StringProcessing.h

#ifndef **INC\_22\_02\_\_STRINGPROCESSING\_H\_**#define **INC\_22\_02\_\_STRINGPROCESSING\_H\_**#include<iostream>  
#include <algorithm>  
#include <string.h>  
#include <sstream>  
#include <vector>  
#include <fstream>  
#include <iterator>  
  
class StringProcessing {  
 public:  
 static void strchr(char\* str, int ch,char\* new\_str);  
 static void strchr(const std::string& str, int ch,std::string new\_str);  
  
 static void compact(char\* str,char\* new\_str);  
 static void compact(std::string& );  
  
 static void deleteWord(char\* str,char \*word,char\* new\_str);  
 static void deleteWord(std::string& str,const std::string& word);  
  
 static void theLongest(char\* str,char\* the\_longest);  
 static void theLongest(const std::string& str,std::string the\_longest);  
  
 static void reverse(char \*str,char\* new\_str);  
 static void reverse(std::string str,std::string rev\_str);  
  
 static void replace(std::string& input\_str, const std::string& replaced\_str,const std::string& new\_str);  
 static void replace(char \*input\_str, char \*replaced\_str, char \*rep\_str,char\* new\_str);  
  
 static std::ofstream writeComments(std::string infile\_name, std::string outfile\_name);  
 static std::ofstream writeCommentsCstring(std::string in\_file, std::string output\_file);  
  
};  
  
#endif *//INC\_22\_02\_\_STRINGPROCESSING\_H\_*

StringProcessing.cpp

#include "StringProcessing.h"  
void StringProcessing::strchr(char\* str, int ch,char\* new\_str) {  
 for (; str!=""; ++str) {  
 if (\*str == ch)  
 new\_str=str;  
 return;  
 }  
 return;  
}  
  
void StringProcessing::strchr(const std::string& str, int ch,std::string new\_str) {  
 auto iterator=std::find(str.begin(),str.end(),(char)ch);  
 std::string result;  
 std::copy(iterator,str.end(),std::inserter(result,result.begin()));  
 new\_str=str;  
}  
  
void StringProcessing::compact(char\* str,char\* new\_str) {  
 for(int i=0;str[i]!='\0';i++){  
 if(str[i]==' ' && str[i+1]==' ') {  
 for (int j = i; str[j] != '\0'; j++) {  
 str[j] = str[j + 1];  
 }  
 i--;  
 }  
 }  
 new\_str=str;  
}  
  
void StringProcessing::compact(std::string& str) {  
 int pos = 0;  
 while (pos != std::string::npos) {  
 pos = str.find(" ");  
 if (pos != std::string::npos) {  
 str.erase(pos, 1);  
 pos++;  
 }  
 }  
}  
  
void StringProcessing::deleteWord(char\* str,char \*word,char\* new\_str) {  
 int word\_len = strlen(word);  
 char \*word\_begin;  
 strchr(str, int(word[0]),word\_begin);  
 int new\_len = strlen(word\_begin);  
 int amount = 0;*//количество совпадающих букв* char\* compacted\_str;  
 if (word\_begin) {  
 while (\*word\_begin != '\0' && \*word\_begin == \*word) {  
 ++word\_begin;  
 ++word;  
 ++amount;  
 }  
 if (amount == word\_len && \*word\_begin == ' ') {  
 int start\_elem = strlen(str) - new\_len;  
 for (int j = 0; j < amount; ++j) {  
 str[start\_elem] = ' ';  
 start\_elem++;  
 }  
 compact(str,compacted\_str);  
 }  
 }  
 new\_str=compacted\_str;  
}  
  
void StringProcessing::deleteWord(std::string& str,const std::string& word) {  
 size\_t pos = 0;  
 size\_t length = word.size();  
 pos = str.find(word, pos);  
 while (pos != std::string::npos) {  
 str.replace(pos, length, "");  
 pos += length;  
 pos = str.find(word, pos);  
 }  
 compact(str);  
}  
  
void StringProcessing::theLongest(char \*str,char\* the\_longest) {  
 char\* word;  
 int i=0,max\_len=0;  
  
 while(str!="\0"){  
 int j=0;  
 while(\*str!=' '){  
 \*word=\*str;  
 word++;  
 str++;  
 }  
 if(strlen(word)>max\_len){  
 the\_longest=word;  
 }  
 }  
}  
  
void StringProcessing::theLongest(const std::string& str,std::string the\_longest) {  
 std::stringstream str\_stream(str);  
 std::string word;  
 while(str\_stream>>word){  
 if(word.size()>the\_longest.size()){  
 the\_longest=word;  
 }  
 }  
}  
  
void StringProcessing::reverse(char \*str,char\* new\_str) {  
 int len = strlen(str);  
 int k = 0,l=-1;  
 for (int i = 0; i <= len; i++) {  
 if (str[i] == ' ') {  
 for (int j = i - 1; j >= k; j--) {  
 new\_str[l+1] = str[j];  
 l++;  
 }  
 new\_str[l+1] = ' ';  
 l++;  
 k = i + 1;  
 }  
 }  
  
 *//last word* l++;  
 new\_str[l] =' ';  
 for (int j = len - 1; j >= k; j--) {  
 l++;  
 new\_str[l] = str[j];  
 }  
  
