|  |  |
| --- | --- |
| *voenmeh* | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** |
| БГТУ.СМК-Ф-4.2-К5-01 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Факультет |  | И |  |  |
|  |  | шифр |  | наименование |
| Кафедра |  | И9 |  |  |
|  |  | шифр |  | наименование |
| Дисциплина |  |  | | |

КУРСОВАЯ РАБОТА

на тему

|  |
| --- |
|  |
|  |
|  |

Выполнил студент группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Фамилия И.О.

**РУКОВОДИТЕЛЬ**

\_\_ \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Фамилия И.О. Подпись

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

САНКТ-ПЕТЕРБУРГ

2018 г.

# Задание

Оглавление

[Задание 2](#_Toc518233209)

[Введение 4](#_Toc518233210)

[1 Исходные данные и способ их ввода 6](#_Toc518233211)

[2 Описание глобальных данных 8](#_Toc518233212)

[3 Описание пользовательских макроопределений 10](#_Toc518233213)

[3.1 Макроопределения модуля main 10](#_Toc518233214)

[3.2 Макроопределения модуля dict 10](#_Toc518233215)

[3.3 Макроопределения модуля menu 10](#_Toc518233216)

[3.4 Макроопределения модуля input 11](#_Toc518233217)

[3.5 Глобальные макроопределения 12](#_Toc518233218)

[4 Описание пользовательских функции 13](#_Toc518233219)

[4.1 Функции модуля main 13](#_Toc518233220)

[4.2 Функции модуля dict 23](#_Toc518233221)

[4.3 Функции модуля menu 24](#_Toc518233222)

[4.4 Функции модуля input 28](#_Toc518233223)

[4.5 Функции модуля list 30](#_Toc518233224)

[5 Нестандартные функции 36](#_Toc518233225)

[6 Внешние файлы 37](#_Toc518233226)

[7 Диагностические сообщения 38](#_Toc518233227)

[8 Описание порядка работы с программой 42](#_Toc518233228)

[Заключение 45](#_Toc518233229)

[Список литературы 46](#_Toc518233230)

[Приложения 47](#_Toc518233231)

# 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.

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.

# Исходные данные и способ их ввода

**Описание исходных данных.**

* Слово – поле данных символьного типа, количество символов которого не должно превышать 25. Состоит из символов латинского алфавита.
* Часть речи – поле данных символьного типа, количество символов которого не должно превышать 25. Состоит из символов латинского алфавита.
* Перевод – поле данных символьного типа, количество символов которого не должно превышать 25. Состоит из символов русского алфавита.

**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.

# Описание глобальных данных

Область видимости глобальных данных – весь проект.

1. **ITEM\_DEF top\_menu\_items[top\_item\_count] -** массив структурного типа с описаниями элементов верхнего меню**.** Тип – структурный. Объём необходимой памяти – 357 машинных слов.
2. **static ITEM\_DEF exit\_menu\_items[exit\_item\_count] -** массив структурного типа с описаниями элементов меню подтверждения выхода**.** Тип – структурный. Объём необходимой памяти – 51 машинное слово.
3. **char\* main\_headers[main\_column\_count] -** массив заголовков основной таблицы. Тип – символьный. Объём необходимой памяти – 4 машинных слова.
4. **char\* exit\_headers[exit\_column\_count] -** массив заголовков меню подтверждения выхода. Тип – символьный. Объём необходимой памяти – одно машинное слово.
5. **int exit\_code = 0 -** код завершения для функций обратного вызова. Тип – целочисленный. Объём необходимой памяти – одно машинное слово.
6. **int redraw\_main = 1 -** признак перезапуска основной таблицы. Тип – целочисленный. Объём необходимой памяти – одно машинное слово.
7. **int data\_modified = 0 -** признак изменившихся данных. Тип – целочисленный. Объём необходимой памяти – одно машинное слово.
8. **int save\_data = 0 -** признак необходимости сохранения данных. Тип – целочисленный. Объём необходимой памяти – одно машинное слово.
9. **int exit\_canceled = 0 -** признак отмены выхода. Тип – целочисленный. Объём необходимой памяти – одно машинное слово.
10. **int initial\_table\_index = 0 -** индекс для инициализации обновлённого меню. Тип – целочисленный. Объём необходимой памяти – одно машинное слово.
11. **CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo -** буфер для хранения настроек консоли. Тип – структурный. Объём необходимой памяти – 22 байта.
12. **MENU top\_menu -** экземпляр верхнего меню. Тип – структурный. Объём необходимой памяти – 106 байт.
13. **MENU \*ptable = NULL -** указатель на экземпляр основной таблицы. Указатель структурного типа. Объём необходимой памяти – одно машинное слово.

# 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

1. **int Run()**

Назначение: основная рабочая функция программы.

Входные параметры: нет.

Выходные параметры: код завершения.

Используемые функции: malloc(), free(), CharToOemA(), SetConsoleTitle(), GetStdHandle(), GetConsoleScreenBufferInfo(), SetConsoleTextAttribute(), system(),menu\_init(),menu\_active\_color(),menu\_inactive\_color(),menu\_add\_hotkey(),menu\_draw(),menu\_clear(),dict\_load(),list1\_for\_each(),list1\_set\_current\_index(),menu\_add\_hotkey().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 1).

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

Назначение: точка входа программы.

Входные параметры: количество и значения параметров командной строки.

Выходные параметры: код завершения программы.

Используемые функции: Run().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 1).

**Функции обратного вызова.**

1. **int Add(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта верхнего меню «Добавить».

Входные параметры: **menu** – указатель на экземпляр вызывающего меню, **item** – указатель на выделенный элемент меню.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: malloc(), memset(), free(), strcpy(), OemToCharA(), box\_init(), box\_save(), box\_draw(), box\_clear(), list1\_push\_back(), dict\_entry\_new().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 4).

1. **int Edit(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта верхнего меню «Изменить».

Входные параметры: **menu** – указатель на экземпляр вызывающего меню, **item** – указатель на выделенный элемент меню.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: malloc(), memset(), free(), strcpy(), CharToOemA(), OemToCharA(), box\_init(), box\_save(), box\_draw(), box\_clear(), list1\_push\_back(), dict\_entry\_new().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 4).

1. **int DelYes(MENU\* menu, ITEM\* item)**

Назначение: обработчик положительного ответа перед удалением записи.

Входные параметры: menu – указатель на экземпляр вызывающего меню; item – указатель на выделенный элемент меню.

Выходные параметры: возвращает -1 для закрытия меню подтверждения удаления записи.

Используемые функции: нет.

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 3).

1. **int DelNo(MENU\* menu, ITEM\* item)**

Назначение: обработчик отрицательного ответа перед удалением записи.

Входные параметры: menu – указатель на экземпляр вызывающего меню; item – указатель на выделенный элемент меню.

Выходные параметры: возвращает -1 для закрытия меню подтверждения удаления записи.

Используемые функции: нет.

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 3).

1. **int Delete(MENU\* menu)**

Назначение: обработчик пункта верхнего меню «Удалить».

Входные параметры: **menu** – указатель на экземпляр вызывающего меню.

Выходные параметры: возвращает -1 для закрытия меню.

Используемые функции: malloc(), memset(), free(), CharToOemA(), menu\_init(), menu\_active\_color(), menu\_inactive\_color(), menu\_add\_hotkey(), menu\_draw(), menu\_clear(), menu\_del\_curr(), menu\_fill\_wnd(), menu\_cls().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 3).

1. **int Search(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта верхнего меню «Поиск».

Входные параметры: **menu** – указатель на экземпляр вызывающего меню, **item** – указатель на выделенный элемент меню.

Выходные параметры: возвращает -1 для закрытия меню.

Используемые функции: malloc(), memset(), free(), strcpy(), OemToCharA(), atoi(), box\_init(), box\_save(), box\_draw(), box\_clear(), ShowMenu(), menu\_fill\_wnd(), menu\_cls().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 6).

1. **int Sort(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта верхнего меню «Сортировка».

Входные параметры: **menu** – указатель на экземпляр вызывающего меню, **item** – указатель на выделенный элемент меню.

Выходные параметры: возвращает -1 для закрытия меню.

Используемые функции: malloc(), memset(), strncpy(), ShowMenu().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 5).

1. **int Save(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта верхнего меню «Сохранить».

Входные параметры: **menu** – указатель на экземпляр вызывающего меню, **item** – указатель на выделенный элемент меню.

Выходные параметры: возвращает -1 для закрытия меню.

Используемые функции: dict\_save().

Диагностические сообщения: нет.

1. **int Help(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта верхнего меню «Справка».

Входные параметры: menu – указатель на экземпляр вызывающего меню, item – указатель на выделенный элемент меню.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: HelpFromFile().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» №№ 9-11).

1. **int Exit(MENU\* menu, ITEM\* item)**»

Назначение: обработчик пункта верхнего меню «Выйти».

Входные параметры: menu – указатель на экземпляр вызывающего меню, item – указатель на выделенный элемент меню.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: menu\_init(), menu\_active\_color(), menu\_inactive\_color(), menu\_add\_hotkey(), menu\_draw(), menu\_clear(), Save().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 8).

1. **int ExitYes(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта подтверждения выхода «Да» (выход с сохранением данных).

Входные параметры: **menu** – указатель на экземпляр вызывающего меню, **item** – указатель на выделенный элемент меню.

Выходные параметры: возвращает -1 для закрытия меню.

Используемые функции: нет.

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 8).

1. **int ExitNo(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта подтверждения выхода «Нет» (выход без сохранения данных).

Входные параметры: **menu** – указатель на экземпляр вызывающего меню, **item** – указатель на выделенный элемент меню.

Выходные параметры: возвращает -1 для закрытия меню.

Используемые функции: нет.

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 8).

1. **int ExitCancel(MENU\* menu, ITEM\* item)**

Назначение: обработчик пункта подтверждения выхода «Отмена».

Входные параметры: **menu** – указатель на экземпляр вызывающего меню, **item** – указатель на выделенный элемент меню.

Выходные параметры: возвращает -1 для закрытия меню.

Используемые функции: нет.

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 8).

Обработчики горячих клавиш:

1. **int DefaultESC(MENU\* menu)**

Назначение: обработчик клавиши Esc по умолчанию, устанавливается для большинства меню и производит закрытие текущего меню без дополнительных действий.

Входные параметры: **menu** – указатель на экземпляр вызывающего меню.

