**Курсовая работа**

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

2018 г

# Задание

Оглавление

[Задание 2](#_Toc516673827)

[Введение 4](#_Toc516673828)

[1 Исходные данные и способ их ввода 5](#_Toc516673829)

[2 Глобальные данные 6](#_Toc516673830)

[2.1 Основные модули программы 6](#_Toc516673831)

[2.2 Описание глобальных данных модуля main.c 6](#_Toc516673832)

[2.3 Описание глобальных данных модуля dict.c 7](#_Toc516673833)

[3 Описание пользовательских макроопределений 8](#_Toc516673834)

[3.1 Макроопределения модуля main 8](#_Toc516673835)

[3.2 Макроопределения модуля dict 8](#_Toc516673836)

[3.3 Макроопределения модуля menu 8](#_Toc516673837)

[3.4 Макроопределения модуля input 9](#_Toc516673838)

[3.5 Глобальные макроопределения 10](#_Toc516673839)

[4 Описание пользовательских функции 11](#_Toc516673840)

[4.1 Функции модуля main 11](#_Toc516673841)

[4.2 Функции модуля dict 13](#_Toc516673842)

[4.3 Функции модуля menu 13](#_Toc516673843)

[4.4 Функции модуля input 14](#_Toc516673844)

[4.5 Функции модуля list 15](#_Toc516673845)

[5 Нестандартные функции 17](#_Toc516673846)

[6 Внешние файлы 18](#_Toc516673847)

[7 Описание порядка работы с программой 19](#_Toc516673848)

[Заключение 20](#_Toc516673849)

[Список литературы 21](#_Toc516673850)

[Приложения 22](#_Toc516673851)

# 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.

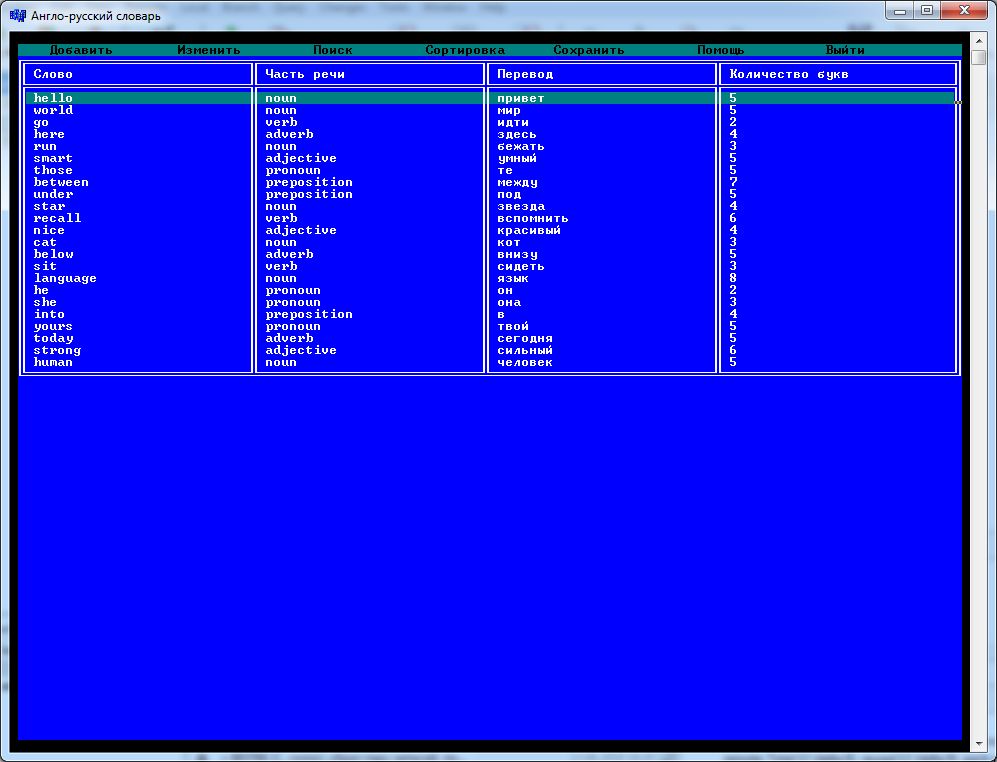


Рисунок 1 – Главное окно программы

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.

