МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №5е

по дисциплине: Основы программирования
тема: «Работа со строками»

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Притчин Иван Сергеевич
Черников Сергей Викторович

Белгород 2022 г.

Цель работы: получение навыков работы со строками в стиле C.
Содержание отчёта:

- Тема лабораторной работы.
- Цель лабораторной работы.
- Исходный код string_.h / string_.c и решения задач.
- Фрагменты кода разделяйте по задачам и пропишите условия (допустимо комментариями).
- Ссылка на открытый репозиторий с решением.
- Скриншот с историей коммитов.

Ссылка на репозиторий: https://github.com/ChuvilkoDEV/lab-5d

Задания к лабораторной работе:
Содержимое библиотеки `string_.h`

```c
#include <ctype.h>
#include "string_.h"

# define ASSERT_STRING(expected, got) assertString(expected, got, \
__FILE__, __FUNCTION__, __LINE__)

// Первая реализация функции, которая возвращает размер строки
size_t strlen1(char *a) {
  size_t i = 0;
  while (a[i] != '\0')
    i++;
  return i;
}

// Вторая реализация функции, которая возвращает размер строки
size_t strlen2(char *s) {
  int i = 0;
  while (*s != '\0') {
    i++;
    s++;
  }
  return i;
}

// Возвращает размер строки
size_t strlen_(const char *begin) {
  char const *end = begin;
  while (*end != '\0')
    end++;
  return end - begin;
}

// Возвращает указатель на первый элемент с кодом ch, расположенным на ленте
// памяти между адресами begin и end не включая end. Если символ не найден,
// возвращается значение end.
char *find(char *begin, char *end, int ch) {
  while (begin != end && *begin != ch)
    begin++;
  return begin;
}

// Возвращает указатель на первый символ, отличный от пробельных, расположенный
// на ленте памяти, начиная с begin и заканчивая ноль-символом. Если символ
// не найден, возвращается адрес первого ноль-символа.
char *findNonSpace(char *begin) {
  while (isspace(*begin))
    begin++;
  return begin;
}
```

```c
// возвращает указатель на первый пробельный символ, расположенный на ленте
// памяти начиная с адреса begin или на первый ноль-символ.
char *findSpace(char *begin) {
  while (!isspace(*begin) && *begin != '\0')
    begin++;
  return begin;
}

// Возвращает указатель на первый справа символ, отличный от пробельных,
// расположенный на ленте памяти, начиная с rbegin (последний символ
// строки, за которым следует ноль-символ) и заканчивая rend
// (адрес символа перед началом строки). Если символ не найден, возвращается
// адрес rend.
char *findNonSpaceReverse(char *rbegin, const char *rend) {
  while (rbegin >= rend && isspace(*rbegin))
    rbegin--;
  return rbegin;
}

// Возвращает указатель на первый пробельный символ справа, расположенный на
// ленте памяти, начиная с rbegin и заканчивая rend. Если символ не найден,
// возвращается адрес rend.
char *findSpaceReverse(char *rbegin, const char *rend) {
  while (rbegin >= rend && !isspace(*rbegin))
    rbegin--;
  return rbegin;
}

// Функция возвращает отрицательное значение, если lhs располагается до rhs
// в лексикографическом порядке (как в словаре), значение 0, если lhs и rhs
// равны, иначе – положительное значение.
int strcmp_(const char *lhs, const char *rhs) {
  while (*lhs && *lhs == *rhs)
    lhs++, rhs++;
  return *lhs - *rhs;
}

// Записывает по адресу beginDestination фрагмент памяти, начиная с адреса
// beginSource до endSource. Возвращает указатель на следующий свободный
// фрагмент памяти в destination
char *copy(const char *beginSource, const char *endSource,
           char *beginDestination) {
  while (beginSource < endSource) {
    *beginDestination = *beginSource;
    beginSource++;
    beginDestination++;
  }
  return beginDestination;
}

// Записывает по адресу beginDestination элементы из фрагмента памяти начиная с
// beginSource заканчивая endSource, удовлетворяющие функции-предикату f.
// Функция возвращает указатель на следующий свободный для записи фрагмент в
// памяти.
char *copyIf(char *beginSource, const char *endSource,
             char *beginDestination, int (*f)(int)) {
  while (beginSource < endSource) {
    if (f(*beginSource))
      *beginDestination++ = *beginSource;
    beginSource++;