Выходные параметры: возвращает -1 для закрытия меню.

Используемые функции: нет.

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 8).

1. **int ESC(MENU\* menu)**

Назначение: обработчик обработчик клавиши Esc для главного и верхнего меню, проверяет необходимость сохранения данных перед выходом.

Входные параметры: **menu** – указатель на экземпляр вызывающего меню.

Выходные параметры: возвращает значение, полученное из вызова Exit().

Используемые функции: Exit().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 8).

1. **int F1(MENU\* menu)**

Назначение: обработчик горячей клавиши «F1» (Справка).

Входные параметры: **menu** – указатель на экземпляр вызывающего меню.

Выходные параметры: возвращает 0.

Используемые функции: Help().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» №№ 9-11).

1. **int F2(MENU\* menu)**

Назначение: обработчик горячей клавиши «F2» (Сохранение).

Входные параметры: **menu** – указатель на экземпляр вызывающего меню.

Выходные параметры: возвращает значение глобальной переменной exit\_code.

Используемые функции: Save.

Диагностические сообщения: нет.

1. **int F3(MENU\* menu)**

Назначение: обработчик горячей клавиши «F3» (Поиск).

Входные параметры: **menu** – указатель на экземпляр вызывающего меню.

Выходные параметры: возвращает значение глобальной переменной exit\_code.

Используемые функции: Search().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 6).

1. **int F4(MENU\* menu)**

Назначение: обработчик горячей клавиши «F4» (Редактирование).

Входные параметры: **menu** – указатель на экземпляр вызывающего меню.

Выходные параметры: возвращает значение глобальной переменной exit\_code.

Используемые функции: Edit().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 4).

1. **int F9(MENU\* menu)**

Назначение: обработчик горячей клавиши «F9» (Переход к верхнему меню).

Входные параметры: **menu** – указатель на экземпляр вызывающего меню.

Выходные параметры: возвращает значение глобальной переменной exit\_code.

Используемые функции: menu\_draw().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» № 2).

Функции обратного вызова для односвязного списка:

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

Назначение: обработчик прокрутки меню на один элемент вперёд или назад.

Входные параметры: **menu** – указатель на экземпляр вызывающего меню; **direction** – целочисленная переменная, принимающая 0, если выделен следующий элемент и 1, если выделен предыдущий элемент; **wrap** – целочисленная переменная, принимающая 1, если при достижении конца меню выделяется элемент с противоположного конца и 0 в противоположном случае.

Выходные параметры: нет.

Используемые функции: list1\_curr\_rev(), list1\_curr\_fwd().

Диагностические сообщения: нет.

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

Назначение: перенос очередного элемента словаря в элемент описания пункта меню (используется для отображения словаря в виде таблицы).

Входные параметры: data – указатель на структуру элемента словаря; index – целочисленная переменная, принимающая значение текущего индекса элемента; param – указатель на структуру, описывающую элемент меню.

Выходные параметры: возвращает 1 для продолжения перебора элементов списка.

Используемые функции: CharToOemA(), strlen(), itoa().

Диагностические сообщения: нет.

1. **int check\_entry\_display (void\* data, int index, void\* param)**

Назначение: перенос подходящего элемента словаря в элемент описания пункта меню (используется для поиска).

Входные параметры: data – указатель на структуру элемента словаря; index – целочисленная переменная, принимающая значение текущего индекса поля данных словаря (0 – «Слово», 1 – «Часть речи», 1 – «Перевод»); param – указатель на структуру SearchCriteria, содержащую параметры поиска и указатель на структуру, описывающую элемент меню.

Выходные параметры: возвращает 1 для продолжения перебора элементов списка.

Используемые функции: CharToOemA(), strlen(), strcmp().

Диагностические сообщения: нет.

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

Назначение: сравнение двух элементов словаря (функция-компаратор).

Входные параметры: data1 и data2 – указатели на структуры двух сравниваемых элементов словаря; указатель на целочисленную переменную, содержащую индекс поля в записи словаря (0 – «Слово», 1 – «Часть речи», 1 – «Перевод»).

Выходные параметры: возвращает 1 для продолжения перебора элементов списка.

Используемые функции: strlen(), strcmp().

Диагностические сообщения: нет.

**Вспомогательные функции.**

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

Назначение: отображение текста справки из файла на всплывающем окне.

Входные параметры: hStdOut – дескриптор потока вывода; file\_name – имя файла для загрузки справочной информации; title – заголовок текстового окна; rect – структура с координатами окна.

Выходные параметры: возвращает значение, полученное в пезультате вызова функции ShowMenu().

Используемые функции: MenuItemsFromFile(), ShowMenu().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» №№ 9-11).

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

Назначение: отображение упрощённого вертикального меню в один столбец.

Входные параметры: hStdOut – дескриптор потока вывода; menu\_items – массив структур, описывающих элементы меню; item\_count – количество элементов меню в массиве; title – заголовок текстового окна; rect – структура с координатами окна; flags – битовая маска, режим отображения меню; f1CB – указатель на функцию-обработчик клавиши F1.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: malloc(), memset(), free(), menu\_init(), menu\_active\_color(), menu\_inactive\_color(), menu\_add\_hotkey(), menu\_draw(), menu\_clear().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» №№ 7, 9-11).

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

Назначение: загрузка пунктов меню из файла.

Входные параметры: file\_name – имя файла для загрузки пунктов меню; max\_len – объём выделяемой под строку меню памяти; max\_count – максимальное количество пунктов меню.

Выходные параметры: pcount – указатель на целочисленную переменную, в которую записывается число скопированных элементов; в случае успеха возвращает указатель структурного типа на выделенную под массив элементов память, в которую записаны содержащиеся в файле элементы меню (строки), в случае ошибки – -1.

Используемые функции: malloc(), memset(), free(), fopen(), fgets(), fclose(), strlen().

Диагностические сообщения: есть (см. раздел «Диагностические сообщения и действия оператора» №№ 7, 9-11).

## Функции модуля dict

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

Назначение: создание нового элемента словаря.

Входные параметры: word\_eng – указатель на строку, содержащую английское слово; word\_part – указатель на строку, содержащую часть речи; word\_rus – указатель на строку, содержащую русский перевод.

Выходные параметры: возвращает указатель на структуру созданного элемента словаря.

Используемые функции: malloc(), memset(), strcpy().

Диагностические сообщения: нет.

1. **int dict\_load(const char\* file\_name)**

Назначение: загрузка словаря из файла.

Входные параметры: file\_name – имя файла, содержащего базу данных англо-русского словаря.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: file\_load(), memset(), malloc(), free(), strcpy(), list1\_push\_back(), dict\_entry\_new().

Диагностические сообщения: нет.

1. **int dict\_entry\_save(void\* data, int index, void\* param)**

Назначение: сохранение одного элемента словаря в файл.

Входные параметры: data – указатель структурного типа на элемент словаря; index – целочисленная переменная, содержащая индекс сохраняемого элемента; param – указатель структурного типа на открытый файл назначения.

Выходные параметры: возвращает 1 для продолжения перебора элементов словаря.

Используемые функции: fwrite().

Диагностические сообщения: нет.

1. **int dict\_save(const char\* file\_name)**

Назначение: сохранение словаря в файл.

Входные параметры: file\_name – имя файла, содержащего базу данных англо-русского словаря.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: fopen(), fclose(), list1\_for\_each(), dict\_entry\_save().

Диагностические сообщения: нет.

## Функции модуля menu

1. 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[])

Назначение: инициализация экземпляра меню.

Входные параметры: menu – указатель структурного типа на экземпляр меню; parent – указатель структурного типа на экземпляр родительского меню (перекрываемого дочерним); hstdout – дескриптор потока вывода; item\_defs – указатель на массив структур, описывающих элементы меню; item\_count – целочисленная переменная, содержащая количество элементов меню; cell\_count – целочисленная переменная, содержащая количество стоблцов табличного меню; orient – целочисленная переменная, содержащая 0, если меню ориентировано горизонтально и 1, если вертикально; prect – указатель на структуру, содержащую координаты окна меню; border count – целочисленная переменная, содержащая 1, если меню нужно рисовать с рамкой и 0 в противоположном случае; headers – указатель структурного типа на массив заголовков меню.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: malloc(), memset(), memcpy(), free(), strlen(), list1\_init(), list1\_push\_back(), list1\_for\_each(), menu\_fill\_wnd(), CharToOemA(), GetConsoleScreenBufferInfo().

Диагностические сообщения: нет.

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

Назначение: добавление обработчика горячей клавиши.

Входные параметры: menu – указатель структурного типа на экземпляр меню; code – целочисленная переменная, содержащая код «горячей» клавиши; cb – указатель на функцию обратного вызова.

Выходные параметры: возвращает новое количество обработчиков.

Используемые функции: malloc(), memset(), list1\_push\_front().

Диагностические сообщения: нет.

1. void menu\_clear(MENU\* menu)

Назначение: очистка экземпляра меню.

Входные параметры: menu – указатель структурного типа на экземпляр меню.

Выходные параметры: нет.

Используемые функции: list1\_clear(), free().

Диагностические сообщения: нет.

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

Назначение: заполнение фона окна меню.

Входные параметры: menu – указатель структурного типа на экземпляр меню; items\_count – целочисленная переменная, содержащая количество элементов меню.

Выходные параметры: нет.

Используемые функции: memset().

Диагностические сообщения: нет.

1. int menu\_draw(MENU\* menu, int flags)

Назначение: основная оконная функция меню.

Входные параметры: menu – указатель структурного типа на экземпляр меню; – целочисленная переменная, содержащая битовые флаги, задающие режим работы меню.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: SetConsoleTextAttribute(), kbhit(), printf(), fflush(), menu\_draw(), menu\_gotoxy(), menu\_next(), menu\_prev(), menu\_cls(), itemMenu(), list1\_for\_each(), list1\_curr(), check\_cb\_retcode().

Диагностические сообщения: нет.

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

Назначение: настройка отображения текущего элемента меню.

Входные параметры: menu – указатель структурного типа на экземпляр меню; attr – целочисленная переменная, содержащая цветовые атрибуты для текущего элемента меню.

Выходные параметры: нет.

Используемые функции: нет.

Диагностические сообщения: нет.

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

Назначение: настройка отображения неактивных элементов меню.

Входные параметры: menu – указатель структурного типа на экземпляр меню; указатель структурного типа на экземпляр меню; attr – целочисленная переменная, содержащая цветовые атрибуты для неактивных элементов меню.