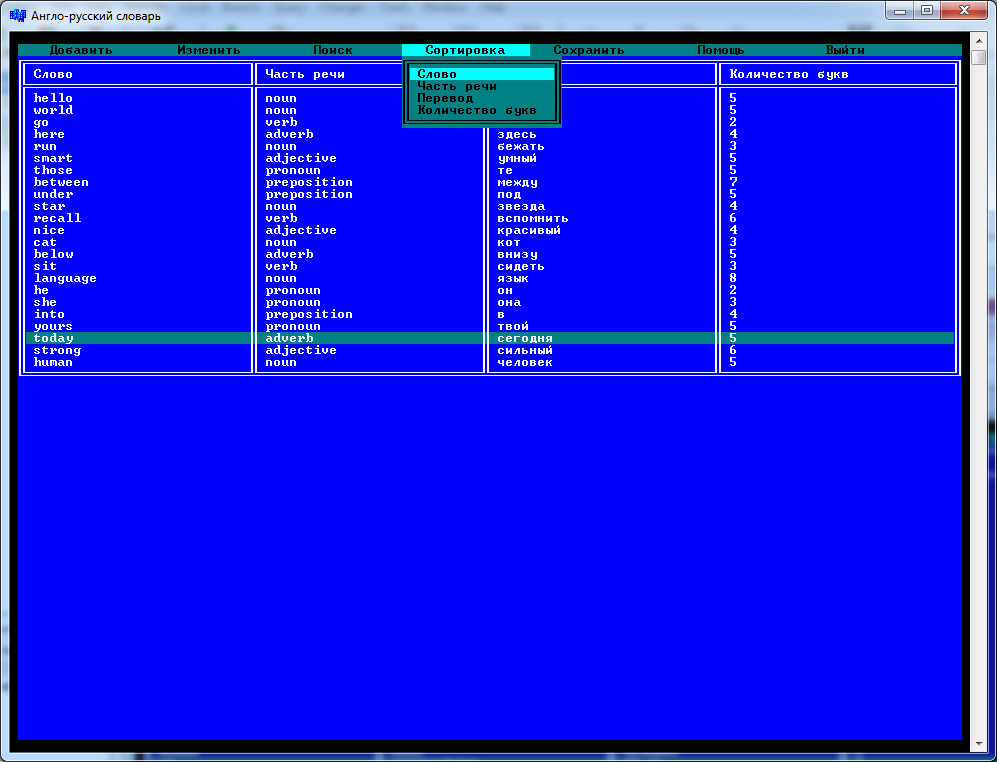


Рисунок 1 – Вызов функции сортировки

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

// Ïðåäâàðèòåëüíîå îáúÿâëåíèå ôóíêöèé

// Îñíîâíàÿ ðàáî÷àÿ ôóíêöèÿ ïðîãðàììû

int Run();

// ôóíêöèè îáðàòíîãî âûçîâà

// Ôóíêöèè, êîòîðûå ïîäêëþ÷àþòñÿ ê ïóíêòàì ìåíþ

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\*);

// Ôóíêöèè, êîòîðûå ïîäêëþ÷àþòñÿ ê ãîðÿ÷èì êëàâèøàì

// Âûõîä - ñîõðàíèòü äàííûå

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

// Âûõîä - ñîõðàíèòü äàííûå

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

// Âûõîä - îòìåíà

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

// Îáðàáîò÷èê êëàâèøè ESC ïî óìîë÷àíèþ âûõîäèò èç öèêëà ñîîáùåíèé òåêóùåãî ìåíþ

int DefaultESC(MENU\* menu);

// Îáðàáîò÷èê êëàâèøè ESC

int ESC(MENU\* menu);

// Îáðàáîò÷èê êëàâèøè F1 (Ñïðàâêà)

int F1(MENU\* menu);

// Îáðàáîò÷èê êëàâèøè F2 (Ñîõðàíèòü)

int F2(MENU\* menu);

// Îáðàáîò÷èê êëàâèøè F3 (Ïîèñê)

int F3(MENU\* menu);

// Îáðàáîò÷èê êëàâèøè F4 (Ðåäàêòèðîâàíèå)

int F4(MENU\* menu);

// Îáðàáîò÷èê êëàâèøè F9 (Ïåðåõîä íà âåðõíåå ìåíþ)

int F9(MENU\* menu);

// Îáðàáîò÷èê ïðîêðóòêè ìåíþ íà îäèí ýëåìåíò âïåð¸ä èëè íàçàä

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

// Ïåðåíîñ î÷åðåäíîãî ýëåìåíòà ñëîâàðÿ â ýëåìåíò îïèñàíèÿ ïóíêòà ìåíþ

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

// Ïðîâåðêà ýëåìåíòà ñëîâàðÿ íà ñîîòâåòñòâèå êðèòåðèþ ïîèñêà

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

// Ñðàâíåíèå äâóõ ýëåìåíòîâ ñëîâàðÿ

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

// âñïîìîãàòåëüíûå ôóíêöèè

// Îòîáðàæåíèå òåêñòà ñïðàâêè èç ôàéëà íà âñïëûâàþùåì îêíå

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

// Îòîáðàæåíèå óïðîù¸ííîãî âåðòèêàëüíîãî ìåíþ â îäèí ñòîëáåö

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

// Çàãðóçêà ïóíêòîâ ìåíþ èç ôàéëà

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

// Çàãðóçêà òåñòîâûõ äàííûõ â ñëîâàðü

void LoadInitialData();

// ìàêñèìàëüíî äîïóñòèìàÿ äëèíà ñòðîêè

#define MAX\_STRING 256

// ïîëüçîâàòåëüñêèå ìåòêè, ÷òîáû ðàçëè÷àòü ìåíþ â ôóíêöèÿõ îáðàòíîãî âûçîâà

#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

// êîëè÷åñòâî ïóíêòîâ âåðõíåãî ìåíþ

#define top\_item\_count 7

// êîëè÷åñòâî ÿ÷ååê (ñòîëáöîâ) îñíîâíîãî òàáëè÷íîãî ìåíþ

#define main\_column\_count 4

// êîëè÷åñòâî ÿ÷ååê (ñòîëáöîâ) ìåíþ ïîäòâåðæäåíèÿ âûõîäà

#define exit\_column\_count 1

// êîëè÷åñòâî ïóíêòîâ ìåíþ ïîäòâåðæäåíèÿ âûõîäà

#define exit\_item\_count 3

// ìàññèâ ñ îïèñàíèÿìè ýëåìåíòîâ âåðõíåãî ìåíþ

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

{ { "Äîáàâèòü", 0 }, Add },

{ { "Èçìåíèòü", 0 }, Edit },

{ { "Ïîèñê", 0 }, Search },

{ { "Ñîðòèðîâêà", 0 }, Sort },

{ { "Ñîõðàíèòü", 0 }, Save },

{ { "Ïîìîùü", 0 }, Help },

{ { "Âûéòè", 0 }, Exit },

};