  }
  return beginDestination;
}
```

```c
// Записывает по адресу beginDestination элементы из фрагмента памяти начиная с
// rbeginSource заканчивая rendSource, удовлетворяющие функции-предикату f.
// Функция возвращает значение beginDestination по окончанию работы функции.
char *copyIfReverse(char *rbeginSource, const char *rendSource,
                    char *beginDestination, int (*f)(int)) {
  while (rbeginSource > rendSource) {
    if (f(*rbeginSource)) {
      *beginDestination = *rbeginSource;
      beginDestination++;
    }
    rbeginSource--;
  }
  return beginDestination;
}

// Возвращает адрес конца строки
char *getEndOfString(char *c) {
  return c + strlen_(c);
}

// Возвращает адрес начала слова
char *getBeginOfWord(char *c) {
  while (*c != '\0' && isspace(*c))
    c++;
  return c;
}

void assertString(const char *expected, char *got,
                  char const *fileName, char const *funcName,
                  int line) {
  if (strcmp_(expected, got)) {
    fprintf(stderr, "File %s\n", fileName);
    fprintf(stderr, "%s - failed on line %d\n", funcName, line);
    fprintf(stderr, "Expected : \t\"%s \"\n", expected);
    fprintf(stderr, "Got: \t\t\"%s\"\n\n", got);
  } else
    fprintf(stdout, "%s - OK\n", funcName);
}

void printErrorInt(const int expected, int got,
                   char const *fileName, char const *funcName,
                   int line) {
  fprintf(stderr, "File %s\n", fileName);
  fprintf(stderr, "%s - failed on line %d\n", funcName, line);
  fprintf(stderr, "Expected : \t\"%d\"\n", expected);
  fprintf(stderr, "Got: \t\t\"%d\"\n\n", got);
}

void assertInt(const int expected, int got,
               char const *fileName, char const *funcName,
               int line) {
  if (expected != got) {
    fprintf(stderr, "File %s\n", fileName);
    fprintf(stderr, "%s - failed on line %d\n", funcName, line);
    fprintf(stderr, "Expected : \t\"%d\"\n", expected);
    fprintf(stderr, "Got: \t\t\"%d\"\n\n", got);
  } else
    fprintf(stdout, "%s - OK\n", funcName);
}
```

```c
// Возвращает значение 0, если слово не было считано, в противном
// случае будет возвращено значение 1 и в переменную word типа WordDescriptor
// будут записаны позиции начала слова, и первого символа после конца слова
int getWord(char *beginSearch, WordDescriptor *word) {
  word->begin = findNonSpace(beginSearch);

  if (*word->begin == '\0')
    return 0;

  word->end = findSpace(word->begin);

  return 1;
}

int wordlen(char *c) {
  char *end = c;
  while (!isspace(*end))
    end++;
  return end - c;
}

// Возвращает 1, если слово w1 равно w2. 0 в обратном случае.
int isEqualWord(WordDescriptor w1, WordDescriptor w2) {
  while (w1.begin < w1.end && w2.begin < w2.end) {
    if (*w1.begin != *w2.begin)
      return 0;
    w1.begin++, w2.begin++;
  }
  return 1;
}

char *recSpace(char *readPtr, char *recPtr) {
  while (*readPtr != '\0' && isspace(*readPtr))
    *(recPtr++) = *(readPtr++);
  return recPtr;
}

void getBagOfWords(BagOfWords *bag, char *s) {
  char *begin = s;
  bag->size = 0;
  while (getWord(begin, &bag->words[bag->size])) {
    begin = bag->words[bag->size].end;
    bag->size++;
  }
}

char *writeWordInString(char *s, WordDescriptor w) {
  while (w.begin < w.end)
    *(s++) = *(w.begin++);
  *(s++) = ' ';
  return s;
}

int wordCmp(WordDescriptor w1, WordDescriptor w2) {
  while (*w1.begin == *w2.begin &&
         w1.begin < w1.end - 1 && w2.begin < w2.end - 1)
    w1.begin++, w2.begin++;
  return *w1.begin - *w2.begin;
}
```

```
void outputWord(WordDescriptor word) {
  while (word.begin < word.end) {
    printf("%c", *word.begin);
    word.begin++;
  }
  printf(" ");
}

void test_strlen() {
  char a[] = "Hello world!";
  char b[100] = "Z";
  char c[] = "\tHi\t";

  assert(strlen1(a) == 12);
  assert(strlen2(b) == 1);
  assert(strlen_(c) == 4);
}

void test_finds() {
  char a[] = "  abcd";
  assert(findNonSpace(a) - a == 2);
  char b[] = "abcd  ";
  assert(findSpace(b) - b == 4);
  char c[] = "abcd    ";
  assert(&c[strlen1(c) - 1] - findNonSpaceReverse(&c[7], c) == 4);
  char d[] = "    abcd";
  assert(&d[strlen1(d) - 1] - findSpaceReverse(&d[7], d) == 4);
}
```

