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);
}
```