**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Программирование»**

Тема: Динамические структуры данных.

|  |  |  |
| --- | --- | --- |
| Студент гр. 6303 |  | Амбарцумян А.В. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2017

**Цель работы:** провести обзор стандартной библиотеки, реализовать стек на базе односвязного 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>  
  
Стек (который потребуется для алгоритма проверки парности тегов) требуется реализовать самостоятельно на базе**массива**.

**Ход работы:**

1. Подключены необходимые для работы заголовочные файлы

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

1. Объявлена структура – массив, на базе которого будет реализован стек.

typedef struct stack\_typ {

char data[STACK\_SIZE][6];

int size;

} stack\_s;

1. Объявлены и определены все функции, необходимые для работы стековой машины.

*Функция для помещения элемента в стек с проверкой на его переполнение*

int push(stack\_s \*st, char \*v) {

if (st->size == STACK\_SIZE) {

printf("\nstack overflow");

return 0;

}

strcpy(st->data[st->size], v);

st->size++;

return 1;

}

*Функция для удаление элемента из стека( с проверкой на его «пустоту»)*

char\* pop(stack\_s \*st) {

if (st->size == 0) {

printf("\nstack empty");

return 0;

}

st->size--;

return st->data[st->size];

}

*Функция для подсчёта кол-ва элементов в стеке*

int count\_stack(stack\_s st)

{

int i;

while (st.size != 0)

{

i++;

st.size--;

}

return i;

}

*Функция для вывода стека*

void print\_stack(stack\_s \*st) {

printf("\n--%d-\n", st->size);

for (int i = 0; i < st->size; i++) {

printf("%s | ", st->data[i]);

}

}

*Функция для возвращения вершины стека*

char\* return\_head(stack\_s \*st)

{

if (st->size != 0) return st->data[st->size-1];

else return 0;

}

**В main()**

1. Объявлен флаг ошибки, массив char str[3001], в которую будет записана поданная строка.

Проинициализирована структура стека

int error = 0;

char str[3001];

stack\_s st;

st.size = 0;

fgets(str, 3001, stdin);

1. Посимвольно перебираем строку для нахождения открывающего тега. Если следующий символ «/» , значит считаем это закрывающим тегом, помечаем его и запоминаем его позицию

Копируем тег из строки str в строку teg и добавляем признак конца строки

memcpy(tag, &str[s], e - s);

tag[e-s] = '\0';

Для тегов <br> и <hr> ничего не делаем, т.к. им не нужны закрывающие теги

if((strcmp(tag,"br")==0)||(strcmp(tag, "hr")==0)) {

close = 0;

open = 0;

continue;

}

Если стек не пуст и существует открытый тег, выполняем сравнение. Если теги совпадают, удаляем тег из стека и обнуляем переменные.

Если нет пары – ошибка

Если стек пуст – добавляем тег в стек

if (st.size != 0 && close == 1) {

if ( strcmp(pop(&st), tag) == 0 ) {

close = 0;

open = 0;

} else {

error = 1;

}

} else {

push(&st, tag);

}

} // end if >

}

1. Если ошибок нет и стек пустой, то пара есть у каждого тега , иначе выводим сообщение об ошибки

if (st.size == 0 && error == 0 && open == 0 && close == 0) {

printf("correct");

} else {

printf("wrong");

}

return 0;

}

1. Затем файл lab1.c с кодом написанной программы и отчёт о проведенной работе был загружен на github c помощью консоли:

* Создана новая ветка Ambartsumyan\_lr\_2

git checkout –b Ambartsumyan\_lr\_2

* Создана новая директория Ambartsumyan\_lr\_2 и файл lab2.c в ней, куда помещен код написанной программы

mkdir Ambartsumyan\_lr\_2

cd Ambartsumyan\_lr\_2

nano lab2.c

* Созданные файлы добавлены для загрузки

git add Ambartsumyan\_lr\_2/

* Добавлен комментарий

git commit –m “2 Lab”

* Файлы загружены на репозиторий Git

git push origin Ambartsumyan\_lr\_2

**Вывод:** в ходе работы проведен обзор стандартной библиотеки и реализован стек на базе массива.