Решение задач:

Задание 1: Удалить из строки все пробельные символы.

```
#include "1_digitToStartTransform.h"

// Задача 1

// Удаляет пробельные символы
void removeNonLetters(char *s) {
  char *endSource = getEndOfString(s);
  char *destination = copyIf(s, endSource, s, isgraph);
  *destination = '\0';
}

// Тесты

void test_removeNonLetters_oneWord() {
  char c[] = " abacaba  ";
  removeNonLetters(c);
  ASSERT_STRING("abacaba", c);
}

void test_removeNonLetters_fewWords() {
  char c[] = "ab\ncd\tef  ";
  removeNonLetters(c);
  ASSERT_STRING("abcdef", c);
}

void test_removeNonLetters() {
  test_removeNonLetters_oneWord();
  test_removeNonLetters_fewWords();
}
```

Задание 2: Преобразовать строку, оставляя только один символ в каждой последовательности подряд идущих одинаковых символов.

```c
#include "2_removeAdjacentEqualLetters.h"

// Задача 2

// Преобразовывает строку, оставляя только один символ в каждой
// последовательности подряд идущих одинаковых символов
void removeAdjacentEqualLetters(char *s) {
  char *beginSource = s;
  while (*beginSource) {
    if (*beginSource != *s)
      *(++s) = *beginSource;
    beginSource++;
  }
  *(s + 1) = '\0';
}

// Тесты

void test_removeAdjacentEqualLetters_noRepeatingCharacters() {
  char c[] = "abcd ";
  removeAdjacentEqualLetters(c);
  ASSERT_STRING("abcd ", c);
}

void test_removeAdjacentEqualLetters_manyRepeatedCharacters() {
  char c[] = "aabbccddeeff  ";
  removeAdjacentEqualLetters(c);
  ASSERT_STRING("abcdef ", c);
}

void test_removeAdjacentEqualLetters_multipleRepeatedCharacters() {
  char c[] = "aabccdecef  \n";
  removeAdjacentEqualLetters(c);
  ASSERT_STRING("abcdecef \n", c);
}

void test_removeAdjacentEqualLetters_void() {
  char c[] = "";
  removeAdjacentEqualLetters(c);
  ASSERT_STRING("", c);
}

void test_removeAdjacentEqualLetters() {
  test_removeAdjacentEqualLetters_noRepeatingCharacters();
  test_removeAdjacentEqualLetters_manyRepeatedCharacters();
  test_removeAdjacentEqualLetters_multipleRepeatedCharacters();
  test_removeAdjacentEqualLetters_void();
}
```

