Содержание

1. Введение

В курсовой работе (КР) разрабатывается приложение Win32 для операционной системы (ОС) семейства Windows. При выполнении курсовой работы должны быть использованы особенности приложений ОС Windows, осуществлена поддержка 32-битного интерфейса программирования Win32 API, освоенные при изучении дисциплины “Операционные системы”. Приложение должно быть разработано с использованием языка C/C++ и стандартных библиотек классов (библиотека классов OWL в интегрированной среде разработки Borland C++5.) или с использованием функций API.

В работе [5] представлены требования по оформлению курсовых работ и основные положения нормативных документов, определяющих состав, содержание и форму программной документации, установленных стандартами ЕСПД и ЕСКД.

2. Цель курсового проектирования

Целью выполнения КР является приобретение практических навыков:

1. разработки:
   - -структуры приложения,
   - -алгоритмов и программ для их реализации с использованием языка C/C++ и стандартной библиотеки классов в интегрированной среде разработки и/или алгоритмов и программ для их реализации с использованием функций API;
2. отладки приложения;
3. написание пояснительной записки;
4. применение нормативных документов, регламентирующих состав, содержание и форму программной документации на разработанное приложение.

3. Порядок выполнения курсовой работы

1. Разработка технического задания для решения поставленной задачи.
2. Разработка алгоритма решения задачи.
3. Разработка структуры файла проекта.
4. Разработка спецификации файлов проекта.
5. Разработка и реализация классов « приложение » и «окно» для создания главного окна приложения.
6. Тестирование и отладка созданных классов.
7. Проектирование интерфейса приложения – структуры главного меню, информационных и диалоговых форм. Главное меню должно иметь команды, позволяющие решать поставленную задачу, получать информацию об авторе программы, завершать работу с программой.

8. Создание файла ресурсов с помощью редактора ресурсов, создание заголовочного файла ресурсов, подключение заголовочного файла ресурсов и идентификаторов ресурсов в соответствующие файлы проекта.
9. Проектирование «горячих клавиш».
10. Тестирование и отладка созданного интерфейса приложения.
11. Разработка и реализация классов и функций, необходимых для решения задачи.
12. Тестирование и отладка окончательного варианта приложения.

4. Содержание отчета по курсовой работе

1. Техническое задание.
2. Краткие теоретические сведения:
   - Особенности создания приложений для ОС семейства Windows.
   - Краткая характеристика функций API.
   - Краткая характеристика стандартной библиотеки классов.
   - Краткая характеристика интегрированной среды разработки.
3. Состав и характеристики файлов проекта.
4. Стандартные классы и функции приложения.
5. Пользовательские классы и функции приложения.
6. Системные требования.
7. Руководство пользователя.
8. Список литературы
9. Приложение. Листинги программ.

Раздел «Пользовательские классы и функции приложения» кроме краткого описания производных классов и функций приложения должен содержать иерархическую структуру классов и алгоритмы всех пользовательских функций.

Раздел «Системные требования» должен содержать минимально требуемые значения программно-аппаратной системы, необходимой для выполнения созданного приложения, а именно: тактовая частота процессора, требуемый объем оперативной памяти и физической памяти, тип операционной системы, наличие системных библиотек при динамической компоновке исполняемого кода, особые требования, определяемые поставленной задачей.

Раздел «Руководство пользователя» включает в себя:

1. правила установки и запуска программы;
2. последовательность действий для запуска программы;
3. последовательность действий для выполнения всех требуемых по техническому заданию функций.

В данный раздел также необходимо включить виды интерфейса приложения с различными вариантами диалога пользователя.

5. Краткие теоретические сведения

5.1. Программирование приложений Windows

Операционные системы Windows обладает рядом особенностей. Прежде всего – это графический интерфейс, обеспечивающий пользователю удобство в работе и привлекательное графическое изображение. ОС Windows поддерживает 32/64-битный интерфейс программирования Win32 API- (Application Programming Interface – интерфейс прикладного программирования). API- это набор, похожих на подпрограммы процедур - функций, которые программы вызывают для решения всех задач, связанных с работой ОС. Реализованы они в виде библиотек динамической компоновки .dll, основными из которых являются gdi, user, kernel. Эти библиотеки отображаются в адресное пространство каждого процесса.

Windows-приложения выполняются в собственных окнах. Каждое приложение располагает, по крайней мере, одним собственным окном. Через окна приложения выполняется ввод/вывод информации пользователя. Главное окно – это и есть само приложение, но окно – это также и визуальный интерфейс.

Работа в Windows ориентирована на события. В Windows приложения выполняются пошагово. После решение одной подзадачи, управление возвращается Windows, которая может вызывать другие программы. Windows переключается между различными приложениями. Программист инициирует событие (вызов команды меню, щелчок мыши на окне), событие обрабатывается, и программное управление передается в соответствующее приложение. Приложение вызывается для обработки события.

Таким образом, разработка приложения – это создание окна приложения (создать окно, зарегистрировать его класс, сделать его видимым) и организация обработки сообщений пользователя.

В ОС Windows для обеспечения взаимодействия различных процессов и потоков в приложении используется механизм обработки сообщений. Для того чтобы иметь возможность работать с каким-либо устройством, например, с клавиатурой или мышью, программам DOS приходилось отслеживать состояние этих устройств и ожидать их реакции на посланные им сообщения. ОС Windows управляется сообщениями, и уже не программа ожидает реакции от устройства, а сообщение о реакции устройства запускает ту или иную программу. Та часть программы, которая запускается в ответ на конкретное сообщение, называется функцией его обработки. Большую часть работы по передаче сообщений и вызову соответствующих функций обработки берут на себя внутренние процедуры Windows.

Приложения для Windows можно разрабатывать различными способами.

Первым, безусловно, является непосредственный вызов функций API.

Для экономии времени можно использовать подход визуального программирования или генераторы приложений. Типичный генератор создает файлы исходного кода из выбранных команд меню, диалоговых окон, управляющих элементов и др. После того, как оболочка приложения таким образом сконструирована, необходимо заполнить её требуемым кодом.

Однако всем генераторам присущи два недостатка. Во-первых, всё равно необходимо писать код для ядра программы, т.к. фактически с использованием генератора создаётся прототип, нечто вроде входного экрана. Во- вторых, автоматически сгенерированные программы не подлежат модернизации, т.е. приложение необходимо переписать заново при внесении изменений.

Подход визуального программирования предлагает набор объектов, которые при их активизации интерпретируют некоторые инструкции и могут объединяться с другими объектами или запрограммированными операциями. Однако подобные методы не позволяют создавать эффективно работающих программ, и приложение содержит большое количество избыточного кода. Поэтому системы визуального программирования используются для создания панелей ввода баз данных, календарей , небольших утилит и т.д.

Более приемлемым подходом является использования языка C/C++ и библиотеки стандартных классов.

