**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**ФГБОУ ВО**

**«БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**Кафедра «Информатика и программное обеспечение»**

**Расчетно-графическая работа**

**По дисциплине «Программирование приложений Windows»**

Направление подготовки «Программная инженерия»

на тему:

**Работа с окнами, пользовательским интерфейсом**

Вариант №39

Выполнил студ. гр. 18-ПРИ:

Кургуз М. А.

Проверил:

к. т. н., доц. Белов Е. А.

Брянск 2021

**ОГЛАВЛЕНИЕ**

1. Задания ................................................................................................................3
2. Теоретический материал .................................................................................. 4 2.1. Вывод графики в окно ............................................................................... 4 2.2. Стили окон .................................................................................................. 6
3. Описание программ .......................................................................................... 7 3.1. Задание №1.................................................................................................. 7 3.1. Задание №2 ................................................................................................. 8
4. Контрольные примеры работы программ ..................................................... 10 4.1. Задание №1................................................................................................ 10 4.2. Задание №2 ............................................................................................... 10

Список литературы .............................................................................................. 12

1. **ЗАДАНИЯ**

* Задание №1. Рабочую область при любых изменениях размеров окна приложения полностью занимает многострочный редактор с полосами просмотра и фокусом ввода.
* Задание №2. На месте нажатия правой клавиши мыши отобразить диалоговую панель с группой зависимых переключателей выбора одного из нескольких размеров шрифта, группой независимых переключателей для выбора стиля шрифта и кнопками "Да" и "Отмена". После нажатия кнопки "Да" в рабочей области окна приложения выбранными атрибутами шрифта отобразить текст "Пробный вывод".

1. **ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ**

**2.1. Вывод графики в окно**

Вывод графики в окна обладает рядом особенностей.

* Нельзя пользоваться функциями вывода библиотеки компилятора, так как они приспособлены для вывода в одно и единственное окно. В операционной системе Windows приложения выводят одновременно в различные окна. Система сама решает все проблемы, связанные с возможным перекрытием или перемещением этих окон. С этим обстоятельством связано то, что в окно стремятся выводить в одном месте приложения – при обработке сообщения WM\_PAINT. Приложение описывают таким образом, чтобы при поступлении сообщения WM\_PAINT функция окна могла перерисовать все окно или любую его заданную часть.
* Интерфейс графических устройств (Graphics Device Interface – GDI) системы открывает доступ к большому числу функций вывода. Приложения, обращаясь к функциям GDI, работают не с физическими устройствами вывода, а с логическими, т. е. описание вызова функций не зависит от физического способа отображения. Функция GDI передает указания о выводе драйверу устройства вывода. Драйвер работает непосредственно с физическим устройством и при управлении выводом учитывает его ограниченные возможности и аппаратные особенности. Благодаря этому приложения способны работать с любым устройством вывода, драйвер которого установлен в системе. Однако не все физические устройства вывода способны поддерживать те режимы, в которых работает видеомонитор, поэтому, описывая последовательность операций вывода, например на принтер, следует учитывать ограниченные возможности установленного принтера.
* Параметры вывода устанавливают в контексте отображения с помощью функций GDI. Контекст отображения – это структура данных, которая содержит характеристики устройства вывода и указатели на выбранные инструменты рисования. Функции GDI используют только выбранные в контекст отображения параметры и инструменты рисования, например для рисования линии некоторой толщины в контекст отображения, приложение должно выбрать перо этой толщины.
* Дескриптор контекста отображения служит первым аргументом вызова всех функций, связанных с выводом в окно.

GDI, с точки зрения программиста – это контекст отображения.

Контекст отображения можно сравнить с листом бумаги, на котором приложение рисует, а также инструментом для рисования. Инструменты для рисования – это перья, кисти, а также шрифты и целые графические изображения, с помощью которых приложение создаст необходимый рисунок.

Кроме контекста отображения и инструмента для рисования, приложению доступны десятки функций программного интерфейса GDI.

Контекст отображения является структурой данных, описывающих устройство отображения. В этой структуре хранятся различные характеристики устройства и набор инструментов для рисования, выбранные по умолчанию. Приложение может выбрать в контекст отображения различные инструменты (например, перья различной толщины и цвета). Поэтому, если необходимо нарисовать линию красного цвета, то перед выполнением операции рисования, необходимо выбрать в контекст отображения соответствующее перо.

