**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. 3342 _____ Епонишникова А.И

Преподаватель _____ Глазунов С.А

Санкт-Петербург

2024

**Цель работы**

Целью работы является на практике изучить динамическую структуру данных стэк на базе списка, а также написать программу, которая проверяет строку, представляющую собой код "простой" html-страницы, на ее валидность.

**Задание**

Вариант 5

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.

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>.

Класс стека (который потребуется для алгоритма проверки парности тегов) требуется реализовать самостоятельно на базе списка. 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* tag) - 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

**Выполнение работы**

file_path(char *dir_name, char *filename, char *path) – в качестве аргументов

Реализован стек на базе списка(ListNode). Имеются следующие методы: push(добавление нового элемента в стек), pop(удаление последнего элемента стека), top(возвращает последний элемент), size(возвращает длину стека), empty(возвращает true, если стек пуст).

Далее происходит проверка полученной строки. Сначала находится открывающаяся и закрывающаяся скобка, строка между ними записывается в tag. Если она удовлетворяет условию(строка не “br” и не “hr”), если нулевой элемент равен “/”, то происходит проверка последнего элемента в стеке и текущего tag. Если совпадают, то удаляет последний элемент, если нет, то выводится wrong и программа завершается. Если нулевой элемент не равен “/”, то элемент добавляется в tag.

Если стек пустой, то выводится correct, иначе wrong.

Разработанный программный код см. в приложении А.

**Тестирование**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

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

**Выводы**

На практике научились работать с динамической структурой данной как стек на базе списка, была реализована программа по установлению валидности строки, которая представляет из себя код html страницы

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack{
    public:
    CustomStack(){
        mHead = nullptr;
        mSize = 0;
    }

    void push(const char* str){
        ListNode *newElement = new ListNode;
        if(newElement == nullptr){
            cout<< "Memory error";
            exit(0);
        }
        newElement->mData = new char[strlen(str)+1];
        if(newElement->mData == nullptr){
            cout<< "Memory error";
            exit(0);
        }
        strcpy(newElement->mData, str);
        newElement->mNext = mHead;
        mHead = newElement;
        mSize++;
    }

    void pop(){
        if (empty())
            return;
        ListNode *deletingElement = mHead->mNext;
        delete[] mHead->mData;
        delete mHead;
        mHead = deletingElement;
        mSize--;
    }

    char* top(){
        if (empty())
            return nullptr;
        return mHead->mData;
    }

    size_t size(){
        return mSize;
    }

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

    ~CustomStack(){
        while(!empty())
```

```
                pop();
        }

        private:
        size_t mSize;

        protected:
        ListNode *mHead;
    };

    int main(){
        CustomStack p;
        string text;
        getline(cin, text);
        string serv_tag1 = "br";
        string serv_tag2 = "hr";
        int start_idx = text.find('<');
        int end_idx = text.find('>');
        while (start_idx != string::npos || end_idx != string::npos) {
            string tag = text.substr(start_idx + 1, end_idx - start_idx
- 1);
            if (tag != serv_tag1 && tag != serv_tag2) {
                if (tag[0] == '/') {
                    string top = p.top();
                    if (top.substr(0) != tag.substr(1)){
                        cout<<"wrong";
                        return 0;
                    }
                    p.pop();
                }
                else {
                    p.push(tag.c_str());
                }
            }
            text = text.substr(end_idx + 1);
            start_idx = text.find('<');
            end_idx = text.find('>');
        }

        if(p.empty()){
            cout<<"correct";
        }
        else{
            cout<<"wrong";
        }
        return 0;
    }
```