Министерство науки и высшего образования
Российской Федерации
Братский педагогический колледж
федерального государственного бюджетного
образовательного учреждения высшего
образования
«Братский государственный университет»

МДК 01.04. СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

методические указания по выполнению лабораторных работ

для студентов III курса
очной формы обучения
специальности
09.02.07 Информационные системы и программирование

Автор: А.В.Конаков

Братск, 2020

МДК 01.04. Системное программирование. Методические рекомендации по выполнению лабораторных работ. / Сост. А.В. Конаков. - Братск, 2020.- 82 с.

Содержат указания к выполнению практических работ по дисциплине «Системное программирование». В практических работах содержатся основные теоретические сведения, касающиеся техники выполнения работ по работе в программе Visual Studio с языками программирования C++ и C#.

Предназначены для студентов специальности 09.02.07 Информационные системы и программирование.

Печатается по решению научно-методического совета
Братского педагогического колледжа ФГБОУ ВО «БрГУ»
665709, г. Братск, ул. Макаренко 40

СОДЕРЖАНИЕ

Пояснительная записка	4
Практическая работа № 1 Осуществление разработки кода программного модуля	5
Практическая работа № 2 Разработка кода модуля с рисованием в окне геометрических фигур	9
Практическая работа № 3 Разработка кода программного модуля с обработкой сообщений мыши. Отслеживание курсора мышки	33
Практическая работа № 4 Разработка кода программного модуля с использованием пути	22
Практическая работа №5 Выполнение отладки и тестирования программы на уровне модуля. Основные этапы разработки программного обеспечения. Работы с окнами в приложении. Отладка модуля.	36
Практическая работа № 6 Разработка кода программного модуля с использованием функций создания графического образа. Отладка программы.	58
Практическая работа №7 Работа с растровым изображением.	62
Практическая работа №8 Разработка кода программного модуля с использованием функций библиотеки DLL.	71
Практическая работа №9 Проектирование структуры документа с технической документацией программного продукта. Расположение материала. Типы информации и их компоновка. Разработка пояснительной записки в текстовом редакторе. Создание руководства пользователя программного продукта.	78
Список литературы	80

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические указания по выполнению лабораторных работ по МДК 01.01 «Системное программирование» предназначены для студентов специальности 09.02.03 «Программирование в компьютерных системах».

Методические указания направлены на формирование умений, знаний, общих и профессиональных компетенций в соответствии с требованиями к результатам освоения программы подготовки специалистов среднего звена ФГОС СПО по специальности 09.02.03 «Программирование в компьютерных системах» и рабочей программы ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Системное программирование».

Лабораторные работы посвящены изучению архитектуры вычислительной машины и ее функционирования, принципов обработки машинных команд, особенностей реализации обработки различных типов данных. Рассматриваются принципы низкоуровневого программирования разветвляющихся и циклических вычислительных процессов, обработки массивов, подпрограмм с особенностями передачи в них параметров. Основная цель практических работ – сформировать у студентов глубокое понимание процессов, протекающих в вычислительных машинах при выполнении различного рода программ, что позволит значительно повысить качественный уровень и эффективность будущих разработок вычислительных систем.

Тема 1.Интерфейс Windows приложений. Осуществление разработки кода программного модуля.

Стандартная заготовка Windows приложений. Обработка сообщений. Создание программы по разработанному алгоритму.

Рассмотрим сначала, как можно "вручную" создать минимальное Win32-приложение. Загрузив Visual Studio 2010, выполним команду File | New | Project… и выберем тип проекта — Win32 Project. В раскрывающемся списке Location выберем путь к рабочей папке, а в поле Name имя проекта (рис. 1.2). В следующем диалоговом окне, приведенном на рис. 1.3, нажимаем кнопку Next, а в окне опций проекта (рис. 1.4) выберем флажок Empty project (Пустой проект) и нажмем кнопку Finish — получим пустой проект, в котором нет ни одного файла.

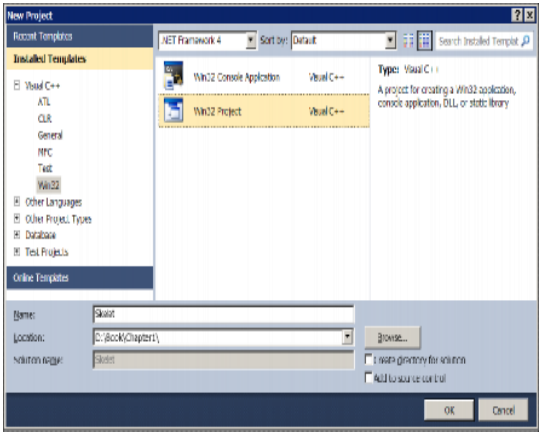

Рис. 1.2. Выбор типа проекта

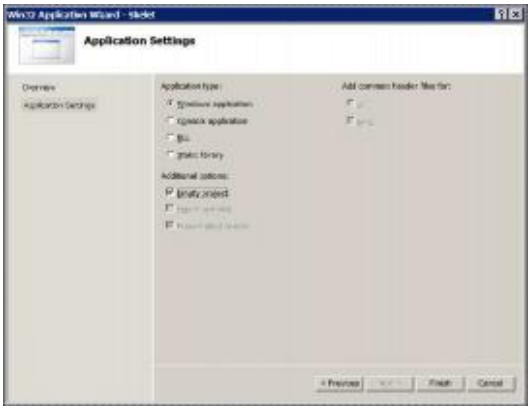

Рис. 1.3. Выбор типа проекта

Рис. 1.4. Окно опций проекта

Рис. 1.5. Добавление к проекту нового объекта с помощью контекстного меню

Рис. 1.6. Выбор шаблона объекта

С помощью контекстного меню (рис. 1.5) добавим файл для кода приложения, имя файла введем в ходе диалога выбора шаблона объекта на рис. 1.6. (Тот же самый диалог мы могли бы получить по команде меню Project | Add New Item….)

ПРИМЕЧАНИЕ

Пока мы создаем простые решения, состоящие из одного проекта, можно убрать флажок Create directory for solution. Это упростит структуру каталога.

#include

#include

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

TCHAR WinName[] = _T("MainFrame");

int APIENTRY _tWinMain(HINSTANCE This, // Дескриптор текущего приложения HINSTANCE Prev, // В современных системах всегда 0

LPTSTR cmd, // Командная строка

int mode) // Режим отображения окна

{

HWND hWnd; // Дескриптор главного окна программы MSG msg; // Структура для хранения сообщения WNDCLASS wc; // Класс окна

// Определение класса окна wc.hInstance = This;

wc.lpszClassName = WinName; // Имя класса окна wc.lpfnWndProc = WndProc; // Функция окна wc.style = CS_HREDRAW | CS_VREDRAW; // Стиль окна

wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // Стандартная иконка wc.hCursor = LoadCursor(NULL,IDC_ARROW); // Стандартный курсор wc.lpszMenuName = NULL; // Нет меню

wc.cbClsExtra = 0; // Нет дополнительных данных класса wc.cbWndExtra = 0; // Нет дополнительных данных окна

// Заполнение окна белым цветом wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);

if(!RegisterClass(&wc)) return 0; // Регистрация класса окна

// Создание окна

hWnd = CreateWindow(WinName, // Имя класса окна

_T("Каркас Windows-приложения"), // Заголовок окна WS_OVERLAPPEDWINDOW, // Стиль окна CW_USEDEFAULT, // x

CW_USEDEFAULT, // y Размеры окна CW_USEDEFAULT, // Width

8 Глава 1

image

CW_USEDEFAULT, // Height

HWND_DESKTOP, // Дескриптор родительского окна NULL, // Нет меню

This, // Дескриптор приложения

NULL); // Дополнительной информации нет ShowWindow(hWnd, mode); //Показать окно

```
// Цикл обработки сообщений while(GetMessage(&msg, NULL, 0, 0))
{
TranslateMessage(&msg);// Функция трансляции кодов нажатой клавиши DispatchMessage(&msg); // Посылает сообщение функции WndProc()
}
return 0;
}
// Оконная функция вызывается операционной системой
// и получает сообщения из очереди для данного приложения LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam)
{ // Обработчик сообщений switch(message)
{
case WM_DESTROY : PostQuitMessage(0);
break;
// Завершение программы
// Обработка сообщения по умолчанию
default : return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}
```

Программа не делает ничего полезного, поэтому, запустив ее на выполнение кнопкой (Start Debugging), мы получим изображенное на рис. 1.7 пустое окно, имеющее заголовок и набор стандартных кнопок.

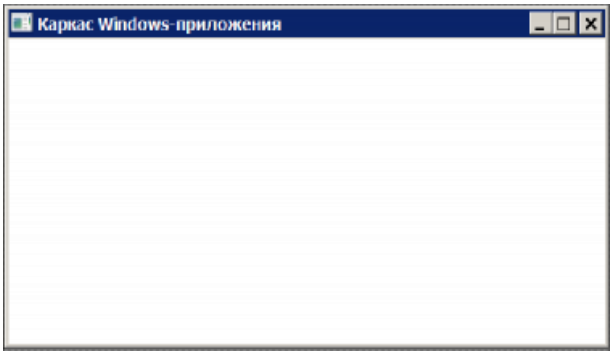

Рис. 1.7. Окно первой Windows-программы

2. Разработка кода модуля с рисованием в окне геометрических фигур

Рисование прямоугольника достаточно простая задача и можно её решать разными способами. При этом для рисования прямоугольника в с++ есть отдельные функции. Сам по себе прямоугольник может представлять собой некоторую закрашенную область или просто обозначение контура линиями.

По большому счету, при получении начальных знаний по рисованию в с++, не особо важно какой метод рисования использовать. Главное суметь понять несложный материал и сделать так, чтобы то что было нужно работало.

Сейчас я остановлюсь на рисовании двумерного прямоугольника с помощью функций rectangle и bar.

rectangle – Функция рисует прямоугольник линией текущего вида, толщины и цвета.

bar – Функция рисует двухмерный заполненный прямоугольник. Прямоугольник заполняется, с использованием текущего цвета и шаблона заполнения. При этом не рисуется контур прямоугольника;

Собственно, теория теорией, но чем больше слов, тем больше могут кипеть мозги, поэтому вспомним только, что чтобы нарисовать прямоугольник, достаточно знать всего две точки (левый верхний угол и правый нижний угол). У каждой точки по две координаты (x,y).

Теперь, чтобы увидеть разницу между этими двумя функциями достаточно написать небольшой пример.

Код С++ Рисование двумерного прямоугольника

==========

```
#include <stdlib.h>
#include <iostream.h>
#include <graphics.h> //Подключаем директиву для графического режима
    void main()
{
 system("CLS");
//Две нижние строчки – это для инициализации графики
    int gdriver = DETECT, gmode, errorcode;
  initgraph(&gdriver, &gmode, "");
```

```
        setfillstyle(1,12); //Устанавливаем цвет и стиль закраски
   rectangle(0,25,300,35); //Рисуем прямоугольник контуром
   bar(0,40,300,50); //Рисуем закрашенный прямоугольник без контур
a
 system("PAUSE");
 return;
}
===============
```

Я использовал функцию setfillstyle, которая устанавливает цвет и стиль закраски, благодаря чему видно, что при рисовании прямоугольника контуром эта функция не имеет значения, но при рисовании закрашенной области, мы получаем некоторый эффект. Чтобы увидеть, как это срабатывает, достаточно своими руками менять значения внутри setfillstyle (стиль, цвет).

Думаю этого, может быть недостаточно для хорошего освоения только что начавшему, поэтому буду копать и объяснять, немного глубже (даже если кому-то это кажется лишним).

При рисовании прямоугольника задаются две координаты. Верхний левый угол и правый нижний. Можно просить ввести эти координаты с клавиатуры, можно написать, как написал я, можно строить прямоугольник после каких-то вычислений нужных точек. Сейчас я опишу только, как хотя бы немного представлять то, что должно из , если использовать цифры, как их использовал я.

В моем примере можно увидеть надпись *...нажмите любую клавишу...,* которая светится в самом верху экрана. чтобы эта надпись не перекрывала прямоугольник, я изначально решил строить его чуть ниже. Чтобы построить прямоугольник ниже, нужно сместить его левый верхний угол. В нашем тут программировании, левый верхний угол прямоугольника является его началом, поэтому от него в первую очередь зависит общее построение.

На самом деле, понятия: (начало – лево-верх), (конец – право-низ) не совсем верно. Начало, как и конец может располагаться где угодно если использовать в параметрах функции знак минус. Просто так и проще и более принято, что началом прямоугольника зовется верхний левый угол, а окончанием правый нижний.

Чтобы вытянуть прямоугольник в длину, нужно следить за параметрами по оси x.

Чтобы вытянуть высоту прямоугольника, нужно следить за параметрами по оси y

rectangle(x1,y1,x2,y2) – общий принцип построения прямоугольника
rectangle(x1,y1,x1+300,y1+10) – то, как строил прямоугольник я.

Несложно увидеть и понять, что 300>10, при этом большая часть относится к оси x, так как часть по оси x больше, то прямоугольник будет вытянут в длину, если бы часть по оси y была бы больше, то прямоугольник бы был вытянут в высоту.

Разработка кода программного модуля с обработкой сообщений мыши. Отслеживание курсора мышки

Частенько приложению требуется знать координаты курсора мышки. Обычно, это графические программы, которые отслеживают координаты курсора во время рисования какого-нибудь рисунка. Так же отслеживать положение мышки необходимо приложениям, работающим с текстом для возможности выделения блоков текста.

Для того, чтобы отслеживать курсор мышки, обычно необходимо обработать три сообщения WM_LBUTTONDOWN, WM_MOUSEMOVE, и WM_LBUTTONUP. Как правио, отслеживание курсора начинается с постуления сообщения WM_LBUTTONDOWN, в параметре *lParam* которого записаны координаты курсора. Далее начинается сам процесс отслеживания путём обработки потока сообщений WM_MOUSEMOVE, которые выводит на само окно при перемещении мышки. Поступление сообщения WM_LBUTTONUP сигнализирует об окончании процесса отслеживания.

Так же можно использовать функцию TrackMouseEvent, чтобы заставить систему посылать другие сообщения необходимые для отслеживания курсора. Сообщение WM_MOUSEHOVER посылается системой, когда мышка попадает в клиентскую область, а сообщение WM_MOUSELEAVE - когда курсор покидает клиентскую область. Соответственно, сообщения WM_NCMOUSEHOVER и WM_NCMOUSELEAVE отвечают за не клиентскую область.

Рисование линий при помощи мышки

В данном разделе статьи представлена часть кода оконной процедуры, которая позволяет пользователю рисовать линии в клиентской области окна путём перетаскивания мышки.

Когда оконная процедура получает сообщение WM_LBUTTON DOWN, то происходит захват мышки и сохранение координат курсора, используяих как начальную точку линии. При этом использ уется функция Clip Cursor, чтобы ограничить перемещение курсора клиентской областью в процессе рисования.

В течение первого сообщения WM_MOUSEMOVE оконная процедура рисует линию от начальной точки до текущей позиции курсора. В течение последующих сообщений WM_MOUSEMOVE, оконная процедура стирает предыдущую линию путём рисования повёрх неё линии инверсного цвета. Затем снова рисуется линия от начальной точки до текущих координат курсора.

Поступление сообщения WM_LBUTTONUP сигнализирует об окончании рисования. Оконная процедура освобождает захват мышки и снимает ограничение движения мышки клиентской областью.