Функции рисования не имеют параметров, указывающих цвет и толщину линий. Такие параметры хранятся в контексте отображения.

Приложение может создать контекст отображения не только для экрана монитора, но и для любого другою графического устройства, например, для принтера. В этом случае оно может рисовать на принтере различные изображения, используя те же функции, что и при рисовании на экране. Можно создать контекст отображения для метафайла. Метафайл – это обычный файл на диске или в памяти, в котором хранятся последовательности команд интерфейса GDI. Приложение может выполнить вывод графической информации в метафайл, как в обычное устройство вывода, а затем «проиграть» метафайл на реальном устройстве вывода.

**2.2. Стили окон**

По совокупности свойств различают перекрывающиеся (overlapped), временные (pop-up) и дочерние (child) окна.

Перекрывающиеся окна чаще используют в качестве окон приложения. Они всегда имеют заголовок, рамку и рабочую область окна, могут иметь системное меню, кнопки восстановления размеров, закрытия и сворачивания окна в пиктограмму, горизонтальную и вертикальную полосы просмотра, меню, панель инструментов и строку состояния. Базовый стиль таких окон описан константой WS\_OVERLAPPED. Чаще используемый стиль окон WS\_OVERLAPPEDWINDOW в дополнение к базовому указывает, что окно имеет системное меню, кнопки восстановления размеров, закрытия и сворачивания окна.

Перекрывающееся окно может принадлежать другому окну. Если окно-владелец сворачивается в пиктограмму, то подчиненные ему окна становятся невидимыми. При уничтожении окна автоматически уничтожаются подчиненные ему окна. Подчиненные окна всегда располагаются над поверхностью окна владельца, загораживая его.

Временные окна обычно используют для вывода сообщений пользователю и остаются на экране непродолжительное время. Базовый стиль временного окна описан константой WS\_POPUP. Такое окно по умолчанию не имеет заголовка. Чаще временное окно описывают константой WS\_POPUPWINDOW. Для добавления к временному окну системного меню и заголовка стиль WS\_POPUPWINDOW комбинируют со стилем WS\_CAPTION. Во всем остальном временные окна – это специальный вид перекрывающихся окон.

Дочерние окна используют для создания элементов управления. Определяемые системой классы элементов управления представляют собой дочерние окна. Базовый стиль дочерних окон описан константой WS\_CHILD. Этот стиль полностью совпадает со стилем WS\_CHILDWINDOW. Дочерние окна не имеют кнопок минимизации и максимального увеличения размера, но всегда имеют окно-родителя. Они «прилипают» к поверхности родителя, перемещаются с ним и не могут выйти за пределы родительского окна.

1. **ОПИСАНИЕ ПРОГРАММ**
   1. **Задание №1**

На листинге 1 представлено создание окна с полосами прокрутки

Листинг 1

|  |
| --- |
| hWnd = CreateWindow(ClassName, AppTitle, WS\_OVERLAPPEDWINDOW | WS\_VSCROLL | WS\_HSCROLL, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, NULL, NULL, hInst, NULL); |
| if (!hWnd) |
| { |
| MessageBox(NULL, |
| "Create: error", AppTitle, MB\_OK | MB\_ICONSTOP); |
| return FALSE; |
| } |

На листинге 2 показана обработка нажатия на символьную клавишу, а также позиционирование каретки, которая указывает в каком месте находится фокус ввода.

