStackEver;
    return 0;
}
```