Министерство образования Республики Беларусь

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра интеллектуальных информационных технологий

"ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ . Процедурный стиль"

Методическое пособие

для студентов специальности «Искусственный интеллект»

часть 1

БРЕСТ 2008

УДК 681.3 (075.8)
ББК с57

Целью пособия является знакомство студентов с базовыми понятиями оконных приложений и каркасного программирования в системе Microsoft Visual Studio C++ . предназначены для студентов специальности «Искусственный интеллект». Издаётся в 3-х частях, часть 1.

Составитель: Г.Л. Муравьев, доцент, к.т.н.

Рецензент: доцент кафедры математического моделирования Брестского государственного университета им. А.С. Пушкина, к.т.н., доцент Пролиско Е.Е.

1. ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ В ОС WINDOWS

1.1. Особенности применения технологий программирования в операционной системе Windows

При разработке программного обеспечения, ориентированного на исполнение с участием операционной системы Windows, говорят о разработке Windows-приложений или просто приложений (либо на момент разработки, т.е. до получения готового к исполнению продукта - говорят о разработке проекта приложения – project). Здесь существует два основных подхода (технологических направления) к разработке приложений.

Первый подход - процедурная разработка (процедурное проектирование, программирование, тестирование), базирующаяся на алгоритмической декомпозиции предметной области (автоматизируемых задач) и принципах структурной разработки программ. При разработке Windows-приложений этот подход называют также стилем низкоуровневого программирования, так как он связан с непосредственным использованием функций операционной системы Windows - Win32 API. Результатом применения этого стиля в среде Visual Studio являются Windows-приложения. Это API-приложения как "консольного", так и "оконного" типов.

Второй подход – объектно-ориентированная разработка (проектирование, программирование, тестирование), базирующаяся на объектной декомпозиции предметной области и принципах объектно-ориентированной модели ПО. При разработке Windows-приложений этот подход называют также стилем высокоуровневого программирования, так как он не ориентирован на непосредственное использование функций операционной системы Windows (хотя и допускает их прямое использование), а базируется на использовании библиотек готовых классов (где и эти функции инкапсулированы в соответствующих классах) и пользовательские классы. Это, например, следующие библиотеки: - MFC (Microsoft Foundation Class Library), используемая для создания сложных Windows-приложений с богатым графическим интерфейсом; - ATL (Microsoft Active Template Library) библиотека шаблонов ActiveX для создания COM-объектов и элементов управления ActiveX; - STL (Standard Template Library) стандартная библиотека шаблонов. Результатом применения этого стиля в среде Visual Studio являются Windows-приложения. Это оконные MFC-приложения как "статичного", так и "динамического" типов.

1.2. Особенности операционной системы Windows

Операционная система Windows базируется на понятии «виртуальная машина». Она воплощает принципы обеспечения независимости пользователя от особенностей конкретных аппаратных средств системы, а также деталей реализации программных компонентов и функций предоставляемого ею программного обеспечения. Это принципы сокрытия от пользователя деталей управления предоставляемыми ресурсами, принципы стандартизации средств управления. Указанная идеология находит отражение в интерфейсе управления ОС - GDI (Graphics Device Interface). Он обеспечивает программиста средствами выстраивания стандартных (оконных) интерфейсов и средствами их реализации (функциями API взамен функций DOS).

Windows отличается также обеспечением многозадачности. Под его управлением одновременно может выполняться много приложений. Более того, в рамках приложений могут быть определены параллельно выполняемые и при необходимости синхронизируемые участки (потоки). Указанное поддерживается аппаратом DLL-функций (реентерабельных модулей), автоматическим управлением памятью системы в защищенном режиме (на базе виртуальной – линейной, плоской модели памяти в 4 Гб с использованием механизмов страничного скроллинга), стратегией, вытесняющей многозадачности

с разнообразными дисциплинами обслуживания решаемых задач, включая механизм разделения времени типа RR.

При этом все выполняемые оконные приложения общаются с другими ресурсами (информационными, аппаратными, программными), с внешним миром (пользователями, ОС, аппаратурой, другими приложениями и т.д.) посредством единого механизма пересылки и обработки сообщений (о происходящих событиях) через одно центральное звено – ОС Windows. Сообщение, полученное приложением, может игнорироваться или обрабатываться в зависимости от задач, решаемых программистом. После завершения обработки текущего сообщения приложение бездействует до прихода следующего сообщения. Обработка сообщения состоит в отработке определенных программистом (пользователем) действий, описанных в приложении.

1.3. Особенности оконных приложений Windows

Такие приложения предлагают пользователю привычный для ОС Windows интерфейс на базе системы, иерархии окон. Это окна различных типов ("классические" всплывающие окна с клиентской областью, пользовательские и стандартные диалоговые окна, включая специальные окна типа Save As и т.п.) и расположенные в них элементы управления (меню, кнопки, списки и т.п.). Одно из окон является "главным", поскольку запускается первым. Телерь происходящие сообщения относятся именно к нему. В частном случае приложение может не иметь окон. Типичное оконное Windows-приложение в качестве главного окна выводит "классическое" всплывающее окно. При необходимости в роли главного окна может использоваться диалоговое окно.

Соответственно типичное оконное Windows-приложение, написанное, например, на языке C++, отличается специфической структурой, предусматривающей использование специальной глобальной функции WinMain (аналог функции main языка C) и, как минимум, одной функции типа обработчик сообщений окна. Это, как правило, обработчик сообщений главного окна. Функция WinMain является точкой первого входа в приложение после его запуска пользователем. Она производит интерфейсные настройки и затем переходит в режим ожидания и диспетчирования потока сообщений. А функция обработчик (главного окна), относящаяся к особому типу функций обратного вызова, запускается со стороны ОС посылкой сообщения, обрабатывает его, ждет следующих сообщений. Эта функция может включать вызовы других функций на языке C, в том числе вызовы библиотечных функций C, C++, вызовы функций Windows, пользовательских функций.

1.4. Особенности программирования оконных Windows-приложений на языках C, C++

При программировании приложений традиционно существуют отличия и особенности использования средств языка C и языка C++.

Есть особенности в написании Windows-приложений обусловленные спецификой 16 или 32 разрядной реализаций ОС Windows. Это: несовпадение разрядностей одних и тех же типов (например, тип int может использовать 16 или 32 бита); несовпадение типов указателей (в DOS и 16 разрядной реализаций ОС Windows используются как 16 битовые указатели типа near, так и 32 битовые указатели типа far и huge, а в 32 разрядной реализаций ОС Windows все указатели являются ближними - near и занимают 32 бита).

Есть особенности, определяемые самой ОС Windows, связанные с уже упоминавшейся особой структурой приложений, особыми типами функций, обработкой сообщений и организацией взаимодействия через сообщения. Это использование большого числа

новых заголовочных файлов, подключаемых, в частности, через хедер windows.h автоматически. Кроме этого существуют и стилевые особенности:

- использование переопределенных, дополнительных имен для обозначения типов данных языка С, С++ (наряду со стандартными именами); - использование новых, специфических типов (имен типов) для новых объектов-данных и указателей (дескрипторов, типов результатов, специфических типов указателей на наиболее распространенные типы данных общего назначения и т.п.), определенных на базе стандартных типов данных С++ ; - широкое применение венгерской нотации - правил записи имен переменных, объектов в соответствии с принципами структурного программирования; - использование специфичных структур данных как для организации функционирования приложений так и для организации пользовательских функций.

1.5. Заголовочные файлы Windows

В системе Visual Studio используется много новых заголовочных файлов (хедеров), подключаемых к приложению. Как правило, если приложение создается программой-мастером на базе специального шаблона, называемого заготовкой, каркасом приложения, то большинство необходимых приложению заголовочных файлов включаются в него мастером автоматически в том числе, например, и через хедер windows.h для оконных приложений. Состав включения зависит от выбранного в системе программирования типа создаваемого приложения и его атрибутов. Заголовочные файлы (несколько сотен) находятся в папке Include системы.

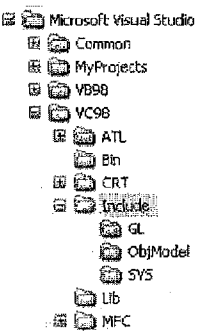


Рис 1. Структура папок

Общая характеристика наиболее часто используемых хедеров дана ниже. Так, заголовочный файл windows.h регулирует в зависимости от версии ОС и других атрибутов подключение дополнительных заголовочных файлов, например,

```
#include <windef.h>
#include <winbase.h>
#include <wingdi.h>
#include <winuser.h>
...
#include <wincon.h>
#include <winver.h>
...
```

```
#ifndef NOGDI
#include <commdlg.h>
#ifndef _MAC
#include <winspool.h>
#ifdef INC_OLE1
#include <ole.h>
#else
#include <ole2.h>
#endif
```

которые в свою очередь могут подключить следующие заголовочные файлы.

Заголовочный файл windowsx.h осуществляет подключение определений макрофункций API, обработчиков оконных сообщений и функций элементов управления.

Заголовочный файл windef.h определяет новые типы данных для специфичных объектов Windows, например, DWORD, BOOL, BYTE, WORD, PFLOAT и другие.

В заголовочном файле winbase.h определены структуры, прототипы базовых API функций, необходимых для управления ресурсами 32 разрядной реализаций ОС Windows. Например, структуры SYSTEMTIME, PROCESS_INFORMATION, прототипы функций WinMain, CreateThread, WriteFile, ReadFile, GetSystemTime и т.д.

В заголовочном файле wingdi.h декларированы прототипы GDI функций, константы и макросы, необходимые для работы с экраном (операций ввода-вывода), например структуры BITMAP, TEXTMETRICA, функции Arc, CreatePalette, CreatePen и другие.

В заголовочном файле winuser.h определены прототипы функций, константы и макросы, используемые программистом в пользовательских приложениях, например, функции WNDPROC, DLGPROC и другие.

Кроме этого, в папке Include представлены специфические хедеры типа cmath, cstdio, cctype, climits, cstring и другие, позволяющие подключать в новом стиле стандартные функции из старых библиотек. Например, для подключения математических функций, описанных хедером math.h, используется директива #include <cmath>. Здесь хедер cmath включает фрагмент для подключения хедера math.h

```
#ifdef   _STD_USING
  #undef  _STD_USING
  #include <math.h>
  #define _STD_USING
#else
  #include <math.h>
#endif   .
```

1.6. Типы данных Windows

Дополнительные имена для обозначения типов данных языка C, C++ (наряду с их стандартными именами), введенные путем переопределения командой typedef стандартных названий, представлены и описаны ниже (см. таблицу 1):

```
typedef unsigned long   DWORD;      typedef CONST void far   *LPCVOID;
typedef int             BOOL;       typedef int              INT;
typedef unsigned char   BYTE;       typedef unsigned int     UINT;
typedef unsigned short  WORD;       typedef unsigned int     *PUINT;
typedef float           FLOAT;      typedef unsigned long    ULONG;
typedef FLOAT           *PFLOAT;    typedef ULONG            *PULONG;
typedef BOOL near       *PBOOL;     typedef unsigned short   USHORT;
typedef BOOL far        *LPBOOL;    typedef USHORT           *PUSHORT;
typedef BYTE near       *PBYTE;     typedef unsigned char    UCHAR;
typedef BYTE far        *LPBYTE;    typedef UCHAR            *PUCHAR;
typedef int near        *PINT;      typedef UINT             WPARAM;
typedef int far         *LPINT;     typedef LONG             LPARAM;
typedef WORD near       *PWORD;     typedef LONG             LRESULT;
typedef WORD far        *LPWORD;    typedef WORD             ATOM;
typedef long far        *LPLONG;    typedef int              HFILE;
typedef DWORD near      *PDWORD;    typedef void far         *LPVOID;
typedef DWORD far       *LPDWORD;
```

Таблица 1. Типы данных общего назначения

Тип данного	Тип-аналог C++	Размер, бит	Описание значений	Диапазон
BOOL	—	32	логическое значение	TRUE (1), FALSE (0)
BOOLEAN	—	32	логическое значение	TRUE(1), FALSE(0)
BYTE	unsigned char	8	байт без знака для хранения числа или кода символа	0...255
CCHAR	char	8	символ Windows	-128...+127
CHAR	char	8	символ Windows	-128...+127
CONST	const	-	константа	-
DWORD	unsigned long	32	двойное слово без знака	0... 42944967295
DWORD-LONG	double	64	число с плавающей точкой со знаком	1.7E-308... 1.7E+308
FLOAT	float	32	число с плавающей точкой со знаком	3.4E-38... 3.4E+38
INT	int, long	32	целое число со знаком	-2147483648... +2147483647
LONG	long, int	32	целое число со знаком	-2147483648... +2147483647
LONGLONG	double	64	число с плавающей точкой со знаком	1.7E-308... 1.7E+308
SHORT	short	16	короткое целое число со знаком	-32768... +32767
TBYTE	unsigned char	8	байт без знака для хранения числа или кода символа	0...255
TCHAR	char	8	символ Windows или Unicode	-128...+127
UCHAR	unsigned char	8	символ Windows без знака	0...255
UINT	unsigned int	32	целое число без знака	0... 4294967295
ULONG	unsigned long	32	целое число без знака	0... 4294967295
USHORT	unsigned short	16	короткое целое число без знака	0...65535
VOID	void	-	любой тип	-
WCHAR	wchar_t	16	символ Unicode	0...65535
WORD	—	16	короткое целое число без знака	0...65535

Специфические типы указателей на наиболее распространенные типы данных общего назначения, определенные на базе стандартных типов данных C++ командой typedef (заголовочный файл windef.h), представлены в таблице 2 ниже.

