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 1.5 «Разработка простейшего переводчика»

Выполнил:
студент гр. ИВ-222 Очнев А. Д.

Проверил:
старший преподаватель Кафедры ВС
Фульман В.О.

Новосибирск, 2023

Оглавление

Тема курсовой работы

ОБРАБОТКА ПОСЛЕДОВАТЕЛЬНОЙ ИНФОРМАЦИИ

Задание на курсовую работу

Задание

Разработать программу ***translate***, выполняющую перевод текста с помощью словаря. Команда ***translate*** принимает на вход 3 файла. Первый содержит исходный текст, который необходимо перевести. Второй файл имеет вид простейшего словаря, где каждому слову на исходном языке соответствует слово на целевом. Третий файл необходимо создать и записать в него результат работы переводчика. Формат исходного текста должен быть сохранен.

Критерии оценки

- **Оценка «удовлетворительно»**: не реализована поддержка файла-словаря, словарь задается статически в программе. Не предусмотрено динамическое выделение памяти под входные данные.
- **Оценка «хорошо»**: программа реализована в полном соответствии с заданием. Обязательно динамическое выделение памяти под входные данные.
- **Оценка «отлично»** не предусмотрена, может быть предложен свой вариант усложнения.

Указания к выполнению задания

Запуск программы должен производиться со следующими аргументами командной строки:

```
$ translate text_rus.txt dictionary.txt text_eng.txt
```

Программа должна перевести текст в файле *text_rus.txt* с помощью словаря *dictionary.txt*, и записать результат в файл *text_eng.txt*. Примерное содержимое файлов и результат работы программы представлен на рисунке 2.

Text_rus.txt:	*Dictionary.txt:*	*Text_eng.txt:*
Тигр, Тигр, жгучий страх. Ты горишь в ночных лесах. Чей бессмертный взор, любя, Создал страшного тебя?	Тигр – Tyger Страх – Fear Ты – You Взор – Eye	Tyger, Tyger, жгучий fear. You горишь в ночных лесах. Чей бессмертный Eye, любя, Создал страшного тебя?

Рисунок 2. Пример работы программы

Форматирование текста в файле text_rus.txt должно быть сохранено и в итоговом файле text_eng.txt, т.е. сохранены все сдвиги и переносы по тексту. Задание не предусматривает поиск однокоренных слов, поэтому замена слова происходит только по полному соответствию.

Анализ задачи

В реализации переводчика главной задачей является замена исходного слова на его перевод в строке символов, ниже представлен алгоритм замены слова на его перевод в виде псевдокода.

```
1  Функция заменить_слово(вывод, слово, перевод)
2    Для i = 0 до длины вывода - 1
3      Если в нижнем регистре вывод[i] == в нижнем регистре слово[0]
4      и сравнить вывод[от i до длины слова] == 0
5      и не является буквой на русском вывод[i + длина слова]
6      и не является буквой на английском вывод[i + длина слова]
7        Если вывод [i] == в нижнем регистре слово[0]
8          вывод[i] = в нижнем регистре перевод[0]
9        Иначе Если вывод[i] == в верхнем регистре слово[0]
10         вывод[i] = в верхнем регистре перевод[0]
11       Иначе
12         вывод[i] = перевод[0]
13       Конец Если
14       Если длина слова == длина перевода
15         k = 1
16         Для j = i + 1 до i + длина слова - 1
17           вывод[j] = в нижнем регистре перевод[k]
18           k = k + 1
19         Конец Для
20       Иначе Если длина слова > длина перевода
21         удалить символы в вывод от i + 1 до длина слова - длина перевода
22         k = 1
23         Для j = i + 1 до i + длина перевода - 1
24           вывод[j] = в нижнем регистре перевод[k]
25           k = k + 1
26         Конец Для
27       Иначе
28         переместить символы вправо в вывод от i + длина слова до длина перевода - длина слова
29         k = 1
30         Для j = i + 1 до i + длина перевода - 1
31           вывод[j] = в нижнем регистре перевод[k]
32           k = k + 1
33         Конец Для
34       Конец Если
35     Конец Если
36   Конец Для
37   Вернуть вывод
38 Конец Функции
```

