**Курсовая работа**

**по дисциплине: «»**

**«»**

2019 г

# Задание

Вариант 2

Написать стековый калькулятор, который принимает в качестве аргумента командной строки имя файла, содержащего команды. Если аргумента нет, то использовать стандартный поток ввода для чтения команд. Использовать вещественные числа.

Реализовать следующий набор команд:

# – строка с комментарием.

POP – снять число со стека.

PUSH – положить число на стек.

+, -, \*, /, SQRT – арифметические операции. Используют один или два верхних элемента стека, измают их из стека, помещая результат назад.

PRINT – печать верхнего элемента стека (без удаления).

DEFINE – задать значение параметра. В дальнейшем везде использовать вместо параметра это значение.

Пример (должно вывести 2):

DEFINE a 4

PUSH a

SQRT

PRINT

Оглавление

[Задание 2](#_Toc517072948)

[Введение 4](#_Toc517072949)

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

[2 Глобальные данные 6](#_Toc517072951)

[2.1 Основные модули программы 6](#_Toc517072952)

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

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

[3 Описание пользовательских макроопределений 8](#_Toc517072955)

[3.1 Макроопределения модуля main 8](#_Toc517072956)

[3.2 Макроопределения модуля dict 8](#_Toc517072957)

[3.3 Макроопределения модуля menu 8](#_Toc517072958)

[3.4 Макроопределения модуля input 9](#_Toc517072959)

[3.5 Глобальные макроопределения 10](#_Toc517072960)

[4 Описание пользовательских функции 11](#_Toc517072961)

[4.1 Функции модуля main 11](#_Toc517072962)

[4.2 Функции модуля dict 13](#_Toc517072963)

[4.3 Функции модуля menu 13](#_Toc517072964)

[4.4 Функции модуля input 14](#_Toc517072965)

[4.5 Функции модуля list 15](#_Toc517072966)

[5 Нестандартные функции 17](#_Toc517072967)

[6 Внешние файлы 18](#_Toc517072968)

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

[Заключение 22](#_Toc517072970)

[Список литературы 23](#_Toc517072971)

[Приложения 24](#_Toc517072972)

# Введение

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

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

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

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

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

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

# Исходные данные и способ их ввода

**Ввод данных из файла.**

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

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

**Ввода данных с клавиатуры.**

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

**Редактирование справочной информации.**

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

# Глобальные данные

## Основные модули программы

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

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

## Описание глобальных данных модуля main.c

Массив с описаниями элементов верхнего меню:

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

Массив с описаниями элементов меню подтверждения выхода:

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

Массив заголовков основной таблицы:

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

Массив заголовков меню подтверждения выхода

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

код завершения для функций обратного вызова

**int exit\_code = 0;**

признак перезапуска основной таблицы

**int redraw\_main = 1;**

признак изменившихся данных

**int data\_modified = 0;**

признак необходимости сохранения данных

**int save\_data = 0;**

признак отмены выхода

**int exit\_canceled = 0;**

индекс для инициализации обновлённого меню

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

буфер для хранения настроек консоли:

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

экземпляр верхнего меню

**MENU top\_menu;**

указатель на экземпляр основной таблицы

**MENU \*ptable = NULL;**

## Описание глобальных данных модуля dict.c

Глобальная строковая константа – имя файла данных:

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

глобальный экземпляр словаря (односвязный список):

**LIST1 dict;**

# Описание пользовательских макроопределений

## Макроопределения модуля main

Максимально допустимая длина строки:

**#define MAX\_STRING 256**

Пользовательские метки, чтобы различать меню в функциях обратного вызова:

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

Количество пунктов верхнего меню

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

Количество пунктов верхнего меню

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

Количество ячеек (столбцов) основного табличного меню

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

Количество пунктов меню подтверждения выхода

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

## Макроопределения модуля dict

Максимальная длина хранимой в словаре строки:

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

Количество полей (Слово, Часть речи, Перевод):

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

## Макроопределения модуля menu

Признаки ориентации меню:

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

Максимальная длина строки заголовка

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

Флаги отрисовки меню и обработки сообщений:

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

Максимальное количество ячеек (столбцов вертикального меню)