Выходные параметры: нет.

Используемые функции: нет.

Диагностические сообщения: нет.

1. void menu\_prev(MENU\* menu)

Назначение: выделение предыдущего пункта.

Входные параметры: menu – указатель структурного типа на экземпляр меню.

Выходные параметры: нет.

Используемые функции: list1\_curr\_rev(), itemMenu(), showCursor().

Диагностические сообщения: нет.

1. void menu\_next(MENU\* menu)

Назначение: выделение следующего пункта.

Входные параметры: menu – указатель структурного типа на экземпляр меню.

Выходные параметры: нет.

Используемые функции: list1\_curr\_rev(), itemMenu(), showCursor().

Диагностические сообщения: нет.

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

Назначение: перемещение курсора по координатам x, y.

Входные параметры: menu – указатель структурного типа на экземпляр меню; x и y целочисленные переменные, содержащие текстовые координаты курсора.

Выходные параметры: нет.

Используемые функции: SetConsoleCursorPosition().

Диагностические сообщения: нет.

1. void itemMenu(MENU\* menu, int activate)

Назначение: выделение пункта меню.

Входные параметры: menu – указатель структурного типа на экземпляр меню; activate – целочисленная переменная, содержащая 1, если нужно выделить пункт меню и 0, если нужно снять выделение.

Выходные параметры: нет.

Используемые функции: SetConsoleTextAttribute(), printf(), list1\_curr(), menu\_gotoxy().

Диагностические сообщения: нет.

1. void menu\_cls(MENU\* menu)

Назначение: отображение фонового окна меню.

Входные параметры: menu – указатель структурного типа на экземпляр меню;.

Выходные параметры: нет.

Используемые функции: SetConsoleTextAttribute(), printf(), list1\_curr(), menu\_gotoxy().

Диагностические сообщения: нет.

1. void showCursor(MENU\* menu, int visible)

Назначение: показать/убрать курсор.

Входные параметры: menu – указатель структурного типа на экземпляр меню; visible – целочисленная переменная, содержащая 1, если нужно показать курсор и 0, если нужно убрать.

Выходные параметры: нет.

Используемые функции: SetConsoleCursorInfo.

Диагностические сообщения: нет.

1. void menu\_del\_curr(MENU\* menu)

Назначение: удалить текущий элемент.

Входные параметры: menu – указатель структурного типа на экземпляр меню.

Выходные параметры: нет.

Используемые функции: list1\_erase\_current().

Диагностические сообщения: нет.

## Функции модуля input

1. int box\_init(InputBox\* box, HANDLE handle, SMALL\_RECT rect, char\*\*\* contents, int row\_count)

Назначение: инициализация экземпляра окна ввода.

Входные параметры: box – указатель структурного типа на экземпляр окна ввода; handle – системный дескриптор стандартного потока ввода, rect – структура с координатами окна ввода; contents – указатель третьего порядка на подписи полей данных и их первоначальное начальное содержимое (указатель на матрицу строк размером row\_count x 2); row\_count – целочисленная переменная, содержащая количество полей ввода данных.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: malloc(), memset(), free().

Диагностические сообщения: нет.

1. void box\_clear(InputBox\* box)

Назначение: очистка полей экземпляра окна ввода.

Входные параметры: box – указатель структурного типа на экземпляр окна ввода.

Выходные параметры: нет.

Используемые функции: free().

Диагностические сообщения: нет.

1. int box\_save(InputBox\* box)

Назначение: сохранить прямоугольник на экране.

Входные параметры: box – указатель структурного типа на экземпляр окна ввода.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: ReadConsoleOutput().

Диагностические сообщения: нет.

1. int box\_draw(InputBox\* box)

Назначение: нарисовать окно ввода и запустить цикл обработки нажатия клавиш.

Входные параметры: box – указатель структурного типа на экземпляр окна ввода.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: SetConsoleTextAttribute(), malloc(), memset(), free(), printf(), kbhit(), \_getch(), strlen(), box\_gotoxy(), showCursor(), accept\_input(), box\_restore(), box\_clear().

Диагностические сообщения: нет.

1. int box\_restore(InputBox\* box)

Назначение: восстановить прямоугольник на экране.

Входные параметры: box – указатель структурного типа на экземпляр окна ввода.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: WriteConsoleOutput().

Диагностические сообщения: нет.

## Функции модуля list

1. LIST1\* list1\_new()

Назначение: создание списка.

Входные параметры: нет.

Выходные параметры: в случае успеха возвращает указатель структурного типа на созданный экземпляр списка, в случае ошибки – NULL.

Используемые функции: malloc(), list1\_init().

Диагностические сообщения: нет.

1. int list1\_delete(LIST1\* list)

Назначение: удаление списка.

Входные параметры: list – указатель структурного типа на экземпляр списка.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: list1\_clear(), free().

Диагностические сообщения: нет.

1. int list1\_init(LIST1\* list)

Назначение: проинициализировать экземпляр списка.

Входные параметры: list – указатель структурного типа на экземпляр списка.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: memset().

Диагностические сообщения: нет.

1. int list1\_clear(LIST1\* list)

Назначение: очистить поля экземпляра списка.

Входные параметры: list – указатель структурного типа на экземпляр списка.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: delete\_item(), memset().

Диагностические сообщения: нет.

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

Назначение: добавить элемент в начало списка.

Входные параметры: list – указатель структурного типа на экземпляр списка; data – указатель на размещаемые в списке данные.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: create\_item().

Диагностические сообщения: нет.

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

Назначение: добавить элемент в конец списка.

Входные параметры: list – указатель структурного типа на экземпляр списка; data – указатель на размещаемые в списке данные.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: create\_item().

Диагностические сообщения: нет.

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

Назначение: перебор всех элементов с использованием функции обратного вызова.

Входные параметры: list – указатель структурного типа на экземпляр списка; find – указатель на функцию обратного вызова, param – указатель на пользовательские данные для передачи в функцию обратного вызова.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: нет.

Диагностические сообщения: нет.

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

Назначение: сместить текущий элемент на одну позицию вперёд.

Входные параметры: list – указатель структурного типа на экземпляр списка; wrap – целочисленная переменная, содержащая 1, если при достижении конца списка нужно перейти в начало и 0 в противном случае.

Выходные параметры: возвращает указатель на пользовательские данные, хранимые в текущем элементе списка.

Используемые функции: нет.

Диагностические сообщения: нет.

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

Назначение: сместить текущий элемент на одну позицию назад.

Входные параметры: list – указатель структурного типа на экземпляр списка; wrap – целочисленная переменная, содержащая 1, если при достижении начала списка нужно перейти в конец и 0 в противном случае.

Выходные параметры: возвращает указатель на пользовательские данные, хранимые в текущем элементе списка.

Используемые функции: нет.

Диагностические сообщения: нет.

1. void\* list1\_front(LIST1\* list)

Назначение: доступ к данным первого элемента.

Входные параметры: list – указатель структурного типа на экземпляр списка.

Выходные параметры: возвращает указатель на пользовательские данные, хранимые в первом элементе списка.

Используемые функции: нет.

Диагностические сообщения: нет.

1. void\* list1\_back(LIST1\* list)

Назначение: доступ к данным последнего элемента.

Входные параметры: list – указатель структурного типа на экземпляр списка.

Выходные параметры: возвращает указатель на пользовательские данные, хранимые в последнем элементе списка.

Используемые функции: нет.

Диагностические сообщения: нет.

1. void\* list1\_curr(LIST1\* list)

Назначение: доступ к данным текущего элемента.

Входные параметры: list – указатель структурного типа на экземпляр списка.

Выходные параметры: возвращает указатель на пользовательские данные, хранимые в текущем элементе списка.

Используемые функции: нет.

Диагностические сообщения: нет.

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

Назначение: удаление элементов, удовлетворяющих условиям функции обратного вызова.

Входные параметры: list – указатель структурного типа на экземпляр списка; cb – указатель на функцию обратного вызова, param – указатель на пользовательские данные для передачи в функцию обратного вызова.

Выходные параметры: возвращает указатель на пользовательские данные, хранимые в текущем элементе списка.

Используемые функции: delete\_item().

Диагностические сообщения: нет.

1. void\* list1\_erase\_current(LIST1\* list)

Назначение: удаление текущего элемента.

Входные параметры: list – указатель структурного типа на экземпляр списка.

Выходные параметры: возвращает указатель на пользовательские данные, хранимые в текущем элементе списка.

Используемые функции: delete\_item().

Диагностические сообщения: нет.

1. int list1\_get\_current\_index(LIST1\* list)

Назначение: получение порядкового номера текущего элемента.

Входные параметры: list – указатель структурного типа на экземпляр списка.

Выходные параметры: возвращает целочисленное значение, равное индексу текущего элемента.

Используемые функции: нет.

Диагностические сообщения: нет.

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

Назначение: установка текущего элемента по индексу.

Входные параметры: list – указатель структурного типа на экземпляр списка; index – целочисленная переменная, содержащая индекс элемента, который нужно сделать текущим.

Выходные параметры: в случае успешного завершения возвращает 0, в случае ошибки – 1.

Используемые функции: нет.

Диагностические сообщения: нет.

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

Назначение: поиск элемента с использованием функции обратного вызова.

Входные параметры: list – указатель структурного типа на экземпляр списка; check – указатель на функцию обратного вызова, param – указатель на пользовательские данные для передачи в функцию обратного вызова.

Выходные параметры: возвращает целочисленное значение, равное индексу найденного элемента.

Используемые функции: нет.

Диагностические сообщения: нет.

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

Назначение: сортировка списка с использованием функции обратного вызова.

Входные параметры: list – указатель структурного типа на экземпляр списка; compare – указатель на функцию обратного вызова, param – указатель на пользовательские данные для передачи в функцию обратного вызова.

Выходные параметры: нет.

Используемые функции: mergeSort().

Диагностические сообщения: нет.

# 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:

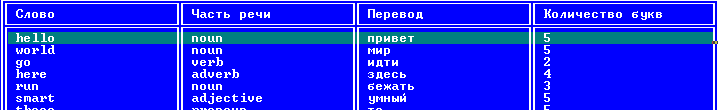
1. dictionary.dat – файл, хранящий содержимое словаря;
2. help.txt – файл с текстом общей справки;
3. sort.txt – файл с текстом контекстной справки о функции сортировки.
4. find.txt – файл с текстом контекстной справки о функции поиска.

