Федеральное государственное бюджетное образовательное учреждение

высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

Факультет ИВТ

Кафедра вычислительных систем

**Курсовая работа**

на тему «ОБРАБОТКА ПОСЛЕДОВАТЕЛЬНОЙ ИНФОРМАЦИИ»

Вариант 1.4 «Поиск палиндромов в тексте»

Выполнил:студент гр. ИВ-122Гердележов Д.Д.

Проверил:Старший преподаватель Кафедры ВС

Фульман В.О.

Новосибирск, 2022

Оглавление

[Тема курсовой работы 3](#_Toc104907989)

[Задание на курсовую работу 3](#_Toc104907990)

[Анализ задачи 4](#_Toc104907991)

[Тестовые данные 9](#_Toc104907992)

[Листинг программы 11](#_Toc104907993)

# Тема курсовой работы

ОБРАБОТКА ПОСЛЕДОВАТЕЛЬНОЙ ИНФОРМАЦИИ

Поиск палиндромов в тексте

# Задание на курсовую работу

* Разработать программу palindrom, выполняющую поиск всех палиндромов в заданном тексте. Команда palindrom принимает в качестве аргумента командной строки имя файла, содержащего текст на русском языке. Все найденные палиндромы распечатываются на экране.

Критерии оценки

* Оценка «удовлетворительно»: реализована проверка того, что весь текст входного файла целиком является палиндромом. Не предусмотрено динамическое выделение памяти под входные данные.
* Оценка «хорошо»: реализована предварительная обработка текста: из каждого предложения удаляются все знаки препинания. После этого осуществляется проверка каждого предложения на выполнение свойства палиндрома. Обязательно динамическое выделение памяти под входные данные.
* Оценка «отлично»: реализована предварительная обработка текста: из текста удаляются все пробелы и знаки препинания так, чтобы получилось одно большое слово. Для поиска подпалиндромов используется алгоритм, основанный на применении динамического программирования (http://comp-science.narod.ru/WebPage/p6.html). Обязательно динамическое выделение памяти под входные данные. Указания к выполнению задания Палиндромом называются слово (потоп, шалаш) или текст (а роза упала на лапу Азора), читающиеся одинаково в обоих направлениях.

Запуск программы должен производиться со следующими аргументами командной строки:



Программа выполняет поиск всех палиндромов в файле text.txt и распечатывает результат работы на экране.

# Анализ задачи

1. На вход программе подается text.txt, в котором находится текст на русском языке, программа должна найти палиндромы в этом тексте и вывести их на экран.

Работа будет вестись с типом данных wchar\_t.

wchar\_t — это тип данных, использующийся для представления широких символов (национальные языки, некоторые символы). В [GNU/Linux](https://ru.wikipedia.org/wiki/GNU/Linux) тип wchar\_t имеет размер 32 бита.

Function Declaration ->int len\_file(FILE\* text)wchar\_t\* init\_wchar(int size)Function Declaration ->int len\_str(wchar\_t\* sentence)Function Declaration ->void wchar\_strcpy(wchar\_t\* sentence, wchar\_t\* sec\_sentence)Function Declaration ->void wchar\_tolower(wchar\_t\* sentence)wchar\_t\* wchar\_strcat(wchar\_t\* dest, const wchar\_t\* src)Function Declaration ->void wchar\_delsymb(wchar\_t\* sentence, int size, char ch)Function Declaration ->void clear(wchar\_t\* sentence, int size)Function Declaration ->int\* sent\_sep(wchar\_t\* sentence, int\* parts)Function Declaration ->int cat\_words(wchar\_t\* sentence, int start)Function Declaration ->void copy\_word(wchar\_t\* sentence, wchar\_t\* word, int start, int end)Function Declaration ->int satisfactorily(wchar\_t\* sentence, int size)Function Declaration ->int find\_palinroms(wchar\_t\* sentence, int size, wchar\_t\* palindroms)Function Declaration ->int good(wchar\_t\* sentence, int\* parts, int size,wchar\_t\* palindroms)