Пример:

```
LRESULT APIENTRY MainWndProc(HWND hwndMain, UINT uMsg,
                             WPARAM wParam, LPARAM lParam)
{
HDC hdc;                    // дескриптор контекста устройства
RECT rcClient;              // прямоугольник клиентской области
POINT ptClientUL;                    // верхний левый угол клиент.области
POINT ptClientLR;                    // нижний правый угол клиент.области
static POINTS ptsBegin;      // начальная точка
static POINTS ptsEnd;        // новая конечная точка
static POINTS ptsPrevEnd;    // предыдущая конечная точка
static BOOL fPrevLine = FALSE; // флаг предыдущей линии
switch (uMsg)
{
case WM_LBUTTONDOWN:
    // Захватываем мышку.
```

```
SetCapture(hwndMain);
    // Получаем экранные координаты клиентской области,
    // и преобразуем их в клиентские координаты.
GetClientRect(hwndMain, &rcClient);
ptClientUL.x = rcClient.left;
ptClientUL.y = rcClient.top;
    // Добавляем один пиксель справа и снизу, так как координаты,
    // полученные из GetClientRect не включают левого и
    // нижнего пикселей.
ptClientLR.x = rcClient.right + 1;
ptClientLR.y = rcClient.bottom + 1;
ClientToScreen(hwndMain, &ptClientUL);
ClientToScreen(hwndMain, &ptClientLR);
    // Копируем клиентские координаты клиентской области
    // в структуру rcClient. Ограничиваем курсор мышки клиентской
    // областью, передав структуру rcClient в
    // функцию ClipCursor.
SetRect(&rcClient, ptClientUL.x, ptClientUL.y,
   ptClientLR.x, ptClientLR.y);
ClipCursor(&rcClient);
    // Преобразуем координаты курсора для структуры POINTS,
    // которая определяет начальную точку рисования линии
    // в течение сообщения WM_MOUSEMOVE.
ptsBegin = MAKEPOINTS(lParam);
return 0;
case WM_MOUSEMOVE:
    // Чтобы рисовалась линия, то при движении мышки
    // пользователь должен удерживать нажатой левую кнопку мышки.
if (wParam & MK_LBUTTON)
{
    // Получаем контекст устройства (DC) для клиентской области
  hdc = GetDC(hwndMain);
    // Следующая функция гарантирует, что пиксели
    // предыдущей линии установлены в белый цвет, а
    // вновь нарисованной линии - в чёрный.
 SetROP2(hdc, R2_NOTXORPEN);
```

```
        // Если линия была нарисована в предыдущем WM_MOUSEMOVE,
        // то рисуем поверх неё. Тем самым, установив пиксели
        // линии в белый цвет, мы сотрём её.
      if (fPrevLine)
     {
     MoveToEx(hdc, ptsBegin.x, ptsBegin.y, (LPPOINT) NULL);
     LineTo(hdc, ptsPrevEnd.x, ptsPrevEnd.y);
     }
        // Преобразуем текущие координаты курсора в структуру
        // POINTS, а затем рисуем новую линию.
     ptsEnd = MAKEPOINTS(lParam);
     MoveToEx(hdc, ptsBegin.x, ptsBegin.y, (LPPOINT) NULL);
     LineTo(hdc, ptsEnd.x, ptsEnd.y);
        // Устанавливаем флаг предыдущей линии, сохраняем конечную
        // точку новой линии, а затем освобождаем DC.
     fPrevLine = TRUE;
     ptsPrevEnd = ptsEnd;
     ReleaseDC(hwndMain, hdc);
     }
     break;
     case WM_LBUTTONUP:
        // Пользователь закончил рисовать линию. Сбрасываем флаг
        // предыдущей линии, освобождаем курсор мышки и
              // освобождаем захват мышки.
              fPrevLine = FALSE;
              ClipCursor(NULL);
              ReleaseCapture();
              return 0;
           case WM_DESTROY:
              PostQuitMessage(0);
              break;
           // Обрабатываем другие сообщения.
```

Обработка двойного щелчка

Чтобы получать сообщения о двойном щелчке (double-click messages), класс окна должен содержать стиль CS_DBLCLKS. Этот

стиль устанавливается при регистрации оконного класса, как показано ниже.

Пример:

```
BOOL InitApplication(HINSTANCE hInstance)
{
WNDCLASS wc;
   wc.style = CS_DBLCLKS | CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc = (WNDPROC) MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
   wc.hCursor = LoadCursor(NULL, IDC_IBEAM);
   wc.hbrBackground = GetStockObject(WHITE_BRUSH);
   wc.lpszMenuName = "MainMenu";
   wc.lpszClassName = "MainWClass";
   return RegisterClass(&wc);
}
```

Сообщение о двойном щелчке всегда предшествует сообщению о нажатии кнопки.

Выделение строки текста

В данном разделе приведён пример, который был взят из обычного текстового редактора. Он включает код, позволяющий пользователю обычным щелчком устанавливать каретку в любом месте текста, а также выделять (подсвечивать) строку текста двойным щелчком.

Пример:

```
LRESULT APIENTRY MainWndProc(HWND hwndMain, UINT uMsg,
                             WPARAM wParam, LPARAM lParam)
{
   HDC hdc;                  // дескриптор контекста устройства
   TEXTMETRIC tm;            // данные о размере шрифта
   int i, j;                 // счётчики цикла
   int cCR = 0;              // счётчик возвратов каретки
```

```
        char ch;                    // символ из буфера ввода
        static int nBegLine;        // начало выделенной линии
        static int nCurrentLine = 0; // текущая выделенная строка
        static int nLastLine = 0;   // последняя строка текста
        static int nCaretPosX = 0;  // x-координата каретки
        static int cch = 0;         // количество введённых символов
        static int nCharWidth = 0;  // точная ширина символа
        static char szHilite[128];      // строка текста, которая будет
выделена
        static DWORD dwCharX;       // средняя ширина символов
        static DWORD dwLineHeight;  // высота строки
        static POINTS ptsCursor;    // координаты курсора мышки
        static COLORREF crPrevText; // предыдущий цвет текста
        static COLORREF crPrevBk;   // предыдущий цвет фона
        static PTCHAR pchInputBuf;  // указатель на буфер ввода
        static BOOL fTextSelected = FALSE; // флаг выделения текста
                                           size_t * pcch;
                                           HRESULT hResult;
        switch (uMsg)
        {
          case WM_CREATE:
            // Получаем параметры текущего шрифта.
            hdc = GetDC(hwndMain);
            GetTextMetrics(hdc, &tm);
            ReleaseDC(hwndMain, hdc);
            // Сохраняем среднюю ширину и высоту символа.
            dwCharX = tm.tmAveCharWidth;
            dwLineHeight = tm.tmHeight;
            // Выделяем буфер для хранения ввода с клавиатуры.
            pchInputBuf = (LPSTR) GlobalAlloc(GPTR,
              BUFSIZE * sizeof(TCHAR));
            return 0;
          case WM_CHAR:
            switch (wParam)
            {
              case 0x08:  // backspace
              case 0x0A:  // перевод строки
              case 0x1B:  // escape
                MessageBeep( (UINT) -1);
                return 0;
```

```
                case 0x09:  // символ табуляции (tab)
                  // Преобразуем символы табуляции в четыре
пробела.
                  for (i = 0; i < 4; i++)
                    SendMessage(hwndMain, WM_CHAR, 0x20, 0);
                  return 0;
                case 0x0D:  // возврат каретки
                 // Записываем символ возврата каретки и помещаем
каретку
                 // в начало новой строки.
                  pchInputBuf[cch++] = 0x0D;
                  nCaretPosX = 0;
                  nCurrentLine += 1;
                  break;
                default:    // отображаемый символ
                  ch = (char) wParam;
                  HideCaret(hwndMain);
                  // Получаем ширину символа и отображаем его.
                  hdc = GetDC(hwndMain);
                  GetCharWidth32(hdc, (UINT) wParam, (UINT)
wParam,
                    &nCharWidth);
                  TextOut(hdc, nCaretPosX,
                    nCurrentLine * dwLineHeight, &ch, 1);
                  ReleaseDC(hwndMain, hdc);
                  // Сохраняем символ в буфере.
                  pchInputBuf[cch++] = ch;

                  // Вычисляем новую горизонтальную координат
каретки.
                  // Если координата достигла максимума, то
вставляем
                  // перевод каретки и перемещаем каретку
                  // в начало следующей строки.
                  nCaretPosX += nCharWidth;
                  if ((DWORD) nCaretPosX > dwMaxCharX)
                  {
                    nCaretPosX = 0;
                    pchInputBuf[cch++] = 0x0D;
                    ++nCurrentLine;
```

```
                }
                ShowCaret(hwndMain);
                break;
            }
            SetCaretPos(nCaretPosX, nCurrentLine * dwLineHeight);
            nLastLine = max(nLastLine, nCurrentLine);
            break;
        // Обрабатываем другие сообщения.
        case WM_LBUTTONDOWN:
            // Если строка текста уже выделена, то перерисовываем
            // текст, чтобы убрать выделение.
            if (fTextSelected)
            {
              hdc = GetDC(hwndMain);
              SetTextColor(hdc, crPrevText);
              SetBkColor(hdc, crPrevBk);
                                                  hResult =
StringCchLength(szHilite, 128/sizeof(TCHAR), pcch);
                                                  if (FAILED(hResult))
                                                  {
                                                  // TODO: обработчик
ошибки
                                                  }
              TextOut(hdc, 0, nCurrentLine * dwLineHeight, szHilite,
*pcch);
              ReleaseDC(hwndMain, hdc);
              ShowCaret(hwndMain);
              fTextSelected = FALSE;
            }
            // Сохраняем текущие координаты курсора мышки.
            ptsCursor = MAKEPOINTS(lParam);
            // Определяем, на какой строке находится курсор, и
сохраняем
            // номер строки. Следим, чтобы номера строк не были
больше
            // номера последней строки текста. Результат используем
            // для установки y-координаты каретки.
            nCurrentLine = min((int)(ptsCursor.y / dwLineHeight),
                nLastLine);
```

```
            // Парсим текст буфера ввода, чтобы найти первый
символ
            // в выделенной строке текста. Каждая строка
оканчивается
            // символом возврата каретки, поэтому, чтобы найти
            // выделенную строку, достаточно сосчитать возвраты
каретки.
            cCR = 0;
            nBegLine = 0;
            if (nCurrentLine != 0)
            {
               for (i = 0; (i < cch) &&
                     (cCR < nCurrentLine); i++)
               {
                  if (pchInputBuf[i] == 0x0D)
                     ++cCR;
               }
               nBegLine = i;
            }
            // Начиная с начала выделенной строки, измеряем
ширину
            // каждого символа, суммируя с шириной уже
измеренного
            // символа. Останавливаемся,
            // когда сумма больше, чем x-координата курсора.
            // Сумма используется для установки x-координаты
каретки.

            hdc = GetDC(hwndMain);
            nCaretPosX = 0;
            for (i = nBegLine;
               (pchInputBuf[i] != 0x0D) && (i < cch); i++)
            {
               ch = pchInputBuf[i];
               GetCharWidth32(hdc, (int) ch, (int) ch, &nCharWidth);
               if ((nCaretPosX + nCharWidth) > ptsCursor.x) break;
               else nCaretPosX += nCharWidth;
            }
            ReleaseDC(hwndMain, hdc);
```

```
            // Устанавливаем каретку в то место, куда кликнул
пользователь.
            SetCaretPos(nCaretPosX, nCurrentLine * dwLineHeight);
            break;
         case WM_LBUTTONDBLCLK:
            // Копируем выделенную строку в буфер.
            for (i = nBegLine, j = 0; (pchInputBuf[i] != 0x0D) &&
                  (i < cch); i++)
            {
               szHilite[j++] = pchInputBuf[i];
            }
            szHilite[j] = '\0';
            // Скрываем каретку, инвертируем цвет фона и символов,
            // а затем перерисовываем выделенную строку.
            HideCaret(hwndMain);
            hdc = GetDC(hwndMain);
            crPrevText = SetTextColor(hdc, RGB(255, 255, 255));
            crPrevBk = SetBkColor(hdc, RGB(0, 0, 0));
                                                hResult
StringCchLength(szHilite, 128/sizeof(TCHAR), pcch);
                                                if (FAILED(hResult))
                                                {
                                                // TODO: обработчик
ошибки
                                                }
            TextOut(hdc, 0, nCurrentLine * dwLineHeight, szHilite,
*pcch);
            SetTextColor(hdc, crPrevText);
            SetBkColor(hdc, crPrevBk);
            ReleaseDC(hwndMain, hdc);
            fTextSelected = TRUE;
            break;
            // Обрабатываем другие сообщения.
         default:
            return DefWindowProc(hwndMain, uMsg, wParam,
lParam);
      }
      return NULL;
   }
```

Разработка кода программного модуля с использованием пути

Обмен информацией с текстовыми файлами обычно реализуется через *текстовые потоки*, а с двоичными файлами – через *двоичные потоки*.

Обмен информацией с файлом предполагает выполнение следующих действий:

- создание потока;
- открытие файла (связь файла с потоком);
- обмен (ввод–вывод) информацией с файлом;
- закрытие файла (разрыв файла с потоком).

Для создания потока необходимо в программе создать объект соответствующего класса – для *потока ввода* необходимо объявить объект типа ifstream, для *потока вывода* – объект типа ofstream. Потоки, которые реализуют одновременно ввод и вывод, должны быть объявлены как объекты типа fstream:

```
ifstream fin; // входной поток (объект) fin
ofstream fout; // выходной поток (объект) fout
fstream fio; // поток ввода–вывода (объект) fio
```

Для открытия файла его нужно связать с потоком. Это можно сделать с помощью функции open():

```
ofstream fout; // создание потока (объекта) fout
fout.open("vix.dat", ios::out); // открытие файла vix.dat
```

В большинстве же случаев для открытия файла используют конструкторы классов, которые *автоматически открывают заданный файл*:

```
ifstream fin("vx.dat", ios::in); // открытие файла vx.dat для ввода
ofstream fout("vix.dat", ios::out); // открытие файла vix.dat для вывода
fstream fio("vvx.dat", ios::in|ios::out); // открытие vvx.dat для в-ы
```

Конструкторы создают объекты соответствующего класса, открывают файл с указанным именем и связывают файл с потоком. Первый аргумент у конструкторов – *имя файла*, и это единственно обязательный аргумент. Второй аргумент задаёт *режим доступа* к файлу.

Режим доступа к файлу служит для описания характера использования файла – для чтения, для записи и т. д. В классе ios определены константы для указания режима доступа к файлу:

ios::in открыть файл только для чтения;
ios::out открыть файл только для записи;
ios::app открыть файл для добавления в конец файла;
ios::trunc если файл существует, удалить его;
ios::binary открыть файл в двоичном режиме.
Режимы доступа можно объединять с помощью операции ИЛИ.

По умолчанию объекты класса ofstream открыты для вывода, а класса ifstream – для ввода, поэтому режимы out и in можно опускать. *По умолчанию* все файлы открываются в *текстовом режиме*.

Любой файл в C++, независимо от того, что в нём содержится – отформатированный текст или неформатированные данные – может быть открыт как в текстовом, так и в двоичном режиме. Но всё же, если необходимо сохранить данные в двоичном виде, лучше использовать двоичные файлы. В C++ это делается путём использования константы ios::binary для указания режима доступа к файлу.

Прежде чем начать обмен данными с файлом, следует убедиться, была ли *операция открытия файла успешной,* так как можно допустить ошибку в имени файла или в указании пути к файлу. Например, проверить правильность открытия файла vx.dat можно следующим образом:

```
if(!fin){cout<<"File no open"; exit(1);}
```

Любой аргумент функции exit(), отличный от нуля, показывает, что программа прекратила выполнение из-за ошибки.

Только после того, как поток (объект) успешно соединён с файлом (*т.е. файл открыт*), можно выполнять *обмен информацией с файлом*.

Разработка кода программного модуля с выводом текста

Все программы на C++, созданные вами в уроках 1 и 2, использовали выходной поток cout для вывода сообщений на экран. В этом уроке вы будете использовать cout для вывода символов,

целых чисел, например 1001, и чисел с плавающей точкой, например 0.12345. К концу данного урока вы освоите следующие основные концепции:

Для вывода символов и чисел на экран вы можете использовать выходной поток cout.

В C++ можно использовать с cout специальные символы для вывода табуляции или новой строки и даже для воспроизведения звука на вашем компьютере.

В C++ можно легко отображать числа в десятичном, восьмеричном (по основанию 8) или шестнадцатеричном (по основанию 16) формате.

Используя в командной строке операционной системы, операторы переназначения, вы можете перенаправить выходные сообщения своей программы, посылаемые в cout, с экрана в файл или на принтер.

Используя выходной поток cerr, ваши программы могут посылать сообщения на стандартное устройство ошибок, избавляя пользователей от необходимости переназначения сообщений.

Вы можете форматировать вывод вашей программы, используя модификатор setw внутри выходного потока.

Почти все создаваемые вами программы на C++ используют cout для вывода сообщений на экран. Из этого урока вы узнаете, как лучше использовать com/.