}  
  
void StringProcessing::reverse(std::string str,std::string rev\_str) {  
 *//divide string into words* std::vector<std::string> words;  
 int count = 0;  
 while (count != str.size()) {  
 int word\_count=0;  
 count++;  
 word\_count++;  
 while(str[count]!=' '){  
 word\_count++;  
 count++;  
 }  
 std::copy(str[count-word\_count],str[count],std::back\_inserter(words));  
}  
  
 *//make reversed str* std::string word;  
 for(int i = 0; i != words.size(); ++i) {  
 word = words[i];  
 std::reverse(word.begin(), word.end());  
 rev\_str += word + ' ';  
 }  
}  
  
  
void StringProcessing::replace(std::string &input\_str, const std::string &replaced\_str,const std::string &new\_str) {  
 while(auto pos = input\_str.find(replaced\_str,0) != std::string::npos)  
 input\_str.replace(pos,replaced\_str.length(),new\_str);  
}  
  
void StringProcessing::replace(char \*input\_str, char \*replaced\_str, char \*rep\_str,char\* new\_str) {  
 bool flag= true;  
 for(int i=0;i<strlen(input\_str);i++){  
 if(input\_str[i]==replaced\_str[0]){  
 int k=0;  
 for(int j=i;j<strlen(replaced\_str);j++){  
 if(input\_str[j]==replaced\_str[k]){  
 k++;  
 continue;  
 }  
 flag=false;  
 }  
  
 if(flag){  
 for(int j=0;j<strlen(rep\_str);j++){  
 \*new\_str=rep\_str[j];  
 new\_str++;  
 }  
 continue;  
 }  
 }  
  
 \*new\_str++=' ';  
 }  
}  
  
std::ofstream StringProcessing::writeComments(std::string in\_file, std::string output\_file) {  
 std::ifstream in(in\_file);  
 std::ofstream out(output\_file);  
 while (!in.eof()) {  
 std::string string;  
 getline(in, string);  
 int len = string.size();  
 char\* str=new char[len];  
 for (int i = 0; i<string.size(); i++) {  
 str[i]=string[i];  
 }  
 for (int i = 0; i < (len-1); i++) {  
 if (str[i] == '/' && str[i + 1] == '/') {  
 for (int j = i + 1; j < (len - 1); j++) {  
 out << str[j + 1];  
 }  
 out << "\n";  
 }  
 }  
 }  
 return out;  
}  
  
std::ofstream StringProcessing::writeCommentsCstring(std::string in\_file, std::string output\_file) {  
 std::ifstream in(in\_file);  
 std::ofstream out(output\_file);  
 while (!in.eof()) {  
 std::string str;  
 getline(in, str);  
 std::copy\_if(str.begin(),str.end(),  
 std::ostream\_iterator<std::string>(out, "\n"),  
 [](std::string str) { return (str[0] == '/' && str[1] == '/'); });  
 }  
 return out;  
}

tests.cpp

#include <gtest/gtest.h>  
#include "StringProcessing.h"  
#include "StringProcessing.cpp"  
  
**TEST**(TestFunction,strchr\_cstr){  
 char str[]="asdfg";  
 int ch='d';  
 char excepted[]="dfg";  
 char\* new\_str;  
 StringProcessing::strchr(str,ch,new\_str);  
 **EXPECT\_STREQ**(new\_str,excepted);  
}  
  
**TEST**(TestFunction,strchr\_str){  
 const std::string str="asdfg";  
 int ch='f';  
 std::string excepted="fg";  
 std::string new\_str;  
 StringProcessing::strchr(str,ch,new\_str);  
 **EXPECT\_EQ**(new\_str,excepted);  
}  
  
**TEST**(TestFunction,compact\_cstr){  
 char str[]="as df g";  
 char excepted[]="as df g";  
 char\* compacted\_str;  
 StringProcessing::compact(str,compacted\_str);  
 **EXPECT\_STREQ**(compacted\_str,excepted);  
}  
**TEST**(TestFunction,compact\_str){  
 std::string str="as df g";;  
 std::string excepted="as df g";  
 StringProcessing::compact(str);  
 **EXPECT\_EQ**(str,excepted);  
}  
  
**TEST**(TestFunction,deleteWord\_cstr){  
 char str[]="as df g";  
 char del\_str[] ="df";  
 char excepted[]="as g";  
 char\* deleted;  
 StringProcessing::deleteWord(str,del\_str,deleted);  
 **EXPECT\_STREQ**(deleted,excepted);  
}  
  
**TEST**(TestFunction,deleteWord\_str){  
 std::string str="as df g";;  
 std::string del\_str="df";  
 std::string excepted="as g";  
 StringProcessing::deleteWord(str,del\_str);  
 **EXPECT\_EQ**(str,excepted);  
}  
  
**TEST**(TestFunction, theLongest\_cstr){  
 char str[] = "a aaa aa a";  
 char excepted[]="aaa";  
 char\* the\_longest;  
 StringProcessing::theLongest(str,the\_longest);  
 **EXPECT\_STREQ**(the\_longest,excepted);  
}  
  
**TEST**(TestFunction, theLongest\_str){  
 std::string str="a aaa aa a";  
 std::string excepted="aaa";  
 std::string the\_longest;  
 StringProcessing::theLongest(str,the\_longest);  
 **EXPECT\_EQ**(the\_longest,excepted);  
}  
  
  
**TEST**(TestFunction,reverse\_cstring){  
 char str[]="ab Abc bA";  
 char rev\_str[]="ba cbA Ab";  
 char\* reversed;  
 StringProcessing::reverse(str,reversed);  
 **EXPECT\_STREQ**(reversed,rev\_str);  
}