Выполнение работы

В файле parser.c представлены функции обработки ошибок и входных данных. Используется тип данных wchar_t для работы с символами русского алфавита.

```
int amount_of_arguments(int argc) {
  if (argc > 4) {
    printf("Слишком большое количество аргуметов, используйте "
           "форму:\n'./translate text_rus.txt dictionary.txt text_eng.txt'\n");
    return -1;
  } else if (argc < 4) {
    printf("Недостаточно аргуметов, используйте форму:\n'./translate "
           "text_rus.txt dictionary.txt text_eng.txt'\n");
    return -1;
  } else {
    return 0;
  }
}
```

Функция amount_of_arguments принимает параметр argc, который обозначает количество аргументов командной строки. Она возвращает целочисленное значение -1, если argc не равно 4, и 0, если argc равно 4. Она также выводит сообщения об ошибках на экран, если argc не соответствует ожидаемому значению.

```
int is_letter_is_rus(wchar_t letter) {
  if ((letter >= 1040 && letter <= 1103) || letter == 1105 || letter == 1025) {
    return 0;
  }
  return -1;
}

int is_letter_is_eng(wchar_t letter) {
  if ((letter >= 65 && letter <= 90) || (letter >= 97 && letter <= 122)) {
    return 0;
  }
  return -1;
}
```

Функции is_letter_is_rus и is_letter_is_eng принимают на вход двухбайтовый символ и возвращают 0, если буква русская или английская соответственно. Иначе возвращают 0.

```
int language_define(wchar_t letter, int flag) {
  if (flag == 0) {
    if (is_letter_is_rus(letter) == 0) {
      return 1;
    } else {
      return 0;
    }
  } else if (flag == 1 && is_letter_is_eng(letter) == 0) {
    return -1;
  } else if (flag == 0 && is_letter_is_eng(letter) == 0) {
    return -1;
  }
  return flag;
}
```

Функция language_define принимает на вход двухбайтовый символ и flag. Возвращает 1, если флаг нулевой и буква русская, возвращает 0, если буква английская. Возвращает -1 если раннее определённый язык не соответствует настоящему.

```
int is_letters_separate(wchar_t *string) {
  for (int i = 0; string[i] != '\0'; i++) {
    if ((is_letter_is_rus(string[i]) == 0 &&
         is_letter_is_eng(string[i + 1]) == 0) ||
        (is_letter_is_rus(string[i + 1]) == 0 &&
         is_letter_is_eng(string[i]) == 0)) {
      return -1;
    }
  }
  return 0;
}
```

Функция is_letters_separate принимает на вход указатель на двухбайтовую строку. Возвращает -1, если после слова сразу идёт слово на другом языке, иначе 0.

```
int string_check(wchar_t *string) {
  int sep = 0;
  int space = 0;
  int dash = 0;
  if (string[0] == 45 || string[0] == 32) {
    return -1;
  }
  for (int i = 0; string[i] != '\n' && string[i] != '\0'; i++) {
    if (string[i] == 45 || string[i] == 32 ||
        is_letter_is_eng(string[i]) == 0 || is_letter_is_rus(string[i]) == 0) {
      if (string[i] == 45 && string[i + 1] == 32 && string[i - 1] == 32 &&
          (is_letter_is_eng(string[i - 2]) == 0 ||
           is_letter_is_rus(string[i - 2]) == 0) &&
          (is_letter_is_eng(string[i + 2]) == 0 ||
           is_letter_is_rus(string[i + 2]) == 0)) {
        sep++;
      }
      if (string[i] == 45) {
        dash++;
      }
      if (string[i] == 32) {
        space++;
      }
    } else {
      return -1;
    }
  }
  if (sep == 1 && space == 2 && dash == 1) {
    return 0;
  }
  return -1;
}
```