Таблица 2. Типы указателей на типы данных общего назначения

Обозначение указателей	Тип адресуемого данного	Примечание
LPBOOL, PBOOL	BOOL	
LPBYTE, PBYTE	BYTE	
LPCCH, PCCH	CONST CHAR	константный символ
LPCH, PCH	CHAR	символ
LPCSTR, PCSTR	CONST CHAR	константная строка с завершающим нулем
LPCTSTR	CONST TCHAR	константная строка символов Windows или Unicode с завершающим нулем
LPCWCH, PCWCH	CONST WCHAR	константный символ Unicode
LPCWSTR, PCWSTR	CONST WCHAR	константная строка Unicode с завершающим нулем
LPDWORD, PDWORD	DWORD	
LPINT, PINT	INT	
LPLONG, PLONG	LONG	
LPSTR, PSTR	CHAR	строка символов с завершающим нулем
LPTCH, PTCH	TCHAR	символ Windows или Unicode
LPTSTR, PTSTR	TCHAR	строка символов Windows или Unicode с завершающим нулем
LPVOID, PVOID	VOID	
LPWCH, PWCH	WCHAR	символ Unicode
LPWORD, PWORD	WORD	
LPWSTR, PWSTR	WCHAR	строка Unicode с завершающим нулем
NPSTR	CHAR	строка символов с завершающим нулем
PBOOLEAN	BOOL	
PCHAR	CHAR	символ Windows
PFLOAT	FLOAT	
PSHORT	SHORT	
PSZ	CHAR	строка символов с завершающим нулем
PTBYTE	TBYTE	символ Windows или Unicode
PTCHAR	TCHAR	символ Windows или Unicode
PUCHAR	UCHAR	символ Windows без знака
PUINT	UINT	
PULONG	ULONG	
PUSHORT	USHORT	
PWCHAR	WCHAR	символ Unicode

Примеры новых типов для описания специфичных объектов Windows, определенных на базе стандартных типов данных C++ (см. заголовочный файл windef.h), представлены ниже. Например, команда typedef LONG LRESULT определяет тип LRESULT, используе-

мый как тип результата работы функции-обработчика сообщений окна; typedef int HFILE определяет тип HFILE, используемый в качестве дескриптора файла и т.п.

1.7. Венгерская нотация. Префиксы идентификаторов

Для повышения читабельности текстов приложений за счет осмысленности используемых в них идентификаторов (имен) в ОС Windows и приложениях Windows применяется венгерская нотация. Это правила записи имен переменных, объектов в соответствии с принципами структурного программирования. Соответственно при описании переменных в C++ при создании Windows-приложений используются специальные правила формирования имен на базе префиксов (см. таблицу 3).

Таблица 3. Типовые префиксы венгерской нотации

Префикс	Полный префикс	Смысловое значение префикса
b	Bool	логическая переменная
c	Character	символ, 1 байт
dw	DoubleWord	двойное слово без знака, 32 бита (целое число)
f / fn	Function	функция
pfn	PointerFunction	указатель на функцию
lpfn	LongPointerFunction	длинный указатель на функцию
h	HANDLE	дескриптор объекта
hDC	HANDLE	дескриптор контекста устройства
id		значение ID-идентификатора
l	LONG	длинное целое со знаком, 32 бита
lp	LongPointer	дальний указатель, 32 бита
lpsz	LongPointer-StringZero	дальний указатель на строку, заканчивающуюся нуль-символом, 32 бита
n	iNt	короткое целое число со знаком
p / np	Pointer	ближний указатель, 32 бита
pt	PoinT	x и y координаты точки, упакованные в 64 бита
s	String	строка
sz	StringZero	символьная строка, заканчивающаяся нуль-символом
pst	PointerStruct	указатель на структуру
psz	PointerStringZero	указатель на строку, заканчивающуюся нуль-символом
u	Uint	беззнаковый символ, целое без знака
w	WORD	беззнаковое значение, 16 бит
by	BYTE	беззнаковый символ
i	Integer	целое число, 32 бита
pv	PointerVoid	указатель на тип void
v	Void	тип void
W	Wide	символ UNICODE, 16-бит

Здесь

префиксное_ИМЯ_ПЕРЕМЕННОЙ ::= <префикс> < основное_ИМЯ_ПЕРЕМЕННОЙ >,

где префиксное_ИМЯ_ПЕРЕМЕННОЙ формируется по правилам венгерской нотации и явно определяет его типовую принадлежность; <префикс>, описываемый строчными буквами, задает тип переменной в виде одного из типовых сокращений; < основ-

ное_ИМЯ_ПЕРЕМЕННОЙ > представляет собой составное мнемоническое имя переменной, записанное строчными символами, без подчеркиваний, каждая часть имени пишется с заглавной буквы.

Ниже приведены примеры описания имен, используемых в различных приложениях:

```
int nCmdShow;          LPSTR lpszCmdLine;        HINSTANCE hInst;
int cbClsExtra;        LPSTR szCmdLine;          HANDLE hInst;
UINT nMessage;         LPCSTR lpszClassName;     HWND hWnd;
DWORD dwStyle;         LPCSTR szProgName;        HICON hIcon;
                       LPCSTR lpszMenuName;      HCURSOR hCursor;
MSG lpMsg;                                       HMENU hMenu;
WPARAM wParam;         WNDPROC lpfnWndProc;
LPARAM lParam;
```

2. ОСОБЕННОСТИ ПРИМЕНЕНИЯ ТЕХНОЛОГИЙ ПРОГРАММИРОВАНИЯ В СИСТЕМЕ VISUAL STUDIO. КАРКАСНОЕ ПРОГРАММИРОВАНИЕ

Visual Studio — это система программирования, комплект (suite) средств разработки, включающий язык Visual C++, комплекс программ, объединенных в интегрированную среду разработки (Integrated Development Environment), и библиотеки функций. Visual Studio ориентирован на создание как автономных (работающих на отдельной ПЭВМ), так и сетевых приложений под Windows. Это оконные Windows-приложения (в частном случае консольные приложения), создаваемые с использованием как технологии процедурной разработки программ (стиль низкоуровневого программирования), так и технологии объектно-ориентированной разработки программ (стиль высокоуровневого программирования). Соответственно Visual Studio содержит в своем составе два набора программных средств для поддержки указанных технологий, включающих, помимо типового набора средств системы программирования, также готовые каркасы типовых приложений и мастера для их создания.

Каркас приложения – это готовая заготовка для каждого типового Windows-приложения (например, каркас консольного приложения и т.п.). Выбор, предварительная настройка конкретного типа каркаса и автоматическая генерация его текста производится с помощью мастеров построения каркасов. Результат – программный текст. Само же программирование в Visual Studio называется каркасным, поскольку базируется на использовании каркасов типовых Windows-приложений с их последующим допрограммированием.

Результатом применения стиля низкоуровневого программирования в системе Visual Studio являются API-приложения, как "консольного" (тип проекта в системе Visual Studio – "Win32 Console Application") так и "оконного" (тип проекта в системе Visual Studio – "Win32 Application"), а также смешанного типов. При создании таких приложений с помощью средств системы Visual Studio можно использовать следующие типовые каркасы:

1) для консольного API-приложения (тип проекта в системе Visual Studio – "Win32 Console Application") каркасы типа пустой - Empty ("An empty project"), простой - Simple ("A simple application"), простой с выводом приветствия - Hello ("A "Hello, World" application");

2) для смешанного консольного API-приложения (тип проекта в системе Visual Studio – "Win32 Console Application") каркас типа API-MFC ("An application that supports MFC");

3) для оконного API-приложения (тип проекта в системе Visual Studio – "Win32 Application") каркасы типа пустой - Empty ("An empty project"), простой - Simple ("A simple Win32 application"), Hello ("A typical "Hello, World" application").

Результатом применения стиля высокоуровневого программирования в системе Visual Studio являются "статические" (тип проекта в системе Visual Studio – "MFC AppWizard(exe)") и "динамические" (тип проекта в системе Visual Studio – "MFC AppWizard(dll)") оконные MFC-приложения. При создании таких приложений с помощью системы Visual Studio можно использовать следующие типовые каркасы:

1) для статического MFC-приложения (тип проекта в системе Visual Studio – "MFC AppWizard(exe)") каркасы с однодокументным интерфейсом ("Single document"), с многодокументным интерфейсом ("Multiple documents"), с интерфейсом в виде диалогового окна ("Dialog based");

2) для динамического MFC-приложения (тип проекта в системе Visual Studio – "MFC AppWizard(dll)") доступны каркасы "Regular DLL with MFC statically linked", "Regular DLL using shared MFC DLL), "MFC extension DLL (using shared MFC DLL)).

3. ОКОННЫЕ WINDOWS-ПРИЛОЖЕНИЯ

3.1. Общая структура оконных приложений

С точки зрения пользователя (на уровне применения), приложение может быть описано законом функционирования с помощью поведенческих моделей. Например, в терминах перечня решаемых задач, соответствующей входной и выходной информации и правил получения выходной информации из входной без раскрытия способов реализации задач (закона функционирования приложения) как на рисунках 2, 3. Каждая из решаемых задач может быть детализирована в виде сценария использования приложения в терминах поддерживаемого им графического интерфейса пользователя.

На рисунках здесь и далее сплошными линиями показаны направления передачи управления от модуля к модулю (например, от ОС к приложению), пунктирным линиям соответствуют направления передачи сообщений (например, от ОС к приложению и обратно). Ломаные стрелки отображают действия, события, инициирующие сообщения (например, действия пользователя - нажатие клавиши, перемещение "мыши" и т.п.), полые стрелки показывают направления движения информации, данных (например, от пользователя к приложению и в обратно).

В структуре приложения (рисунок 4) можно выделить часть, видимую пользователем – интерфейс приложения, и сами программы, поддерживающие интерфейс и выполняющие обработку данных в соответствии с принятыми сообщениями, - реализация приложения.

Интерфейс оконного приложения (рисунок 5) в общем случае может быть представлен, как правило, системой окон различных классов и типов с расположенными на их поверхности элементами управления. Пользователь может воздействовать на элементы управления (например, нажатием кнопки, выбором пункта меню и т.д.) и тем самым посылать сообщения. При загрузке приложения, как правило, выводится начальное (главное) окно приложения и начинает работать связанная с ним программа-обработчик сообщений, адресуемых этому окну. При этом возможна передача управления (активности) одному из других окон интерфейса (обработчику его сообщений). В текущем времени только одно из окон приложения является активным, а следовательно все полученные в этот момент сообщения адресуются именно этому окну (точнее на обработку программе-обработчику сообщений активного окна). Сообщения приложению направляются со стороны ОС.

Более детально Windows-приложение может быть специфицировано в терминах обрабатываемых сообщений. Здесь поведенческое описание может представлять спецификацию сообщений, направляемых приложению, и спецификацию реакций приложения

на полученные сообщения (при этом приложение по прежнему рассматривается как "черный" ящик).