**#define MAX\_CELLS 50**

Флаги прокрутки текущего пункта

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

Символ фоновой зарисовки

**#define MENU\_WHITESPACE ' '**

Наибольшее из двух значений:

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

## Макроопределения модуля input

Максимальная длина подписи поля

**#define MAX\_TITLE 50**

Предопределённые индексы для обращения к содержимому:

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

Общее количество колонок (заголовок и данные)

**#define COLUMNS 2**

## Глобальные макроопределения

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

Коды клавиатуры:

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

Символы псевдографики:

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

# Описание пользовательских функции

## Функции модуля main

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

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

**Функции обратного вызова.**

Обработчики пунктов верхнего меню:

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

Обработчики пунктов меню подтверждения выхода:

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

Обработчики горячих клавиш:

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

Функции обратного вызова для односвязного списка:

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

**Вспомогательные функции.**

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

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

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

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

## Функции модуля dict

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

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

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

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

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

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

## Функции модуля menu

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

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

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

char\* headers[]) –

инициализация экземпляра меню.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Функции модуля input

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

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

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

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

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

## Функции модуля list

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Нестандартные функции

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

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

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

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

# Внешние файлы

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

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

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

Ожидаемая кодировка тестовых файлов справки – CP866.

# Описание порядка работы с программой

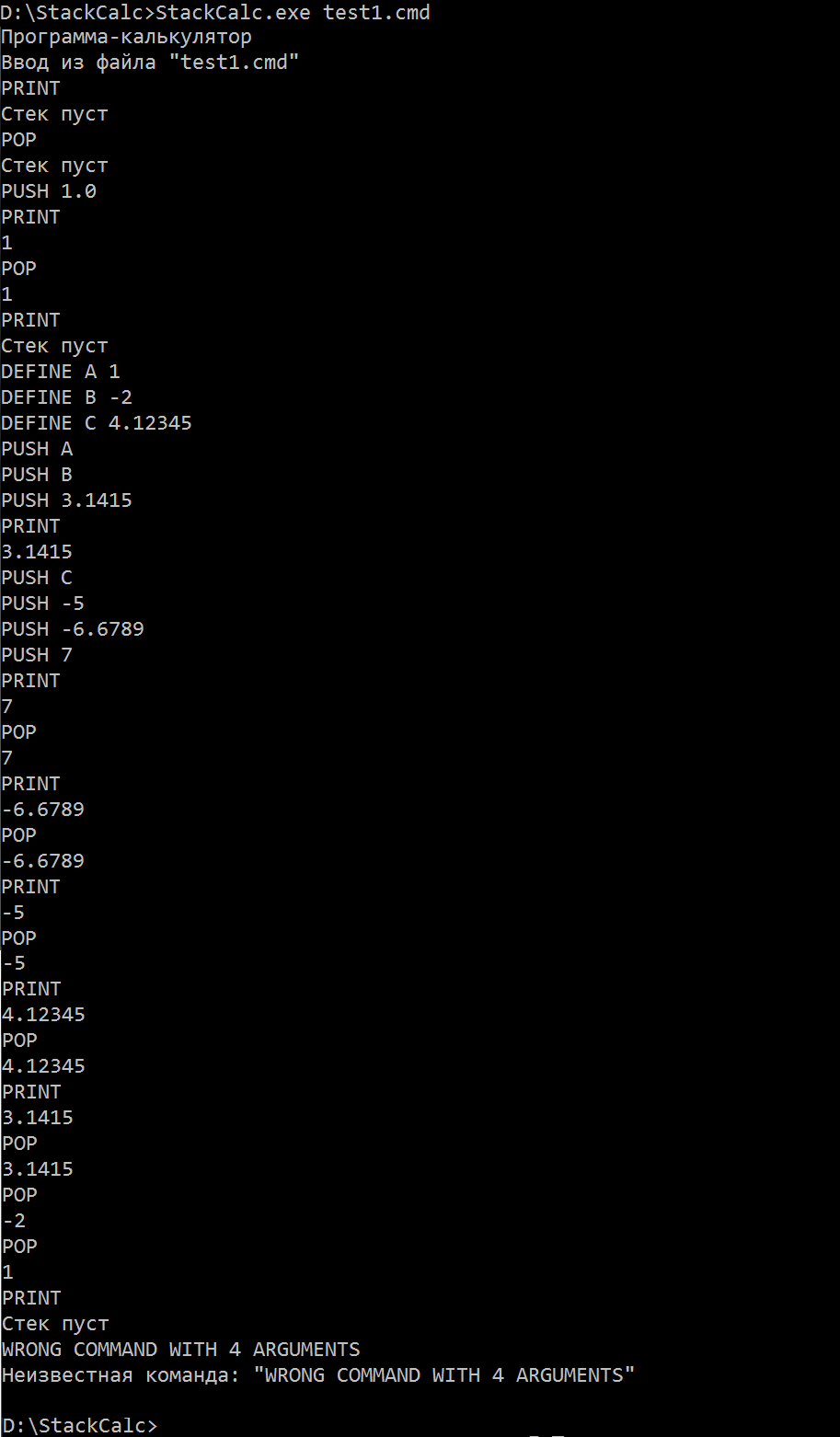


Рисунок 1.

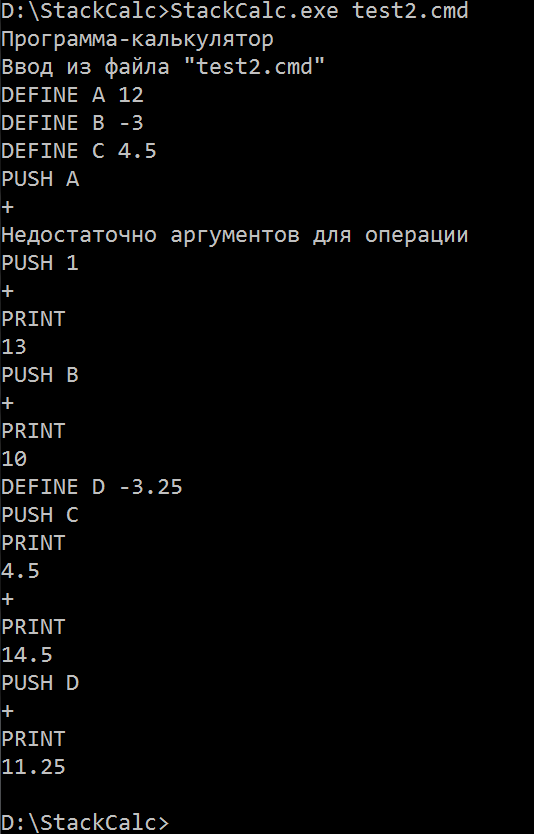


Рисунок 2.

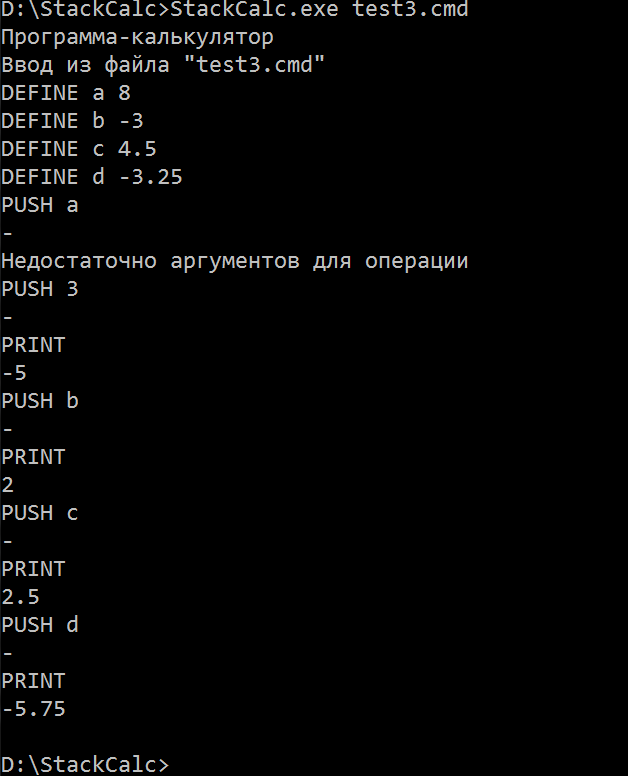


Рисунок 3.

