**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Программирование»**

Тема: **Динамические структуры данных. Тестирование.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Тимофеев А.А. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2017

Оглавление

[**Оглавление** 2](#_Toc483382275)

[Цель: 3](#_Toc483382276)

[Задание: 3](#_Toc483382277)

[Содержание: 3](#_Toc483382278)

[**Вывод** 7](#_Toc483382279)

Цель:

Написание программы с использованием динамических структур данных.

Задание:

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: <br>, <hr>

Стек (который потребуется для алгоритма проверки парности тегов) требуется реализовать самостоятельно на базе списка.

Содержание:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define EXCEPTIONAL\_TAG\_1 "<br>"

#define EXCEPTIONAL\_TAG\_2 "<hr>"

struct stack{

char value[15];

struct stack \*prev;

};

typedef struct stack stack;

stack \*push(stack \*head,char\* elem) {

stack\* tmp = (stack\*)malloc(sizeof(stack));

strcpy(tmp->value, elem);

if(head!=NULL) tmp->prev = head;

else tmp->prev = NULL;

return tmp;

}

stack \*pop(stack \*head) {

stack\* tmp = head->prev;

free(head);

return tmp;

}

int checkingException(char\* str, int i){

char\* exception=(char\*)malloc(sizeof(char)\*4);

int j=0;

for(j=0;j<4;j++)

{ \*(exception+j)=\*(str+i);

i++;

}

\*(exception+j)='\0';

if ((strcmp(exception,EXCEPTIONAL\_TAG\_1))==0 || strcmp(exception,EXCEPTIONAL\_TAG\_2)==0) {

free(exception);

return 1;

}

}

char\* creatingTag(int i,char\* str){

char\* tag=(char\*)malloc(sizeof(char)\*15);

int j=0;

while(\*(str+i)!='>'){

if(\*(str+i)=='/') {

i++;

continue;

}

{

\*(tag+j)=\*(str+i);

j++;

i++;

}

}

\*(tag+j)='\0';

return tag;

}

void checkingStr(char\* str){

stack\* head=NULL;

int i=0;

int len;

len=strlen(str);

char tag[15];

while(i<len){

if(\*(str+i)=='<'){

if(checkingException(str,i)==1) {

i+=4;

continue;

}

else if(\*(str+i+1)!='/') {

head=push(head, creatingTag(i,str));

i+=strlen(head->value);

}

else {

if(strcmp(head->value,creatingTag(i,str))==0) {

i+=strlen(head->value); head=pop(head);

}

else {

printf("wrong");

return;

}

}

}

else i++;

}

if (head==NULL) printf("correct\n");

else printf("wrong\n");

return;

}

int main() {

char\* str=(char\*)malloc(3000\*sizeof(char));

int i=0,k=0;

fgets(str,3000,stdin);

(\*strstr(str,"\n"))= 0;

checkingStr(str);

return 0;

}

# Вывод

В ходе выполнения данной работы было закреплено написание программы, моделирующей работу стека, реализованного на базе списка.