Во-первых, классы C++ лучше позволяют моделировать архитектуру системы. Класс окна, например, инкапсулирует данные и функции, т.е. автоматически берет на себя выполнение некоторых внутренних требований, которые необходимо было ранее обеспечить в явном виде. Кроме того, функции-элементы класса могут, при помощи таблицы отклика на сообщение, независимо отвечать на сообщения системы. Это очень важно для управляемой событиями ОС.

Во-вторых, библиотека классов представляет каркас прикладных программ, на основе которых можно создавать приложения. Опираясь на механизм наследования, можно создавать новые классы, расширяя функциональные возможности базовых.

К недостаткам подобного подхода можно отнести, прежде всего, наличие избыточного кода при статической компоновке приложения или требование наличия в системе соответствующих библиотек при динамической компоновке, хотя визуально код приложения, составленный с помощью библиотеки классов, выглядит существенно более компактным.

К данному методу необходимо отнести следующие библиотеки.

OWL (Object Windows Library) предоставляет каркас прикладных программ, на основе которых строятся приложения (механизм наследования).

MFC (Microsoft Foundation Classes) – библиотека базовых классов, предусматривает использование классов-оболочек, заменяющих функции Windows.

В КР должен быть применен метод, заключающийся в использовании одной из стандартных библиотек или непосредственный вызов функций API.

5.2. Интегрированная среда разработки (Integrated Development Environment, IDE)

Программы для разработки приложений для Windows обычно состоят из нескольких файлов, в том числе файлов исходного кода, заголовочных, библиотек и т.д. Чтобы создать конечный исполняемый файл, необходимо не только скомпилировать все файлы с определенными заданными параметрами для компилятора, но и установить все связи между ними. Существует два способа создания исполняемого файла. Во-первых, – использование утилиты командной строки Make и создание специального файла makefile, который содержит список команд, выполняемый для создания приложения. Однако создание подобных файлов - процесс достаточно сложный, требующий, в том числе, знания языка makefile.

Все разрабатываемые среды IDE позволяют упростить этот процесс, путем использования файлов проекта. Файлы проекта используются для организации проектов программирования. Они позволяют компилятору автоматически просматривать исходные файлы и находить все взаимосвязи. Кроме этого файлы проекта визуально демонстрируются в среде и ими несложно манипулировать.

5.3. Пакет для разработки приложений для Windows Visual Studio

Наиболее распространенный на данный момент пакет для разработки приложений для Windows является Visual Studio — это полный набор инструментов и служб для создания различных приложений как для платформы Microsoft, так и для других платформ.

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Microsoft Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения

функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Набор легковесных сред разработки, представляет собой урезанную версию Visual Studio. В отличие от полной версии, каждая такая среда предназначена для какой-то одной платформы. Она включает в себя небольшой набор инструментов, в отличие от полных версий: отсутствует дизайнер классов и многие другие инструменты, а также поддержка плагинов и удалённых баз данных в дизайнере данных. Компиляторы в 64-битный код также недоступны в Express редакциях до версий 2012 года (хотя компилятор бесплатно распространяется с Windows SDK и его можно использовать, компилировать автоматически из IDE нельзя). Microsoft позиционирует эту линейку IDE для студентов и любителей. На настоящий момент существуют следующие Express редакции:

1. Visual Basic Express;
2. Visual C++ Express;
3. Visual C# Express;
4. Visual Web Developer Express.

Вместе с Visual Studio 2012, были выпущены новые Express-версии продукта:

1. Visual Studio Express 2012 for Web - для web-разработчиков;
2. Visual Studio Express 2012 for Windows - для разработки программ с modern-интерфейсом (языки: C#, VB.Net, C++, JavaScript);
3. Visual Studio Express 2012 for Windows Desktop – для разработки обычных десктопных приложений (языки: C#, Visual Basic.Net, C++);
4. Visual Studio Express 2012 for Windows Phone - для разработчиков под платформы Windows Phone 7.5 и 8.0;
5. Visual Studio Team Foundation Server Express 2012.

Ключевыми особенностями этих express-версий продуктов являются:

1. ориентирование на цель разработки, а не на язык;
2. необходимость регулярно продлевать бесплатную регистрацию для индивидуальных разработчиков, если разработка на Express-версии ведется не с целью обучения;
3. поддержка компиляции 64-битного кода;
4. поддержка unit-тестов.