Использование cout для вывода чисел

До сих пор созданные вами программы использовали com/ для вывода символьных строк (букв и чисел, взятых в кавычки). Теперь вы узнаете, что com/ можно также использовать для вывода чисел. Следующая программа 1001.CPP выводит число 1001 на ваш экран:

```
#include <iostream.h>
void main(void)
{
  cout << 1001;
}
```

Откомпилируйте и запустите эту программу. На вашем экране будет отображено число 1001, как показано ниже:

```
C:\> 1001 <ENTER>
```

1001

Далее отредактируйте программу и измените оператор cout , чтобы вывести число 2002, как показано ниже:

```
cout << 2002;
```

Кроме отображения целых чисел (чисел без десятичной точки), com/также позволяет вашим программам отображать числа с плавающей точкой, например 1.2345. Следующая программа FLOATING.CPP использует com/для вывода числа 0.12345 на экран:

```
#include <iostream.h>
void main(void)
{
  cout << 0.12345;
}
```

Как и ранее, откомпилируйте и запустите эту программу. На вашем экране появится следующий вывод:

```
C:\> FLOATING <ENTER>
0.12345
```

Вывод нескольких значений одновременно

Как вы уже знаете, двойной знак "меньше" является операцией вставки (эта операция вставляет символы в выходной поток для отображения). С помощью cout вы можете использовать несколько операций вставки в пределах одного оператора. Например, следующая программа 1001TOO. CPP использует эту операцию четыре раза для отображения числа 1001 на вашем экране:

```
#include <iostream.h>
void main(void)
(
  cout << 1 << 0 << 0 << 1;
}
```

Когда вы откомпилируете и запустите эту программу, на вашем экране появится следующее:

```
C:\> 1001TOO <ENTER>
1001
```

Каждый раз, когда в C++ встречается операция вставки, число или символы просто добавляются к тем, что находятся в настоящее

время в выходном потоке. Следующая программа SHOW1001.CPP с помощью cout выводит символьную строку и число:

```
#include <iostream.h>
void main(void)
{
  cout << "Мое любимое число равно " << 1001;
}
```

Обратите внимание, что пробел, следующий за словом равно (внутри кавычек), служит для отделения числа 1001 от этого слова. Без пробела число сливается со следующим словом (равно 1001). Подобным образом следующая программа 1001MID.CPP отображает число 1001 в середине символьной строки:

```
#include <iostream.h>
void main(void)
{
  cout << "Число " << 1001 << " мне очень нравится";
}
```

Как и ранее, обратите внимание на расстановку пробелов до и после числа 1001.

Наконец, следующая программа MIXMATCH.CPP комбинирует строки, символы, целые числа и числа с плавающей точкой внутри одного и того же выходного потока:

```
#include <iostream.h>
void main(void)
{
  cout << "В " << 20 << " лет мой оклад был " << 493.34 << endl;
}
```

Когда вы откомпилируете и запустите эту программу, на вашем экране появится следующий вывод:

```
C:\> MIXMATCH <ENTER>
В 20 лет мой оклад был 493.34
```

Тема 2. Технология работы с файлами при создании модуля

Разработка кода программного модуля по организации диалога выбора файла. Выполнение отладки и тестирования программы на уровне модуля. Организация скроллинга.

Активный курсор в Scroll Bar.

Компилятор: Visual C++ 6

Данный пример позволяет динамически менять курсор на scrollbar-е окна. Переменные m_nVIDResource и m_nHIDResource используются для предотвращения множественной загрузки курсора.

CMyWnd наследован от CWnd.

```
UINT m_nVIDResource;          //Текущий ID ресурса курсора
                              //(вертикальный scroll bar)
UINT m_nHIDResource;          //Текущий ID ресурса курсора
                              //(горизонтальный scroll bar)
CMyWnd::CMyWnd()
{
 m_nVIDResource = 0;
 m_nHIDResource = 0;
}
BOOL CMyWnd::OnSetCursor(CWnd* pWnd, UINT nHitTest,
UINT message)
{
 if(nHitTest == HTVSCROLL)
 {
  //Меняем курсор на горизонтальном скролбаре
  SetCursor(AfxGetApp()->LoadCursor(IDC_VSCROLLBAR));
  if(m_nVIDResource != IDC_VSCROLLBAR)
    m_nVIDResource = IDC_VSCROLLBAR;
  return true;
 }
 else if(nHitTest == HTHSCROLL)
 {
  //Меняем курсор на вертикальном скролбаре
  SetCursor(AfxGetApp()->LoadCursor(IDC_HSCROLLBAR));
  if(m_nHIDResource != IDC_HSCROLLBAR)
    m_nHIDResource = IDC_HSCROLLBAR;
  return true;
 }
 else
{
  if(m_nVIDResource != 0) m_nVIDResource = 0;
  if(m_nHIDResource != 0) m_nHIDResource = 0;
}
  return BaseClass::OnSetCursor(pWnd, nHitTest, message);
}
```

```
void CMyWnd::OnVScroll(UINT nSBCode, UINT nPos,
                       CScrollBar* pScrollBar)
{
  switch (nSBCode) {
  case SB_THUMBTRACK:        //Перемещаем ползунок в
                             //определённое положение
    if(m_nVIDResource != IDC_VTHUMBTRACK)
    {
      SetCursor(AfxGetApp()-
>LoadCursor(IDC_VTHUMBTRACK));
      m_nVIDResource = IDC_VTHUMBTRACK;
    }
    break;
  case SB_LINEDOWN:          //Скроллинг на одну линию вниз
    if(GetScrollPos(SB_VERT) == GetScrollLimit(SB_VERT))
    {
      if(m_nVIDResource != IDC_VLINEDOWN_DISABLE)
      {
        SetCursor(AfxGetApp()-
>LoadCursor(IDC_VLINEDOWN_DISABLE));
        m_nVIDResource = IDC_VLINEDOWN_DISABLE;
      }
    }
    else if(m_nVIDResource != IDC_VLINEDOWN)
    {
      SetCursor(AfxGetApp()->LoadCursor(IDC_VLINEDOWN));
      m_nVIDResource = IDC_VLINEDOWN;
    }
    break;
  case SB_LINEUP:            //Скроллинг на одну линию вверх
    SCROLLINFO  info;
    info.cbSize = sizeof(SCROLLINFO);
    GetScrollInfo(SB_VERT, >info);
    if(info.nPos == info.nMin)
    {
      if(m_nVIDResource != IDC_VLINEUP_DISABLE)
      {
        SetCursor(AfxGetApp()-
>LoadCursor(IDC_VLINEUP_DISABLE));
        m_nVIDResource = IDC_VLINEUP_DISABLE;
```

```
        }
      }
      else if(m_nVIDResource != IDC_VLINEUP)
      {
        SetCursor(AfxGetApp()->LoadCursor(IDC_VLINEUP));
        m_nVIDResource = IDC_VLINEUP;
      }
      break;
    case SB_PAGEDOWN:          //Скроллинг на одну страницу
вниз
      if(m_nVIDResource != IDC_VPAGEDOWN)
      {
        SetCursor(AfxGetApp()->LoadCursor(IDC_VPAGEDOWN));
        m_nVIDResource = IDC_VPAGEDOWN;
      }
      break;
    case SB_PAGEUP:          //Скроллинг на одну страницу вверх
      if(m_nVIDResource != IDC_VPAGEUP)
      {
        SetCursor(AfxGetApp()->LoadCursor(IDC_VPAGEUP));
        m_nVIDResource = IDC_VPAGEUP;
      }
      break;
    }
    BaseClass::OnVScroll(nSBCode, nPos, pScrollBar);
  }
  void CMyWnd::OnHScroll(UINT nSBCode, UINT nPos,
                CScrollBar* pScrollBar)
  {
    switch (nSBCode) {
    case SB_THUMBTRACK:      //Перемещаем ползунок в
                    //определённое положение
      if(m_nHIDResource != IDC_HTHUMBTRACK)
      {
        SetCursor(AfxGetApp()-
>LoadCursor(IDC_HTHUMBTRACK));
        m_nHIDResource = IDC_HTHUMBTRACK;
      }
      break;
    case SB_LINEDOWN:        //Скроллинг на одну линию вниз
```

```
        if(GetScrollPos(SB_HORZ) == GetScrollLimit(SB_HORZ))
        {
          if(m_nHIDResource != IDC_HLINEDOWN_DISABLE)
          {
            SetCursor(AfxGetApp()-
>LoadCursor(IDC_HLINEDOWN_DISABLE));
            m_nHIDResource = IDC_HLINEDOWN_DISABLE;
          }
        }
        else if(m_nHIDResource != IDC_HLINEDOWN)
        {
          SetCursor(AfxGetApp()->LoadCursor(IDC_HLINEDOWN));
          m_nHIDResource = IDC_HLINEDOWN;
        }
        break;
      case SB_LINEUP:           //Скроллинг на одну линию вверх
        SCROLLINFO  info;
        info.cbSize = sizeof(SCROLLINFO);
        GetScrollInfo(SB_HORZ, &info);
        if(info.nPos == info.nMin)
        {
          if(m_nHIDResource != IDC_HLINEUP_DISABLE)
          {
            SetCursor(AfxGetApp()-
>LoadCursor(IDC_HLINEUP_DISABLE));
            m_nHIDResource = IDC_HLINEUP_DISABLE;
          }
        }
        else if(m_nHIDResource != IDC_HLINEUP)
        {
          SetCursor(AfxGetApp()->LoadCursor(IDC_HLINEUP));
          m_nHIDResource = IDC_HLINEUP;
        }
        break;
      case SB_PAGEDOWN:           //Скроллинг на одну страницу
вниз
        if(m_nHIDResource != IDC_HPAGEDOWN)
        {
          SetCursor(AfxGetApp()->LoadCursor(IDC_HPAGEDOWN));
          m_nHIDResource = IDC_HPAGEDOWN;
```

```
    }
    break;
  case SB_PAGEUP:          //Скроллинг на одну страницу вверх
    if(m_nHIDResource != IDC_HPAGEUP)
    {
      SetCursor(AfxGetApp()->LoadCursor(IDC_HPAGEUP));
      m_nHIDResource = IDC_HPAGEUP;
    }
    break;
  }
  BaseClass::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

Выполнение отладки и тестирования программы на уровне модуля. Чтение и запись файлов в библиотеке Win32 API. Для этого нам понадобятся следующие процедуры:

```
CreateFile(szName, dwAccess, dwShareMode, lpSecurityAttributes,
        dwCreationDisposition, dwFlags, hTemplateFile);
```

В Windows для того, чтобы открыть или создать файл, нужно вызвать процедуру, имеющую целых семь аргументов. К счастью, большинство из них приходится использовать крайне редко. Аргумент szName задает имя файла, а dwAccess — желаемый доступ к файлу, обычно это GENERIC_READ, GENERIC_WRITE или оба значения, объединенные логическим или. Параметр dwShareMode определяет, что могут делать с файлом другие процессы, пока мы с ним работаем. Возможные значения — FILE_SHARE_READ, FILE_SHARE_WRITE, FILE_SHARE_DELETE и их комбинации, однако часто этот параметр просто устанавливают в ноль. Параметр dwCreationDisposition определяет, как именно мы хотим открыть файл, может быть, например, CREATE_NEW, CREATE_ALWAYS, OPEN_EXISTING, OPEN_ALWAYS. О семантике этого хозяйства нетрудно догадаться самостоятельно. С помощью dwFlags можно указать дополнительные свойства файла, например, хранить ли его в зашифрованном или сжатом виде, или сказать, что файл является скрытым, временным или системным. Обычно сюда передают FILE_ATTRIBUTE_NORMAL. Наконец, про lpSecurityAttributes и hTemplateFile сейчас знать не нужно, сюда можно смело передавать NULL.

В случае успешного создания или открытия файла, процедура CreateFile возвращает его хэндл. В случае ошибки возвращается

специальное значение INVALID_HANDLE_VALUE. Узнать подробности об ошибке можно с помощью GetLastError.

ReadFile(hFile, lpBuff, dwBuffSize, &dwCount, NULL);

Чтение из файла в буфер lpBuff размером dwBuffSize. В переменную dwCount записывается реальное количество прочитанных байт. Последний опциональный аргумент называется lpOverlapped и о нем сейчас знать не нужно.

WriteFile(hFile, lpBuff, dwBuffSize, &dwCount, NULL);

Аргументы и семантика процедуры WriteFile полностью аналогичны ReadFile.

CloseHandle(hFile);

Файловые дескрипторы закрываются с помощью CloseHandle. На самом деле, эта процедура используется не только для работы с файлами, так что мы еще не единожды с нею встретимся.

Посмотрим теперь на все это хозяйство в действии. Следующая программа пишет в файл counter.dat количество собственных запусков. Первые пять запусков ничем не примечательны. На шестой и последующие запуски программа сообщает, что у нее истек триал и просит приобрести полную версию.

```
#include <windows.h>
#define MAX_TRIAL_RUNS 5
const TCHAR szCounterFileName[] = L"counter.dat";
const TCHAR szMsgTmpl[] = L"Вы запустили программу в %d-й раз. %s.";
const TCHAR szCheckOk[] = L"Все в порядке, продолжайте работу";
const TCHAR szCheckFailed[] = L"Триал истек, купите полную версию";
DWORD ReadCounter() {
  DWORD dwCounter, dwTemp;
  HANDLE hFile = CreateFile(szCounterFileName, GENERIC_READ, 0, NULL,
                            OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  if(INVALID_HANDLE_VALUE == hFile) {
    return 1;
  }
  ReadFile(hFile, &dwCounter, sizeof(dwCounter), &dwTemp, NULL);
  if(sizeof(dwCounter) != dwTemp) {
```