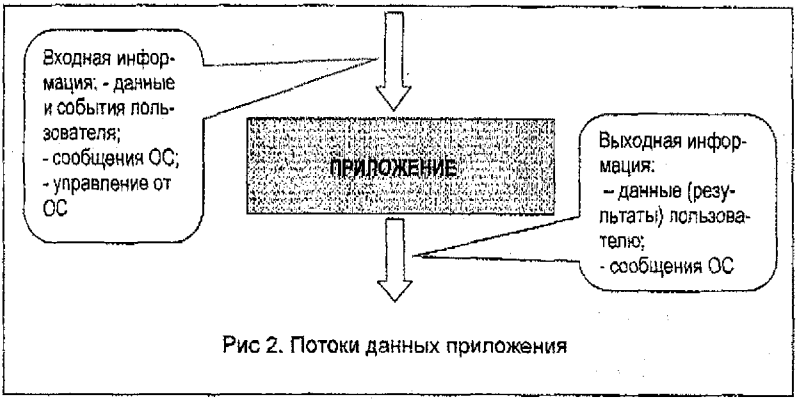


Рис 2. Потоки данных приложения

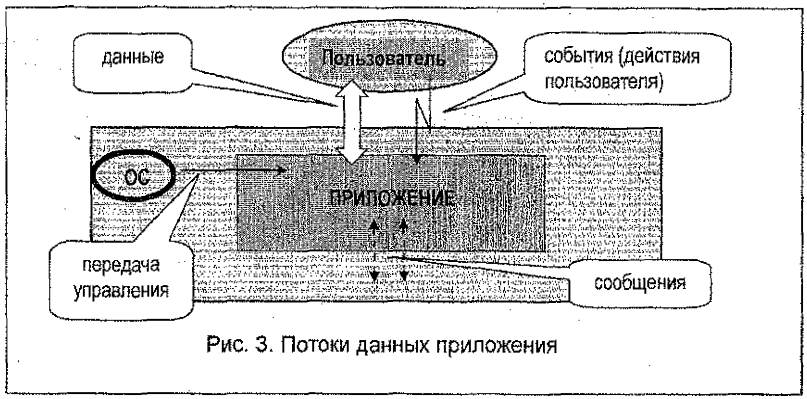


Рис. 3. Потоки данных приложения

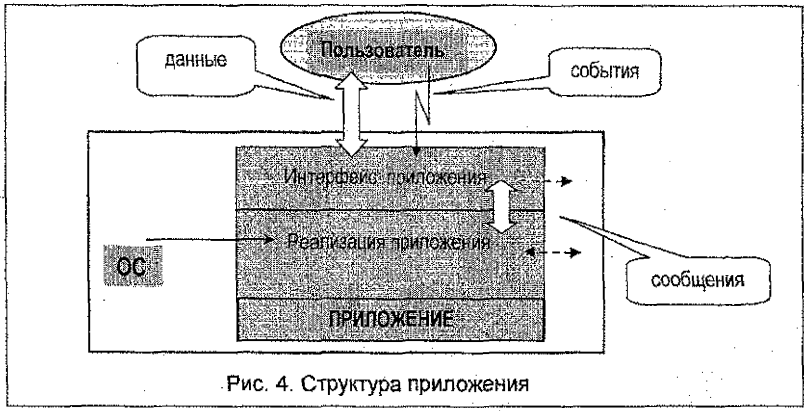


Рис. 4. Структура приложения

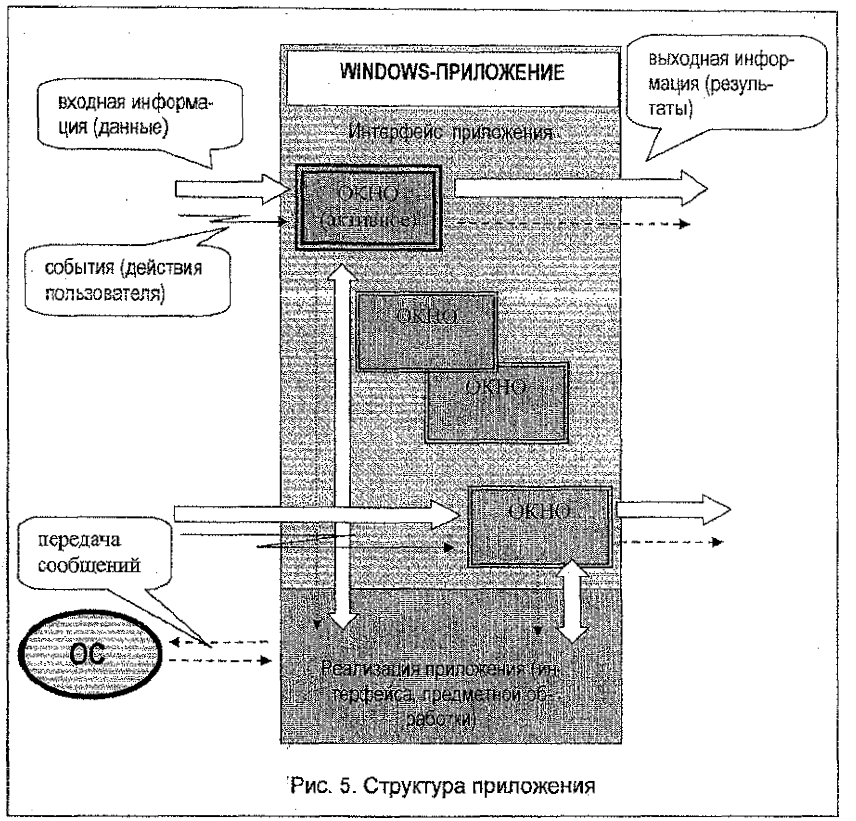


Рис. 5. Структура приложения

Более детально Windows-приложение может быть специфицировано в терминах обрабатываемых сообщений. Здесь поведенческое описание может представлять спецификацию сообщений, направляемых приложению, и спецификацию реакций приложения на полученные сообщения (при этом приложение по-прежнему рассматривается как "черный" ящик).

Приложения общаются с другими ресурсами, другими приложениями и даже со своими модулями, функциями через посредника – ОС, путем посылки сообщений. Через каждое активное приложение проходит поток сообщений. Приложение циклически отбирает те, к которым оно чувствительно, и организует их обработку (см. рисунок 6).

С учетом оконного интерфейса приложений схема их взаимодействия может быть представлена как на рисунке 7.

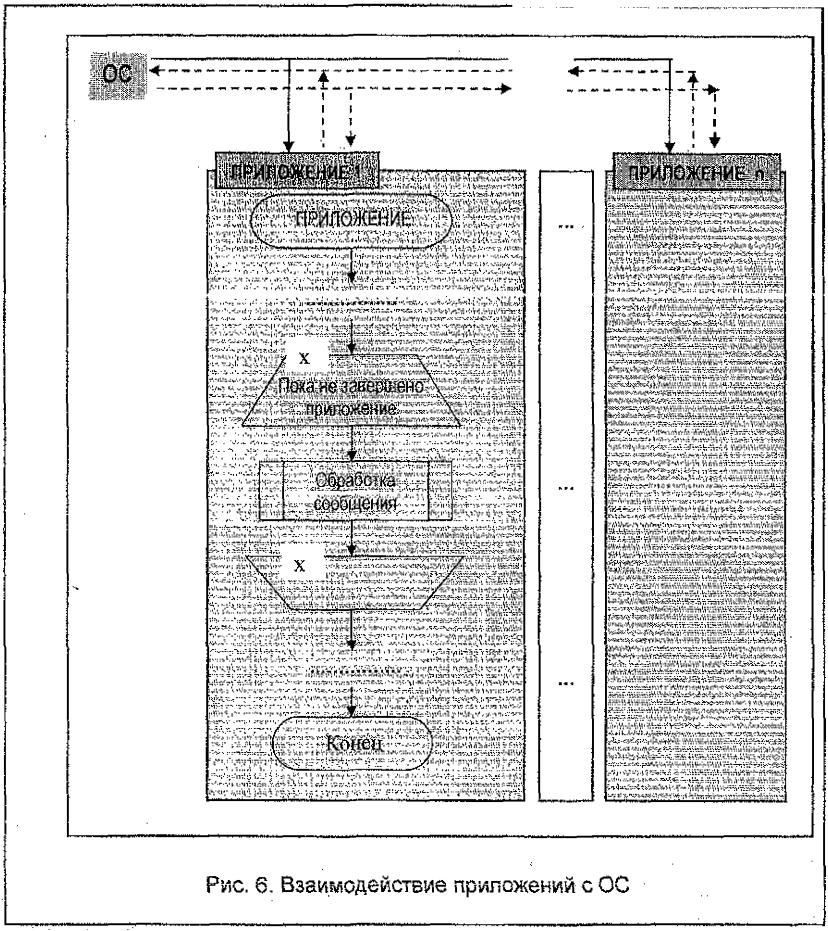


Рис. 6. Взаимодействие приложений с ОС

3.2. Общая структура оконных приложений (уровень реализации)

С точки зрения программиста (уровень реализации), приложение как набор описаний (текстов) может быть описано в терминах модулей, функций, ресурсов. Это структурное описание (в виде спецификации модулей, схемы иерархии модулей, схемы включения модулей в другие модули), функциональное описание (порядок функционирования приложения на уровне модулей, представленный, например, в виде графической схемы обобщенного алгоритма работы приложения). Более детально на этом уровне Windows-приложение специфицируется алгоритмически с учетом специфики обрабатываемых сообщений.

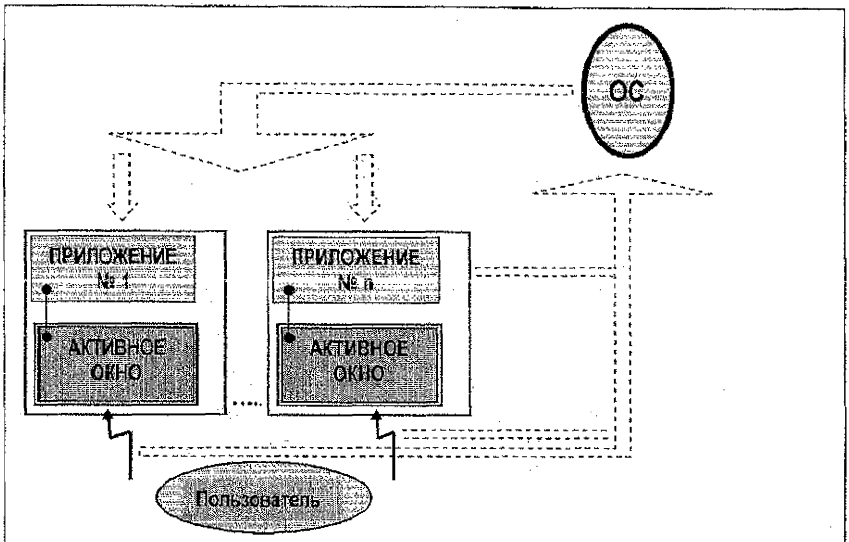


Рис. 7. Взаимодействие приложений с ОС

Соответственно оконное приложение можно рассматривать как совокупность описаний - основной программы WinMain (аналога main) и набора типовых комплектов. Типовой комплект компонентов приложения представлен на рисунке 8. Он включает описание ресурсов окна (стиля, вида, состава и размещения элементов управления в окне и т.п.) и описание алгоритмов работы программы – обработчика сообщений, адресуемых окну. Оба описания каждого комплекта компонентов связываются при создании приложения.

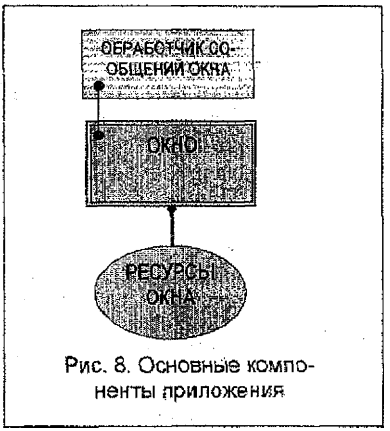


Рис. 8. Основные компоненты приложения

При активизации окно предоставляется пользователю (визуализируется на основе описания ресурсов окна) системой как элемент интерфейса. Пользователь, совершая события – действия над элементами управления окна, над самим окном либо другие действия, непосредственно не связанные с окном в момент его активизации (перемещение или нажатие клавиш "мыши", манипуляции с клавиатурой и т.п.) посылает соответствующие сообщения, пересылаемые ОС обработчику сообщений, связанному с окном. Указанное иллюстрируется рисунком 9.

С учетом изложенного структура приложения и состав приложения представлены на рисунках 10, 11.

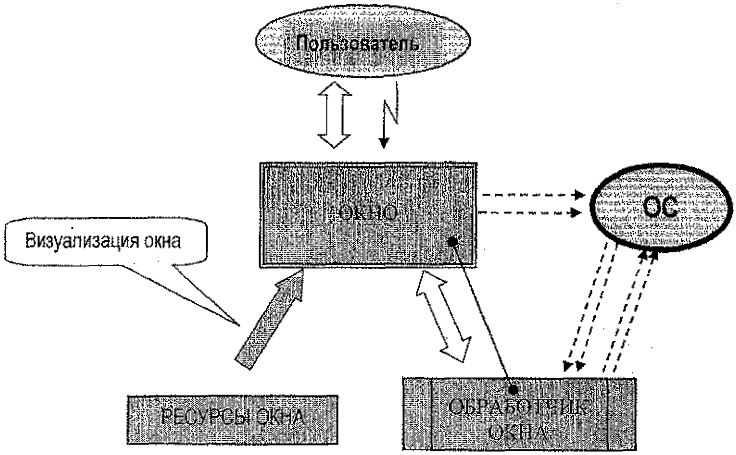


Рис. 9. Взаимодействие основных компонентов приложения

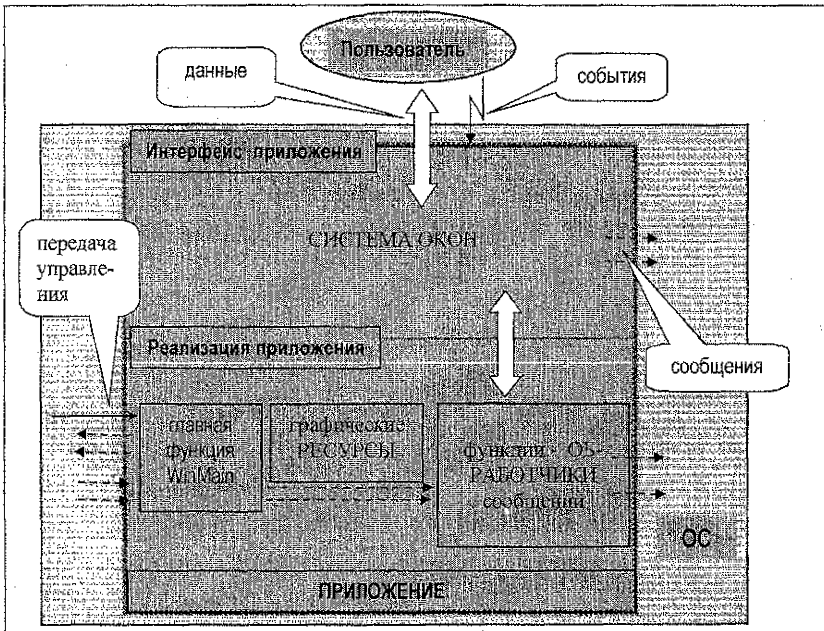


Рис. 10. Структура приложения

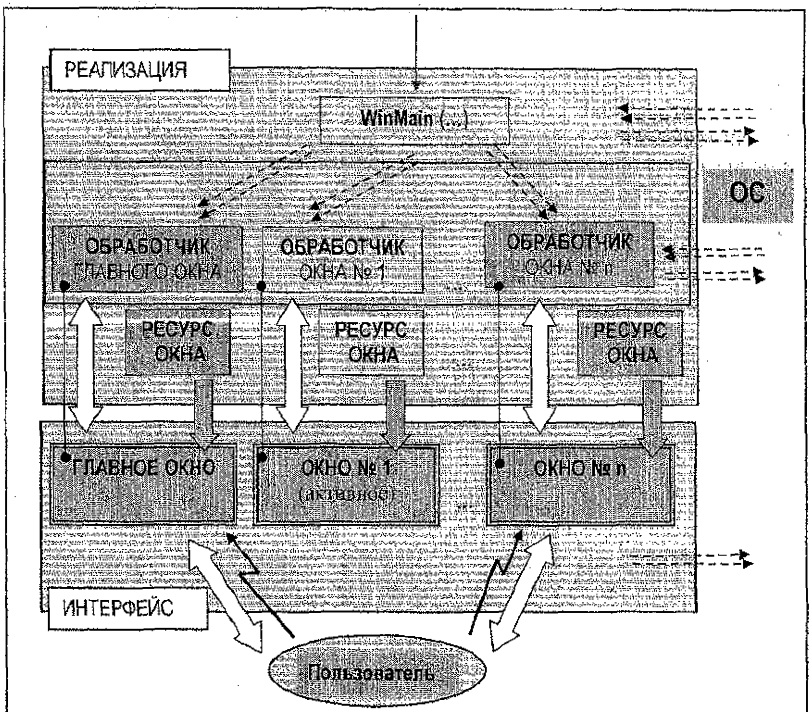


Рис. 11. Состав приложения

3.3. Структура программного обеспечения оконных приложений

Примерная структура программ оконного приложения представлена ниже

< Команды_препроцессора >
< Прототип_функции_обработчика_главного_окна >
[< Прототипы_других_функций_обработчиков >]
[< Прототипы_прикладных_функций >]
[< Описания_глобальных_объектов >]
< Описание_главной_функции_WinMain >
< Описание_функции_обработчика_главного_окна >
[< Описание_других_функций_обработчиков >]
[< Описание_прикладных_функций >]

4. СОСТАВ, СТРУКТУРА И ФУНКЦИОНИРОВАНИЕ ТИПОВОГО КАРКАСА ОКОННОГО ПРИЛОЖЕНИЯ (ТКП)

4.1. Структура ТКП

Ниже рассмотрена базовая структура оконных приложений, называемая типовым каркасом приложения (ТКП). Реальное приложение создается путем допрограммирования и, или модификации типового каркаса.