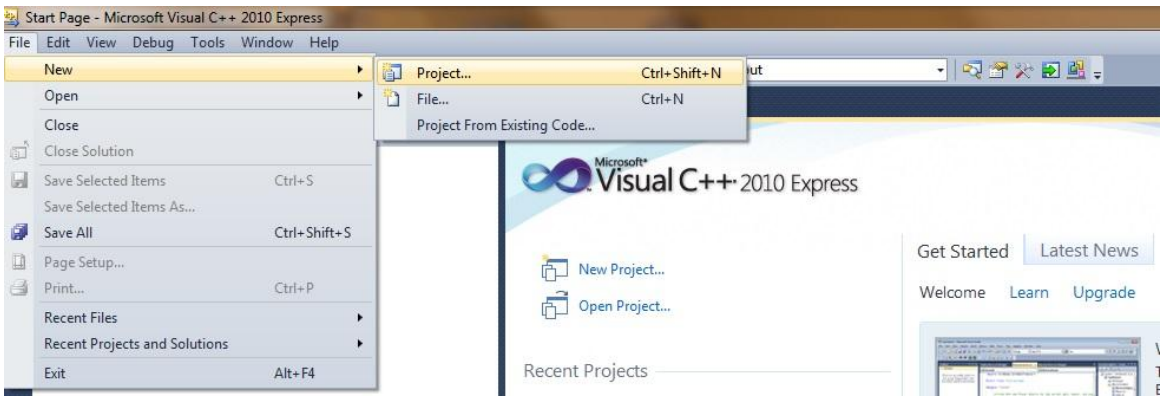

Рис. 1. – Среда разработки Visual C++ Express

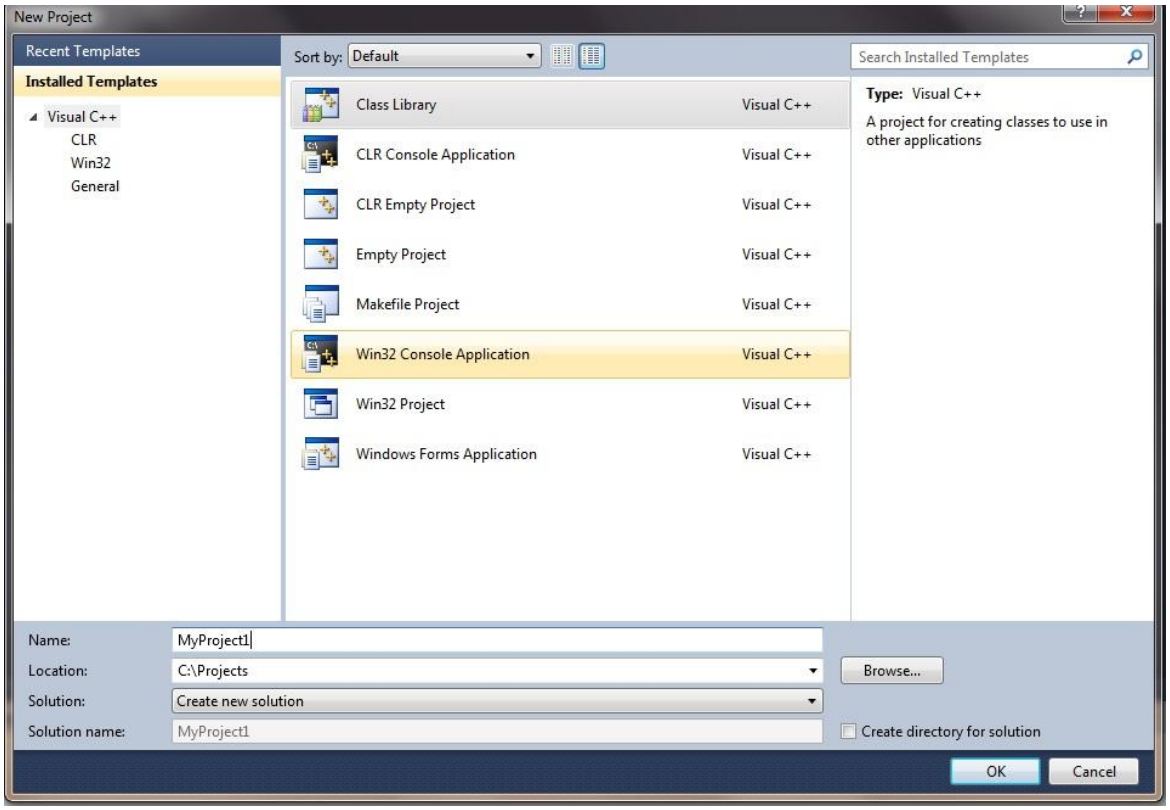

Рис. 2. – Создание файла проекта

5.4. Интегрированная среда разработки Borland C++5 (Integrated Development Environment, IDE)

Для разработки приложений Windows с поддержкой библиотеки классов OWL используется IDE Borland C++5.0, в основном, предназначенная для разработки 32-разрядных приложений, хотя в ней можно создавать и 16-ти разрядные приложения, статические и динамические библиотеки, а также консольные приложения. В данной среде можно также создавать приложения с

использованием стандартных функций API, однако, в ней затруднительно использование дополнительных библиотек динамической компоновки, описывающие дополнительные функции API.

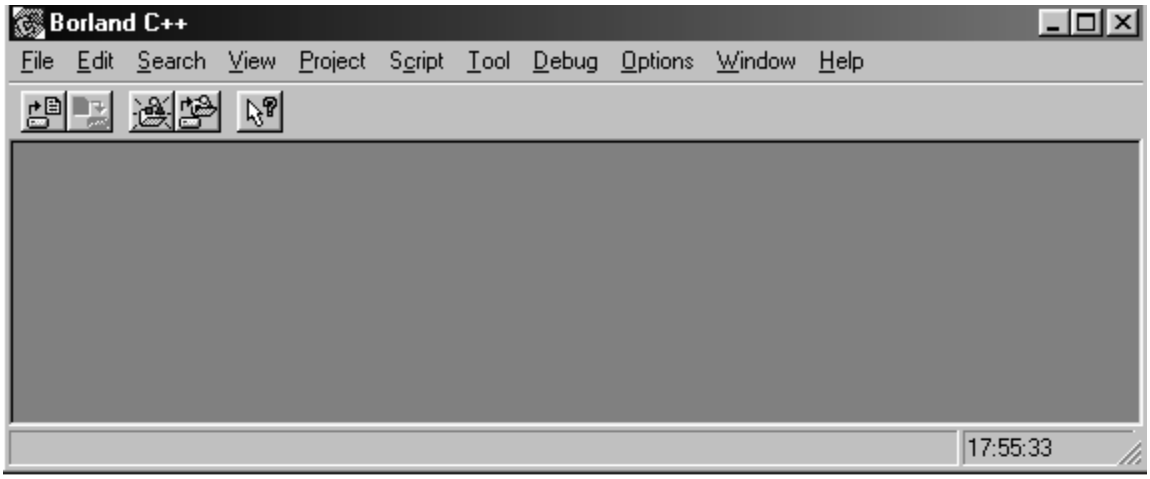

Рис. 3. - Начальный кадр IDE Borland C++5.0

Среда содержит большое число настраиваемых параметров, доступ к которым осуществляется через команду Options главного меню.

Создание проекта.

Как уже указывалось выше, проект служит для объединения всех файлов, входящих в приложение в единый комплекс. Обычно исходные тексты самого простого приложения содержит не один, а, по меньшей мере, три файла:

1. файл -.cpp с текстом программы;
2. файл ресурсов -.rc, в котором описываются конфигурации меню, и диалоговых окон, курсоры, иконы приложения, битовые матрицы и т. д.;
3. файл определения модуля -.def с информацией о типе создаваемого модуля, некоторых его характеристиках.

Сам файл проекта имеет расширение -.ide.

Для создания файла проекта выберем команду File ->New-> Project. Откроется панель New Target (новая мишень) (рис. 2). В верхних окошках Project Path и Name этой панели задаются имя файла проекта. В первом окне пропишется каталог, который был задан.

В рамке Platform необходимо установить Win32 (32-битовое приложение), в рамке Target Model устанавливается GUI, в рамке Target Type установится тип мишени – выполняемый файл приложения Application [-.exe].

Если не требуются специальные условия выполнения исполняемого файла, файл определения модуля можно не создавать, а воспользоваться стандартным файлом default.def, который есть в системе по умолчанию.

Пример стандартного файла определения модуля представлен в приложении (пример 1).

Рис. 4. – Создание файла проекта

После нажатия всех OK появится окно проекта:

Рис. 5.– Окно проекта

Чтобы посмотреть содержание текстовых файлов (-.cpp, -.def) надо дважды щелкнуть на их имени мышью или выделить соответствующую строку и нажать Enter.

В окне проекта можно корректировать состав проекта (добавление новых файлов и удаление существующих) и изменять характеристики проекта (опять открыть Target Expert- щелкнуть дважды правой кнопкой мыши по строке с .exe файлом).

Для запуска процесса компиляции и компоновки можно воспользоваться пунктом меню Project->Build all (построение проекта) или нажать кнопку Build Project из строки кнопок, находящихся в подменю.

После окончания обработки, откроется окно построителя задачи, в котором сообщается об отсутствии или наличии ошибок и предупреждений.

Если ошибки и предупреждения были, список их представляется в окне сообщений Message.

