**Курсовая работа**

**по дисциплине: «Основы программирования на языке C»**

2018 г

# Задание

Оглавление

[Задание 2](#_Toc517072948)

[Введение 4](#_Toc517072949)

[1 Исходные данные и способ их ввода 5](#_Toc517072950)

[2 Глобальные данные 6](#_Toc517072951)

[2.1 Основные модули программы 6](#_Toc517072952)

[2.2 Описание глобальных данных модуля main.c 6](#_Toc517072953)

[2.3 Описание глобальных данных модуля dict.c 7](#_Toc517072954)

[3 Описание пользовательских макроопределений 8](#_Toc517072955)

[3.1 Макроопределения модуля main 8](#_Toc517072956)

[3.2 Макроопределения модуля dict 8](#_Toc517072957)

[3.3 Макроопределения модуля menu 8](#_Toc517072958)

[3.4 Макроопределения модуля input 9](#_Toc517072959)

[3.5 Глобальные макроопределения 10](#_Toc517072960)

[4 Описание пользовательских функции 11](#_Toc517072961)

[4.1 Функции модуля main 11](#_Toc517072962)

[4.2 Функции модуля dict 13](#_Toc517072963)

[4.3 Функции модуля menu 13](#_Toc517072964)

[4.4 Функции модуля input 14](#_Toc517072965)

[4.5 Функции модуля list 15](#_Toc517072966)

[5 Нестандартные функции 17](#_Toc517072967)

[6 Внешние файлы 18](#_Toc517072968)

[7 Описание порядка работы с программой 19](#_Toc517072969)

[Заключение 22](#_Toc517072970)

[Список литературы 23](#_Toc517072971)

[Приложения 24](#_Toc517072972)

# Введение

Данная работа выполнена в виде программы на языке программирования C в среде разработки C++ Builder.

Программа предназначена для работы с англо-русским словарём. Содержимое словаря хранится в файле dictionary.dat. Графический интерфейс выполнен в стиле MSDOS с использованием символов псевдографики. Он предоставляет полное управление словарём – хранение, отображение, добавление и изменение данных, поиск слов, сортировку по заданному полю, а также предоставление общей и контекстной справки.

Концепция пользовательского интерфейса программы – «всё есть меню». На самом деле, простое всплывающее окно с текстом – это, по сути, многострочное меню с отключенной навигацией (read-only), таблица – это многострочное меню с активной навигацией и несколькими колонками для каждого пункта, и т.д. Таким образом, в программе разработан тип данных для представления универсального семейства меню, с соответствующим набором функций.

Внутреннее представление словаря состоит из тройки строковых значений – «Слово», «Часть Речи» и «Перевод», в отображении также участвует «Количество букв».

Содержимое словаря хранится в файле, находящемся в той же папке, что и исполняемый файл программы.

Ключевые слова: программирование, C, C++ Builder, conio, консоль, меню, символы псевдографики.

# Исходные данные и способ их ввода

**Ввод данных из файла.**

Исходными данными в программе является двоичный файл, хранящий содержимое англо-русского словаря. После запуска и инициализации переменных, программа открывает файл данных на чтение в двоичном режиме и считывает в оперативную память всё содержимое файла, распределяя элементы словаря по узлам односвязного списка.

После того как все содержимое файла прочитано, программа инициализирует основное оконное меню для отображения в табличном виде, и пересылает ему на вход всё содержимое словаря. После этого основная функция программы запускает навигацию по таблице и ждёт действий пользователя.

**Ввода данных с клавиатуры.**

Во время сеанса пользователь может изменять и дополнять содержимое словаря, а также удалять выбранные записи. Для этого необходимо активировать соответствующую функцию через пункт меню или горячую клавишу. При этом на экран будет выведено окно для ввода текстовой информации. Как при добавлении новой, так и при изменении существующей записи, пользователь может редактировать все три поля элемента словаря – слово, часть речи и перевод – разница заключается лишь в том, что при редактировании окно ввода инициализируется текущими значениями выбранной записи.

**Редактирование справочной информации.**

Поскольку вся справочная информация доступна в виде текстовых файлов, находящихся папке с файлом программы, существует возможность изменять и дополнять эту информацию вне сеанса работы с программой. При вызове справки программа всегда отображает текущее содержимое соответствующего справочного файла.

# Глобальные данные

## Основные модули программы

Программа состоит из 5 модулей, каждый из которых отвечает за свой функционал:

* **main** – основной модуль, управляющий текущим состоянием программы;
* **list** – контейнер для пользовательских данных, организованный в виде односвязного списка;
* **dict** – модуль, управляющий загрузкой и сохранением словаря.
* menu – библиотека функций для управления различными видами окон – от обычных текстовых до интерактивных, организованных в виде меню из нескольких пунктов.
* **input** – модуль, отвечающий за управление окнами ввода пользовательской информации.

## Описание глобальных данных модуля main.c

Массив с описаниями элементов верхнего меню:

**ITEM\_DEF top\_menu\_items[top\_item\_count];**

Массив с описаниями элементов меню подтверждения выхода:

**static ITEM\_DEF exit\_menu\_items[exit\_item\_count];**

Массив заголовков основной таблицы:

**char\* main\_headers[main\_column\_count];**

Массив заголовков меню подтверждения выхода

**char\* exit\_headers[exit\_column\_count];**

код завершения для функций обратного вызова

**int exit\_code = 0;**

признак перезапуска основной таблицы

**int redraw\_main = 1;**

признак изменившихся данных

**int data\_modified = 0;**

признак необходимости сохранения данных

**int save\_data = 0;**

признак отмены выхода

**int exit\_canceled = 0;**

индекс для инициализации обновлённого меню

**int initial\_table\_index = 0;**

буфер для хранения настроек консоли:

**CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo;**

экземпляр верхнего меню

**MENU top\_menu;**

указатель на экземпляр основной таблицы

**MENU \*ptable = NULL;**

## Описание глобальных данных модуля dict.c

Глобальная строковая константа – имя файла данных:

**const char\* dict\_file\_name = "dictionary.dat";**

глобальный экземпляр словаря (односвязный список):

**LIST1 dict;**

# Описание пользовательских макроопределений

## Макроопределения модуля main

Максимально допустимая длина строки:

**#define MAX\_STRING 256**

Пользовательские метки, чтобы различать меню в функциях обратного вызова:

* **#define MENU\_TAG\_DEFAULT 0** – метка по умолчанию;
* **#define MENU\_TAG\_TOP 1** – верхнее меню;
* **#define MENU\_TAG\_MAIN 2** – основное меню (таблица данных) ;
* **#define MENU\_TAG\_EXIT 3** – меню подтверждения выхода;
* **#define MENU\_TAG\_HELP\_GLOBAL 4** – окно общей справки;
* **#define MENU\_TAG\_SORT 6** – меню выбора поля для сортировки;

Количество пунктов верхнего меню

**#define top\_item\_count 7**

Количество пунктов верхнего меню

**#define main\_column\_count 4**

Количество ячеек (столбцов) основного табличного меню

**#define exit\_column\_count 1**

Количество пунктов меню подтверждения выхода

**#define exit\_item\_count 3**

## Макроопределения модуля dict

Максимальная длина хранимой в словаре строки:

**#define MAX\_DICT\_STR\_LEN 32**

Количество полей (Слово, Часть речи, Перевод):

**#define DICT\_FLD\_CNT 3**

## Макроопределения модуля menu

Признаки ориентации меню:

* **#define MENU\_ORIENT\_HORZ 0** – горизонтальное
* **#define MENU\_ORIENT\_VERT 1** – вертикальное

Максимальная длина строки заголовка

**#define MAX\_MENU\_HDR 256**

Флаги отрисовки меню и обработки сообщений:

* **#define MENU\_FLAG\_WND 0x**1 – отобразить окно
* **#define MENU\_FLAG\_ITEMS 0x2** – отобразить элементы
* **#define MENU\_DRAW\_SEL 0x4** – отобразить текущий элемент
* **#define MENU\_NAVIGATOR 0x8** – обработка нажатия клавиш навигации и Enter
* **#define MENU\_HOTKEYS 0x10** – обработка дополнительно заданных горячих клавиш
* **#define MENU\_FULL\_FLAGS 0x1F** – все флаги включены

Максимальное количество ячеек (столбцов вертикального меню)

**#define MAX\_CELLS 50**

Флаги прокрутки текущего пункта

* **#define MENU\_CURR\_FWD 0** – на один пункт вперёд
* **#define MENU\_CURR\_REV 1** – на один пункт назад

Символ фоновой зарисовки

**#define MENU\_WHITESPACE ' '**

Наибольшее из двух значений:

**#define MAX(a, b) (((a) > (b)) ? (a) : (b))**

## Макроопределения модуля input

Максимальная длина подписи поля

**#define MAX\_TITLE 50**

Предопределённые индексы для обращения к содержимому:

* **#define TITLE 0** – заголовок
* **#define BUFFER 1** – редактируемое поле данных

Общее количество колонок (заголовок и данные)

**#define COLUMNS 2**

## Глобальные макроопределения

В заголовочном файле cоdes.h объявлены макроопределения для кодов нажатых клавиш и символов псевдографики.

Коды клавиатуры:

* **#define KEY\_ARROW\_UP 72** – стрелка вверх
* **#define KEY\_ARROW\_LEFT 75** – стрелка влево
* **#define KEY\_ARROW\_RIGHT 77** – стрелка вправо
* **#define KEY\_ARROW\_DOWN 80** – стрелка вниз
* **#**define **KEY\_TAB 9** – клавиша табуляции
* **#define KEY\_ENTER 13** – забой (клавиша ввода)
* **#define KEY\_ESC 27** – клавиша Esc
* **#define KEY\_BACKSPACE 8** – клавиша Backspace
* **#define KEY\_SPACE 32** – пробел
* **#define KEY\_DEL 83** – клавиша Del
* **#define KEY\_HOME 71** – клавиша Home
* **#define KEY\_END 79** – клавиша End
* **#define KEY\_F1 59** – клавиша F1
* **#define KEY\_F2 60** – клавиша F2
* **#define KEY\_F3 61** – клавиша F3
* **#define KEY\_F4 62** – клавиша F4
* **#define KEY\_F5 63** – клавиша F5
* **#define KEY\_F6 64** – клавиша F6
* **#define KEY\_F7 65** – клавиша F7
* **#define KEY\_F8 66** – клавиша F8
* **#define KEY\_F9 67** – клавиша F9
* **#define KEY\_F10 68** – клавиша F10
* **#define KEY\_F11 133** – клавиша F11
* **#define KEY\_F12 134** – клавиша F12

Символы псевдографики:

* **#define CHAR\_BORDER\_HORZ 205**
* **#define CHAR\_BORDER\_VERT 186**
* **#define CHAR\_BORDER\_LEFT\_TOP 201**
* **#define CHAR\_BORDER\_LEFT\_BOTTOM 200**
* **#define CHAR\_BORDER\_RIGHT\_TOP 187**
* **#define CHAR\_BORDER\_RIGHT\_BOTTOM 188**
* **#define CHAR\_BORDER\_LEFT\_JOINT 204**
* **#define CHAR\_BORDER\_RIGHT\_JOINT 185**
* **#define CHAR\_BORDER\_TOP\_JOINT 203**
* **#define CHAR\_BORDER\_BOTTOM\_JOINT 202**
* **#define CHAR\_BORDER\_CROSS\_JOINT 206**

# Описание пользовательских функции

## Функции модуля main

**int Run()** – основная рабочая функция программы

**int main(int argc, char\* argv[])** – точка входа программы.

**Функции обратного вызова.**

Обработчики пунктов верхнего меню:

* **int Add(MENU\*, ITEM\*)** – пункт «Добавить»
* **int Edit(MENU\*, ITEM\*)** – пункт «Изменить»
* **int Delete(MENU\*)** – пункт «Удалить»
* **int Search(MENU\*, ITEM\*)** – пункт «Поиск»
* **int Sort(MENU\*, ITEM\*)** – пункт «Сортировка»
* **int Save(MENU\*, ITEM\*)** – пункт «Сохранить»
* **int Help(MENU\*, ITEM\*)** – пункт «Справка»
* **int Exit(MENU\*, ITEM\*)** – пункт «Выйти»

Обработчики пунктов меню подтверждения выхода:

* int **ExitYes(MENU\*, ITEM\*)** – выход с сохранением данных.
* **int ExitNo(MENU\*, ITEM\*)** – выход без сохранения данных.
* **int ExitCancel(MENU\*, ITEM\*)** – отмена выхода.

Обработчики горячих клавиш:

* **int DefaultESC(MENU\*)** – обработчик клавиши Esc по умолчанию, устанавливается для большинства меню и производит закрытие текущего меню без дополнительных действий.
* **int ESC(MENU\*)** – обработчик клавиши Esc для главного и верхнего меню, проверяет необходимость сохранения данных перед выходом.
* **int F1(MENU\*)** – обработчик горячей клавиши F1 (Справка).
* **int F2(MENU\*)** – обработчик горячей клавиши F2 (Сохранение).
* **int F3(MENU\*)** – обработчик горячей клавиши F3 (Поиск).
* **int F4(MENU\*)** – обработчик горячей клавиши F4 (Изменить).
* **int F9(MENU\*)** – обработчик горячей клавиши F9 (Переход к верхнему меню).

Функции обратного вызова для односвязного списка:

* **void curr\_menu\_changed(MENU\* menu, int direction, int wrap)** – обработчик прокрутки меню на один элемент вперёд или назад.
* **int dict\_entry\_display(void\* data, int index, void\* param)** – перенос очередного элемента словаря в элемент описания пункта меню.
* **int dict\_entry\_find(void\* data, void\* param)** – проверка элемента словаря на соответствие критерию поиска.
* **int dict\_entry\_compare(void\* data1, void\* data2, void\* arg)** – сравнение двух элементов словаря.

**Вспомогательные функции.**

**int HelpFromFile(HANDLE hStdOut, const char\* file\_name, const char\* title, SMALL\_RECT rect)** – отображение текста справки из файла на всплывающем окне.

**int ShowMenu(HANDLE hStdOut, ITEM\_DEF\* menu\_items, int item\_count, const char\* title, SMALL\_RECT rect, int flags, int user\_tag, ExecuteHotketCB f1CB)** – отображение упрощённого вертикального меню в один столбец.

**ITEM\_DEF\* MenuItemsFromFile(const char\* file\_name, int max\_count, int max\_len, int\* pcount)** – загрузка пунктов меню из файла.

**void LoadInitialData()** – загрузка тестовых данных в словарь (отладочная функция).

## Функции модуля dict

**DICT\_ENTRY\* dict\_entry\_new(const char\* word\_eng, const char\* word\_part, const char\* word\_rus)** – создание нового элемента словаря.