```
    CloseHandle(hFile);
    return 1;
   }
   CloseHandle(hFile);
   return dwCounter;
  }
  VOID WriteCounter(DWORD dwCounter) {
   DWORD dwTemp;
   HANDLE hFile = CreateFile(szCounterFileName,
GENERIC_WRITE, 0, NULL,
                    CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
   if(INVALID_HANDLE_VALUE == hFile) {
    return;
   }
   WriteFile(hFile, &dwCounter, sizeof(dwCounter), &dwTemp,
NULL);
   CloseHandle(hFile);
  }
  int main() {
   TCHAR szMsg[256];
   DWORD dwCounter = ReadCounter();
   LPCWSTR lpCheckResult = dwCounter > MAX_TRIAL_RUNS ?
                szCheckFailed : szCheckOk;
   wsprintf(szMsg, szMsgTmpl, dwCounter, lpCheckResult);
   MessageBox(0, szMsg, L"Сообщение", 0);
   if(dwCounter <= MAX_TRIAL_RUNS) {
    WriteCounter(dwCounter+1);
   }
   ExitProcess(0);
  }
```

Как обычно, программа также успешно компилируется при помощи MinGW и запускается под Wine.

В качестве домашнего задания можете попробовать модифицировать программу так, чтобы она выводила время, когда производились все ее запуски. Для этого вам понадобятся процедуры GetLocalTime, SetFilePointer и GetFileSizeEx. Если это задание покажется вам слишком простым, попробуйте найти информацию о том, как при помощи процедур, упомянутых в этой

заметке, (1) написать консольное приложение и (2) открыть диск C: на чтение, словно он является обычным файлом.

Тема 3. Технология разработки модуля с использованием окон и элементов управления.

Выполнение отладки и тестирования программы на уровне модуля. Основные этапы разработки программного обеспечения. Работы с окнами в приложении. Отладка модуля.

Условие задачи

1. Разрабатывоние диалогового окна.

Которое содержит две кнопки OK и Cancel (рис.1).

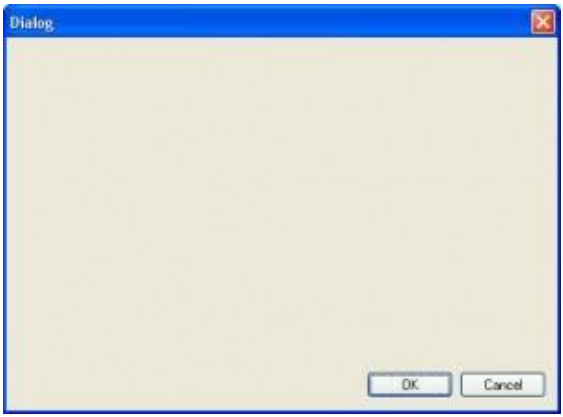

Рис. 1. Форма, которую нужно создать

2. Осуществить вызов диалогового окна из основного окна приложения. Вывести соответствующее сообщение, если в диалоговом окне нажатая одна из двух клавиш: OK и Cancel.

Выполнение

1. Запустить MS Visual Studio 2010
2. Создать проект на C++ по шаблону Dialog Based Application

В окне создания нового проекта задать имя приложения как MFCApp. В будущем будем привязываться к этому имени.

Папка приложения в нашем случае (не важно):

E:\Programs\CPP\Train-04

Во время создания нового проекта в мастере выбираем Dialog Based Application. Все другие настройки оставляем по умолчанию.

В результате будут сгенерированы два класса с именами CMFC AppApp и CMFCAppDlg.

Окно MS Visual Studio после выполненных действий будет иметь вид как показано на рисунке 2.

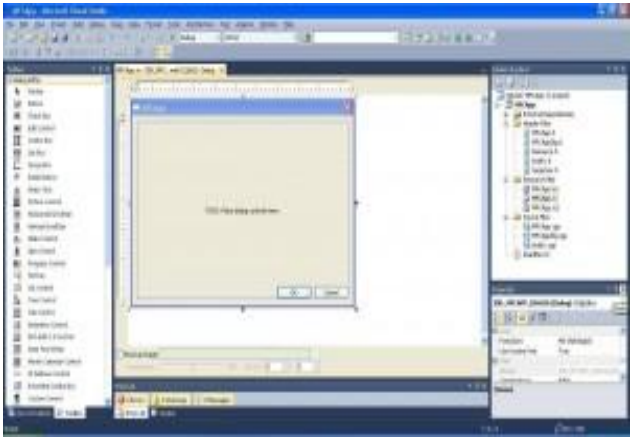

Рис. 2. Форма приложения после создания Dialog Based Application

2. Изменение внешнего вида главной формы

Осуществим удаление кнопки OK. А кнопку Cancel переименуем в Exit.

Для удаления кнопки OK нужно сначала ее выделить, а потом нажать клавишу Delete.

Таким самым образом удаляем элемент в котором написано:

"TODO: Place dialog controls here."

Чтобы переименовать кнопку Cancel, сначала выделим ее, а потом в окне Properties в свойстве Captionнабираем слово «Exit» (рис. 3). Таким образом, приложение уже настроено на нажатие на кнопку «Exit» таким образом, что происходит выход из приложения.

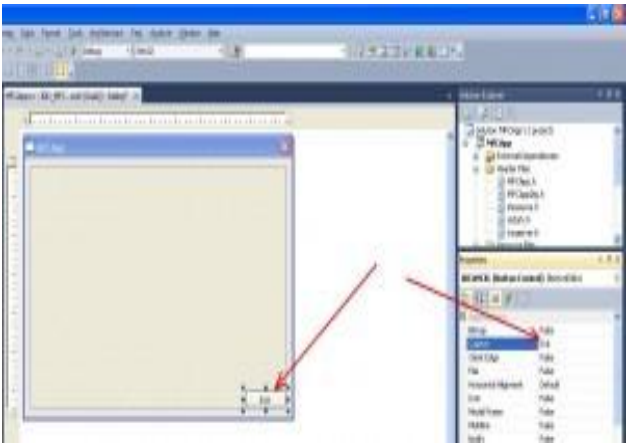

Рис. 3. Переименование имени кнопки с «Cancel» на «Exit»

Пока что, в окне панели Properties имеем два элемента:

- IDD_MFCAPP_DIALOG – главное окно приложения;
- IDCANCEL – кнопка «Exit» (бывшая «Cancel»).

Можно загрузить приложение на выполнение и протестовать его работу.

Выносим на форму другую кнопку, не привязанную к обработчикам событий. В окне панели Properties появляется еще один элемент IDC_BUTTON1. Форма приложения будет иметь вид, как показано на рисунке 4.

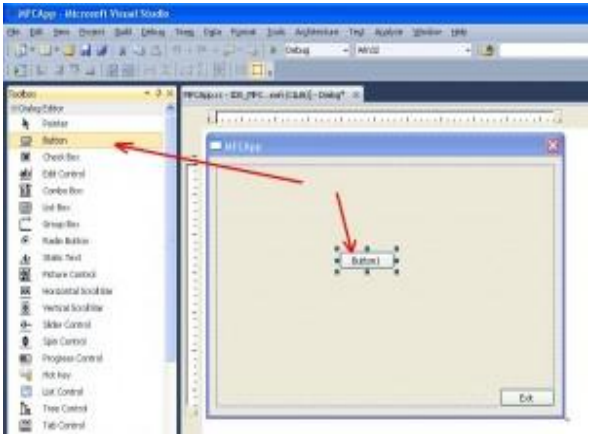

Рис. 4. Вынесение кнопки, не привязанной к обработчику событий

Свойство Caption кнопки IDC_BUTTON1 изменяем на Form2. После нажатия на этой кнопке будет вызываться диалоговое окно.

3. Этапы построения диалогового окна

В Visual C++ любое диалоговое окно строится в три этапа:

- формирование ресурсов диалогового окна;
- создание класса диалогового окна;
- использование класса диалогового окна.

4. Формирование класса и ресурсов диалогового окна

В MS Visual Studio ресурсы диалогового окна автоматически создаются после создания класса окна. Таким образом, создав класс окна, мы автоматически создаем ресурс окна.

Для работы с классами используется мастер классов Class Wizard. Чтобы вызвать Class Wizard делаем клик правой кнопкой мышки и в контекстном меню выбираем команду «Class Wizard…» (Рис. 5).

Рис. 5. Вызов «Class Wizard…» для создания класса и ресурса диалогового окна

В результате откроется окно, которое изображено на рисунке 6.

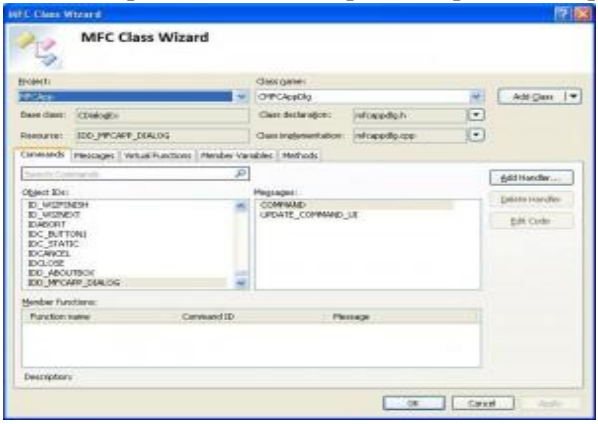

Рис. 6. Окно «MFC Class Wizard»

Окно имеет такие поля:

1. Поле «Project:» – имеющиеся проекты в решении (Solution).
2. Поле «Class Name:» указывает имеющиеся классы в проекте. В нашем случае есть три класса с именами:

- CMFCAppApp – класс приложения в целом;
- CMFCAppDlg – класс диалогового окна приложения;
- CAboutDlg – класс диалогового окна About, что было сформировано при создании проекта мастером AppWizard.

3. Поле «Base Class:» указывает на базовый класс, из которого унаследован класс, который отображается в поле «Class Name:».
4. Поле «Resource:» определяет название ресурса, который отвечает классу из поля «Class Name:».
5. Поле «Class declaration:» определяет название файла-заголовка для класса, который отображается в поле «Class Name:».
6. Поле «Class implementation:» определяет название файла, в котором реализованы функции из класса, который отображается в поле «Class Name:».

Также окно имеет пять вкладок, которые для класса из поля «Class Name:» описывают:

- команды из карты сообщений;
- сообщения Windows, которые генерируются в классе и могут обрабатываться;
- виртуальные функции;
- внутренние переменные, описанные в классе;

- методы, имеющиеся в классе.

Чтобы прибавить новый класс, вызовем команду «Add Class» (рис. 7).

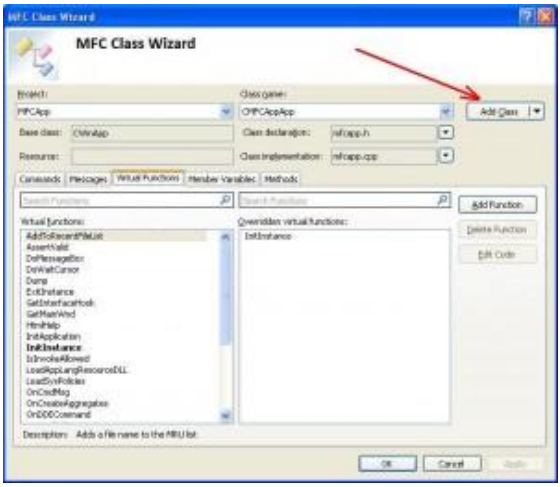

Рис. 7. Команда добавления нового класса

В результате откроется окно «MFC Add Class Wizard», в котором устанавливаем поля в значения, как изображено на рисунке 8. Для установления полей, достаточно в поле «Class Name:» набрать текст «CForm2». Автоматически будут заполнены все другие поля за исключением поля «Base class:».

В поле «Base class:» указывается базовый класс CDialog.

Название ресурса устанавливается как IDD_FORM2.

Заголовочный файл и файл реализации класса имеют названия «Form2.h» и «Form2.cpp».

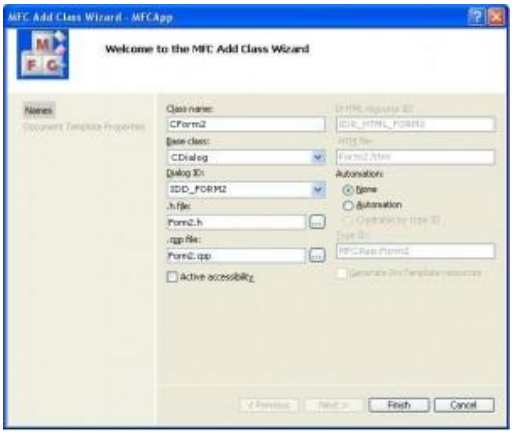

Рис. 8. Создание нового класса CForm2 и ресурса IDD_FORM2, которые базируются на классе CDialog

Делаем клик на «Finish«. В предшествующем окне видно, что к трем предшествующим классам добавлен класс CForm2.

Снова делаем клик на «OK«.

5. Окно MS Visual Studio и панель Solution Explorer

После выполненных действий, окно панели Solution Explorer имеет вид, как показано на рисунке 9.

Рис. 9. Окно Solution Explorer

Как видно из рисунка 9, в списке заголовочных файлов появились файлы «Form2.h» и «Form2.cpp». Соответственно, это являются файлы заголовка и реализации для новообразованного класса CForm2.

7. Вызов диалогового окна класса CForm2 как ресурса

Чтобы начать формировать новое диалоговое окно (класс CForm2), нужно вызвать его как ресурс. Для этого в Solution Explorer делаем двойной клик мышкой на файле «MFCApp.rc» из вкладыша Resources (рис. 10).

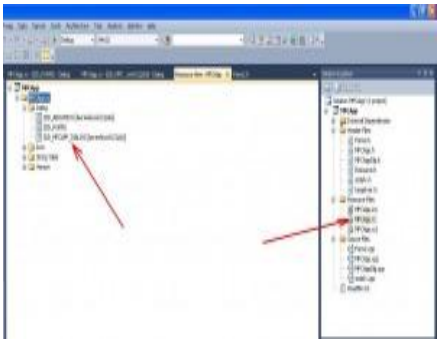

Рис. 10. Вызов списка имен ресурсов приложения

Получаем список имен имеющихся ресурсов:

- IDD_ABOUTBOX – ресурс диалогового окна About;
- IDD_FORM2 – ресурс нового созданного окна;
- IDD_MFC_APP_DIALOG – ресурс главного окна приложения.

Для вызова нужного ресурса, необходимо на нем сделать двойной клик мышкой. Делаем двойной клик на ресурсе IDD_FORM2. В результате, откроется новосозданное диалоговое окно (рис. 11). Как видно из рисунка 11, оно похоже на начальное окно приложения типа Dialog Based Application.

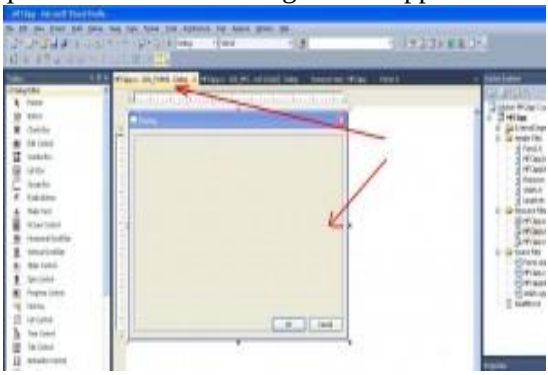

Рис. 11. Окно класса CForm2, который унаследован от класса CDialog

Можно изменить размеры формы и размещение кнопок «OK» и «Cancel».

8. Программирование события клика
на кнопке «Form2» главного окна приложения

Используя «Solution Explorer» и файл ресурсов «MFCApp.rc» (см. п. 7) переходим к ресурсу IDD_MFCAPP_DIALOG главного окна приложения (рис. 12).

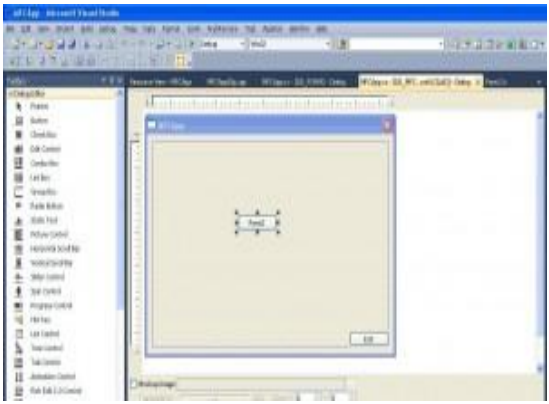

Рис. 12. Главное окно приложения

В окне «Properties» выделяем кнопку с идентификатором IDC_BUTTON1. Потом переходим на вкладыш «Control Events» (рис. 13).

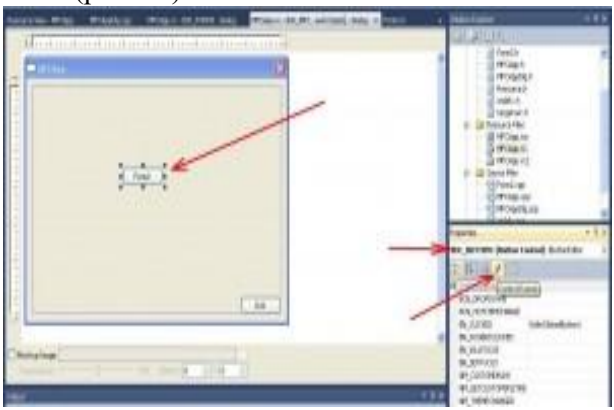

Рис. 13. Список событий из вкладыша Control Events элемента управления IDC_BUTTON1

В списке событий находим событие с названием BN_CLICKED. Потом выбираем кнопку выбора нисходящего меню и в меню выбираем «<Add> OnBnClickedButton1» (рис. 14).

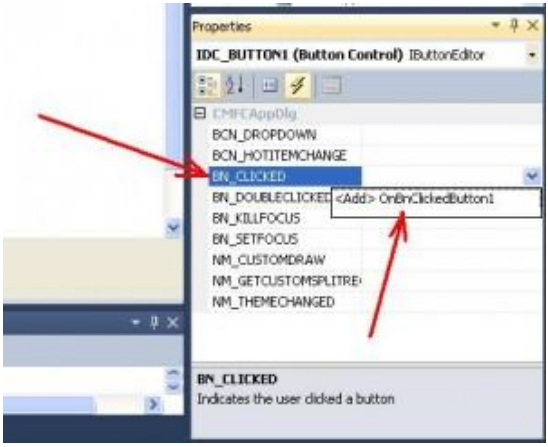

Рис. 14. Выбор события BN_CLICKED и вызов обработчика события OnBnClickedButton1

В результате откроется окно файла «MFCAppDlg.cpp» с выделенным фрагментом кода обработчика события OnBnClickedButton1.