* Поиск длины файла:

Function int len\_file(FILE \*text)

int size = 0

While ((symb = fgetwc(text)) != EOF) do

size++;

end while

return size

end function

* Выделение памяти для wchar\_t:

Function wchar\_t\* init\_wchar(int size)

    wchar\_t\* point = malloc(sizeof(wchar\_t) \* (size));

    if (!point) then

        return (0);

    else then

        return point;

    end else

end function

* Поиск длины строки:

Function int len\_str(wchar\_t\* sentence)

int i = 0;

while (sentence[i] != '\0') do

i++;

end while

return (i + 1);

end function

* Копирование строки:

Function void wchar\_strcpy (wchar\_t\* sentence, wchar\_t\* sec\_sentence)

int i;

for (i = 0; sentence[i] != '\0'; i++) do

sec\_sentence[i] = sentence[i];

end for

sec\_sentence[i] = '\0';

end function

* Приведение к нижнему регистру:

Function void wchar\_tolower(wchar\_t\* sentence)

int i = 0;

while (sentence[i] != '\0') do:

if (sentence[i] >= 'A' && sentence[i] <= 'Z') then

sentence[i] += 32;

end if

if (sentence[i] >= 1040 && sentence[i] <= 1071) then

sentence[i] += 32;

end if // 1040 - 'А', 1071 - 'Я' (UTF-8)

i++;

end while

end function

* Объединение строк:

Function wchar\_t\* wchar\_strcat(wchar\_t\* dest, const wchar\_t\* src)

wchar\_t\* tmp = dest;

while (\*dest)

dest++;

while ((\*dest++ = \*src++) != '\0')

;

return tmp;

end function

* Удаление символа из строки:

Function void wchar\_delsymb(wchar\_t\* sentence, int size, char ch)

int i, j, k = 0;

while (k < size) do

i = 0;

j = 0;

while (i < size) do

if (sentence[i] == ch) then

sentence[i] = sentence[i + 1];

for (j = i; j < size; j++) do

sentence[j] = sentence[j + 1];

end for

end if

i++;

end while

k++;

end while

end function

* Очищает предложение от знаков препинания:

function void clear(wchar\_t\* sentence, int size)

    char prip[] = {'"', ',', ';', ':', '!', '?', '(', ')', '-'};

    int i;

    for (i = 0; i < 9; i++) do

        wchar\_delsymb(sentence, size, prip[i]);

    end for

end function

* Возвращает указатель на массив, первым элементом которого является число предложений, второй и последующие – номер последнего символа предложения:

function int\* sent\_sep(wchar\_t\* sentence, int\* parts)

int i = 0, count = 1;

for (i = 0; sentence[i] != '\0'; i++) do

if (sentence[i] == '.' || sentence[i] == '?' || sentence[i] == '!'

|| sentence[i] == '\0') then

count++;

end if

end for

parts = malloc(sizeof(int) \* (count + 1));

parts[0] = count;

count = 0;

int num = 0;

for (i = 0; sentence[i] != '\0'; i++) do

if (sentence[i] == '.' || sentence[i] == '?' || sentence[i] == '!'

|| sentence[i] == '\0') then

if (count > 1) then

parts[count + 1] = i - count + 1;

end if

else do

parts[count + 1] = i;

end else

count++;

num += count;

end if

parts[count + 1] = i - count + 1;

end for

return parts;

end function

* Возвращает номер последнего символа слова (на вход принимает номер первого символа):

Function int cat\_words(wchar\_t\* sentence, int start)

int i = start;

while (sentence[i] != '\0' && sentence[i] != ',' && sentence[i] != '!'

