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. 0382 _____ Охотникова Г.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Освоить работу с динамическими структурами данных в языке 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.

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

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.

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.

Реализация класса *CustomStack:*

Конструктор *CustomStack()* для инициализации начальных значений.

Деструктор *~CustomStack()*, в котором происходит освобождение памяти.

Метод *void push(const char* val),* в котором, если это необходимо, происходит перевыделение памяти вызовом метода *extend(),* а также добавление элемента в двумерный массив.

Метод *void pop()*, в котором удаляется верхний элемент стека. Если элементов нет, то происходит выход из метода.

Метод *char* top()*, в котором, если стек не пуст, возвращается адрес верхнего элемента.

Метод *size_t size()*, в котором возвращается количество элементов стека.

Метод *bool empty()*, в котором возвращаемое значение равно единице, если стек пуст, и нулю, если в стеке есть элементы.

Метод *extend()*, в котором происходит выделение дополнительной памяти.

В функции *main()* происходит посимвольное считывание и проверка тегов на валидность в цикле *while*. Таким образом, если стек оказывается пуст после всех проверок, то выводится сообщение о том, что теги валидны. Если это не так, выводится сообщение о том, что теги не валидны.

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>HTMLDocument<title><head>\<body> This text is bold, this is bold and italics <	correct	Программа работает верно.

	/body></html>		
2.	<tag1><tag2></tag1></tag2>	wrong	Программа работает верно.
3.	<tag1><tag2></tag2></tag1>	correct	Программа работает верно.

Выводы.

Были исследованы методы работы с динамическими структурами данных на языке C++, а также динамическое выделение и очищение памяти.

Разработана программа, выполняющая считывание с клавиатуры кода простой html-страницы и проверяющая его на валидность.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb4.cpp

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

using namespace std;

class CustomStack {
public:
    explicit CustomStack(int initialSize = 10) {                //
конструктор
        size_d = 0;
        size_max = initialSize;
        mData = new char* [size_max];
    };

    ~CustomStack() {                                            //
деструктор
        for (size_t i = 0; i < size_d; i++) {
            delete [] mData[i];
        }
        delete [] mData;
    }

    void push(const char* val) {
        if (size_d >= size_max) {
            extend(size_plus);
        }
        mData[size_d] = new char;
        strcpy(mData[size_d], val);
        size_d++;
    }

    void pop() {
        if (size_d == 0) {
            return;
        }
        delete [] mData[size_d-1];
        size_d--;
    }

    char* top() {
        if (empty()) {
            return nullptr;
        }
        return mData[size_d-1];
    }

    size_t size() {
        return size_d;
```

```
        }

        bool empty() {
            return (size_d == 0);
        }

        void extend(int n) {
            size_max += n;
            mData = new char* [size_max];
        }

    private:
        size_t size_d;
        size_t size_max;

    protected:
        const size_t size_plus = 20;
        char** mData;
    };

    int main() {
        CustomStack stack(100);
        char* elem = new char [100];
        size_t index = 0;
        bool flag = false;
        char c = cin.get();

        while (c != '\n') {
            if (c == '<') {
                flag = true;
            }
            else if (c == '>') {
                elem[index] = '\0';
                flag = false;
                index = 0;
                if (strcmp(elem, "br") != 0 && strcmp(elem, "hr") !=
0) {
                    if (!stack.empty() && strcmp(elem+1, stack.top())
== 0) {
                        stack.pop();
                    }
                    else {
                        stack.push(elem);
                    }
                }
            }
            else if (flag) {
                elem[index++] = c;
            }
            c = cin.get();
        }

        if (stack.empty()) {
            cout << "correct" << endl;
        }
```

```
    else {
        cout << "wrong" << endl;
    }
    return 0;
}
```