Модуль №4

Занятие №3

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Контекстное меню.
3. Акселераторы.
4. Практическая часть.
5. Подведение итогов.
6. Домашнее задание.

1. Повторение пройденного материала

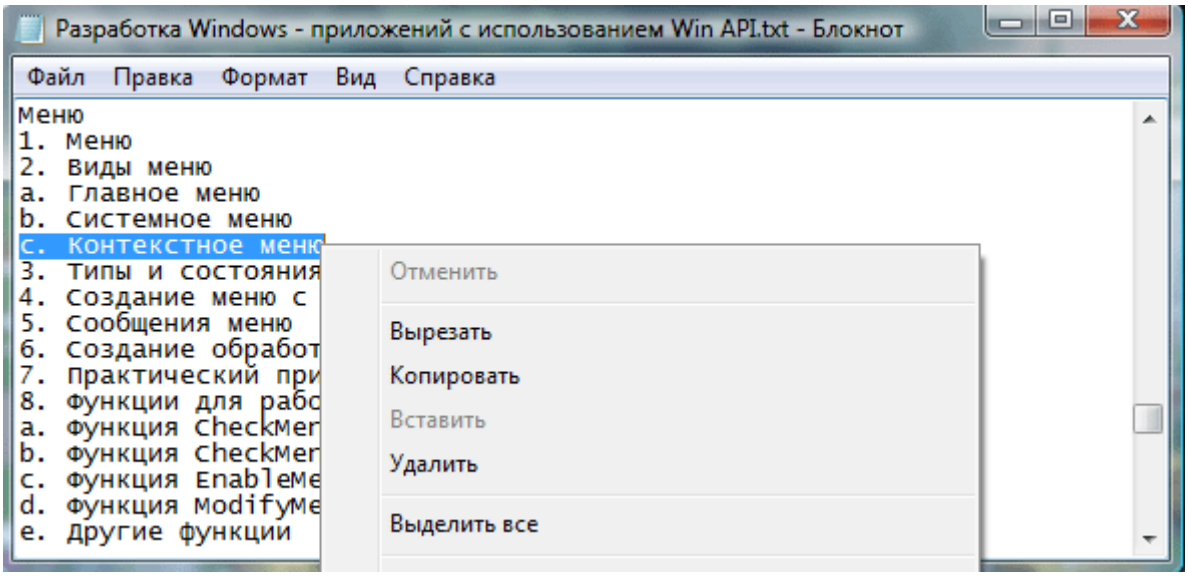
Данное занятие необходимо начать с краткого повторения материала предыдущего занятия. При общении со слушателями можно использовать следующие контрольные вопросы:

1. С помощью какой функции API создается главное меню?
2. С помощью какой функции API создается подменю?
3. Какая функция API позволяет добавить пункт в меню?
4. Какая функция API позволяет вставить пункт в произвольную позицию меню?
5. Как добавлять подменю в меню?
6. Какие существуют способы идентификации пунктов меню?
7. Какая функция API позволяет модифицировать пункт меню?
8. Каким образом можно удалить пункт меню или подменю?
9. Какая функция API служит для уничтожения меню?

10. С помощью какой функции API можно получить строку текста указанного пункта меню?

2. Контекстное меню

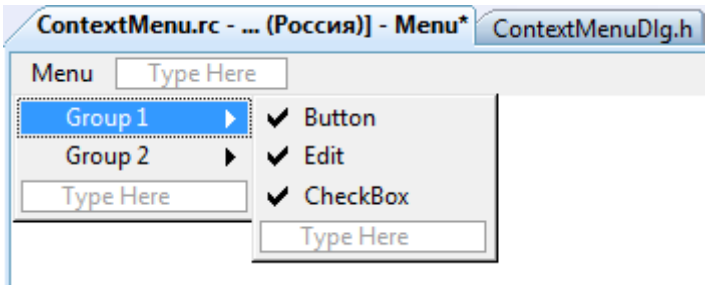
Как отмечалось ранее, **контекстное меню** – это меню, которое появляется в любой части окна приложения при щелчке правой кнопкой мыши. При этом содержание контекстного меню изменяется в зависимости от «**контекста**» — места, где произведен щелчок правой кнопкой мыши. Обычно контекстное меню содержит пункты, которые дублируют команды основного меню, но сгруппированные иначе, чтобы максимально облегчить пользователю работу с приложением.