&& sentence[i] != '.' && sentence[i] != ' ' && sentence[i] != '?') do

i++;

end while

return i;

end function

* Копирует отрывок одного предложения в другое:

function void copy\_word(wchar\_t\* sentence, wchar\_t\* word, int start, int end)

int i;

for (i = 0; i < (end - start); i++) do

word[i] = sentence[i + start];

end for

word[i] = '\0';

end function

* Проверяет, является ли весь текст входного файла целиком палиндромом:

function int satisfactorily(wchar\_t\* sentence, int size)

wchar\_tolower(sentence);

    // wchar\_delsymb(sentence, size, ' ');

    size = len\_str(sentence);

    int flag = 1;

    int i;

for (i = 0; i < (size / 2); i++) do

if (sentence[i] != sentence[size - i - 2]) then

flag = 0;

break;

end if

end for

if (i == 1) then

flag = 0;

end if

return flag;

end function

* Находит в предложении слова палиндромы и печатает их на экран:

function int find\_palinroms(wchar\_t\* sentence, int size, wchar\_t\* palindroms)

    wchar\_t\* space = init\_wchar(1);

    \*space = ' ';

    wchar\_t\* buf\_sentence = init\_wchar(size);

    wchar\_strcpy(sentence, buf\_sentence);

    int start = 0, end, flag = 0;

    clear(buf\_sentence, size);

    do

        end = cat\_words(buf\_sentence, start);

        int size = end - start + 1;

        wchar\_t\* word;

        word = init\_wchar(size);

        copy\_word(buf\_sentence, word, start, end);

        if (satisfactorily(word, (end - start)) == 1) then

            wchar\_strcat(palindroms, space);

            wchar\_strcat(palindroms, word);

            flag = 1;

        end if

        start = end + 1;

    while (buf\_sentence[end] != '\0');

    return flag;

end function

* Находит в ведённом тексте слова палиндромы и печатает их на экран:

function int good(wchar\_t\* sentence, int\* parts, int size, wchar\_t\* palindroms)

    int flag = 0;

    wchar\_t\* new = L"\n     ";

    wchar\_t\* buf\_part;

    clear(sentence, size);

    parts = sent\_sep(sentence, parts);

    wchar\_delsymb(sentence, size, '.');

    int i = 0;

    int j = 0;

    for (i = 1; i < parts[0] + 1; i++) do

        if (j != parts[parts[0]]) then

            buf\_part = init\_wchar((parts[i] - j + 1));

            copy\_word(sentence, buf\_part, j, parts[i]);

            wchar\_delsymb(buf\_part, parts[i] - j + 1, ' ');

            if (satisfactorily(buf\_part, parts[i]) == 1) then

                if (flag == 1) then

                    wchar\_strcat(palindroms, new);

                end if

                wchar\_strcat(palindroms, buf\_part);

                flag = 1;

            end if

            j = parts[i];

        end if

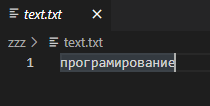
    end for

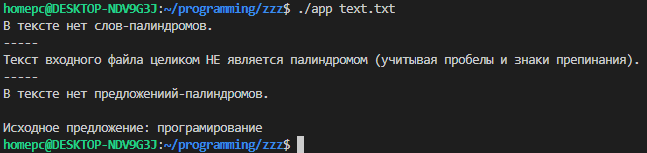
    return flag;

end function

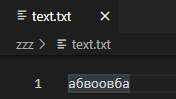
# Тестовые данные

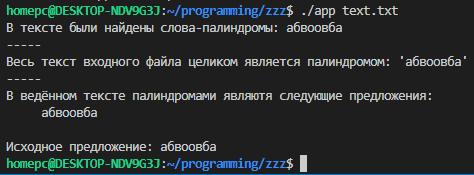
Для начала попробуем ввести одно слово, которое не является палиндромом:



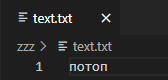


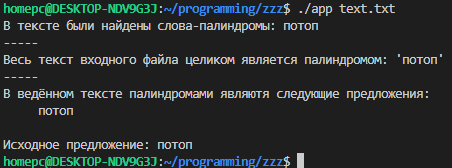
Теперь введём одно слово, которое является палиндромом и имеет четное количество букв:



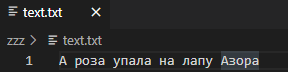


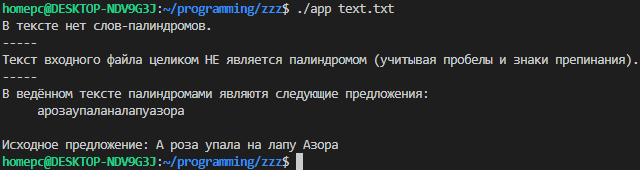
Теперь введём одно слово, которое является палиндромом и имеет нечетное количество букв:



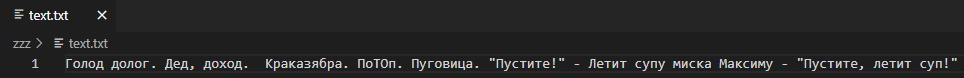


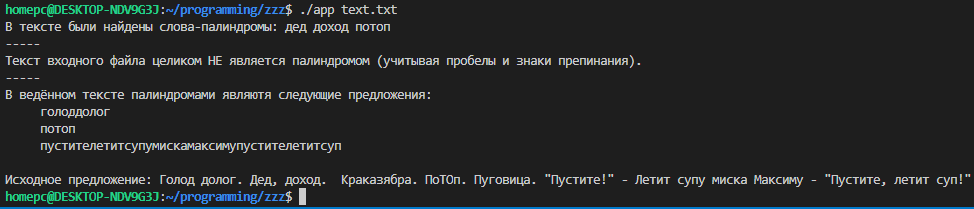
Теперь введём предложение-палиндром без знаков препинания





Теперь ведём несколько предложений среди которых есть предложения-палиндромы и слова-палиндромы, слова, которые не являются палиндромами.





# Листинг программы

main.c

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | #include "palindrom.h"  #include <locale.h>  #include <stdio.h>  #include <stdlib.h>  #include <wchar.h>  **int** **main**(**int** argc, **char**\*\* argv)  {  setlocale(LC\_ALL, "");  **FILE**\* text;  text = fopen(argv[**1**], "r");  **int** flag = **0**;  **int** size = **0**;  **wchar\_t**\* sentence;  **wchar\_t**\* buf\_sentence;  **wchar\_t**\* palindroms;  **int**\* parts = NULL;  size = len\_file(text);  size++;  fseek(text, **0**, SEEK\_SET);  sentence = init\_wchar(size);  buf\_sentence = init\_wchar(size);  palindroms = init\_wchar(size + **99**);  fgetws(sentence, size, text);  wchar\_strcpy(sentence, buf\_sentence);  //----- главное  flag = find\_palinroms(buf\_sentence, size, palindroms);  **if** (flag == **1**) {  wprintf(L"В тексте были найдены слова-палиндромы:%S**\n**", palindroms);  } **else** {  wprintf(L"В тексте нет слов-палиндромов.**\n**");  }  wprintf(L"-----**\n**");  //----- удовл.  flag = satisfactorily(buf\_sentence, size);  **if** (flag == **1**) {  wprintf(L"Весь текст входного файла целиком является палиндромом: ");  wprintf(L"'%S'**\n**-----**\n**", sentence);  } **else** {  wprintf(L"Текст входного файла целиком НЕ является палиндромом "  L"(учитывая пробелы и знаки препинания).**\n**-----**\n**");  }  //----- хорошо  palindroms = init\_wchar(size);  flag = good(buf\_sentence, parts, size, palindroms);  **if** (flag == **1**) {  wprintf(L"В ведённом тексте палиндромами являютя следующие "  L"предложения: **\n** %S**\n**",  palindroms);  } **else** {  wprintf(L"В тексте нет предложениий-палиндромов.**\n**");  }  wprintf(L"**\n**Исходное предложение: %S**\n**", sentence);  fclose(text);  free(sentence);  free(buf\_sentence);  free(parts);  **return** **0**;  } |