Функция string_check принимает на вход указатель на двухбайтовую строку. Возвращает -1, если слова не разделены символами “ - “. Иначе возвращает 0.

```c
int dictionary_check(char *name) {
  FILE *fp = fopen(name, "r");
  if (!fp) {
    printf("Словарь не найден\n");
    return -1;
  }
  wchar_t string[100];
  int count = 0;
  int flag = 0;
  while (fgetws(string, 100, fp) != NULL) {
    count++;
    if (string_check(string) == -1 || is_letters_separate(string) == -1) {
      printf(
          "Ошибка в %d строке словаря, используйте форму:\n'Слово - Перевод'\n",
          count);
          fclose(fp);
      return -1;
    }
    flag = language_define(string[0], flag);
    if (flag == -1) {
      printf("В %d строке словаря неправильное следование слов, используйте "
             "форму:\n'Слово - Перевод'\n",
             count);
             fclose(fp);
      return -1;
    }
  }
  fclose(fp);
  return 0;
}
```

Функция dictionary_check принимает на вход указатель на строку. Возвращает -1, если в словаре есть ошибка или не удалось открыть файл со словарём. Иначе возвращает 0.

В файле translator.c представлены функции считывания, записи и функции замены слова в строке на его перевод.

```c
typedef struct {
  wchar_t *word;
  wchar_t *translation;
} dictionary;
```

.

В файле translator.h представлен прототип структуры, хранящей слово и его перевод.

```
wchar_t *delwchar(wchar_t *str, int number, int count) {
  for (int i = 0; i != count; i++) {
    for (int j = number; str[j] != '\0'; j++) {
      str[j] = str[j + 1];
    }
  }
  return str;
}
```

Функция delwchar принимает на вход указатель на двухбайтовую строку, номер символа типа int в строке, с которого будет производиться удаление и количество символов типа int, которое будет удалено. Возвращает указатель на двухбайтовую строку.

```
wchar_t *movewchar(wchar_t *str, int n, int count) {
  for (int i = 0; i != count; i++) {
    wcscat(str, L" ");
    for (int j = wcslen(str)-1; j != n; j--){
      str[j] = str[j - 1];
    }
  }
  return str;
}
```

Функция movewchar принимает на вход указатель на двухбайтовую строку, номер символа типа int в строке, с которого будет производиться сдвиг и количество сдвигов типа int. Возвращает указатель на двухбайтовую строку.

```
int wcmp(wchar_t *str, wchar_t *ptr, int n) {
  int j = 0;
  for (int i = n; ptr[j] != '\0'; i++) {
    if (towlower(ptr[j]) != towlower(str[i])) {
      return -1;
    }
    j++;
  }
  return 0;
}
```

Функция wcmp принимает на вход 2 указателя на двухбайтовые строки, номер символа типа int в строке, с которого будет производиться сравнение. Возвращает 0, если строки равны. Иначе возвращает 0.

```c
size_t first_word_size(FILE *fp) {
  size_t count = 0;
  for (wchar_t c = getc(fp); c != ' '; c = getc(fp)) {
    count++;
  }
  return count;
}

size_t second_word_size(FILE *fp) {
  size_t count = 0;
  for (wchar_t c = getc(fp); c != '\n' && c != EOF; c = getc(fp)) {
    count++;
  }
  return count;
}
```

Функции first_word_size и second_word_size принимают на вход указатель на поток. Возвращают размер первого и второго слова типа size_t соответственно.

```c
int ammount_of_lines(char *name) {
  FILE *fp = fopen(name, "r");
  int count = 0;
  for (char c = getc(fp); c != EOF; c = getc(fp)) {
    if (c == '\n' || c == '\0') {
      count++;
    }
  }
  fclose(fp);
  return count;
}
```

Функция ammount_of_lines принимает на вход указатель на строку. Возвращает количество строк данного файла.

```
dictionary *create_dictionary(char *name, int lines) {
  FILE *fp = fopen(name, "r");
  dictionary *d = NULL;
  d = malloc(lines * sizeof(*d));
  if (!d) {
    fclose(fp);
    return NULL;
  }
  for (int i = 0; i != lines; i++) {
    size_t size1 = first_word_size(fp);
    size_t size2 = second_word_size(fp);
    fseek(fp, -size1 - size2 - 2, SEEK_CUR);
    size1++;
    d[i].word = calloc((size1 + 1), sizeof(wchar_t));
    d[i].translation = calloc((size2 + 1), sizeof(wchar_t));
    if (!d[i].word || !d[i].translation) {
      dictionary_free(i - 1, d);
      fclose(fp);
      return NULL;
    }
    fscanf(fp, "%ls - %ls\n", d[i].word, d[i].translation);
  }
  fclose(fp);
  return d;
}
```