Акцентировать внимание слушателей на том, что создание контекстного меню выполняется аналогично созданию главного меню приложения. Для этого используется два подхода:

- определение шаблона меню в файле описания ресурсов;
- программное (динамическое) создание меню.

Создавая шаблон контекстного меню с помощью редактора меню, необходимо определить нулевой пункт меню нулевого уровня как подменю, имеющее какое-нибудь условное имя. Этот пункт нигде не будет отображаться. Он необходим только для получения дескриптора подменю с помощью функции API **GetSubMenu**, рассмотренной ранее.



После определения контекстного меню в файле описания ресурсов необходимо его загрузить с помощью функции API **LoadMenu**. Данная функция вернет дескриптор фиктивного меню нулевого уровня, которое не должно отображаться на экране.

Содержанием контекстного меню является нулевой пункт указанного меню, поэтому окончательное значение дескриптора контекстного меню определяется вызовом функции **GetSubMenu**. Полученный дескриптор контекстного меню затем передается функции **TrackPopupMenu**, которая выводит всплывающее контекстное меню на экран.

```
BOOL TrackPopupMenu(
    HMENU hMenu, // дескриптор контекстного меню
    UINT uFlags, // флаги, определяющие позиционирование и другие опции меню
    int x, // горизонтальное расположение контекстного меню в экранных
      // координатах
    int y, // вертикальное расположение контекстного меню в экранных
      // координатах
    int nReserved, // зарезервированное значение; должно быть равно 0
    HWND hWnd, // дескриптор окна, которому принадлежит контекстное меню
    HWND prcRect // параметр игнорируется
);
```

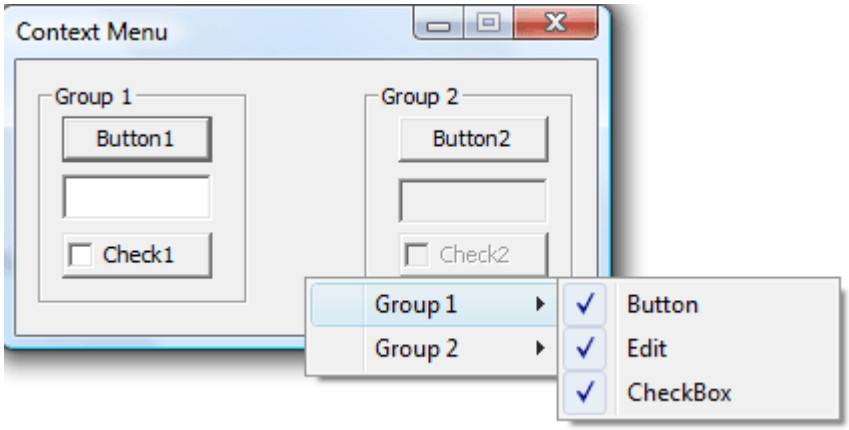
Описанные выше действия иллюстрируются следующим фрагментом кода:

```
// Загрузим меню из ресурсов приложения
HMENU hMenu = LoadMenu(GetModuleHandle(NULL), MAKEINTRESOURCE(IDR_MENU1));
// Получим дескриптор подменю
HMENU hSubMenu = GetSubMenu(hMenu, 0);
// Отобразим контекстное меню, левый верхний угол которого привязывается
// к точке текущего положения курсора мыши
TrackPopupMenu(hSubMenu, TPM_LEFTALIGN, xPos, yPos, 0, hDialog, NULL);
```

Как известно, при щелчке правой кнопкой мыши Windows отправляет оконной процедуре приложения сообщение **WM_RBUTTONDOWN**, содержащее в **LPARAM** клиентские координаты курсора мыши в момент щелчка. Помимо этого сообщения в оконную процедуру приложения приходит сообщение **WM_CONTEXTMENU**, содержащее в **LPARAM** экранные координаты курсора мыши (в младшей части **LPARAM** находится координата **X** положения курсора мыши, а в старшей части - координата **Y**), а в **WPARAM** дескриптор окна, в котором произведен щелчок.

Учитывая, что рассмотренная выше функция отображения контекстного меню **TrackPopupMenu** принимает экранные координаты позиции расположения меню, то было бы удобно предусмотреть в приложении обработчик сообщения **WM_CONTEXTMENU** вместо **WM_RBUTTONDOWN** для организации вызова контекстного меню. В этом случае не нужно выполнять преобразование клиентских координат в экранные координаты.