**int dict\_load(const char\* file\_name)** – загрузка словаря из файла.

**int dict\_entry\_save(void\* data, int index, void\* param)** – сохранение одного элемента словаря в файл.

**int dict\_save(const char\* file\_name)** – сохранение словаря в файл.

**void dict\_entry\_clear(DICT\_ENTRY\* entry)** – удаление элемента словаря.

**void dict\_clear()** – удаление всего словаря из памяти.

## Функции модуля menu

int menu\_init(MENU\* menu, MENU\* parent, HANDLE hstdout,

ITEM\_DEF\* item\_defs, int item\_count, int cell\_count,

int orient, const SMALL\_RECT\* prect, int border,

char\* headers[]) –

инициализация экземпляра меню.

int menu\_add\_hotkey(MENU\* menu, int code, ExecuteHotketCB cb) – добавить обработчик горячей клавиши.

void menu\_clear(MENU\* menu) – очистка экземпляра меню.

void menu\_fill\_wnd(MENU\* menu, int items\_count) – заполнение фона окна меню.

int menu\_draw(MENU\* menu, int flags) – основная оконная функция меню.

void menu\_active\_color(MENU\* menu, WORD attr) – настройка отображения текущего элемента меню.

void menu\_inactive\_color(MENU\* menu, WORD attr) – настройка отображения неактивных элементов меню.

void menu\_prev(MENU\* menu) – выделить предыдущий пункт.

void menu\_next(MENU\* menu) – выделить следующий пункт.

void menu\_gotoxy(MENU\* menu, int x, int y) – переместить курсор по координатам x, y.

void itemMenu(MENU\* menu, int activate) – выделить пункт меню или снять выделение.

void saveCursorPosition(MENU\* menu) – сохранить текущую позицию курсора в переменной.

void menu\_cls(MENU\* menu) – отобразить фон окна меню.

void showCursor(MENU\* menu, int visible) – показать/убрать курсор.

void menu\_del\_curr(MENU\* menu) – удалить текущий элемент.

## Функции модуля input

int box\_init(InputBox\* box, HANDLE handle, SMALL\_RECT rect, char\*\*\* contents, int row\_count) – инициализация экземпляра окна ввода.

void box\_clear(InputBox\* box) – очистка полей экземпляра окна ввода.

int box\_save(InputBox\* box) – сохранить прямоугольник на экране.

int box\_draw(InputBox\* box) – нарисовать окно ввода и запустить цикл обработки нажатия клавиш.

int box\_restore(InputBox\* box) – восстановить прямоугольник на экране.

## Функции модуля list

LIST1\* list1\_new() – создать список.

int list1\_delete(LIST1\* list) – удалить список.

int list1\_init(LIST1\* list) – проинициализировать экземпляр списка

int list1\_clear(LIST1\* list) – очистить поля экземпляра списка.

int list1\_push\_front(LIST1\* list, void\* data) – добавить элемент в начало списка.

int list1\_push\_back(LIST1\* list, void\* item) – добавить элемент в конец списка.

int list1\_for\_each(LIST1\* list, LIST\_ITEM\_PROC find, void\* param) – перебор всех элементов с использованием функции обратного вызова.

void\* list1\_curr\_fwd(LIST1\* list, int wrap) – сместить текущий элемент на одну позицию вперёд.

void\* list1\_curr\_rev(LIST1\* list, int wrap) – сместить текущий элемент на одну позицию назад.

void\* list1\_front(LIST1\* list) – доступ к данным первого элемента.

void\* list1\_back(LIST1\* list) – доступ к данным последнего элемента.

void\* list1\_curr(LIST1\* list) – доступ к данным текущего элемента.

void\* list1\_erase(LIST1\* list, LIST\_ITEM\_FIND cb, void\* param) – удаление элементов, удовлетворяющих условиям функции обратного вызова.

void\* list1\_erase\_current(LIST1\* list) – удаление текущего элемента.

int list1\_get\_current\_index(LIST1\* list) – получение порядкового номера текущего элемента.

int list1\_set\_current\_index(LIST1\* list, int index) – установление текущего элемента по индексу.

int list1\_search(LIST1\* list, LIST\_ITEM\_FIND check, void\* param) – поиск элемента с использованием функции обратного вызова.

void list1\_sort(LIST1\* list, LIST\_ITEM\_COMP compare, void\* param) – сортировка списка с использованием функции обратного вызова.

# Нестандартные функции

В программе использованы функции, не входящие ни в библиотеку Windows API, ни стандартную библиотеку языка C:

int kbhit() – определяет, что была нажата клавиша:

int \_getch() – считывает символ с консоли, не отображая его при этом.

Эти две функции используются в двух модулях – в модуле menu для управления навигацией и выбором пунктов, а также в модуле input для управлением текстовым полем ввода.

# Внешние файлы

Внешними по отношению к программе являются файл данных, в котором хранятся англо-русский словарь, а также файлы общей и контекстной справки:

1. dictionary.dat – файл, хранящий содержимое словаря;
2. help.txt – файл с текстом общей справки;
3. sort.txt – файл с текстом контекстной справки о функции сортировки.

Все внешние файлы хранятся в том же каталоге, что и исполняемый файл программы.

Ожидаемая кодировка тестовых файлов справки – CP866.

# Описание порядка работы с программой

При запуске программы происходит считывание содержимого словаря из файла dictionary.dat и отображение его на главной экранной форме в виде текстовой таблицы. В базе данных для каждого английского слова хранится 3 строки – само слово, часть речи, которой оно является, и перевод на русский язык. В главной таблице отображаются все эти поля и количество букв в английском слове.

Вид главной экранной формы приведён на рисунке 1.

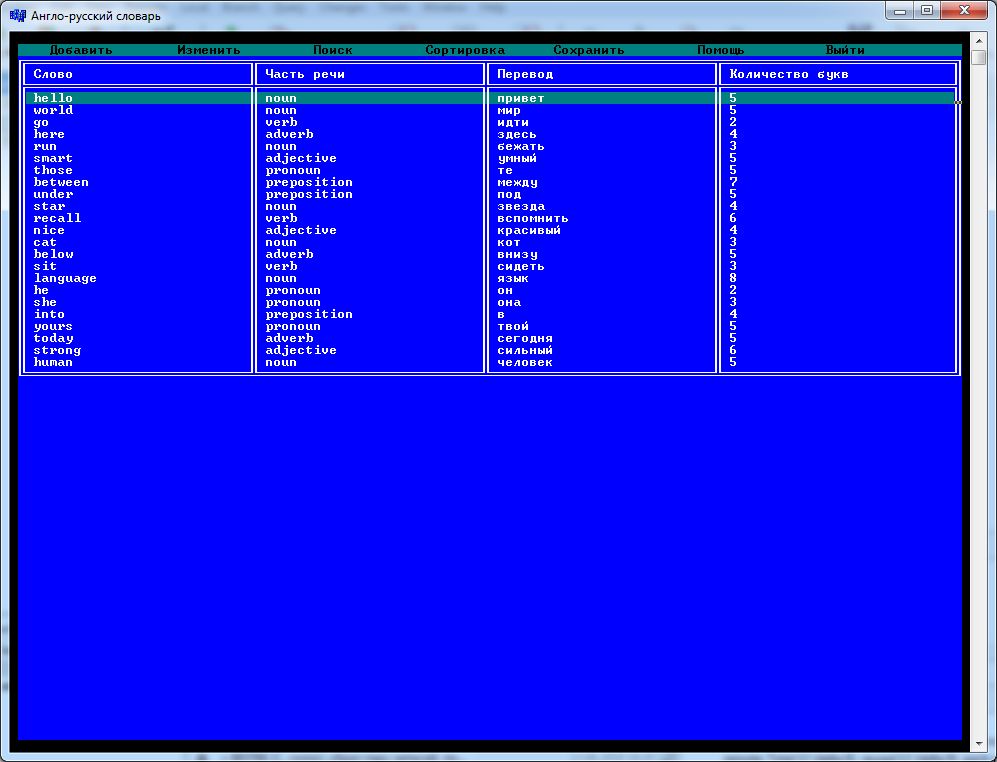


Рисунок 1 – Главное окно программы

Пользователю доступны функции добавления новых записей, а также редактирование и удаление существующих. Кроме того, есть возможность отсортировать все записи по одному из столбцов и найти запись, соответствующую интересующему слову. Все функции доступны через верхнее меню, часть из них также можно вызвать с помощью «горячих» клавиш:

* F1 – вызов справки (если находимся в режиме навигации по таблице, отобразится общая справка, если в режиме выбора поля для сортировки – отобразится контекстная помощь по этой функции);
* F2 – сохранить изменения в файл;
* F3 – поиск;
* F4 и Enter – редактирование текущей записи;
* F9 – переход на верхнее меню.

Пример вызова функции через верхнее меню изображён на рисунке 2.

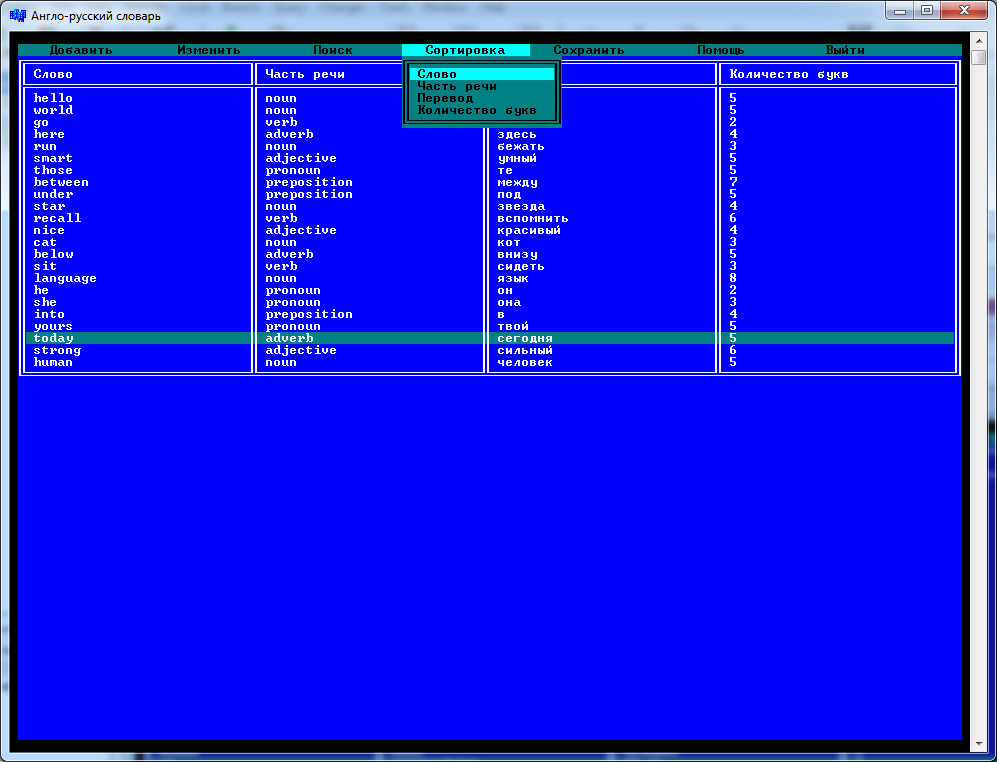


Рисунок 1 – Вызов функции сортировки

Клавиша ESC, как правило, просто закрывает текущее открытое меню и возвращает управление предыдущему. В случае, если ESC нажата в режиме навигации по основной таблице, программа завершится. В том случае, если во время сеанса работы содержимое словаря было изменено – удалена или изменена запись, либо добавлена новая – и пользователь пытается выйти из программы, появится диалоговое окно, напоминающее о несохранённых данных и запрашивающее одно из действий – сохранить перед выходом, выйти без сохранения или отменить выход.

# Заключение

В результате работы была спроектирована и разработана программная система, позволяющая работать с базой данный, представляющей собой англо-русский словарь.

В ходе работы над проектом получены практические навыки разработки программ на языке высокого уровня C. Изучено множество аспектов разработки программ на этом языке – в первую очередь, модульная организация сложной программной системы и управление памятью. Изучены особенности создания с помощью этого языка программирования продвинутого пользовательского интерфейса с использованием только консольного окна.

Кроме того, получены навыки работы в среде C++ Builder на чистом C и без использования библиотеки компонентов VCL.

# Список литературы

1. Стивен Прата Язык программирования C. Лекции и упражнения / Стивен Прата. - М.: Вильямс, 2015. - 928 c.

2. Подбельский В.В, Фомин С.С. Программирование на языке C / М.: «Финансы и статистика». 2003.

3. Дейтел Х.М., Дейтел П.Дж. Как программировать на C / М: «БИНОМ», 2000.

4. Шилдт, Г. Справочник по C. Диасофт М.: 2000.

5. ГОСТ 2.105-95 Общие требования к текстовым документам.

# Приложения

**main.c**

#include <windows.h>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#pragma hdrstop

#include "menu.h"

#include "dict.h"

#include "codes.h"

#include "list.h"

#include "input.h"

#pragma argsused

// Критерий поиска

typedef struct SearchCriteria {

// индес поля для сравнения (Слово, Часть Речи, Перевод, Количесвто букв)

int index;

// фильтр - максимальное значение длины строки Слова

int value;

// строка поиска (для первых трёх полей)

const char\* str;

} SearchCriteria;

// Предварительное объявление функций

// Основная рабочая функция программы

int Run();

// функции обратного вызова

// Функции, которые подключаются к пунктам меню

int Add(MENU\* pm, ITEM\*);

int Edit(MENU\* menu, ITEM\*);

int Delete(MENU\* menu);

int Search(MENU\* menu, ITEM\*);

int Sort(MENU\* menu, ITEM\*);

int Save(MENU\* menu, ITEM\*);

int Help(MENU\* menu, ITEM\*);

int Exit(MENU\* menu, ITEM\*);

// Функции, которые подключаются к горячим клавишам

// Выход - сохранить данные

int ExitYes(MENU\* menu, ITEM\* item);

// Выход - сохранить данные

int ExitNo(MENU\* menu, ITEM\* item);

// Выход - отмена

int ExitCancel(MENU\* menu, ITEM\* item);

// Обработчик клавиши ESC по умолчанию выходит из цикла сообщений текущего меню

int DefaultESC(MENU\* menu);

// Обработчик клавиши ESC

int ESC(MENU\* menu);

// Обработчик клавиши F1 (Справка)

int F1(MENU\* menu);

// Обработчик клавиши F2 (Сохранить)

int F2(MENU\* menu);

// Обработчик клавиши F3 (Поиск)

int F3(MENU\* menu);

// Обработчик клавиши F4 (Редактирование)

int F4(MENU\* menu);

// Обработчик клавиши F9 (Переход на верхнее меню)

int F9(MENU\* menu);

// Обработчик прокрутки меню на один элемент вперёд или назад

void curr\_menu\_changed(MENU\* menu, int direction, int wrap);

// Перенос очередного элемента словаря в элемент описания пункта меню

int dict\_entry\_display(void\* data, int index, void\* param);

// Проверка элемента словаря на соответствие критерию поиска

int dict\_entry\_find(void\* data, void\* param);