Запустить приложение можно, используя пункты меню, а можно щелкнуть по кнопке Run.

Создание файла сценария ресурсов.

Ресурсы представляют собой двоичные данные, хранящиеся внутри exe–файла программы. Команды меню, например, организованы как ресурсы, которые программа загружает во время исполнения.

Ресурсы экономят время, сокращая объем подготовительных операций, выполняемый программой, и экономят память, так как двоичные ресурсы остаются на диске до тех пор, пока они не потребуются.

Файл сценария ресурсов представляют собой обыкновенный текстовой файл, содержащий операторы исходного кода ресурсов приложения, т.е. он может быть составлен в текстовом редакторе с использованием команд сценария ресурсов. Команды сценария ресурсов полностью документируются оперативной подсказкой среды. В приложении представлен пример сценария для простого меню (пример 2).

Однако в состав среды входит обычно интерактивный визуальный редактор ресурсов RW (Resource Workshop), который позволяет визуально создавать ресурсы пользователю. Законченный сценарий затем можно просматривать и редактировать в текстовом виде.

После создания или изменения в визуальном редакторе требуемого ресурса, необходимо сохранить файл сценария ресурсов.

Заголовочные файлы.

Помимо описанных выше файлов в проект могут входить заголовочные файлы -.h и заголовочные файлы ресурса -.rh, подключаемые в исходный текст (-.cpp) с помощью директивы #include. Сами файлы добавляются непосредственно в проект. Необходимости в этих файлах, особенно в -.h нет, так как все их содержимое может быть размещено в исходном файле, однако, их использование уменьшает объем исходного файла и облегчает работу с ним. В заголовочные файлы -.h обычно выносят прототипы прикладных функций,

используемых в программе, определения констант, разработанные макросы и так далее.

Заголовочные файлы ресурса -.rh включают в себя идентификаторы ресурсов, определяемые директивой #define. В случае отсутствия заголовочного файла идентификаторы определяются непосредственно в файле ресурса и, следовательно, сам файл ресурса должен быть подключен в исходный текст приложения с помощью директивы #include, что полностью лишает приложение преимуществ, связанных с выделением ресурсов в отдельный файл.

В приложении представлен пример заголовочного файла ресурса (пример 3).

5.5. Основы программирования для Windows с помощью функций API

API это программный интерфейс приложения. Другими словами, это те возможности, которые предоставляет операционная система Windows для использования прикладными программами. Системные функции, которые предоставляет Windows программисту, называются ещё функциями API. Программирование с использованием только этих функций называется API-программированием.

Структура API-программ

Классическая структура API-программы определяется четырьмя компонентами: инициализация; цикл ожидания, или цикл обработки сообщений; функция главного окна; другие функции. В простейшем случае последний компонент может отсутствовать. Два первых компонента располагаются в функции WinMain.

Функция WinMain:

```
int WINAPI WinMain
(
HISTANCE hInstance,
HINSTANCE hPrevInctance,
LPSTR lpCmdLine,
int nCmdShow
).
```

Функция WinMain вызывается системой, в которую передаются четыре параметра:

1. hInstance - дескриптор текушего экземпляра приложения;
2. hPrevInctance - всегда равен NUL;
3. lpCmdLine - указатель на командную строку запускаемой программы;
4. nCmdShow - способ визуализации окна.

Инициализация

Здесь производится регистрация класса окна, его создание и вывод на экран. Регистрация класса окна осуществляется функцией

RegisterClass(CONST WNDCLASS *lpwcx).

Единственный параметр функции - указатель на структуру WNDCLASS. После того как класс будет зарегистрирован, окно из данного класса может быть создано функцией CreateWindow.

```
typedef struct _WNDCLASS
{
UNIT    style;
WNDPROC lpfnWndProc;
int  cbClsExtra;
int  cbWndExtra;
HANDLE  hInstance;
HICON  hIcon;
HCURSOR hCursor;
HBRUSH  hbrBackground;
LPCTSTR lpszMenuName;
LPCTSTR lpszClassName;
} WNDClASS
```

Перечислим некоторые типичные значения членов структуры:

1. Стили класса окон. Стиль окна определяется комбинацией нескольких предопределённых констант. Довольно часто он полагается нулю, что означает "стиль по умолчанию".

2. Дескриптор иконки окна. Определяется с помощью функции LoadIcon. Первым параметром данной функции является дескриптор приложения, вторым - строка, определяющая имя иконки в ресурсах. Для того чтобы задать одну из стандартных иконок, первый параметр должен иметь значение NULL, а второй значение одной из следующих констант:

- IDI_APLICATION - стандартная иконка приложения;
- IDI_ASTERISK - иконка "информация";
- IDI_EXCLAMATION - "восклицательный знак";
- IDI_HAND - "знак Стоп";
- IDI_QUESTION - "вопросительный знак".

3. Дескриптор курсора. Для определения курсора используется API-функция LoadCursor. Функция похожа на функцию LoadIcon.

4. Имя класса. Название класса - это просто строка, которая потом используется при создании окна.

Создаётся окно функцией CreteWindow:

```
HWND CreateWindow
(
 LPCTSTR lpClassName, //указывает на имя зарегистрированного окна;
 LPCTSTR lpWindowName, //название окна;
 DWORD dwStyle, //стиль окна:
 int x, //горизонтальная координата;
 int y, //вертикальная координата;
 int nWidth, //ширина окна;
```

```
int nHeight, //высота окна;
HWND hWndParent, //дескриптор родителя или владельца окна;
HMENU hMenu, //дескриптор меню окна;
HANDLE hINSTANCE, //дескриптор приложения;
LPVOID lpParam //указатель на дополнительную информацию;
).
```

Функция возвращает дескриптор созданного окна, при ошибке – 0.

Для того чтобы корректно отобразить окно на экране, следует выполнить ещё две функции:

3. BOOL ShowWindow(HWND hWnd, int nCmdShow) - эта функция отображает окно на экране. Первый параметр - дескриптор окна, второй - режим отображения.
4. BOOL UpdateWindow(HWND hWnd) - вызов данной функции приводит к немедленной перерисовке окна и посылке функции окна сообщения WM_PAINT.