// ìàññèâ ñ îïèñàíèÿìè ýëåìåíòîâ ìåíþ ïîäòâåðæäåíèÿ âûõîäà

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

{ { "Äà", 0 }, ExitYes },

{ { "Íåò", 0 }, ExitNo },

{ { "Îòìåíà", 0 }, ExitCancel },

};

// ìàññèâ çàãîëîâêîâ îñíîâíîé òàáëèöû

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

// ìàññèâ çàãîëîâêîâ (1 çàãîëîâîê) ìåíþ ïîäòâåðæäåíèÿ âûõîäà

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

// êîä çàâåðøåíèÿ äëÿ ôóíêöèé îáðàòíîãî âûçîâà

int exit\_code = 0; // íå íóëåâîå çíà÷åíèå ñ÷èòàåòñÿ îáÿçàòåëüíûì

// ïðèçíàê ïåðåçàïóñêà îñíîâíîé òàáëèöû

int redraw\_main = 1;

// ïðèçíàê èçìåíèâøèõñÿ äàííûõ

int data\_modified = 0;

// ïðèçíàê íåîáõîäèìîñòè ñîõðàíåíèÿ äàííûõ

int save\_data = 0;

// ïðèçíàê îòìåíû âûõîäà

int exit\_canceled = 0;

// èíäåêñ äëÿ èíèöèàëèçàöèè îáíîâë¸ííîãî ìåíþ

int initial\_table\_index = 0;

// áóôåð äëÿ õðàíåíèÿ íàñòðîåê êîíñîëè

CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo;

// ýêçåìïëÿð âåðõíåãî ìåíþ

MENU top\_menu;

// óêàçàòåëü íà ýêçåìïëÿð îñíîâíîé òàáëèöû

MENU \*ptable = NULL;

// Òî÷êà âõîäà ïðîãðàììû

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

{

return Run();

}

// Âûõîä - ñîõðàíèòü äàííûå

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

save\_data = 1;

return -1;

}

// Âûõîä - íå ñîõðàíÿòü äàííûå

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

save\_data = 0;

return -1;

}

// Âûõîä - îòìåíà

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

exit\_canceled = 1;

return -1;

}

// Îáðàáîò÷èê êëàâèøè ESC ïî óìîë÷àíèþ âûõîäèò èç öèêëà ñîîáùåíèé òåêóùåãî ìåíþ

int DefaultESC(MENU\* menu) {

return -1;

}

// Îáðàáîò÷èê êëàâèøè ESC

int ESC(MENU\* menu) {

return Exit(menu, NULL);

}

// Îáðàáîò÷èê êëàâèøè F1 (Ñïðàâêà)

int F1(MENU\* menu) {

Help(menu, NULL);

return 0;

}

// Îáðàáîò÷èê êëàâèøè F2 (Ñîõðàíèòü)

int F2(MENU\* menu) {

Save(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Îáðàáîò÷èê êëàâèøè F3 (Ïîèñê)

int F3(MENU\* menu) {

Search(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Îáðàáîò÷èê êëàâèøè F4 (Ðåäàêòèðîâàíèå)

int F4(MENU\* menu) {

Edit(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Îáðàáîò÷èê êëàâèøè F9 (Ïåðåõîä íà âåðõíåå ìåíþ)

int F9(MENU\* menu) {

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

if(exit\_code)

return exit\_code;

return 0;

}

// Îáðàáîò÷èê ïðîêðóòêè ìåíþ íà îäèí ýëåìåíò âïåð¸ä èëè íàçàä

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;

}

}

// Ïåðåíîñ î÷åðåäíîãî ýëåìåíòà ñëîâàðÿ â ýëåìåíò îïèñàíèÿ ïóíêòà ìåíþ

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; // ïðîäîëæèòü èòåðàöèè ïî îñòàëüíûì ýëåìåíòàì

}

// Ïðîâåðêà - ïîäõîäèò ëè ýëåìåíò ñëîâàðÿ êðèòåðèþ ïîèñêà

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

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

const char\* str = (const char\*)param;

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

}

// Ñðàâíåíèå äâóõ ýëåìåíòîâ ñëîâàðÿ

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]) < 0 ? 1 : 0;

}

// Îñíîâíàÿ ðàáî÷àÿ ôóíêöèÿ ïðîãðàììû

int Run() {

char\* title = "Àíãëî-ðóññêèé ñëîâàðü";

int i, fld;

HANDLE hstdout;

SMALL\_RECT rect;

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

int main\_menu\_items\_count = 0;