```
void CMFCAppDlg::OnBnClickedButton1()
{
 // TODO: Add your control notification handler code here
}
```

Между скобками { } нужно вставить свой собственный код обработки события.

Программный код функции обработки события клика на кнопке IDC_BUTTON1 следующий:

```
void CMFCAppDlg::OnBnClickedButton1()
{
 // TODO: Add your control notification handler code here
 CForm2 dlg; // создание объекта типа "диалоговое окно"
 CString res_msg; // описание дополнительной переменной для вывода результата
  if (dlg.DoModal()==IDOK) // DoModal() - функция открытия окна
```

```
{
res_msg = "Return from Form2 is OK";
}
else
{
res_msg = "Return from Form2 is Cancel";
}
AfxMessageBox(res_msg); // вывод сообщения на экран
}
```

Теперь можно запустить приложение на выполнение и протестировать его работу.

Разработка кода программного модуля с простым текстовым редактором на элементе управления EditBoxControl.

Запустите C++ и создайте новый проект с именем edit. Поскольку текстовый редактор предназначен для работы с текстовыми строками, нам понадобится компонент Memo из вкладки Standard. Поместите его в центре формы проекта. Теперь поместите на форму компонент MainMenu из той же вкладки. Наконец, добавьте на форму три стандартные кнопки BitButton из вкладки Additional. Они понадобятся для формирования в программе органов управления для открытия и сохранения файлов и для других функций. Поскольку редактор должен при открытии и сохранении файлов обращаться к дисковым накопителям, необходимо добавить на форму диалоги Open Dialog и Save Dialog из вкладки Dialogs. Кроме того, добавьте на форму диалог Font Dialog из этой же вкладки. Он облегчит нам задачу настройки шрифтов из программы. Эти компоненты являются невидимыми при работе программы, поэтому место их размещения на форме не имеет значения. Но все же лучше располагать такие компоненты без наложения, для того чтобы они не скрывали друг друга. Последний компонент, который необходимо поместить на форму, — PopupMenu из вкладки Standard. Он позволит создать в программе контекстное меню, появляющееся при нажатии правой кнопки мыши.

Теперь разместим выбранные компоненты на форме и настроим их свойства. Начнем с главного компонента — формы приложения. Выберите в инспекторе объектов в поле селектора объектов Form1. Измените свойство Caption этого объекта на название программы редактора Edit. Размеры формы установите с помощью мыши, захватив любой угол или сторону формы и перемещая их в любом направлении. Размер формы можно будет поправить при

необходимости на любом этапе разработки программы. В свойстве Position установите значение poDesigned. Это обеспечит центровку нашего приложения при запуске по центру экрана. Щелкните по компоненту MainMenu левой кнопкой мыши. При этом откроется окно дизайнера меню. В свойстве Caption инспектора объектов введите название &File и нажмите клавишу <Enter>. На форме приложения при этом появится строка главного меню. Таким образом, на форме можно будет видеть поле главного меню и учитывать его размеры. Теперь можно временно закрыть окно дизайнера меню и приступить к размещению остальных компонентов на форме. Вначале разместите на форме компонент Memo так, чтобы он занимал всю ее свободную область за исключением правой полосы для кнопок. Кнопки расположите друг под другом, не изменяя при этом их размеров. В оставшейся свободной части формы можно расположить оставшиеся невидимые компоненты приложения. Не забывайте, что перемещать компоненты можно группами, предварительно выделив эти компоненты с помощью мыши. В результате после размещения всех компонентов должна получиться форма, подобная приведенной на рис. 21.1.

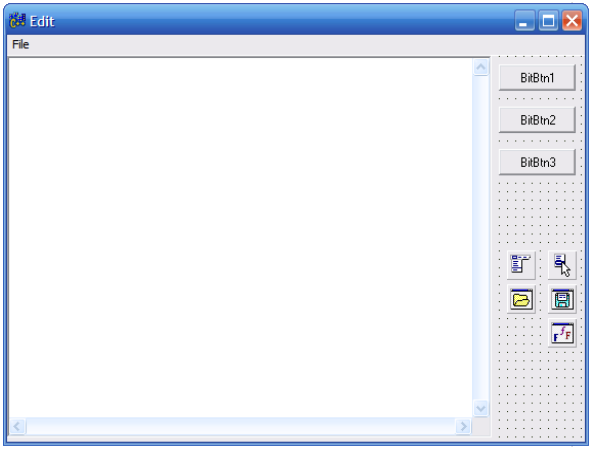

Рис. 21.1. Окно формы программы

Теперь приступим к изменению свойств остальных компонентов. Начнем с компонента Memo, выбрав его в инспекторе объектов. Вначале удалим надпись Memo1 с компонента, для того чтобы она не появлялась при запуске приложения. Для этого

изменим свойство Lines, щелкнув по кнопке с тремя точками правее надписи Tstrings этого свойства. При этом откроется окно редактора строки String List Editor, в котором необходимо удалить строку Memo1 и нажать кнопку OK. Теперь добавим для этого компонента линейки прокрутки для обеспечения возможности просмотра и редактирования больших областей текста с помощью мыши. Для этого в свойстве ScrollBars необходимо выбрать значение ssBoth. Для того чтобы работало контекстное меню, необходимо в свойстве PopupMenu выбрать значение PopupMenu1. Это значение стало доступным после помещения на форму проекта диалога PopupMenu. Настройку данного меню произведем позже.

Настроим свойства кнопок в соответствии с выполняемыми ими функциями. Для этого изменим поочередно свойство Caption для всех трех кнопок на Open, Save и Font соответственно. Для того чтобы тип курсора изменялся при попадании на эти кнопки, изменим еще одно групповое свойство кнопок. Для этого с помощью мыши выделите все три кнопки, и в появившемся объекте 3 items selected в инспекторе объектов измените свойство Cursor на значение crHandPoint.

Настройку главного меню произведем с помощью дизайнера меню, вызываемого двойным щелчком левой кнопки мыши по компоненту MainMenu. Ранее мы уже ввели один пункт меню под названием &File. Символ "ампер-санд" перед File производит выделение первой буквы названия в меню для быстрого доступа, путем ее подчеркивания. Следующий по горизонтали пункт меню назовем &Edit. Теперь заполним пункты меню по вертикали. Щелкните левой кнопкой мыши по пункту меню File, а затем ниже него по пустому полю. Свойство Caption этого пункта меню измените на Open и нажмите клавишу <Enter>. Следующий пункт назовите Save. Аналогично расширим по вертикали пункты меню Edit, добавив пункты Font и Clear.

На этом этап создания главного меню программы редактора завершен. Все введенные пункты меню теперь будут присутствовать в приложении, но пока не будут выполняться, поскольку не созданы функции для выполнения соответствующих команд. Создадим их. Для этого воспользуемся компонентами диалогов, помещенных ранее на форму приложения. С помощью диалогов работы с файлами можно задать направленность приложения путем

определения типов открываемых файлов. Это делается с помощью свойства Filter.

Выберите в инспекторе объектов компонент OpenDialog1 и нажмите кнопку с тремя точками правее свойства Filter. При этом появится диалоговое окно Filter Editor, в котором производится настройка типов файлов. Окно разбито на два поля. В левом поле Filter Name вводятся строки пояснения. В правом поле Filter — расширения файлов. Для указания нескольких типов файлов необходимо записать их через точку с запятой. Если задать несколько строк с пояснениями и типами файлов, то при работе приложения их можно будет выбирать с помощью выпадающего списка. Заполните оба поля диалогового окна Filter Editor в соответствии с рис. 21.2 и нажмите кнопку OK.

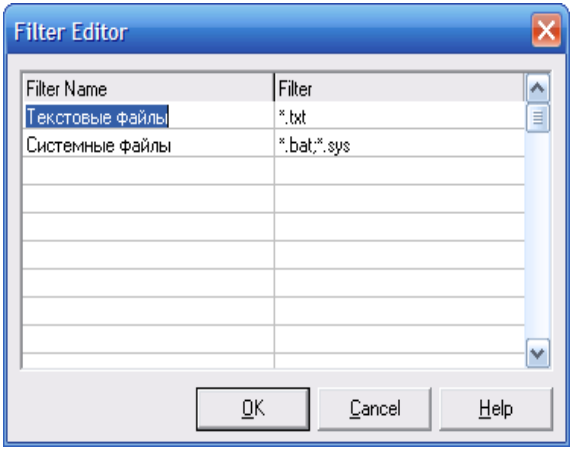

Рис. 21.2. Окно Filter Editor

Аналогично выполните настройку компонента SaveDialog1. Для корректности работы приложения оба эти диалога должны иметь одинаковую настройку свойства Filter. Чтобы облегчить эту работу и избежать ошибок, можно использовать два способа. Первый способ уже был показан ранее, когда несколько компонентов кнопок выделялось в группу и задавалось свойство одновременно для всех выделенных компонентов. Второй способ заключается в том, что, задав свойство для одного из компонентов, можно скопировать и вставить его для других компонентов с помощью комбинации клавиш

<Ctrl>+<C> и <Ctrl>+<V> соответственно. Так, после задания свойства Filter компонента OpenDialog1 с помощью Filter Editor в поле этого свойства появится запись: Текстовые файлы |*.txt| Системные файлы |*.bat; *.sys. Выделив эту строку с помощью клавиш <Home> и <Shift>+<End>, скопируйте ее с помощью клавиш <Ctrl>+<C>. Теперь можно вставить эту строку в свойство Filter для компонента SaveDialog1. Замечательно то, что работу этих компонентов можно проверить еще до выполнения программы. Дело в том, что после задания свойства Filter этих компонентов двойной щелчок по компонентам приводит к открытию диалога на основе заданных свойств. Например, после двойного щелчка по компоненту SaveDialog1 откроется окно, изображенное на рис. 21.3.

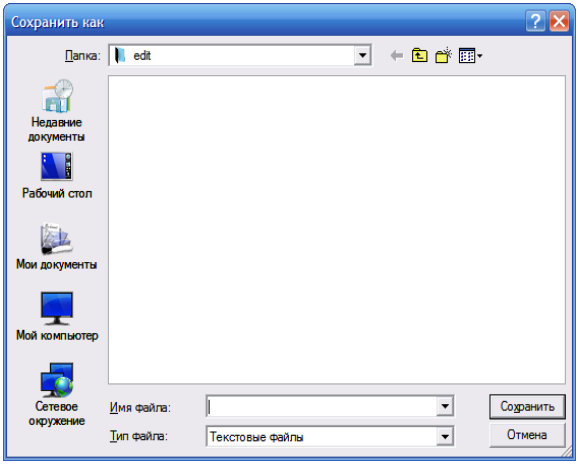

Рис. 21.3. Окно компонента Save Dialog

Теперь настроим компонент FontDialog1. Основным свойством данного компонента является Font, с помощью которого можно выбрать тип шрифта, его цвет, размер, стиль и т. п. Эти установки будут использоваться в программе редактора по умолчанию. С помощью свойства Options можно настроить вид и поведение диалога.Например, можно добавить в него кнопку

справки, установив для подсвойства fdShowHelp значение true.

Нам осталось произвести настройку контекстного меню PopupMenu. Она производится аналогично настройке главного меню MainMenu. Добавим в это меню пункты с

названиями Font и Clear через свойство Caption, как это показано на рис. 21.4.

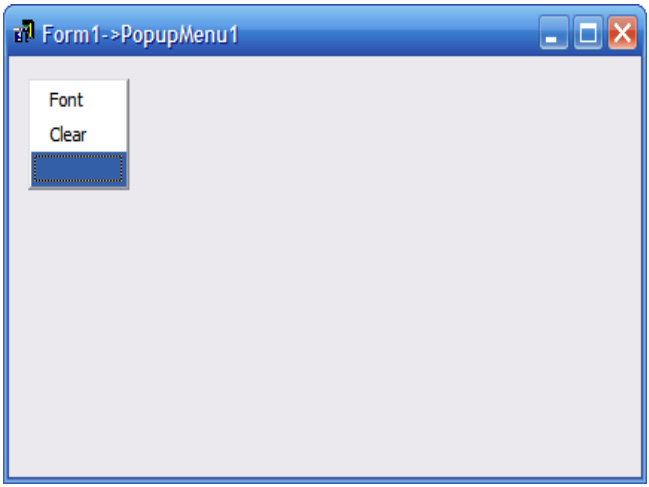

Рис. 21.4. Настройка контекстного меню PopMenu

Крме того, добавьте название PopupMenu1 в свойство PopupMe nu главной формы приложения.

На этом настройка компонентов завершена и получена визуальная оболочка программы редактора. Однако без программных строк это приложение не будет работать. При запуске приложения будут открываться пункты меню, вводиться и редактироваться текст, но только потому, что эти функции заложены в сами компоненты. Нажатие на кнопки и вызов команд из главного меню не будут приводить к каким-либо действиям программы, поскольку не заданы функции обработки событий. Вдохнем в приложение жизнь. В программе редактора имеется четыре типа операций. Это открытие и сохранение файлов, настройка шрифтов и очистка окна редактора. Все эти операции можно вызвать разными способами. Например, с помощью главного меню, кнопок или контекстного меню. Можно написать функции для каждого способа, но это усложнит программу и внесет трудности при ее создании и отладке. Правильнее будет создать такую функцию один раз, а затем задать ее выполнение другим компонентам с помощью инспектора объектов. Создайте функцию для команды главного меню Open. Для этого откройте на главной форме пункт меню File и щелкните левой кнопкой мыши по команде Open. При этом активизируется редактор

кода с заготовкой функции обработки. Введите между фигурными скобками следующий текст программы:

```
if (OpenDialog1->Execute() )
Memo1->Lines->LoadFromFile (OpenDialog1->FileName);
```

Обратите внимание на то, что среда разработки сама предоставляет выбрать функцию из множества разрешенных для компонента функций после введения символа стрелки. Эти программные строки позволят при запуске диалога открытия файла загрузить содержимое открываемого файла в свойство Lines компонента Memo1 для отображения на экране. Теперь задайте выполнение этой функции кнопке команды Open с помощью инспектора объектов. Для этого щелкните левой кнопкой мыши по кнопке Open и выберите в инспекторе объектов вкладку Events (События). Раскройте список в событии OnClick и выберите в нем строку Open1Click. Таким образом будет задано, что при нажатии кнопки в программе редактора запустится та же функция открытия файла, что и для команды Open из пункта главного меню File. Запустите программу с помощью команды Run и убедитесь, что команды открытия файла работают и выбирают заданные нами типы файлов.

Аналогично создадим функции для оставшихся трех типов команд. Начнем с команды сохранения файлов Save. Откройте на главной форме пункт меню File и щелкните левой кнопкой мыши по команде Save. При этом активизируется редактор кода с заготовкой функции обработки. Введите между фигурными скобками следующий текст программы:

```
if (SaveDialog1->Execute() )
Memo1->Lines-> SaveToFile (SaveDialog1->FileName);
```

Эти программные строки позволят при запуске диалога сохранения файла загрузить содержимое свойство Lines компонента Memo1 в сохраняемый файл. Задайте выполнение этой функции кнопке с командой Save с помощью инспектора объектов. Для этого щелкните левой кнопкой мыши по кнопке Save и выберите в инспекторе объектов вкладку Events. Раскройте список в событии OnClick и выберите в нем строку Save1Click. Таким образом будет задано, что при нажатии кнопки Save в программе редактора запустится та же функция сохранения файла, что и для команды Save из пункта главного меню File.

Для команды Font из пункта Edit главного меню введите с помощью инспектора кодов строки:

if (FontDialog1->Execute())

Memo1-> Font=FontDialog1-> Font;

Это позволит изменять шрифт текста компонента Memo1 при открытии диалога FontDialog1. Задайте выполнение этой функции кнопке с командой Font с помощью инспектора объектов, для чего раскройте список в событии OnClick и выберите в нем строку Font1Click.

Теперь нужно добавить вызов этой функции в контекстное меню. Выделите на форме компонент PopupMenu1 и в инспекторе объектов щелкните по кнопке с тремя точками в строке свойства Items. При этом откроется окно Form1 | PopupMenu. В этом окне щелкните на поле Font и в инспекторе объектов на закладке Events выберите для события OnClick строку Font1Click. Наконец создадим функцию обработки очистки редактора. Для команды Clear из пункта Edit главного меню введите с помощью инспектора кодов одну-единственную строку:

Memo1->Clear ();

Эта программная строка будет производить очистку содержимого компонента Memo1. Аналогично предыдущему описанию назначьте выполнение данной функции через контекстное меню, выбрав строку Clear1Click для события OnClick. На этом создание программы редактора заканчивается, и можно приступить к его проверке. Предварительно сохраните весь проект в отдельном каталоге, например, с именем Edit, а затем запустите его на выполнение. На экране должно появиться окно, аналогичное изображенному на рис. 21.5.

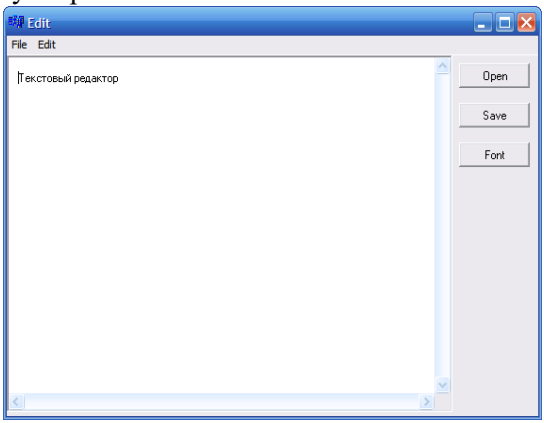

Рис. 21.5. Окно работающей программы

Нажмите кнопку Open и выберите какой-либо файл на диске. После чего загрузите его в редактор. Измените с помощью клавиатуры текст файла и сохраните его с новым именем, нажав кнопку Save. Нажмите кнопку Font и измените шрифт текста редактора. Убедитесь в том, что текст изменился в соответствии с вашими установками. Из главного меню выполните команду Clear и убедитесь, что поле редактирования очистилось. Проверьте работу остальных пунктов главного меню. Далее проверьте работу контекстного меню, нажав правую кнопку мыши на поле редактора и выбрав одну из двух команд. Обратите внимание на то, что при наведении курсора на кнопки он должен менять свой вид.

Редактор готов к работе. Создайте для него иконку, как это делали раньше, и используйте редактор как отдельное законченное приложение. Со временем вы сможете добавлять в него новые функции и возможности, повышая тем самым его функциональность и свое мастерство.

Тема 4. Растровая графика. Разработка модуля с элементами графики.

Разработка кода программного модуля с использованием функций создания графического образа. Отладка программы.

Инициализация графического режима

Самым первым вопросом работы с графикой является проверка работоспособности графического режима в вашей среде программирования. Следующий пример (листинг 4.1) написан именно с данной целью. Скомпилируйте его.

Листинг 2.1. Демонстрация работы графического режима

// proba1.cpp #include <graphics.h> #include <stdlib.h> #include <stdio.h> #include <conio.h> int main(void)

{/* 1. Автоматическое определение наибольшего графического режима: gdriver = DETECT */