Цикл обработки сообщений

Цикл обработки сообщений присутствует во всех приложениях Windows. Правда, не всегда этот цикл представлен явно в программе.

```
while (GetMessage(&msg, NULL, 0, 0))
{
  TranslateMessage(&msg);
  DispatchMessage(&msg);
}
```

В цикле сообщения присутствует три функции. Эти функции есть там всегда, но кроме них в цикле могут быть и другие. Функция GetMessage() выбирает из очереди сообщений приложения очередное приложение. вместо этой функции используют так же функции PostMessage() и PeekMessage(). Во всех трех функциях присутствует указатель на строку MSG:

```
typedef struct tagMSG
{
HWND hwnd;
UINT message;
WPARAM wParam;
LPARAM lParam;
DWORD time;
POINT pt;
}
```

MSG:

1. hwnd - дискриптор окна;
2. message - код сообщения;
3. wParam - дополнительный параметр;
4. lParam - дополнительный параметр;
5. time - время посылки сообщения;

6. pt - положение курсора мыши;
7. Прототип функции MessageBox().

```
BOOL
GetMessageBox
(
LPMSG lpMsg;
HWND hWnd;
UINT wMsgFilterMin,
UINT wMsgFilterMax;
).
```

Первый параметр функции - указатель на строку MSG, куда и будет помещена получаемая информация. Вторым параметром является дескриптор окна, которому предназначено сообщение. Если параметр равен NULL, то "отталкиваются" все сообщения, получаемые приложением. Два последних параметра определяют диапазон сообщений. Для того чтобы получать сообщения из всего диапазона, эти параметры должны быть равны 0.

Функция TransleteMessage() преобразует сообщения в единый формат. Функция DispatchMessage() перенаправляет сообщение оконной процедуре.

Оконная функция

Это еще один компонент приложения, отвечающий за обработку сообщений окна. Эта функция вызывается системой и имеет четыре параметра, совпадающих с первыми членами структуры MSG. В приложении представлен листинг программы, создания простого окна приложения (пример 4).

5.6. Основы программирования для Windows с помощью библиотеки объектов OWL

Программирование на основе библиотеки классов C++ позволяет достаточно хорошо моделировать архитектуру ОС Windows, систему, хотя и не являющуюся формально объектно-ориентированной, но основанную на объектно-ориентированной идеологии.

Эти библиотеки представляют собой составленные, готовые описания всех объектов, с которыми приходится иметь дело при разработке приложения: окон, кнопок, линеек прокрутки, инструментальных панелей, окон диалога и т.д.

Классы каждого из этих объектов включают все необходимые для работы с объектом данные и функции. Класс автоматически берет на себя выполнение некоторых внутренних требований, которое при использовании только C, необходимо было обеспечить в явном виде.

OWL представляет каркас прикладных программ, на основе которых можно строить свои приложения. На основе механизма наследования, создаются новые классы, расширяя функциональные возможности базовых.

OWL-программа как правило состоит из трех выделенных частей:

1) описания главной функции OwlMain();

2) описания класса приложения и входящих в него функций;

3) описания класса главного окна приложения и входящих в него функций.

Главная функция OwlMain() берет на себя управление при запуске приложения, заменяет функцию WinMain() в «классических» приложениях Windows и main() в приложениях DOS.

Функция OwlMain() в качестве параметров получает два аргумента: значение типа int и массив указателей на char. Эти параметры обеспечивают связь программы с командной строкой. Первый параметр указывает количество параметров командной строки, массив указателей представляет сами строковые параметры. Функция возвращает значение типа – результат выполнения приложения (успешное завершение или нет). Заголовок:

int OwlMain (int argc, char* argv[ ])

Функция выполняет две основные обязанности:

1. создает объект приложения класса TApplication или производного от него;

2. вызывает функцию – элемент Run этого объекта.

Так как библиотека OWL содержит описания классов для реализации практически всех средств Windows, то задача программиста заключается в подборе библиотечных классов для реализации всех свойств приложения, и создании на базе библиотечных классов производных, в которых библиотечные функции–члены могут при надобности модифицироваться (замещаться). Во многих случаях библиотечные функции не требуют модификации и тогда используются сами библиотечные классы, создаются их объекты и вызываются для них соответствующие компонентные функции.

Для создания приложения (экземпляра класса) используется класс приложений TApplication производный от класса TModule. Функции этого класса организуют создание главного окна и обрабатывают сообщения.

Базовый класс TModule определяет фундаментальные свойства модулей – приложений и библиотек с исполняемым кодом.

Объект Tapplication – это определение функций и данных приложения, которые без OWL определялись бы глобально.

Приложение и окно – это не одно и тоже. Для задания характеристик и свойств окна приложения нужно вмешиваться в работу функции Run() – компонентной функции класса TApplication. Необходимо создать производный класс приложения от класса TApplication

Рассмотрим функцию Run (). Основной задачей этой функции является последовательный вызов других функций, принадлежащих классу TApplication и классу TWindow. Во-первых, эта функция инициализирует приложение, вызывая InitApplication() для первого экземпляра приложения и InitInstance() для всех экземпляров. Во- вторых, если инициализация прошла успешно,

вызывается функция MessageLoop() для инициализации цикла обработки сообщений.

Функции InitMainWindow() создает новый объект класса TFrameWindow, определяющий характеристики главного окна.

После создания объекта класса TFrameWindow можно обращаться к компонентным функциям этого класса, что и делает функция InitMainWindow(). Она вызывает функции TWindow::Creat() и ShowWindow(), выполняющие создание и показ окна и вызывает компонентную функцию SetMainWindow() , которая объявляет новое окно главным.

Именно на этом этапе можно изменить расположение, размеры и стиль окна. Так как при создании объекта класса TApplication создается объект TFrameWindow - чистое ни к чему не пригодное окно с рамкой, необходимо создать от TframeWindow новый класс и объекты собственного класса (собственные окна).

Класс TFrameWindow является проводным от класса Twindow и наследует огромное количество функций, обеспечивающих общие черты поведения окон,

Помимо функций в класс TWindow входят ряд данных с элементами, определяющими атрибуты окна :стиль, координаты, размеры.

Важным элементом любого приложения Windows является меню, которое позволяет, с помощью выбора пункта меню, выполнять определенные действия. Для обработки сообщений Windows необходимо в программе предусмотреть следующие действия и элементы

1. объявление в классе главного окна таблицы отклика (макрос DECLARE_RESPONSE_TABLE с именем конкретного класса в качестве параметра);

2. таблица откликов (макросы DEFINE_RESPONSE_TABLEx и END_RESPONSE_TABLE, а также помещаемые между ними макросы, определяющие конкретные действия);

3. набор функций, обрабатывающих по заданным алгоритмам описанные в таблице откликов сообщения Windows.

Следует отметить, что для всех типов ресурсов, кроме создания файла ресурса, требуется выполнить следующие операции:

1. загрузить с использованием соответствующей ресурсу функцией, требуемый ресурс, где параметром функции является идентификатор ресурса;
2. вызвать обработку ресурса явно или через сообщения Windows с использованием макросов или определенных для данного ресурса функций.

Создание меню приложения представлено в примере КР.

Таким образом, для разработки приложения необходимо выполнить прежде всего действия, описанные ниже.

1. Определить функцию OwlMain(). Внутри функции создается объект приложения класса, производного от Tapplication, и вызывается функция Run(),

которая вызывает приведенные выше виртуальные функции для создания объекта

2. Определить собственный класс приложения, производного от TApplication, в котором следует дать прототипы функций-замещений.

3. Определить класс главного окна приложения, производного от класса окна с рамкой (TFrameWindow). В случае необходимости можно задать параметры окна.