Примерная структура ТКП представлена ниже на рисунке 12.

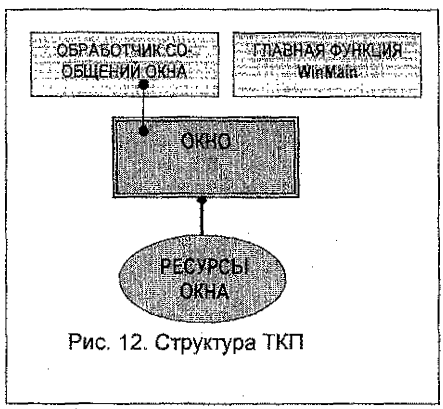


Рис. 12. Структура ТКП

Прохождение потоков данных в ТКП представлено ниже на рисунке 13.

4.2. Структура программного обеспечения ТКП

ТКП содержит две основные функции – главную WinMain (аналог main) и обработчик сообщений окна приложения. Они связаны друг с другом ассоциативно (обслуживают одно приложение) и информационно через общее главное окно интерфейса. Главная функция его создает и визуализирует, передавая активность и управление, и обеспечивая пользователя инструментом взаимодействия с приложением. Функция-обработчик (функция окна, оконная функция) относится к функциям обратного вызова, инициируемых ОС посылкой сообщений с уточняющими параметрами. Обработчик, обрабатывая сообщения окну, реализует нужные пользователю алгоритмы. С точки зрения классического управления в схеме иерархии эти модули стоят на одном уровне, так как не запускают друг друга непосредственно. Примерная структура ТКП оконного приложения представлена ниже, а алгоритмы указанных функций представлены на рисунке 14.

< Команды_препроцессора >
< Прототип_функции_обработчика_главного_окна >
[< Описания_глобальных_объектов >]
< Описание_главной_функции_WinMain >
< Описание_функции_обработчика_главного_окна > .

Ниже представлена структура функции WinMain, вызываемой из ОС,

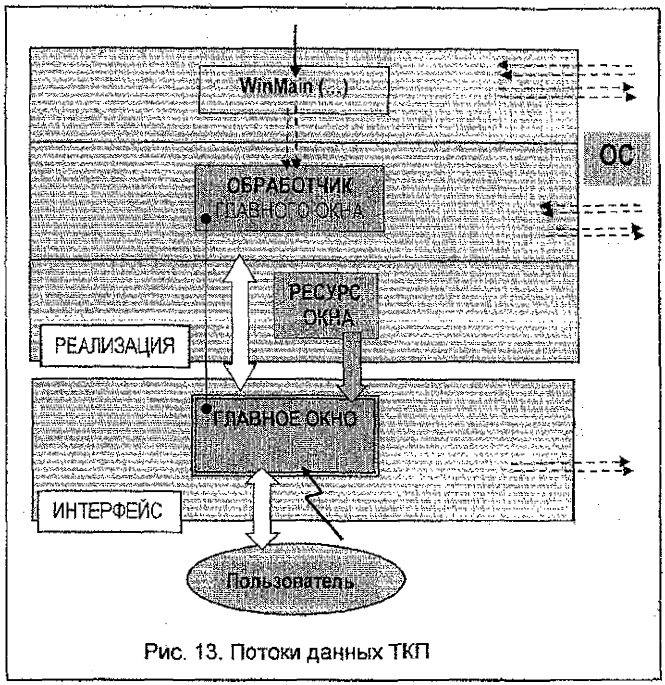


Рис. 13. Потоки данных ТКП

СОЗДАНИЕ-ГЛАВНОГО-ОКНА-ПРИЛОЖЕНИЯ
  Описание стиля главного окна
  Регистрация стиля окна
  Создание экземпляра окна
  Визуализация окна
  Обновление окна
РАБОТА-С-СООБЩЕНИЯМИ
  ЦИКЛ-ПОКА не завершено приложение (не поступило сообщение WM_QUIT)
    Получить очередное сообщение из очереди сообщений приложения
    Диспетчировать сообщение (передать обработчику соответствующего окна)
  КОНЕЦ-ЦИКЛА
Завершение работы приложения

4.3. Сообщения. Общая схема обработки сообщений

Сообщение – уведомление приложения о том, что произошло событие, требующее обработки, реакции приложения. Сообщения приходят от пользователей (например, при выборе пункта меню, нажатии кнопки и т.п.), от других приложений, от самого приложения (например, посылкой сообщения на перерисовку клиентской области окна), от ОС (например, сообщения таймера). Каждое сообщение сопровождается рядом атрибутов (как то системное время, состояние клавиатуры, мыши, идентификация источника сообщения и т.д.), исчерпывающе характеризующих сообщение для его обработки.

Сообщения обозначаются как WM_SIZE, WM_MOVE, WM_CLOSE, WM_COMMAND, WM_LBUTTONDOWN, WM_DESTROY, WM_QUIT, WM_KEYDOWN, WM_TIMER, WM_CHAR, WM_CUT, WM_COPY и т.п. Непосредственно с приложением, окном, кли-

ентской областью связаны сообщения WM_PAINT (перерисовать окно при изменении его размеров), WM_CLOSE при свертывании клиентского окна, WM_DESTROY при его закрытии, WM_QUIT при закрытии приложения.

Предварительная обработка сообщений с целью их отправки в соответствующие обработчики выполняется фрагментом WinMain, представленным ниже, а общая схема обработки сообщений иллюстрируются рисунком 15.

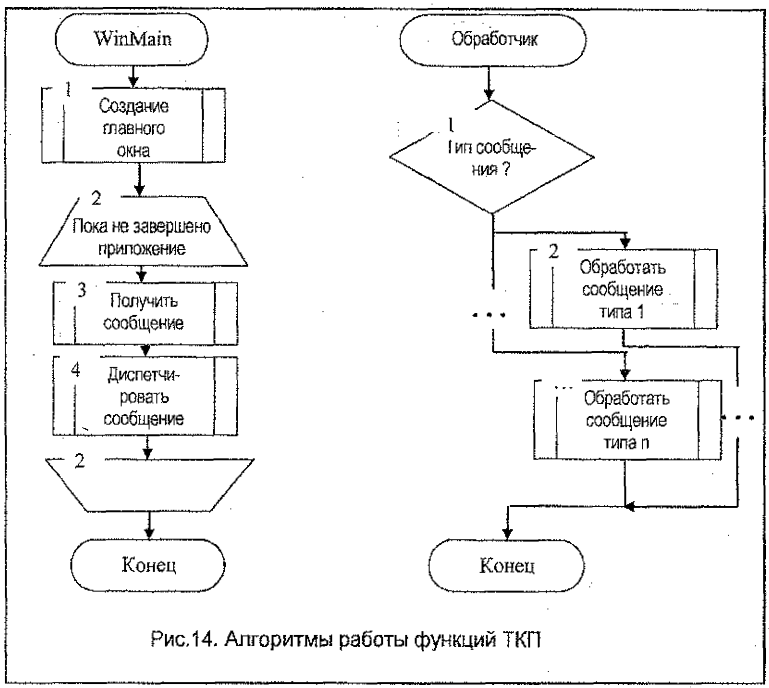


Рис.14. Алгоритмы работы функций ТКП

```
while (GetMessage (&lpMsg, NULL, ... , ... ))
{
   TranslateMessage (&lpMsg);
   DispatchMessage (&lpMsg);
}
```

4.4. Информационная модель ТКП

С информационной точки зрения приложение может рассматриваться как совокупность специальных структур данных, обеспечивающих его работу. Работа ТКП базируется на использовании структур типа WNDCLASS, PAINTSTRUCT, HDC, MSG и других.

WNDCLASS (См. ПРИЛОЖЕНИЕ А) - структура для хранения информации о стиле окна. Описывается пользователем и после регистрации стиля окна в ОС <ИмяСтиляОкна> может использоваться для создания конкретных экземпляров окна. Описание полей структуры WNDCLASS.

```
typedef struct tagWNDCLASS
{
```

```
    LPCSTR lpszClassName;
    HINSNANCE hInstance;
    WNDPROC lpfnWndProc;
    HCURSOR hCursor;
    HICON hIcon;
    LPCSTR lpszMenuName;
    HBRUSH hbrBackground;
    UINT Style;
    cbClsExtra;
    cbWndExtra;
} WNDCLASS.
```

Здесь

- lpszClassName = <ИмяСтиляОкна> - имя класса, стиля окна;
- hInstance = <ДескрипторПриложения> - дескриптор приложения, которое регистрирует класс окна (берется из значения соответствующего параметра функции WinMain, которое задается ОС при запуске приложения);
- lpfnWndProc = <ИмяОбработчикаОкна> - значение указателя на функцию-обработчик сообщений окна, которая выполняет все задачи, связанные с окном (функция описывается пользователем);

hCursor – ресурс окна, тип курсора окна. См. ПРИЛОЖЕНИЕ Б. Встроенные курсоры;

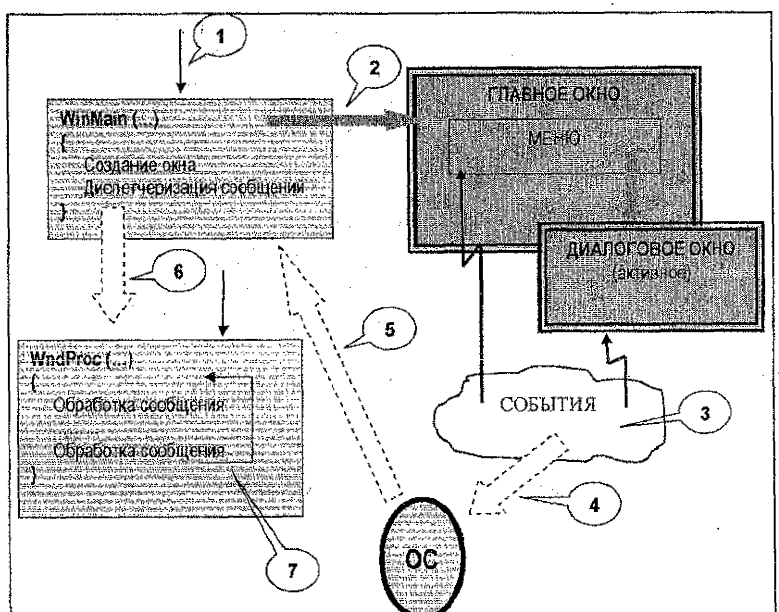


Рис.15. Схема обработки сообщений в ТКП

hIcon - ресурс окна, задает вид пиктограммы при выводе окна в свернутом виде (в виде пиктограммы, обычно NULL);

lpszMenuName = <ИмяРесурсаМеню> – ресурс окна - пользовательское меню;

hbrBackground - ресурс окна, определяет кисть, используемую для закраски фона окна (значением данного параметра может быть как идентификатор физической кисти, так и значение цвета). См. ПРИЛОЖЕНИЕ В. Встроенные кисти (закраски фона окна);

style – доопределяет класс, стиль окна (например, параметр CS_VREDRAW обеспечивает перерисовку содержимого клиентской области окна при изменении размера окна по вертикали и т.п.). См. ПРИЛОЖЕНИЕ Г. Значения параметра Style структуры WNDCLASS;

cbClsExtra - задает количество дополнительных байт, выделяемых структуре WNDCLASS (обычно NULL);

cbWndExtra - задается количество дополнительных байт, выделяемых для всех дополнительных структур, которые создаются с использованием данного класса, стиля окна (обычно NULL).

Использование структуры иллюстрируется ниже

```
char  szWindowClass []= <ИмяСтиляОкна> ;
.....
WNDCLASS    <СтруктураДанныхОкна>;
<СтруктураДанныхОкна>.lpszClassName  = (LPCSTR) <ИмяСтиляОкна>;
<СтруктураДанныхОкна>.hInstance      = (HINSTANCE) <ДескрипторПриложения>
<СтруктураДанныхОкна>.lpfnWndProc    = (WNDPROC) <ИмяОбработчикаОкна>
<СтруктураДанныхОкна>.hCursor        = (HCURSOR) <ТипКурсора>
                                     = LoadCursor  (NULL, IDC_ARROW);
<СтруктураДанныхОкна>.hIcon          = (HICON) <Пиктограмма>
                                     = 0;
<СтруктураДанныхОкна>.lpszMenuName   = (LPCSTR) <ИмяРесурсаМеню>;
<СтруктураДанныхОкна>.hbrBackground  = (HBRUSH) <КистьФона>
                                     = GetStockObject (WHITE_BRUSH);
<СтруктураДанныхОкна>.style          = (UINT) <ДополнительныйСтиль1>
                                     = CS_HREDRAW|CS_VREDRAW;
<СтруктураДанныхОкна>.cbClsExtra     = (UINT) <ЧислоДополнительныхБайт>
                                     = 0;
<СтруктураДанныхОкна>.cbWndExtra     = (UINT) <ЧислоДополнительныхБайт>
                                     = 0
```

MSG - структура данных для хранения информации о сообщении

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

Здесь
- HWND hWnd - <ДескрипторОкна>;
- UINT message – код <ТипСообщения>;
- WPARAM wParam <ПараметрыСообщения> (ПервыйПараметрСообщения);
- LPARAM lParam <РасширенныеПараметрыСообщения> (ВторойПараметрСообщения);
- DWORD time время возникновения сообщения;
- POINT pt позиция курсора мыши.