```
int gdriver = DETECT, gmode, errorcode; /* Инициализация графического режима */ initgraph(&gdriver, &gmode, "");
/* Результат инициализации */ errorcode = graphresult();
if (errorcode != grOk) /* Если ошибка */
{printf("Ошибка инициализации: %s\n", grapherrormsg(errorcode)); printf("Нажмите любую клавишу");
getch(); exit(1); /* Завершить программу */
}
/* 2. Рисование линии из правого верхнего угла в левый нижний */ line(0, 0, getmaxx(), getmaxy());
/* 3. Закрытие графического режима */ getch(); closegraph();
return 0;
}
```

Если вы не внесли опечаток в текст программы, тогда файл proba1.exe начнет работать. По диагонали экрана должна рисоваться на черном фоне белая линия из правого верхнего угла в левый нижний угол.

2.1.2 Типовые ошибки инициализации

Возможны три причины, по которым программа proba1.exe с графикой будет работать не так, как мы предположили.

Первая — отсутствие в рабочей папке с исполняемым файлом proba1.exe файла драйвераegavga.bgi. Результатом работы программы будет сообщение:

Ошибка инициализации: Device driver file not found <EGAVGA.BGI> Нажми любую клавишу (Press any key to halt)

Это как-тодаже неуместно называть проблемой. Из папкиBGI среды программирования файлegavga.bgi переписывается в вашу рабочую папку и все.

Другой вариант. Файл egavga.bgi «лежит» в папке рядом с файломproba1.exe, но линия не рисуется. Если программаproba1.exe начинает вытворять неизвестно что (или выдает другую ошибку или по экрану начинает «бегать» курсор), тогда смело сотрите старый драйверegavga.bgi и поищите у знакомых или на дисках новую версию. «Ломаные» драйверы встречаются не так редко.

Третья причина — это экзотический «баг». Она встречается в основном в компьютерных сетях. По неизвестной причине файл

драйвера egavga.bgi отказывается работать до тех пор, пока с этого файла не снять атрибут «только для чтения». Надеюсь, что вы не работаете с удаленного компьютера в сети, поскольку там есть еще ряд особенностей работы среды программирования, связанных загрузкой среды с другого компьютера и атрибутами «только для чтения».

2.1.3 Графические примитивы

Основу мы уже заложили: знаем, как переключиться в графический режим, где применить функции и как правильно завершить работу графической части. Теперь займемся непосредственно разбором примеров с рисованием графических функций. Самые простые функции называются графическими примитивами. Условно примитивы можно разделить на группы: примитивы рисования контуров и площадные фигуры. К примитивам контуров относятся: линии (line), прямоугольники (rectangle), дуги (arc), окружности (circle), эллипсы (ellipse), многоугольники (drawpoly) и прочие не закрашиваемые внутри фигуры. Площадными, закрашиваемыми внутри фигурами являются прямоугольники (bar,bar3d), круговые и эллиптические секторы (pieslice, sector). Если у вас фигура замкнута, но не залита, например, многоугольник (drawpoly), то его всегда можно закрасить внутри с помощью функций закраски (floodfill,setfillstyle). Для выбора цвета рисования применяется функция задания текущего (по умолчанию) цвета графических примитивов (setcolor). Для определения цвета точки по ее координатам на экране существует функциягetpixel, а для вывода точки заданным цветом на экран — setpixel. Вот и весь смысл работы с выводом графических примитивов. Данная логика интуитивно понятна обычному школьнику. Подробное описание и параметры графических функций можно найти в любом справочнике по C++. Если у вас под рукой нет справочника, тогда наберите в редакторе среды программирования интересующее вас имя функции, наведите на него курсор и нажмите комбинацию клавиш Ctrl+F1. На экране появится описание функции и внизу подсказки ссылка на пример программы с выбранной графической функцией. Например, если вы введете имя функцииrectangle и нажмете нужные клавиши, появится подсказка по функции рисования прямоугольника с характерным примером применения.

Нокое-чтовы можете увидеть и в примере из листинга 2.2, в котором воедино собраны некоторые распространенные функции для работы с примитивами.

Листинг 2.2. Демонстрация работы графических функций

```
//rectangl.cpp #include <graphics.h> #include <stdlib.h> #include <stdio.h> #include <conio.h> void main(void)
{int gdriver = DETECT, gmode, errorcode; int xr=50,yr=20,left, top, right, bottom;
int stangle=0, endangle=360, xradius=40, yradius=25; int radius=25;
int stangle_arc = 90, endangle_arc = 225; int xb=0, yb=200, midx, midy;
//1 Инициализация графического режима
initgraph(&gdriver, &gmode, ""); errorcode = graphresult(); if (errorcode != grOk) {
printf("Ошибка: %s\n", grapherrormsg(errorcode)); printf("Нажмите любую клавишу"); getch(); exit(1); }

// 2 Рисование

setcolor(WHITE);

outtextxy(xr,yr+100,
                                                                    arc"
"rectangle ellipse circle );
left = xr +0; top = yr +0; right = xr+100; bottom = yr+50;
rectangle(left,top,right,bottom);
ellipse(xr+150, yr+25, stangle, endangle, xradius, yradius); circle(xr+250, yr+25, radius);
arc(xr+350, yr+25, stangle_arc, endangle_arc, radius); midx = xb; midy = yb;
7
outtextxy(xb+100, yb+60, "bar + setfillstyle"); outtextxy(xb+100, yb+200, "sector + setfillstyle"); for (int i=SOLID_FILL; i<USER_FILL; i++)
{ setfillstyle(i, YELLOW);
bar(midx+i*50, midy, midx+40+i*50,midy+40);
sector(midx+i*50+35, midy+150, stangle_arc, endangle_arc, xradius, yradius);
}
// 3 Закрытие графического режима getch(); closegraph(); }
```

В этой простой программе обратите внимание на перечисление типов закраски площадных фигур в цикле for (int i=SOLID_FILL; i<USER_FILL; i++) отI=1 доI=11. Всего существует

13 типов стандартных заливок замкнутых областей, назначение которых перечислено в табл. 4.1.

Работа с растровым изображением.

Создайте у себя в проекте следующие файлы:

1. Image.h - header file;
2. Image.cpp - cpp code file;

В файле Image.h будут храниться описания функций, а в файле Image.cpp - реализации функций.

В названиях функций и констант вы встретите приставку "ag", вместо неё вы можете поставить любую другую. Это сделано для того чтобы не было дубликатов функций или констант в разных API.

Файл Image.h

Код:

```
/*http://esate.ru, Alexei16*/
#pragma once
#include <windows.h>
#include <il.h>                    //|
                                // Подключаем DevIL
#pragma comment(lib, "DevIL.lib") //|
enum Pixelformat
{
  AG_BGR,  //Формат пикселей B|G|R
      AG_BGRA, //Формат пикселей B|G|R|A
      AG_RGB,  //Формат пикселей R|G|B
      AG_RGBA  //Формат пикселей R|G|B|A
};
enum ImageType
{
  AG_BMP = 0x0420,   //!< Microsoft Windows Bitmap - .bmp extension
  AG_JPG = 0x0425,  //!< JPEG - .jpg, .jpe and .jpeg extensions
  AG_PNG = 0x042A,   //!< Portable Network Graphics - .png
```

```
extension
        AG_TGA = 0x042D,  //!< TrueVision Targa File - .tga, .vda, .icb
and .vst extensions
        AG_TIF = 0x042E,  //!< Tagged Image File Format - .tif and .tiff
extensions
        AG_JP2 = 0x0441   //!< Jpeg 2000 - .jp2 extension
      };
      typedef struct BaseIMGInfo
      {
        int Depth;      //Глубина изобаржения(нужно только для 3D
текстур), по умолчанию = 1
        int Format;  //Формат пикселей
        int Type;          //Тип данных в памяти, в основном =
IL_UNSIGNED_BYTE
      };
      typedef struct ImageData
      {
        unsigned char * Data; //Указатель на данные изображения
        int Width;          //Ширина изображения
        int Height;         //Высота изображения
        int Bpp;            //Количество байт на пиксель
        int Stride;         //Количество байт в одной строке пикселей
        Pixelformat PixFormat;//Формат пикселей
        BaseIMGInfo BaseInfo; //Основные параметры изображения,
нужные для сохранения изображений в DevIL
        int R_idx;          //Позиция красного в пикселе
        int G_idx;          //Позиция зелёного в пикселе
        int B_idx;          //Позиция синего в пикселе
        int A_idx;          //Позиция альфа-канала в пикселе, только для
AG_RGBA или AG_BGRA
      };
      ImageData * agLoadImage(char * FileName); //Функция для
загрузки изображения
      bool agSaveImage(ImageData * IMG, char * FileName, ImageType
IType); //Функция для сохранения изображения
      ImageData * agNewImage(int Width, int Height,  Pixelformat
PFormat); //Функция для создания нового,пустого, изображения
      ImageData * agCloneImage(ImageData * Src); //Функция для
создания копии изображения
```

Директива препроцессора #pragma once нужна для контроля: конкретный файл должен подключаться при компиляции только один раз.

Теперь перейдём к реализации.

Откройте файл Image.cpp и включите в него Image.h:

Код:

```
/*http://esate.ru,
Alexei16*/
#include "Image.h"
```

Теперь разберём каждую функцию по отдельности.

1. Функция agLoadImage

Код:

```
/*http://esate.ru, Alexei16*/
ImageData * agLoadImage(char * FileName) //FileName - указатель на строку с именем файла
{
  ILuint * Texture = new ILuint; //Указатель на текстуру
  ImageData * Result = NULL; //Изображение
  ilInit();              //И и и л з ц я D v L
  ilEnable(IL_ORIGIN_SET); // н ц а и а и  е I
  ilGenImages(1, Texture); //Генерируем текстуру
  ilBindImage(*Texture);  //Делаем текстуру текущей
  if(ilLoadImage(FileName) == true) //Пытаемся загрузить изображение
  { // если да, то
    Result = new ImageData(); //Создаём изображение
    Result -> Width = ilGetInteger(IL_IMAGE_WIDTH); //Получаем ширину изображения
    Result -> Height = ilGetInteger(IL_IMAGE_HEIGHT);//Получаем высоту изображения
    Result -> Bpp = ilGetInteger(IL_IMAGE_BYTES_PER_PIXEL);//Получаем количество байт на пиксель
    Result -> Stride = Result -> Width * Result -> Bpp;//Вычислем количество байт в одной строке пикселей
    Result -> Data = new unsigned char[Result -> Width * Result -> Height * Result -> Bpp]; //Создаём указатель на данные изображения
    memcpy(Result -> Data, ilGetData(), ilGetInteger(IL_IMAGE_SIZE_OF_DATA)); //Копируем данные из памяти в наше изображение
```

```
        Result -> BaseInfo.Type = ilGetInteger(IL_IMAGE_TYPE);//Получаем тип данных в памяти
        Result -> BaseInfo.Format = ilGetInteger(IL_IMAGE_FORMAT);//Получаем формат пикселей
        Result -> BaseInfo.Depth = ilGetInteger(IL_IMAGE_DEPTH);//Получаем глубину изображения
        if(Result -> BaseInfo.Format == IL_RGB) //Если формат пикселей BGR, то
        {                                //данные располагаются в памяти
          Result -> R_idx = 0;          //так ->
          Result -> G_idx = 1;          //0|1|2|0|1|2|0|1|2|0|1
          Result -> B_idx = 2;          //R|G|B|R|G|B|R|G|B|R|G
        }
        else if(Result -> BaseInfo.Format == IL_RGBA) //Если формат пикселей RGBA,то
        {                                //данные располагаются в памяти
          Result -> R_idx = 0;          //так ->
          Result -> G_idx = 1;          //0|1|2|3|0|1|2|3|0|1|2|3|0
          Result -> B_idx = 2;          //R|G|B|A|R|G|B|A|R|G|B|A|R
          Result -> A_idx = 3;
        }
        else if(Result -> BaseInfo.Format == IL_BGR) //Если формат пикселей BGR,то
        {                                //данные располагаются в памяти
          Result -> R_idx = 2;          //так ->
          Result -> G_idx = 1;          //0|1|2|0|1|2|0|1|2|0|1|2
          Result -> B_idx = 0;          //B|G|R|B|G|R|B|G|R|B|G|R
        }
        else if(Result -> BaseInfo.Format == IL_BGRA) //Если формат пикселей BGRA, то
        {                                //данные располагаются в памяти
          Result -> R_idx = 2;          //так ->
          Result -> G_idx = 1;          //0|1|2|3|0|1|2|3|0|1|2|3|0
          Result -> B_idx = 0;          //B|G|R|A|B|G|R|A|B|G|R|A|B
          Result -> A_idx = 3;
        }
        else
        {
          MessageBoxA(0, "Unsuitable pixel format", "Error", MB_OK | MB_ICONERROR);
```

```
    }
  }
  else
  { //Если нет, то выдаём сообщение об ошибке
    MessageBoxA(0, "Could not load image", "Error", MB_OK |
MB_ICONEXCLAMATION);
  }
  ilDeleteImages(1, Texture); //Удаляем текстуру ->
  delete Texture;          //->|

  return Result;
}
```

2. Функция agSaveImage

Код:

```
/*http://esate.ru, Alexei16*/
bool agSaveImage(ImageData * IMG, char * FileName, ImageType
IType) //IMG - изображение, FileName - имя сохраняемого файла,
IType - расширение файла
{
  bool Res;//Результат функции
  if(IMG != NULL && strlen(FileName) > 0)//Если изображение
не пустое и длинна имени файла больше нуля,то
  {
    ILuint * Texture = new ILuint;//Указатель на текстуру
    ilInit(); //Инициализация DevIL
    ilGenImages(1, Texture);//Генерируем текстуру
    ilBindImage(*Texture);  //Делаем её текущей
    ilTexImage(IMG -> Width,IMG -> Height,
        IMG -> BaseInfo.Depth,IMG -> Bpp,IMG ->
BaseInfo.Format,IMG -> BaseInfo.Type,
        IMG -> Data);// Закидываем данные в память
    Res = ilSave(IType, FileName);//И сохраняем изображение
    ilDeleteImages(1, Texture); //Удаляем текстуру ->
    delete Texture;          //->|
  }
  else
  { //Иначе,выдаем сообщене об ошибке
    MessageBoxA(0, "Pointer to the image is empty or the wrong
name of the saved file", "Error", MB_OK | MB_ICONERROR);
  }
```

```
  return Res;
}
```

3. Функция agNewImage

Код:

```
/*http://esate.ru, Alexei16*/
ImageData * agNewImage(int Width, int Height,  Pixelformat PFormat)
{
  ImageData * Result = NULL;//Результат функции
  if(Width > 0 && Height > 0) //Если длинна и ширина - положительные величины, то
  {
    Result = new ImageData();//Создаем изображение
    Result -> Width = Width; // Присваиваем длинну
    Result -> Height = Height; // Присваиваем ширину
    Result -> PixFormat = PFormat; // Присваиваем формат пикселей
    Result -> BaseInfo.Depth = 1;
    Result -> BaseInfo.Type = IL_UNSIGNED_BYTE;
    if(Result -> PixFormat == AG_BGR)
    {
      Result -> R_idx = 2;
      Result -> G_idx = 1;
      Result -> B_idx = 0;
      Result -> BaseInfo.Format = IL_BGR;
      Result -> Bpp = 3; //3 байта на пиксель, потомучто B(1 byte) + G(1 byte) + R(1 byte) = 3 bytes
    }
    else if(Result -> PixFormat == AG_BGRA)
    {
      Result -> R_idx = 2;
      Result -> G_idx = 1;
      Result -> B_idx = 0;
      Result -> A_idx = 3;
      Result -> BaseInfo.Format = IL_BGRA;
      Result -> Bpp = 4; //4 байта на пиксель, потомучто B(1 byte) + G(1 byte) + R(1 byte) + A(1 byte) = 3 bytes
    }
    else if(Result -> PixFormat == AG_RGB)
    {
```

```
            Result -> R_idx = 0;
            Result -> G_idx = 1;
            Result -> B_idx = 2;
            Result -> BaseInfo.Format = IL_RGB;
            Result -> Bpp = 3;
          }
          else if(Result -> PixFormat == AG_RGBA)
          {
            Result -> R_idx = 0;
            Result -> G_idx = 1;
            Result -> B_idx = 2;
            Result -> A_idx = 3;
            Result -> BaseInfo.Format = IL_RGBA;
            Result -> Bpp = 4;
          }
          Result -> Data = new unsigned char[Result -> Width * Result ->
Height * Result -> Bpp]; //Выделяем память под данные изображения
          Result -> Stride = Result -> Width * Result -> Bpp;
//Вычисляем количество байт на строку пикселей
        }
        else
        { // Иначе выдаем сообщение об ошибке
          MessageBoxA(0, "Invalid input parameters", "Error", MB_OK |
MB_ICONERROR);
        }
        return Result;
      }
```

4. Функция agCloneImage

Код:

```
/*http://esate.ru, Alexei16*/
ImageData * agCloneImage(ImageData * Src) //Src - исходное
изображение
{
  ImageData * Dst = NULL; //Новое изображение
  if(Src != NULL) // Если исходное изображение не пустое,то
  {
    Dst = new ImageData();//Создаем новое изображение
    Dst -> B_idx = Src -> B_idx;// далее все просто копируем
    Dst -> G_idx = Src -> G_idx;
```

```
            Dst -> R_idx = Src -> R_idx;
            Dst -> A_idx = Src -> A_idx;
            Dst -> Width = Src -> Width;
            Dst -> Height = Src -> Height;
            Dst -> Stride = Src -> Stride;
            Dst -> Bpp = Src -> Bpp;
            Dst -> PixFormat = Src -> PixFormat;
            Dst -> BaseInfo.Depth = Src -> BaseInfo.Depth;
            Dst -> BaseInfo.Format = Src -> BaseInfo.Format;
            Dst -> BaseInfo.Type = Src -> BaseInfo.Type;
            try
            {
              Dst -> Data = new unsigned char[Dst -> Width * Dst ->
Height * Dst -> Bpp];
              memcpy(Dst -> Data, Src -> Data, Dst -> Width * Dst ->
Height * Dst -> Bpp);
            }
            catch(...)
            {
              MessageBoxA(0, "Can not allocate memory", "Error",
MB_OK | MB_ICONERROR);
              delete Dst;
            }
          }
          else
          { //иначе, выдаем сообщение об ошибке
            MessageBoxA(0, "Invalid input parameters", "Error", MB_OK |
MB_ICONERROR);
          }
          return Dst;
        }
```

Тема 5. Библиотека динамической компоновки DLL. Использование принципов структурного программирования

Разработка кода программного модуля с использованием функций библиотеки DLL.

Создание DLL

Ничего особенного здесь нет. Как обычно, вы просто пишите функции, как в обычной программе. Если вы используете MSVC, создайте новый проект и укажите, что вы создаете Win32 Dynamic-Link Library. После компиляции вы получите DLL, библиотеку импорта (.lib) и библиотеку экспорта (.exp). Далее показан примерный код вашей DLL:

Заголовочный файл (DLLTEST.H)

```
#ifndef _DLLTEST_H_
#define _DLLTEST_H_
#include <iostream.h>
#include <stdio.h>
#include <windows.h>
extern "C" __declspec(dllexport) void NumberList();
extern "C" __declspec(dllexport) void LetterList();
#endif
```

Код библиотеки (DLLTEST.CPP)

```
#include "dlltest.h"
#define MAXMODULE 50
char module[MAXMODULE];

extern "C" __declspec(dllexport) void NumberList()
{
  GetModuleFileName(NULL, (LPTSTR)module, MAXMODULE);
    cout << "\n\nThis function was called from "
    << module
    << endl << endl;
    cout << "NumberList(): ";
    for(int i=0; i<10; i++)
    {
      cout << i << " ";
    }
```

```
    cout << endl << endl;
}
extern "C" __declspec(dllexport) void LetterList()
{
    GetModuleFileName(NULL, (LPTSTR)module,
MAXMODULE);
    cout << "\n\nThis function was called from "
    << module
    << endl << endl;
    cout << "LetterList(): ";
    for(int i=0; i<26; i++)
    {
      cout << char(97 + i) << " ";
    }
    cout << endl << endl;
}
```

Как видите, ничего особенного в коде нет. Приложение, используемое для примера - консольное, так что здесь просто запрограммированы две функции, выводящие текст. Строка

```
extern "C" __declspec(dllexport)
```

означает, что функция будет видна вне DLL (т.е. ее можно вызывать из нашей программы).

После компиляции мы получим библиотеку. Теперь посммотрим, как ее можно использовать.

Использование DLL (с библиотекой импорта)

Сначала посмотрим, как использовать DLL вместе с библиотекой импорта (dlltest.lib), которая получается при компиляции предыдущего примера. Этот метод очень прост, так как в таком случае нужно просто включить заголовочный файл библиотеки и саму библиотеку в проект. Пример:

Исходный файл приложения, использующего библиотеку DLL (DLLRUN01.EXE)

```
#include <conio.h>
#include <dlltest.h>
void main()
{
    NumberList();
    LetterList();
    getch();
```

```
}
```

Это будет прекрасно работать, если у вас есть заголовочный файл и библиотека импорта (dlltest.lib) находится в каталоге, прописанном в библиотечных путях. Перед запуском приложения, убедитесь, что DLL находится в каталоге, прописанном в системной переменной PATH или в том же каталоге, что и исполняемый файл, иначе вы получите сообщение об ошибке. Однако если 10 программ используют эту DLL, вам нужна всего одна ее копия, лежащая, например, в каталоге Windows\System.

Результаты работы DLLRUN01.EXE

```
This function was called from C:\DLLTEST\DLLRUN01.EXE
NumberList(): 0 1 2 3 4 5 6 7 8 9
This function was called from C:\DLLTEST\DLLRUN01.EXE
LetterList(): a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Использование DLL без библиотеки импорта

Теперь посмотрим, как загрузить DLL "на лету". Это нужно в случае, если не вы разрабатывали, эту DLL и у вас нет заголовочного файла и библиотеки импорта.

Исходный файл приложения, использующего библиотеку DLL - консольное приложение Win32 (DLLRUN01.EXE).

```
#include <windows.h>
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#define MAXMODULE 50
typedef void (WINAPI*cfunc)();
cfunc NumberList;
cfunc LetterList;
void main()
{
  HINSTANCE hLib=LoadLibrary("DLLTEST.DLL");
   if(hLib==NULL)
   {
     cout << "Unable to load library!" << endl;
     getch();
     return;
   }
  char mod[MAXMODULE];
```

```
        GetModuleFileName((HMODULE)hLib,          (LPTSTR)mod,
MAXMODULE);
        cout << "Library loaded: " << mod << endl;
        NumberList=(cfunc)GetProcAddress((HMODULE)hLib,
"NumberList");
        LetterList=(cfunc)GetProcAddress((HMODULE)hLib,
"LetterList");
         if((NumberList==NULL) || (LetterList==NULL))
         {
           cout << "Unable to load function(s)." << endl;
           FreeLibrary((HMODULE)hLib);
           return;
         }
         NumberList();
         LetterList();
         FreeLibrary((HMODULE)hLib);
         getch();
      }
```

Этот код загружает DLL (если она находится в путях или в текущем каталоге), а затем определяет адреса функций, которые мы будем вызывать. Конечно, в этом случае пришлось написать намного больше кода, и, соответственно, придется отловить немало ошибок. Однако такой подход универсальный.

Результаты работы DLLRUN02.EXE

Library loaded: C:\DLLTEST\DLLTEST.DLL

This function was called from C:\DLLTEST\DLLRUN02.EXE

NumberList(): 0 1 2 3 4 5 6 7 8 9

This function was called from C:\DLLTEST\DLLRUN02.EXE

LetterList(): a b c d e f g h i j k l m n o p q r s t u v w x y z

Разработка кода программного модуля с использованием неявного связывания.

Рассмотрим пример неявного связывания, которое менее гибко по сравнению с явным связыванием. В данном случае будем использовать ту же самую динамическую библиотеку «dll1.asm». Содержание файла «dllex1.asm» приведено в приложении 3. Как видно, текст программы стал несколько проще. Здесь важно заметить, что, во-первых, необходимо объявить вызываемую из динамической библиотеки процедуру как внешнюю, а, во-вторых, подключить статическую библиотеку «dll1.lib», которая

автоматически создается трансляторомMASMпри создании динамической библиотеки «dll1. dll».

Трансляция файла «dllex1.asm» производится с помощью команд:

ml /c /coff /DMASM dllex1.asm

link /subsystem:windows dllex1.obj

Результат работы программы «dllex1.asm» изображен на рисунке 3.

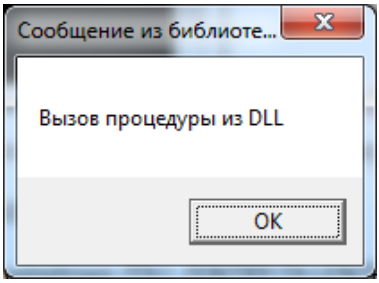

Рисунок 3 – Результат работы программы «dllex1.asm»

3. Общая память

Пример использования динамической библиотекой и программой общего адресного пространства показан в приложении 4. В этом приложении приведены примеры программ «dll2.asm» и «dllex2.asm». Процесс передает адреса строк, которые находятся в блоке данных основного процесса. В свою очередь, процедура возвращает в основной процесс адрес строки, находящейся в блоке данных динамической библиотеки.

Трансляция файла «dllex2.asm» производится с помощью команд:

ml /c /coff /DMASM dllex2.asm

link /subsystem:windows dllex2.obj

Результат работы программы «dllex2.asm» изображен на рисунке 4.

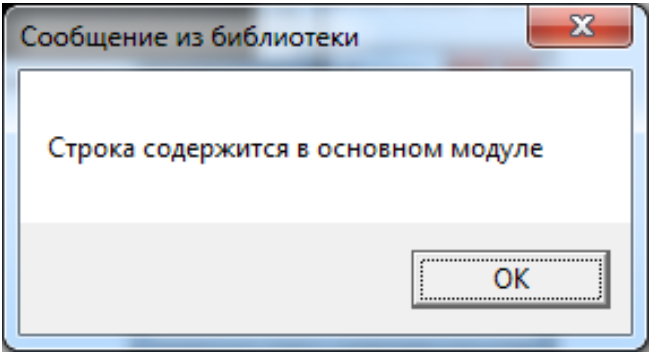

Рисунок 4 – Результат работы программы «dllex2.asm»

Рассмотрим еще один пример, приведенный в приложении 5. В этом приложении основной процесс использует ресурсы загруженной им динамической библиотеки. Программа вначале загружает иконку из ресурсов динамической библиотеки и устанавливает ее на окно. Если нажать левой кнопкой мыши, направив курсор на окно, то будет вызываться процедура из динамической библиотеки, которая будет поочередно устанавливать то один, то другой значок на окно.

Трансляция файлов, приведенных в приложении 5, производится с помощью команд:

```
ml /c /coff /DMASM dllex3.asm
rc dllex3.rc
link /subsystem:windows dllex3.obj dllex3.res
ml /c /coff /DMASM dll3.asm
rc dll3.rc
link /subsystem:windows /DLL /ENTRY:DLLENTRY dll3.obj dll3.res
```

Результат работы программы «dllex3.asm» изображен на рисунках 5 и 6.

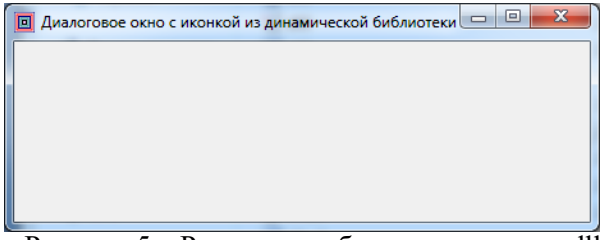

Рисунок 5 – Результат работы программы «dllex3.asm» при запуске программы

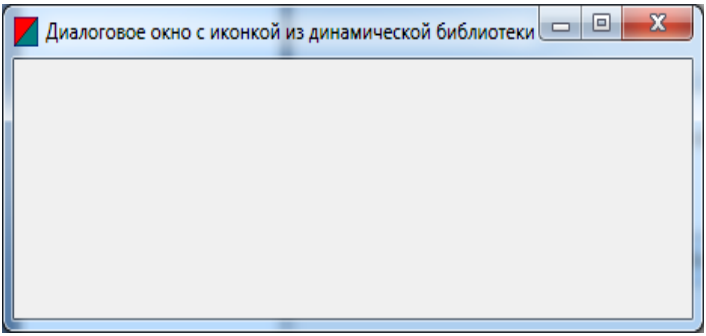

Рисунок 6 – Результат работы программы «dllex3.asm» при запуске программы

Тема 6. Методы и средства разработки технической документации программного продукта.

Проектирование структуры документа с технической документацией программного продукта. Расположение материала. Типы информации и их компоновка. Разработка пояснительной записки в текстовом редакторе. Создание руководства пользователя программного продукта.

Цель работы: Получение практических навыков в оформление документации на программные средства с использованием инструментальных средств

Документирование программного обеспечения

Качество программного обеспечения, наряду с другими факторами, определяется полнотой и качеством пакета документов, сопровождающих ПО. К программным документам относятся документы, содержащие сведения, необходимые для разработки, изготовления, сопровождения программ и эксплуатации.

Техническое задание

Техническое задание. Требование к содержанию и оформлению. Напомним, что техническое за-дание (ТЗ) содержит совокупность требований к ПС и может использоваться как критерий проверки и приемки разработанной программы. Поэтому достаточно полно составленное (с учетом возможности внесения дополнительных разделов) и принятое заказчиком и разработчиком, ТЗ является одним из ос-новополагающих документов проекта программного средства.

Техническое задание на разработку ПО должно включать следующие разделы:

- введение;
- основания для разработки;
- назначение разработки;
- требования к программе;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки;
- приложения.

В зависимости от особенностей разрабатываемого ПО стандарт допускает уточнение содержания разделов, введение новых разделов или их объединение.

В разделе “Введение” указывается наименование, краткая характеристика области применения ПО.

В разделе “Основания для разработки” указывается: документ (документы), на основании которых ведется разработка; организация, утвердившая документ, и дата утверждения; наименование (условное обозначение) темы разработки.

В разделе Назначение разработки должно быть указано функциональное и эксплуатационное назначение ПО.

Например, UML – как универсальный язык моделирования. Может использоваться и для поста-новки технического задания.

Внешние и внутренние языки спецификации

В процессе разработки ПО появляются следующие документы, перечисленные ниже в хронологическом порядке: Соглашение о требованиях; Внешняя спецификация; Внутренняя спецификация.

Документ “Соглашение о требованиях” должен содержать первое письменное соглашение между заказчиком и разработчиком о том, что будет сделано, и что не будет делаться при разработке и выпуске программного обеспечения. В отличие от него спецификация предполагает наличие более точных и исчерпывающих формулировок и определений. При этом, первые два документа содержат информацию о том, что представляет собой ПО; а третий должен объяснять, как ПО устроено и как достигаются установленные для него цели и требования. Все документы имеют схожую структуру для облегчения контроля над проектом, а также для обеспечения прослеживаемости всех технических решений от требований до их реализации. По мере продвижения проекта разделы документа либо просто копируются в соответствующие разделы следующего создаваемого документа, либо расширяются описаниями технических решений текущего этапа.

Ниже приведена общая структура документа “Внешняя спецификация”, с развернутыми комментариями в тех пунктах, которые касаются технической стороны дела:

Описание программного изделия

1.1. Наименование и шифры ПО (полное наименование, сокращенные наименования, шифры ПО и проекта).

1.2. Краткое описание ПО (включая сведения об авторском праве, иерархию документов, с указанием документов вышестоящих уровней).

1.3. Результирующие компоненты ПО (оформляется в виде таблицы или другой формы и включает в себя, перечень спецификаций, другой документации и компонентов программного обеспечения).

2. Цели

Этот раздел содержит причины выпуска ПО с указанием различного типа заявок, планов и т.п. и носит полностью управленческий характер.

Стратегия

3.1. Соглашения относительно представления материала.

3.1.1. Обозначения (определяются все обозначения, используемые в требованиях: например, если применяются индексы, то дается пример их использования и определяется принцип индексации).

3.1.2. Терминология (особенно специфическая для данного изделия).

3.1.3. Синтаксис (приводятся, если необходимо, синтаксические правила для дальнейшего описания требований).

3.2. Генерируемое программное обеспечение (классифицируется как вспомогательное и порождаемое описываемым изделием).

3.3. Системное программное обеспечение (все остальное ПО, включая ОС, утилиты, пакеты прикладных программ, которое классифицируется как основное, поскольку оно генерирует ПО предыдущего пункта).

3.3.n. Общие характеристики функции n. Если технически затруднительно и неестественно рассматривать ПО как один большой функциональный модуль, то следует привести его функциональную декомпозицию, показав связи между функциями (функциональными модулями) и присвоив каждой функции некоторое уникальное имя n. Затем для каждой функции отводится подраздел раздела 3.3 (т.е. 3.3.1, 3.3.2 и т.д.), в заглавии которого используется слово функция с последующим именем функционального модуля. Такая функциональная декомпозиция не указывает, как именно ПО будет фактически разбито на программные модули (это составляет содержание документа Внутренняя спецификация). Для удобства работы, конечно, полезно иметь некоторое соответствие функционального и фактического разбиения, но это не является требованием и не должно уводить с правильного пути проектирования изделия.