palindrom.c

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223 | #include <locale.h>  #include <stdio.h>  #include <stdlib.h>  #include <wchar.h>  **int** **len\_file**(**FILE**\* text)  {  **int** size = **0**;  **wchar\_t** symb;  **while** ((symb = fgetwc(text)) != EOF) {  size++;  }  **return** size;  }  **wchar\_t**\* **init\_wchar**(**int** size)  {  **wchar\_t**\* point = malloc(**sizeof**(**wchar\_t**) \* (size));  **if** (!point) {  **return** (**0**);  } **else** {  **return** point;  }  }  **int** **len\_str**(**wchar\_t**\* sentence)  {  **int** i = **0**;  **while** (sentence[i] != '\0') {  i++;  }  **return** (i + **1**);  }  **void** **wchar\_strcpy**(**wchar\_t**\* sentence, **wchar\_t**\* sec\_sentence)  {  **int** i;  **for** (i = **0**; sentence[i] != '\0'; i++) {  sec\_sentence[i] = sentence[i];  }  sec\_sentence[i] = '\0';  }  **void** **wchar\_tolower**(**wchar\_t**\* sentence)  {  **int** i = **0**;  **while** (sentence[i] != '\0') {  **if** (sentence[i] >= 'A' && sentence[i] <= 'Z') {  sentence[i] += **32**;  }  **if** (sentence[i] >= **1040** && sentence[i] <= **1071**) {  sentence[i] += **32**;  } // 1040 - 'А', 1071 - 'Я' (UTF-8)  i++;  }  }  **wchar\_t**\* **wchar\_strcat**(**wchar\_t**\* dest, **const** **wchar\_t**\* src)  {  **wchar\_t**\* tmp = dest;  **while** (\*dest)  dest++;  **while** ((\*dest++ = \*src++) != '\0')  ;  **return** tmp;  }  **void** **wchar\_delsymb**(**wchar\_t**\* sentence, **int** size, **char** ch)  {  **int** i, j, k = **0**;  **while** (k < size) {  i = **0**;  j = **0**;  **while** (i < size) {  **if** (sentence[i] == ch) {  sentence[i] = sentence[i + **1**];  **for** (j = i; j < size; j++) {  sentence[j] = sentence[j + **1**];  }  }  i++;  }  k++;  }  }  **void** **clear**(**wchar\_t**\* sentence, **int** size)  {  **char** prip[] = {'"', ',', ';', ':', '!', '?', '(', ')', '-'};  **int** i;  **for** (i = **0**; i < **9**; i++) {  wchar\_delsymb(sentence, size, prip[i]);  }  }  **int**\* **sent\_sep**(**wchar\_t**\* sentence, **int**\* parts)  {  **int** i = **0**, count = **1**;  **for** (i = **0**; sentence[i] != '\0'; i++) {  **if** (sentence[i] == '.' || sentence[i] == '?' || sentence[i] == '!'  || sentence[i] == '\0') {  count++;  }  }  parts = malloc(**sizeof**(**int**) \* (count + **1**));  parts[**0**] = count;  count = **0**;  **int** num = **0**;  **for** (i = **0**; sentence[i] != '\0'; i++) {  **if** (sentence[i] == '.' || sentence[i] == '?' || sentence[i] == '!'  || sentence[i] == '\0') {  **if** (count > **1**) {  parts[count + **1**] = i - count + **1**;  } **else** {  parts[count + **1**] = i;  }  count++;  num += count;  }  parts[count + **1**] = i - count + **1**;  }  **return** parts;  }  **int** **cat\_words**(**wchar\_t**\* sentence, **int** start)  {  **int** i = start;  **while** (sentence[i] != '\0' && sentence[i] != ',' && sentence[i] != '!'  && sentence[i] != '.' && sentence[i] != ' ' && sentence[i] != '?') {  i++;  }  **return** i;  }  **void** **copy\_word**(**wchar\_t**\* sentence, **wchar\_t**\* word, **int** start, **int** end)  {  **int** i;  **for** (i = **0**; i < (end - start); i++) {  word[i] = sentence[i + start];  }  word[i] = '\0';  }  **int** **satisfactorily**(**wchar\_t**\* sentence, **int** size)  {  wchar\_tolower(sentence);  //wchar\_delsymb(sentence, size, ' ');  size = len\_str(sentence);  **int** flag = **1**;  **int** i;  **for** (i = **0**; i < (size / **2**); i++) {  **if** (sentence[i] != sentence[size - i - **2**]) {  flag = **0**;  **break**;  }  }  **if** (i <= **1**) {  flag = **0**;  }  **return** flag;  }  **int** **find\_palinroms**(**wchar\_t**\* sentence, **int** size, **wchar\_t**\* palindroms)  {  **wchar\_t**\* space = init\_wchar(**1**);  \*space = ' ';  **wchar\_t**\* buf\_sentence = init\_wchar(size);  wchar\_strcpy(sentence, buf\_sentence);  **int** start = **0**, end, flag = **0**;  clear(buf\_sentence, size);  **do** {  end = cat\_words(buf\_sentence, start);  **int** size = end - start + **1**;  **wchar\_t**\* word;  word = init\_wchar(size);  copy\_word(buf\_sentence, word, start, end);  **if** (satisfactorily(word, (end - start)) == **1**) {  wchar\_strcat(palindroms, space);  wchar\_strcat(palindroms, word);  flag = **1**;  }  start = end + **1**;  } **while** (buf\_sentence[end] != '\0');  **return** flag;  }  **int** **good**(**wchar\_t**\* sentence, **int**\* parts, **int** size, **wchar\_t**\* palindroms)  {  **int** flag = **0**;  **wchar\_t**\* new = L"**\n** ";  **wchar\_t**\* buf\_part;  clear(sentence, size);  parts = sent\_sep(sentence, parts);  wchar\_delsymb(sentence, size, '.');  **int** i = **0**;  **int** j = **0**;  **for** (i = **1**; i < parts[**0**] + **1**; i++) {  **if** (j != parts[parts[**0**]]) {  buf\_part = init\_wchar((parts[i] - j + **1**));  copy\_word(sentence, buf\_part, j, parts[i]);  wchar\_delsymb(buf\_part, parts[i] - j + **1**, ' ');  **if** (satisfactorily(buf\_part, parts[i]) == **1**) {  **if** (flag == **1**) {  wchar\_strcat(palindroms, new);  }  wchar\_strcat(palindroms, buf\_part);  flag = **1**;  }  j = parts[i];  }  }  **return** flag;  } |