#if 0

LoadInitialData();

return 0;

#endif

// Èíèöèàëèçàöèÿ çàãîëîâêîâ ãëîáàëüíûõ ìåíþ

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

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

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

CharToOemA("×àñòü ðå÷è", main\_headers[1]);

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

CharToOemA("Êîëè÷åñòâî áóêâ", main\_headers[3]);

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

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

CharToOemA("Ñîõðàíèòü èçìåíåíèÿ?", 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); // äåñêðèïòîð êîíñîëüíîãî îêíà

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

return -1;

}

// Ïîëó÷àåì ðàçìåðû êîíñîëè

GetConsoleScreenBufferInfo(hstdout, &csbInfo);

rect.Left = csbInfo.srWindow.Left + 1;

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

rect.Top = csbInfo.srWindow.Top + 1;

rect.Bottom = rect.Top;

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);

// Çàãðóçêà ñëîâàðÿ èç ôàéëà

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; // ëþáîé callback ìîæåò çàïëàíèðîâàòü ïåðåðèñîâêó

//

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; // ñëîâàðü ìîã áûòü èçìåí¸í

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));

// ïåðåêèäûâàåì ñëîâàðü â òàáëèöó

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

// ñîçäà¸ì ýëåìåíòû ìåíþ

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

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

// óñòàíàâëèâàåì òåêóùèé èíäåêñ

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

initial\_table\_index = 0;

// ïðîäîëæàåì íàñòðîéêó ìåíþ

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\_INTENSITY | BACKGROUND\_BLUE

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

);

// áëîêèðóþùèé âûçîâ: îñíîâíîé öèêë îáðàáîòêè ñîîáùåíèé ãëàâíîé òàáëèöû

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

// âîçâîæíî, âûøëè ïî ãëîáàëüíîìó êîäó çàâåðøåíèÿ, ñáðàñûâàåì åãî

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;

}

// Ôóíêöèÿ ìåíþ <Âûõîä>

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;

}

// Ôóíêöèÿ ìåíþ <Äîáàâèòü>

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

int i;

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

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

DICT\_ENTRY\* entry;

InputBox box;

char\*\*\* contents;

int row\_count = DICT\_FLD\_CNT;

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\*));

// íàäïèñü

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

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

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

// áóôåð ââîäà

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, wndRect, 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;

}

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

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

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

}\*/

list1\_push\_back(&dict,

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

);

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;

data\_modified = 1;

redraw\_main = 1;

exit\_code = -1;

return exit\_code;

}

// Ôóíêöèÿ ìåíþ <Èçìåíèòü>

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

int i;

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

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

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

InputBox box;

char\*\*\* contents;

int row\_count = DICT\_FLD\_CNT;

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\*));

// íàäïèñü

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

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

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

// áóôåð ââîäà

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';

}

if(-1 == box\_init(&box, pm->hStdOut, wndRect, 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;

}

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

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

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

}

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;

data\_modified = 1;

redraw\_main = 1;

exit\_code = -1;

return exit\_code;

}

// Ôóíêöèÿ ìåíþ <Óäàëèòü>

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;

}

// Ôóíêöèÿ ìåíþ <Ïîèñê>

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

int i;

int index;

int row = 0;

SMALL\_RECT wndRect = { 3, 11, 29, 13 };

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

InputBox box;

char\*\*\* contents;

int row\_count = 1;

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\*));

// íàäïèñü

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

//strcpy(contents[i][TITLE], "Search:");

CharToOemA("Ñëîâî: ", contents[i][TITLE]);

// áóôåð ââîäà

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';

}

if(-1 == box\_init(&box, pm->hStdOut, wndRect, 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;

}

index = list1\_search(&dict, dict\_entry\_find, contents[row][BUFFER]);

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;

}

// Ôóíêöèÿ ïîäìåíþ <Ñîðòèðîâêà>

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;

}

// Ôóíêöèÿ ìåíþ <Ñîðòèðîâêà>

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 âåðò. ëèíèè ïî êðàÿì

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;

}

// Ôóíêöèÿ ìåíþ <Ñîõðàíèòü>

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

save\_data = 0;

data\_modified = 0;

dict\_save(dict\_file\_name);

return -1;

}

// Ôóíêöèÿ ìåíþ <Ïîìîùü>. Çàïîëíÿåòñÿ êîäîì ïîëüçîâàòåëÿ

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", "Ñïðàâêà î ïðîãðàììå", 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", "Ñïðàâêà: Ôóíêöèÿ ñîðòèðîâêè", rect);

break;

}

return result;

}

// Âñïîìîãàòåëüíûå ôóíêöèè

// Çàãðóçêà ïóíêòîâ ìåíþ èç ôàéëà

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]);

// óäàëÿåì ïðåäïîñëåäíèé ñèìâîë - '\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; // äîñòèãíóò ëèìèò ñòðîê

}

}

fclose(file);

item\_count = i;

if(pcount)

\*pcount = item\_count;

return menu\_items;

}

// Îòîáðàæåíèå óïðîù¸ííîãî âåðòèêàëüíîãî ìåíþ â îäèí ñòîëáåö

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;

// êîëè÷åñòâî ÿ÷ååê ìåíþ (ñòîëáöîâ)

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;