Функция create_dictionary принимает на вход указатель на строку, и количество строк файла. Возвращает указатель на массив структур типа dictionary. Если файл не удалось открыть или выделение памяти не произошло, то возвращается NULL.

```
wchar_t *input_reader(char *name) {
  FILE *fin = fopen(name, "r");
  if (!fin) {
    printf("Файл для перевода не найден\n");
    return NULL;
  }
  fseek(fin, 0, SEEK_END);
  size_t size = ftell(fin);
  fseek(fin, 0, SEEK_SET);
  char *tmp = calloc(size + 1, sizeof(char));
  if (!tmp) {
    fclose(fin);
    return NULL;
  }
  char *str = calloc(size + 1, sizeof(char));
  if (!str) {
    free(tmp);
    fclose(fin);
    return NULL;
  }
  while (fgets(str, 100, fin) != NULL) {
    strcat(tmp, str);
  }
  wchar_t *input = calloc((size + 1), sizeof(wchar_t));
  if (!input) {
    fclose(fin);
    free(str);
    free(tmp);
    return NULL;
  }
  mbstowcs(input, tmp, size + 1);
  free(tmp);
  free(str);
  fclose(fin);
  return input;
}
```

Функция input_reader принимает на вход указатель на строку. Возвращает указатель на двухбайтовую строку, считанную с файла.

```
wchar_t *word_replace(wchar_t *output, wchar_t *word, wchar_t *translation) {
  for (int i = 0; output[i] != '\0'; i++) {
    if (towlower(output[i]) == towlower(word[0]) &&
        wcmp(output, word, i) == 0 &&
        is_letter_is_rus(output[i + wcslen(word)]) == -1 &&
        is_letter_is_eng(output[i + wcslen(word)]) == -1) {
      if (output[i] == towlower(word[0])) {
        output[i] = towlower(translation[0]);
      } else if (output[i] == towupper(word[0])) {
        output[i] = towupper(translation[0]);
      } else {
        output[i] = translation[0];
      }
      if (wcslen(word) == wcslen(translation)) {
        int k = 1;
        for (int j = i + 1; translation[k] != '\0'; j++) {
          output[j] = towlower(translation[k]);
          k++;
        }
      } else if (wcslen(word) > wcslen(translation)) {
        output = delwchar(output, i + 1, wcslen(word) - wcslen(translation));
        int k = 1;
        for (int j = i + 1; translation[k] != '\0'; j++) {
          output[j] = towlower(translation[k]);
          k++;
        }
      } else {
        output = realloc(output, (wcslen(translation) + wcslen(output)) * sizeof(wchar_t));
        if (!output){
        return NULL;
        }
        output = movewchar(output, i + wcslen(word), wcslen(translation) - wcslen(word));
        int k = 1;
        for (int j = i + 1; translation[k] != '\0'; j++) {
          output[j] = towlower(translation[k]);
          k++;
        }
      }
    }
  }
  return output;
}
```

Функция word_replace принимает указатели на двухбайтовые строки со строкой текста, который нужно перевести, слово и его перевод. Возвращает указатель на двухбайтовую строку, содержащую текст с одним переведённым словом. Возвращает NULL, если не произошло перевыделение памяти.

```
wchar_t *replacer(wchar_t *input, dictionary *d, int lines) {
  wchar_t *output = calloc(wcslen(input), sizeof(wchar_t));
  if (!output) {
    input_free(input);
    dictionary_free(lines, d);
    return NULL;
  }
  const wchar_t *cinput = input;
  wcscpy(output, cinput);
  for (int i = 0; i != lines; i++) {
    output = word_replace(output, d[i].word, d[i].translation);
    if (output == NULL) {
      dictionary_free(lines, d);
      input_free(input);
      return NULL;
    }
  }
  return output;
}
```

Функция replacer принимает указатель на двухбайтовую строку с текстом, указатель на массив структур типа dictionary и количество строк в файле. Возвращает указатель на двухбайтовую с полностью переведённым текстом.

```
int printer(wchar_t *str, char *name){
  FILE *fout = fopen(name, "w");
  if (!fout) {
    printf("Ошибка при создании файла записи\n");
    return -1;
  }
  fputws(str, fout);
  fclose(fout);
  return 0;
}
```