// Сравнение двух элементов словаря

int dict\_entry\_compare(void\* data1, void\* data2, void\* arg);

// вспомогательные функции

// Отображение текста справки из файла на всплывающем окне

int HelpFromFile(HANDLE hStdOut, const char\* file\_name, const char\* title, SMALL\_RECT rect);

// Отображение упрощённого вертикального меню в один столбец

int ShowMenu(HANDLE hStdOut, ITEM\_DEF\* menu\_items, int item\_count, const char\* title, SMALL\_RECT rect, int flags, int user\_tag, ExecuteHotketCB f1CB);

// Загрузка пунктов меню из файла

ITEM\_DEF\* MenuItemsFromFile(const char\* file\_name, int max\_count, int max\_len, int\* pcount);

// Загрузка тестовых данных в словарь

void LoadInitialData();

// максимально допустимая длина строки

#define MAX\_STRING 256

// пользовательские метки, чтобы различать меню в функциях обратного вызова

#define MENU\_TAG\_DEFAULT 0

#define MENU\_TAG\_TOP 1

#define MENU\_TAG\_MAIN 2

#define MENU\_TAG\_EXIT 3

#define MENU\_TAG\_HELP\_GLOBAL 4

#define MENU\_TAG\_SORT 5

// количество пунктов верхнего меню

#define top\_item\_count 7

// количество ячеек (столбцов) основного табличного меню

#define main\_column\_count 4

// количество ячеек (столбцов) меню подтверждения выхода

#define exit\_column\_count 1

// количество пунктов меню подтверждения выхода

#define exit\_item\_count 3

// массив с описаниями элементов верхнего меню

ITEM\_DEF top\_menu\_items[top\_item\_count] = {

{ { "Добавить", 0 }, Add },

{ { "Изменить", 0 }, Edit },

{ { "Поиск", 0 }, Search },

{ { "Сортировка", 0 }, Sort },

{ { "Сохранить", 0 }, Save },

{ { "Помощь", 0 }, Help },

{ { "Выйти", 0 }, Exit },

};

// массив с описаниями элементов меню подтверждения выхода

static ITEM\_DEF exit\_menu\_items[exit\_item\_count] = {

{ { "Да", 0 }, ExitYes },

{ { "Нет", 0 }, ExitNo },

{ { "Отмена", 0 }, ExitCancel },

};

// массив заголовков основной таблицы

char\* main\_headers[main\_column\_count];

// массив заголовков (1 заголовок) меню подтверждения выхода

char\* exit\_headers[exit\_column\_count];

// код завершения для функций обратного вызова

int exit\_code = 0; // не нулевое значение считается обязательным

// признак перезапуска основной таблицы

int redraw\_main = 1;

// признак изменившихся данных

int data\_modified = 0;

// признак необходимости сохранения данных

int save\_data = 0;

// признак отмены выхода

int exit\_canceled = 0;

// индекс для инициализации обновлённого меню

int initial\_table\_index = 0;

// буфер для хранения настроек консоли

CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo;

// экземпляр верхнего меню

MENU top\_menu;

// указатель на экземпляр основной таблицы

MENU \*ptable = NULL;

// Точка входа программы

int main(int argc, char\* argv[])

{

return Run();

}

// Выход - сохранить данные

int ExitYes(MENU\* menu, ITEM\* item) {

save\_data = 1;

return -1;

}

// Выход - не сохранять данные

int ExitNo(MENU\* menu, ITEM\* item) {

save\_data = 0;

return -1;

}

// Выход - отмена

int ExitCancel(MENU\* menu, ITEM\* item) {

exit\_canceled = 1;

return -1;

}

// Обработчик клавиши ESC по умолчанию выходит из цикла сообщений текущего меню

int DefaultESC(MENU\* menu) {

return -1;

}

// Обработчик клавиши ESC

int ESC(MENU\* menu) {

return Exit(menu, NULL);

}

// Обработчик клавиши F1 (Справка)

int F1(MENU\* menu) {

Help(menu, NULL);

return 0;

}

// Обработчик клавиши F2 (Сохранить)

int F2(MENU\* menu) {

Save(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Обработчик клавиши F3 (Поиск)

int F3(MENU\* menu) {

Search(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Обработчик клавиши F4 (Редактирование)

int F4(MENU\* menu) {

Edit(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Обработчик клавиши F9 (Переход на верхнее меню)

int F9(MENU\* menu) {

menu\_draw(&top\_menu, MENU\_FULL\_FLAGS);

if(exit\_code)

return exit\_code;

return 0;

}

// Обработчик прокрутки меню на один элемент вперёд или назад

void curr\_menu\_changed(MENU\* menu, int direction, int wrap) {

switch(direction) {

case MENU\_CURR\_REV:

list1\_curr\_rev(&dict, wrap);

break;

case MENU\_CURR\_FWD:

list1\_curr\_fwd(&dict, wrap);

break;

}

}

// Перенос очередного элемента словаря в элемент описания пункта меню

int dict\_entry\_display(void\* data, int index, void\* param) {

int fld;

DICT\_ENTRY\* entry = (DICT\_ENTRY\*)data;

ITEM\_DEF\* main\_menu\_items = (ITEM\_DEF\*)param;

for(fld = 0; fld < DICT\_FLD\_CNT; fld++)

CharToOemA(entry->field[fld], main\_menu\_items[index].str[fld]);

itoa(strlen(entry->field[0]), main\_menu\_items[index].str[DICT\_FLD\_CNT], 10);

main\_menu\_items[index].cb = Edit;

return 1; // продолжить итерации по остальным элементам

}

// Проверка - подходит ли элемент словаря критерию поиска

int dict\_entry\_find(void\* data, void\* param) {

DICT\_ENTRY\* entry = (DICT\_ENTRY\*)data;

SearchCriteria\* search = (SearchCriteria\*)param;

if(search->index < DICT\_FLD\_CNT)

return strlen(entry->field[0]) <= search->value &&

strcmp(search->str, entry->field[search->index]) == 0 ? 1 : 0;

// поиск по "Количеству букв" (закомментирован как бессмысленный)

return 0;//atoi(search->str) == strlen(entry->field[0]) ? 1 : 0;

}

// Сравнение двух элементов словаря

int dict\_entry\_compare(void\* data1, void\* data2, void\* arg) {

DICT\_ENTRY \*a, \*b;

int index;

if(!arg)

return 0;

a = (DICT\_ENTRY\*)data1;

b = (DICT\_ENTRY\*)data2;

index = \*(int\*)arg;

if(index < 3)

return strcmp(a->field[index], b->field[index]) < 0 ? 1 : 0;

return strlen(a->field[0]) < strlen(b->field[0]) ? 1 : 0;

}

// Основная рабочая функция программы

int Run() {

char\* title = "Англо-русский словарь";

int i, fld;

HANDLE hstdout;

SMALL\_RECT rect;

ITEM\_DEF\* main\_menu\_items = NULL;

int main\_menu\_items\_count = 0;

WORD background = BACKGROUND\_INTENSITY | BACKGROUND\_BLUE;

#if 0

LoadInitialData();

return 0;

#endif

// Инициализация заголовков глобальных меню

for(i = 0; i < main\_column\_count; i++)

main\_headers[i] = (char\*)malloc(MAX\_MENU\_HDR \* sizeof(char));

CharToOemA("Слово", main\_headers[0]);

CharToOemA("Часть речи", main\_headers[1]);

CharToOemA("Перевод", main\_headers[2]);

CharToOemA("Количество букв", main\_headers[3]);

for(i = 0; i < exit\_column\_count; i++)

exit\_headers[i] = (char\*)malloc(MAX\_MENU\_HDR \* sizeof(char));

CharToOemA("Сохранить изменения?", exit\_headers[0]);

for(i = 0; i < exit\_item\_count; i++)

CharToOemA(exit\_menu\_items[i].str[0], exit\_menu\_items[i].str[0]);

CharToOemA(title, title);

SetConsoleTitle(title);

hstdout = GetStdHandle(STD\_OUTPUT\_HANDLE); // дескриптор консольного окна

if(INVALID\_HANDLE\_VALUE == hstdout) {

return -1;

}

// Получаем размеры консоли

GetConsoleScreenBufferInfo(hstdout, &csbInfo);

rect.Left = csbInfo.srWindow.Left;

rect.Right = csbInfo.srWindow.Right;

rect.Top = csbInfo.srWindow.Top;

rect.Bottom = rect.Top;

SetConsoleTextAttribute(hstdout, background);

system("cls");

menu\_init(&top\_menu, NULL, hstdout, top\_menu\_items, top\_item\_count, 1,

MENU\_ORIENT\_HORZ, &rect, 0, NULL);

top\_menu.user\_tag = MENU\_TAG\_TOP;

menu\_active\_color(&top\_menu, BACKGROUND\_INTENSITY | BACKGROUND\_BLUE | BACKGROUND\_GREEN);

menu\_inactive\_color(&top\_menu, BACKGROUND\_BLUE | BACKGROUND\_GREEN);

menu\_add\_hotkey(&top\_menu, KEY\_ESC, DefaultESC);

menu\_draw(&top\_menu, MENU\_FLAG\_WND | MENU\_FLAG\_ITEMS);

// Загрузка словаря из файла

if(-1 == dict\_load(dict\_file\_name)) {

return -1;

}

rect.Top++;

rect.Bottom = csbInfo.srWindow.Bottom - 1;

main\_menu\_items\_count = dict.count;

while (redraw\_main) {

redraw\_main = 0; // любой callback может запланировать перерисовку

//

if(main\_menu\_items) {

for(i = 0; i < main\_menu\_items\_count; i++)

for(fld = 0; fld < DICT\_FLD\_CNT + 1; fld++)

if(main\_menu\_items[i].str[fld])

free(main\_menu\_items[i].str[fld]);

free(main\_menu\_items);

main\_menu\_items = NULL;

}

main\_menu\_items\_count = dict.count; // словарь мог быть изменён

main\_menu\_items = (ITEM\_DEF\*)malloc(main\_menu\_items\_count \* sizeof(ITEM\_DEF));

memset(main\_menu\_items, 0x00, main\_menu\_items\_count \* sizeof(ITEM\_DEF));

for(i = 0; i < main\_menu\_items\_count; i++) {

for(fld = 0; fld < DICT\_FLD\_CNT; fld++)

main\_menu\_items[i].str[fld] = (char\*)malloc(MAX\_STRING \* sizeof(char));

main\_menu\_items[i].str[DICT\_FLD\_CNT] = (char\*)malloc(32 \* sizeof(char));

}

//

if(ptable) {

menu\_clear(ptable);

free(ptable);

}

ptable = (MENU\*)malloc(sizeof(MENU));

// перекидываем словарь в таблицу

list1\_for\_each(&dict, dict\_entry\_display, main\_menu\_items);

// создаём элементы меню

menu\_init(ptable, NULL, hstdout, main\_menu\_items, main\_menu\_items\_count, main\_column\_count,

MENU\_ORIENT\_VERT, &rect, 1, main\_headers);

// устанавливаем текущий индекс

list1\_set\_current\_index(&ptable->items, initial\_table\_index);

initial\_table\_index = 0;

// продолжаем настройку меню

ptable->user\_tag = MENU\_TAG\_MAIN;

menu\_add\_hotkey(ptable, KEY\_F1, F1);

menu\_add\_hotkey(ptable, KEY\_F2, F2);

menu\_add\_hotkey(ptable, KEY\_F3, F3);

menu\_add\_hotkey(ptable, KEY\_F4, F4);

menu\_add\_hotkey(ptable, KEY\_F9, F9);

menu\_add\_hotkey(ptable, KEY\_DEL, Delete);

menu\_add\_hotkey(ptable, KEY\_ESC, ESC);

ptable->changed\_cb = curr\_menu\_changed;

menu\_active\_color(ptable,

BACKGROUND\_BLUE | BACKGROUND\_GREEN

| FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE

);

menu\_inactive\_color(ptable,

background | FOREGROUND\_INTENSITY | FOREGROUND\_RED |

FOREGROUND\_GREEN | FOREGROUND\_BLUE);

// блокирующий вызов: основной цикл обработки сообщений главной таблицы

menu\_draw(ptable, MENU\_FULL\_FLAGS);

// возвожно, вышли по глобальному коду завершения, сбрасываем его

exit\_code = 0;

}

menu\_clear(ptable);

free(ptable);

for(i = 0; i < main\_menu\_items\_count; i++)

for(fld = 0; fld < DICT\_FLD\_CNT + 1; fld++)

if(main\_menu\_items[i].str[fld])

free(main\_menu\_items[i].str[fld]);

free(main\_menu\_items);

menu\_clear(&top\_menu);

for(i = 0; i < main\_column\_count; i++)

if(main\_headers[i])

free(main\_headers[i]);

return 0;

}

//--------------------------------------------

// Функции меню

//--------------------------------------------

// Функция меню <Выход>

int Exit(MENU\* menu, ITEM\* item) {

int result;

if(data\_modified) {

int i;

MENU exit\_menu;

SMALL\_RECT rect;

const int width = 24;

const int height = 7;

rect.Left = (csbInfo.srWindow.Right - csbInfo.srWindow.Left - width) / 2;

rect.Right = rect.Left + width - 1;

rect.Top = (csbInfo.srWindow.Bottom - csbInfo.srWindow.Top - height) / 2;

rect.Bottom = rect.Top + height - 1;

menu\_init(&exit\_menu, ptable, menu->hStdOut, exit\_menu\_items, exit\_item\_count,

exit\_column\_count, MENU\_ORIENT\_VERT, &rect, 1, exit\_headers);

exit\_menu.user\_tag = MENU\_TAG\_EXIT;

menu\_active\_color(&exit\_menu, BACKGROUND\_INTENSITY | BACKGROUND\_BLUE | BACKGROUND\_GREEN);

menu\_inactive\_color(&exit\_menu, BACKGROUND\_BLUE | BACKGROUND\_GREEN);

menu\_add\_hotkey(&exit\_menu, KEY\_ESC, DefaultESC);

menu\_draw(&exit\_menu, MENU\_FULL\_FLAGS);

menu\_clear(&exit\_menu);

if(save\_data) {

Save(menu, item);

}

}

if(exit\_canceled) {

exit\_canceled = 0;

result = 0;

} else {

exit\_code = -1;

result = -1;

}

return result;

}

// Функция меню <Добавить>

int Add(MENU\* pm, ITEM\* item) {

int i;

SMALL\_RECT rect = { 10, 11, 49, 15 };

int max\_width;

int menu\_width = 40;

int menu\_height = DICT\_FLD\_CNT + 2; // поля и отступ

int row\_count = DICT\_FLD\_CNT;

char\*\*\* contents;

DICT\_ENTRY\* entry;