Задание 3: Преобразовать строку таким образом, чтобы цифры каждого слова были перенесены в начало слова и изменить порядок следования цифр в слове на обратный, а буквы – в конец слова, без изменения порядка следования.

```c
#include "3_getWord.h"

// Задача 3

// Записывает цифры из слова word в конец слова newWord в обратном порядке
void reverseDigitBack(WordDescriptor word, WordDescriptor *newWord) {
  while (word.begin < word.end) {
    if (isdigit(*word.begin))
      *(--newWord->end) = *word.begin;
    word.begin++;
  }
}

// Записывает буквы из слова word в начало слова newWord в том же порядке
void orderedGraphFront(WordDescriptor word, WordDescriptor *newWord) {
  while (word.begin < word.end) {
    if (isgraph(*word.begin))
      *(newWord->begin++) = *word.begin;
    word.begin++;
  }
}

// Записывает в отдельном слове цифры в конец слова, а цифры в начало.
void wordOrderedDigitReversed(char *c) {
  WordDescriptor word;
  word.begin = c;
  while (getWord(word.begin, &word)) {
    int len = word.end - word.begin;
    char newC[len];
    WordDescriptor newWord = (WordDescriptor) {newC, newC + len};
    reverseDigitBack(word, &newWord);
    orderedGraphFront(word, &newWord);
    memcpy(word.begin, newWord.begin, len);
    word.begin = word.end;
  }
}


// Тесты

void test_wordOrderedDigitReversed_oneWordOnlyGraph() {
  char c[] = "abcd";
  wordOrderedDigitReversed(c);
  ASSERT_STRING("abcd", c);
}

void test_wordOrderedDigitReversed_oneWordOnlyDigit() {
  char c[] = "1234";
  wordOrderedDigitReversed(c);
  ASSERT_STRING("4321", c);
}

void test_wordOrderedDigitReversed_WordOnlyDigitAndWordOnlyDigit() {
  char c[] = "1234 abcd";
  wordOrderedDigitReversed(c);
  ASSERT_STRING("4321 abcd", c);
}

void test_wordOrderedDigitReversed_WordsDigitAndDigit() {
  char c[] = "1234abcd 1a2b3c4d";
  wordOrderedDigitReversed(c);
  ASSERT_STRING("abcd4321 abcd4321", c);
}
```

```c
void test_wordOrderedDigitReversed() {
  test_wordOrderedDigitReversed_oneWordOnlyGraph();
  test_wordOrderedDigitReversed_oneWordOnlyDigit();
  test_wordOrderedDigitReversed_WordOnlyDigitAndWordOnlyDigit();
  test_wordOrderedDigitReversed_WordsDigitAndDigit();
}
```

Задание 4: Преобразовать строку, заменяя каждую цифру соответствующим ей числом пробелов. Имеются такие задачи, при которых размер строки может увеличиться. Мы пока что будем исходить из предположения, что размер итоговой строки не превысит некоторого `MAX_STRING_SIZE` определенного в `string_.h`:

```c
#include "4_replaceNumberDigitsBySpace.h"


// Задание 4

// Преобразовать строку, заменяя каждую цифру соответствующим ей числом
// пробелов
void replaceNumberDigitBySpace(char *c) {
  char newString[MAX_STRING_SIZE];
  char *beginNewString = newString;
  char *begin = c;
  while (*begin != '\0') {
    if (isdigit(*begin))
      for (int i = 0; i < *begin - '0'; i++)
        *(beginNewString++) = ' ';
    else
      *(beginNewString++) = *begin;
    begin++;
  }
  memcpy(c, newString, strlen_(newString));
}

// Тесты

void test_replaceNumberDigitBySpace_oneWord() {
  char c[] = "a1b2c3d4";
  replaceNumberDigitBySpace(c);
  ASSERT_STRING("a b  c   d    ", c);
}

void test_replaceNumberDigitBySpace_anyWord() {
  char c[] = "a1, b1, c1, d1";
  replaceNumberDigitBySpace(c);
  ASSERT_STRING("a , b , c , d ", c);
}

void test_replaceNumberDigitBySpace() {
  test_replaceNumberDigitBySpace_oneWord();
  test_replaceNumberDigitBySpace_anyWord();
}
```

Задание 5: Заменить все вхождения слова `w1` на слово `w2`.

```
#include "5_replace.h"

// Заменяет все вхождения слова w1 на w2
void replace(char *source, char *w1, char *w2) {
  size_t w1Size = strlen_(w1);
  size_t w2Size = strlen_(w2);
  WordDescriptor word1 = {w1, w1 + w1Size};
  WordDescriptor word2 = {w2, w2 + w2Size};

  char *readPtr, *recPtr;
  if (w1Size >= w2Size) {
    readPtr = source;
    recPtr = source;
  } else {
    copy(source, getEndOfString(source), _stringBuffer);
    readPtr = _stringBuffer;
    recPtr = source;
  }

  WordDescriptor word;
  word.begin = readPtr;
  while (getWord(word.begin, &word)) {
    if (isEqualWord(word, word1)) {
      memcpy(recPtr, word2.begin, w2Size);
      recPtr += w2Size;
    } else {
      int len = wordlen(word.begin);
      memcpy(recPtr, word.begin, len);
      recPtr += len;
    }
    recPtr = recSpace(word.end, recPtr);
    word.begin = word.end;
  }
  *recPtr = '\0';
}

// Тесты

void test_replace_noMatches() {
  char c[] = "abc def";
  char w1[] = "ac";
  char w2[] = "ca";
  replace(c, w1, w2);
  ASSERT_STRING("abc def", c);
}

void test_replace_word1LessWord2() {
  char c[] = "happy";
  char w1[] = "happy";
  char w2[] = "sad";
  replace(c, w1, w2);
  ASSERT_STRING("sad", c);
}

void test_replace_word1MoreWord2() {
  char c[] = "sad";
  char w1[] = "sad";
  char w2[] = "happy";
  replace(c, w1, w2);
  ASSERT_STRING("happy", c);
}
```

```
void test_replace_words() {
  char c[] = "abc acb abc bca";
  char w1[] = "abc";
  char w2[] = "a";
  replace(c, w1, w2);
  ASSERT_STRING("a acb a bca", c);
}

void test_replace() {
  test_replace_noMatches();
  test_replace_word1LessWord2();
  test_replace_word1MoreWord2();
  test_replace_words();
}
```