PAINTSTRUCT (см. winuser.h) - структура данных (заполняется каждый раз, когда приложение перехватывает сообщение WM_PAINT) для хранения информации о клиентской области окна, с которой связан контекст устройства КУ. КУ – системный ресурс, (область памяти) для хранения атрибутов объектов, связанных с рисованием (информация о кисти, перьях, шрифтах и т.п.). Используется графическими функциями GDI через <ДескрипторКУ>. При этом предполагается последовательность действий: Получить КУ. Изменить атрибуты КУ. Использовать функции GDI (рисовать). Вернуть КУ. Описание

```
typedef struct tagPAINTSTRUCT
{
    HDC hdc;
    BOOL fErase;
    RECT rcPaint;
    BOOL fRestore;
    BOOL fIncUpdate;
    BYTE rgbReserved [32];
}  PAINTSTRUCT.
```

Здесь
- HDC hdc - <ДескрипторКУ>;
- BOOL fErase - флаг перерисовки фона окна (0), иначе используется прозрачное окно;
- RECT rcPaint – описание прямоугольной области окна, той части, которую надо перерисовать;
- RECT - структура данных для описания прямоугольной области, части окна, которую надо перерисовать

```
typedef struct tagRECT
{
   LONG   left;
   LONG   top;
   LONG   right;
   LONG   bottom;
} RECT, *PRECT, NEAR *NPRECT, FAR *LPRECT;
```

Пример использования для подготовки рисования

```
PAINTSTRUCT  ps;
ps.rcPaint = { 0, 0 , 290, 73 };
HDC hdc = BeginPaint (hWnd, &ps);  .
```

TEXTMETRIC (см. winuser.h) - информационная структура данных, которая хранит атрибуты (метрики) текущего шрифта, связанного с КУ. Задание атрибутов выполняется функциями SetTextColor(), SetBkColor() и др. Поле tmExternalLeading определяет расстояние между строками, поле tmHeight задает полную высоту символов используемого шрифта и т.д.

```
typedef struct tagTEXTMETRICA
{
    LONG    tmHeight; Полная высота символов шрифта
    LONG    tmAscent;
    LONG    tmDescent;
    LONG    tmInternalLeading;
    LONG    tmExternalLeading; Расстояние между строками
    LONG    tmAveCharWidth;
    LONG    tmMaxCharWidth;
    LONG    tmWeight;
    LONG    tmOverhang;
    LONG    tmDigitizedAspectX;
    LONG    tmDigitizedAspectY;
    BYTE    tmFirstChar;
    BYTE    tmLastChar;
    BYTE    tmDefaultChar;
    BYTE    tmBreakChar;
    BYTE    tmItalic;
    BYTE    tmUnderlined;
    BYTE    tmStruckOut;
    BYTE    tmPitchAndFamily;
    BYTE    tmCharSet;
} TEXTMETRICA, *PTEXTMETRICA, FAR *LPTEXTMETRICA;
```

4.5. Главная функция ТКП

Прототип главной функции приложения

```
int WINAPI WinMain ( HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine,
                     int nCmdShow); .
```

Здесь

- HINSTANCE hInstance – идентификатор <ДескрипторПриложения> текущего экземпляра приложения (определяется ОС);
- HINSTANCE hPrevInstance - всегда NULL;
- LPSTR lpszCmdLine - указатель на командную строку приложения;
- int nCmdShow - способ <ИзображенияОкнаПриПервомВыводе> (например, параметр SW_SHOWDEFAULT активизирует окно и выводит его с использованием текущих параметров – по умолчанию; параметр SW_SHOWNORMAL активизирует окно и если окно было увеличено или уменьшено до пиктограммы, то оно восстанавливается с учетом начального положения и размера окна).

```
int WINAPI WinMain ( HINSTANCE <ДескрипторПриложения>,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine,
                     int <ИзображениеОкнаПриПервомВыводе>);
```

4.6. Функция-обработчик сообщений главного окна ТКП

Назначение функции – обработка сообщений, адресуемых окну. Соответственно обработчик реагирует на сообщения.

Основные сообщения: – при визуализации окна принимается сообщение WM_CREATE; - при изменении размеров окна принимается сообщение WM_PAINT с требованием перерисовки окна; – при завершении работы приложения принимается сообщение WM_QUIT; - закрытие окна сопровождается сообщением WM_DESTROY, что при закрытии главного окна означает, как правило, и закрытие приложения. Соответственно при обработке сообщения WM_DESTROY системе надо переслать сообщение WM_QUIT; - сообщение WM_COMMAND о выборе пункта пользовательского меню окна приложения и другие.

Функции-обработчики сообщений окон приложения, включая главное и диалоговые окна, имеют общий прототип

LRESULT CALLBACK <ИмяОбработчикаОкна> (HWND hWnd, UINT Message, WPARAM wParam, LPARAM lParam); .

Здесь
- HWND hWnd - <ДескрипторОкна> или <ДескрипторДиалоговогоОкна >, чьи сообщения обрабатываются;
- UINT Message - <ТипСообщения>, которое обрабатывается;
- WPARAM wParam - < ПараметрыСообщения >, которое обрабатывается;
- LPARAM lParam - < РасширенныеПараметрыСообщения >, которое обрабатывается.

Для передачи параметров используются типы, определенные как

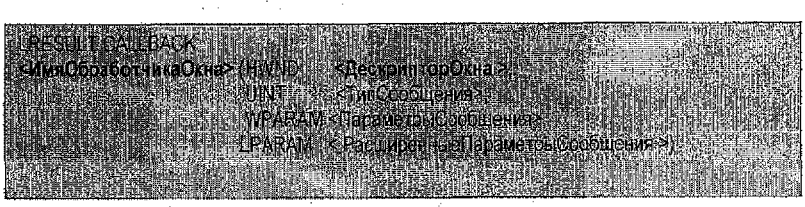
typedef UINT WPARAM;
typedef LONG LPARAM; .

Тип результата работы функции-обработчика окна (главного окна) определен как

typedef LONG LRESULT; .

Указатели на соответствующую функцию-обработчик описаны как

typedef LRESULT (CALLBACK* WNDPROC) (HWND, UINT, WPARAM, LPARAM);
typedef BOOL (CALLBACK* DLGPROC) (HWND, UINT, WPARAM, LPARAM); .



Например, прототип функции-обработчика главного окна WndProc

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

и описание функции-обработчика

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT Message,
                          WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    switch (Message)
    {
        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);
            < Фрагмент_пользователя >
            ValidateRect(hWnd,NULL);
            EndPaint(hWnd, &ps);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc(hWnd, Message, wParam, lParam));
 }
 return 0;
```

4.7. Примерный текст ТКП. Состав функций

Примерный текст типового каркаса пользователя приведен ниже

```
#include <windows.h>
LRESULT CALLBACK   <ИмяОбработчика> (HWND, UINT, WPARAM, LPARAM);
char  szWindowClass [ ] = <ПользовательскоеИмяСтиляОкна>;

int  WINAPI  WinMain (HINSTANCE <ДескрипторПриложения>,
                      HINSTANCE hPreInstance,
                      LPSTR lpszCmdLine, int  nCmdShow)
{
   < Описание_главной_функции >
}

LRESULT CALLBACK  <ИмяОбработчика> (HWND <ДескрипторОкна>,
                                    UINT  <КодСообщения>,
                                    WPARAM wParam, LPARAM lParam)
{
    < Описание_обработки_сообщений_главного_окна >
}
```

Более детальный текст ТКП приведен ниже

```
#include <windows.h>
LRESULT CALLBACK  <ИмяОбработчика> (HWND, UINT, WPARAM, LPARAM);
char  szWindowClass [ ] = <ПользовательскоеИмяСтиляОкна>;

int  WINAPI  WinMain (HINSTANCE <ДескрипторПриложения>,
                      HINSTANCE hPreInstance,
                      LPSTR lpszCmdLine, int  nCmdShow)
```

```
{
      СОЗДАНИЕ-ГЛАВНОГО-ОКНА-ПРИЛОЖЕНИЯ
      РАБОТА-С-СООБЩЕНИЯМИ
        MSG      lpMessage;
        while (GetMessage (&lpMessage, NULL, 0, 0))
        {
          TranslateMessage (&lpMessage);
          DispatchMessage (&lpMessage);
        }
} .
```

Здесь полученное приложением сообщение копируется в структуру данных типа MSG, на которую указывает адрес lpMsg, и передает его далее на обработку соответствующей функции-обработчику окна

```
LRESULT CALLBACK <ИмяОбработчика> (HWND <ДескрипторОкна>,
                                   UINT <КодСообщения>,
                                   WPARAM wParam, LPARAM lParam)
{
   HDC  <ДескрипторКонтекстаУстройства>;
   PAINTSTRUCT <СтруктураКлиентскойОбластиОкна>;
   switch  (<КодСообщения>)
   {
      case WM_PAINT:
         <ДескрипторКонтекстаУстройства>=BeginPaint(<ДескрипторОкна>,
               &ps);
         < Фрагмент_пользователя >
         ValidateRect (<ДескрипторОкна>, NULL);
         EndPaint (<ДескрипторОкна>, <СтруктураКлиентскойОбластиОкна>);
         break;
      case <ЗначениеКодаСообщения> :
           < Обработка_сообщения >
      case <ЗначениеКодаСообщения> :
           < Обработка_сообщения >
             .....
   }
  .....
}
```

Состав модулей и структур данных ТКП представлен на рисунке 16.

4.8. Описание и регистрация класса (стиля) окна. Использование структуры WNDCLASS

При создании приложения создается два основных компонента: - окно, как интерфейсное средство общения пользователя с приложением; - функция, обрабатывающая все события (сообщения), происходящие как в окне так и во внешней среде приложения. Окно описывается разработчиком. Данные о каждом окне хранятся в полях структуры

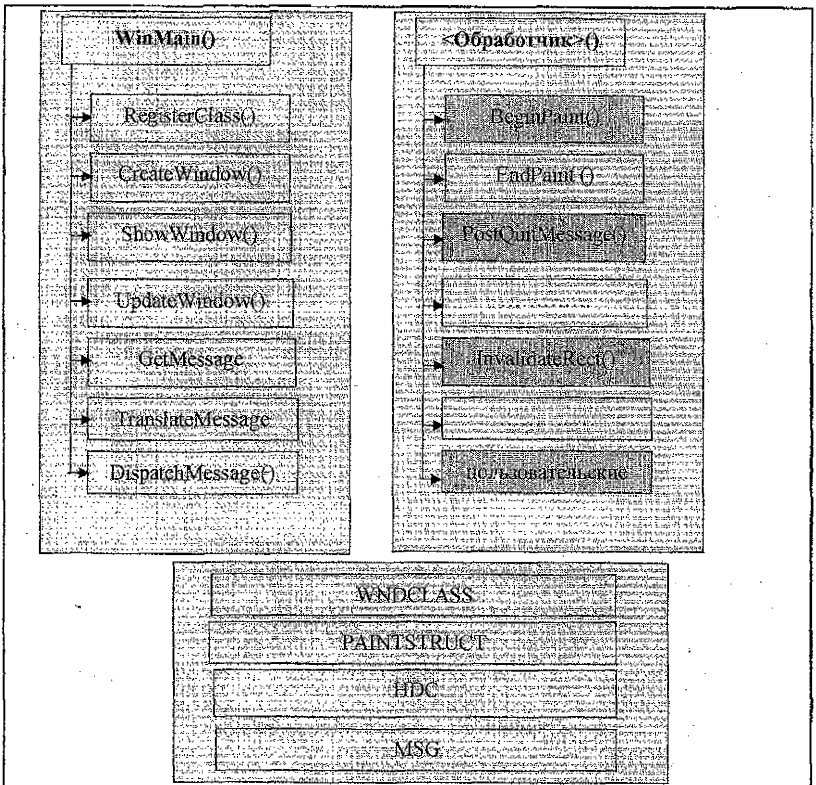


Рис.16. Функциональный состав ТКП

WNDCLASS (в функции WinMain это переменная типа WNDCLASS wcApp). Переменная для обозначения стиля, класса окна, на который можно ссылаться при его инициализации, задается пользователем

char szWindowClass [ ] = <ИмяСтиляОкна> ; .

Например,
char szWindowClass [ ] = "myWindowClass"; .

Пример инициализации структуры WNDCLASS приведен ниже, значения полей структуры указаны в ПРИЛОЖЕНИИ А. Описание полей структуры WNDCLASS. При этом **<ИмяСтиляОкна>** может использоваться для создания конкретных экземпляров окна

Например,

```
WNDCLASS     wcApp;
wcApp.lpszClassName  = myWindowClass;
```

```
wcApp.hInstance = hInst;
wcApp.lpfnWndProc = WndProc;
wcApp.hCursor = LoadCursor (NULL, IDC_ARROW); // курсор окна - стрелка
wcApp.hIcon = 0;  //без пиктограммы при выводе окна в свернутом виде
wcApp.lpszMenuName = 0; // без меню
wcApp.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
wcApp.style = CS_HREDRAW|CS_VREDRAW;// перерисовывать окно при изменении размеров
wcApp.cbClsExtra = 0;
wcApp.cbWndExtra = 0;
```

Пример использования расширенной структуры WNDCLASSEX приведен ниже

```
WNDCLASSEX wcex;
wcex.cbSize = sizeof (WNDCLASSEX);
wcex.style = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc = (WNDPROC) WndProc;
wcex.cbClsExtra = 0;
wcex.cbWndExtra = 0;
wcex.hInstance = hInstance;
wcex.hIcon = LoadIcon ( hInstance, (LPCTSTR) IDI_WORK);
wcex.hCursor = LoadCursor (NULL, IDC_ARROW);
wcex.hbrBackground = (HBRUSH) (COLOR_WINDOW+1);
wcex.lpszMenuName = (LPCSTR) IDC_ WORK;
wcex.lpszClassName = szWindowClass;
wcex.hIconSm = LoadIcon(wcex.hInstance, (LPCTSTR) IDI_SMALL);
```

Регистрация окна производится функцией RegisterClass, в которую передается структура данных типа WNDCLASS, описывающая стиль окна. Прототип функции

