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. Тест.**

Студент гр. 3344 _____ Хангулян С. К.

Преподаватель _____ Глазунов С. А.

Санкт-Петербург

2024

Цель работы

Изучение основ работы с языком C++, изучение и создание самодельного стека на базе однонаправленного списка, функции, способной проверять на валидность код html-страницы.

Задание

Вариант 3

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

Выполнение работы

Реализация стека:

Определены два конструктора и деструктор. Определен метод push, который с помощью цикла идет до конца списка из узлов ListNode и добавляет новый элемент. Определен метод pop, удаляющий верхний элемент стека (последний элемент списка). Стек проверяется на наличие элементов, если их нет – выносится ошибка. С помощью цикла ищется и удаляется последний элемент. Определен метод pop, который аналогично идет до конца списка и возвращает верхний элемент стека. Определен метод size, возвращающий размер стека. Если стек пуст – 0, иначе с помощью цикла считается количество элементов в нем. Наконец, определен метод empty, проверяющий стек на наличие элементов.

Функция check():

r – флаг, определяющий валидность или невалидность строки. Изначально уставлен единицей. Создается самодельный стек, объявляется и считывается строка. Далее идет цикл, который находит теги с помощью вложенного цикла и функции find и записывает их в переменную temp. Если найденный тег является парным, то идет дальнейшая проверка. Если тег является открывающим, то он добавляется в стек. Если он является закрывающим, то, если стек пустой – строка не подходит, происходит выход из цикла. Иначе верхний элемент стека записывается в переменную temp_candidate и сравнивается с закрывающим тегом. Если они не равны – строка не подходит, происходит выход из цикла. По завершении цикла (а это случается либо в одном из вышеперечисленных случаев, либо когда кончаются символы «<») стек проверяется на отсутствие элементов в нем. В противном случае – строка не подходит. Наконец, если флаг до сих пор определен единицей – строка подходит, иначе – нет.

Тестирование

Результаты тестирования представлены в таблице 1.

Таблица 1 – Результаты тестирования

№	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	Корректно

Выводы

Были изучены основы работы с языком C++, был создан самодельный стек на базе однонаправленного списка, была написана функция, способная распознавать на валидность код html-страницы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Khangulyan_Sargis_lb4.cpp

```
class CustomStack{
        public:
        CustomStack(): mHead{nullptr} {}

        CustomStack(ListNode* head): mHead{head} {}

        ~CustomStack() {
            while (!empty())
                pop();
        }

        void push(const char* tag){
            char* new_tag = new char[strlen(tag) + 1];
            strcpy(new_tag, tag);
            if (empty()){
                mHead = new ListNode;
                mHead->mNext = nullptr;
                mHead->mData = (char*)new_tag;
                return;
            }
            ListNode* temp = mHead;
            while(temp->mNext != nullptr){
                temp = temp->mNext;
            }
            ListNode* node = new ListNode;
            node->mNext = nullptr;
            node->mData = (char*)new_tag;
            temp->mNext = node;

            return;
        }

        void pop(){
            if (empty()){
                cout << "error" << endl;
                exit(0);
            }
            ListNode* temp = mHead;
            if(temp->mNext == nullptr){
                delete temp->mNext;
                mHead = nullptr;
                return;
            }
            while(temp->mNext->mNext != nullptr){
                temp = temp->mNext;
            }
            delete temp->mNext->mNext;
            temp->mNext = nullptr;
            return;
        }
```

```
        char* top(){
            if(empty()){
                cout << "error" << endl;
                exit(0);
            }
            ListNode* temp = mHead;
            while(temp->mNext != nullptr){
                temp = temp->mNext;
            }
            return temp->mData;
        }

        size_t size(){
            if (empty()){
                return 0;
            }
            size_t len = 1;
            ListNode * temp = mHead;
            while(temp->mNext != nullptr){
                len++;
                temp = temp->mNext;
            }
            return len;
        }

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

        protected:
            ListNode* mHead;
};

void check(){
    int r = 1;
    CustomStack stack;
    string str;
    getline(cin, str);
    while (1){
        string temp;

        if (str.find("<") == -1){
            break;
        }

        for (int i = str.find("<") + 1; i < str.find(">"); i++){
            temp += str[i];
        }
        str.replace(str.find("<"), 1, "!");
        str.replace(str.find(">"), 1, "!");

        if (temp != "br" && temp != "hr"){
            if (temp[0] != '/'){
                stack.push(temp.c_str());
            }
            else if (temp[0] == '/'){
                if (stack.empty()){
```

```
                    r = 0;
                }
                string temp_candidate = (string)stack.top();
                stack.pop();

                if (('/' + temp_candidate) != temp){
                    r = 0;
                }
            }
        }
    }
    if (r && !stack.empty()){
        r = 0;
    }

    if (r){
        cout << "correct" << endl;
    }
    else{
        cout << "wrong" << endl;
    }
}

int main(){
    check();
    return 0;
}
```