Функция replacer принимает указатель на двухбайтовую строку с переведённым текстом и указатель на строку с именем файла. Если не удалось открыть файл, возвращается -1, иначе возвращается 0.

```c
void dictionary_free(int lines, dictionary *d) {
  for (int i = 0; i != lines; i++) {
    free(d[i].word);
    free(d[i].translation);
  }
  free(d);
}
```

```c
void input_free(wchar_t *input) { free(input); }

void output_free(wchar_t *output) { free(output); }
```

Функции очистки dictionary_free, input_free и output_free очищают выделенную память для указанных данных.

```c
int main(int argc, char *argv[]) {
  setlocale(LC_ALL, "");
  if (amount_of_arguments(argc) == -1) {
    return -1;
  }
  if (dictionary_check(argv[2]) == -1) {
    return -1;
  }
  int lines = ammount_of_lines(argv[2]);

  dictionary *d = NULL;
  d = create_dictionary(argv[2], lines);

  wchar_t *input = input_reader(argv[1]);
  if (input == NULL) {
    dictionary_free(lines, d);
    return -1;
  }
  wchar_t *output = replacer(input, d, lines);
  if (output == NULL) {
    return -1;
  }
  if (printer(output, argv[3]) == -1) {
    dictionary_free(lines, d);
    input_free(input);
    output_free(output);
    return -1;
  }
  dictionary_free(lines, d);
  input_free(input);
  output_free(output);
  return 0;
}
```

В файле main.c поочерёдно вызываются функции проверки входных данных, после чего читается содержимое словаря и файла с текстом. Вызывается функция replacer, которая заменяет в строке слова на соответствующий им перевод. Выводится строка с готовым текстом, после чего очищается выделенная память. Если в ходе выполнения программы возникает ошибка, она завершается.

Тестовые данные

При вызове с двумя аргументами:

```
[artem@fedora bin]$ ./translate Dictionary.txt output.tx
Недостаточно аргуметов, используйте форму:
'./translate text_rus.txt dictionary.txt text_eng.txt'
```

При вызове с некорректным словарём:

```
Tyger - jТигр
Fear - страх
```

```
[artem@fedora bin]$ ./translate input.txt Dictionary.txt
Ошибка в 1 строке словаря, используйте форму:
'Слово - Перевод'
```

При вызове с входными данными, взятыми из задания:

```
Тигр,
    Тигр,
жгучий страх.
Ты горишь в ночных лесах.
  Чей бессмертный взор,
любя,
Создал страшного тебя?
```

```
Тигр - Tyger
Страх - Fear
Ты - You
Взор - Eye
```

```
Tyger,
    Tyger,
жгучий fear.
You горишь в ночных лесах.
  Чей бессмертный eye,
любя,
Создал страшного тебя?
```

Проверяем с помощью команды valgrind:

```
[artem@fedora bin]$ valgrind ./translate input.txt Dictionary.txt output.txt
==94370== Memcheck, a memory error detector
==94370== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==94370== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==94370== Command: ./translate input.txt Dictionary.txt output.txt
==94370==
==94370==
==94370== HEAP SUMMARY:
==94370==     in use at exit: 0 bytes in 0 blocks
==94370==   total heap usage: 59 allocs, 59 frees, 63,289 bytes allocated
==94370==
==94370== All heap blocks were freed -- no leaks are possible
==94370==
==94370== For lists of detected and suppressed errors, rerun with: -s
==94370== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Листинг программы

main.c

```
1  #include <locale.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <wchar.h>
5
6  #include <libtranslator/parser.h>
7  #include <libtranslator/translator.h>
8
9  int main(int argc, char *argv[]) {
10   setlocale(LC_ALL, "");
11   if (amount_of_arguments(argc) == -1) {
12     return -1;
13   }
14   if (dictionary_check(argv[2]) == -1) {
15     return -1;
16   }
17   int lines = ammount_of_lines(argv[2]);
18
19   dictionary *d = NULL;
20   d = create_dictionary(argv[2], lines);
21
22   wchar_t *input = input_reader(argv[1]);
23   if (input == NULL) {
24     dictionary_free(lines, d);
25     return -1;
26   }
27   wchar_t *output = replacer(input, d, lines);
28   if (output == NULL) {
29     return -1;
30   }
31   if (printer(output, argv[3]) == -1) {
32     dictionary_free(lines, d);
33     input_free(input);
34     output_free(output);
35     return -1;
36   }
37   dictionary_free(lines, d);
38   input_free(input);
39   output_free(output);
40   return 0;
41 }
```

parser.h