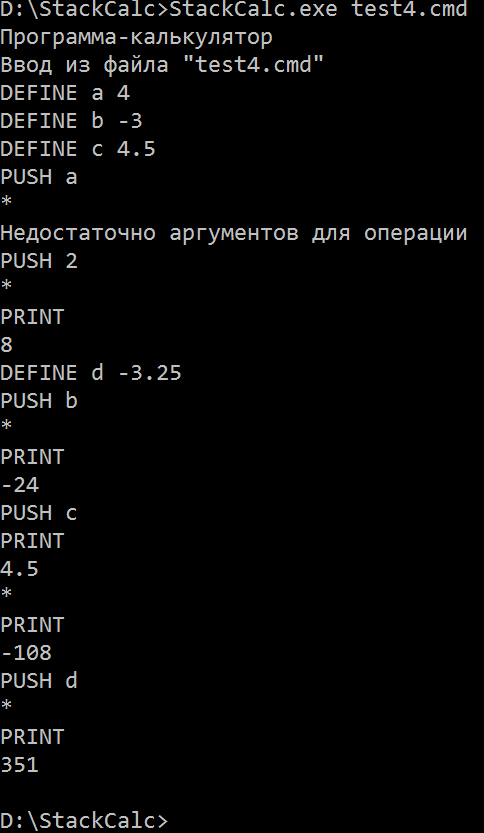


Рисунок 4.

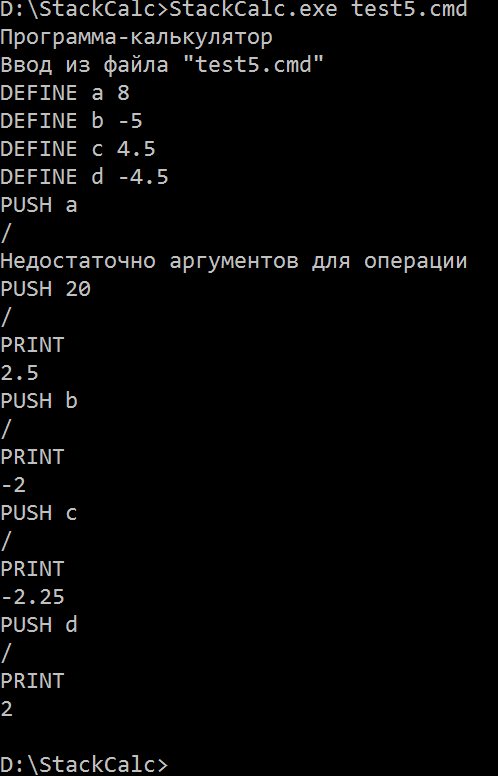


Рисунок 5.

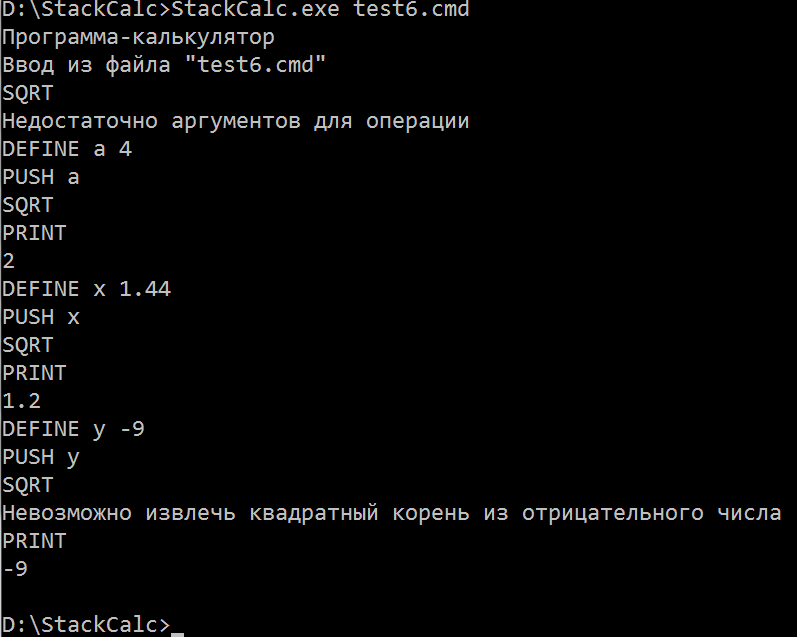


Рисунок 6.