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

Ожидаемая кодировка тестовых файлов справки – CP866.

# Диагностические сообщения

1. Диагностическое сообщение:



Действия оператора:

Для перемещения используются клавиши «↑», «↓».

Вызов общей справки – «F1».

Сохранение данных – «F2».

Поиск записей – «F3».

Редактирование – клавиши «F4» и «Enter».

Переход в верхнее меню – клавиша «F9».

Выход из программы – клавиша «Esc».

1. Диагностическое сообщение:

2.png

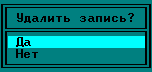
Действия оператора:

Для перемещения используются клавиши «←» и «→».

Выбор – клавиша «Enter».

Возврат в главное меню – клавиша «Esc».

1. Диагностическое сообщение:



Действия оператора:

Для перемещения используются клавиши «↑», «↓».

Подтверждение – клавиша «Enter».

1. Диагностическое сообщение:

edit_menu.png

Действия оператора:

Переход к следующему полю – клавиша «Tab».

Перемещение по символам редактируемого поля – стрелки «←» и «→».

Переход в начало строки – клавиша «Home».

Переход к концу строки – клавиша «End».

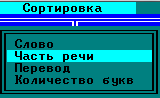
Удаление символа слева от курсора – клавиша «Backspace».

Удаление символа справа от курсора – клавиша «Delete».

Подтверждение ввода – клавиша «Enter».

Возврат в главное меню – клавиша «Esc».

1. Диагностическое сообщение:



Действия оператора:

Для перемещения используются клавиши «↑», «↓».

Подтверждение – клавиша «Enter».

Вызов контекстной справки – клавиша «F1».

1. Диагностическое сообщение:

search_menu.png

Действия оператора:

Переход к следующему полю – клавиша «Tab».

Перемещение по символам редактируемого поля – стрелки «←» и «→».

Переход в начало строки – клавиша «Home».

Переход к концу строки – клавиша «End».

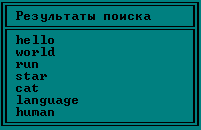
Удаление символа слева от курсора – клавиша «Backspace».

Удаление символа справа от курсора – клавиша «Delete».

Подтверждение ввода – клавиша «Enter».

Возврат в главное меню – клавиша «Esc».

1. Диагностическое сообщение:

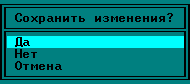


Действия оператора:

Вызов контекстной справки – клавиша «F1».

Возврат в главное меню – клавиша «Esc».

1. Диагностическое сообщение:



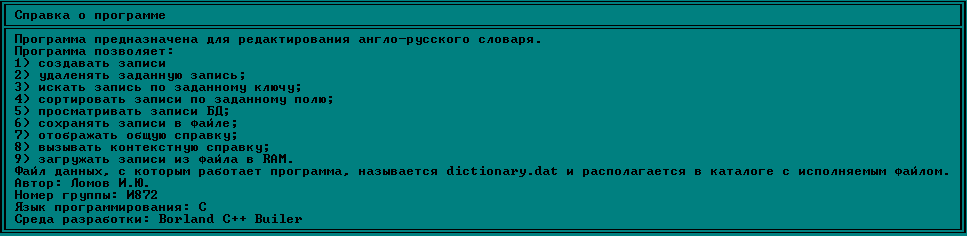
Действия оператора:

Для перемещения используются клавиши «↑», «↓».

Подтверждение – клавиша «Enter».

Возврат в главное меню – клавиша «Esc».

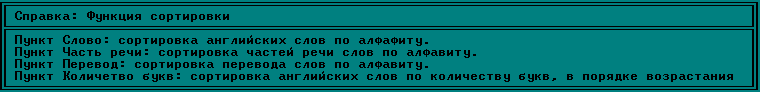
1. Диагностическое сообщение:



Действия оператора:

Возврат в главное меню – клавиши «Esc» или «F1».

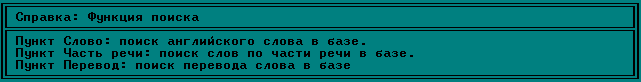
1. Диагностическое сообщение:



Действия оператора:

Возврат в главное меню – клавиши «Esc» или «F1».

1. Диагностическое сообщение:



Действия оператора:

Возврат в главное меню – клавиши «Esc» или «F1».

# 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.

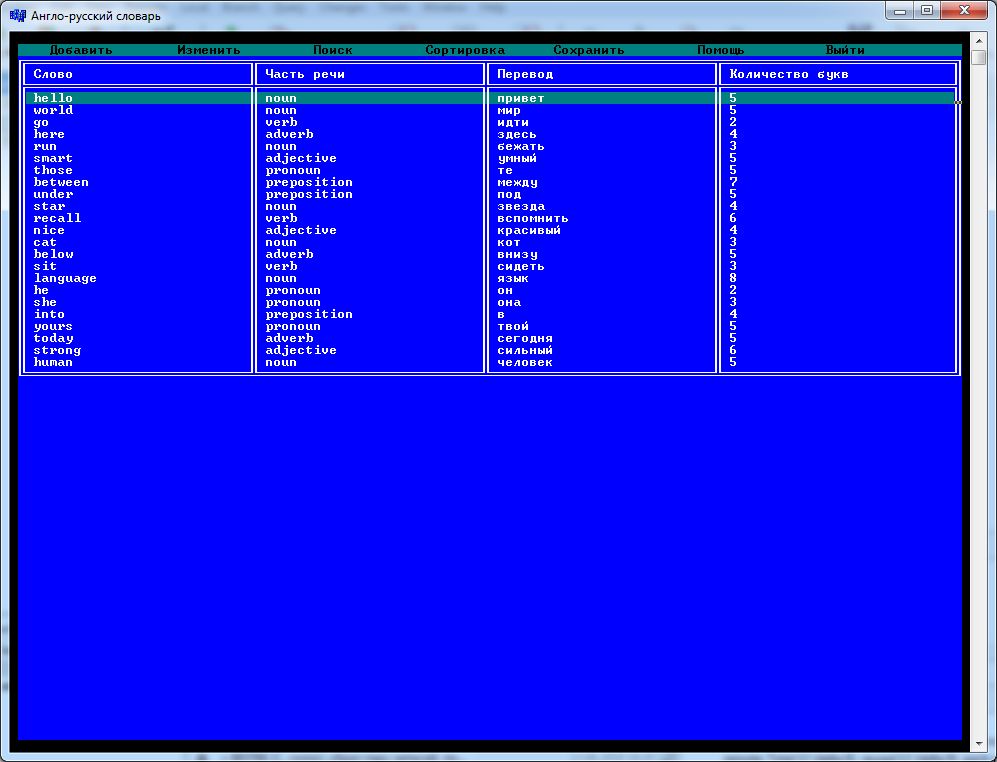


Рисунок 1 – Главное окно программы

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.

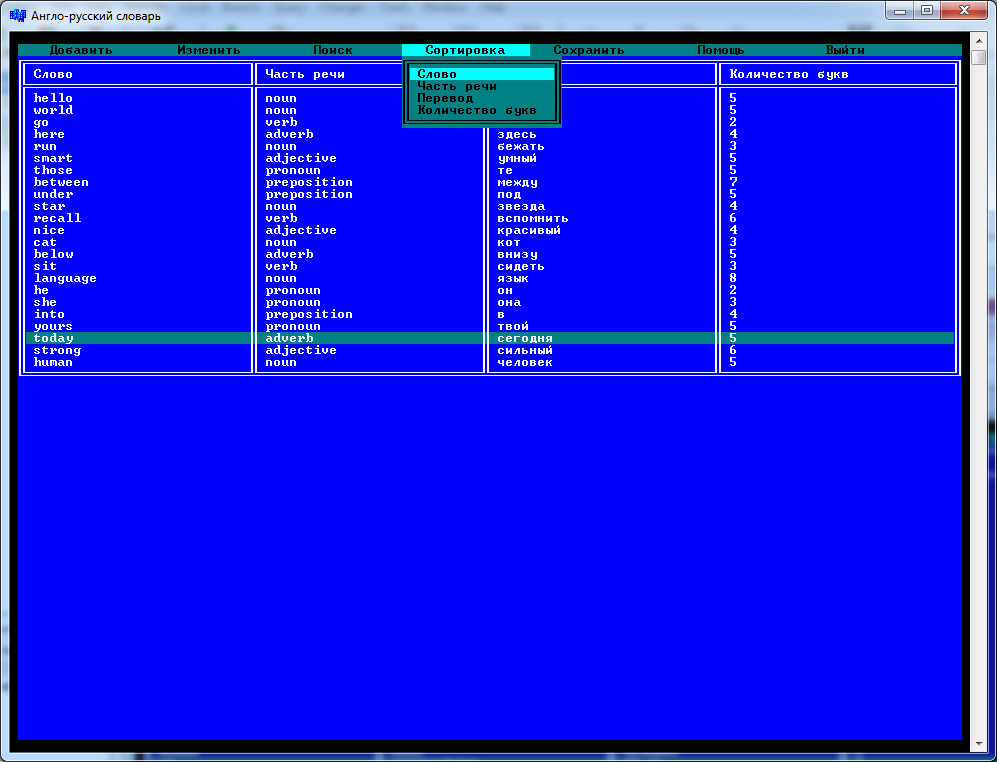


Рисунок 1 – Вызов функции сортировки

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

**common.h**

#ifndef \_\_MODULE\_HEADER\_\_

#define \_\_MODULE\_HEADER\_\_

#ifdef \_\_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\_CODES\_H\_\_\*/

#ifdef \_\_LIST1\_H\_\_

struct LIST1\_ITEM;

// структура элемента односвязного списка

typedef struct LIST1\_ITEM {

struct LIST1\_ITEM\* next;

void\* data;

} LIST1\_ITEM;

// односвязный список

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

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

// если вернёт 0, обработка прекратится на этой итерации

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

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

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

// структура односвязного списка

typedef struct LIST1 {

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

LIST\_ITEM\_FREE item\_free;

int count;

} LIST1;

//

LIST1\* list1\_new();

//

int list1\_delete(LIST1\* list);

//

int list1\_init(LIST1\* list);

//

int list1\_clear(LIST1\* list);

