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. 3342 _____ Русанов А.И.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Изучить динамическую структуру данных “стек”. Используя стек, написать программу, решающая задачу определения правильный html-строки на языке программирования C++, используя объектно-ориентированную парадигму программирования.

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.

<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>.

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

Реализован стек на базе линейного списка (класс ListNode). Класс имеет следующие методы: push (добавление элемента в стек), pop (удаление последнего элемента из стека), top (просмотр последнего элемента из стека), size (получение размера стека), empty (проверка стека на пустоту). Для проверки последовательности используется алгоритм нахождения правильной скобочной последовательности, суть которого состоит в том, что каждую новую открывающая скобку (в данной задаче начало тега) добавляют в стек. Когда встречается закрывающая скобка, проверяется, подходит ли она соответствующей закрывающей скобки, если да, то она удаляется из стека, если нет, значит последовательность неверная.

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.	<tag1></tag1>	Correct	Верный вывод
2.	<tag1></tag2>	Wrong	Верный вывод
3.	<tag1><tag2> </tag1><tag2>	Correct	Верный вывод

Выводы

Была реализована программа на языке C++, с использованием объектно-ориентированной парадигмы программирования, которая решает задачу определения корректной html-строки с помощью структуры данных “стек” и алгоритма нахождения правильной скобочной последовательности.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack
{
public:
    CustomStack()
    {
        mHead = new ListNode;
        if (mHead == nullptr)
        {
            cout << "Memory allocation error!";
            exit(1);
        }
        mHead->mNext = nullptr;
    }

    void push(const char *tag)
    {
        if (stack_size == 0)
        {
            mHead->mData = new char[strlen(tag) + 1];
            if (mHead->mData == nullptr)
            {
                cout << "Memory allocation error!";
                exit(1);
            }
            strcpy(mHead->mData, tag);
        }
        else
        {
            ListNode *new_node = new ListNode;
            if (new_node == nullptr)
            {
                cout << "Memory allocation error!";
                exit(1);
            }
            new_node->mData = new char[strlen(tag) + 1];
            if (new_node->mData == nullptr)
            {
                cout << "Memory allocation error!";
                exit(1);
            }
            strcpy(new_node->mData, tag);
            new_node->mNext = mHead;
            mHead = new_node;
        }
        stack_size++;
    }

    void pop()
    {
        ListNode *new_head = mHead->mNext;
```

```
        delete[] mHead->mData;
        delete mHead;
        mHead = new_head;
        stack_size--;
    }

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

    size_t size()
    {
        return stack_size;
    }

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

    ~CustomStack()
    {
        ListNode *tmp = mHead;
        while (tmp->mNext != nullptr)
        {
            tmp = tmp->mNext;
            delete[] mHead->mData;
            delete mHead;
            mHead = tmp;
        }
    }

private:
    size_t stack_size;

protected:
    ListNode *mHead;
};

int main()
{
    string line;
    getline(cin, line);
    int start = line.find('<');
    int end = line.find('>');
    CustomStack stack;
    bool answer = true;
    while (start != string::npos || end != string::npos)
    {
        string tag = line.substr(start + 1, end - (start + 1));
        if (tag != "br" && tag != "hr")
        {
            if (tag[0] != '/')
```

```
            {
                stack.push(tag.c_str());
            }
            else
            {
                if (strcmp(stack.top(), tag.c_str() + 1) != 0)
                {
                    answer = false;
                    break;
                }
                else
                {
                    stack.pop();
                }
            }
        }
        line = line.substr(end + 1);
        start = line.find('<');
        end = line.find('>');
    }
    if (answer)
    {
        cout << "correct";
    }
    else
    {
        cout << "wrong";
    }
    return 0;
}
```