# 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.

# ПРИЛОЖЕНИЕ А. Исходные текст программы

**Program.cs**

using System;

using System.IO;

namespace StackCalc

{

// Основной класс приложения

class Program

{

// Ссылка на класс калькулятора

static Calc calc = new Calc();

// Основная функця (точка входа) приложения

static void Main(string[] args)

{

Console.WriteLine("Программа-калькулятор");

try

{

// Проверяем наличие аргументов командной строки

if (args.Length > 0)

{

// Первый аргумент считаем именем файла,

// считываем команды из него

string fileName = args[0];

Console.WriteLine("Ввод из файла \"" + fileName + "\"");

// Открываем файл на чтение

StreamReader reader = new StreamReader(fileName);

// Читаем строки до конца файла

while (true)

{

string cmd = reader.ReadLine();

if (cmd != null)

{

// Каждую прочитанную строку печатаем

// и отправляем в калькулятор

if (processCommand(cmd, true))

{

break;

}

}

else

{

break;

}

}

// Закрываем файл

reader.Close();

}

else

{

// Программа запущена без параментров,

// считываем команды из консоли

Console.WriteLine("Ввод с клавиатуры");

while (true)

{

// Каждую прочитанную строку передаём

// калькулятору без вывода на экран

if (processCommand(Console.ReadLine(), false))

{

break;

}

}

}

}

catch (System.Exception exception)

{

Console.WriteLine("Ошибка: "+ exception.Message);

}

}

// Функция обработки отдельной команды

// Возвражаемое значение:

// false - продолжаем ввод команд

// true - завершаем работу

static bool processCommand(string cmd, bool trace)

{

// Комментарии и пустые строки пропускаем

if (cmd.Length == 0 || cmd[0] == '#')

{

return false;

}

// Выводим исходную команду на консоль

// (при вводе с клавиатуры не имеет смысла,

// т.к. пользователь уже её сам набрал в консоли)

if (trace)

{

Console.WriteLine(cmd);

}

// Переводим все символы в верхний регистр

// (убираем зависимость от регистра)

cmd = cmd.ToUpper();

// Разбиваем команду на имя команды

// и аргументы (если есть)

string[] tokens = cmd.Split(new char[] { ' ' });

// Первый элемент обязан содержать имя команды

if (tokens.Length == 0)

{

return false;

}

if (tokens[0] == "DEFINE")

{

// Задать значение параметра

// (ожидаем: DEFINE имя значение)

if (tokens.Length == 3)

{

try

{

calc.Define(tokens[1], tokens[2]);

}

catch (System.FormatException exeption)

{

Console.WriteLine("Formatxception: " +

exeption.Message);

}

}

}

else if (tokens[0] == "PUSH")

{

// Положить число на стек

// (ожидаем: PUSH значение)

if (tokens.Length == 2)

{

calc.Push(tokens[1]);

}

}

else if (cmd == "POP")

{

// Снять число со стека

calc.Pop();

}

else if (cmd == "PRINT")

{

// Печатаем верхушку стека

calc.Print();

}

else if (cmd == "+")

{

// Складываем два верхних операнда

calc.Add();

}

else if (cmd == "-")

{

// Вычитаем два верхних операнда

calc.Sub();

}

else if (cmd == "\*")

{

// Перемножаем два верхних операнда

calc.Mul();

}

else if (cmd == "/")

{

// Делим два верхних операнда

calc.Div();

}

else if (cmd == "SQRT")

{

// Извлекаем корень из операнда на верхушине стека

calc.Sqrt();

}

else if (cmd == "END")

{

// Возвращаем признак завершения работы

return true;

}

else

{

// Сообщаем об ошибке, но не прерываем работу

Console.WriteLine("Неизвестная команда: \"" + cmd + "\"");

}

// Возвращаем признак продолжения работы

return false;

}

}

}

**Calc.cs**

using System;

using System.Collections.Generic;

using System.Globalization;

namespace StackCalc

{

/\*

\* Класс, отвечающий за основные функции программы -

\* управление стеком, хранение параментров и выполнение операций

\*/

class Calc

{

Stack<string> stack = new Stack<string>();