4. Переопределить (в случае необходимости) виртуальные функции класса TApplication.

5. Назначить типовые ресурсы главного окна.

6. Назначить дополнительные ресурсы.

Следует отметить, что подобные действия, хотя и являются обязательными, позволяют создать лишь каркас приложения, фактически конструируя его интерфейс. Для выполнения приложением конкретных задач, необходимо сконструировать необходимые функции и разработать соответствующие классы, выполняющие требуемые действия.

Контекст устройства относится к числу системных ресурсов, количество которых в системе ограничено, а работа с такими ресурсами протекает одинаково: сначала надо получить у системы требуемый ресурс, но, закончив с ним работу, – вернуть системе. Таким образом, для того, чтобы вывести в окно некоторое изображение, необходимо выполнить действия, последовательность которых, в сущности, определяется алгоритмом обработки сообщения WM_PAINT для данного приложения:

1. получить у системы контекст устройства для данного приложения (для вывода в окне приложения);
2. изменить при необходимости режимы рисования или характеристики конкретных инструментов;
3. сформировать с помощью графических функций GDI изображение;
4. вернуть Windows занятый у нее контекст устройства.

Библиотека классов OWL обеспечивает поддержку Графического интерфейса Windows (GDI), основанную на классах, наследуемых от базового класса TGDIBase.

Иерархия классов идет по двум главным ветвям: производные классы контекста устройства и производные классы графических объектов.

Контекст устройства - связывающее звено между программным кодом и областью рисования. Надо позаботится о взаимодействии программы с контекстом устройства. Для этого необходимо создать объект одного из классов контекста устройства и вызывать для этого объекта графические функции GDI, инкапсулированные в данном классе.

Следует отметить особенности вывода битовых матриц в Windows. Нет функций непосредственного вывода растровых изображений на экран (даже через контекст окна). Можно копировать изображения с одного устройства на другое устройство, используя соответствующие контексты. Функции– BitBlt()

для копирования без изменения масштаба и StretchBlt() – для копирования с уменьшением или с увеличением. Устройство приемник – окно, псевдоустройство источник – участок оперативной памяти по своим возможностям совместимым с экраном. Сначала надо поместить изображение на псевдоустройство, с ним связан контекст памяти, а затем скопировать его в окно. Для проверки правильности вводимых значений в конструкторах классов диалога создаются следующие объекты:

6. Примерное содержание курсовой работы

6.1. Техническое задание

Разработать приложение, выводящее в окно графики стандартных функций,

В меню ввести следующие пункты: «О программе», «Выход», «Ввод размеров осей координат», «Очистка», пункт меню, изменяющий цвет графиков, а также создать объект плавающего меню, в котором можно выбрать масштаб вывода графиков. Создать собственную иконку и курсор для приложения, а также выполнение пунктов меню с помощью “горячих” клавиш.

6.2. Результаты выполнения программы

Предусмотрена возможность отображения на экране любого сочетания доступных графиков. Для вывода в пункте меню «Графики» необходимо щелкнуть левой клавишей мыши на нужном графике На рис 6. произведен вывод всех графиков .

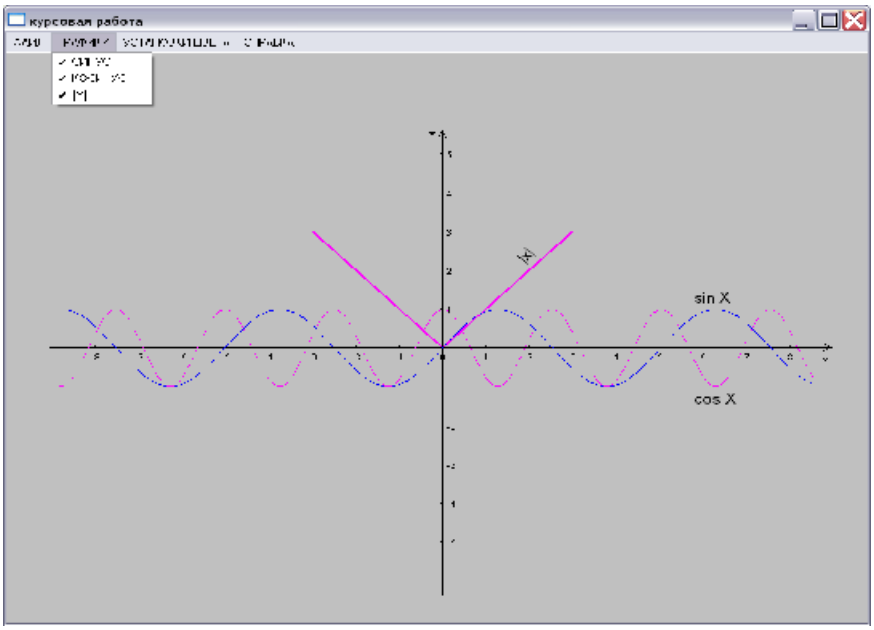

Рис. 6.- Вывод всех графиков

Вывод графика сопровождается появлением маркера перед соответствующей командой меню, а вторичный выбор этой команды гасит как

маркер, так и сам график. В меню &Файл с помощью выбора пункта меню &Очистка все выведенные на экран графики удаляются.

Щелчком правой клавиши мыши активизируется плавающее меню, в котором можно выбрать масштаб вывода графиков. На рисунке 7. выбран масштаб 100 пикселей.