}

// Îòîáðàæåíèå òåêñòà ñïðàâêè èç ôàéëà íà âñïëûâàþùåì îêíå

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 = çàãîëîâîê + 3 ãîðèç. ëèíèèè

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);

}

// Çàãðóçêà òåñòîâûõ äàííûõ â ñëîâàðü

void LoadInitialData() {

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

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

return -1;

list1\_clear(&dict);

getchar();

}

list.h

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

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

struct LIST1\_ITEM;

// ñòðóêòóðà ýëåìåíòà îäíîñâÿçíîãî ñïèñêà

typedef struct LIST1\_ITEM {

struct LIST1\_ITEM\* next;

void\* data;

} LIST1\_ITEM;

// îäíîñâÿçíûé ñïèñîê

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

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

// åñëè âåðí¸ò 0, îáðàáîòêà ïðåêðàòèòñÿ íà ýòîé èòåðàöèè

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);

// ñòðóêòóðà îäíîñâÿçíîãî ñïèñêà

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);

// Ïðîêðóòèòü òåêóùèé óêàçàòåëü âïåð¸ä

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

// Ïðîêðóòèòü òåêóùèé óêàçàòåëü íàçàä

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

//

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

//

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

//

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

// Óäàëèòü ýëåìåíò ñïèñêà

// âîçâðàùàåò òåêóùèé ýëåìåíò (âîçìîæíî, íîâûé)

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

// Óäàëèòü òåêóùèé ýëåìåíò ñïèñêà

// âîçâðàùàåò íîâûé òåêóùèé ýëåìåíò

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

//

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

//

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

// ïîèñê ýëåìåíòà (óñòàíîâèò òåêóùèé è âåðí¸ò åãî ïîðÿäêîâûé íîìåð)

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

// ñîðòèðîâêà ñïèñêà

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

#endif

list.cpp

#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 { // äîáàâëÿåòñÿ ïåðâûé ýëåìåíò

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 { // äîáàâëÿåòñÿ ïåðâûé ýëåìåíò

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;

}

// Ïðîêðóòèòü òåêóùèé óêàçàòåëü âïåð¸ä

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;

}

// Ïðîêðóòèòü òåêóùèé óêàçàòåëü íàçàä

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)) {

// ñìåñòèòü òåêóùèé óêàçàòåëü ïðè íåîáõîäèìîñòè

if(curr == list->curr) {

list->curr = prev;

}

// óäàëèòü íàéäåííûé ýëåìåíò âìåñòå ñ äàííûìè

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) {

// îïðåäåëèòü íîâûé òåêóùèé ýëåìåíò

prev->next = next;

list->curr = next;

if(curr == list->head) {

list->head = next; // åñëè ýòî ïîñëåäíèé ýëåìåíò, next ðàâåí 0

}

if(curr == list->tail) {

if(list->head == NULL)

list->tail = NULL; // åñëè ýòî ïîñëåäíèé ýëåìåíò, list->head òóò óæå ðàâåí 0

else

list->tail = prev;

}

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

list->curr = prev;

}

// óäàëèòü ñòàðûé òåêóùèé ýëåìåíò âìåñòå ñ äàííûìè

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);

}

// âñïîìîãàòåëüíûå ôóíêöèè

// ñîçäàòü ýëåìåíò ñïèñêà - êîíòåéíåð äàííûõ

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;

}

// óäàëèòü ýëåìåíò ñïèñêà - êîíòåéíåð äàííûõ

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;

}

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 // ìàêñèìàëüíàÿ äëèíà çàãîëîâêà

// Ýòàïû ÷àñòè÷íîé îòðèñîâêè ìåíþ

#define MENU\_FLAG\_WND 0x1 // îòðèñîâêà ôîíîâîãî îêíà ìåíþ

#define MENU\_FLAG\_ITEMS 0x2 // îòðèñîâêà ýëåìåíòîâ ìåíþ

#define MENU\_DRAW\_SEL 0x4 // âûäåëåíèå òåêóùåãî ýëåìåíòà

#define MENU\_NAVIGATOR 0x8 // íàâèãàöèÿ ïî óìîë÷àíèþ (ñòðåëêè è Enter)

#define MENU\_HOTKEYS 0x10 // îáðàáîòêà ãîðÿ÷èõ êëàâèø

// ïîëíîöåííûé çàïóñê îêíà ìåíþ ñ öèêëîì ñîîáùåíèé

#define MENU\_FULL\_FLAGS 0x1F

// Ìàêñèìàëüíîå ÷èñëî ÿ÷ååê â îäíîì ïóíêòå ìåíþ (ìèíèìóì îäíà)

#define MAX\_CELLS 50

// Íàïðàâëåíèå ïåðåìåùåíèå êóðñîðà

#define MENU\_CURR\_FWD 0 // âïåð¸ä

#define MENU\_CURR\_REV 1 // íàçàä

struct ITEM;

// Óêàçàòåëè íà ôóíêöèè void f(void) - îíè áóäóò âûïîëíÿòü ïóíêòû ìåíþ

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

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

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

// Ñòðóêòóðà îïðåäåëåíèÿ ýëåìåíòà ìåíþ

typedef struct {

char\* str[MAX\_CELLS]; // Íàèìåíîâàíèå ïóíêòà ìåíþ (íåñêîëüêî ÿ÷ååê)