//

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

//

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

//

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

// Прокрутить текущий указатель вперёд

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

// Прокрутить текущий указатель назад

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

//

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

//

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

//

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

// Удалить элемент списка

// возвращает текущий элемент (возможно, новый)

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

// Удалить текущий элемент списка

// возвращает новый текущий элемент

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

//

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

//

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

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

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

// сортировка списка

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

#endif /\*\_\_LIST1\_H\_\_\*/

#ifdef \_\_DICT\_H\_\_

#define MAX\_DICT\_STR\_LEN 32

#define DICT\_FLD\_CNT 3

// структура записи словаря

typedef struct DICT\_ENTRY {

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

} DICT\_ENTRY;

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

struct MENU;

//

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

//

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

//

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

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

extern const char\* dict\_file\_name;

extern LIST1 dict;

#endif /\*\_\_DICT\_H\_\_\*/

#ifdef \_\_MENU\_H\_\_

struct MENU;

#define MENU\_ORIENT\_HORZ 0

#define MENU\_ORIENT\_VERT 1

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

// Этапы частичной отрисовки меню

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

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

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

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

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

// полноценный запуск окна меню с циклом сообщений

#define MENU\_FULL\_FLAGS 0x1F

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

#define MAX\_CELLS 50

// Направление перемещение курсора

#define MENU\_CURR\_FWD 0 // вперёд

#define MENU\_CURR\_REV 1 // назад

struct ITEM;

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

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

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

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

// Структура определения элемента меню

typedef struct {

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

ExecuteCurrentCB cb; // Функция, привязанная к пункту меню

} ITEM\_DEF;

// Структура для хранения элемента меню

typedef struct ITEM {

int x, y; // Столбец и строка консоли

int index;

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

ExecuteCurrentCB cb; // Функция, привязанная к пункту меню

} ITEM;

// Область окна меню

typedef struct {

SMALL\_RECT rect; // координаты углов консольного окна

int M, N; // размер матрицы символов

char\*\* m; // матрица символов

} MENU\_WND;

// Горячая клавиша

typedef struct {

int code;

ExecuteHotketCB cb; // обработчик

} HOT\_KEY;

// Структура для всего меню

typedef struct MENU {

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

struct MENU\* parent;

// Глобальные переменные, используемые в функциях меню

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

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

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

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

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

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

MENU\_WND wnd;

int cb\_retcode;

int last\_key;

LIST1 hk\_list;

CurrentChangedCB changed\_cb;

char\* hdr;

LIST1 items;

int cell\_count; // количество ячеек

int cell\_width; // ширина ячейки

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

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

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

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

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

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

int border\_bottom\_index;

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

int left\_pad;

int top\_pad;

// пользовательская метка

int user\_tag;

} MENU;

// Инициализация полей экземпляра меню

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

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

// Добавить обработчик горячей клавиши

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

// Очистка полей экземпляра меню

void menu\_clear(MENU\* menu);

// Прорисовка фона и рамки

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

// Прорисовка меню

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

// Установка цветовых атрибутов активного пункта

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

// Установка цветовых атрибутов неактивного пункта

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

// Выделить предыдущий пункт меню

void menu\_prev(MENU\* menu);

// Выделить следующий пункт меню

void menu\_next(MENU\* menu);

// Перевод курсора в точку x, y

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

// Выделить пункт меню

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

// Очистка окна

void menu\_cls(MENU\* menu);

// Сохранить координаты курсора в переменную curspos

void showCursor(MENU\* menu, int visible); // скрыть/показать курсор

// Удалить текущий элемент

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

#endif /\*\_\_MENU\_H\_\_\*/

#ifdef \_\_DICTIONARY\_INPUT\_DLG\_H\_\_

#define MAX\_TITLE 50

#define TITLE 0

#define BUFFER 1

#define COLUMNS 2

typedef struct InputBox {

HANDLE handle;

SMALL\_RECT rect;

CHAR\_INFO\* bak;

CHAR\_INFO\* wnd;

COORD size; // width, height

WORD edit\_attr;

int max\_width;

// буфер ввода

//char\* buffer;

// 2-мерный массив строк: 1я колонка - надписи

// 2я колонка - редактируемые поля (можно передавать начальные значения)

char\*\*\* contents;

int row\_count; // количество полей (пар надпись/значение)

int row; // текущее редактируемое поле

} InputBox;

int box\_init(InputBox\* box, HANDLE handle, SMALL\_RECT rect, char\*\*\* contents, int row\_count);

void box\_clear(InputBox\* box);

int box\_save(InputBox\* box);

int box\_draw(InputBox\* box);

int box\_restore(InputBox\* box);

#endif /\*\_\_DICTIONARY\_INPUT\_DLG\_H\_\_\*/

#endif /\*\_\_MODULE\_HEADER\_\_\*/

**main.c**

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

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#pragma hdrstop

#include <conio.h>

#define \_\_DICT\_H\_\_

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

#define \_\_MENU\_H\_\_

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

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

#define \_\_DICTIONARY\_INPUT\_DLG\_H\_\_

#include "common.h"

//#include "menu.h"

//#include "dict.h"

//#include "codes.h"

//#include "list.h"

//#include "input.h"

#pragma argsused

// Критерий поиска

typedef struct SearchCriteria {

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

int index;

// фильтр - максимальное значение длины строки Слова

int value;

// строка поиска (для первых трёх полей)

const char\* str;

// указатель на массив пунктов меню

ITEM\_DEF\* menu\_items;

// количество пунктов

int item\_count;

} SearchCriteria;

// Предварительное объявление функций

// Основная рабочая функция программы

int Run();

// функции обратного вызова

// Функции, которые подключаются к пунктам меню

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

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

int Delete(MENU\* menu);

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

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

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

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

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

// Функции, которые подключаются к горячим клавишам

// Выход - сохранить данные

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

// Выход - сохранить данные

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

// Выход - отмена

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

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

int DefaultESC(MENU\* menu);

// Обработчик клавиши ESC

int ESC(MENU\* menu);

// Обработчик клавиши F1 (Справка)

int F1(MENU\* menu);

// Обработчик клавиши F2 (Сохранить)

int F2(MENU\* menu);

// Обработчик клавиши F3 (Поиск)

int F3(MENU\* menu);

// Обработчик клавиши F4 (Редактирование)

int F4(MENU\* menu);

// Обработчик клавиши F9 (Переход на верхнее меню)

int F9(MENU\* menu);

// Обработчик прокрутки меню на один элемент вперёд или назад

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

// Перенос очередного элемента словаря в элемент описания пункта меню

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

// Проверка элемента словаря на соответствие критерию поиска

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

// Сравнение двух элементов словаря

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

// вспомогательные функции

// Отображение текста справки из файла на всплывающем окне

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

// Отображение упрощённого вертикального меню в один столбец

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

// Загрузка пунктов меню из файла

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

// максимально допустимая длина строки

#define MAX\_STRING 256

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

#define MENU\_TAG\_DEFAULT 0

#define MENU\_TAG\_TOP 1

#define MENU\_TAG\_MAIN 2

#define MENU\_TAG\_EXIT 3

#define MENU\_TAG\_HELP\_GLOBAL 4

#define MENU\_TAG\_SORT 5

#define MENU\_TAG\_FIND 6

// количество пунктов верхнего меню

#define top\_item\_count 7

// количество ячеек (столбцов) основного табличного меню

#define main\_column\_count 4

// количество ячеек (столбцов) меню подтверждения выхода

#define exit\_column\_count 1

// количество пунктов меню подтверждения выхода

#define exit\_item\_count 3

// массив с описаниями элементов верхнего меню

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

{ { "Добавить", 0 }, Add },

{ { "Изменить", 0 }, Edit },

{ { "Поиск", 0 }, Search },

{ { "Сортировка", 0 }, Sort },

{ { "Сохранить", 0 }, Save },

{ { "Помощь", 0 }, Help },

{ { "Выйти", 0 }, Exit },

};

// массив с описаниями элементов меню подтверждения выхода

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

{ { "Да", 0 }, ExitYes },

{ { "Нет", 0 }, ExitNo },

{ { "Отмена", 0 }, ExitCancel },

};

// массив заголовков основной таблицы

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

// массив заголовков (1 заголовок) меню подтверждения выхода

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

// код завершения для функций обратного вызова

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

// признак перезапуска основной таблицы

int redraw\_main = 1;

// признак изменившихся данных

int data\_modified = 0;

// признак необходимости сохранения данных

int save\_data = 0;

// признак отмены выхода

int exit\_canceled = 0;

// индекс для инициализации обновлённого меню

int initial\_table\_index = 0;

// буфер для хранения настроек консоли

CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo;

// экземпляр верхнего меню

MENU top\_menu;

// указатель на экземпляр основной таблицы

MENU \*ptable = NULL;

// Точка входа программы

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

{

return Run();

}

// Выход - сохранить данные

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

save\_data = 1;

return -1;

}

// Выход - не сохранять данные

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

save\_data = 0;

return -1;

}

// Выход - отмена

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

exit\_canceled = 1;

return -1;

}

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

int DefaultESC(MENU\* menu) {

return -1;

}

// Обработчик клавиши ESC

int ESC(MENU\* menu) {

return Exit(menu, NULL);

}

// Обработчик клавиши F1 (Справка)

int F1(MENU\* menu) {

Help(menu, NULL);

return 0;

}

// Обработчик клавиши F2 (Сохранить)

int F2(MENU\* menu) {

Save(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Обработчик клавиши F3 (Поиск)

int F3(MENU\* menu) {

Search(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Обработчик клавиши F4 (Редактирование)

int F4(MENU\* menu) {

Edit(menu, NULL);

if(exit\_code)

return exit\_code;

return 0;

}

// Обработчик клавиши F9 (Переход на верхнее меню)

int F9(MENU\* menu) {

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

if(exit\_code)

return exit\_code;

return 0;

}

// Обработчик прокрутки меню на один элемент вперёд или назад

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

switch(direction) {

case MENU\_CURR\_REV:

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

break;

case MENU\_CURR\_FWD:

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

break;

}

}

// Перенос очередного элемента словаря в элемент описания пункта меню

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

int fld;

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

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

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

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

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

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

return 1; // продолжить итерации по остальным элементам

}

// Перенос подходящего слова в элемент описания пункта меню

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

int fld;

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

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

ITEM\_DEF\* menu\_items = search->menu\_items;//(ITEM\_DEF\*)param;

if(strlen(entry->field[0]) <= search->value && strcmp(search->str, entry->field[search->index]) == 0) {