```
1  #pragma once
2
3  int amount_of_arguments(int argc);
4  int is_letter_is_rus(wchar_t letter);
5  int is_letter_is_eng(wchar_t letter);
6  int language_define(wchar_t letter, int flag);
7  int is_letters_separate(wchar_t *string);
8  int string_check(wchar_t *string);
9  int dictionary_check(char *name);
```

parser.c

```
1  #include <locale.h>
2  #include <stdio.h>
3  #include <wchar.h>
4
5  #include <libtranslator/parser.h>
6
7  int amount_of_arguments(int argc) {
8    if (argc > 4) {
9      printf("Слишком большое количество аргуметов, используйте "
10           "форму:\n'./translate text_rus.txt dictionary.txt
     text_eng.txt'\n");
11     return -1;
12   } else if (argc < 4) {
13     printf("Недостаточно аргуметов, используйте форму:\n'./translate "
14           "text_rus.txt dictionary.txt text_eng.txt'\n");
15     return -1;
16   } else {
17     return 0;
18   }
19 }
20
21 int is_letter_is_rus(wchar_t letter) {
22 if ((letter >= 1040 && letter <= 1103) || letter == 1105 || letter == 1025){
23     return 0;
24   }
25   return -1;
26 }
27
28 int is_letter_is_eng(wchar_t letter) {
29   if ((letter >= 65 && letter <= 90) || (letter >= 97 && letter <= 122)) {
30     return 0;
31   }
32   return -1;
33 }
34
35 int language_define(wchar_t letter, int flag) {
36   if (flag == 0) {
37     if (is_letter_is_rus(letter) == 0) {
38       return 1;
39     } else {
40       return 0;
41     }
42   } else if (flag == 1 && is_letter_is_eng(letter) == 0) {
43     return -1;
44   } else if (flag == 0 && is_letter_is_eng(letter) == 0) {
45     return -1;
```

```
46    }
47    return flag;
48  }
49
50  int is_letters_separate(wchar_t *string) {
51    for (int i = 0; string[i] != '\0'; i++) {
52      if ((is_letter_is_rus(string[i]) == 0 &&
53           is_letter_is_eng(string[i + 1]) == 0) ||
54          (is_letter_is_rus(string[i + 1]) == 0 &&
55           is_letter_is_eng(string[i]) == 0)) {
56        return -1;
57      }
58    }
59    return 0;
60  }
61
62  int string_check(wchar_t *string) {
63    int sep = 0;
64    int space = 0;
65    int dash = 0;
66    if (string[0] == 45 || string[0] == 32) {
67      return -1;
68    }
69    for (int i = 0; string[i] != '\n' && string[i] != '\0'; i++) {
70      if (string[i] == 45 || string[i] == 32 ||
71      is_letter_is_eng(string[i]) == 0 || is_letter_is_rus(string[i]) == 0){
72        if (string[i] == 45 && string[i + 1] == 32 && string[i - 1] == 32 &&
73            (is_letter_is_eng(string[i - 2]) == 0 ||
74             is_letter_is_rus(string[i - 2]) == 0) &&
75            (is_letter_is_eng(string[i + 2]) == 0 ||
76             is_letter_is_rus(string[i + 2]) == 0)) {
77          sep++;
78        }
79        if (string[i] == 45) {
80          dash++;
81        }
82        if (string[i] == 32) {
83          space++;
84        }
85      } else {
86        return -1;
87      }
88    }
89    if (sep == 1 && space == 2 && dash == 1) {
90      return 0;
91    }
```