```
Unsigned short__cdecl RegisterClass (tagWNDCLASS * <СтруктураДанныхОкна>)
```

Например

```
if ( !RegisterClass (&wcApp) )
        return 0;
```

4.9. Создание и визуализация окна

Когда стиль (класс, вид) окна описан и зарегистрирован в Windows под соответствующим именем (здесь имя задано строкой – szWindowClass), то может быть создано произвольное число таких окон и получены их дескрипторы для дальнейшего использования. Это выполняется с помощью функции CreateWindow.

Эта функция конкретизирует вид создаваемого окна в соответствии со своими параметрами, а ОС Windows возвращает его уникальный номер, т.е. дескриптор (handle) – переменную типа HWND hWnd = CreateWindow (...). Параметры CreateWindow, доопределяющие стиль окна, приведены в ПРИЛОЖЕНИИ Д. Значения параметра dwStyle функции CreateWindow (доопределение стиля окна).

Сама визуализация окна выполняется функцией ShowWindow (hWnd, nCmdShow). Значения второго параметра функции приведены в ПРИЛОЖЕНИИ Е. Значения параметра nCmdShow функции ShowWindow (доопределение состояния окна при начальном запуске).

Прототип функции создания окна приложения CreateWindow приведен ниже

```
HWND CreateWindow ( LPSTR    lpszClassName,
                    LPSTR    lpszWindowName,
                    DWORD    dwStyle,
                    int      x,
                    int      y,
                    int      Width,
                    int      Height,
                    HWND     hWndParent,
                    HMENU    hMenu,
                    HANDLE   hInst,
                    LPSTR    lpParam);
```

```
HWND <ДескрипторОкна> =
CreateWindow (LPSTR <ИмяСтиляОкна>
              LPSTR (<ЗаголовокОкна>,
              DWORD <ДополнительныйСтиль2> = WS_OVERLAPPEDWINDOW,
              int <КоординатаЛевогоУглаОкнаX> = CW_USEDEFAULT,
              int <КоординатаЛевогоУглаОкнаY> = CW_USEDEFAULT,
              int <ШиринаОкна> = CW_USEDEFAULT,
              int <ВысотаОкна> = CW_USEDEFAULT,
              HWND <РодительскоеОкно> = (HWND) NULL,
              HMENU <Меню> = (HMENU) NULL,
              HANDLE <ДескрипторПриложения> = hInstance,
              LPSTR <ДопИнформация> = NULL);
```

Здесь

- HWND <ДескрипторОкна> = CreateWindow (...) – дескриптор окна, используемый в функциях управления окном, например ShowWindow;
- LPSTR lpszClassName = <ИмяСтиляОкна> - имя зарегистрированного стиля, класса окна;
- LPSTR lpszWindowName - заголовок (название) окна;
- DWORD dwStyle <ДополнительныйСтиль2> - доопределяет класс, стиль окна (например, параметр WS_OVERLAPPEDWINDOW обеспечивает создание перекрывающегося окна с системным типовым меню, типовыми кнопкам);
- int x – задает начальное положение окна по оси X;
- int y – задает начальное положение окна по оси Y;
- int Width - задает ширину окна в единицах устройства;
- int Height - задает высоту окна в единицах устройства;
- HWND hWndParent - указатель на родительское окно (у перекрывающегося окна его нет);
- HMENU hMenu - указывает на меню окна;
- HANDLE hInst = <ДескрипторПриложения> - определяет копию модуля, связанного с окном (модуля, который создал окно);
- LPSTR lpParam - адрес дополнительной информации, нужной для создания окна.

Например,

```
hWnd = CreateWindow( szWindowClass, "ПРИМЕР", WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);  .
```

Визуализация окна производится функцией ShowWindow с прототипом

```
int__cdecl ShowWindow (HWND hWnd,  int nCmdShow)  .
```

```
int__cdecl  ShowWindow (HWND <ДескрипторОкна>,
                        int <ИзображениеОкнаПриПервомВыводе>)
```

Здесь
- HWND hWnd - <ДескрипторОкна>, которое надо вывести на экран;
- int nCmdShow - способ <ИзображенияОкнаПриПервомВыводе>, чье значение либо берется из соответствующего параметра, переданного ОС в функции WinMain, либо переустанавливается пользователем по его усмотрению.

Для диалоговых окон аналогом функций CreateWindow и ShowWindow служит функция DialogBox. Она инициализирует диалоговое окно, связывая его ресурсы (**<ID_РесурсовДиалоговогоОкна>**) с приложением hInstance (**<ДескрипторПриложения>**), родительским окном hWnd (**<ДескрипторРодительскогоОкна** >) и обработчиком сообщений диалогового окна (**<ИмяОбработчикаОкна>**). Прототип функции

```
DialogBox  (HANDLE hInstance,
            LPCSTR lpszDialogName,
            HWND hWnd,
            DLGPROC lpfnDlgProc);
```

```
DialogBox (HANDLE <ДескрипторПриложения>,
           LPCSTR <ДескрипторРесурсовДиалоговогоОкна>,
           HWND <ДескрипторРодительскогоОкна>,
           DLGPROC <ИмяОбработчикаОкна>);
```

Здесь
- HANDLE hInstance - **<ДескрипторПриложения>**;
- LPCSTR lpszDialogName - **<ID_РесурсовДиалоговогоОкна>**;
- HWND hWnd - <ДескрипторОкна >;
- DLGPROC lpfnDlgProc - **<ИмяОбработчикаОкна>**.

Например,

```
DialogBox( hInst, (LPCTSTR) IDD_DIALOG1, hWnd,
          (DLGPROC) TO_PROCESS_DIALOG_BOX_MESSAGES);
DialogBox( hInst, (LPCTSTR) IDD_ABOUTBOX, hWnd, (DLGPROC) About);  .
```

4.10. Управление окном приложения

Обновление (перерисовка) окна производится функцией UpdateWindow, т.к. она генерирует сообщение WM_PAINT

int__cdecl UpdateWindow (HWND hWnd);

получение контекста устройства для окна производится функцией

HDC BeginPaint (hWnd, &ps);

освобождение контекста устройства

EndPaint (hWnd, &ps);

обновление клиентской области окна (посылка сообщения на перерисовку) вызывается функцией

ValidateRect (hWnd, NULL) .

4.11. Завершение работы с окном приложения

Работа с окном приложения завершается функцией (при этом посылается сообщение WM_DESTROY)

int__cdecl DestroyWindow (HWND hWnd); .

Если закрытие окна должно привести и к завершению работы с приложением, то можно обработать сообщение WM_DESTROY с помощью функции (при этом посылается сообщение WM_QUIT)

VOID __cdecl PostQuitMessage (int);

Закрытие диалогового окна производится функцией

int__cdecl EndDialog (HWND hWnd, int <КодЗавершения>); .

5. РАЗРАБОТКА И ИСПОЛЬЗОВАНИЕ ПРИЛОЖЕНИЙ НА БАЗЕ ТИПОВОГО КАРКАСА ОКОННОГО ПРИЛОЖЕНИЯ (ТКП)

5.1. Общая схема технологии разработки приложений на базе ТКП

Приложение при проектировании может рассматриваться как совокупность однотипных компонентов, включающих окна и их графические ресурсы (описания окон и их элементов) и программное обеспечение в виде функций-обработчиков (описания функций, реализующих реакцию на сообщения и выполняющих необходимую предметную обработку).

РАЗРАБОТКА приложения включает следующие этапы.

1. Проектирование.
1.1. Проектирование интерфейсного компонента.
1.2. Проектирование программного компонента.
2. Реализация.
2.1. Реализация интерфейсных компонентов.
2.1.1. Создание файла описания ресурсов.

2.1.2. Кодирование интерфейсных компонентов.

2.1.3. Подключение к приложению файла описания ресурсов.

2.2. Реализация (кодие) провантограммных компонентов. Инициализация и визуализация интерфейсных компонентов.

5.2. Характеристика этапов разработки приложений на базе ТКП

1. Проектирование. Результат – спецификация компонентов приложения.

1.1. Проектирование интерфейсного компонента. Включает проектирование окна и его элементов. Результат - вид (облик), состав элементов (кнопок, меню, и т.д.), их свойства (атрибуты), события, сообщения, спецификация реакций на сообщения.

Для **окна** (главного окна) проектируется пользовательское меню, определяются ресурсы окна (кисти, перья, пиктограммы и т.п.).

Для **диалогового окна** проектируются элементы управления (кнопки, поля редактирования и т.п.).

1.2. Проектирование программного компонента (программного обеспечения). Результат – алгоритмы работы функций-обработчиков по спецификациям реакций на сообщения.

2. Реализация. Результат – исполнимый код приложения.

2.1. Реализация интерфейсных компонентов. Означает описание на соответствующих языках интерфейсных компонентов приложения (графических ресурсов).

2.1.1. Создание файла описания ресурсов.

2.1.2. Кодирование интерфейсных компонентов. Означает описание на соответствующих языках интерфейсных компонентов приложения (графических ресурсов).

Для **окна** (главного окна) создается стиль окна с названием **<ИмяСтиляОкна>**. В дальнейшем зарегистрированный стиль окна с названием **<ИмяСтиляОкна>** может использоваться для создания и визуализации конкретного окна (окон). А после его создания для управления окном в функциях используется его **< ДескрипторОкна >**. Для создания стиля окна:

- кодируются ресурсы, при необходимости - пользовательское меню с идентификатором **<ИмяРесурсаМеню>**. Ресурсы кодируются с помощью специальных команд описания или визуально в редакторе ресурсов. Фиксируются <ID_идентификаторы> ресурсов, например, исполнимых пунктов меню (кнопок, команд);

- определяется имя функции - **<ИмяОбработчикаОкна>**;

- описывается стиль окна с названием **<ИмяСтиляОкна>**, путем задания значений атрибутов окна в полях структуры данных описания стиля окна типа **WNDCLASS**, где в том числе указываются **<ДескрипторПриложения>** (как параметр функции WinMain), **<ИмяСтиляОкна>**, **<ИмяРесурсаМеню>**, **<ИмяОбработчикаОкна>**, <Пиктограмма>, <КистьФона> и т.п.;

- стиль окна связывается с его обработчиком путем указания его названия **<ИмяОбработчикаОкна>** в соответствующем поле структуры данных описания окна **WNDCLASS**;

- стиль окна регистрируется в ОС функцией RegisterClass (tagWNDCLASS *), в которую передается ссылка на заполненную структуру данных описания стиля окна типа **WNDCLASS**.

Для **диалогового окна** описываются составляющие его ресурсы с названием **<ID_РесурсовДиалоговогоОкна>**. В дальнейшем название созданных **<ID_РесурсовДиалоговогоОкна>** может использоваться для создания и визуализации конкретного окна (окон). А после его создания для управления окном в функциях используется его **< ДескрипторОкна >**. Для создания ресурсов диалогового окна:

- выполняется описание его элементов управления с помощью команд описания или визуально в редакторе ресурсов;

- в редакторе ресурсов определяется название диалогового окна **<ID_РесурсовДиалоговогоОкна>** и <ID_идентификаторы> его элементов управления (кнопок, списков и т.п.).

2.1.3. Подключение к приложению файла описания ресурсов.

2.2. Реализация (кодирование) программных компонентов. Означает описание на соответствующих языках программных компонентов приложения - функций-обработчиков. В том числе этот этап предполагает выполнение таких действий, как инициализация и визуализация интерфейсных компонентов. Означает задание значений дополнительных атрибутов, связывание с функциями-обработчиками и как результат – получение **< ДескриптораОкна >**, вывод окна на экран.

Для **окна** (главного окна) связывание с функциями-обработчиками было выполнено при описании стиля окна. Поэтому здесь только уточняются параметры инициализации - <ЗаголовокОкна>, < КоординатаЛевогоВерхнегоУглаОкна >, < ШиринаОкна >, < ВысотаОкна >, стиль окна. Выполняется указанное с помощью функции

HWND **<ДескрипторОкна>** = **CreateWindow** (LPSTR **<ИмяСтиляОкна>**, ...).

Окно визуализируется функцией

ShowWindow (<ДескрипторОкна>, <ИзображениеОкнаПриПервомВыводе>).

Для **диалогового окна** инициализация, включая связывание ресурсов (**<ID_РесурсовДиалоговогоОкна>**) с приложением hInstance (**<ДескрипторПриложения>**), родительским окном hWnd (**<ДескрипторРодительскогоОкна** >) и обработчиком сообщений диалогового окна (**<ИмяОбработчикаОкна>**), а также визуализация производится одной функцией (аналогом функций CreateWindow и ShowWindow)

DialogBox (<ДескрипторПриложения>, <ДескрипторДиалоговогоОкна>,
<ДескрипторРодительскогоОкна >, <ИмяОбработчикаОкна>); .

5.3. Общая схема описания, реализации главного окна ТКП

Окно является основным компонентом интерфейса приложения. С понятием окно связаны графические ресурсы окна (пиктограмма, кисть, меню и т.п.) и функция-обработчик сообщений окна. Создание и работа с окнами базируется на понятии стиль (класс) окна. Основные этапы представлены ниже.

1. Пользователь по своему усмотрению определяет переменную для обозначения стиля, класса окна, на который можно ссылаться при его инициализации

char szWindowClass [] = **<ИмяСтиляОкна>** ; .

2. Атрибуты стиля задаются в структуре данных **WNDCLASS** для хранения информации о стиле окна. После регистрации стиля окна в ОС **<ИмяСтиляОкна>** может использоваться для создания конкретных экземпляров окна. В структуре данных **WNDCLASS**, в частности, задается параметр, доопределяющий класс, стиль окна (например, параметр CS_VREDRAW обеспечивается перерисовку содержимого клиентской области окна при

изменении размера окна по вертикали). См. ПРИЛОЖЕНИЕ Г. Значения параметра Style структуры WNDCLASS.

Для этого описывается переменная

WNDCLASS <СтруктураДанныхОкна>

и задаются атрибуты стиля, **<СтруктураДанныхОкна>**.lpszClassName = (LPCSTR) **<ИмяСтиляОкна>** и т.д.

3. Окно регистрируется в ОС. Регистрация окна производится функцией RegisterClass, в которую передается структура данных типа WNDCLASS, описывающая стиль окна

RegisterClass (tagWNDCLASS * **<СтруктураДанныхОкна>**).

4. Экземпляр окна создается функцией