ExecuteCurrentCB cb; // Ôóíêöèÿ, ïðèâÿçàííàÿ ê ïóíêòó ìåíþ

} ITEM\_DEF;

// Ñòðóêòóðà äëÿ õðàíåíèÿ ýëåìåíòà ìåíþ

typedef struct ITEM {

int x, y; // Ñòîëáåö è ñòðîêà êîíñîëè

int index;

char\* str; // Íàèìåíîâàíèå ïóíêòà ìåíþ (íåñêîëüêî ÿ÷ååê)

ExecuteCurrentCB cb; // Ôóíêöèÿ, ïðèâÿçàííàÿ ê ïóíêòó ìåíþ

} ITEM;

// Îáëàñòü îêíà ìåíþ

typedef struct {

SMALL\_RECT rect; // êîîðäèíàòû óãëîâ êîíñîëüíîãî îêíà

int M, N; // ðàçìåð ìàòðèöû ñèìâîëîâ

char\*\* m; // ìàòðèöà ñèìâîëîâ

} MENU\_WND;

// Ãîðÿ÷àÿ êëàâèøà

typedef struct {

int code;

ExecuteHotketCB cb; // îáðàáîò÷èê

} HOT\_KEY;

// Ñòðóêòóðà äëÿ âñåãî ìåíþ

typedef struct MENU {

// Óêàçàòåëü íà ðîäèòåëüñêîå ìåíþ (íå âûçûâàþùåå, à òî, êîòîðîå ïåðåêðûâàåì)

struct MENU\* parent;

// Ãëîáàëüíûå ïåðåìåííûå, èñïîëüçóåìûå â ôóíêöèÿõ ìåíþ

HANDLE hStdOut;// = INVALID\_HANDLE\_VALUE; // äåñêðèïòîð êîíñîëüíîãî îêíà

///SMALL\_RECT consolRect; // êîîðäèíàòû óãëîâ êîíñîëüíîãî îêíà

// COORD curspos;//={0,1}; // êîîðäèíàòû òåêñòîâîãî êóðñîðà

WORD workWindowAttributes;// = 158;// àòðèáóòû ðàáî÷åé îáëàñòè

WORD inactiveItemAttributes;// = 31; // àòðèáóòû öâåòà íåàêòèâíîãî ïóíêòà ìåíþ

WORD activeItemAttributes;// = 160; // àòðèáóòû öâåòà àêòèâíîãî ïóíêòà ìåíþ

MENU\_WND wnd;

int cb\_retcode;

int last\_key;

LIST1 hk\_list;

CurrentChangedCB changed\_cb;

char\* hdr;

LIST1 items;

int cell\_count; // êîëè÷åñòâî ÿ÷ååê

int cell\_width; // øèðèíà ÿ÷åéêè

int cell\_start[MAX\_CELLS]; // èíäåêñû íà÷àëà ÿ÷ååê âíóòðè ñòðîêè êàæäîãî ïóíêòà

int item\_width; // êîëè÷åñòâî ñèìâîëîâ â ñòðîêå êàæäîãî ïóíêòà

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

int has\_headers; // 0 - íåò çàãîëîâêà, 1 - åñòü çàãîëîâîê

int has\_border; // 0 - íåò ðàìêè, 1 - åñòü ðàìêà

// èíäåêñ ñòðîêè ñ íèæíåé ãðàíèöåé (äëÿ ïðàâèëüíîé îòðèñîâêè)

int border\_bottom\_index;

// îòñòóïû ñëåâà è ñâåðõó, ñ ó÷¸òîì íàëè÷èÿ ðàìêè è çàãîëîâêà

int left\_pad;

int top\_pad;

// ïîëüçîâàòåëüñêàÿ ìåòêà

int user\_tag;

} MENU;

// Èíèöèàëèçàöèÿ ïîëåé ýêçåìïëÿðà ìåíþ

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[]);

// Äîáàâèòü îáðàáîò÷èê ãîðÿ÷åé êëàâèøè

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

// Î÷èñòêà ïîëåé ýêçåìïëÿðà ìåíþ

void menu\_clear(MENU\* menu);

// Ïðîðèñîâêà ôîíà è ðàìêè

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

// Ïðîðèñîâêà ìåíþ

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

// Óñòàíîâêà öâåòîâûõ àòðèáóòîâ àêòèâíîãî ïóíêòà

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

// Óñòàíîâêà öâåòîâûõ àòðèáóòîâ íåàêòèâíîãî ïóíêòà

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

// Âûäåëèòü ïðåäûäóùèé ïóíêò ìåíþ

void menu\_prev(MENU\* menu);

// Âûäåëèòü ñëåäóþùèé ïóíêò ìåíþ

void menu\_next(MENU\* menu);

// Ïåðåâîä êóðñîðà â òî÷êó x, y

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

// Âûäåëèòü ïóíêò ìåíþ

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

// Çàïîìíèòü ïîëîæåíèå êóðñîðà

void saveCursorPosition(MENU\* menu);

// Î÷èñòêà îêíà

void menu\_cls(MENU\* menu);