Листинг 2

|  |
| --- |
| case WM\_CHAR: |
| { |
|  |
| // проверяем, если текущее место достаточно близко |
| // к концу буфера, что может привести к переполнению |
| // его. Если это так, то добавляем нуль и показываем |
| // содержимое. |
|  |
| if (cch > BUFSIZE - 5) |
| { |
| pchInputBuf[cch] = 0x00; |
| SendMessage(hWnd, WM\_PAINT, 0, 0); |
| } |
| switch (wParam) |
| { |
|  |
|  |
| case 0x09: // табуляция |
|  |
| // Преобразование табуляции в четыре последовательных пробела. |
|  |
| for (i = 0; i < 4; i++) |
| SendMessage(hWnd, WM\_CHAR, 0x20, 0); |
| return 0; |
|  |
| case 0x0D: // возврат каретки |
|  |
| // Запись перевода каретки и позиции |
| // каретки в начале новой строки. |
|  |
| pchInputBuf[cch++] = 0x0D; |
| nCaretPosX = 0; |
| nCaretPosY += 1; |
| break; |
|  |
| default: // отображаемый символ |
|  |
| ch = (TCHAR)wParam; |
| HideCaret(hWnd); |
|  |
| // Извлечение данных о ширине |
| // символа и вывод символа. |
|  |
| hdc = GetDC(hWnd); |
| GetCharWidth32(hdc, (UINT)wParam, (UINT)wParam, |
| &nCharWidth); |
| TextOut(hdc, nCaretPosX, nCaretPosY \* dwCharY, |
| &ch, 1); |
| ReleaseDC(hWnd, hdc); |
|  |
| // Сохранение символа в буфере. |
|  |
| pchInputBuf[cch++] = ch; |
|  |
| // Вычисление новой горизонтальной позиции каретки. |
| // Если позиция превышает максимум, вставьте перевод |
| // каретки, и переместите каретку в начало |
| // следующей строки. |
|  |
| nCaretPosX += nCharWidth; |
| if ((DWORD)nCaretPosX > dwLineLen) |
| { |
| nCaretPosX = 0; |
| pchInputBuf[cch++] = 0x0D; |
| ++nCaretPosY; |
| } |
| nCurChar = cch; |
| ShowCaret(hWnd); |
| break; |
| } |
| SetCaretPos(nCaretPosX, nCaretPosY \* dwCharY); |
| }break; |

Для установки позиции каретки используется функция SetCaretPos(). Для ее скрывания HideCaret(), а для отображения каретки ShowCaret().

На листинге 3 показана работа с одной из полос прокрутки, в данном листинге – вертикальной.

Листинг 3

|  |
| --- |
| case WM\_VSCROLL: |
| { |
| int count = 0; int temp = 0; |
| for (int i = 0; i < BUFSIZE; i++) |
| { |
| temp++; |
| if (pchInputBuf[i] == '\r') |
| count++; |
| } |
|  |
|  |
| SetScrollRange(hWnd, SB\_VERT, 0, count, FALSE); |
| int lastPos = nScrollPos; |
| switch (wParam) |
| { |
| case SB\_TOP: |
| { |
| nScrollPos = 0; |
| break; |
| } |
| case SB\_BOTTOM: |
| { |
| nScrollPos = count; |
| break; |
| } |
| case SB\_LINEUP: |
| { |
| nScrollPos -= 1; |
|  |
| break; |
| } |
| case SB\_LINEDOWN: |
| { |
| nScrollPos += 1; |
|  |
| break; |
| } |
| case SB\_PAGEUP: |
| { |
| nScrollPos -= dwClientY / cyChar; |
| break; |
| } |
| case SB\_PAGEDOWN: |
| { |
| nScrollPos += dwClientY / cyChar; |
| break; |
| } |
| case SB\_THUMBPOSITION: |
| { |
| nScrollPos = LOWORD(lParam); |
| break; |
| } |
| // Блокируем для того чтобы избежать |
| // мерцания содержимого окна при |
| // перемещении ползунка |
| case SB\_THUMBTRACK: |
| { |
| return 0; |
| } |
| default: |
| break; |
| } |
|  |
| if (nScrollPos > count) nScrollPos = count; |
| if (nScrollPos < 0) nScrollPos = 0; |
|  |
| SetScrollPos(hWnd, SB\_VERT, nScrollPos, TRUE); |
|  |
| InvalidateRect(hWnd, NULL, TRUE); |
|  |
| return 0; |
| }break; |

В коде показано, что количество итераций полосы зависит от количества строк текста. Позиция полосы прокрутки устанавливается с помощью функции SetScrollPos(), а диапазон с помощью SetScrollRange().

* 1. **Задание №2**

На листинге 4 представлена обработка нажатия правой кнопки мыши, при нажатии которой подгружается диалоговое окно из файла ресурсов.

Также здесь представлен обработчик сообщения WM\_PAINT, в котором происходит отрисовка текста с заданными в модальном окне параметрами.