Задание 6: Определить, упорядочены ли лексикографически слова данного предложения

```
#include "6_isOrderedWord.h"

// Задание 6

int areOrderedWords(WordDescriptor w1, WordDescriptor w2) {
  while (w1.begin < w1.end && w2.begin < w2.end) {
    if (*w1.begin < *w2.begin)
      return 1;
    if (*w1.begin > *w2.begin)
      return 0;
    w1.begin++, w2.begin++;
  }
  return 1;
}

int isOrderedString(char *c) {
  WordDescriptor w1;
  if (!getWord(c, &w1))
    return 1;

  WordDescriptor w2;
  while (getWord(w1.end, &w2)) {
    if (!areOrderedWords(w1, w2))
      return 0;
    w1 = w2;
  }
  return 1;
}


// Тесты

void test_isOrderedString_oneWord() {
  char c[] = "ab";
  ASSERT_INT(1, isOrderedString(c));
}

void test_isOrderedString_orderedWords() {
  char c[] = "aa ab ac ca cb";
  ASSERT_INT(1, isOrderedString(c));
}

void test_isOrderedString_unorderedWords() {
  char c[] = "aa ab ac cz cb";
  ASSERT_INT(0, isOrderedString(c));
}
```

```
void test_isOrderedString() {
  test_isOrderedString_oneWord();
  test_isOrderedString_orderedWords();
  test_isOrderedString_unorderedWords();
}
```

Задание 7: Вывести слова данной строки в обратном порядке по одному в строке экрана

```
#include "7_outputReverse.h"

// Выводит слова данной строки в обратном порядке
void outputReverse(char *c) {
  getBagOfWords(&_bag, c);
  for (int i = _bag.size - 1; i > 0; i--)
    outputWord(_bag.words[i]);
}
```

Задание 8: В данной строке соседние слова разделены запятыми. Определить количество слов-палиндромов.

```
#include "8_countPalindroms.h"

int isPalindrome(char *begin, char *end) {
  end--;
  while (end - begin > 0) {
    if (*begin != *end)
      return 0;

    begin++;
    end--;
  }
  return 1;
}

int countOfPalindrome(char *s) {
  WordDescriptor word;
  word.begin = s;
  int count = 0;
  char *endOfString = getEndOfString(s);
  if (endOfString == s)
    return 0;

  do {
    word.end = find(word.begin, endOfString, ',');
    count += isPalindrome(word.begin, word.end);
    word.begin = getBeginOfWord(word.end + 1);
  } while (word.end != endOfString);
  return count;
}

void test_countOfPalindrome_allPalindromes() {
  char c[] = "abba, boob, bib, ABOBA";
  ASSERT_INT(4, countOfPalindrome(c));
}

void test_countOfPalindrome_anyPalindromes() {
  char c[] = "abba, boob, CCCP, ABOBA";
  ASSERT_INT(3, countOfPalindrome(c));
}

void test_countOfPalindrome_noPalindromes() {
  char c[] = "no, one, Palindrome";
  ASSERT_INT(0, countOfPalindrome(c));
}
```