// Ñîõðàíèòü êîîðäèíàòû êóðñîðà â ïåðåìåííóþ curspos

void showCursor(MENU\* menu, int visible); // ñêðûòü/ïîêàçàòü êóðñîð

// Óäàëèòü òåêóùèé ýëåìåíò

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);

// Îòðèñîâêà ïóíêòà ìåíþ (callback äëÿ ýëåìåíòà ñïèñêà)

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

// Ïðîâåðèòü êîä, âîçâðàù¸ííûé îáðàáîò÷èêîì

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

// Î÷èñòêà îäíîãî ýëåìåíòà ìåíþ

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

// Îáðàáîòêà ãîðÿ÷åé êëàâèøè (åñëè áûëà íàæàòà)

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;

// ïðîõîä ïî ñòðîêàì îêíà

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) {

// îòðèñîâêà âåðòèêàëüíîé ãðàíèöû ñëåâà è ñïðàâà

if(i < border\_bottom\_index) {

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

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

}

// îòðèñîâêà ãîðèçîíòàëüíîé ëèíèè îòäåëåíèÿ çàãîëîâêà è å¸ ñòûêîâî÷íûõ ëèíèé ñëåâà/ñïðàâà

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) {

// îòðèñîâêà âåðõíåé è íèæíåé ãîðèçîíòàëüíûõ ëèíèé, à òàêæå 4 âíåøíèõ óãëîâ

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

if(0 == i) { // ïåðâàÿ ñòðîêà

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) { // ïîñëåäíÿÿ ñòðîêà

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) {

// îòðèñîâêà âåðòèêàëüíûõ ëèíèé â çàãîëîâêå

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);

// Ñîõðàíÿåì äåñêðèïòîð êîíñîëüíîãî âûâîäà

menu->hStdOut = hstdout;

// Ñîõðàíÿåì ïîëó÷åííûå ðàçìåðû îêíà

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 {

// Ïîëó÷àåì ðàçìåðû êîíñîëè

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;

}

// íå ìîæåì îòîáðàçèòü íå ïîìåùàþùèåñÿ â îêíî ýëåìåíòû

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);

// êîîðäèíàòû óãëîâ êîíñîëè

// 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;

// Èíèöèàëèçàöèÿ ýëåìåíòîâ ìåíþ

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; // ïî öåíòðó

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) {

// îòðèñîâêà ãîðèçîíòàëüíîé ëèíèè îòäåëåíèÿ çàãîëîâêà è å¸ ñòûêîâî÷íûõ ëèíèé ñëåâà/ñïðàâà

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) {

// îòðèñîâêà âåðòèêàëüíûõ ëèíèé â çàãîëîâêå

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);

// íåõîðîøî ìåíÿòü êàæäûé ðàç îðèãèíàëû 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)

//ïîðòèò ïñåâäîãðàôèêó 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); // ñäåëàòü íåàêòèâíûì ïóíêò ìåíþ

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

itemMenu(menu, 1); // âûäåëèòü àêòèâíûé ïóíêò ìåíþ

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); // ñäåëàòü íåàêòèâíûì ïóíêò ìåíþ

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

itemMenu(menu, 1); // âûäåëèòü àêòèâíûé ïóíêò ìåíþ

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; // ïðîäîëæèòü èòåðàöèè ïî îñòàëüíûì ýëåìåíòàì

}

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; // ïðîäîëæèòü èòåðàöèè ïî îñòàëüíûì ýëåìåíòàì

}

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

int result = 0, i, run = 1;

if(flags & MENU\_FLAG\_WND) {

// Óñòàíàâëèâàåì öâåòîâûå ïàðàìåòðû òåêñòà

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

//system("CLS"); // óñòàíîâêà àòðèáóòîâ öâåòà ðàáî÷åé îáëàñòè

menu\_cls(menu);

}

if(flags & MENU\_FLAG\_ITEMS) {

// Íîìåð òåêóùåãî ïóíêòà ìåíþ

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

// ðèñóåì çàãîëîâîê

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);

}

// ðèñóåì ìåíþ

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

fflush(stdout);

}

if(flags & MENU\_DRAW\_SEL) {

itemMenu(menu, 1); // âûäåëèòü ïóíêò ìåíþ

}

//fflush(stdin); // î÷èñòèòü áóôåð êëàâèàòóðû

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

return 0; // íå çàïóñêàåì îáðàáîòêó ñîîáùåíèé

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:

// Âîçâðàùàåì êóðñîð èç ñòðîêè ìåíþ â ïðåæíþþ ïîçèöèþ

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

// Óñòàíîâèòü öâåò ðàáî÷èõ ñîîáùåíèé

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

///showCursor(menu, 1);

// Âûçûâàåì îáðàáîò÷èê ïóíêòà ìåíþ

{

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) {

// îáðàáîòêà ãîðÿ÷èõ êëàâèø

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; // íå ïðîâåðÿòü îñòàëüíûå îáðàáîò÷èêè

}

return 1; // ïðîäîëæèòü èòåðàöèè ïî îñòàëüíûì ýëåìåíòàì

}

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;// èíôîðìàöèÿ î êîíñîëüíîì îêíå â ñòðóêòóðå 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); // ñäåëàòü íåàêòèâíûì ïóíêò ìåíþ

return 0;

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

saveCursorPosition(menu);

}

// î÷èñòèòü áóôåð êëàâèàòóðû

fflush(stdin);