```
92    return -1;
93  }
94
95  int dictionary_check(char *name) {
96    FILE *fp = fopen(name, "r");
97    if (!fp) {
98      printf("Словарь не найден\n");
99      return -1;
100   }
101   wchar_t string[100];
102   int count = 0;
103   int flag = 0;
104   while (fgetws(string, 100, fp) != NULL) {
105     count++;
106     if (string_check(string) == -1 || is_letters_separate(string) == -1) {
107       printf(
108     "Ошибка в %d строке словаря, используйте форму:\n'Слово - Перевод'\n",
109           count);
110           fclose(fp);
111       return -1;
112     }
113     flag = language_define(string[0], flag);
114     if (flag == -1) {
115     printf("В %d строке словаря неправильное следование слов, используйте"
116             "форму:\n'Слово - Перевод'\n",
117             count);
118             fclose(fp);
119       return -1;
120     }
121   }
122   fclose(fp);
123   return 0;
124 }
```

translator.h

```
#pragma once

typedef struct {
  wchar_t *word;
  wchar_t *translation;
} dictionary;

wchar_t *delwchar(wchar_t *str, int number, int count);
int wcmp(wchar_t *str, wchar_t *ptr, int n);
size_t first_word_size(FILE *fp);
size_t second_word_size(FILE *fp);
int ammount_of_lines(char *name);
dictionary *create_dictionary(char *name, int lines);
void dictionary_free(int lines, dictionary *d);
wchar_t *input_reader(char *name);
void input_free(wchar_t *input);
void output_free(wchar_t *output);
wchar_t *word_replace(wchar_t *output, wchar_t *word, wchar_t *translation);
wchar_t *replacer(wchar_t *input, dictionary *d, int lines);
int printer(wchar_t *str, char *name);
```

translator.c

```
1  #include <locale.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <wchar.h>
6  #include <wctype.h>
7
8  #include <libtranslator/parser.h>
9  #include <libtranslator/translator.h>
10
11 wchar_t *delwchar(wchar_t *str, int number, int count) {
12   for (int i = 0; i != count; i++) {
13     for (int j = number; str[j] != '\0'; j++) {
14       str[j] = str[j + 1];
15     }
16   }
17   return str;
18 }
19
20 wchar_t *movewchar(wchar_t *str, int n, int count) {
21   for (int i = 0; i != count; i++) {
22     wcscat(str, L" ");
23     for (int j = wcslen(str)-1; j != n; j--){
24       str[j] = str[j - 1];
25     }
26   }
27   return str;
28 }
29
30 int wcmp(wchar_t *str, wchar_t *ptr, int n) {
31   int j = 0;
32   for (int i = n; ptr[j] != '\0'; i++) {
33     if (towlower(ptr[j]) != towlower(str[i])) {
34       return -1;
35     }
36     j++;
37   }
38   return 0;
39 }
40
41 size_t first_word_size(FILE *fp) {
42   size_t count = 0;
43   for (wchar_t c = getc(fp); c != ' '; c = getc(fp)) {
44     count++;
45   }
46   return count;
```

```
47  }
48
49  size_t second_word_size(FILE *fp) {
50    size_t count = 0;
51    for (wchar_t c = getc(fp); c != '\n' && c != EOF; c = getc(fp)) {
52      count++;
53    }
54    return count;
55  }
56
57  int ammount_of_lines(char *name) {
58    FILE *fp = fopen(name, "r");
59    int count = 0;
60    for (char c = getc(fp); c != EOF; c = getc(fp)) {
61      if (c == '\n' || c == '\0') {
62        count++;
63      }
64    }
65    fclose(fp);
66    return count;
67  }
68
69  dictionary *create_dictionary(char *name, int lines) {
70    FILE *fp = fopen(name, "r");
71    dictionary *d = NULL;
72    d = malloc(lines * sizeof(*d));
73    if (!d) {
74      fclose(fp);
75      return NULL;
76    }
77    for (int i = 0; i != lines; i++) {
78      size_t size1 = first_word_size(fp);
79      size_t size2 = second_word_size(fp);
80      fseek(fp, -size1 - size2 - 2, SEEK_CUR);
81      size1++;
82      d[i].word = calloc((size1 + 1), sizeof(wchar_t));
83      d[i].translation = calloc((size2 + 1), sizeof(wchar_t));
84      if (!d[i].word || !d[i].translation) {
85        dictionary_free(i - 1, d);
86        fclose(fp);
87        return NULL;
88      }
89      fscanf(fp, "%ls - %ls\n", d[i].word, d[i].translation);
90    }
91    fclose(fp);
92    return d;
```