В качестве примера использования контекстного меню привести слушателям следующее приложение, в котором на форме диалога расположены две группы элементов управления. С помощью контекстного меню происходит управление состоянием элементов управления (разрешение или запрещение элементов управления).




```
                                    // header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include "resource.h"


                                // ContextMenuDlg.h
#pragma once
#include "header.h"

class CContextMenuDlg
{
public:
      CContextMenuDlg(void);
public:
      static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
      static CContextMenuDlg* ptr;
      BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
      void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
      void Cls_OnClose(HWND hwnd);
      void Cls_OnContextMenu(HWND hwnd, HWND hwndContext, UINT xPos,
                             UINT yPos);
      HWND hDialog, hButton1, hButton2, hEdit1, hEdit2, hCheck1, hCheck2;
      HMENU hMenu, hSubMenu, hSubMenu2, hSubMenu3;
      DWORD dwButtonState, dwEditState, dwCheckboxState;
};


                               // ContextMenuDlg.cpp
#include "ContextMenuDlg.h"

CContextMenuDlg* CContextMenuDlg::ptr = NULL;

CContextMenuDlg::CContextMenuDlg(void)
{
      ptr = this;
      dwButtonState = dwEditState = dwCheckboxState = MF_CHECKED;
}
```

```
void CContextMenuDlg::Cls_OnClose(HWND hwnd)
{
	EndDialog(hwnd, 0);
}

BOOL CContextMenuDlg::Cls_OnInitDialog(HWND hwnd, HWND hwndFocus,
                                       LPARAM lParam)
{
	hDialog = hwnd;

	// Получим дескрипторы элементов управления
	hEdit1 = GetDlgItem(hDialog, IDC_EDIT1);
	hEdit2 = GetDlgItem(hDialog, IDC_EDIT2);
	hButton1 = GetDlgItem(hDialog, IDC_BUTTON1);
	hButton2 = GetDlgItem(hDialog, IDC_BUTTON2);
	hCheck1 = GetDlgItem(hDialog, IDC_CHECK1);
	hCheck2 = GetDlgItem(hDialog, IDC_CHECK2);

	// Загрузим меню из ресурсов приложения
	hMenu = LoadMenu(GetModuleHandle(NULL), MAKEINTRESOURCE(IDR_MENU1));

	// Получим дескрипторы подменю
	hSubMenu = GetSubMenu(hMenu, 0);
	hSubMenu2 = GetSubMenu(hSubMenu, 0);
	hSubMenu3 = GetSubMenu(hSubMenu, 1);

	// Установим отметку на первом пункте в радио-группе элементов меню
	CheckMenuRadioItem(hSubMenu3, ID_GROUP2_BUTTON, ID_GROUP2_CHECKBOX,
			ID_GROUP2_BUTTON, MF_BYCOMMAND);
	return TRUE;
}

// Обработчик сообщения WM_COMMAND будет вызван при выборе пункта меню
void CContextMenuDlg::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                    UINT codeNotify)
{
	if(id >= ID_GROUP1_BUTTON && id <= ID_GROUP1_CHECKBOX)
	{
		switch(id)
		{
			case ID_GROUP1_BUTTON:
				// Узнаем текущее состояние пункта меню
				if(dwButtonState == MF_CHECKED)
					EnableWindow(hButton1, FALSE);
				else
					EnableWindow(hButton1, TRUE);
		// Изменим состояние пункта меню на противоположное,
		// т.е. если пункт меню был отмечен, то снимем отметку и наоборот
				dwButtonState ^= MF_CHECKED;
				CheckMenuItem(hSubMenu2, ID_GROUP1_BUTTON,
					MF_BYCOMMAND | dwButtonState);
				break;
			case ID_GROUP1_EDIT:
				if(dwEditState == MF_CHECKED)
					EnableWindow(hEdit1, FALSE);
				else
					EnableWindow(hEdit1, TRUE);
				dwEditState ^= MF_CHECKED;
```

```
                        CheckMenuItem(hSubMenu2, ID_GROUP1_EDIT,
                              MF_BYCOMMAND | dwEditState);
                        break;
                  case ID_GROUP1_CHECKBOX:
                        if(dwCheckboxState == MF_CHECKED)
                              EnableWindow(hCheck1, FALSE);
                        else
                              EnableWindow(hCheck1, TRUE);
                        dwCheckboxState ^= MF_CHECKED;
                        CheckMenuItem(hSubMenu2, ID_GROUP1_CHECKBOX,
                                    MF_BYCOMMAND | dwCheckboxState);
                        break;
            }
      }

      if(id >= ID_GROUP2_BUTTON && id <= ID_GROUP2_CHECKBOX)
      {

      // Установим отметку на выбранном пункте в радио-группе элементов меню
            CheckMenuRadioItem(hSubMenu3, ID_GROUP2_BUTTON,
                              ID_GROUP2_CHECKBOX, id, MF_BYCOMMAND);
            switch(id)
            {
                  case ID_GROUP2_BUTTON:
                              EnableWindow(hButton2, TRUE);
                              EnableWindow(hEdit2, FALSE);
                              EnableWindow(hCheck2, FALSE);
                              break;
                  case ID_GROUP2_EDIT:
                              EnableWindow(hButton2, FALSE);
                              EnableWindow(hEdit2, TRUE);
                              EnableWindow(hCheck2, FALSE);
                              break;
                  case ID_GROUP2_CHECKBOX:
                              EnableWindow(hButton2, FALSE);
                              EnableWindow(hEdit2, FALSE);
                              EnableWindow(hCheck2, TRUE);
                              break;
            }
      }
}

// Обработчик сообщения WM_CONTEXTMENU будет вызван при щелчке правой кнопкой
// мыши в любой области диалогового окна
void CContextMenuDlg::Cls_OnContextMenu(HWND hwnd, HWND hwndContext,
                                        UINT xPos, UINT yPos)
{
      // Отобразим контекстное меню, левый верхний угол которого привязывается
      // к точке текущего положения курсора мыши
      TrackPopupMenu(hSubMenu, TPM_LEFTALIGN, xPos, yPos, 0, hDialog, NULL);
}

BOOL CALLBACK CContextMenuDlg::DlgProc(HWND hwnd, UINT message,
                                       WPARAM wParam, LPARAM lParam)
{

      switch(message)
      {
```

```
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_INITDIALOG, ptr->Cls_OnInitDialog);
            HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
            HANDLE_MSG(hwnd, WM_CONTEXTMENU, ptr->Cls_OnContextMenu);
      }
      return FALSE;
}
                              // ContextMenuDlg.cpp
#include "ContextMenuDlg.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev, LPTSTR lpszCmdLine,
                  int nCmdShow)
{
      CContextMenuDlg dlg;
      return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        CContextMenuDlg::DlgProc);
}
```

