**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](https://ru.wikipedia.org/wiki/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><p><b>This text is bold,<br><i>this  is bold and italics</i></b></p></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 httml 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;

}