Dictionary<string, double> defs = new Dictionary<string, double>();

// Задать значение сохраняемого параметра

// параметры функции:

// name - имя сохраняемого параметра

// str - строковое значение сохраняемого параметра

public void Define(string name, string str)

{

// Преобразовать строковое значение

// в число с плавающей точкой

double value = double.Parse(str.Trim(),

CultureInfo.InvariantCulture.NumberFormat);

if (defs.ContainsKey(name))

{

// Заменить существующее значение

defs[name] = value;

}

else

{

// Добавить новое определение

defs.Add(name, value);

}

}

// Поместить операнд на вершину стека

// параметры функции:

// arg - строковое представление операнда

public void Push(string arg)

{

stack.Push(arg);

}

// Снять операнд с вершины стека и напечатать

public void Pop()

{

if (stack.Count > 0)

{

Console.WriteLine(GetValue(stack.Pop()).ToString());

}

else

{

Console.WriteLine("Стек пуст");

}

}

// Напечатать значение операнда на вершине стека

// если на вершине стека находится имя сохраённого

// параметра, то его значение извлекается из хранилища

public void Print()

{

if (stack.Count > 0)

{

Console.WriteLine(GetValue(stack.Peek()).ToString());

}

else

{

Console.WriteLine("Стек пуст");

}

}

// Складывает два верхних операнда,

// результат помещает обратно в стек

public void Add()

{

if (stack.Count > 1)

{

// Забрать из стека первый операнд

double a = GetValue(stack.Pop());

// Забрать из стека второй операнд

double b = GetValue(stack.Pop());

// Выполнить операцию (сложение)

double c = a + b;

// Поместить результат обратно в стек

stack.Push(c.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Вычитает два верхних операнда,

// результат помещает обратно в стек

public void Sub()

{

if (stack.Count > 1)

{

// Забрать из стека первый операнд

double a = GetValue(stack.Pop());

// Забрать из стека второй операнд

double b = GetValue(stack.Pop());

// Выполнить операцию (вычитание)

double c = a - b;

// Поместить результат обратно в стек

stack.Push(c.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Перемножает два верхних операнда,

// результат помещает обратно в стек

public void Mul()

{

if (stack.Count > 1)

{

// Забрать из стека первый операнд

double a = GetValue(stack.Pop());

// Забрать из стека второй операнд

double b = GetValue(stack.Pop());

// Выполнить операцию (умножение)

double c = a \* b;

// Поместить результат обратно в стек

stack.Push(c.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Делит два верхних операнда,

// результат помещает обратно в стек

public void Div()

{

if (stack.Count > 1)

{

// Забрать из стека первый операнд

double a = GetValue(stack.Pop());

// Забрать из стека второй операнд

double b = GetValue(stack.Pop());

// Выполнить операцию (деление)

double c = a / b;

// Поместить результат обратно в стек

stack.Push(c.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Извлекает корень из операнда на верхушине стека,

// результат помещает обратно в стек

public void Sqrt()

{

if (stack.Count > 0)

{

// Забрать из стека аргумент

double x = GetValue(stack.Pop());

if(x >= 0)

{

// Выполнить операцию (взятие квадратного корня)

x = Math.Sqrt(x);

}

else

{

Console.WriteLine("Невозможно извлечь квадратный корень из отрицательного числа");

}

// Поместить результат обратно в стек

stack.Push(x.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Возвращает действительное число, соответствующее

// переданному строковому аргументу.

// Если это число, то функция преобразует строковое

// представление в число с плавающей точкой, иначе

// извлекает число из хранилища именованных параметров,

// параметры функции:

// arg - строковое представление операнда

private double GetValue(string arg)

{

double value = 0.0;

try

{

// Пробуем преобразовать строковый аргумент в число с плавающей точкой

value = double.Parse(arg.Trim(),

CultureInfo.InvariantCulture.NumberFormat);

}

catch (System.FormatException)

{

// Ошибка - возможно, это имя параметра,

// пробуем найти его в коллекции определений

if (defs.ContainsKey(arg))

{

// Заменить существующее значение

value = defs[arg];

}

else

{

// Ничего не получилось - выдать ошибку

throw new System.Exception("Некорректный аргумент: " + arg);

}

}

return value;

}

}

}