InputBox box;

int left = csbInfo.srWindow.Left, right = csbInfo.srWindow.Right;

int top = csbInfo.srWindow.Top, bottom = csbInfo.srWindow.Bottom;

rect.Left = left + (right - left + 1 - menu\_width) / 2;

rect.Right = rect.Left + menu\_width - 1;

rect.Top = top + (bottom - top + 1 - menu\_height) / 2;

rect.Bottom = rect.Top + menu\_height - 1;

max\_width = (rect.Right - rect.Left - 1) / 2;

contents = (char\*\*\*)malloc(row\_count \* sizeof(char\*\*));

memset(contents, 0x00, row\_count \* sizeof(char\*\*));

for(i = 0; i < row\_count; ++i) {

contents[i] = (char\*\*)malloc(COLUMNS \* sizeof(char\*));

// надпись

contents[i][TITLE] = (char\*)malloc((MAX\_TITLE + 1) \* sizeof(char));

strcpy(contents[i][TITLE], main\_headers[i]);

//CharToOemA(main\_headers[i], contents[i][TITLE]);

// буфер ввода

contents[i][BUFFER] = (char\*)malloc((max\_width + 1) \* sizeof(char));

memset(contents[i][BUFFER], ' ', max\_width \* sizeof(char));

//strcpy(contents[i][BUFFER], entry->field[i]);

//CharToOemA(entry->field[i], contents[i][BUFFER]);

contents[i][BUFFER][0] = '\0';

contents[i][BUFFER][max\_width] = '\0';

}

if(-1 == box\_init(&box, pm->hStdOut, rect, contents, row\_count)) {

return -1;

}

if(-1 == box\_save(&box)) {

box\_clear(&box);

return -1;

}

if(1 == box\_draw(&box)) {

for(i = 0; i < row\_count; ++i) {

//strcpy(entry->field[i], contents[i][BUFFER]);

OemToCharA(contents[i][BUFFER], contents[i][BUFFER]);

}

list1\_push\_back(&dict,

dict\_entry\_new(contents[0][BUFFER], contents[1][BUFFER], contents[2][BUFFER])

);

initial\_table\_index = dict.count - 1;

data\_modified = 1;

redraw\_main = 1;

exit\_code = -1;

}

box\_clear(&box);

for(i = 0; i < row\_count; ++i) {

if(contents[i]) {

if(contents[i][TITLE])

free(contents[i][TITLE]);

if(contents[i][BUFFER]);

free(contents[i][BUFFER]);

free(contents[i]);

}

}

free(contents);

contents = NULL;

return exit\_code;

}

// Функция меню <Изменить>

int Edit(MENU\* pm, ITEM\* item) {

int i, box\_result;

SMALL\_RECT rect = { 10, 11, 49, 15 };

int max\_width;

int menu\_width = 40;

int menu\_height = DICT\_FLD\_CNT + 2; // поля и отступ

int row\_count = DICT\_FLD\_CNT;

char\*\*\* contents;

DICT\_ENTRY\* entry = (DICT\_ENTRY\*)list1\_curr(&dict);

InputBox box;

int left = csbInfo.srWindow.Left, right = csbInfo.srWindow.Right;

int top = csbInfo.srWindow.Top, bottom = csbInfo.srWindow.Bottom;

rect.Left = left + (right - left + 1 - menu\_width) / 2;

rect.Right = rect.Left + menu\_width - 1;

rect.Top = top + (bottom - top + 1 - menu\_height) / 2;

rect.Bottom = rect.Top + menu\_height - 1;

max\_width = (rect.Right - rect.Left - 1) / 2;

contents = (char\*\*\*)malloc(row\_count \* sizeof(char\*\*));

memset(contents, 0x00, row\_count \* sizeof(char\*\*));

for(i = 0; i < row\_count; ++i) {

contents[i] = (char\*\*)malloc(COLUMNS \* sizeof(char\*));

// надпись

contents[i][TITLE] = (char\*)malloc((MAX\_TITLE + 1) \* sizeof(char));

strcpy(contents[i][TITLE], main\_headers[i]);

//CharToOemA(main\_headers[i], contents[i][TITLE]);

// буфер ввода

contents[i][BUFFER] = (char\*)malloc((max\_width + 1) \* sizeof(char));

memset(contents[i][BUFFER], ' ', max\_width \* sizeof(char));

//strcpy(contents[i][BUFFER], entry->field[i]);

CharToOemA(entry->field[i], contents[i][BUFFER]);

contents[i][BUFFER][max\_width] = '\0';

}

initial\_table\_index = list1\_get\_current\_index(&dict);

if(-1 == box\_init(&box, pm->hStdOut, rect, contents, row\_count)) {

return -1;

}

if(-1 == box\_save(&box)) {

box\_clear(&box);

return -1;

}

box\_result = box\_draw(&box);

if(1 == box\_result) {

for(i = 0; i < row\_count; ++i) {

//strcpy(entry->field[i], contents[i][BUFFER]);

OemToCharA(contents[i][BUFFER], entry->field[i]);

}

data\_modified = 1;

redraw\_main = 1;

exit\_code = -1;

}

box\_clear(&box);

for(i = 0; i < row\_count; ++i) {

if(contents[i]) {

if(contents[i][TITLE])

free(contents[i][TITLE]);

if(contents[i][BUFFER]);

free(contents[i][BUFFER]);

free(contents[i]);

}

}

free(contents);

contents = NULL;

return exit\_code;

}

// Функция меню <Удалить>

int Delete(MENU\* menu) {

list1\_erase\_current(&dict);

menu\_del\_curr(menu);

menu\_fill\_wnd(menu, 0);

menu\_cls(menu);

menu\_draw(menu, MENU\_FLAG\_WND | MENU\_FLAG\_ITEMS | MENU\_DRAW\_SEL);

data\_modified = 1;

return 0;

}

// Функция подменю <Поиск>

int SearchItem(MENU\* pm, ITEM\* item) {

int i;

int index;

SMALL\_RECT rect = { 3, 11, 50, 14 };

int menu\_width = 48;

int menu\_height = 4;

int max\_width;

char\* titles[] = { "Строка поиска:", "Лимит букв в Слове" };

char\*\*\* contents;

int row\_count = 2;

SearchCriteria search;

InputBox box;

int left = csbInfo.srWindow.Left, right = csbInfo.srWindow.Right;

int top = csbInfo.srWindow.Top, bottom = csbInfo.srWindow.Bottom;

rect.Left = left + (right - left + 1 - menu\_width) / 2;

rect.Right = rect.Left + menu\_width - 1;

rect.Top = top + (bottom - top + 1 - menu\_height) / 2 + 4;

rect.Bottom = rect.Top + menu\_height - 1;

max\_width = (rect.Right - rect.Left - 1) / 2;

contents = (char\*\*\*)malloc(row\_count \* sizeof(char\*\*));

memset(contents, 0x00, row\_count \* sizeof(char\*\*));

for(i = 0; i < row\_count; ++i) {

contents[i] = (char\*\*)malloc(COLUMNS \* sizeof(char\*));

// надпись

contents[i][TITLE] = (char\*)malloc((MAX\_TITLE + 1) \* sizeof(char));

if (0 == i) {

strcpy(contents[i][TITLE], item->str + 1); // копируем без начального пробела

} else {

CharToOemA(titles[i], contents[i][TITLE]);

}

// буфер ввода

contents[i][BUFFER] = (char\*)malloc((max\_width + 1) \* sizeof(char));

memset(contents[i][BUFFER], ' ', max\_width \* sizeof(char));

contents[i][BUFFER][0] = '\0';

contents[i][BUFFER][max\_width] = '\0';

}

strcpy(contents[1][BUFFER], "20");

if(-1 == box\_init(&box, pm->hStdOut, rect, contents, row\_count)) {

return -1;

}

if(-1 == box\_save(&box)) {

box\_clear(&box);

return -1;

}

if(-1 == box\_draw(&box)) {

box\_clear(&box);

return -1;

}

search.str = contents[0][BUFFER];

OemToCharA(search.str, search.str);

search.index = item->index;

search.value = atoi(contents[1][BUFFER]);

index = list1\_search(&dict, dict\_entry\_find, &search);

list1\_set\_current\_index(&ptable->items, index);

menu\_fill\_wnd(ptable, 0);

menu\_cls(ptable);

menu\_draw(ptable, MENU\_FLAG\_WND | MENU\_FLAG\_ITEMS | MENU\_DRAW\_SEL);

box\_clear(&box);

for(i = 0; i < row\_count; ++i) {

if(contents[i]) {

if(contents[i][TITLE])

free(contents[i][TITLE]);

if(contents[i][BUFFER]);

free(contents[i][BUFFER]);

free(contents[i]);

}

}

free(contents);

contents = NULL;

return -1;

}

// Функция меню <Поиск>

int Search(MENU\* pm, ITEM\* item) {

int i;

int max\_len;

// без бессмысленного поиска по "Количеству букв"

int item\_count = main\_column\_count - 1;

int slen;

int menu\_width = 20;

int menu\_height = item\_count + 2;

SMALL\_RECT rect;

ITEM\_DEF\* menu\_items;

if(item) {

rect.Left = item->x;

rect.Right = rect.Left + menu\_width - 1;

rect.Top = csbInfo.srWindow.Top + 2;

rect.Bottom = rect.Top + menu\_height - 1;

} else {

int left = csbInfo.srWindow.Left, right = csbInfo.srWindow.Right;

int top = csbInfo.srWindow.Top, bottom = csbInfo.srWindow.Bottom;

rect.Left = left + (right - left + 1 - menu\_width) / 2;

rect.Right = rect.Left + menu\_width - 1;

rect.Top = top + (bottom - top + 1 - menu\_height) / 2;

rect.Bottom = rect.Top + menu\_height - 1;

}

max\_len = menu\_width - 2; // 2 верт. линии по краям

menu\_items = (ITEM\_DEF\*)malloc(item\_count \* sizeof(ITEM\_DEF));

memset(menu\_items, 0x00, item\_count \* sizeof(ITEM\_DEF));

for(i = 0; i < item\_count; i++) {

menu\_items[i].str[0] = (char\*)malloc(max\_len \* sizeof(char));

strncpy(menu\_items[i].str[0], main\_headers[i], max\_len);

menu\_items[i].str[0][max\_len] = '\0';

menu\_items[i].cb = SearchItem;

}

ShowMenu(pm->hStdOut, menu\_items, item\_count, NULL, rect, MENU\_FULL\_FLAGS, MENU\_TAG\_SORT, F1);

return -1;

}

// Функция подменю <Сортировка>

int SortItem(MENU\* pm, ITEM\* item) {

list1\_sort(&dict, dict\_entry\_compare, &item->index);

initial\_table\_index = list1\_get\_current\_index(&dict);

redraw\_main = 1;

exit\_code = -1;

return exit\_code;

}

// Функция меню <Сортировка>

int Sort(MENU\* pm, ITEM\* item) {

int i;

int max\_len;

int item\_count = main\_column\_count;

int slen;

int menu\_width = 20;

int menu\_height = item\_count + 2;

SMALL\_RECT rect;

ITEM\_DEF\* menu\_items;

rect.Left = item->x;

rect.Right = item->x + menu\_width - 1;

rect.Top = csbInfo.srWindow.Top + 2;

rect.Bottom = rect.Top + menu\_height - 1;

max\_len = menu\_width - 2; // 2 верт. линии по краям

menu\_items = (ITEM\_DEF\*)malloc(item\_count \* sizeof(ITEM\_DEF));

memset(menu\_items, 0x00, item\_count \* sizeof(ITEM\_DEF));

for(i = 0; i < item\_count; i++) {

menu\_items[i].str[0] = (char\*)malloc(max\_len \* sizeof(char));

strncpy(menu\_items[i].str[0], main\_headers[i], max\_len);

menu\_items[i].str[0][max\_len] = '\0';

menu\_items[i].cb = SortItem;

}

ShowMenu(pm->hStdOut, menu\_items, item\_count, NULL, rect, MENU\_FULL\_FLAGS, MENU\_TAG\_SORT, F1);

return -1;

}

// Функция меню <Сохранить>

int Save(MENU\* menu, ITEM\* item) {

save\_data = 0;

data\_modified = 0;

dict\_save(dict\_file\_name);

return -1;

}

// Функция меню <Помощь>. Заполняется кодом пользователя

int Help(MENU\* pm, ITEM\* item) {

int result = 0;

SMALL\_RECT rect;

int width = 0, height = 0;

switch(pm->user\_tag) {

case MENU\_TAG\_TOP:

case MENU\_TAG\_MAIN:

rect.Left = csbInfo.srWindow.Left + 2;

rect.Right = csbInfo.srWindow.Right - 2;

rect.Top = csbInfo.srWindow.Top + 3;

rect.Bottom = csbInfo.srWindow.Bottom - 2;

result = HelpFromFile(pm->hStdOut, "help.txt", "Справка о программе", rect);

break;

case MENU\_TAG\_SORT:

width = 55;

height = 6;

rect.Left = (csbInfo.srWindow.Right - csbInfo.srWindow.Left - width) / 2;

rect.Right = rect.Left + width - 1;

rect.Top = (csbInfo.srWindow.Bottom - csbInfo.srWindow.Top - height) / 2;

rect.Bottom = rect.Top + height - 1;

result = HelpFromFile(pm->hStdOut, "sort.txt", "Справка: Функция сортировки", rect);

break;

}

return result;

}

// Вспомогательные функции

// Загрузка пунктов меню из файла

ITEM\_DEF\* MenuItemsFromFile(const char\* file\_name, int max\_count, int max\_len, int\* pcount) {

FILE\* file;

int i;

int item\_count = 0;

int slen;

ITEM\_DEF\* menu\_items = (ITEM\_DEF\*)malloc(max\_count \* sizeof(ITEM\_DEF));

if(!menu\_items)

return NULL;

memset(menu\_items, 0x00, max\_count \* sizeof(ITEM\_DEF));

file = fopen(file\_name, "r");

if(!file) {

free(menu\_items);

return NULL;

}

i = 0;

while(!feof(file)) {

menu\_items[i].str[0] = (char\*)malloc(max\_len \* sizeof(char));

if(!menu\_items[i].str[0])

break;

menu\_items[i].str[0][0] = '\0';

if(fgets(menu\_items[i].str[0], max\_len, file)) {

slen = strlen(menu\_items[i].str[0]);

// удаляем предпоследний символ - '\n'

menu\_items[i].str[0][slen > 1 ? slen - 1 : 0] = '\0';

} else {

menu\_items[i].str[0][0] = '\0';

}

if(++i == max\_count) {

break; // достигнут лимит строк

}

}

fclose(file);

item\_count = i;

if(pcount)

\*pcount = item\_count;

return menu\_items;

}

// Отображение упрощённого вертикального меню в один столбец

int ShowMenu(HANDLE hStdOut, ITEM\_DEF\* menu\_items, int item\_count, const char\* title, SMALL\_RECT rect, int flags, int user\_tag, ExecuteHotketCB f1CB) {

int i;

MENU menu;