```
}

void dictionary_free(int lines, dictionary *d) {
  for (int i = 0; i != lines; i++) {
    free(d[i].word);
    free(d[i].translation);
  }
  free(d);
}

wchar_t *input_reader(char *name) {
  FILE *fin = fopen(name, "r");
  if (!fin) {
    printf("Файл для перевода не найден\n");
    return NULL;
  }
  fseek(fin, 0, SEEK_END);
  size_t size = ftell(fin);
  fseek(fin, 0, SEEK_SET);
  char *tmp = calloc(size + 1, sizeof(char));
  if (!tmp) {
    fclose(fin);
    return NULL;
  }
  char *str = calloc(size + 1, sizeof(char));
  if (!str) {
    free(tmp);
    fclose(fin);
    return NULL;
  }
  while (fgets(str, 100, fin) != NULL) {
    strcat(tmp, str);
  }
  wchar_t *input = calloc((size + 1), sizeof(wchar_t));
  if (!input) {
    fclose(fin);
    free(str);
    free(tmp);
    return NULL;
  }
  mbstowcs(input, tmp, size + 1);
  free(tmp);
  free(str);
  fclose(fin);
  return input;
}
```

```
139
140 void input_free(wchar_t *input) { free(input); }
141
142 void output_free(wchar_t *output) { free(output); }
143
144 wchar_t *word_replace(wchar_t *output, wchar_t *word, wchar_t *translation)
    {
145   for (int i = 0; output[i] != '\0'; i++) {
146     if (towlower(output[i]) == towlower(word[0]) &&
147         wcmp(output, word, i) == 0 &&
148         is_letter_is_rus(output[i + wcslen(word)]) == -1 &&
149         is_letter_is_eng(output[i + wcslen(word)]) == -1) {
150       if (output[i] == towlower(word[0])) {
151         output[i] = towlower(translation[0]);
152       } else if (output[i] == towupper(word[0])) {
153         output[i] = towupper(translation[0]);
154       } else {
155         output[i] = translation[0];
156       }
157       if (wcslen(word) == wcslen(translation)) {
158         int k = 1;
159         for (int j = i + 1; translation[k] != '\0'; j++) {
160           output[j] = towlower(translation[k]);
161           k++;
162         }
163       } else if (wcslen(word) > wcslen(translation)) {
164         output = delwchar(output, i + 1, wcslen(word) - wcslen(translation));
165         int k = 1;
166         for (int j = i + 1; translation[k] != '\0'; j++) {
167           output[j] = towlower(translation[k]);
168           k++;
169         }
170       } else {
171         output = realloc(output, (wcslen(translation) + wcslen(output)) *
    sizeof(wchar_t));
172         if (!output){
173         return NULL;
174         }
175         output = movewchar(output, i + wcslen(word), wcslen(translation) -
    wcslen(word));
176         int k = 1;
177         for (int j = i + 1; translation[k] != '\0'; j++) {
178           output[j] = towlower(translation[k]);
179           k++;
180         }
181       }
182     }
183   }
```

```
184   return output;
185 }
186
187 wchar_t *replacer(wchar_t *input, dictionary *d, int lines) {
188   wchar_t *output = calloc(wcslen(input), sizeof(wchar_t));
189   if (!output) {
190     input_free(input);
191     dictionary_free(lines, d);
192     return NULL;
193   }
194   const wchar_t *cinput = input;
195   wcscpy(output, cinput);
196   for (int i = 0; i != lines; i++) {
197     output = word_replace(output, d[i].word, d[i].translation);
198     if (output == NULL) {
199       dictionary_free(lines, d);
200       input_free(input);
201       return NULL;
202     }
203   }
204   return output;
205 }
206
207 int printer(wchar_t *str, char *name){
208   FILE *fout = fopen(name, "w");
209   if (!fout) {
210     printf("Ошибка при создании файла записи\n");
211     return -1;
212   }
213   fputws(str, fout);
214   fclose(fout);
215   return 0;
216 }
```