Листинг 4

|  |
| --- |
| case WM\_PAINT: |
| { |
| HDC hDC; |
| PAINTSTRUCT ps; |
| hDC = BeginPaint(hWnd, &ps); |
|  |
| if (show) |
| { |
| HFONT hFont = CreateFontIndirect(&font); |
| SelectObject(hDC, hFont); |
| TextOut(hDC, 100, 100, str, strlen(str)); |
|  |
| DeleteObject(hFont); |
| } |
|  |
| EndPaint(hWnd, &ps); |
|  |
| }; break; |
|  |
| case WM\_RBUTTONDOWN: |
| { |
| DialogBox(hInst, MAKEINTRESOURCE(IDD\_DIALOG1), hWnd, Param); |
| }break; |

На листинге 5 представлена оконная функция диалогового окна, в котором задаются параметры текста, а также обработчики сообщений, которые передаются данной функции

Листинг 5

|  |
| --- |
| INT\_PTR CALLBACK Param(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) |
| { |
| int id = 1001; |
| RECT r = { 0,0,0,0 }; |
| const int N = 100; |
| char buff[N]; |
| bool flag = false; |
|  |
| switch (message) |
| { |
| case WM\_INITDIALOG: |
| { |
| if (font.lfItalic != 0) |
| SendMessage(GetDlgItem(hDlg, IDC\_CHECK2), BM\_SETCHECK, BST\_CHECKED, NULL); |
|  |
| if (font.lfWeight == FW\_EXTRABOLD) |
| SendMessage(GetDlgItem(hDlg, IDC\_CHECK1), BM\_SETCHECK, BST\_CHECKED, NULL); |
|  |
| if(font.lfUnderline != 0) |
| SendMessage(GetDlgItem(hDlg, IDC\_CHECK3), BM\_SETCHECK, BST\_CHECKED, NULL); |
|  |
|  |
| }break; |
| case WM\_COMMAND: |
| { |
| if (LOWORD(wParam) == IDOK) |
| { |
| if (bool checkBold = SendMessage(GetDlgItem(hDlg, IDC\_CHECK1), BM\_GETCHECK, 0, 0)) { font.lfWeight = FW\_EXTRABOLD; flag = true; } |
| else font.lfWeight = FW\_THIN; |
|  |
| if (bool checkItalic = SendMessage(GetDlgItem(hDlg, IDC\_CHECK2), BM\_GETCHECK, 0, 0)) { font.lfItalic = 1; flag = true; } |
| else font.lfItalic = 0; |
|  |
| if (bool checkBold = SendMessage(GetDlgItem(hDlg, IDC\_CHECK3), BM\_GETCHECK, 0, 0)) { font.lfUnderline = 1; flag = true; } |
| else font.lfUnderline = 0; |
|  |
| for (int i = id; i < id+5; i++) |
| { |
| if (bool checkSize = SendMessage(GetDlgItem(hDlg, i), BM\_GETCHECK, 0, 0)) |
| { |
| GetWindowText(GetDlgItem(hDlg, i), buff, N / 10); |
| int size = atoi(buff); |
| font.lfHeight = size; |
| flag = true; |
| break; |
| } |
| else continue; |
|  |
| } |
|  |
| if (flag == false) { MessageBox(NULL, "Выберите характеристику", "Ошибка", MB\_OK); } |
|  |
| else |
| { |
| EndDialog(hDlg, LOWORD(wParam)); |
| show = true; |
| GetClientRect(hwndMain, &r); |
| InvalidateRect(hwndMain, &r, TRUE); |
|  |
| return (INT\_PTR)TRUE; |
| } |
|  |
| } |
| if (LOWORD(wParam) == IDCANCEL) |
| { |
| EndDialog(hDlg, LOWORD(wParam)); |
| return (INT\_PTR)FALSE; |
| } |
| } |
|  |
| default: |
| break; |
| } |
| return (INT\_PTR)FALSE; |
| } |

Здесь функция GetDlgItem() позволяет получить дескриптор элемента управления по его идентификатору.

