**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. 9304 |  | Каменская Е.К. |
| Преподаватель |  | Чайка К.В |

Санкт-Петербург

2020

## Цель работы.

Используя возможности языка C++, написать собственный класс для создания структуры данных стэк и работы с ней на основе односвязного списка.

## Задание.

Вариант 5

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>

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:

Указатель на голову должен быть protected.

Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено(<cstring> и <iostream>)

Предполагается, что пространство имен std уже доступно

Использование ключевого слова using также не требуется

Структуру ListNode реализовывать самому не надо, она уже реализована

## Выполнение работы.

**Структура ListNode** содержит указатель на следующий элемент односвязного списка и данные mData.

**Класс CustomStack:**

struct ListNode\* mHead — указатель на начало стэка.

size\_t curSize — текущий размер стэка.

void push(const char\* data) — создает новый элемент, записывает указатель на него в предыдущий.

void pop() - если стэк не пуст, удаляет последний элемент и изменяет указатель предпоследнего на NULL. В противном случае, не делает ничего.

char\* top() - если стэк пуст, вернет NULL, иначе — значение mData верхнего элемента стэка.

size\_t size() - возвращает размер стэка.

bool isEmpty() - возвращает true, если стэк пуст и false в противном случае.

## Выводы.

В ходе выполнения работы были изучены способы работы с динамической памятью на языке C++, создан свой класс с различными уровнями доступности переменных.

# Приложение А Исходный код программы

Название файла: stack.cpp

/\*#include <string>

#include <iostream>

#include <cstring>

using namespace std;

typedef struct ListNode {

ListNode\* mNext;

char\* mData;

}ListNode;\*/

class CustomStack {

public: // методы push, pop, size, empty, top + конструкторы, деструктор

CustomStack(){

mHead = new struct ListNode;

mHead->mData = NULL;

mHead->mNext = NULL;

curSize = 0;

}

~CustomStack(){

struct ListNode\* mTmp = NULL;

while(mHead->mNext){

delete[] mHead->mData;

mTmp = mHead;

mHead = mHead->mNext;

delete mTmp;

}

delete[] mHead->mData;

delete mHead;

}

void push(const char\* data){

struct ListNode\* mTmp = mHead;

while(mTmp->mNext) // reach last element

mTmp = mTmp->mNext;

struct ListNode\* newNode = new struct ListNode;

newNode->mData = new char [strlen(data)+1];

strcpy(newNode->mData, data);

newNode->mNext = NULL;

mTmp->mNext = newNode;

curSize++;

}

void pop(){

if(!isEmpty()){

struct ListNode\* mTmp = mHead;

struct ListNode\* mPrev = NULL;

while(mTmp->mNext){ // reach last element

mPrev = mTmp;

mTmp = mTmp->mNext;

}

delete[] mTmp->mData;

delete mTmp;

curSize--;

if(mPrev)

mPrev->mNext = NULL;

}

}

char\* top(){

if(isEmpty())

return NULL;

struct ListNode\* mTmp = mHead;

while(mTmp->mNext) // reach last element

mTmp = mTmp->mNext;

return mTmp->mData;

}

size\_t size(){

return curSize;

}

bool isEmpty(){

return (curSize==0);

}

private: // поля класса, к которым не должно быть доступа извне

protected: // в этом блоке должен быть указатель на голову

struct ListNode\* mHead;

//ListNode\* mTail;

size\_t curSize;

};

int main(){

char text[3001];

string line;

getline(cin, line);

strcpy(text, line.c\_str());

bool isCorrect = true;

try{

CustomStack stack;

int i = 0;

char\* start = NULL;

while(isCorrect && text[i]!='\0'){

if(text[i] == '<'){

start = text + i; // mark the start

}

if(text[i] == '>'){

char temp = text[i+1];

text[i+1] = '\0';

if(strcmp(start, "<br>")!=0 && strcmp(start, "<hr>")!=0){ // not br hr

if(start[1] == '/'){ // if closing tag

char\* lastTag = stack.top();

if(!lastTag){

isCorrect = false;

break;

}

if(!strcmp(&start[2], &lastTag[1])){

stack.pop();

start = NULL;

}else{

isCorrect = false;

}

}else{ // if opening tag

stack.push(start);

}

}

text[i+1] = temp;

}

i++;

}

if(!stack.isEmpty()){

isCorrect=false;

}

if(isCorrect)

cout << "correct" << endl;

else

cout << "wrong" << endl;

}

catch(bad\_alloc& ex){

cout << "Error: " << ex.what() << endl;

}

return 0;

}