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. 1304 Спасов Д.В.

Преподаватель Чайка К.В.

Санкт-Петербург
2022

Цель работы.

Написать программу в соответствии с условием задачи.

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:
, <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*.**

Выполнение работы.

Структура Teg – хранит флаги для определения типа тега и его состояния (закрывающий/ открывающий).

Функция html_check – принимает на вход строку, из нее сохраняет только теги в стек. После проверки в массив типа Teg сохраняются только парные теги. Далее идет проверка на парность, если количество пар равно половине количества элементов, то выводится “correct”, в противном случае выводится “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	Комментарии
1.	<html><head><title>HTML Document</title></head><body> This text is bold, this is bold and italics </body></html>	correct	Ответ верный

Вывод.

В соответствии с условием задачи была реализована программа.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Н**азвание файла: main.cpp**

```
struct Teg{
    int open;
    int close;
    char* type;
};
class CustomStack {

public:
    CustomStack(){
        mHead=new ListNode;
        mHead=NULL;
```

```
    size_st=0;
}

void push(const char * el){

    ListNode* new_el=new ListNode;
    new_el->mData=new char[strlen(el)];
    strcpy(new_el->mData,el);
    if(mHead==NULL){
        mHead=new_el;
    }
    else{
    new_el->mNext=mHead;
    mHead=new_el;
    }
    size_st++;

}
char*top(){
   return mHead->mData;
}
void print(){
    while(mHead!=NULL){
        puts(mHead->mData);
        mHead=mHead->mNext;

    }
}
size_t size(){

    return size_st;
}
void pop() {
    ListNode* buff = mHead;
    mHead = mHead->mNext;
    size_st--;

}
bool empty(){
    if(mHead==NULL)return true;
```

```
    else return false;
}

void  html_check(char * s){
    char one_el[3000];
    int kol,check=0;

    for(int i=0;i< strlen(s);i++){
        if(s[i]=='<' && check==0){
            kol=0;
            check=1;
            continue;
        }
        if(check==1&&s[i]!='>'){
            one_el[kol]=s[i];
            kol++;
        }
        if(s[i]=='>'){
            one_el[kol]='\0';
            kol=0;
            check=0;

            push(one_el);

        }

    }


    char answer[1000][1000];
    int kol_teg=0;


          //  size_st--;
            pop();
        }else{
            strcpy(answer[kol_teg],top());
            pop();
            //size_st--;
            kol_teg++;
```

```
    }
  }

  if(kol_teg%2!=0 || kol_teg<0){
    cout<< "wrong"<<endl;

  }else{

    struct Teg teg[kol_teg];
    for(int i=0;i<kol_teg;i++){
      if(answer[i][0]=='/'){

        teg[i].type=new char[strlen(answer[i])];
        strcpy(teg[i].type,&answer[i][1]);
        teg[i].open=0;
        teg[i].close=1;
      }else{

        teg[i].type=new char[strlen(answer[i])];
        strcpy(teg[i].type,answer[i]);
        teg[i].open=1;
        teg[i].close=0;
      }
    }
    int kol_checked=0;
   for(int i=0;i<kol_teg;i++){
      int kol_ins_o=0;
      int kol_ins_c=0;
      for(int j=kol_teg-1;j>i;j--){

        if(strcmp(teg[i].type,teg[j].type)==0 &&
teg[i].close==1 && teg[i].open==0 && teg[j].close==0 && teg[j].open==1){
          //cout<<"okok"<<endl;
          for(int q=i;q<=j;q++){
            if(teg[q].close==1)kol_ins_c++;
            if(teg[q].open==1)kol_ins_o++;
          }
         // cout<<kol_ins_o<<" "<<kol_ins_c<<endl;
          if(kol_ins_o==kol_ins_c){
            teg[i].open=1;
```

```
                    teg[j].close=1;
                    kol_checked++;
                }

            }
        }

    }
    if(kol_teg/2==kol_checked)cout<<"correct"<<endl;
    else cout<<"wrong"<<endl;
  }

}


protected:
  ListNode* mHead;
  size_t size_st;

};


int main(){
  CustomStack st;

  char* s= new char[3000] ;
  cin.getline(s,3000,'\n');
  st.html_check(s);
  return 0;
}while(size_st>0){

    if(strcmp(top(),"hr")==0 || strcmp(top(),"br")==0){
      //  size_st--;
        pop();
    }else{
        strcpy(answer[kol_teg],top());
        pop();
        //size_st--;
        kol_teg++;
    }
```

```
}

if(kol_teg%2!=0 || kol_teg<0){
    cout<< "wrong"<<endl;

}else{

    struct Teg teg[kol_teg];
    for(int i=0;i<kol_teg;i++){
        if(answer[i][0]=='/'){

            teg[i].type=new char[strlen(answer[i])];
            strcpy(teg[i].type,&answer[i][1]);
            teg[i].open=0;
            teg[i].close=1;
        }else{

            teg[i].type=new char[strlen(answer[i])];
            strcpy(teg[i].type,answer[i]);
            teg[i].open=1;
            teg[i].close=0;
        }
    }
    int kol_checked=0;
   for(int i=0;i<kol_teg;i++){
        int kol_ins_o=0;
        int kol_ins_c=0;
        for(int j=kol_teg-1;j>i;j--){

                if(strcmp(teg[i].type,teg[j].type)==0 &&
teg[i].close==1 && teg[i].open==0 && teg[j].close==0 && teg[j].open==1){
                    //cout<<"okok"<<endl;
                    for(int q=i;q<=j;q++){
                        if(teg[q].close==1)kol_ins_c++;
                        if(teg[q].open==1)kol_ins_o++;
                    }
                   // cout<<kol_ins_o<<" "<<kol_ins_c<<endl;
                    if(kol_ins_o==kol_ins_c){
                        teg[i].open=1;
                        teg[j].close=1;
```

```
                    kol_checked++;
                }

            }
        }

    }
    if(kol_teg/2==kol_checked)cout<<"correct"<<endl;
    else cout<<"wrong"<<endl;
  }

 }


protected:
 ListNode* mHead;
 size_t size_st;

};


int main(){
 CustomStack st;

 char* s= new char[3000] ;
 cin.getline(s,3000,'\n');
 st.html_check(s);
 return 0;
}
```