1. **КОНТРОЛЬНЫЕ ПРИМЕРЫ РАБОТЫ ПРОГРАММ**
   1. **Задание №1**

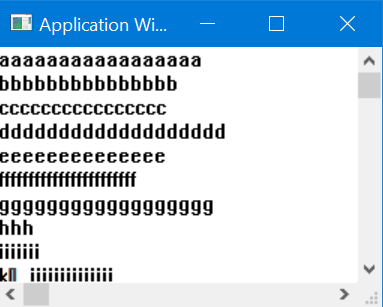
Запуск приложения и ввод текста

Рис. 1. Окно после запуска и ввода текста

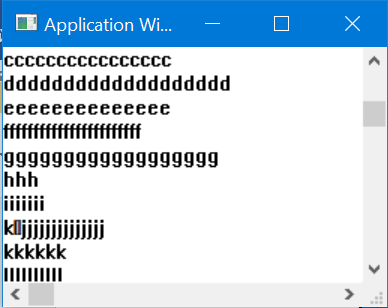


Рис. 2. Окно после нескольких скроллов

* 1. **Задание №2**

На рисунке 3 показано состояние окна, после вызова диалогового окна с параметрами текста.

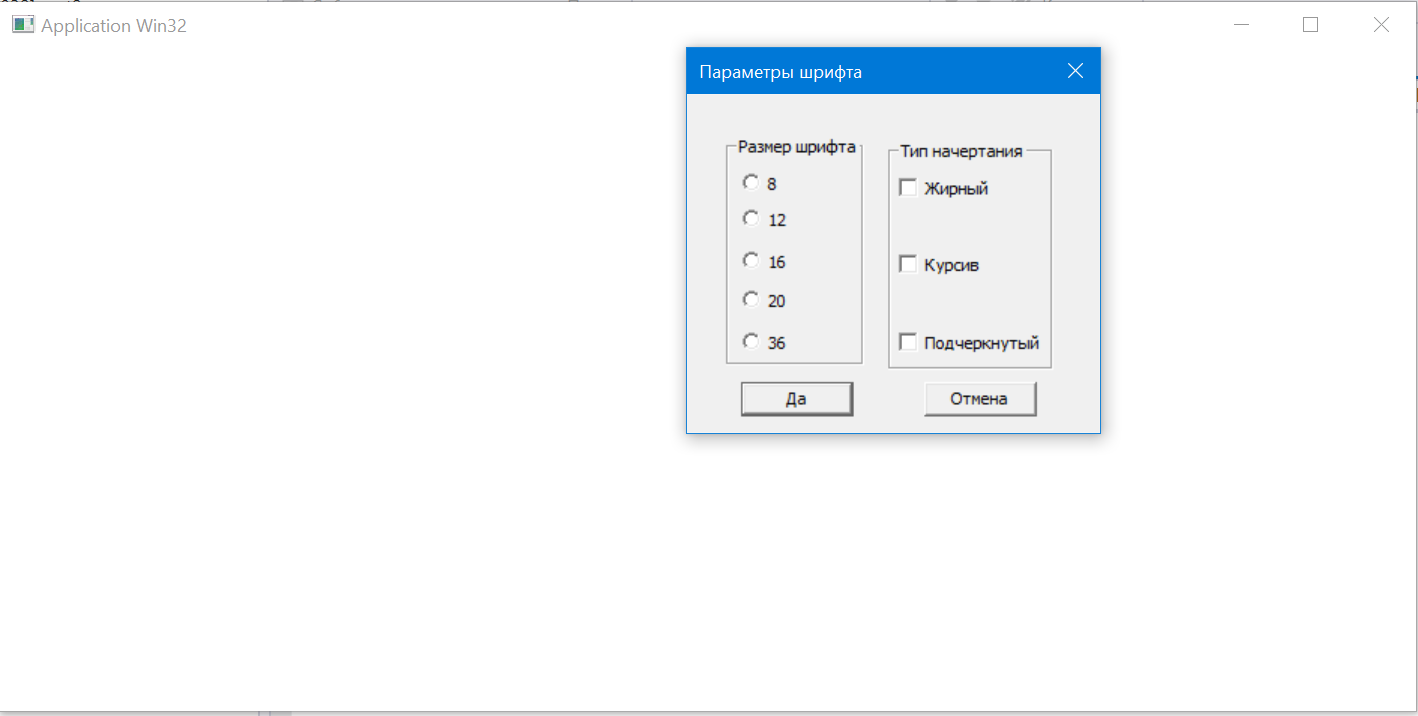
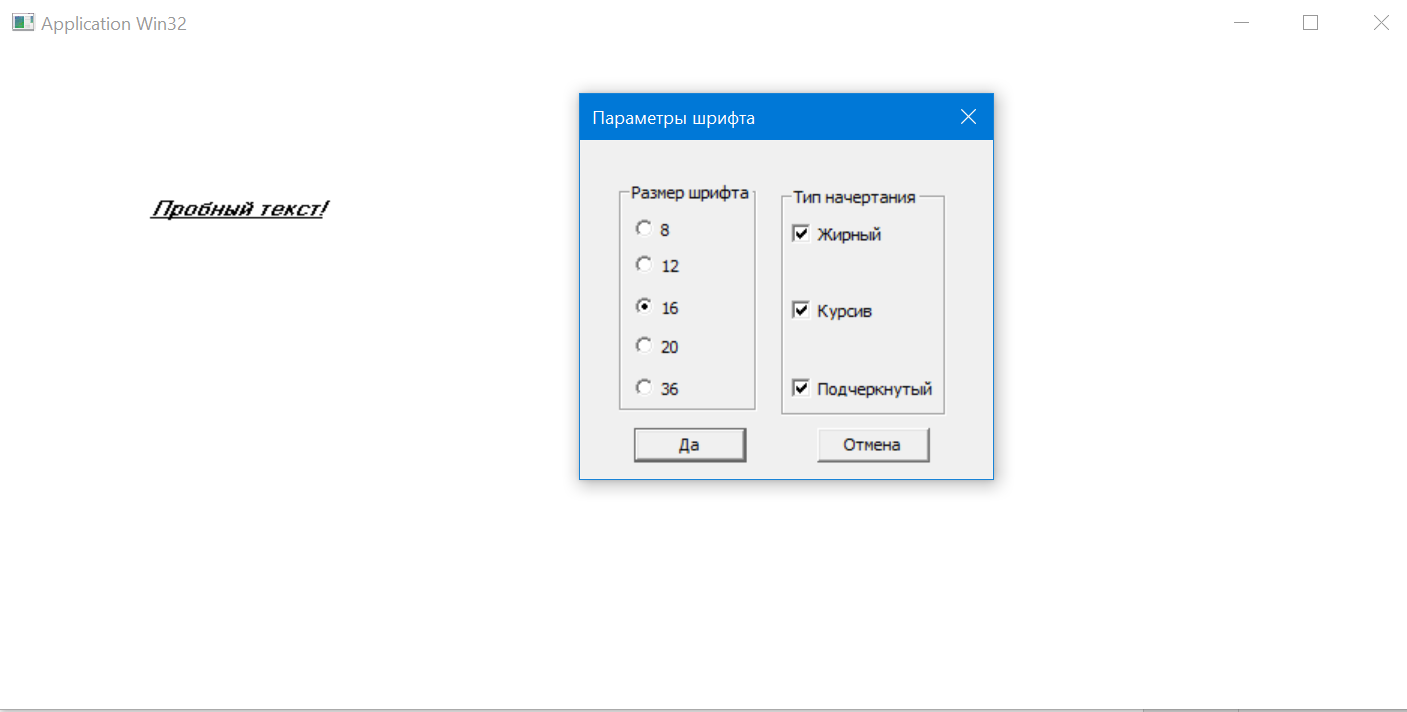
Рисунок 4 показывает результат работы после выбора некоторых параметров и подтверждения кнопкой «Да».

Рис. 4. Текст с установленными параметрами

Рис. 3. Окно с параметрами текста

**СПИСОК ЛИТЕРАТУРЫ**

1. Белов, Е. А. Программирование приложений Windows. API графического интерфейса пользователя: учеб. пособие / Е. А. Белов. – Брянск: БГТУ, 2010. – 350 с.

2. Рихтер, Дж. Windows для профессионалов: создание эффективных Win32-приложений (с CD-ROM) / Дж. Рихтер. – М.: Русская редакция, 2000. – 752 с.

3. Рихтер, Дж. WINDOWS для профессионалов. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии WINDOWS: [перевод] / Дж. Рихтер – 4-е изд. – СПб. [и др.]: Питер, Русская редакция, 2001. – 722 с.

4. Инструменты разработки и языки программирования [Электронный ресурс]. – Режим доступа: http://msdn.microsoft.com/. – Загл. с экрана. – Яз. рус.

5. RSDN [Электронный ресурс]. – Режим доступа: http://www.rsdn.ru/. – Загл. с экрана. – Яз. рус.