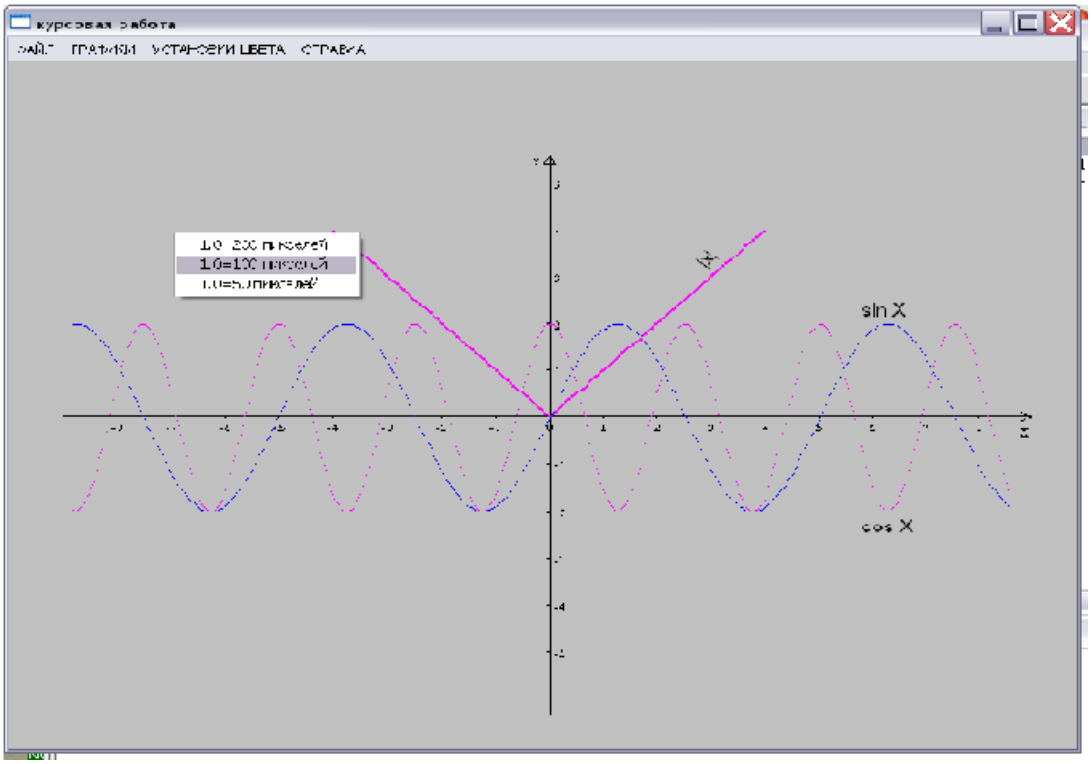

Рис. 7.-Вызов плавающего меню

С помощью пункта меню &Ввод размеров осей на экране появляется диалоговое окно, в котором можно задать значения для изменения длины осей координат. На рисунке 6. изображено диалоговое окно для ввода.

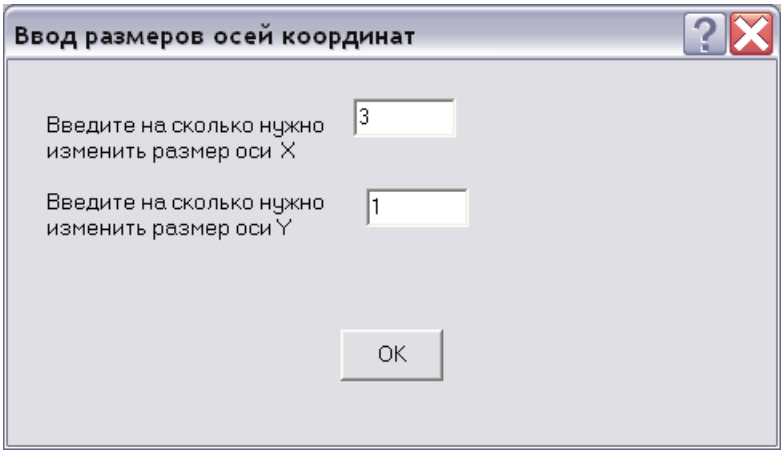

Рис. 8. – Диалоговое окно ввода

Кроме обязательных классов, определяющих интерфейс приложения: класс приложения MyApplication, производный от Tapplication, класс главного окна MyWindow от TframeWindow , и главной функции OwlMain(), в программе вводятся класс диалога для введения осей размеров в программу - Сами формы окон диалога разработаны в RW.

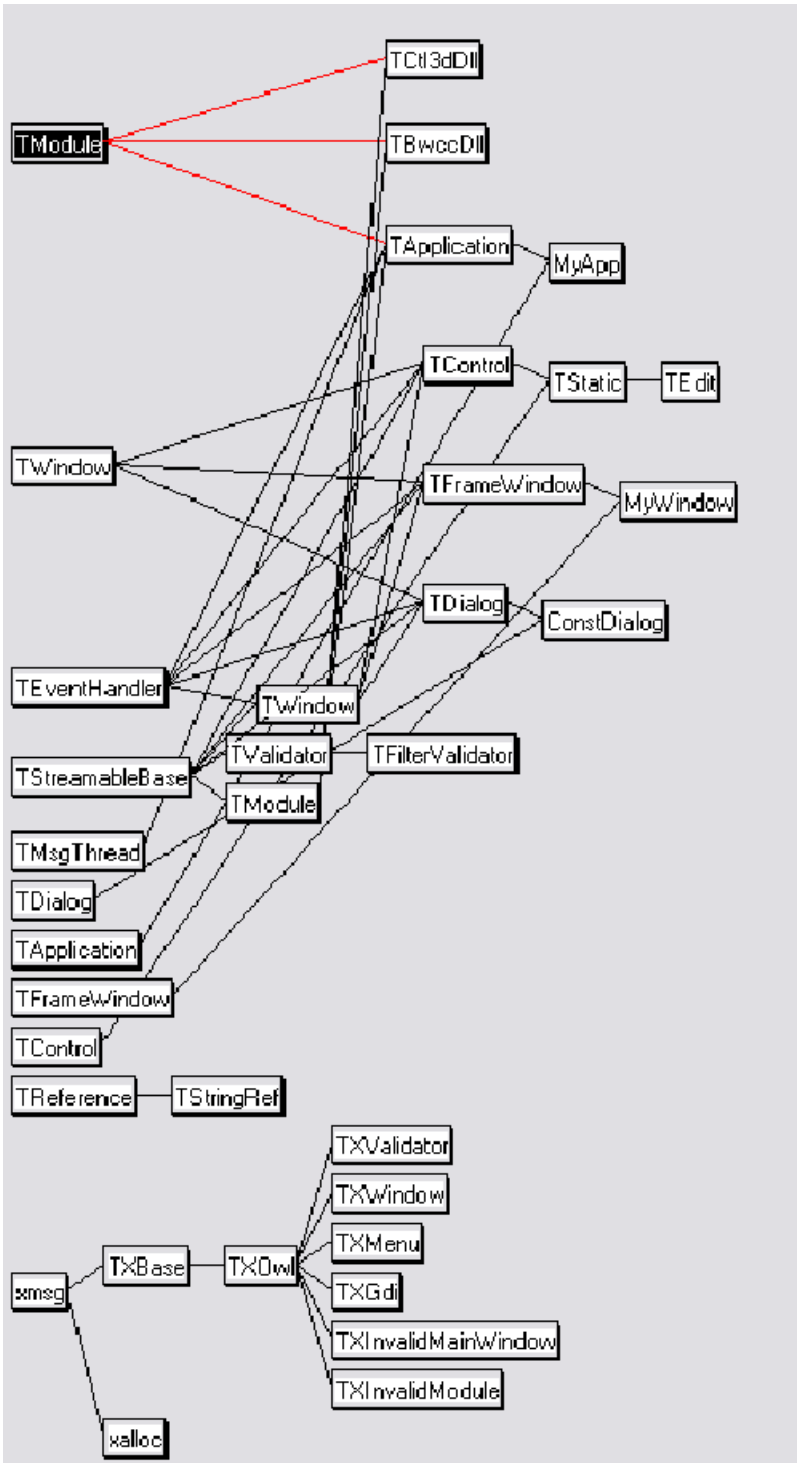

Рис. 9. - Структура классов приложения

7. Примерные задания на курсовую работу

Создать приложение :