Следует дать подробные комментарии к данной программе. В частности, отметить, что контекстное меню приложения содержит два подменю:

- подменю **Group 1** включает в себя **Checkbox**-элементы меню;
- подменю **Group 2** включает в себя **Radio**-элементы меню.

Акцентировать внимание слушателей на том, что в подменю, содержащем **Radio**-элементы, может быть выбран (отмечен) только один пункт меню, в то время как в подменю, содержащем **Checkbox**-элементы, одновременно могут быть отмечены несколько пунктов меню.

Напомнить слушателям, что для установки или снятия отметки предназначена функция API **CheckMenuItem**, рассмотренная ранее.

Функция API **CheckMenuRadioItem** отмечает выбранный пункт меню и снимает отметку со всех других опций меню в указанной группе.

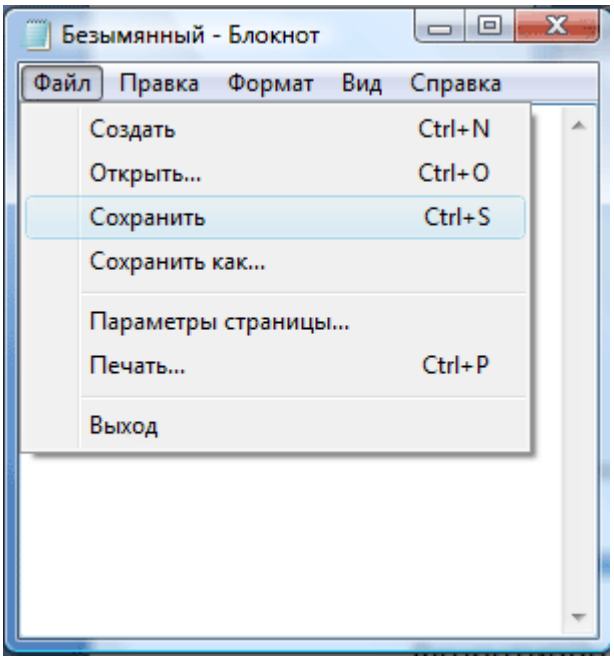
```
BOOL CheckMenuRadioItem(
    HMENU hmenu, // дескриптор меню, которое содержит группу Radio-элементов
    UINT idFirst,// первый пункт меню в группе
    UINT idLast, // последний пункт меню в группе
    UINT idCheck, // пункт меню, который должен быть отмечен
    UINT uFlags // интерпретация второго, третьего и четввертого параметров
);
```

Последний параметр может принимать одно из следующих значений:

- **MF_BYCOMMAND** – в этом случае второй, третий и четвертый параметры должны содержать идентификатор пункта меню;
- **MF_BYPOSITION** – в этом случае второй, третий и четвертый параметры должны содержать относительную позицию пункта меню с отсчетом от нуля.

3. Акселераторы

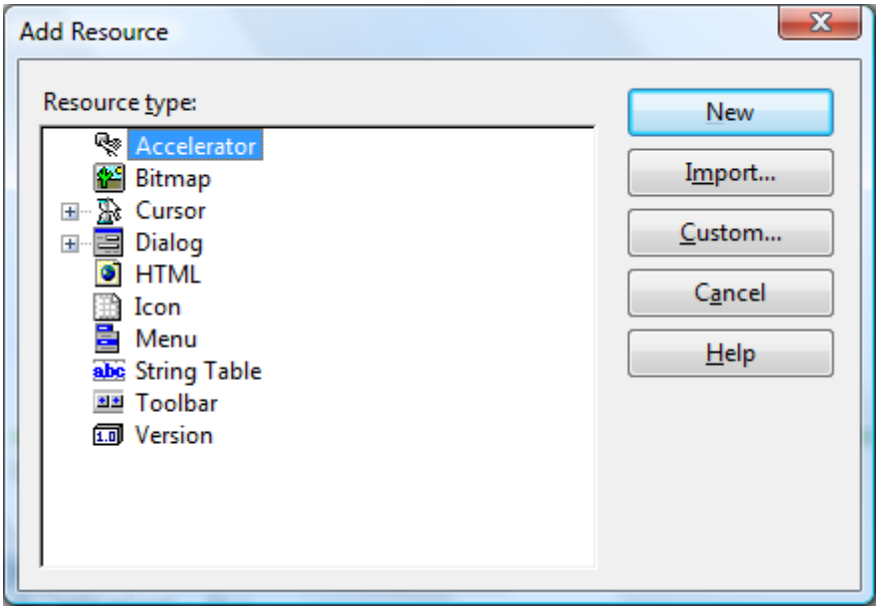
Акцентировать внимание слушателей на клавиатурных комбинациях, находящихся рядом с пунктами меню во многих программах. Обычно эти клавиатурные комбинации (**акселераторы**) дублируют пункты меню, предоставляя альтернативный способ вызова команд меню.



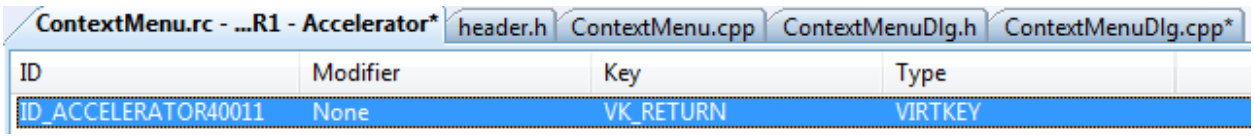
Чтобы добавить в приложение обработку акселераторов, необходимо выполнить следующую последовательность действий:

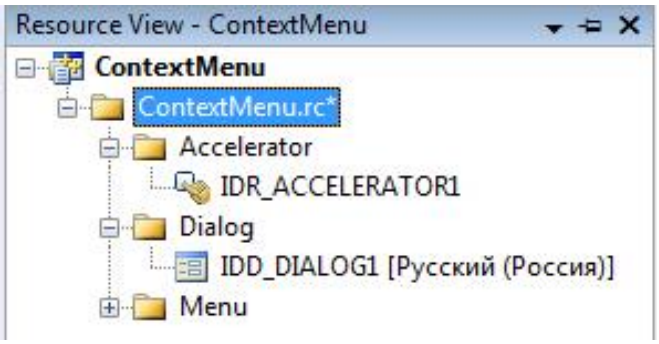
- модифицировать определение ресурса меню, добавив к имени каждого дублируемого пункта информацию о быстрой клавише;
- определить таблицу акселераторов в файле описания ресурсов;
- обеспечить загрузку таблицы акселераторов в память приложения;
- модифицировать цикл обработки сообщений в функции **WinMain**.

Для того чтобы определить таблицу акселераторов в файле описания ресурсов необходимо воспользоваться редактором таблиц акселераторов. Для этого необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource...**



В этом диалоговом окне необходимо выбрать из списка **Accelerator** и нажать кнопку **New**, в результате чего будет открыто окно редактора таблицы акселераторов.





По умолчанию редактор присваивает таблице акселераторов имя **IDR_ACCELERATOR1**. Впоследствии этот идентификатор можно изменить на другой идентификатор, отражающий семантику ресурса.

Поскольку акселераторы ставятся в соответствие командам меню, то они должны иметь те же идентификаторы, что и элементы меню, которым они соответствуют. Данное требование не является обязательным при реализации акселераторов, однако если оно не будет соблюдаться, то идея акселераторов как средства быстрого вызова команд меню потеряет смысл.

ID	Modifier	Key	Type
ID_CREATE	Ctrl	N	VIRTKEY
ID_FIND	Ctrl	F	VIRTKEY
ID_GO_TO	Ctrl	G	VIRTKEY
ID_OPEN	Ctrl	O	VIRTKEY
ID_QUIT	Ctrl	Q	VIRTKEY
ID_REPLACE	Ctrl	H	VIRTKEY
ID_SAVE	Ctrl	S	VIRTKEY
ID_SELECT_ALL	Ctrl	A	VIRTKEY
ID_TIME_DATE	None	VK_F5	VIRTKEY
ID_UNDO	Ctrl	Z	VIRTKEY
IDC_DECREASE	None	VK_SUBTRACT	VIRTKEY
IDC_INCREASE	None	VK_ADD	VIRTKEY

Как видно из вышеприведенного рисунка для 12 команд меню определены акселераторы. В первом столбце таблицы акселераторов указаны идентификаторы акселераторов, совпадающие с идентификаторами пунктов меню. Во втором и третьем столбцах указаны клавиатурные комбинации. Причем клавиша, указанная в третьем столбце определена как виртуальная, о чем свидетельствует ключевое слово **VIRTKEY**

в четвертом столбце (для алфавитно-цифровых клавиш и клавиш символов пунктуации используется тип **ASCII**).

Во время работы приложения для загрузки таблицы акселераторов в память и получения ее дескриптора используется функция API **LoadAccelerators**:

```
HACCEL LoadAccelerators(
    HINSTANCE hInstance, // дескриптор приложения
    LPCTSTR lpTableName // указатель на строку, содержащую имя таблицы
      // акселераторов
);
```

Для обработки акселераторов приложение должно перехватывать сообщения клавиатуры, анализировать их коды и в случае совпадения с кодом, определенным в таблице акселераторов, направлять соответствующее сообщение в оконную процедуру главного окна. С этой целью используется функция API **TranslateAccelerator**:

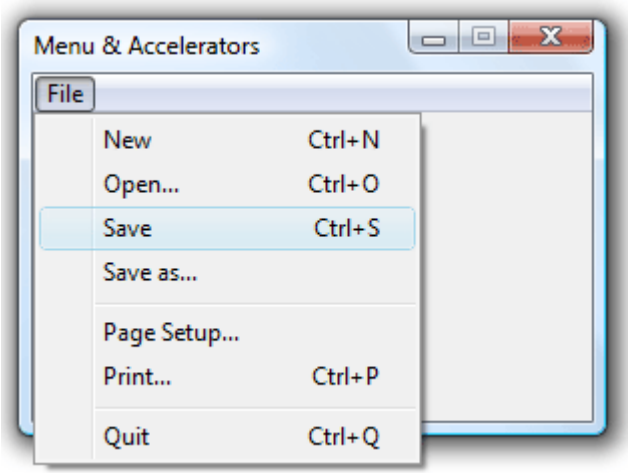
```
int TranslateAccelerator(
    HWND hWnd, // дескриптор окна-получателя сообщений
    HACCEL hAccTable,// дескриптор таблицы акселераторов
    LPMSG lpMsg // указатель на структуру MSG
);
```