palindrom.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #pragma once  #include <stddef.h>  #include <stdio.h>  **int** **len\_file**(**FILE**\* text);  **wchar\_t**\* **init\_wchar**(**int** size);  **int** **len\_str**(**wchar\_t**\* sentence);  **void** **wchar\_strcpy**(**wchar\_t**\* sentence, **wchar\_t**\* sec\_sentence);  **void** **wchar\_tolower**(**wchar\_t**\* sentence);  **wchar\_t**\* **wchar\_strcat**(**wchar\_t**\* dest, **const** **wchar\_t**\* src);  **void** **wchar\_delsymb**(**wchar\_t**\* sentence, **int** size, **char** ch);  **void** **clear**(**wchar\_t**\* sentence, **int** size);  **int**\* **sent\_sep**(**wchar\_t**\* sentence, **int**\* parts);  **int** **cat\_words**(**wchar\_t**\* sentence, **int** start);  **void** **copy\_word**(**wchar\_t**\* sentence, **wchar\_t**\* word, **int** start, **int** end);  **int** **satisfactorily**(**wchar\_t**\* sentence, **int** size);  **int** **find\_palinroms**(**wchar\_t**\* sentence, **int** size, **wchar\_t**\* palindroms);  **int** **good**(**wchar\_t**\* sentence, **int**\* parts, **int** size,**wchar\_t**\* palindroms); |