CharToOemA(entry->field[0], menu\_items[search->item\_count].str[0]);

search->item\_count++;

}

return 1;

}

// Сравнение двух элементов словаря

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

DICT\_ENTRY \*a, \*b;

int index;

if(!arg)

return 0;

a = (DICT\_ENTRY\*)data1;

b = (DICT\_ENTRY\*)data2;

index = \*(int\*)arg;

if(index < 3)

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

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

}

// Основная рабочая функция программы

int Run() {

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

int i, fld;

HANDLE hstdout;

SMALL\_RECT rect;

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

int main\_menu\_items\_count = 0;

WORD background = BACKGROUND\_INTENSITY | BACKGROUND\_BLUE;

// Инициализация заголовков глобальных меню

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

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

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

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

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

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

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

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

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

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

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

CharToOemA(title, title);

SetConsoleTitle(title);

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

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

return -1;

}

// Получаем размеры консоли

GetConsoleScreenBufferInfo(hstdout, &csbInfo);

rect.Left = csbInfo.srWindow.Left;

rect.Right = csbInfo.srWindow.Right;

rect.Top = csbInfo.srWindow.Top;

rect.Bottom = rect.Top;

SetConsoleTextAttribute(hstdout, background);

system("cls");

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

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

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

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

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

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

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

// Загрузка словаря из файла

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

return -1;

}

rect.Top++;

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

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

while (redraw\_main) {

redraw\_main = 0;

//

if(main\_menu\_items) {

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

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

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

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

free(main\_menu\_items);

main\_menu\_items = NULL;

}

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

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

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

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

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

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

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

}

//

if(ptable) {

menu\_clear(ptable);

free(ptable);

}

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

// перекидываем словарь в таблицу

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

// создаём элементы меню

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

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

// устанавливаем текущий индекс

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

list1\_set\_current\_index(&dict, initial\_table\_index);

initial\_table\_index = 0;

// продолжаем настройку меню

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

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

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

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

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

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

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

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

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

menu\_active\_color(ptable,

BACKGROUND\_BLUE | BACKGROUND\_GREEN

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

);

menu\_inactive\_color(ptable,

background | FOREGROUND\_INTENSITY | FOREGROUND\_RED |

FOREGROUND\_GREEN | FOREGROUND\_BLUE);

// блокирующий вызов: основной цикл обработки сообщений главной таблицы

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

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

exit\_code = 0;

}

menu\_clear(ptable);

free(ptable);

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

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

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

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

free(main\_menu\_items);

menu\_clear(&top\_menu);

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

if(main\_headers[i])

free(main\_headers[i]);

return 0;

}

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

// Функции меню

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

// Функция меню <Выход>

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

int result;

if(data\_modified) {

int i;

MENU exit\_menu;

SMALL\_RECT rect;

const int width = 24;

const int height = 7;

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

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

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

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

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

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

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

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

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

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

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

menu\_clear(&exit\_menu);

if(save\_data) {

Save(menu, item);

}

}

if(exit\_canceled) {

exit\_canceled = 0;

result = 0;

} else {

exit\_code = -1;

result = -1;

}

return result;

}

// Функция меню <Добавить>

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

int i;

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

int max\_width;

int menu\_width = 40;

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

int row\_count = DICT\_FLD\_CNT;

char\*\*\* contents;

DICT\_ENTRY\* entry;

InputBox box;

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

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

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

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

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

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

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

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

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

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

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

// надпись

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

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

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

// буфер ввода

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

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

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

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

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

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

}

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

return -1;

}

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

box\_clear(&box);

return -1;

}

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

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

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

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

}

list1\_push\_back(&dict,

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

);

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

data\_modified = 1;

redraw\_main = 1;

exit\_code = -1;

}

box\_clear(&box);

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

if(contents[i]) {

if(contents[i][TITLE])

free(contents[i][TITLE]);

if(contents[i][BUFFER]);

free(contents[i][BUFFER]);

free(contents[i]);

}

}

free(contents);

contents = NULL;

return exit\_code;

}

// Функция меню <Изменить>

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

int i, box\_result;

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

int max\_width;

int menu\_width = 40;

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

int row\_count = DICT\_FLD\_CNT;

char\*\*\* contents;

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

InputBox box;

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

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

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

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

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

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

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

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

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

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

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

// надпись

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

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

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

// буфер ввода

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

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

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

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

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

}

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

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

return -1;

}

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

box\_clear(&box);

return -1;

}

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

if(1 == box\_result) {

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

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

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

}

data\_modified = 1;

redraw\_main = 1;

exit\_code = -1;

}

box\_clear(&box);

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

if(contents[i]) {

if(contents[i][TITLE])

free(contents[i][TITLE]);

if(contents[i][BUFFER]);

free(contents[i][BUFFER]);

free(contents[i]);

}

}

free(contents);

contents = NULL;

return exit\_code;

}

// флаг удаления записи

int del\_record = 0;

// Выход - сохранить данные

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

del\_record = 1;

return -1;

}

// Выход - не сохранять данные

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

del\_record = 0;

return -1;

}

// Функция меню <Удалить>

int Delete(MENU\* pm) {

// количество ячеек (столбцов) меню подтверждения удаления

#define del\_column\_count 1

// количество пунктов меню подтверждения удаления

#define del\_item\_count 2

// массив с описаниями элементов меню подтверждения удаления

const char\* item\_captions[del\_item\_count] = { "Да", "Нет" };

ITEM\_DEF menu\_items[del\_item\_count] = {

{ { NULL, 0 }, DelYes },

{ { NULL, 0 }, DelNo },

};

// массив заголовков (1 заголовок) меню подтверждения выхода

char\* headers[del\_column\_count];

int i;

MENU menu;

SMALL\_RECT rect;

const int width = 19;

const int 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;

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

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

CharToOemA("Удалить запись?", headers[0]);

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

menu\_items[i].str[0] = (char\*)malloc(MAX\_MENU\_HDR \* sizeof(char));

CharToOemA(item\_captions[i], menu\_items[i].str[0]);

}

menu\_init(&menu, ptable, pm->hStdOut, menu\_items, del\_item\_count,

del\_column\_count, MENU\_ORIENT\_VERT, &rect, 1, headers);

menu.user\_tag = MENU\_TAG\_EXIT;

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

menu\_clear(&menu);

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

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

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

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

if(headers[i])

free(headers[i]);

if(del\_record) {

del\_record = 0;

list1\_erase\_current(&dict);

menu\_del\_curr(pm);

menu\_fill\_wnd(pm, 0);

menu\_cls(pm);

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

data\_modified = 1;

}

return 0;

}

// Функция меню <Поиск>

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

int i, retc;

int index;

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

int menu\_width = 48;

int menu\_height = 4;

int max\_width;

char\* titles[] = { "Часть речи", "Лимит букв в Слове" };

char\*\*\* contents;

int row\_count = 2;

SearchCriteria search;

InputBox box;

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

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

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

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

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

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

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

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

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

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

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

// надпись

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

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

// буфер ввода

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

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

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

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

}

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

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

return -1;

}

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

box\_clear(&box);

return -1;

}

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

box\_clear(&box);

return -1;

}

memset(&search, 0x00, sizeof(search));

search.menu\_items = (ITEM\_DEF\*)malloc(dict.count \* sizeof(ITEM\_DEF));

if(search.menu\_items) {

SMALL\_RECT rect;

int menu\_width = 25;

int menu\_height = 0;

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

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

memset(search.menu\_items, 0x00, dict.count \* sizeof(ITEM\_DEF));

for(i = 0; i < dict.count; i++) {

search.menu\_items[i].str[0] = (char\*)malloc(256 \* sizeof(char));

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

}

// ищем слова для отображения

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

OemToCharA(search.str, search.str);

search.index = 1;

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

list1\_for\_each(&dict, check\_entry\_display, &search);

menu\_height = search.item\_count + 4;

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

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

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

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

ShowMenu(pm->hStdOut, search.menu\_items, search.item\_count, "Результаты поиска",

rect, MENU\_FLAG\_WND | MENU\_FLAG\_ITEMS | MENU\_HOTKEYS, MENU\_TAG\_FIND, F1);

memset(&search, 0x00, sizeof(search));

}

box\_clear(&box);

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

if(contents[i]) {

if(contents[i][TITLE])

free(contents[i][TITLE]);

if(contents[i][BUFFER])

free(contents[i][BUFFER]);

free(contents[i]);

}

}

free(contents);

contents = NULL;

return -1;

}

// Функция подменю <Сортировка>

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

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

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

redraw\_main = 1;

exit\_code = -1;

return exit\_code;

}

// Функция меню <Сортировка>

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

int i;

int max\_len;

int item\_count = main\_column\_count;

int slen;

int menu\_width = 20;

int menu\_height = item\_count + 2;

SMALL\_RECT rect;

ITEM\_DEF\* menu\_items;

rect.Left = item->x;

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

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

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

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

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

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

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

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

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

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

menu\_items[i].cb = SortItem;

}

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

return -1;

}

// Функция меню <Сохранить>

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

save\_data = 0;

data\_modified = 0;

dict\_save(dict\_file\_name);

return -1;

}

// Функция меню <Помощь>.