// количество ячеек меню (столбцов)

const int column\_count = 1;

char\*\* headers = NULL;

int border = 1;

if(title) {

headers = (char\*\*)malloc(column\_count \* sizeof(char\*));

memset(headers, 0x00, column\_count \* sizeof(char\*));

headers[0] = (char\*)malloc(MAX\_MENU\_HDR \* sizeof(char));

if(!headers[0]) {

free(menu\_items);

return -1;

}

strcpy(headers[0], title);

CharToOemA(headers[0], headers[0]);

}

menu\_init(&menu, ptable, hStdOut, menu\_items, item\_count, column\_count,

MENU\_ORIENT\_VERT, &rect, 1, headers);

menu.user\_tag = user\_tag;

menu\_active\_color(&menu, BACKGROUND\_INTENSITY | BACKGROUND\_BLUE | BACKGROUND\_GREEN);

menu\_inactive\_color(&menu, BACKGROUND\_BLUE | BACKGROUND\_GREEN);

menu\_add\_hotkey(&menu, KEY\_ESC, DefaultESC);

menu\_add\_hotkey(&menu, KEY\_F1, f1CB);

menu\_draw(&menu, flags);

menu\_clear(&menu);

for(i = 0; i < item\_count; ++i)

if(menu\_items[i].str[0])

free(menu\_items[i].str[0]);

free(menu\_items);

if(headers) {

for(i = 0; i < column\_count; ++i)

if(headers[i])

free(headers[i]);

free(headers);

}

return 0;

}

// Отображение текста справки из файла на всплывающем окне

int HelpFromFile(HANDLE hStdOut, const char\* file\_name, const char\* title, SMALL\_RECT rect) {

int i;

//int menu\_width = rect.Right - rect.Left + 1;

int menu\_height = rect.Bottom - rect.Top + 1;

int item\_count = 0;

// 4 = заголовок + 3 гориз. линиии

ITEM\_DEF\* menu\_items = MenuItemsFromFile(file\_name, menu\_height - 4, 256, &item\_count);

if(!menu\_items) {

return -1;

}

return ShowMenu(hStdOut, menu\_items, item\_count, title, rect, MENU\_FLAG\_WND | MENU\_FLAG\_ITEMS | MENU\_HOTKEYS, MENU\_TAG\_HELP\_GLOBAL, DefaultESC);

}

// Загрузка тестовых данных в словарь

void LoadInitialData() {

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("hello", "noun", "привет")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("world", "noun", "мир")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("go", "verb", "идти")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("here", "adverb", "здесь")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("run", "noun", "бежать")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("smart", "adjective", "умный")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("those", "pronoun", "те")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("between", "preposition", "между")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("under", "preposition", "под")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("star", "noun", "звезда")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("label", "noun", "метка")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("recall", "verb", "вспомнить")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("nice", "adjective", "красивый")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("cat", "noun", "кот")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("below", "adverb", "внизу")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("snake", "noun", "змея")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("sit", "verb", "сидеть")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("language", "noun", "язык")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("he", "pronoun", "он")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("she", "pronoun", "она")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("into", "preposition", "в")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("yours", "pronoun", "твой")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("today", "adverb", "сегодня")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("strong", "adjective", "сильный")))

return -1;

if(-1 == list1\_push\_back(&dict, dict\_entry\_new("human", "noun", "человек")))

return -1;

if(-1 == dict\_save(dict\_file\_name))

return -1;

list1\_clear(&dict);

getchar();

}

**menu.h**

#ifndef menuH

#define menuH

//--------------------------------------------------------

#include "list.h"

//--------------------------------------------------------

struct MENU;

#define MENU\_ORIENT\_HORZ 0

#define MENU\_ORIENT\_VERT 1

#define MAX\_MENU\_HDR 256 // максимальная длина заголовка

// Этапы частичной отрисовки меню

#define MENU\_FLAG\_WND 0x1 // отрисовка фонового окна меню

#define MENU\_FLAG\_ITEMS 0x2 // отрисовка элементов меню

#define MENU\_DRAW\_SEL 0x4 // выделение текущего элемента

#define MENU\_NAVIGATOR 0x8 // навигация по умолчанию (стрелки и Enter)

#define MENU\_HOTKEYS 0x10 // обработка горячих клавиш

// полноценный запуск окна меню с циклом сообщений

#define MENU\_FULL\_FLAGS 0x1F

// Максимальное число ячеек в одном пункте меню (минимум одна)

#define MAX\_CELLS 50

// Направление перемещение курсора

#define MENU\_CURR\_FWD 0 // вперёд

#define MENU\_CURR\_REV 1 // назад

struct ITEM;

// Указатели на функции void f(void) - они будут выполнять пункты меню

typedef int(\*ExecuteHotketCB)(struct MENU\*);

typedef int(\*ExecuteCurrentCB)(struct MENU\*, struct ITEM\*);

typedef void(\*CurrentChangedCB)(struct MENU\*, int direction, int wrap);

// Структура определения элемента меню

typedef struct {

char\* str[MAX\_CELLS]; // Наименование пункта меню (несколько ячеек)

ExecuteCurrentCB cb; // Функция, привязанная к пункту меню

} ITEM\_DEF;

// Структура для хранения элемента меню

typedef struct ITEM {

int x, y; // Столбец и строка консоли

int index;

char\* str; // Наименование пункта меню (несколько ячеек)

ExecuteCurrentCB cb; // Функция, привязанная к пункту меню

} ITEM;

// Область окна меню

typedef struct {

SMALL\_RECT rect; // координаты углов консольного окна

int M, N; // размер матрицы символов

char\*\* m; // матрица символов

} MENU\_WND;

// Горячая клавиша

typedef struct {

int code;

ExecuteHotketCB cb; // обработчик

} HOT\_KEY;

// Структура для всего меню

typedef struct MENU {

// Указатель на родительское меню (не вызывающее, а то, которое перекрываем)

struct MENU\* parent;

// Глобальные переменные, используемые в функциях меню

HANDLE hStdOut;// = INVALID\_HANDLE\_VALUE; // дескриптор консольного окна

///SMALL\_RECT consolRect; // координаты углов консольного окна

// COORD curspos;//={0,1}; // координаты текстового курсора

WORD workWindowAttributes;// = 158;// атрибуты рабочей области

WORD inactiveItemAttributes;// = 31; // атрибуты цвета неактивного пункта меню

WORD activeItemAttributes;// = 160; // атрибуты цвета активного пункта меню

MENU\_WND wnd;

int cb\_retcode;

int last\_key;

LIST1 hk\_list;

CurrentChangedCB changed\_cb;

char\* hdr;

LIST1 items;

int cell\_count; // количество ячеек

int cell\_width; // ширина ячейки

int cell\_start[MAX\_CELLS]; // индексы начала ячеек внутри строки каждого пункта

int item\_width; // количество символов в строке каждого пункта

int orient; // ориентация (MENU\_ORIENT\_HORZ/MENU\_ORIENT\_VERT)

int has\_headers; // 0 - нет заголовка, 1 - есть заголовок

int has\_border; // 0 - нет рамки, 1 - есть рамка

// индекс строки с нижней границей (для правильной отрисовки)

int border\_bottom\_index;

// отступы слева и сверху, с учётом наличия рамки и заголовка

int left\_pad;

int top\_pad;

// пользовательская метка

int user\_tag;

} MENU;

// Инициализация полей экземпляра меню

int menu\_init(MENU\* menu, MENU\* parent, HANDLE hstdout, ITEM\_DEF\* item\_defs, int item\_count, int cell\_count,

int orient, const SMALL\_RECT\* prect, int border, char\* headers[]);

// Добавить обработчик горячей клавиши

int menu\_add\_hotkey(MENU\* menu, int code, ExecuteHotketCB cb);

// Очистка полей экземпляра меню

void menu\_clear(MENU\* menu);

// Прорисовка фона и рамки

void menu\_fill\_wnd(MENU\* menu, int items\_count);

// Прорисовка меню

int menu\_draw(MENU\* menu, int flags);

// Установка цветовых атрибутов активного пункта

void menu\_active\_color(MENU\* menu, WORD attr);

// Установка цветовых атрибутов неактивного пункта

void menu\_inactive\_color(MENU\* menu, WORD attr);

// Выделить предыдущий пункт меню

void menu\_prev(MENU\* menu);

// Выделить следующий пункт меню

void menu\_next(MENU\* menu);

// Перевод курсора в точку x, y

void menu\_gotoxy(MENU\* menu, int x, int y);

// Выделить пункт меню

void itemMenu(MENU\* menu, int activate);

// Запомнить положение курсора

void saveCursorPosition(MENU\* menu);

// Очистка окна

void menu\_cls(MENU\* menu);

// скрыть/показать курсор

void showCursor(MENU\* menu, int visible);

// Удалить текущий элемент

void menu\_del\_curr(MENU\* menu);

#endif

**menu.c**

//--------------------------------------------------------

#include <windows.h>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#pragma hdrstop

#include "menu.h"

#include "dict.h"

#include "codes.h"

//--------------------------------------------------------

#pragma package(smart\_init)

#define MAX(a, b) (((a) > (b)) ? (a) : (b))

#define MENU\_WHITESPACE ' '

int menu\_calc\_item\_pos(void\* data, int index, void\* ptr);

// Отрисовка пункта меню (callback для элемента списка)

int menu\_draw\_item(void\* data, int index, void\* ptr);

// Проверить код, возвращённый обработчиком

int check\_cb\_retcode(MENU\* menu);

// Очистка одного элемента меню

void menu\_item\_release(void\* ptr);

// Обработка горячей клавиши (если была нажата)

int menu\_proc\_hotkey(void\* data, int index, void\* ptr);

void menu\_active\_color(MENU\* menu, WORD attr) {

menu->activeItemAttributes = attr;

}

void menu\_inactive\_color(MENU\* menu, WORD attr) {

menu->inactiveItemAttributes = attr;

menu->workWindowAttributes = attr;

}

void menu\_fill\_wnd(MENU\* menu, int items\_count) {

int i, border\_bottom\_index;

if(items\_count == 0)

items\_count = menu->items.count;

border\_bottom\_index = menu->border\_bottom\_index + items\_count;

// проход по строкам окна

for(i = 0; i < menu->wnd.M; i++) {

//menu->wnd.m[i] = (char\*)malloc((menu->wnd.N + 1) \* sizeof(char));

memset(menu->wnd.m[i], MENU\_WHITESPACE, menu->wnd.N \* sizeof(char));

menu->wnd.m[i][menu->wnd.N] = '\0';

if(menu->has\_border) {

// отрисовка вертикальной границы слева и справа

if(i < border\_bottom\_index) {

menu->wnd.m[i][0] = CHAR\_BORDER\_VERT;

menu->wnd.m[i][menu->wnd.N - 1] = CHAR\_BORDER\_VERT;

}

// отрисовка горизонтальной линии отделения заголовка и её стыковочных линий слева/справа

if(menu->hdr != NULL && 2 == i) {

memset(menu->wnd.m[i], CHAR\_BORDER\_HORZ, menu->wnd.N \* sizeof(char));

menu->wnd.m[i][0] = CHAR\_BORDER\_LEFT\_JOINT;

menu->wnd.m[i][menu->wnd.N - 1] = CHAR\_BORDER\_RIGHT\_JOINT;

}

if(0 == i || border\_bottom\_index == i) {

// отрисовка верхней и нижней горизонтальных линий, а также 4 внешних углов

memset(menu->wnd.m[i], CHAR\_BORDER\_HORZ, menu->wnd.N \* sizeof(char));

if(0 == i) { // первая строка

menu->wnd.m[i][0] = CHAR\_BORDER\_LEFT\_TOP;

menu->wnd.m[i][menu->wnd.N - 1] = CHAR\_BORDER\_RIGHT\_TOP;

}

if(border\_bottom\_index == i) { // последняя строка

menu->wnd.m[i][0] = CHAR\_BORDER\_LEFT\_BOTTOM;

menu->wnd.m[i][menu->wnd.N - 1] = CHAR\_BORDER\_RIGHT\_BOTTOM;

}

}

}

}

if(menu->has\_border) {

for (i = 0; i < menu->cell\_count; i++) {

if(menu->has\_headers) {

if(i < menu->cell\_count - 1) {

// отрисовка вертикальных линий в заголовке

menu->wnd.m[0][menu->cell\_start[i] + menu->cell\_width - 0] = CHAR\_BORDER\_TOP\_JOINT;

menu->wnd.m[2][menu->cell\_start[i] + menu->cell\_width - 0] = CHAR\_BORDER\_BOTTOM\_JOINT;

}

}

if(i < menu->cell\_count - 1) {

int top\_shift = menu->top\_pad - 1;

if(menu->has\_headers) {

menu->wnd.m[top\_shift][menu->cell\_start[i] + menu->cell\_width - 0] = CHAR\_BORDER\_CROSS\_JOINT;

} else {

menu->wnd.m[top\_shift][menu->cell\_start[i] + menu->cell\_width - 0] = CHAR\_BORDER\_TOP\_JOINT;

}

menu->wnd.m[border\_bottom\_index][menu->cell\_start[i] + menu->cell\_width - 0] = CHAR\_BORDER\_BOTTOM\_JOINT;

}

} // if(menu->has\_headers)

} // if(menu->has\_border)

}

int menu\_init(MENU\* menu, MENU\* parent, HANDLE hstdout, ITEM\_DEF\* item\_defs, int item\_count, int cell\_count,

int orient, const SMALL\_RECT\* prect, int border, char\* headers[])

{

int left\_indent = 1;

int i, j, len;

if(NULL == menu || NULL == item\_defs || INVALID\_HANDLE\_VALUE == hstdout)

return -1;

memset(menu, 0x00, sizeof(MENU));

menu->parent = parent;

menu->orient = orient;

list1\_init(&menu->hk\_list);

// Сохраняем дескриптор консольного вывода

menu->hStdOut = hstdout;

// Сохраняем полученные размеры окна

if(prect != NULL ) {

menu->wnd.rect.Left = prect->Left;

menu->wnd.rect.Top = prect->Top;

menu->wnd.rect.Right = prect->Right;

menu->wnd.rect.Bottom = prect->Bottom;

} else {

// Получаем размеры консоли

CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo;

GetConsoleScreenBufferInfo(menu->hStdOut, &csbInfo);

menu->wnd.rect.Left = csbInfo.srWindow.Left;

menu->wnd.rect.Top = csbInfo.srWindow.Top;

menu->wnd.rect.Right = csbInfo.srWindow.Right;

menu->wnd.rect.Bottom = csbInfo.srWindow.Bottom;

}

menu->wnd.M = menu->wnd.rect.Bottom - menu->wnd.rect.Top + 1;

menu->wnd.N = menu->wnd.rect.Right - menu->wnd.rect.Left + 1;

menu->wnd.m = (char\*\*)malloc((menu->wnd.M) \* sizeof(char\*));

for(i = 0; i < menu->wnd.M; i++) {

menu->wnd.m[i] = (char\*)malloc((menu->wnd.N + 1) \* sizeof(char));

memset(menu->wnd.m[i], MENU\_WHITESPACE, menu->wnd.N \* sizeof(char));

menu->wnd.m[i][menu->wnd.N] = '\0';

