**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. 3343 |  | Пухов А.Д. |
| Преподаватель |  | Государкин Я.С. |

Санкт-Петербург

2024

## Цель работы.

Изучение основ и классов в языке программирования С++. Разработка динамической структуры данных стек на основе списка.

## Задание.

**Расстановка тегов.**  
Требуется написать программу, получающую на вход строку, (без кириллических символов и не более 3000 символов) представляющую собой код "простой" [html](https://ru.wikipedia.org/wiki/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: <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\*.**

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.

Функции:

* void push(const char \*tag) — добавляет в конец списка новый элемента.
* void pop() - удаляет последний элемент из списка.
* char \*top() - возвращает последний элемент из списка.
* size\_t size() - возвращает количество элементов в списке.

Переменные:

* char \*html — хранит текст введённый с клавиатуры.
* char \*open — хранит адрес на начало тега.
* char \*close — хранит адрес на конец тега.
* char tag[close — open] — хранит тег.

Разработанную программу смотрите в приложении А.

## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | <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** | ОК |

## Выводы.

В данной лабораторной работе было изучены основы и классы языка программирования С++.

# Приложение А Исходный код программы

Название файла: lb4.cpp

#include <cstring>

#include <iostream>

#define SIZE 3000

class CustomStack

{

public:

size\_t mSize = 0;

void push(const char \*tag)

{

ListNode \*newtag = new ListNode;

newtag->mData = new char[strlen(tag) + 1];

newtag->mNext = mHead;

strcpy(newtag->mData, tag);

mHead = newtag;

mSize++;

}

void pop()

{

if (!empty())

{

ListNode \*temp = mHead;

mHead = mHead->mNext;

delete[] temp->mData;

delete temp;

mSize--;

}

}

char \*top()

{

if (!empty())

{

return mHead->mData;

}

else

{

std::cerr << "Error: Stack is empty\n";

exit(EXIT\_FAILURE);;

}

}

size\_t size()

{

return mSize;

}

bool empty()

{

return mSize == 0;

}

protected:

ListNode \*mHead = NULL;

};

int main()

{

char \*html = new char[SIZE];

fgets(html, SIZE, stdin);

CustomStack stack;

int fl = 0;

char \*open;

char \*close;

open = strchr(html, '<');

close = strchr(html, '>');

while (open != NULL && close != NULL)

{

char tag[close - open];

strncpy(tag, open + 1, close - open - 1);

tag[close - open - 1] = '\0';

if ((tag[0] != '/') && (strcmp(tag,"br\0") != 0) && (strcmp(tag,"hr\0") != 0))

{

stack.push(tag);

if (fl == 0)

{

fl = 1;

}

}

else if ((strcmp(tag,"br\0") != 0) && (strcmp(tag,"hr\0") != 0))

{

if (strstr(tag, stack.top()) == NULL)

{

if (fl == 1)

{

fl = 0;

}

break;

}

else

{

stack.pop();

}

}

open = strchr(open + 1, '<');

close = strchr(close + 1, '>');

tag[0] = '\0';

}

if (fl == 1 && stack.empty())

{

std::cout << "correct" << std::endl;

}

else

{

std::cout << "wrong" << std::endl;

}

if (!stack.empty())

{

while (stack.mSize)

{

stack.pop();

}

}

delete[] html;

}