3.3.n.1. Внешние ограничения

3.3.n.1.1. Стандарты (список используемых промышленных стандартов и собственных стандартов предприятия).

3.3.n.1.2. Ограничения на совместимость. Необходимо рассматривать несколько аспектов совместимости: исходный язык, машинный язык, форматы данных и сообщений, форматы отчетов, форматы листингов и т.п.

Специально должна оговариваться совместимость со следующими программными изделиями: изделиями-предшественниками (т.е. такими, которые пользователь может заменить новым изделием; если число функций при такой замене уменьшается, то следует привести обоснование этому); изделиями-компаньонами (т.е. относящимися к той же группе средств и являющимися альтернативой); подобными изделиями (т.е. выполняющих похожие функции в других программных изделиях); конкурирующими изделиями (других организаций).

3.3.n.1.3. Программные ограничения. Описываются программное окружение разрабатываемого ПО, включая указание средств для его загрузки и запуска. Также отмечаются все действующие программные ограничения, например использование вычислений с удвоенной точностью для некоторых функций.

3.3.n.1.4. Аппаратные ограничения. Приводится перечень устройств, необходимых для работы ПО (с указанием минимальной, оптимальной и максимальной конфигурации). Указываются все действующие ограничения на оборудование, например, физические характеристики терминала или требование запрещения использования звукового сигнального устройства.

3.3.n.2. Внешние характеристики.

3.3.n.2.1. Результаты. Описываются все выходные данные ПО с точки зрения их функционального содержания и назначения (например, файлы, сообщения, программно устанавливаемые сигналы и прерывания). При этом должны быть рассмотрены все возможные в системе носители и средства отображения информации. Указываются тип, структура, формат, объем, расположение и диапазон изменения. Для всех выходных данных, читаемых людьми (сообщения и отчеты) должны быть приведены образцы.

3.3.n.2.2. Процессы обработки. Описываются операции, выполняемые ПО в целом или функциональными модулями, рассматриваемыми как черный ящик. Если обсуждение идет на уровне модулей или этапов разработки, указываются также модули или этапы, требуемые для получения определенной выходной информации. Точно определяются все возможные ошибки, потенциальные условия их возникновения и способы рестарта и восстановления. Подраздел должен описывать инициацию, преобразование данных, все варианты завершения работы (нормального и аварийного).

3.3.n.2.3. Входы. Описание подобно п. 3.3.2.1

3.3.n.3. Эргономические характеристики.

Примечание. Этот раздел описывает свойства, обеспечивающие надежность, комфорт и продуктивность работы пользователей и операторов, а также вопросы безопасности, секретности, восстанавливаемости после сбоев, мобильности ПО. Остановимся более подробно на двух подразделах: Надежность и Рабочие характеристики.

В разделе Надежность (это свойство программы понимается здесь как способность к восстановлению нормальной работы при ошибках и сбоях в работе оборудования) рассматриваются следующие вопросы: защита данных пользователя; степень защиты программ от ошибок, возникающих в других частях системы (например, работающих одновременно с данной программой в другой области памяти). Следует рассмотреть, как могут повлиять на работу предлагаемых программных средств такие ошибки, учитывая следующие моменты: распределение ресурсов памяти (указать, если предусмотрено обеспечение изоляции отводимых областей памяти); связь программ через общие аппаратные ресурсы.

Раздел “Рабочие характеристики” описывает основные параметры или принципы, по которым должна оцениваться эффективность работы программы, по возможности в количественном виде с указанием возможных допусков. Все параметры должны быть измеряемыми, к их числу могут относиться быстродействие, пропускная способность, скорость передачи данных, расход машинных ресурсов, время реакции (или задержки) и т.д.

3.3.n.4. Внутренние характеристики (этот раздел полностью расширяется в документе “Внутренняя Спецификация”, однако

частично может быть заполнен с целью полного описания соответствующих внешних свойств).

3.4. Внутренние ограничения (здесь речь идет о тех свойствах, которые пользователю логично ожидать, но которые по тем или иным причинам будут исключены из программного изделия или потенциально оставлены на усмотрение разработчика: например, неполная взаимозаменяемость носителей, отсутствие поддержки каких-либо возможностей оборудования, и т.п.).

3. Используемые материалы (в т.ч. справочные)

4. Передача заказчику и ввод в действие

Руководство пользователя

Под документацией пользователя понимается документация, которая обеспечивает конечного пользователя информацией по установке и эксплуатации программного пакета. Под информацией на упаковке понимают информацию, воспроизводимую на внешней упаковке программного пакета. Ее целью является предоставление потенциальным покупателям первичных сведений о программном пакете.

Пользовательская документация программного средства объясняет пользователям, как они должны действовать, чтобы применить данную программу. Она необходима, если программа предполагает какое-либо взаимодействие с пользователями. К такой документации относятся документы, которыми руководствуется пользователь при установке программы с соответствующей настройкой на среду применения, при применении программы для решения своих задач и при управлении программой (например, когда данное программное средство взаимодействует с другими системами). Эти документы частично затрагивают вопросы сопровождения программного средства, но не касаются вопросов, связанных с модификацией программ.

В связи с этим следует различать две категории пользователей: ординарных пользователей программы и администраторов. Ординарный пользователь программы (end-user) использует программу для решения своих задач (в своей предметной области).

Это может быть инженер, проектирующий техническое устройство, или кассир, продающий железнодорожные билеты с помощью данной программы. Он может и не знать многих деталей работы компьютера или принципов программирования. Администратор программны (system administrator) управляет использованием программы ординарными пользователями и осуществляет сопровождение программного средства, не связанное с модификацией программ. Например, он может регулировать права доступа к программе между ординарными пользователями, поддерживать связь с поставщиками программы или выполнять определенные действия, чтобы поддерживать программу в рабочем состоянии, если оно включено как часть в другую систему.

Состав пользовательской документации зависит от аудиторий пользователей, на которые оно ориентировано, и от режима использования документов. Под аудиторией здесь понимается контингент пользователей, у которого есть необходимость в определенной пользовательской документации. Удачный пользовательский документ существенно зависит от точного определения аудитории, для которой он предназначен. Пользовательская документация должна содержать информацию, необходимую для каждой аудитории. Под режимом использования документа понимается способ, определяющий, каким образом используется этот документ. Обычно пользователю достаточно больших программных систем требуются либо документы для изучения программного средства (использование в виде инструкции), либо для уточнения некоторой информации (использование в виде справочника).

Можно считать типичным следующий состав пользовательской документации для достаточно больших программных средств:

- Общее функциональное описание программного средства. Дает краткую характеристику функциональных возможностей программного средства. Предназначено для пользователей, которые должны решить, насколько необходимо им данное программного средства.
- Руководство по инсталляции программного средства. Предназначено для системных администраторов. Он должен детально предписывать, как устанавливать системы в конкретной среде. Он должен содержать описание машинно-считываемого носителя, на котором поставляется программное средство, файлы,

представляющие программное средство, и требования к минимальной конфигурации аппаратуры.

- Инструкция по применению программного средства. Предназначена для ординарных пользователей. Содержит необходимую информацию по применению программного средства, организованную в форме удобной для ее изучения.
- Справочник по применению программного средства. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению программного средства, организованную в форме удобной для избирательного поиска отдельных деталей.
- Руководство по управлению программным средством. Предназначено для системных администраторов. Оно должно описывать сообщения, генерируемые, когда программные средства взаимодействует с другими системами, и как реагировать на эти сообщения. Кроме того, если программное средство использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

Разработка пользовательской документации начинается сразу после создания внешнего описания. Качество этой документации может существенно определять успех программы. Она должна быть достаточно проста и удобна для пользователя (в противном случае это программное средство, вообще, не стоило создавать). Поэтому, хотя черновые варианты (наброски) пользовательских документов создаются основными разработчиками программного средства, к созданию их окончательных вариантов часто привлекаются профессиональные технические писатели. Кроме того, для обеспечения качества пользовательской документации разработан ряд стандартов, в которых предписывается порядок разработки этой документации, формулируются требования к каждому виду пользовательских документов и определяются их структура и содержание.

Руководство программиста

Документация по сопровождению программного средства (system documentation) описывает программное средство с точки зрения ее разработки. Эта документация необходима, если

программное средство предполагает изучение того, как оно устроена (сконструирована), и модернизацию его программ.

Как уже отмечалось, сопровождение — это продолжающаяся разработка. Поэтому в случае необходимости модернизации программного средства к этой работе привлекается специальная команда разработчиков-сопроводителей. Этой команде придется иметь дело с такой же документацией, которая определяла деятельность команды первоначальных (основных) разработчиков программного средства — с той лишь разницей, что эта документация для команды разработчиков-сопроводителей будет, как правило, чужой (она создавалась другой командой). Команда разработчиков-сопроводителей должна будет изучать эту документацию, чтобы понять строение и процесс разработки модернизируемого программного средства, и внести в эту документацию необходимые изменения, повторяя в значительной степени технологические процессы, с помощью которых создавалось первоначальное программное средство.

Документация по сопровождению программного средства можно разбить на две группы:

1. документация, определяющая строение программ и структур данных ПС и технологию их разработки;
2. документацию, помогающую вносить изменения в программное средство.

Документация первой группы содержит итоговые документы каждого технологического этапа разработки программного средства. Она включает следующие документы:

- Внешнее описание программного средства (Requirements document).
- Описание архитектуры программного средства (description of the system architecture), включая внешнюю спецификацию каждой ее программы.
- Для каждой программы программного средства — описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля.
- Для каждого модуля — его спецификация и описание его строения (design description). Тексты модулей на выбранном языке программирования (program source code listings). Документы установления достоверности программного средства (validation

documents), описывающие, как устанавливалась достоверность каждой программы программного средства и как информация об установлении достоверности связывалась с требованиями к программному средству.

Документы установления достоверности включают прежде всего документацию по тестированию (схема тестирования и описание комплекта тестов), но могут включать и результаты других видов проверки программного средства, например, доказательства свойств программ.

Документация второй группы содержит:

- Руководство по сопровождению программного средства (system maintenance guide), которое описывает известные проблемы вместе с программным средством, описывает, какие части системы являются аппаратно- и программно-зависимыми, и как развитие программного средства принято в расчет в его строении (конструкции).
- Общая проблема сопровождения программного средства — обеспечить, чтобы все его представления шли в ногу (оставались согласованными), когда программное средство изменяется. Чтобы этому помочь, связи и зависимости между документами и их частями должны быть зафиксированы в базе данных управления конфигурацией.

Содержание отчета

1. Название, цель, задание лабораторной работы.
2. Выполненное задание
3. Ответы на контрольные вопросы.

Рекомендуемая литература

Основные источники:

1. Федорова Г.Н. Разработка программных модулей программного обеспечения для компьютерных систем: учебник. Среднее профессиональное образование, профессиональная подготовка / Г.Н Федорова. – М.: Академия, 2016. – 336 с.
2. Тузовский А.Ф. Объектно-ориентированное программирование. - М.: Юрайт, 2017.-206 с.
3. Семакова А. Введение в разработку приложений для смартфонов на ОС Android / А. Семакова. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 103 с.: ил.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429181.
4. Седжвик Р. Алгоритмы на C++ / Р. Седжвик. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 1773 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page =book&id=429164.
5. Смирнов А.А. Прикладное программное обеспечение: учебное пособие / А.А. Смирнов. - Москва; Берлин: Директ-Медиа, 2017. - 358 с.: ил., табл. - Библиогр. в кн. - ISBN 978-5-4475-8780-2; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=457616.

Дополнительные источники:

1. Ашарина И.В., Крупская Ж.Ф. Язык C++ и объектно-ориентированное программирование в C++. Лабораторный практикум. - М.: Горячая линия-Телеком, 2016. -232 с.
2. Белоцерковская И.Е. Алгоритмизация. Введение в язык программирования C++ / И.Е. Белоцерковская, Н.В. Галина, Л.Ю. Катаева. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 197 с. : ил.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428935.
3. Заика А.А. Основы разработки для платформы 1С: Предприятие 8.2 в режиме "Управляемое приложение" / А.А. Заика. - 2-е изд., испр. - Москва: Национальный Открытый

Университет «ИНТУИТ», 2016. - 254 с. : ил.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429115.
4. Заика А.А. Основы разработки прикладных решений для 1С: Предприятие 8.1 / А.А. Заика. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 208 с.: ил.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429116.
5. Конова Е.А., Поллак Г.А. Алгоритмы и программы. Язык С++: Учебное пособие - СПб.: Лань, 2017. -384 с.
6. Литвиненко В.А. Программирование на С++ задач на графах: учебное пособие / В.А. Литвиненко; Министерство образования и науки РФ, Южный федеральный университет, Инженерно-технологическая академия. - Таганрог: Издательство Южного федерального университета, 2016. - 83 с.: схем., ил. - Библиогр. в кн. - ISBN 978-5-9275-2311-5; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=493250.
7. Немцова Т.И., Голова С.Ю., Терентьев А.И. Программирование на языке высокого уровня. Программирование на языке С. – М.: ФОРУМ: ИНФРА-М, 2016.-512 с.
8. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft. NET Framework 4.5 на языке C#. - СПб.: Питер, 2017. -896 с.
9. Савельев А.О. Введение в облачные решения Microsoft / А.О. Савельев. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 231 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429155.
10. Спиридонов О.В. Создание электронных интерактивных мультимедийных книг и учебников в iBooks Author / О.В. Спиридонов. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 629 с.: ил.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428992.
11. Технологии разработки Internet-приложений: лабораторный практикум / авт.-сост. Е.В. Крахоткина; Министерство образования и науки Российской Федерации, Федеральное государственное автономное образовательное учреждение

высшего профессионального образования «Северо-Кавказский федеральный университет». - Ставрополь: СКФУ, 2016. - 102 с.: ил.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=459285.
12. Хвощев С. Основы программирования в Delphi для ОС Android: лекции / С. Хвощев. - 2-е изд., исправ. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 86 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428830.
13. Языки программирования: лабораторный практикум / сост. Е.А. Малиновская, Р.А. Рыскаленко; Министерство образования и науки РФ, Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет». - Ставрополь: СКФУ, 2016. - Ч. 1. - 103 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=467412.

Периодические издания:

1. Компоненты и технологии. ООО Издательство «Файнстрит»;
2. Проблемы информатики. Издательство «Федеральное государственное бюджетное учреждение науки Институт вычислительной математики и математической геофизики Сибирского отделения Российской академии наук»;
3. Проблемы информационной безопасности. Компьютерные системы. Издательство «Федеральное государственное автономное образовательное учреждение высшего образования Санкт-Петербургский политехнический университет Петра Великого»;
4. Linux Format: главное в мире Linux / ред. К. Степанов - Санкт-Петербург: Мезон.Ру; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page= book&id=238521;
5. Системный администратор: ежемесячный журнал / изд. ООО «Синдикат 13»; гл. ред. Г. Положевец - Москва: Синдикат 13, - ISSN 1813-5579; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=430336;
6. Информационно-управляющие системы: научный журнал / гл. ред. М.Б. Сергеев; изд. Санкт-Петербургский государственный университет аэрокосмического приборостроения; учред. ООО «Информационно-

управляющие системы» - Санкт-Петербург: Санкт-Петербургский государственный университет аэрокосмического приборостроения - ISSN 1684-8853; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=494277;
7. Прикладная информатика: научно-практический журнал / гл. ред. А.А. Емельянов - Москва: Университет «Синергия» - ISSN 1993-8314; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=495388;
8. Прикладная информатика: Университет «Синергия»;
9. Компоненты и технологии: Медиа КиТ.

Интернет-ресурсы:

1. Федеральный центр информационно-образовательных услуг. Режим доступа: [http://fcior.edu.ru/ 14.04.2020].
2. Учебники по программированию. Режим доступа:[http://programm.ws/index.php/ 14.04.2020].
3. Федеральные образовательные ресурсы. Режим доступа: [http://www.edu.ru/ 14.04.2020].
4. 101 LINQ Samples in C# Режим доступа: [http://code.msdn.microsoft.com> 101 LINQ Samples-3fb9811b 14.04.2020].
5. Библиотека учебных курсов Microsoft. Режим доступа: [http://msdn.microsoft.com/ru-ru/gg638594 14.04.2020].
6. Единая система программной документации. Режим доступа: [http://prog-cpp.ru/espd/ 14.04.2020].
7. Страуструп Б. Язык программирования C++ для профессионалов. Режим доступа: [http://old.intuit.ru/department/pl/cpp2/, свободный 14.04.2020].
8. CIT-Forum: Центр информационных технологий: материалы сайта. Режим доступа: [http://citforum.ru/, свободный 14.04.2020].
9. CodeNet - все для программиста. Режим доступа: [http://www.codenet.ru/ 14.04.2020].