}

//

menu->has\_border = border;

menu->has\_headers = (int)headers;

menu->cell\_count = cell\_count;

menu->left\_pad = 0;

menu->top\_pad = 0;

if(MENU\_ORIENT\_VERT == orient) { //TODO MENU\_ORIENT\_HORZ

menu->item\_width = (menu->wnd.rect.Right - menu->wnd.rect.Left + 1) - (menu->has\_border ? 2 : 0);

menu->cell\_width = menu->item\_width / cell\_count;

for (j = 0; j < cell\_count; j++) {

menu->cell\_start[j] = j \* menu->cell\_width;

}

if(menu->has\_headers) {

menu->top\_pad += 2;

menu->border\_bottom\_index += 2;

}

if(menu->has\_border) {

menu->left\_pad += 1;

menu->top\_pad += 1;

menu->border\_bottom\_index += 1;

}

// не можем отобразить не помещающиеся в окно элементы

if(item\_count > menu->wnd.M - menu->border\_bottom\_index - 1) {

item\_count = menu->wnd.M - menu->border\_bottom\_index - 1;

}

}

menu\_fill\_wnd(menu, item\_count);

// координаты углов консоли

// menu->curspos.X = menu->wnd.rect.Left;///0;

// menu->curspos.Y = menu->wnd.rect.Top;///1;

menu->workWindowAttributes = //0x9E;

BACKGROUND\_RED | BACKGROUND\_GREEN | FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE;

menu->inactiveItemAttributes = 0x1F;

menu->activeItemAttributes = 0xA0;

// Инициализация элементов меню

list1\_init(&menu->items);

menu->items.item\_free = menu\_item\_release;

//int next = 0

switch(orient) {

case MENU\_ORIENT\_HORZ:

menu->item\_width = (menu->wnd.rect.Right - menu->wnd.rect.Left + 1) / item\_count;

menu->cell\_count = 1; // cell\_count;

//next = menu->wnd.rect.Left;

for(i = 0; i < item\_count; ++i) {

int width;

ITEM\* item = (ITEM\*)malloc(sizeof(ITEM));

memset(item, 0x00, sizeof(ITEM));

item->cb = item\_defs[i].cb;

//item->x = next + menu->left\_pad; // menu->wnd.rect.Left + index \* menu->item\_width + menu->left\_pad;

//item->y = menu->wnd.rect.Top + menu->top\_pad;

if(item\_defs[i].str[0]) {

len = strlen(item\_defs[i].str[0]);

width = MAX(len, menu->item\_width);

item->str = (char\*)malloc((width + 1) \* sizeof(char));

memset(item->str, MENU\_WHITESPACE, width);

CharToOemA(item\_defs[i].str[0], item\_defs[i].str[0]);

if(len < menu->item\_width) {

int indent = (menu->item\_width - len) / 2; // по центру

memcpy(item->str + indent, item\_defs[i].str[0], len);

} else {

memcpy(item->str, item\_defs[i].str[0], menu->item\_width - 1);

item->str[menu->item\_width - 1] = '\_';

}

item->str[width] = '\0';

//next += menu->item\_width;

}

item->index = i;

list1\_push\_back(&menu->items, item);

} // for(item)

break;

case MENU\_ORIENT\_VERT:

if(menu->has\_headers) {

if(menu->has\_border) {

// отрисовка горизонтальной линии отделения заголовка и её стыковочных линий слева/справа

memset(menu->wnd.m[2], CHAR\_BORDER\_HORZ, menu->wnd.N \* sizeof(char));

menu->wnd.m[2][0] = CHAR\_BORDER\_LEFT\_JOINT;

menu->wnd.m[2][menu->wnd.N - 1] = CHAR\_BORDER\_RIGHT\_JOINT;

}

menu->hdr = (char\*)malloc((menu->item\_width + 1) \* sizeof(char));

memset(menu->hdr, MENU\_WHITESPACE, menu->item\_width);

menu->hdr[menu->item\_width] = '\0';

for (j = 0; j < cell\_count; j++) {

if(headers[j]) {

len = strlen(headers[j]);

//width = MAX(len, menu->cell\_width);

//CharToOemA(headers[j], headers[j]);

if(len + left\_indent < menu->cell\_width) {

memcpy(menu->hdr + menu->cell\_start[j] + left\_indent, headers[j], len);

} else {

memcpy(menu->hdr + menu->cell\_start[j] + left\_indent, headers[j], menu->cell\_width - 1 - left\_indent);

menu->hdr[menu->cell\_start[j] + menu->cell\_width - 1 - left\_indent] = '\_';

}

if(menu->has\_border && j < cell\_count - 1) {

// отрисовка вертикальных линий в заголовке

menu->wnd.m[0][menu->cell\_start[j] + menu->cell\_width - 0] = CHAR\_BORDER\_TOP\_JOINT;

menu->hdr[menu->cell\_start[j] + menu->cell\_width - 1] = CHAR\_BORDER\_VERT;

menu->wnd.m[2][menu->cell\_start[j] + menu->cell\_width - 0] = CHAR\_BORDER\_BOTTOM\_JOINT;

}

}

} // for(cell)

} // if(has\_headers)

//next = menu->wnd.rect.Top;

for(i = 0; i < item\_count; ++i) {

ITEM\* item = (ITEM\*)malloc(sizeof(ITEM));

memset(item, 0x00, sizeof(ITEM));

item->cb = item\_defs[i].cb;

//item->x = menu->wnd.rect.Left + menu->left\_pad;

//item->y = next + menu->top\_pad; // menu->wnd.rect.Top + menu->top\_pad + index

item->str = (char\*)malloc((menu->item\_width + 1) \* sizeof(char));

memset(item->str, MENU\_WHITESPACE, menu->item\_width);

item->str[menu->item\_width] = '\0';

for (j = 0; j < cell\_count; j++) {

if(item\_defs[i].str[j]) {

len = strlen(item\_defs[i].str[j]);

//width = MAX(len, menu->cell\_width);

// нехорошо менять каждый раз оригиналы CharToOemA(item\_defs[i].str[j], item\_defs[i].str[j]);

if(len + left\_indent < menu->cell\_width) {

memcpy(item->str + menu->cell\_start[j] + left\_indent, item\_defs[i].str[j], len);

} else {

memcpy(item->str + menu->cell\_start[j] + left\_indent, item\_defs[i].str[j], menu->cell\_width - 1 - left\_indent);

item->str[menu->cell\_start[j] + menu->cell\_width - 1 - left\_indent] = '\_';

}

if(menu->has\_border && j < cell\_count - 1) {

int top\_shift;

if(menu->has\_headers) {

top\_shift = 2;

menu->wnd.m[top\_shift][menu->cell\_start[j] + menu->cell\_width - 0] = CHAR\_BORDER\_CROSS\_JOINT;

} else {

top\_shift = 0;

menu->wnd.m[top\_shift][menu->cell\_start[j] + menu->cell\_width - 0] = CHAR\_BORDER\_TOP\_JOINT;

}

item->str[menu->cell\_start[j] + menu->cell\_width - 1] = CHAR\_BORDER\_VERT;

menu->wnd.m[i + top\_shift + 2][menu->cell\_start[j] + menu->cell\_width - 0] = CHAR\_BORDER\_BOTTOM\_JOINT;

}

}

} // for(cell)

//портит псевдографику CharToOemA(item->str, item->str);

item->index = i;

list1\_push\_back(&menu->items, item);

//next++;

} // for(item)

break;

}

list1\_for\_each(&menu->items, menu\_calc\_item\_pos, menu);

return 0;

}

void menu\_item\_release(void\* ptr) {

ITEM\* item = (ITEM\*)ptr;

if(item->str)

free(item->str);

free(item);

}

void menu\_del\_curr(MENU\* menu) {

list1\_erase\_current(&menu->items);

}

void menu\_clear(MENU\* menu) {

int i;

list1\_clear(&menu->items);

if(menu->hdr) {

free(menu->hdr);

menu->hdr = NULL;

}

for(i = 0; i < menu->wnd.M; i++)

if(menu->wnd.m[i]) {

free(menu->wnd.m[i]);

menu->wnd.m[i] = NULL;

}

free(menu->wnd.m);

menu->wnd.m = NULL;

list1\_clear(&menu->hk\_list);

menu->hStdOut = INVALID\_HANDLE\_VALUE;

}

void menu\_prev(MENU\* menu) {

int wrap = 1;

itemMenu(menu, 0); // сделать неактивным пункт меню

list1\_curr\_rev(&menu->items, wrap);

itemMenu(menu, 1); // выделить активный пункт меню

showCursor(menu, 0);

if(menu->changed\_cb) {

menu->changed\_cb(menu, MENU\_CURR\_REV, wrap);

}

}

void menu\_next(MENU\* menu) {

int wrap = 1;

itemMenu(menu, 0); // сделать неактивным пункт меню

list1\_curr\_fwd(&menu->items, wrap);

itemMenu(menu, 1); // выделить активный пункт меню

showCursor(menu, 0);

if(menu->changed\_cb) {

menu->changed\_cb(menu, MENU\_CURR\_FWD, wrap);

}

}

int menu\_calc\_item\_pos(void\* data, int index, void\* ptr) {

ITEM\* item = (ITEM\*)data;

MENU\* menu = (MENU\*)ptr;

item->x = menu->wnd.rect.Left + menu->left\_pad;

item->y = menu->wnd.rect.Top + menu->top\_pad;

switch(menu->orient) {

case MENU\_ORIENT\_HORZ:

item->x += index \* menu->item\_width;

break;

case MENU\_ORIENT\_VERT:

item->y += index;

break;

}

return 1; // продолжить итерации по остальным элементам

}

int menu\_draw\_item(void\* data, int index, void\* ptr) {

ITEM\* item = (ITEM\*)data;

MENU\* menu = (MENU\*)ptr;

menu\_calc\_item\_pos(data, index, ptr);

menu\_gotoxy(menu, item->x, item->y);

printf(item->str);

return 1; // продолжить итерации по остальным элементам

}

int menu\_draw(MENU\* menu, int flags) {

int result = 0, i, run = 1;

if(flags & MENU\_FLAG\_WND) {

// Устанавливаем цветовые параметры текста

SetConsoleTextAttribute(menu->hStdOut, menu->workWindowAttributes);

//system("CLS"); // установка атрибутов цвета рабочей области

menu\_cls(menu);

}

if(flags & MENU\_FLAG\_ITEMS) {

// Номер текущего пункта меню

SetConsoleTextAttribute(menu->hStdOut, menu->inactiveItemAttributes);

// рисуем заголовок

if(menu->hdr) {

int top\_pad = menu->has\_border ? 1 : 0;

menu\_gotoxy(menu, menu->wnd.rect.Left + menu->left\_pad, menu->wnd.rect.Top + top\_pad);

printf(menu->hdr);

}

// рисуем меню

list1\_for\_each(&menu->items, menu\_draw\_item, menu);

fflush(stdout);

}

if(flags & MENU\_DRAW\_SEL) {

itemMenu(menu, 1); // выделить пункт меню

}

//fflush(stdin); // очистить буфер клавиатуры

if(!(flags & (MENU\_NAVIGATOR | MENU\_HOTKEYS)))

return 0; // не запускаем обработку сообщений

while (run) {

if (kbhit()) {

menu->cb\_retcode = 0;

menu->last\_key = \_getch();

if(flags & MENU\_NAVIGATOR) {

switch (menu->last\_key) {

case KEY\_ARROW\_UP:

if(MENU\_ORIENT\_VERT == menu->orient)

menu\_prev(menu);

case KEY\_ARROW\_LEFT:

if(MENU\_ORIENT\_HORZ == menu->orient)

menu\_prev(menu);

break;

case KEY\_ARROW\_RIGHT:

if(MENU\_ORIENT\_HORZ == menu->orient)

menu\_next(menu);

break;

case KEY\_ARROW\_DOWN:

if(MENU\_ORIENT\_VERT == menu->orient)

menu\_next(menu);

break;

case KEY\_ENTER:

// Возвращаем курсор из строки меню в прежнюю позицию

///menu\_gotoxy(menu, menu->curspos.X, menu->curspos.Y);

// Установить цвет рабочих сообщений

///SetConsoleTextAttribute(menu->hStdOut, menu->workWindowAttributes);

///showCursor(menu, 1);

// Вызываем обработчик пункта меню

{

ITEM\* item = (ITEM\*)list1\_curr(&menu->items);

menu->cb\_retcode = item->cb(menu, item);

}

run = check\_cb\_retcode(menu);

break;

} // switch(iKey)

}

if(flags & MENU\_HOTKEYS) {

// обработка горячих клавиш

list1\_for\_each(&menu->hk\_list, menu\_proc\_hotkey, menu);

run = check\_cb\_retcode(menu);

}

} // if(kbhit())

} // while(run)

if(menu->parent) {

menu\_draw(menu->parent, MENU\_FLAG\_WND | MENU\_FLAG\_ITEMS | MENU\_DRAW\_SEL);

}

return result;

}

void itemMenu(MENU\* menu, int activate)

{

ITEM\* item = (ITEM\*)list1\_curr(&menu->items);

WORD itemAttributes;

if(!item)

return;

if (activate)

itemAttributes = menu->activeItemAttributes;

else

itemAttributes = menu->inactiveItemAttributes;

menu\_gotoxy(menu, item->x, item->y);

SetConsoleTextAttribute(menu->hStdOut, itemAttributes);

printf(item->str);

}

void menu\_cls(MENU\* menu)

{

int i, y;

SetConsoleTextAttribute(menu->hStdOut, menu->workWindowAttributes);

for (i = 0, y = menu->wnd.rect.Top; i < menu->wnd.M; i++, y++) {

menu\_gotoxy(menu, menu->wnd.rect.Left, y);

printf(menu->wnd.m[i]);

}

}

int menu\_add\_hotkey(MENU\* menu, int code, ExecuteHotketCB cb) {

HOT\_KEY\* hk = (HOT\_KEY\*)malloc(sizeof(HOT\_KEY));

memset(hk, 0x00, sizeof(HOT\_KEY));

hk->code = code;

hk->cb = cb;

list1\_push\_front(&menu->hk\_list, hk);

return menu->hk\_list.count;

}

int menu\_proc\_hotkey(void\* data, int index, void\* ptr) {

HOT\_KEY\* hk = (HOT\_KEY\*)data;

MENU\* menu = (MENU\*)ptr;

if(menu->last\_key == hk->code) {

menu->cb\_retcode = hk->cb(menu);

return 0; // не проверять остальные обработчики

}

return 1; // продолжить итерации по остальным элементам

}

void menu\_gotoxy(MENU\* menu, int x, int y)

{

COORD cursorPos;

cursorPos.X = x;

cursorPos.Y = y;

SetConsoleCursorPosition(menu->hStdOut, cursorPos);

///SetConsoleCursorPosition(hStdOut, {x,y});