Функция **TranslateAccelerator** преобразует сообщение **WM_KEYDOWN** или **WM_SYSKEYDOWN** в сообщение **WM_COMMAND** или **WM_SYSCOMMAND**, если таблица акселераторов содержит соответствующий код виртуальной клавиши (с учетом состояния клавиш <**Ctrl**>, <**Alt**> и <**Shift**>).

Сформированное сообщение, содержащее идентификатор акселератора в младшем слове параметра **wParam**, отправляется непосредственно в оконную процедуру, минуя очередь сообщений. Возврат из функции **TranslateAccelerator** происходит только после того, как оконная процедура обработает посланное сообщение.

Если функция **TranslateAccelerator** возвращает ненулевое значение, это значит, что преобразование комбинации клавиш и обработка отправленного сообщения завершились успешно. В этом случае приложение не должно повторно обрабатывать ту же самую комбинацию клавиш при помощи функций **TranslateMessage** и **DispatchMessage**.

Для демонстрации техники использования акселераторов предлагается рассмотреть со слушателями следующее приложение.




```
                                    // header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include "resource.h"

                                  // AcceleratorsDlg.h
#pragma once
#include "header.h"

class CAcceleratorsDlg
{
public:
      CAcceleratorsDlg(void);
public:
      static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
      static CAcceleratorsDlg* ptr;
      BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
      void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
      void Cls_OnClose(HWND hwnd);
};
```

```
                              // AcceleratorsDlg.cpp
#include "AcceleratorsDlg.h"

CAcceleratorsDlg* CAcceleratorsDlg::ptr = NULL;

CAcceleratorsDlg::CAcceleratorsDlg(void)
{
      ptr = this;
}

void CAcceleratorsDlg::Cls_OnClose(HWND hwnd)
{
      DestroyWindow(hwnd);
      PostQuitMessage(0);
}

void CAcceleratorsDlg::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                     UINT codeNotify)
{
      TCHAR str1[300], str2[50];
      HMENU hMenu = GetMenu(hwnd);
      GetMenuString(hMenu, id, str2, 50, MF_BYCOMMAND);
      if(codeNotify == 1)
            _tcscpy(str1, TEXT("Пункт меню выбран с помощью акселератора\n"));
      else if(codeNotify == 0)
            _tcscpy(str1,
      TEXT("Пункт меню выбран при непосредственном обращении к меню\n"));
      _tcscat(str1, str2);
      MessageBox(hwnd, str1, TEXT("Меню и акселераторы"),
                  MB_OK | MB_ICONINFORMATION);
}


BOOL CALLBACK CAcceleratorsDlg::DlgProc(HWND hwnd, UINT message,
                                        WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
      }
      return FALSE;
}


                              // AcceleratorsDlg.cpp
#include "AcceleratorsDlg.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev, LPTSTR lpszCmdLine,
                     int nCmdShow)
{
      CAcceleratorsDlg dlg;
      MSG msg;
      HWND hDialog = CreateDialog(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                           CAcceleratorsDlg::DlgProc);
      HACCEL hAccel = LoadAccelerators(hInst,
                                       MAKEINTRESOURCE(IDR_ACCELERATOR1));
```

```
	ShowWindow(hDialog, nCmdShow);
	while(GetMessage(&msg, 0, 0, 0))
	{
		if(!TranslateAccelerator(hDialog, hAccel, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}
	return msg.wParam;
}
```

Как видно из вышеприведенного кода можно легко отличить способ выбора пункта меню, если проанализировать четвертый параметр **codeNotify** функции – обработчика сообщения **WM_COMMAND**.

При выборе пункта меню с помощью акселератора в параметре **codeNotify** будет единичное значение, в то время как при непосредственном обращении к меню в этом параметре будет нулевое значение.