// êóðñîð â òåêóùèé ïóíêò ìåíþ

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

// ñïðÿòàòü êóðñîð

showCursor(menu, 0);

return 1;

}

dict.h

#ifndef dictH

#define dictH

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

#define MAX\_DICT\_STR\_LEN 32

#define DICT\_FLD\_CNT 3

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

#include "list.h"

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

// ñòðóêòóðà çàïèñè ñëîâàðÿ

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";

// ãëîáàëüíûé ýêçåïëÿð ñëîâàðÿ

LIST1 dict;

// Âñïîìîãàòåëüíàÿ ôóíêöèÿ çàãðóçêè èç ôàéëà

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

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

// Ôóíêöèè ñëîâàðÿ

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

//

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; // ïðîäîëæèòü èòåðàöèè ïî îñòàëüíûì ýëåìåíòàì

}

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;

// áóôåð ââîäà

//char\* buffer;

// 2-ìåðíûé ìàññèâ ñòðîê: 1ÿ êîëîíêà - íàäïèñè

// 2ÿ êîëîíêà - ðåäàêòèðóåìûå ïîëÿ (ìîæíî ïåðåäàâàòü íà÷àëüíûå çíà÷åíèÿ)

char\*\*\* contents;

int row\_count; // êîëè÷åñòâî ïîëåé (ïàð íàäïèñü/çíà÷åíèå)

int row; // òåêóùåå ðåäàêòèðóåìîå ïîëå

} 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

#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 // èñïîëüçóåì âíåøíþþ ïàìÿòü

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\*));

// íàäïèñü

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

strcpy(box->contents[i][TITLE], contents[i][TITLE]);

// áóôåð ââîäà

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, // ýêðàííûé áóôåð, èç êîòîðîãî ÷èòàåì

box->bak, // áóôåð, â êîòîðûé êîïèðóåì

box->size, // ðàçìåðû bak: êîëîíêè/ñòðîêè

coord, // âåðõíÿÿ ëåâàÿ ÿ÷åéêà íàçíà÷åíèÿ â bak

&box->rect); // èñòî÷íèê - ïðÿìîóãîëüíèê ýêðàííîãî áóôåðà

return ok ? 0 : -1;

}

int box\_restore(InputBox\* box) {

COORD coord = { 0, 0 };

int ok = WriteConsoleOutput(

box->handle, // ýêðàííûé áóôåð, â êîòîðûé ïèøåì

box->bak, // áóôåð, èç êîòîðîãî êîïèðóåì

box->size, // ðàçìåðû bak: êîëîíêè/ñòðîêè

coord, // èñõîäíàÿ âåðõíÿÿ ëåâàÿ ÿ÷åéêà â bak

&box->rect); // ïðè¸ìíèê - ïðÿìîóãîëüíèê ýêðàííîãî áóôåðà

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, // ýêðàííûé áóôåð, â êîòîðûé ïèøåì

box->wnd, // áóôåð, èç êîòîðîãî êîïèðóåì

box->size, // ðàçìåðû wnd: êîëîíêè/ñòðîêè

coord, // èñõîäíàÿ âåðõíÿÿ ëåâàÿ ÿ÷åéêà â wnd

&box->rect); // ïðè¸ìíèê - ïðÿìîóãîëüíèê ýêðàííîãî áóôåðà

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;

// êîîðäèíàòû íà÷àëà ââîäà

left = box->rect.Left + box->max\_width; // 1;

top = box->rect.Top + 1;

// Draw wnd

/\*SetConsoleTextAttribute(box->handle, pm->activeItemAttributes);

box\_gotoxy(box, left + 0, top + 0);

printf(" ");

box\_gotoxy(box, left + 0, top);

printf(" ");

box\_gotoxy(box, left + 0, top + 2);

printf(" ");

box\_gotoxy(box, left, top);

SetConsoleTextAttribute(box->handle, box->inactiveItemAttributes);

printf(" "); \*/

SetConsoleTextAttribute(box->handle, box->edit\_attr);

// Input

box\_gotoxy(box, left, top);

showCursor(box->handle, 1);

// áóôåð ââîäà

//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 // îòëàäêà

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;

}

codes.h

#ifndef \_\_DICTIONARY\_DB\_CODES\_H\_\_

#define \_\_DICTIONARY\_DB\_CODES\_H\_\_

#define KEY\_ARROW\_UP 72

#define KEY\_ARROW\_LEFT 75

#define KEY\_ARROW\_RIGHT 77

#define KEY\_ARROW\_DOWN 80

#define KEY\_TAB 9

#define KEY\_ENTER 13

#define KEY\_ESC 27

#define KEY\_BACKSPACE 8

#define KEY\_SPACE 32

#define KEY\_DEL 83

#define KEY\_HOME 71

#define KEY\_END 79

#define KEY\_F1 59

#define KEY\_F2 60

#define KEY\_F3 61

#define KEY\_F4 62

#define KEY\_F5 63

#define KEY\_F6 64

#define KEY\_F7 65

#define KEY\_F8 66

#define KEY\_F9 67

#define KEY\_F10 68

#define KEY\_F11 133

#define KEY\_F12 134

#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

#endif // \_\_DICTIONARY\_DB\_KEY\_CODES\_H\_\_