}

void saveCursorPosition(MENU\* menu)

{

CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo;// информация о консольном окне в структуре csbInfo

GetConsoleScreenBufferInfo(menu->hStdOut, &csbInfo);

/// menu->curspos = csbInfo.dwCursorPosition;

}

void showCursor(MENU\* menu, int visible)

{

CONSOLE\_CURSOR\_INFO ccInfo;

ccInfo.bVisible = visible;

ccInfo.dwSize = 20;

SetConsoleCursorInfo(menu->hStdOut, &ccInfo);

}

int check\_cb\_retcode(MENU\* menu) {

if(-1 == menu->cb\_retcode) {

///menu\_gotoxy(menu, 0, 0);

///menu\_cls(menu, WholeWindow);

itemMenu(menu, 0); // сделать неактивным пункт меню

return 0;

} else if(0 == menu->cb\_retcode) {

saveCursorPosition(menu);

}

// очистить буфер клавиатуры

fflush(stdin);

// курсор в текущий пункт меню

///menu\_gotoxy(menu, menu->items[menu->current].x, menu->items[menu->current].y);

// спрятать курсор

showCursor(menu, 0);

return 1;

}

**list.h**

#ifndef \_\_LIST1\_H\_\_

#define \_\_LIST1\_H\_\_

struct LIST1\_ITEM;

// структура элемента односвязного списка

typedef struct LIST1\_ITEM {

struct LIST1\_ITEM\* next;

void\* data;

} LIST1\_ITEM;

// односвязный список

typedef void (\*LIST\_ITEM\_FREE)(void\* item);

typedef void (\*LIST\_ITEM\_UPD)(void\* item);

// если вернёт 0, обработка прекратится на этой итерации

typedef int (\*LIST\_ITEM\_PROC)(void\* item, int index, void\* param);

typedef int (\*LIST\_ITEM\_FIND)(void\* item, void\* param);

typedef int (\*LIST\_ITEM\_COMP)(void\* item1, void\* item2, void\* param);

// структура односвязного списка

typedef struct LIST1 {

struct LIST1\_ITEM \*head, \*tail, \*curr;

LIST\_ITEM\_FREE item\_free;

int count;

} LIST1;

//

LIST1\* list1\_new();

//

int list1\_delete(LIST1\* list);

//

int list1\_init(LIST1\* list);

//

int list1\_clear(LIST1\* list);

//

int list1\_push\_front(LIST1\* list, void\* data);

//

int list1\_push\_back(LIST1\* list, void\* item);

//

int list1\_for\_each(LIST1\* list, LIST\_ITEM\_PROC find, void\* param);

// Прокрутить текущий указатель вперёд

void\* list1\_curr\_fwd(LIST1\* list, int wrap);

// Прокрутить текущий указатель назад

void\* list1\_curr\_rev(LIST1\* list, int wrap);

//

void\* list1\_front(LIST1\* list);

//

void\* list1\_back(LIST1\* list);

//

void\* list1\_curr(LIST1\* list);

// Удалить элемент списка

// возвращает текущий элемент (возможно, новый)

void\* list1\_erase(LIST1\* list, LIST\_ITEM\_FIND cb, void\* param);

// Удалить текущий элемент списка

// возвращает новый текущий элемент

void\* list1\_erase\_current(LIST1\* list);

//

int list1\_get\_current\_index(LIST1\* list);

//

int list1\_set\_current\_index(LIST1\* list, int index);

// поиск элемента (установит текущий и вернёт его порядковый номер)

int list1\_search(LIST1\* list, LIST\_ITEM\_FIND check, void\* param);

// сортировка списка

void list1\_sort(LIST1\* list, LIST\_ITEM\_COMP compare, void\* param);

#endif

**list.c**

#include "list.h"

#include <stdlib.h>

#include <string.h>

static LIST1\_ITEM\* create\_item(void\* data);

static void delete\_item(LIST1\* list, LIST1\_ITEM\* item);

static void merge(LIST1\_ITEM \*a, LIST1\_ITEM \*b, LIST1\_ITEM \*\*c, LIST\_ITEM\_COMP compare, void\* param);

static void split(LIST1\_ITEM \*src, LIST1\_ITEM \*\*low, LIST1\_ITEM \*\*high);

static void mergeSort(LIST1\_ITEM \*\*head, LIST\_ITEM\_COMP compare, void\* param);

LIST1\* list1\_new() {

LIST1\* list = (LIST1\*)malloc(sizeof(LIST1));

if(-1 == list1\_init(list))

return NULL;

return list;

}

int list1\_delete(LIST1\* list) {

if(!list1\_clear(list))

return -1;

free(list);

return 0;

}

int list1\_init(LIST1\* list) {

if(!list)

return -1;

memset(list, 0x00, sizeof(LIST1));

list->item\_free = free;

return 0;

}

int list1\_clear(LIST1\* list) {

LIST1\_ITEM\* curr;

if(!list)

return -1;

curr = list->head;

while(curr) {

LIST1\_ITEM\* next = curr->next;

delete\_item(list, curr);

curr = next;

}

memset(list, 0x00, sizeof(LIST1));

return 0;

}

int list1\_push\_front(LIST1\* list, void\* data) {

LIST1\_ITEM\* item = create\_item(data);

if(!item)

return -1;

if(list->head) {

item->next = list->head;

} else { // добавляется первый элемент

list->curr = item;

list->tail = item;

}

list->head = item;

list->count++;

return 0;

}

int list1\_push\_back(LIST1\* list, void\* data) {

LIST1\_ITEM\* item = create\_item(data);

if(!item)

return -1;

if(list->head) {

list->tail->next = item;

} else { // добавляется первый элемент

list->head = item;

list->curr = item;

}

list->tail = item;

list->count++;

return 0;

}

int list1\_for\_each(LIST1\* list, LIST\_ITEM\_PROC cb, void\* param) {

LIST1\_ITEM\* curr;

int i = 0;

if(!cb)

return -1;

curr = list->head;

while(curr) {

cb(curr->data, i++, param);

curr = curr->next;

}

return 0;

}

// Прокрутить текущий указатель вперёд

void\* list1\_curr\_fwd(LIST1\* list, int wrap) {

if(!list || !list->curr)

return NULL;

list->curr = list->curr->next;

if(!list->curr && wrap)

list->curr = list->head;

return list->curr->data;

}

// Прокрутить текущий указатель назад

void\* list1\_curr\_rev(LIST1\* list, int wrap) {

LIST1\_ITEM \*curr, \*prev;

if(!list || !list->head)

return NULL;

curr = list->head;

prev = list->head;

while(curr) {

if(curr == list->curr) {

if(curr == list->head) {

if(wrap) {

list->curr = list->tail;

}

} else {

list->curr = prev;

}

break;

}

prev = curr;

curr = curr->next;

}

return list->curr->data;

}

void\* list1\_front(LIST1\* list) {

if(!list->head)

return NULL;

return list->head->data;

}

void\* list1\_back(LIST1\* list) {

if(!list->tail)

return NULL;

return list->tail->data;

}

void\* list1\_curr(LIST1\* list) {

if(!list->curr)

return NULL;

return list->curr->data;

}

void\* list1\_erase(LIST1\* list, LIST\_ITEM\_FIND cb, void\* param) {

LIST1\_ITEM \*curr, \*prev;

if(!list || !list->head)

return NULL;

curr = list->head;

prev = list->head;

while(curr) {

LIST1\_ITEM\* next = curr->next;

if(cb(curr->data, param)) {

// сместить текущий указатель при необходимости

if(curr == list->curr) {

list->curr = prev;

}

// удалить найденный элемент вместе с данными

delete\_item(list, curr);

} else {

prev = curr;

}

curr = next;

}

if(!list->curr)

return NULL;

return list->curr->data;

}

void\* list1\_erase\_current(LIST1\* list) {

LIST1\_ITEM \*curr, \*prev;

if(!list || !list->head)

return NULL;

curr = list->head;

prev = list->head;

while(curr) {

LIST1\_ITEM \*next = curr->next;

if(curr == list->curr) {

// определить новый текущий элемент

prev->next = next;

list->curr = next;

if(curr == list->head) {

list->head = next; // если это последний элемент, next равен 0

}

if(curr == list->tail) {

if(list->head == NULL)

list->tail = NULL; // если это последний элемент, list->head тут уже равен 0

else

list->tail = prev;

}

if(!list->curr && list->head) {

list->curr = prev;

}

// удалить старый текущий элемент вместе с данными

delete\_item(list, curr);

break;

}

prev = curr;

curr = next;

}

if(!list->curr)

return NULL;

return list->curr->data;

}

int list1\_get\_current\_index(LIST1\* list) {

int i = 0;

LIST1\_ITEM\* curr = list->head;

while(curr) {

if(curr == list->curr)

return i;

i++;

curr = curr->next;

}

return 0;

}

int list1\_set\_current\_index(LIST1\* list, int index) {

int i = 0;

LIST1\_ITEM\* curr = list->head;

while(curr) {

if(i == index) {

list->curr = curr;

break;

}

i++;

curr = curr->next;

}

return 0;

}

int list1\_search(LIST1\* list, LIST\_ITEM\_FIND check, void\* param) {

int i = 0;

LIST1\_ITEM\* curr = list->head;

while(curr) {

if(check(curr->data, param)) {

list->curr = curr;

break;

}

i++;

curr = curr->next;

}

return i;

}

void list1\_sort(LIST1\* list, LIST\_ITEM\_COMP compare, void\* param) {

mergeSort(&list->head, compare, param);

}

// вспомогательные функции

// создать элемент списка - контейнер данных

LIST1\_ITEM\* create\_item(void\* data) {

LIST1\_ITEM\* item;

if(NULL == data)

return NULL;

item = (LIST1\_ITEM\*)malloc(sizeof(LIST1\_ITEM));

if(!item)

return NULL;

memset(item, 0x00, sizeof(LIST1\_ITEM));

item->data = data;

return item;

}

// удалить элемент списка - контейнер данных

void delete\_item(LIST1\* list, LIST1\_ITEM\* item) {

if(list->item\_free)

list->item\_free(item->data);

free(item);

list->count--;

}

void mergeSort(LIST1\_ITEM \*\*head, LIST\_ITEM\_COMP compare, void\* param) {

LIST1\_ITEM \*low = NULL;

LIST1\_ITEM \*high = NULL;

if ((\*head == NULL) || ((\*head)->next == NULL)) {

return;

}

split(\*head, &low, &high);

mergeSort(&low, compare, param);

mergeSort(&high, compare, param);

merge(low, high, head, compare, param);

}

void split(LIST1\_ITEM \*src, LIST1\_ITEM \*\*low, LIST1\_ITEM \*\*high) {

LIST1\_ITEM\* fast = NULL;

LIST1\_ITEM\* slow = NULL;

if (src == NULL || src->next == NULL) {

(\*low) = src;

(\*high) = NULL;

return;

}

slow = src;

fast = src->next;

while (fast != NULL) {

fast = fast->next;

if (fast != NULL) {

fast = fast->next;

slow = slow->next;

}

}

(\*low) = src;

(\*high) = slow->next;

slow->next = NULL;

}

void merge(LIST1\_ITEM \*a, LIST1\_ITEM \*b, LIST1\_ITEM \*\*c, LIST\_ITEM\_COMP compare, void\* param) {

LIST1\_ITEM tmp;

\*c = NULL;

if (a == NULL) {

\*c = b;

return;

}

if (b == NULL) {

\*c = a;

return;

}

if (compare(a->data, b->data, param)) { // a->value < b->value

\*c = a;

a = a->next;

} else {

\*c = b;

b = b->next;

}

tmp.next = \*c;

while (a && b) {

if (compare(a->data, b->data, param)) { // a->value < b->value

(\*c)->next = a;

a = a->next;

} else {

(\*c)->next = b;

b = b->next;

}

(\*c) = (\*c)->next;

}

if (a) {

while (a) {

(\*c)->next = a;

(\*c) = (\*c)->next;

a = a->next;

}

}

if (b) {

while (b) {

(\*c)->next = b;

(\*c) = (\*c)->next;

b = b->next;

}

}

\*c = tmp.next;

}

**dict.h**

#ifndef dictH

#define dictH

//--------------------------------------------------------

#define MAX\_DICT\_STR\_LEN 32

#define DICT\_FLD\_CNT 3

//--------------------------------------------------------

#include "list.h"

//--------------------------------------------------------

// структура записи словаря

typedef struct DICT\_ENTRY {

char\* field[DICT\_FLD\_CNT];

} DICT\_ENTRY;

//--------------------------------------------------------

struct MENU;

//

DICT\_ENTRY\* dict\_entry\_new(const char\* word\_eng, const char\* word\_part, const char\* word\_rus);

//

int dict\_load(const char\* file\_name);

//

int dict\_save(const char\* file\_name);

//

void dict\_clear();

//

void dict\_entry\_clear(DICT\_ENTRY\* entry);

//--------------------------------------------------------

extern const char\* dict\_file\_name;

extern LIST1 dict;

//--------------------------------------------------------

#endif

**dict.c**

//--------------------------------------------------------

#include <windows.h>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#pragma hdrstop

#include <sys/types.h>

#include <sys/stat.h>

#include "dict.h"

//--------------------------------------------------------

#pragma package(smart\_init)

//--------------------------------------------------------

const char\* dict\_file\_name = "dictionary.dat";

// глобальный экзепляр словаря

LIST1 dict;

// Вспомогательная функция загрузки из файла

char\* file\_load(const char\* file\_name, off\_t\* psize);

//--------------------------------------------------------

// Функции словаря

//--------------------------------------------------------

//

int dict\_load(const char\* file\_name) {

off\_t i, fld, size = 0;

char\* buf = file\_load(file\_name, &size);

if(buf) {

int field\_start = 0, nfield = 0;

DICT\_ENTRY temp\_entry;

memset(&temp\_entry, 0x00, sizeof(DICT\_ENTRY));

for(i = 0; i < size; i++) {

char ch = buf[i];

if('\0' == ch) {

temp\_entry.field[nfield] = (char\*)malloc((i - field\_start + 1) \* sizeof(char));

strcpy(temp\_entry.field[nfield], &buf[field\_start]);

field\_start = i + 1;

nfield++;

if(DICT\_FLD\_CNT == nfield) {

list1\_push\_back(&dict,

dict\_entry\_new(temp\_entry.field[0], temp\_entry.field[1], temp\_entry.field[2])

);

for(fld = 0; fld < DICT\_FLD\_CNT; fld++)

if(temp\_entry.field[fld])

free(temp\_entry.field[fld]);

nfield = 0;

}

}

} // for(i)

free(buf);

} // if(buf)

return 0;

}

int dict\_entry\_save(void\* data, int index, void\* param) {

const char\* empty = "";

int i;

DICT\_ENTRY\* entry = (DICT\_ENTRY\*)data;

FILE\* file = (FILE\*)param;

for(i = 0; i < DICT\_FLD\_CNT; i++) {

if(entry->field[i]) {

fwrite(entry->field[i], strlen(entry->field[i]) + 1, 1, file);