4. Практическая часть

1. Модифицировать приложение «Текстовый редактор» (см. Модуль 4 Занятие 1), дополнив его возможностью выбора пунктов меню с помощью акселераторов.
2. Добавить акселераторы в приложение «Счётчик» (см. Модуль 4 Занятие 2 раздел «Практическая часть»).

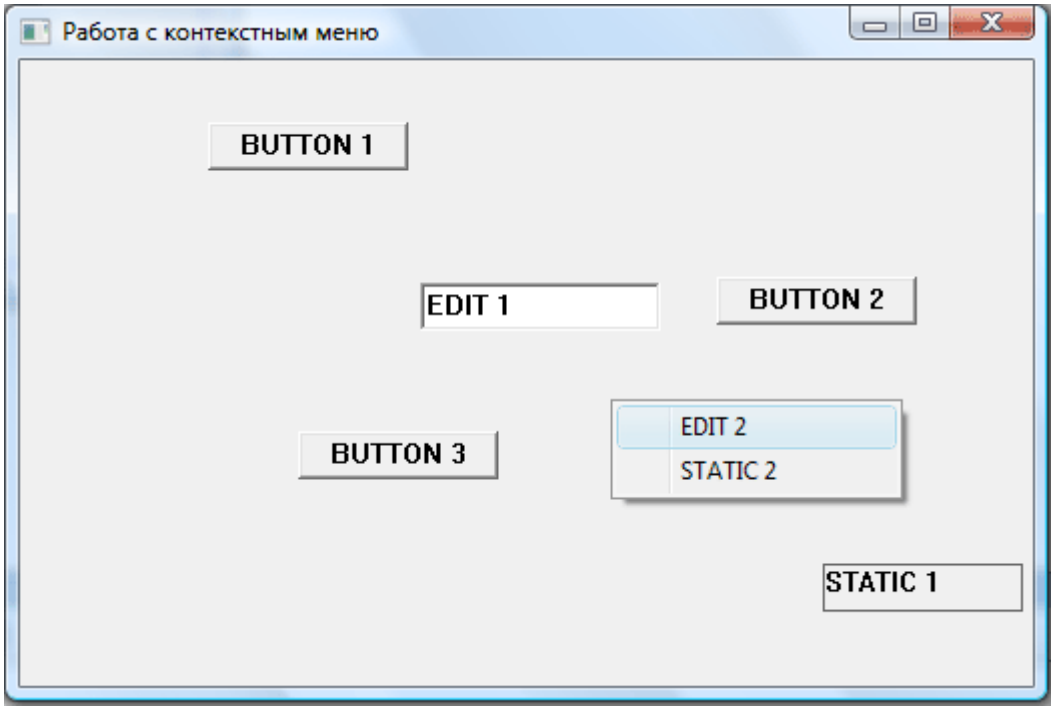
5. Подведение итогов

Подвести общие итоги занятия. Ещё раз подчеркнуть необходимость создания в приложении контекстного меню как средства, максимально упрощающего работу пользователя. Напомнить слушателям суть, назначение и общий принцип работы акселераторов. Отметить, как добавлять акселераторы к проекту и как их загружать в приложение. Ак-

центировать внимание слушателей на наиболее тонких моментах изложенной темы.

6. Домашнее задание

Разработать приложение, работа которого состоит в следующем. По щелчку правой кнопкой мыши на форме диалога выпадает контекстное меню со следующими пунктами: «**Надпись 3**», «**Текст 3**», «**Кнопка 3**». Когда пользователь выбирает пункт меню, то на форме в точке щелчка появляется соответствующий элемент управления (**Static**, **Edit** или **Button**). При этом счетчик для созданного элемента управления уменьшается на единицу. Например, пользователь выбрал пункт меню «**Надпись 3**». В этом случае на форме в точке щелчка появится элемент управления **Static**, а при следующем отображении контекстное меню будет содержать пункты: «**Надпись 2**», «**Текст 3**», «**Кнопка 3**». Если на форме диалога выставлены все элементы одного типа, например, текст, то данный пункт исчезает из меню.



Для определения позиции курсора мыши в момент выбора пункта меню рекомендовать слушателям функцию API **GetCursorPos**.

```
BOOL GetCursorPos(
      LPPOINT lpPoint // указатель на структуру POINT, в которую будут
// записаны экранные координаты текущей позиции курсора мыши
);
```