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

int result = 0;

SMALL\_RECT rect;

int width = 0, height = 0;

switch(pm->user\_tag) {

case MENU\_TAG\_TOP:

case MENU\_TAG\_MAIN:

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

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

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

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

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

break;

case MENU\_TAG\_SORT:

width = 95;

height = 8;

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

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

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

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

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

break;

case MENU\_TAG\_FIND:

width = 80;

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;

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

break;

}

return result;

}

// Вспомогательные функции

// Загрузка пунктов меню из файла

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

FILE\* file;

int i;

int item\_count = 0;

int slen;

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

if(!menu\_items)

return NULL;

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

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

if(!file) {

free(menu\_items);

return NULL;

}

i = 0;

while(!feof(file)) {

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

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

break;

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

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

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

// удаляем предпоследний символ - '\n'

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

} else {

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

}

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

break; // достигнут лимит строк

}

}

fclose(file);

item\_count = i;

if(pcount)

\*pcount = item\_count;

return menu\_items;

}

// Отображение упрощённого вертикального меню в один столбец

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

int i;

MENU menu;

// количество ячеек меню (столбцов)

const int column\_count = 1;

char\*\* headers = NULL;

int border = 1;

if(title) {

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

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

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

if(!headers[0]) {

free(menu\_items);

return -1;

}

strcpy(headers[0], title);

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

}

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

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

menu.user\_tag = user\_tag;

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

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

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

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

menu\_draw(&menu, flags);

menu\_clear(&menu);

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

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

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

free(menu\_items);

if(headers) {

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

if(headers[i])

free(headers[i]);

free(headers);

}

return 0;

}

// Отображение текста справки из файла на всплывающем окне

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

int i;

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

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

int item\_count = 0;

// 4 = заголовок + 3 гориз. линиии

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

if(!menu\_items) {

return -1;

}

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

}

**menu.c**

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

#include <windows.h>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#pragma hdrstop

#define \_\_DICT\_H\_\_

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

#define \_\_MENU\_H\_\_

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

#include "common.h"

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

#pragma package(smart\_init)

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

#define MENU\_WHITESPACE ' '

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

// Отрисовка пункта меню (callback для элемента списка)

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

// Проверить код, возвращённый обработчиком

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

// Очистка одного элемента меню

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

// Обработка горячей клавиши (если была нажата)

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

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

menu->activeItemAttributes = attr;

}

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

menu->inactiveItemAttributes = attr;

menu->workWindowAttributes = attr;

}

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

int i, border\_bottom\_index;

if(items\_count == 0)

items\_count = menu->items.count;

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

// проход по строкам окна

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

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

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

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

if(menu->has\_border) {

// отрисовка вертикальной границы слева и справа

if(i < border\_bottom\_index) {

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

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

}

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

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

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

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

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

}

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

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

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

if(0 == i) { // первая строка

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

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

}

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

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

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

}

}

}

}

if(menu->has\_border) {

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

if(menu->has\_headers) {

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

// отрисовка вертикальных линий в заголовке

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

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

}

}

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

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

if(menu->has\_headers) {

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

} else {

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

}

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

}

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

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

}

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

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

{

int left\_indent = 1;

int i, j, len;

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

return -1;

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

menu->parent = parent;

menu->orient = orient;

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

// Сохраняем дескриптор консольного вывода

menu->hStdOut = hstdout;

// Сохраняем полученные размеры окна

if(prect != NULL ) {

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

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

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

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

} else {

// Получаем размеры консоли

CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo;

GetConsoleScreenBufferInfo(menu->hStdOut, &csbInfo);

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

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

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

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

}

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

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

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

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

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

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

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

}

//

menu->has\_border = border;

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

menu->cell\_count = cell\_count;

menu->left\_pad = 0;

menu->top\_pad = 0;

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

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

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

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

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

}

if(menu->has\_headers) {

menu->top\_pad += 2;

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

}

if(menu->has\_border) {

menu->left\_pad += 1;

menu->top\_pad += 1;

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

}

// не можем отобразить не помещающиеся в окно элементы

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

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

}

}

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

// координаты углов консоли

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

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

menu->workWindowAttributes = //0x9E;

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

menu->inactiveItemAttributes = 0x1F;

menu->activeItemAttributes = 0xA0;

// Инициализация элементов меню

list1\_init(&menu->items);

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

//int next = 0

switch(orient) {

case MENU\_ORIENT\_HORZ:

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

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

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

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

int width;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

} else {

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

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

}

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

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

}

item->index = i;

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

} // for(item)

break;

case MENU\_ORIENT\_VERT:

if(menu->has\_headers) {

if(menu->has\_border) {

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

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

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

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

}

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

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

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

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

if(headers[j]) {

len = strlen(headers[j]);

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

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

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

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

} else {

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

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

}

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

// отрисовка вертикальных линий в заголовке

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

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

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

}

}

} // for(cell)

} // if(has\_headers)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

} else {

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

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

}

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

int top\_shift;

if(menu->has\_headers) {

top\_shift = 2;

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

} else {

top\_shift = 0;

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

}

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

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

}

}

} // for(cell)

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

item->index = i;

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

//next++;

} // for(item)

break;

}

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

return 0;

}

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

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

if(item->str)

free(item->str);

free(item);

}

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

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

}

void menu\_clear(MENU\* menu) {

int i;

list1\_clear(&menu->items);

if(menu->hdr) {

free(menu->hdr);

menu->hdr = NULL;

}

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

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

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

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

}

free(menu->wnd.m);

menu->wnd.m = NULL;

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

menu->hStdOut = INVALID\_HANDLE\_VALUE;

}

void menu\_prev(MENU\* menu) {

int wrap = 1;

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

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

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

showCursor(menu, 0);

if(menu->changed\_cb) {

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

}

}

void menu\_next(MENU\* menu) {

int wrap = 1;

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

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

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

showCursor(menu, 0);

if(menu->changed\_cb) {

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

}

}

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

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

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

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

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

switch(menu->orient) {

case MENU\_ORIENT\_HORZ:

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

break;

case MENU\_ORIENT\_VERT:

item->y += index;

break;

}

return 1; // продолжить итерации по остальным элементам

}

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

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

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

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

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

printf(item->str);

return 1; // продолжить итерации по остальным элементам

}

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

int result = 0, i, run = 1;

if(flags & MENU\_FLAG\_WND) {

// Устанавливаем цветовые параметры текста

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

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

menu\_cls(menu);

}

if(flags & MENU\_FLAG\_ITEMS) {

// Номер текущего пункта меню

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

// рисуем заголовок

if(menu->hdr) {

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

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

printf(menu->hdr);

}

// рисуем меню

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

fflush(stdout);

}

if(flags & MENU\_DRAW\_SEL) {

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

}

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

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

return 0; // не запускаем обработку сообщений

while (run) {

if (kbhit()) {

menu->cb\_retcode = 0;

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

if(flags & MENU\_NAVIGATOR) {

switch (menu->last\_key) {

case KEY\_ARROW\_UP:

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

menu\_prev(menu);

case KEY\_ARROW\_LEFT:

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

menu\_prev(menu);

break;

case KEY\_ARROW\_RIGHT:

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

menu\_next(menu);

break;

case KEY\_ARROW\_DOWN:

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

menu\_next(menu);

break;

case KEY\_ENTER:

// Возвращаем курсор из строки меню в прежнюю позицию

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

// Установить цвет рабочих сообщений

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

///showCursor(menu, 1);

// Вызываем обработчик пункта меню

{

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

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

}

run = check\_cb\_retcode(menu);

break;

} // switch(iKey)

}

if(flags & MENU\_HOTKEYS) {

// обработка горячих клавиш

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

run = check\_cb\_retcode(menu);

}

} // if(kbhit())

} // while(run)

if(menu->parent) {

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

}

return result;

}

void itemMenu(MENU\* menu, int activate)

{

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

WORD itemAttributes;

if(!item)

return;

if (activate)

itemAttributes = menu->activeItemAttributes;

else

itemAttributes = menu->inactiveItemAttributes;

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

SetConsoleTextAttribute(menu->hStdOut, itemAttributes);

printf(item->str);

}

void menu\_cls(MENU\* menu)

{

int i, y;

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

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

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

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

}

}

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

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

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

hk->code = code;

hk->cb = cb;

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

return menu->hk\_list.count;

}

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

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

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

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

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

return 0; // не проверять остальные обработчики

}

return 1; // продолжить итерации по остальным элементам

}

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

{

COORD cursorPos;

cursorPos.X = x;

cursorPos.Y = y;

SetConsoleCursorPosition(menu->hStdOut, cursorPos);

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

}

void showCursor(MENU\* menu, int visible)

{

CONSOLE\_CURSOR\_INFO ccInfo;

ccInfo.bVisible = visible;

ccInfo.dwSize = 20;

SetConsoleCursorInfo(menu->hStdOut, &ccInfo);

}

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

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

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

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

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

return 0;

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

;

}

// очистить буфер клавиатуры

fflush(stdin);

// курсор в текущий пункт меню

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

// спрятать курсор

showCursor(menu, 0);

return 1;

}

**list.c**

#include <stdlib.h>

#include <string.h>

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

#include "common.h"

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

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

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

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

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

LIST1\* list1\_new() {

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

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

return NULL;

return list;

}

int list1\_delete(LIST1\* list) {

if(!list1\_clear(list))

return -1;

free(list);

return 0;

}

int list1\_init(LIST1\* list) {

if(!list)

return -1;

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

list->item\_free = free;

return 0;

}

int list1\_clear(LIST1\* list) {

LIST1\_ITEM\* curr;

if(!list)

return -1;

curr = list->head;

while(curr) {

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

delete\_item(list, curr);

curr = next;

}

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

return 0;

}

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

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

if(!item)

return -1;

if(list->head) {

item->next = list->head;

} else { // добавляется первый элемент

list->curr = item;

list->tail = item;

}

list->head = item;

list->count++;

return 0;

}

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

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

if(!item)

return -1;

if(list->head) {

list->tail->next = item;

} else { // добавляется первый элемент

list->head = item;

list->curr = item;

}

list->tail = item;

list->count++;

return 0;

}

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

LIST1\_ITEM\* curr;

int i = 0;

if(!cb)

return -1;

curr = list->head;

while(curr) {

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

curr = curr->next;

}

return 0;

}

// Прокрутить текущий указатель вперёд

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

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

return NULL;

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

if(!list->curr && wrap)

list->curr = list->head;

return list->curr->data;

}

// Прокрутить текущий указатель назад

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

LIST1\_ITEM \*curr, \*prev;

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

return NULL;

curr = list->head;

prev = list->head;

while(curr) {

if(curr == list->curr) {

if(curr == list->head) {

if(wrap) {

list->curr = list->tail;

}

} else {

list->curr = prev;

}

break;

}

prev = curr;

curr = curr->next;

}

return list->curr->data;

}

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

if(!list->head)

return NULL;

return list->head->data;

}

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

if(!list->tail)

return NULL;

return list->tail->data;

}

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

if(!list->curr)

return NULL;

return list->curr->data;

}

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

LIST1\_ITEM \*curr, \*prev;

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

return NULL;

curr = list->head;

prev = list->head;

while(curr) {

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

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

// сместить текущий указатель при необходимости

if(curr == list->curr) {

list->curr = prev;

}

// удалить найденный элемент вместе с данными

delete\_item(list, curr);

} else {

prev = curr;

}

curr = next;

}

if(!list->curr)

return NULL;

return list->curr->data;

}

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

LIST1\_ITEM \*curr, \*prev;