} else {

fwrite(empty, 2, 1, file);

}

}

return 1; // продолжить итерации по остальным элементам

}

int dict\_save(const char\* file\_name) {

int nwrote;

FILE\* file = fopen(file\_name, "wb");

if(!file)

return -1;

#if 1

list1\_for\_each(&dict, dict\_entry\_save, file);

#else

DICT\_ENTRY\* curr = dict\_head;

while(curr) {

int i, nwrote;

for(i = 0; i < DICT\_FLD\_CNT; i++) {

if(curr->field[i]) {

nwrote = fwrite(curr->field[i], strlen(curr->field[i]) + 1, 1, file);

} else {

nwrote = fwrite(empty, 2, 1, file);

}

if(nwrote != 1) {

return -1;

}

}

curr = curr->next;

}

#endif

fclose(file);

return 0;

}

//

DICT\_ENTRY\* dict\_entry\_new(const char\* word\_eng, const char\* word\_part, const char\* word\_rus) {

DICT\_ENTRY\* new\_entry;

if(NULL == word\_eng | NULL == word\_part | NULL == word\_rus)

return NULL;

new\_entry = (DICT\_ENTRY\*)malloc(sizeof(DICT\_ENTRY));

memset(new\_entry, 0x00, sizeof(DICT\_ENTRY));

new\_entry->field[0] = (char\*)malloc((strlen(word\_eng) + 1) \* sizeof(char));

strcpy(new\_entry->field[0], word\_eng);

new\_entry->field[1] = (char\*)malloc((strlen(word\_part) + 1) \* sizeof(char));

strcpy(new\_entry->field[1], word\_part);

new\_entry->field[2] = (char\*)malloc((strlen(word\_rus) + 1) \* sizeof(char));

strcpy(new\_entry->field[2], word\_rus);

return new\_entry;

}

//

void dict\_entry\_delete(void\* data) {

int i;

DICT\_ENTRY\* entry = (DICT\_ENTRY\*)data;

if(entry) {

for(i = 0; i < DICT\_FLD\_CNT; i++)

if(entry->field[i])

free(entry->field[i]);

free(entry);

}

}

char\* file\_load(const char\* file\_name, off\_t\* psize) {

int nread;

struct \_stat stat\_buf;

FILE\* file;

char\* buf;

int i, j, field\_start = 0, nfield = 0;

\*psize = 0;

if(0 != \_stat(file\_name, &stat\_buf))

return NULL;

if(0 == stat\_buf.st\_size)

return NULL;

buf = (char\*)malloc(stat\_buf.st\_size \* sizeof(char));

if(!buf)

return NULL;

file = fopen(file\_name, "rb");

if(!file) {

free(buf);

return NULL;

}

nread = fread(buf, stat\_buf.st\_size, 1, file);

fclose(file);

if(0 == nread) {

free(buf);

return NULL;

} // if(nread > 0)

\*psize = stat\_buf.st\_size;

return buf;

}

**input.h**

#ifndef \_\_DICTIONARY\_INPUT\_DLG\_H\_\_

#define \_\_DICTIONARY\_INPUT\_DLG\_H\_\_

#include <windows.h>

#define MAX\_TITLE 50

#define TITLE 0

#define BUFFER 1

#define COLUMNS 2

typedef struct InputBox {

HANDLE handle;

SMALL\_RECT rect;

CHAR\_INFO\* bak;

CHAR\_INFO\* wnd;

COORD size; // width, height

WORD edit\_attr;

int max\_width;

// буфер ввода

//char\* buffer;

// 2-мерный массив строк: 1я колонка - надписи

// 2я колонка - редактируемые поля (можно передавать начальные значения)

char\*\*\* contents;

int row\_count; // количество полей (пар надпись/значение)

int row; // текущее редактируемое поле

} InputBox;

int box\_init(InputBox\* box, HANDLE handle, SMALL\_RECT rect, char\*\*\* contents, int row\_count);

void box\_clear(InputBox\* box);

int box\_save(InputBox\* box);

int box\_draw(InputBox\* box);

int box\_restore(InputBox\* box);

#endif

**input.c**

#include "input.h"

#include "codes.h"

#include <stdlib.h>

#include <string.h>

#include <conio.h>

int box\_init(InputBox\* box, HANDLE handle, SMALL\_RECT rect, char\*\*\* contents, int row\_count) {

int i, size;

if(!contents || !row\_count)

return -1;

memset(box, 0x00, sizeof(InputBox));

box->handle = handle;

box->rect = rect;

box->size.X = rect.Right - rect.Left + 1;

box->size.Y = rect.Bottom - rect.Top + 1;

size = box->size.X \* box->size.Y \* sizeof(CHAR\_INFO);

box->bak = (CHAR\_INFO\*)malloc(size);

if(!box->bak) {

return -1;

}

memset(box->bak, 0x00, size);

box->wnd = (CHAR\_INFO\*)malloc(size);

if(!box->wnd) {

free(box->bak);

return -1;

}

memset(box->wnd, 0x00, size);

box->max\_width = (box->size.X - 2) / 2;

/\*box->buffer = (char\*)malloc((box->max\_width + 1) \* sizeof(char));

if(!box->buffer) {

free(box->bak);

free(box->wnd);

return -1;

}

memset(box->buffer, 0x00, box->max\_width \* sizeof(char));\*/

box->row\_count = row\_count;

#if 1 // используем внешнюю память

box->contents = contents;

#else

box->contents = (char\*\*\*)malloc(box->row\_count \* sizeof(char\*\*));

if(!box->contents) {

free(box->bak);

free(box->wnd);

//free(box->buffer);

return -1;

}

memset(box->contents, 0x00, box->row\_count \* sizeof(char\*\*));

for(i = 0; i < row\_count; ++i) {

box->contents[i] = (char\*\*)malloc(COLUMNS \* sizeof(char\*));

// надпись

box->contents[i][TITLE] = (char\*)malloc((MAX\_TITLE + 1) \* sizeof(char));

strcpy(box->contents[i][TITLE], contents[i][TITLE]);

// буфер ввода

box->contents[i][BUFFER] = (char\*)malloc((box->max\_width + 1) \* sizeof(char));

memset(box->contents[i][BUFFER], ' ', (box->max\_width + 1) \* sizeof(char));

if(contents[i][BUFFER]) {

strncpy(box->contents[i][BUFFER], contents[i][BUFFER], box->max\_width);

}

box->contents[i][BUFFER][box->max\_width] = '\0';

}

#endif

box->edit\_attr = BACKGROUND\_INTENSITY | BACKGROUND\_GREEN | BACKGROUND\_BLUE

//| FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE

;

return 0;

}

void box\_clear(InputBox\* box) {

int i;

if(box->bak) {

free(box->bak);

box->bak = NULL;

}

if(box->wnd) {

free(box->wnd);

box->wnd = NULL;

}

/\*if(box->buffer) {

free(box->buffer);

box->buffer = NULL;

}

if(box->contents) {

for(i = 0; i < box->row\_count; ++i) {

if(box->contents[i]) {

if(box->contents[i][TITLE])

free(box->contents[i][TITLE]);

if(box->contents[i][BUFFER]);

free(box->contents[i][BUFFER]);

free(box->contents[i]);

}

}

free(box->contents);

box->contents = NULL;

}\*/

}

int box\_save(InputBox\* box) {

COORD coord = { 0, 0 };

int ok = ReadConsoleOutput(

box->handle, // экранный буфер, из которого читаем

box->bak, // буфер, в который копируем

box->size, // размеры bak: колонки/строки

coord, // верхняя левая ячейка назначения в bak

&box->rect); // источник - прямоугольник экранного буфера

return ok ? 0 : -1;

}

int box\_restore(InputBox\* box) {

COORD coord = { 0, 0 };

int ok = WriteConsoleOutput(

box->handle, // экранный буфер, в который пишем

box->bak, // буфер, из которого копируем

box->size, // размеры bak: колонки/строки

coord, // исходная верхняя левая ячейка в bak

&box->rect); // приёмник - прямоугольник экранного буфера

if (!ok) {

return -1;

}

return ok ? 0 : -1;

}

int draw\_background(InputBox\* box) {

COORD coord = { 0, 0 };

int i, j, index = 0, ok, len;

WORD background = BACKGROUND\_BLUE | BACKGROUND\_GREEN;

for(i = 0; i < box->size.Y; ++i) {

for(j = 0; j < box->size.X; ++j) {

index = (i \* box->size.X) + j;

box->wnd[index].Attributes = background;

box->wnd[index].Char.AsciiChar = ' ';

}

}

for(i = 1; i <= box->row\_count; ++i) {

for(j = 1; j < box->max\_width - 1; ++j) {

index = (i \* box->size.X) + j;

box->wnd[index].Attributes = background; // box->edit\_attr;

len = strlen(box->contents[i - 1][TITLE]);

if(j <= len) {

box->wnd[index].Char.AsciiChar = box->contents[i - 1][TITLE][j-1];

} else {

box->wnd[index].Char.AsciiChar = ' ';

}

}

for(j = box->max\_width; j < box->size.X - 1; ++j) {

index = (i \* box->size.X) + j;

box->wnd[index].Attributes = box->edit\_attr;

box->wnd[index].Char.AsciiChar = ' ';

}

}

ok = WriteConsoleOutput(

box->handle, // экранный буфер, в который пишем

box->wnd, // буфер, из которого копируем

box->size, // размеры wnd: колонки/строки

coord, // исходная верхняя левая ячейка в wnd

&box->rect); // приёмник - прямоугольник экранного буфера

return ok ? 0 : -1;

}

void box\_gotoxy(InputBox\* box, int x, int y)

{

COORD cursorPos;

cursorPos.X = x;

cursorPos.Y = y;

SetConsoleCursorPosition(box->handle, cursorPos);

}

void showCursor(HANDLE handle, int visible)

{

CONSOLE\_CURSOR\_INFO ccInfo;

ccInfo.bVisible = visible;

ccInfo.dwSize = 20;

SetConsoleCursorInfo(handle, &ccInfo);

}

static int accept\_input(char ch) {

if(isalnum(ch))

return 1;

if(ispunct(ch) || ' ' == ch)

return 1;

if((-96 <= ch && ch <= -81) || (-32 <= ch && ch <= -17) || -15 == ch)

return 1;

return 0;

}

void trim(char \*str) {

int len = strlen(str);

int beg = 0;

while(str[len - 1] == ' ')

str[--len] = 0;

while(str[beg] == ' ')

str[beg++] = 0;

if(beg)

memmove(str, str + beg, len - beg + 1);

}

int box\_draw(InputBox\* box) {

int i, left, top, run = 1;

int \*pos, \*len;

if(0 == box->row\_count)

return -1;

box->row = 0;

pos = (int\*)malloc(box->row\_count \* sizeof(int));

memset(pos, 0x00, box->row\_count \* sizeof(int));

len = (int\*)malloc(box->row\_count \* sizeof(int));

memset(len, 0x00, box->row\_count \* sizeof(int));

if(-1 == draw\_background(box))

return -1;

// координаты начала ввода

left = box->rect.Left + box->max\_width; // 1;

top = box->rect.Top + 1;

// Draw wnd

SetConsoleTextAttribute(box->handle, box->edit\_attr);

// Input

box\_gotoxy(box, left, top);

showCursor(box->handle, 1);

// буфер ввода

//memset(box->contents[box->row][BUFFER], ' ', (box->max\_width + 1) \* sizeof(char));

box->contents[box->row][BUFFER][box->max\_width] = '\0';

//char tmp[2];

//tmp[1] = '\0';

for(i = 0; i < box->row\_count; ++i) {

len[i] = strlen(box->contents[i][BUFFER]);

box\_gotoxy(box, left, top + i);

printf("%s", box->contents[i][BUFFER]);

}

box\_gotoxy(box, left, top + box->row);

while (run) {

if (kbhit()) {

int ch = \_getch();

//tmp[0] = ch;

//CharToOemA(tmp, tmp);

//OemToCharA(tmp, tmp);

//ch = tmp[0];

switch (ch) {

case KEY\_HOME:

pos[box->row] = 0;

box\_gotoxy(box, left + pos[box->row], top);

break;

case KEY\_ARROW\_LEFT:

if(pos[box->row] > 0) {

pos[box->row]--;

box\_gotoxy(box, left + pos[box->row], top);

}

break;

case KEY\_ARROW\_RIGHT:

if(pos[box->row] < len[box->row]) {

pos[box->row]++;

box\_gotoxy(box, left + pos[box->row], top);

}

break;

case KEY\_END:

pos[box->row] = len[box->row];

box\_gotoxy(box, left + pos[box->row], top);

break;

//case KEY\_ARROW\_UP: case KEY\_ARROW\_DOWN:

// break;

case KEY\_ENTER:

for(i = 0; i < box->row\_count; ++i)

trim(box->contents[i][BUFFER]);

//printf("\n%s", box->contents[box->row][BUFFER]);

run = 0;

break;

case KEY\_BACKSPACE:

if(pos[box->row] > 0) {

for(i = pos[box->row]; i < len[box->row]; ++i)

box->contents[box->row][BUFFER][i - 1] = box->contents[box->row][BUFFER][i];

box->contents[box->row][BUFFER][len[box->row] - 1] = ' ';

pos[box->row]--;

len[box->row]--;

}

break;

case KEY\_DEL:

if(pos[box->row] < len[box->row]) {

for(i = pos[box->row]; i < len[box->row]; ++i)

box->contents[box->row][BUFFER][i] = box->contents[box->row][BUFFER][i + 1];

box->contents[box->row][BUFFER][len[box->row] - 1] = ' ';

len[box->row]--;

}

break;

case KEY\_TAB:

if(box->row\_count && ++box->row == box->row\_count)

box->row =0;

break;

case KEY\_ESC:

run = 0;

break;

default:

if(accept\_input(ch)) {

if(len[box->row] < box->max\_width) {

if(pos[box->row] < len[box->row]) {

for(i = len[box->row]; i > pos[box->row]; --i)

box->contents[box->row][BUFFER][i] = box->contents[box->row][BUFFER][i - 1];

len[box->row]++;

box->contents[box->row][BUFFER][pos[box->row]] = ch;

} else {

box->contents[box->row][BUFFER][pos[box->row]] = ch;

pos[box->row]++;

len[box->row]++;

}

//printf("%c", ch);

}

}

break;

} // switch(iKey)

#if 0 // отладка

box\_gotoxy(box, left, top + 10);

printf("%c\t%d\tpos %d\tlen %d", ch, ch, pos[box->row], len[box->row]);

#endif

box\_gotoxy(box, left, top + box->row);

printf("%s", box->contents[box->row][BUFFER]);

box\_gotoxy(box, left + pos[box->row], top + box->row);

} // if(kbhit())

} // while(run)

free(pos);

if(-1 == box\_restore(box)) {

box\_clear(&box);

return -1;

}

return 0;

}