1. выводящее изображение файла .bmp.;
2. выводящее изображение файла .bmp и использующее функцию ползунка;
3. выводящее в окне диаграмму статического массива данных, каждый элемент диаграммы представить различными цветами;
4. выводящее в окне диаграмму динамического массива данных;
5. выводящее в окне диаграмму динамического массива данных, каждый элемент диаграммы представить различными цветами;
6. выводящее в окне диаграмму динамического массива данных, каждый элемент диаграммы представить различными цветами, добавить пункт меню, изменяющий цвет диаграммы;
7. выводящее в окне значения элементов статического массива умноженного на заданную константу, добавить пункт меню, изменяющий цвет массива;
8. выводящее в окне значения элементов динамического массива умноженного на заданную константу;
9. выводящее в окне значения элементов динамического массива умноженного на заданную константу, добавить пункт меню, изменяющий цвет массива;
10. выводящее в окне графики стандартных функций, размеры осей графика заданы;
11. выводящее в окне графики стандартных функций, размеры осей графика задаются динамически;
12. выводящее в окне графики стандартных функций, размеры осей графика задаются динамически, добавить пункт меню, изменяющий цвет графика;
13. выводящее в окне графики произвольных функций;
14. -текстовый редактор;
15. -редактор системного реестра;
16. -диспетчер задач ОС;
17. -картотека данных;
18. -тестирование.

Дополнения:

1. Ввести акселераторы клавиатуры.
2. Ввести в меню дополнительные пункты – “Выход”, “О программе”.
3. Созлать собственную иконку и курсор для приложения.

8. Литература

1.Подбельский В.В. Язык Си++. – М: Финансы и статистика, 1996.

2.Финогенов К.Г. Прикладное программирование для Windows на Borland C++. – Обнинск: Принтер, 1999.

3. Т.Сван. Программирование для Windows в Borland C++.- М: Бином, 1995.

4.Черкасова Н.И. Тексты лекций по дисциплине «Операционные системы» для студентов специальности 220100 дневного обучения, часть 2,.- М., РИО МГТУ ГА, 2003.

5.Горнец Н.Н., Рощин А.Г., Черкасова Н.И., Половов Р.М., Климова Т.Д Пособие по выполнению курсовых и дипломных проектов и работ для студентов специальности 220100 дневного обучения.- М., РИО МГТУ ГА, 2002.

Приложение

Пример 1

Листинг файла определения модуля - .def.

```
EXETYPE WINDOWS // тип приложения
CODE PRELOAD MOYEABLE DISCARDABLE
/*Сегмент загружается в память при запуске приложения, сегмент может быть перемещен, сегмент может быть выгружен, чтобы освободить пространство в памяти */
DATA PRELOAD MOVEABLE MULTIPLE
/* Существуют несколько сегментов данных, загружающихся автоматически заранее, возможно перемещение сегментов */
HEATSIZE    4096 //устанавливает размер локальной дин. памяти
STACKSIZE 8192 // устанавливает размер стека
```

Пример 2.

Листинг файла сценария ресурса меню - .rc.

```
#define MENU_1   1
MENU_1 MENU
{
POPUP "menu"
 {
MENUITEM "test", CM_POPUPITEM10
MENUITEM "exit", CM_POPUPITEM11
 }
 POPUP "HELP"
 {
  MENUITEM "ABOUT", CM_POPUPITEM12
}
}
```

Пример 3
Листинг заголовочного файла ресурса - rh.

```
#define MENU_1   1
```

Пример 4.
Листинг программы функций API;

```
#include <windows.h>
#include <windowsx.h>
#include <string.h>
void Register (HINSTANCE );
void Create (HINSTANCE, int);
LRESULT CALLBACK   WndProc (HWND ,  UINT ,
WPARAM  ,  LPARAM );
char  szClassName[] = " mainWindow";
char szTitle[] = "Progr -1";
HINSTANCE hPrevInst;

// Инициализация

// Функциян WinMain является точкой входа в программу
// Регистрирует окно, создает его и входит в цикл сообщений
// Прекращается при неудачном первом или втором шаге
int WINAPI WinMain (HINSTANCE  hInstance, HINSTANCE hPrevInstance,
                                    LPSTR lpszCmdParam, int nCmdShow)
{
MSG  Msg;
char szText[80];
hPrevInst = hPrevInstance;
if  (hPrevInstance==NULL)
Register(hInstance);
else
{
int                    err=MessageBox(                    NULL,"exit?",
"Info",MB_YESNO|MB_ICONQUESTION);
 if(err==IDYES)
 return 0;
 }
Create(hInstance, nCmdShow);
while  (GetMessage(&Msg, NULL, 0, 0))
DispatchMessage(&Msg);
```

```
wsprintf(szText, "kod vozvrata =%d, kod soobseniy = %x" ,Msg.wParam,
Msg.message);
MessageBox( NULL,szText, "kod vozvrata",MB_ICONINFORMATION);
return Msg.wParam;
}

// Регистрация окна

void Register (HINSTANCE hInst)
{
WNDCLASS WndClass;
memset(&WndClass, 0, sizeof(WndClass ));
WndClass.lpfnWndProc       = WndProc;
WndClass.hInstance         = hInst;
WndClass.hIcon             = LoadIcon (hInst, "Icon");
WndClass.hCursor           = LoadCursor (hInst, "Cursor");
WndClass. hbrBackground    = (HBRUSH) (COLOR_WINDOW+1);
WndClass.lpszClassName     = szClassName;
 RegisterClass (&WndClass);
}

// Создать и показать окно

void Create (HINSTANCE hInst, int nCmdShow)
{

HWND hwnd = CreateWindow(szClassName, szTitle,
WS_OVERLAPPEDWINDOW, 10, 10, 300,100,
HWND_DESKTOP, NULL,hInst, NULL);
ShowWindow (hwnd, nCmdShow);
char szText[80];
wsprintf(szText, "hPrevInstance  = %x, hInst  = %x, hwnd = %x" ,hPrevInst  ,
hInst , hwnd );
MessageBox( NULL,szText, "deskriptori",MB_ICONINFORMATION);
}

// Реализация
// Оконная процедура помогает управлять выполнением программы

LRESULT CALLBACK   WndProc (HWND hwnd,  UINT Message,
                    WPARAM  wParam,  LPARAM lParam)
{ HDC hdc;
char szText[]="bbbbbbbb hhhhhhh kkkkkkkk";
```

```
switch (Message)
{
 case WM_PAINT:
 PAINTSTRUCT ps;
 hdc =BeginPaint(hwnd, &ps);
 TextOut(hdc, 5, 30, szText, strlen(szText));
 EndPaint(hwnd, &ps);
 return 0;
case WM_DESTROY:
PostQuitMessage(255);
return 0;
default:
return (DefWindowProc(hwnd, Message, wParam, lParam));
}
}
```