Задание 9: Даны две строки. Получить строку, в которой чередуются слова первой и второй строки. Если в одной из строк число слов больше, чем в другой, то оставшиеся слова этой строки должны быть дописаны в строку-результат. В качестве разделителя между словами используйте пробел.

```
#include "9_wordAlternation.h"

int max(int a, int b) {
  return a > b ? a : b;
}

char *wordAlternation(char *s1, char *s2, char *res) {
  getBagOfWords(&_bag, s1);
  getBagOfWords(&_bag2, s2);
  char *beginPtr = res;

  for (int i = 0; i < max(_bag.size, _bag2.size); i++) {
    if (i < _bag.size)
      beginPtr = writeWordInString(beginPtr, _bag.words[i]);
    if (i < _bag2.size)
      beginPtr = writeWordInString(beginPtr, _bag2.words[i]);
  }

  *(--beginPtr) = '\0';
  return res;
}

void test_wordAlternation_S1EqualS2() {
  char s1[] = "ab ef";
  char s2[] = "cd gh";
  char res[MAX_STRING_SIZE];
  ASSERT_STRING("ab cd ef gh", wordAlternation(s1, s2, res));
}

void test_wordAlternation_S1lessS2() {
  char s1[] = "ab ef";
  char s2[] = "cd gh 12";
  char res[MAX_STRING_SIZE];
  ASSERT_STRING("ab cd ef gh 12", wordAlternation(s1, s2, res));
}

void test_wordAlternation_S1moreS2() {
  char s1[] = "ab ef 34 56";
  char s2[] = "cd gh 12";
  char res[MAX_STRING_SIZE];
  ASSERT_STRING("ab cd ef gh 12 34 56", wordAlternation(s1, s2, res));
}

void test_wordAlternation() {
  test_wordAlternation_S1EqualS2();
  test_wordAlternation_S1lessS2();
  test_wordAlternation_S1moreS2();
}
```

Задание 10: Преобразовать строку, изменив порядок следования слов в строке на обратный.

```
#include "10_reverseWordInString.h"

char *reverseWriteWord(char *rBegin, WordDescriptor w) {
  while (w.begin < w.end)
    *(--rBegin) = *(--w.end);
  return rBegin;
}
```

```
void reverseWordInString(char *s) {
  char *end = getEndOfString(s);
  copy(s, end + 1, _stringBuffer);
  WordDescriptor word;
  word.begin = _stringBuffer;


  while (getWord(word.begin, &word)) {
    end = reverseWriteWord(end, word);
    if (end > s)
      *(--end) = ' ';
    word.begin = word.end;
  }
}

void test_reverseWordInString_oneWord() {
  char c[] = "abc";
  reverseWordInString(c);
  ASSERT_STRING("abc", c);
}

void test_reverseWordInString_moreWord() {
  char c[] = "abc def";
  reverseWordInString(c);
  ASSERT_STRING("def abc", c);
}

void test_reverseWordInString() {
  test_reverseWordInString_oneWord();
  test_reverseWordInString_moreWord();
}
```

Задание 11: Вывести слово данной строки, предшествующее первому из слов, содержащих букву "a"

```
#include "11_printWordBeforeFirstWordWithA.h"

int wordHasA(WordDescriptor w) {
  while (w.begin < w.end) {
    if (*w.begin == 'a' || *w.begin == 'A')
      return 1;
    w.begin++;
  }
  return 0;
}

WordBeforeFirstWordWithAReturnCode printWordBeforeFirstWithA(char *s) {
  getBagOfWords(&_bag, s);
  if (_bag.size == 0) {
    return EMPTY_STRING;
  } else if (_bag.size == 1) {
    return FIRST_WORD_WITH_A;
  }

  for (int i = 0; i < _bag.size; i++) {
    if (wordHasA(_bag.words[i])) {
      if (i == 0)
        return FIRST_WORD_WITH_A;
      outputWord(_bag.words[i - 1]);
      return WORD_FOUND;
    }
  }
  return NOT_FOUND_A_WORD_WITH_A;
}
```

```
WordBeforeFirstWordWithAReturnCode getWordBeforeFirstWordWithA(char *s,
                                                               char **beginWord,
                                                               char **endWord) {
  getBagOfWords(&_bag, s);
  if (_bag.size == 0) {
    return EMPTY_STRING;
  } else if (_bag.size == 1) {
    return FIRST_WORD_WITH_A;
  }

  for (int i = 0; i < _bag.size; i++) {
    if (wordHasA(_bag.words[i])) {
      if (i == 0)
        return FIRST_WORD_WITH_A;
      *beginWord = _bag.words[i - 1].begin;
      *endWord = _bag.words[i - 1].end;
      return WORD_FOUND;
    }
  }
  return NOT_FOUND_A_WORD_WITH_A;
}

void testAll_getWordBeforeFirstWordWithA() {
  char *beginWord, *endWord;

  char s1[] = "";
  assert (
    getWordBeforeFirstWordWithA(s1, &beginWord, &endWord)
    == EMPTY_STRING
  );

  char s2[] = " ABC";
  assert (
    getWordBeforeFirstWordWithA(s2, &beginWord, &endWord)
    == FIRST_WORD_WITH_A
  );

  char s3[] = "BC A";
  assert (
    getWordBeforeFirstWordWithA(s3, &beginWord, &endWord)
    == WORD_FOUND
  );
  char got[MAX_WORD_SIZE];
  copy(beginWord, endWord, got);
  got[endWord - beginWord] = '\0';
  ASSERT_STRING ("BC", got);

  char s4[] = "B Q WE YR OW IUWR ";
  assert (getWordBeforeFirstWordWithA(s4, &beginWord, &endWord) ==
          NOT_FOUND_A_WORD_WITH_A);
}
```

