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. 3344 _____ Валиев Р.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Изучить динамическую структуру данных «стек». Используя стек, разработать программу, проверяющую корректность html-строки на языке C++.

Задание

Вариант 5.

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. атрибутов у тегов также нет.

Теги, которые не требуют закрывающего тега: *
*, *<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**.

Структура класса узла списка:

```
struct ListNode {
   ListNode* mNext;
   char* mData;
};
```

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 голову
   ListNode* mHead;
};
```

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

Примечания:

1. Указатель на голову должен быть *protected*.
2. Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено(<*cstring*> и <*iostream*>).
3. Предполагается, что пространство имен *std* уже доступно.
4. Использование ключевого слова *using* также не требуется.

Структуру *ListNode* реализовывать самому не надо, она уже реализована.

Выполнение работы

Класс стек *CustomStack* реализован на базе односвязного списка *ListNode*. В блоке *protected* определен указатель на первый элемент стека. В блоке *public* определён конструктор и деструктор, а также методы для работы со стеком. Метод *push* добавляет новый элемент в стек, метод *pop* удаляет из стека последний элемент, метод *top* возвращает указатель на строку верхнего элемента, лежащего в стеке, метод *size* возвращает количество элементов в стеке, метод *empty* проверяет стек на пустоту.

В главной функции считывается строка. Для проверки корректности расстановки тегов в ней используется алгоритм проверки строки на валидную расстановку скобок. Суть алгоритма состоит в том, что каждая новая открывающая скобка (т.е. начало тега) добавляется в стек. Когда в строке встречается закрывающий тег (начинается с символа ‘/’), происходит проверка на наличие соответственного открывающего тега. При нахождении соответствия, извлекается верхний тег из стека, а при его отсутствии цикл прерывается, и результирующему значению приравнивается значение false.

Разработанный программный код см. в приложении А.

Выводы

Была изучена динамическая структура данных «стек», а также реализована программа на языке *C++*, проверяющая строку (код *HTML*-страницы) на корректность расстановки парных тегов. В программе разработан класс стек *CustomStack.*

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {
public:

    CustomStack() {
        mHead = nullptr;
    }

    void push(const char* tag)
    {
        ListNode* Node = new ListNode;
        Node->mData = new char[sizeof(tag) + 1];
        strcpy(Node->mData, tag);
        Node->mNext = mHead;
        mHead = Node;
    }

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

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

    size_t size()
    {
        size_t count = 0;
        if (empty()) {
            return count;
        } else {
            while (mHead)
            {
                count++;
                pop();
            }
            return count;
        }
    }

    bool empty() {return (!mHead);}
```

```
    ~CustomStack()
    {
        while (mHead)
        {
            pop();
        }
    }

protected:
    ListNode* mHead;
};


int main()
{

    string str;
    getline(cin, str);
    CustomStack stack;
    string serv_tag1 = "br";
    string serv_tag2 = "hr";
    int start = str.find('<');
    int end = str.find('>');
    bool result = true;

    while (start != string::npos || end != string::npos)
    {
        string tag = str.substr(start + 1, end - start - 1);
        if (tag != serv_tag1 && tag != serv_tag2) {
            if (tag[0] != '/') {
                stack.push(tag.c_str());
            } else {
                string top_str = stack.top();
                if (top_str.substr(0) != tag.substr(1))
                {
                    result = false;
                    break;
                }
                else {
                    stack.pop();
                }
            }
        }

        str = str.substr(end + 1);
        start = str.find('<');
        end = str.find('>');

    }

    if (result) {
        cout << "correct" << endl;
    } else {
        cout << "wrong" << endl;
    }
    return 0;
}
```