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

return NULL;

curr = list->head;

prev = list->head;

while(curr) {

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

if(curr == list->curr) {

// определить новый текущий элемент

prev->next = next;

list->curr = next;

if(curr == list->head) {

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

}

if(curr == list->tail) {

if(list->head == NULL)

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

else

list->tail = prev;

}

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

list->curr = prev;

}

// удалить старый текущий элемент вместе с данными

delete\_item(list, curr);

break;

}

prev = curr;

curr = next;

}

if(!list->curr)

return NULL;

return list->curr->data;

}

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

int i = 0;

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

while(curr) {

if(curr == list->curr)

return i;

i++;

curr = curr->next;

}

return 0;

}

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

int i = 0;

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

while(curr) {

if(i == index) {

list->curr = curr;

break;

}

i++;

curr = curr->next;

}

return 0;

}

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

int i = 0;

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

while(curr) {

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

list->curr = curr;

break;

}

i++;

curr = curr->next;

}

return i;

}

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

LIST1\_ITEM\* curr;

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

curr = list->head;

while(curr) {

list->tail = curr;

curr = curr->next;

}

}

// вспомогательные функции

// создать элемент списка - контейнер данных

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

LIST1\_ITEM\* item;

if(NULL == data)

return NULL;

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

if(!item)

return NULL;

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

item->data = data;

return item;

}

// удалить элемент списка - контейнер данных

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

if(list->item\_free)

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

free(item);

list->count--;

}

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

LIST1\_ITEM \*low = NULL;

LIST1\_ITEM \*high = NULL;

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

return;

}

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

mergeSort(&low, compare, param);

mergeSort(&high, compare, param);

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

}

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

LIST1\_ITEM\* fast = NULL;

LIST1\_ITEM\* slow = NULL;

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

(\*low) = src;

(\*high) = NULL;

return;

}

slow = src;

fast = src->next;

while (fast != NULL) {

fast = fast->next;

if (fast != NULL) {

fast = fast->next;

slow = slow->next;

}

}

(\*low) = src;

(\*high) = slow->next;

slow->next = NULL;

}

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

LIST1\_ITEM tmp;

\*c = NULL;

if (a == NULL) {

\*c = b;

return;

}

if (b == NULL) {

\*c = a;

return;

}

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

\*c = a;

a = a->next;

} else {

\*c = b;

b = b->next;

}

tmp.next = \*c;

while (a && b) {

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

(\*c)->next = a;

a = a->next;

} else {

(\*c)->next = b;

b = b->next;

}

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

}

if (a) {

while (a) {

(\*c)->next = a;

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

a = a->next;

}

}

if (b) {

while (b) {

(\*c)->next = b;

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

b = b->next;

}

}

\*c = tmp.next;

}

**dict.c**

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#pragma hdrstop

#include <sys/types.h>

#include <sys/stat.h>

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

#define \_\_DICT\_H\_\_

#include "common.h"

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

#pragma package(smart\_init)

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

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

// глобальный экзепляр словаря

LIST1 dict;

// Вспомогательная функция загрузки из файла

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

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

// Функции словаря

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

//

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

off\_t i, fld, size = 0;

char\* buf = file\_load(file\_name, &size);

if(buf) {

int field\_start = 0, nfield = 0;

DICT\_ENTRY temp\_entry;

memset(&temp\_entry, 0x00, sizeof(DICT\_ENTRY));

for(i = 0; i < size; i++) {

char ch = buf[i];

if('\0' == ch) {

temp\_entry.field[nfield] = (char\*)malloc((i - field\_start + 1) \* sizeof(char));

strcpy(temp\_entry.field[nfield], &buf[field\_start]);

field\_start = i + 1;

nfield++;

if(DICT\_FLD\_CNT == nfield) {

list1\_push\_back(&dict,

dict\_entry\_new(temp\_entry.field[0], temp\_entry.field[1], temp\_entry.field[2])

);

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

if(temp\_entry.field[fld])

free(temp\_entry.field[fld]);

nfield = 0;

}

}

} // for(i)

free(buf);

} // if(buf)

return 0;

}

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

const char\* empty = "";

int i;

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

FILE\* file = (FILE\*)param;

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

if(entry->field[i]) {

fwrite(entry->field[i], strlen(entry->field[i]) + 1, 1, file);

} else {

fwrite(empty, 2, 1, file);

}

}

return 1; // продолжить итерации по остальным элементам

}

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

int nwrote;

FILE\* file = fopen(file\_name, "wb");

if(!file)

return -1;

#if 1

list1\_for\_each(&dict, dict\_entry\_save, file);

#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;

}

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.c**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <windows.h>

#pragma hdrstop

#include <conio.h>

#define \_\_DICTIONARY\_INPUT\_DLG\_H\_\_

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

#include "common.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->row\_count = row\_count;

#if 1 // используем внешнюю память

box->contents = contents;

#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;

}

}

int box\_save(InputBox\* box) {

COORD coord = { 0, 0 };

int ok = ReadConsoleOutput(

box->handle, // экранный буфер, из которого читаем

box->bak, // буфер, в который копируем

box->size, // размеры bak: колонки/строки

coord, // верхняя левая ячейка назначения в bak

&box->rect); // источник - прямоугольник экранного буфера

return ok ? 0 : -1;

}

int box\_restore(InputBox\* box) {

COORD coord = { 0, 0 };

int ok = WriteConsoleOutput(

box->handle, // экранный буфер, в который пишем

box->bak, // буфер, из которого копируем

box->size, // размеры bak: колонки/строки

coord, // исходная верхняя левая ячейка в bak

&box->rect); // приёмник - прямоугольник экранного буфера

if (!ok) {

return -1;

}

return ok ? 0 : -1;

}

int draw\_background(InputBox\* box) {

COORD coord = { 0, 0 };

int i, j, index = 0, ok, len;

WORD background = BACKGROUND\_BLUE | BACKGROUND\_GREEN;

for(i = 0; i < box->size.Y; ++i) {

for(j = 0; j < box->size.X; ++j) {

index = (i \* box->size.X) + j;

box->wnd[index].Attributes = background;

box->wnd[index].Char.AsciiChar = ' ';

}

}

for(i = 1; i <= box->row\_count; ++i) {

for(j = 1; j < box->max\_width - 1; ++j) {

index = (i \* box->size.X) + j;

box->wnd[index].Attributes = background; // box->edit\_attr;

len = strlen(box->contents[i - 1][TITLE]);

if(j <= len) {

box->wnd[index].Char.AsciiChar = box->contents[i - 1][TITLE][j-1];

} else {

box->wnd[index].Char.AsciiChar = ' ';

}

}

for(j = box->max\_width; j < box->size.X - 1; ++j) {

index = (i \* box->size.X) + j;

box->wnd[index].Attributes = box->edit\_attr;

box->wnd[index].Char.AsciiChar = ' ';

}

}

ok = WriteConsoleOutput(

box->handle, // экранный буфер, в который пишем

box->wnd, // буфер, из которого копируем

box->size, // размеры wnd: колонки/строки

coord, // исходная верхняя левая ячейка в wnd

&box->rect); // приёмник - прямоугольник экранного буфера

return ok ? 0 : -1;

}

void box\_gotoxy(InputBox\* box, int x, int y)

{

COORD cursorPos;

cursorPos.X = x;

cursorPos.Y = y;

SetConsoleCursorPosition(box->handle, cursorPos);

}

void showCursor(HANDLE handle, int visible)

{

CONSOLE\_CURSOR\_INFO ccInfo;

ccInfo.bVisible = visible;

ccInfo.dwSize = 20;

SetConsoleCursorInfo(handle, &ccInfo);

}

static int accept\_input(char ch) {

if(isalnum(ch))

return 1;

if(ispunct(ch) || ' ' == ch)

return 1;

if((-96 <= ch && ch <= -81) || (-32 <= ch && ch <= -17) || -15 == ch)

return 1;

return 0;

}

void trim(char \*str) {

int len = strlen(str);

int beg = 0;

while(str[len - 1] == ' ')

str[--len] = 0;

while(str[beg] == ' ')

str[beg++] = 0;

if(beg)

memmove(str, str + beg, len - beg + 1);

}

int box\_draw(InputBox\* box) {

int i, left, top, run = 1, result = 0;

int \*pos, \*len;

if(0 == box->row\_count)

return -1;

box->row = 0;

pos = (int\*)malloc(box->row\_count \* sizeof(int));

memset(pos, 0x00, box->row\_count \* sizeof(int));

len = (int\*)malloc(box->row\_count \* sizeof(int));

memset(len, 0x00, box->row\_count \* sizeof(int));

if(-1 == draw\_background(box))

return -1;

// координаты начала ввода

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

top = box->rect.Top + 1;

// Draw wnd

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

// Input

box\_gotoxy(box, left, top);

showCursor(box->handle, 1);

// буфер ввода

//memset(box->contents[box->row][BUFFER], ' ', (box->max\_width + 1) \* sizeof(char));

box->contents[box->row][BUFFER][box->max\_width] = '\0';

//char tmp[2];

//tmp[1] = '\0';

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

len[i] = strlen(box->contents[i][BUFFER]);

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

printf("%s", box->contents[i][BUFFER]);

}

box\_gotoxy(box, left, top + box->row);

while (run) {

if (kbhit()) {

int ch = \_getch();

//tmp[0] = ch;

//CharToOemA(tmp, tmp);

//OemToCharA(tmp, tmp);

//ch = tmp[0];

switch (ch) {

case KEY\_HOME:

pos[box->row] = 0;

box\_gotoxy(box, left + pos[box->row], top);

break;

case KEY\_ARROW\_LEFT:

if(pos[box->row] > 0) {

pos[box->row]--;

box\_gotoxy(box, left + pos[box->row], top);

}

break;

case KEY\_ARROW\_RIGHT:

if(pos[box->row] < len[box->row]) {

pos[box->row]++;

box\_gotoxy(box, left + pos[box->row], top);

}

break;

case KEY\_END:

pos[box->row] = len[box->row];

box\_gotoxy(box, left + pos[box->row], top);

break;

//case KEY\_ARROW\_UP: case KEY\_ARROW\_DOWN:

// break;

case KEY\_ENTER:

for(i = 0; i < box->row\_count; ++i)

trim(box->contents[i][BUFFER]);

//printf("\n%s", box->contents[box->row][BUFFER]);

result = 1;

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:

result = 0;

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)

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 result;

}