Задание 12: Даны две строки. Определить последнее из слов первой строки, которое есть во второй строке.

```
#include "12_lastCommonWord.h"


WordDescriptor lastCommonWord(char *s1, char *s2) {
  getBagOfWords(&_bag, s1);
  getBagOfWords(&_bag2, s2);

  for (int i = _bag.size - 1; i >= 0; i--)
    for (int j = 0; j < _bag2.size; j++)
      if (wordCmp(_bag.words[i], _bag2.words[j]) == 0)
        return _bag.words[i];
  return (WordDescriptor){s1, s1};
}

void wordDescriptorToString(WordDescriptor word, char *destination) {
  destination = copy(word.begin, word.end, destination);
  *destination = '\0';
}

void test_lastCommonWord_noWord() {
  char s1[] = "aa ab ac";
  char s2[] = "cc cb ca";
  WordDescriptor word = lastCommonWord(s1, s2);
  char res[MAX_WORD_SIZE];
  wordDescriptorToString(word, res);
  ASSERT_STRING("", res);
}

void test_lastCommonWord_oneWord() {
  char s1[] = "aa";
  char s2[] = "aa";
  WordDescriptor word = lastCommonWord(s1, s2);
  char res[MAX_WORD_SIZE];
  wordDescriptorToString(word, res);
  ASSERT_STRING("aa", res);
}

void test_lastCommonWord_anyWord() {
  char s1[] = "aa bb cc";
  char s2[] = "aa bb cc";
  WordDescriptor word = lastCommonWord(s1, s2);
  char res[MAX_WORD_SIZE];
  wordDescriptorToString(word, res);
  ASSERT_STRING("cc", res);
}

void test_lastCommonWord_sadadasd() {
  char s1[] = "aa bb cc";
  char s2[] = "asd asd asd";
  WordDescriptor word = lastCommonWord(s1, s2);
  char res[MAX_WORD_SIZE];
  wordDescriptorToString(word, res);
  ASSERT_STRING("", res);
}

void test_lastCommonWord() {
  test_lastCommonWord_noWord();
  test_lastCommonWord_oneWord();
  test_lastCommonWord_anyWord();
  test_lastCommonWord_sadadasd();
}
```

Задание 13: Определить, есть ли в данной строке одинаковые слова.

```c
#include "13_haveSameWord.h"

int haveSameWord(char *s1, char *s2) {
  WordDescriptor w = lastCommonWords(s1, s2);
  if (w.end == w.begin)
    return 0;
  return 1;
}

void test_haveSameWord_noWord() {
  char s1[] = "aa ab ac";
  char s2[] = "cc cb ca";
  assert(haveSameWord(s1, s2) == 0);
}

void test_haveSameWord_oneWord() {
  char s1[] = "aa";
  char s2[] = "aa";
  assert(haveSameWord(s1, s2) == 1);
}

void test_haveSameWord_anyWord() {
  char s1[] = "aa bb cc";
  char s2[] = "aa bb cc";
  assert(haveSameWord(s1, s2) == 1);
}

void test_haveSameWord_sadadasd() {
  char s1[] = "aa bb cc";
  char s2[] = "asd asd asd";
  assert(haveSameWord(s1, s2) == 0);
}

void test_haveSameWord() {
  test_haveSameWord_noWord();
  test_haveSameWord_oneWord();
  test_haveSameWord_anyWord();
  test_haveSameWord_sadadasd();
}
```