```
HWND <ДескрипторОкна> = CreateWindow (LPSTR <ИмяСтиляОкна>,
                LPSTR <ЗаголовокОкна>,
                DWORD <ДополнительныйСтиль2> = WS_OVERLAPPEDWINDOW,
                int < КоординатаЛевогоУглаОкнаX > = CW_USEDEFAULT,
                Int < КоординатаЛевогоУглаОкнаY > = CW_USEDEFAULT,
                Int < ШиринаОкна > = CW_USEDEFAULT,
                Int < ВысотаОкна > = CW_USEDEFAULT,
                HWND <РодительскоеОкно > = (HWND)NULL,
                HMENU <Меню> = (HMENU) NULL,
                HANDLE <ДескрипторПриложения> = hInstance,
                LPSTR <ДопИнформация> = NULL);
```

Здесь **<ДополнительныйСтиль2>** – доопределяет класс, стиль окна (например, параметр WS_OVERLAPPEDWINDOW обеспечивает создание перекрывающегося окна с системным типовым меню, типовыми кнопкам). См. ПРИЛОЖЕНИЕ Д. Значения параметра dwStyle функции CreateWindow.

Теперь **<ДескрипторОкна>** можно использовать в функциях управления окном.

5. Визуализация окна производится функцией ShowWindow с прототипом

ShowWindow (HWND **<ДескрипторОкна>**, int **<ИзображениеОкнаПриПервомВыводе>**)

Здесь параметр **<ИзображенияОкнаПриПервомВыводе>**, задает способ его вывода. Значение либо берется из соответствующего параметра - int nCmdShow, переданного ОС в функции WinMain, или переустанавливается пользователем по его усмотрению. см. ПРИЛОЖЕНИЕ Е. Значения параметра nCmdShow функции ShowWindow (доопределение состояния окна при начальном запуске)

Общая схема функционирования приложений на базе ТКП иллюстрируется рисунком 17.

5.4. ТКП. Создание в системе Visual Studio

Ниже описана структура Windows-приложения на базе ТКП с типовым минимальным набором графических ресурсов.

Соответственно интерфейс такого приложения базируется на использовании классического окна с клиентской областью (в качестве главного) с пользовательским стилем (зарегистрированным здесь как – myWindowStyle). Это стиль (окно), определяющий (см. ПРИЛОЖЕНИЕ А. Описание полей структуры WNDCLASS):

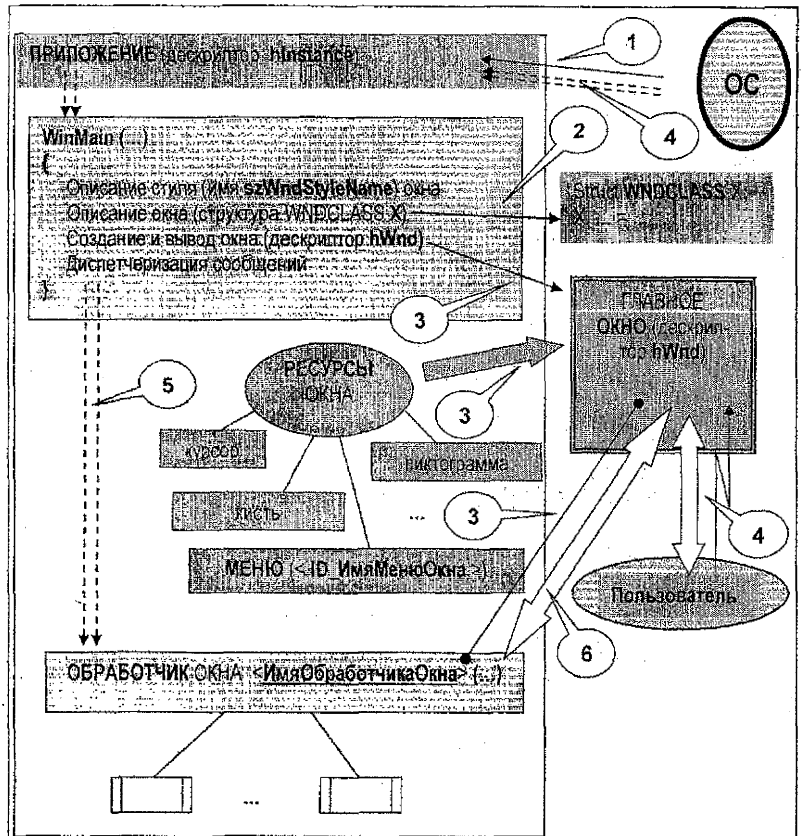


Рис.17. Схема обработки сообщений в ТКП

- отсутствие пользовательского меню; - перерисовку содержимого окна при изменении его размеров (параметры CS_HREDRAW, CS_VREDRAW. См. ПРИЛОЖЕНИЕ Г. Значения параметра Style структуры WNDCLASS); - светлый фон (параметр WHITE_BRUSH. См. ПРИЛОЖЕНИЕ В. Встроенные кисти); - курсор типа "стрелка" (параметр IDC_ARROW. См. ПРИЛОЖЕНИЕ Б. Встроенные курсоры). С окном связан обработчик сообщений (здесь с именем WndProc).

При создании окна (в функции CreateWindow) дополнительно указаны:

- заголовок ("ТИПОВОЙ КАРКАС Windows-приложения ... "); - стиль - перекрывающееся окно с системным меню и системными кнопками (параметр WS_OVERLAPPEDWINDOW). См. ПРИЛОЖЕНИЕ Д. Значения параметра dwStyle функции CreateWindow (доопределение стиля окна).

При визуализации окна в параметрах функции ShowWindow (hWnd, nCmdShow) указан вывод окна в развернутом виде. См. ПРИЛОЖЕНИЕ Е. Значения параметра

nCmdShow функции ShowWindow (доопределение состояния окна при начальном запуске). Соответственно окно выглядит как показано ниже.

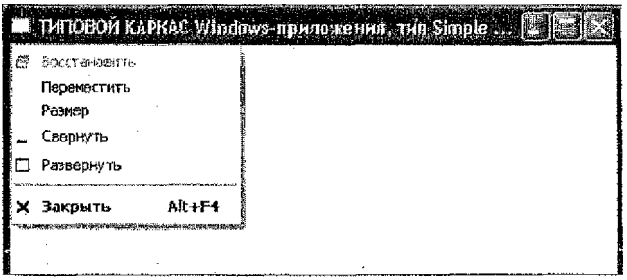


Рис.18. Интерфейс ТКП

Текст приложения приведен ниже.

```
#include <windows.h>        // описания Windows
LRESULT CALLBACK      WndProc (HWND, UINT, WPARAM, LPARAM);
char  szWindowStyle [ ] = "myWindowStyle";

//=================================
//  ГЛАВНАЯ ФУНКЦИЯ
int  WINAPI  WinMain  (HINSTANCE hInst,  //дескриптор приложения
                       HINSTANCE hPreInst,  //всегда NULL
                       LPSTR       lpszCmdLine,  //командная строка
                       int           nCmdShow) //окно при первом выводе
{
    HWND                 hWnd;          //дескриптор окна
    MSG                  lpMsg;         //структура хранения сообщений
    WNDCLASS             wcApp;         //структура описания стиля окна
    wcApp.lpszClassName  = szWindowStyle;  //имя стиля окна
    wcApp.hInstance      = hInst;       //дескриптор приложения
    wcApp.lpfnWndProc    = WndProc;     //указатель на обработчик сообщений
    wcApp.hCursor        = LoadCursor(NULL, IDC_ARROW);   //курсор - "стрелка"
    wcApp.hIcon          = 0;           //без использования пиктограммы
    wcApp.lpszMenuName   = 0;           //дескриптор меню (без меню)
    wcApp.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH); //цвет фона
    wcApp.style          = CS_HREDRAW | CS_VREDRAW; //перерисовывать окно
    wcApp.cbClsExtra     = 0;           //число доп. байт для WNDCLASS
    wcApp.cbWndExtra     = 0;           //общее число доп. байт

    if   ( ! RegisterClass (&wcApp) )  //регистрация окна
           return  0;

   hWnd = CreateWindow (szWindowStyle,
                "ТИПОВОЙ КАРКАС Windows-приложения ... ",
                 WS_OVERLAPPEDWINDOW, //окно перекрывающееся, меню, кнопки
                  CW_USEDEFAULT,      //координата x - левый верхний угол окна
```

```
                    CW_USEDEFAULT,      //координата y - левый верхний угол окна
                    CW_USEDEFAULT,      //ширина окна в единицах устройства
                    CW_USEDEFAULT,      //высота окна в единицах устройства
                    (HWND) NULL,        //указатель на родительское окно
                    (HMENU) NULL,       //зависит от стиля окна (указатель меню)
                    hInst,              //дескриптор приложения
                    NULL );     //адрес дополнительной информации об окне
    ShowWindow (hWnd, nCmdShow);    //вывод окна
    UpdateWindow (hWnd);            //перерисовка окна
    while (GetMessage ( &lpMsg, NULL, 0, 0) )
    {
        TranslateMessage (&lpMsg);      //преобразование виртуальных клавиш
        DispatchMessage (&lpMsg);       //передача сообщения обработчику
    }
    return ( lpMsg.wParam );
}

//================================
//  ФУНКЦИЯ-ОБРАБОТЧИК ГЛАВНОГО ОКНА. (имя выбирает пользователь)
LRESULT CALLBACK WndProc (HWND hWnd,        //дескриптор окна
                          UINT messg,       //код сообщения
                          WPARAM wParam, LPARAM lParam)
{
    HDC         hdc;        //дескриптор контекста устройства
    PAINTSTRUCT ps;         //структура для клиентской области окна
    switch (messg)
    {
        case WM_PAINT:              //перерисовать окно
            hdc = BeginPaint (hWnd, &ps);
            //-----Начало фрагмента пользователя
            //-----Конец фрагмента пользователя
            ValidateRect (hWnd,NULL);
            EndPaint (hWnd, &ps);
            break;
        case WM_DESTROY:    //послать сообщение о завершении приложения
            PostQuitMessage (0);
            break;
        default:
            return ( DefWindowProc (hWnd, messg, wParam, lParam));
    }
    return 0;
}
```

Аналогичный по функциональности и интерфейсу текст можно получить автоматически, используя мастер построения каркасов системы Visual Studio - Win32 Application (Simple) или Application (Hello). Однако в последнем случае он будет отличаться от приведенного здесь структурированием – составом модулей, функций.

6. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Все практические действия выполняются в системе Visual Studio.

1. Изучить теоретический материал.

2. Создать каркас (EX1) Windows-приложения (тип Empty) средствами мастера Win32 Application. Изучить его свойства (привести в ОТЧЕТЕ): - состав интерфейса, предоставляемые возможности; - файловый состав (структуру проекта, дерево папок, состав, назначение файлов и их соподчиненность по включению); - функциональный состав (привести схему иерархии функций приложения, описать назначение и прототипы функций).

Для этого в системе Visual Studio создать новый проект (New Project):

- выбрать пункт меню File-New, в появившемся диалоговом окне выбрать вкладку проекты Projects и выбрать пункт Win32 Application, т.е. создание Windows-приложения;

- набрать имя проекта (здесь EX1) в строке Project name, установить переключатель типа проекта в An empty project (пустой проект), завершить создание каркаса (в результате будет создан проект с пустыми папками, дерево проекта можно увидеть в окне рабочего пространства – вкладка FileView) и запустить приложение.

3. Создать проект (EX1) из одного файла с одной, главной функцией, выводящей сообщения с помощью функций типа MessageBox (результаты привести в ОТЧЕТЕ). Для этого доработать каркас - создать файл в папке Source Files проекта приложения:

- выбрать пункт меню File-New, в появившемся диалоговом окне выбрать вкладку файлы Files и пункт C++ Source File, т.е. создание файла с текстом на языке С;

- набрать имя файла в поле File name и завершить его создание (в результате будет создан пустой файл, содержимое которого можно увидеть на вкладке FileView);

- включить в созданный файл заголовок #include <windows.h>, создать пустую функцию WinMain, например, как показано ниже, и выполнить приложение