Задание 15: Получить строку из слов данной строки, которые отличны от последнего слова.

```c
#include "15_differentFromTheLastWord.h"

void differentFromTheLastWord(char *s) {
  copy(s, getEndOfString(s) + 1, _stringBuffer);
  getBagOfWords(&_bag, _stringBuffer);

  for (int i = 0; i < _bag.size - 1; i++) {
    if (wordCmp(_bag.words[i], _bag.words[_bag.size - 1]) != 0)
      s = writeWordInString(s, _bag.words[i]);
  }
  *s = '\0';
}

void test_differentFromTheLastWord_one() {
  char c[] = "aa";
  differentFromTheLastWord(c);
  ASSERT_STRING("", c);
}
```

```
void test_differentFromTheLastWord_any() {
  char c[] = "aa bb cc aa";
  differentFromTheLastWord(c);
  ASSERT_STRING("bb cc", c);
}

void test_differentFromTheLastWord() {
  test_differentFromTheLastWord_one();
  test_differentFromTheLastWord_any();
}
```

Задание 16: Даны две строки. Определить последнее из слов первой строки, которое есть во второй строке.

```
#include "16_lastCommonWords.h"


WordDescriptor lastCommonWords(char *s1, char *s2) {
  getBagOfWords(&_bag, s1);
  getBagOfWords(&_bag2, s2);

  for (int i = _bag.size - 1; i >= 0; i--)
    for (int j = 0; j < _bag2.size; j++)
      if (wordCmp(_bag.words[i], _bag2.words[j]) == 0)
        return _bag.words[i];
  return (WordDescriptor){s1, s1};
}

void wordsDescriptorToString(WordDescriptor word, char *destination) {
  destination = copy(word.begin, word.end, destination);
  *destination = '\0';
}

void test_lastCommonWords_noWord() {
  char s1[] = "aa ab ac";
  char s2[] = "cc cb ca";
  WordDescriptor word = lastCommonWords(s1, s2);
  char res[MAX_WORD_SIZE];
  wordDescriptorToString(word, res);
  ASSERT_STRING("", res);
}

void test_lastCommonWords_oneWord() {
  char s1[] = "aa";
  char s2[] = "aa";
  WordDescriptor word = lastCommonWords(s1, s2);
  char res[MAX_WORD_SIZE];
  wordDescriptorToString(word, res);
  ASSERT_STRING("aa", res);
}

void test_lastCommonWords_anyWord() {
  char s1[] = "aa bb cc";
  char s2[] = "aa bb cc";
  WordDescriptor word = lastCommonWords(s1, s2);
  char res[MAX_WORD_SIZE];
  wordDescriptorToString(word, res);
  ASSERT_STRING("cc", res);
}
```

```
void test_lastCommonWords_sadadasd() {
  char s1[] = "aa bb cc";
  char s2[] = "asd asd asd";
  WordDescriptor word = lastCommonWords(s1, s2);
  char res[MAX_WORD_SIZE];
  wordDescriptorToString(word, res);
  ASSERT_STRING("", res);
}

void test_lastCommonWords() {
  test_lastCommonWords_noWord();
  test_lastCommonWords_oneWord();
  test_lastCommonWords_anyWord();
  test_lastCommonWords_sadadasd();
}
```

Задание 17: Удалить из строки слова, совпадающие с последним словом.

```
#include "15_differentFromTheLastWord.h"

void differentFromTheLastWord(char *s) {
  copy(s, getEndOfString(s) + 1, _stringBuffer);
  getBagOfWords(&_bag, _stringBuffer);

  for (int i = 0; i < _bag.size - 1; i++) {
    if (wordCmp(_bag.words[i], _bag.words[_bag.size - 1]) != 0)
      s = writeWordInString(s, _bag.words[i]);
  }
  *s = '\0';
}

void test_differentFromTheLastWord_one() {
  char c[] = "aa";
  differentFromTheLastWord(c);
  ASSERT_STRING("", c);
}

void test_differentFromTheLastWord_any() {
  char c[] = "aa bb cc aa";
  differentFromTheLastWord(c);
  ASSERT_STRING("bb cc", c);
}

void test_differentFromTheLastWord() {
  test_differentFromTheLastWord_one();
  test_differentFromTheLastWord_any();
}
```