```
int  WINAPI  WinMain (HINSTANCE H1, HINSTANCE H2, LPSTR Str, int I)
{
    return  0;
}   ;
```

- включить в приложение вывод окна сообщения, например, MessageBox (NULL, "Работает приложение", "Информационное сообщение", 1) и снова выполнить приложение;

- включить в приложение функцию из библиотеки MFC, для чего заменить заголовочный файл <windows.h> на файл <afxwin.h>, выполняющий роль шлюза для доступа к классам MFC, и подключить библиотеки через главное меню - Project-Settings-General-Use MFC;

- включить в приложение вывод окна сообщения, например, AfxMessageBox ("Вот ОНО и заработало !", 1) и выполнить приложение.

4. Создать проект (EX2) из двух файлов – один, главный (с расширением cpp) с текстом программы - функцией WinMain, второй - заголовочный (с расширением h) с командами препроцессора. Для этого повторить п.3, модифицировать приложение, заголовочный файл подключить к главному командой препроцессора #include "*.h". Выполнить приложение.

5. Создать каркас (EX3) Windows-приложения (тип Simple) средствами мастера Win32 Application (аналогично п.2). Изучить его свойства: - интерфейс приложения; - файловый состав; - функциональный состав; - ресурсный состав (результаты привести в ОТЧЕТЕ).

6. Создать проект (EX3), модифицировав созданный каркас путем вставки в WinMain функций MessageBox (аналогично п.3).

7. Создать приложение (EX4) на базе типового каркаса приложения ТКП, взяв за основу автоматический каркас мастера Win32 Application типа Simple или Hello. Изучить его свойства (привести в ОТЧЕТЕ): - интерфейс приложения; - файловый состав; - функциональный состав; - ресурсный состав. Создать приложение, например, используя первый способ: создать каркас-проект (EX4), тип Simple (см. п.5). В основной файл проекта (EX4.cpp) вместо имеющегося там текста вставить текст ТКП (см. § 5.4 или ПРИЛОЖЕНИЕ 7), сохранив только команду #include "stdafx.h". Выполнить приложение.

8. Создать приложение (EX5) модифицировав приложение на базе ТКП. Изменить его интерфейс (вид окна, состав элементов окна - кнопок и т.п.) (результаты привести в ОТЧЕТЕ). Для этого создать прилоэение (EX5) на базе ТКП (см. п.7) и внести изменения в параметры:

- структуры WNDCLASS (см. ПРИЛОЖЕНИЕ А. Описание полей структуры WNDCLASS, ПРИЛОЖЕНИЕ Б. Встроенные курсоры, ПРИЛОЖЕНИЕ В. Встроенные кисти, ПРИЛОЖЕНИЕ Г. Значения параметра Style);

- функции CreateWindow (см. ПРИЛОЖЕНИЕ Д. Значения параметра dwStyle). Например, добавить в окно рамки, полосы прокрутки, убрать или добавить командные кнопки. Внести изменения в координаты и размеры окна, использовав соответствующие параметры функции (x, y, Width, Height);

- функции ShowWindow (см. ПРИЛОЖЕНИЕ Е. Значения параметра nCmdShow).

9. Создать приложение (EX6) на базе ТКП. Добавить строку приветствия (координаты 0, 0). Для этого вставить в обработчик пользовательский фрагмент, например

```
char  szText [25] = "Работает ТКП" ;
TextOut ( hdc, 0, 0, szText, strlen ( szText ) );  .
```

Организовать вывод данных, например,

```
int  X = 2007; char szXasString[100];
wsprintf ( szXasString, "Значение X - %d", X );
TextOut ( hdc, 0, 130, szXasString, strlen ( szXasString ) );  .
```

Вывести в столбик для значений X от 1 до 10 пары значений: X - квадрат X.

При каждой перерисовке окна увеличивать и выводить номер перерисовки ReDrawNumber++ . Здесь int ReDrawNumber = 0 - глобальная переменная.

10. Создать приложение (EX7) на базе ТКП. Протабулировать заданную функцию и вывести ее значения в виде таблицы.

11. Создать приложение (EX8) на базе ТКП. Включить чувствительность приложения к нажатию левой клавиши "мыши" (сообщению WM_LBUTTONDOWN). Выводить с соответствующим сообщением MessageBox и выполнять задачи п.9-10.

12. Повторить п.5-11 на базе каркаса типа Hello.

ЛИТЕРАТУРА

1. Б. Страуструп. Язык программирования СИ++. – М.: Радио и связь, 1991. – 352 с.
2. Паппас К., Мюррей У. Visual C++. Руководство для профессионалов: Пер. с англ. - СПб: BHV -Санкт-Петербург, 1996.
3. Орлов С.А. Технологии разработки программного обеспечения: Учебник для вузов. – СПб.: Питер, 2004. – 527 с.
4. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на С++, 2-е изд./ Пер. с англ.. – М.: Издательство БИНОМ, СПб: Невский диалект, 1998. – 560 с.
5. Соммервилл И. Инженерия программного обеспечения. – СПб, 2003.
6. Уилсон С. Принципы проектирования и разработки программного обеспечения. Учебный курс. – СПб, 2003.
7. Шилдт Г. Справочник программиста по C/C++, 3-е изд. – М.: Изд. Дом Вильямс, 2003. – 432 с.
8. Одинцов И. Профессиональное программирование. Системный подход. – СПб.: БХВ-Петербург, 2002. – 512 с.
9. Паппас К., Мюррей У. Эффективная работа: Visual C++.NET. – СПб.: Питер, 2002. – 816 с.
10. Франка П. С++: учебный курс. – СПб.: Питер, 2005. – 522 с.
11. Мэтт П. Внутренний мир Windows: Пер. с англ.- Киев: НИПФ ДиаСофтЛТД, 1995.

ПРИЛОЖЕНИЕ 1. Описание полей структуры WNDCLASS

Таблица П1.1 Поля структуры WNDCLASS

Имя поля	Значение
Style	определяет стиль класса. Различные стили могут комбинироваться при помощи побитовой операции ("логическое ИЛИ")
lpfnWndProc	указатель на функцию-обработчик окна
cbClsExtra	число дополнительных байтов, выделяемых структуре WNDCLASS (обычно NULL)
cbClsExtra	число дополнительных байтов, выделяемых для всех дополнительных структур окна (обычно NULL)
hinstance	дескриптор экземпляра приложения, регистрирующий стиль окна
hIcon	пиктограмма окна в свернутом виде (обычно NULL)
hCursor	тип курсора, используемый для данного окна (обычно NULL)
hbrBackGround	тип кисти, используемой для фона окна (значением данного параметра может быть как идентификатор физической кисти, так и значение цвета)
lpszMenuName	указатель строки – имени, дескриптора ресурса - меню (по умолчанию NULL)
lpszClassName	указатель строки – уникального имени класса окна

ПРИЛОЖЕНИЕ 2. Значения параметра hCursor структуры WNDCLASS (встроенные курсоры)

Таблица П2.1. Некоторые стандартные курсоры

Имя курсора	Форма курсора
IDC_ARROW	стандартный указатель типа "стрелка"
IDC_CROSS	указатель типа "перекрестие"
IDC_IBEAM	текстовый курсор
IDC_WAIT	указатель типа "песочные часы"

ПРИЛОЖЕНИЕ 3. Значения параметра hbrBackGround структуры WNDCLASS (встроенные кисти и цвета закраски фона окна)

Таблица П3.1. Некоторые стандартные кисти

Имя кисти, цвет кисти	Тип фона
BLACK_BRUSH	черный
DKGRAY_BRUSH	темно-серый
HOLLOW_BRUSH	прозрачный
LTGRAY_BRUSH	светло-серый
WHITE BRUSH	белый
COLOR_SCROLLBAR, COLOR_BACKGROUND, COLOR_ HIGHLIGHT, COLOR_ACTIVECAPTION, COLOR_HIGHLIGHTTEXT, COLOR_WINDOW, COLOR_WINDOWTEXT, COLOR_CAPTIONTEXT, COLOR_APPWORKSPACE	при использовании цвета к его значению нужно прибавить 1

ПРИЛОЖЕНИЕ 4. Значения параметра Style структуры WNDCLASS

Таблица П4.1. Значения параметра Style

Параметр	Значение
CS_BYTEALIGNCLIENT (CS_BYTEALIGNWINDOW)	использовать границы по байту по оси X и производить выравнивание клиентской области окна (использовать границы по байту по оси X и производить выравнивание окна)
CS_DBCLKS	посылать окну сообщения о двойном щелчке "мыши"
CS_HREDRAW	перерисовывать содержимое клиентской области окна при изменении размера окна по горизонтали
CS_KEYCVTWINDOW	выполнять преобразование виртуальных клавиш
CS_NOCLOSE	без пункта закрытия окна в системном меню
CS_NOKEYCVT	не выполнять преобразование виртуальных клавиш
CS_OWNDC	присваивать каждому экземпляру окна свой контекст устройства
CS_PARENTDC	окну передавать контекст устройства родительского окна
CS_SAVEBITS	сохранять часть окна, перекрытую на экране другим окном
CS_VREDRAW	перерисовывать окно при изменении размера по вертикали

ПРИЛОЖЕНИЕ 5. Значения параметра dwStyle функции CreateWindow (определение стиля окна)

Таблица П5.1. Значения параметра dwStyle

Параметр стиля	Описание
WS_BORDER	создание окна с рамкой
WS_CAPTION	добавление к окну с рамкой заголовка
WS_CHILD	создание дочернего окна
WS_CHILDWINDOW	создание дочернего окна стиля WS_CHILD
WS_CLIPCHILDREN	при создании родительского окна запрещает его рисование в области, занятой любым дочерним окном
WS_CLIPSIBLINGS	(только со стилем WS_CHILD) не использовать при получении данным окном сообщения о перерисовке области, занятые другими дочерними окнами. Иначе разрешается рисовать в клиентской области другого дочернего окна
WS_DISABLED	создание первоначально неактивного окна
WS_DLGFRAME	создание окна с двойной рамкой без заголовка
WS_EX_ ACCEPTFILES	создание окна, поддерживающего метод ""drag-and-drop"
WS_EX_ DLGMODALFRAME	(только для значения dwExStyle) создание окна с двойной рамкой и дополнительным заголовком
WS_EX_ NOPARENTNOTIFY	создание дочернего окна, которое не будет посылать родительскому окну сообщение WM_PARENTNOTIFY при создании или разрушении

WS_EX_TOPMOST	созданное окно должно располагаться поверх всех окон, не имеющих этого стиля, и оставаться поверх остальных даже в неактивном состоянии
WS_EX_ TRANSPARENT	созданное окно должно быть прозрачным, т.е. все окна, расположенные под данным окном, не заслоняются им
WS_GROUP	(только в диалоговых окнах) стиль указывается для первого элемента управления в группе элементов, пользователь перемещается между ними с помощью клавиш-стрелок
WS_HSCROLL	создание окна с горизонтальной полосой прокрутки
WS_MAXIMIZE WS_MINIMIZE	создание окна максимального (минимального) размера
WS_MAXIMIZEBOX WS_MINIMIZEBOX	создание окна с кнопкой максимизации (минимизации)
WS_OVERLAPPED	создание перекрывающегося окна
WS_OVERLAPPED WINDOW	создание перекрывающегося окна на базе стилей WS_OVERLAPPED, WS_THICKFRAME и WS_SYSMENU
WS_POPUP	(не используется с окнами стиля WS_CHILD) создание временного окна
WS_ POPUPWINDOW	создание временного окна на базе стилей WS_BORDER, WS_POPUP и WS_SYSMENU
WS_SYSMENU	(только для окон с заголовком) создание окна с кнопкой вызова системного меню вместо стандартной кнопки, позволяющей закрыть окно при использовании этого стиля для дочернего окна
WS_TABSTOP	(только в диалоговых окнах) указывает на произвольное количество элементов управления (стиль WS_TABSTOP), между которыми можно перемещаться клавишей <Tab>
WS_THICKFRAME	создание окна с толстой рамкой, используемой для изменения размеров окна
WS_VISIBLE	создание окна, видимого сразу после создания
WS_VSCROLL	создание окна с вертикальной полосой прокрутки

ПРИЛОЖЕНИЕ 6. Значения параметра nCmdShow функции ShowWindow (доопределение состояния окна при начальном запуске)

Таблица П6.1. Состояния окна при запуске (параметры в ShowWindow)

Параметр состояния	Описание
SW_HIDE	прячет окно и переводит в активное состояние другое окно
SW_MIINIMIZE	минимизирует окно и активизирует окно верхнего уровня
SW_RESTORE	действует так же, как и SW_SHOWNORMAL
SW_SHOW	активизирует окно и выводит его в текущей позиции и текущего размера
SW_SHOWDEFAULT	активизирует окно и выводит его с использованием текущих умолчаний
SW_SHOWMAXIMIZED	активизирует окно и выводит его с максимальным размером

SW_ SHOWMINIMIZED	активизирует окно и выводит его в виде пиктограммы
SW_SHOWMINNOACTIVATE	выводит окно как пиктограмму, при этом ранее активное окно сохраняет свою активность
SW_SHOWNA	выводит окно с учетом его состояния в данный момент; (активное в данный момент окно остается активным)
SW_SHOWNOACTIVATE	выводит окно в его прежней позиции и прежнего размера (активное в данный момент окно остается активным)
SW_SHOWNORMAL	активизирует окно и выводит его на экран. Если окно было увеличено или уменьшено до пиктограммы, то система Windows восстановит начальное положение и размер окна
SW_SHOWSMOOTH	Выводит окно так, чтобы оно минимально перекрывалось другими окнами

ПРИЛОЖЕНИЕ 7. Описание ТКП

```
//============ описания глобальных объектов =============
LRESULT CALLBACK <ИмяОбработчикаОкна> (HWND, UINT, WPARAM, LPARAM);
char  szWindowStyle [ ] = <ИмяСтиляОкна>;

//================ГЛАВНАЯ ФУНКЦИЯ====================
int  WINAPI  WinMain (HINSTANCE hInst,  HINSTANCE hPreInst,
                      LPSTR  lpszCmdLine,   int  nCmdShow)
{
    HWND                hWnd;
    MSG                 lpMsg;
    WNDCLASS            wcApp;
    wcApp.lpszClassName  = szWindowStyle;
    wcApp.hInstance      = hInst;
    wcApp.lpfnWndProc    = <ИмяОбработчикаОкна>;
    wcApp.hCursor        = LoadCursor (NULL, IDC_ARROW);
    wcApp.hIcon          = 0;
    wcApp.lpszMenuName   = 0;
    wcApp.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH);
    wcApp.style          = CS_HREDRAW | CS_VREDRAW;
    wcApp.cbClsExtra     = 0;
    wcApp.cbWndExtra     = 0;
    if  ( !RegisterClass (&wcApp))  return  0;
    hWnd = CreateWindow  (szWindowStyle, "КАРКАС ПРИЛОЖЕНИЯ",
                          WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, (HWND) NULL,
                          ( HMENU )NULL,  hInst,  NULL);
    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);
    while ( GetMessage (&lpMsg, NULL, 0, 0) )
```

```
    {
        TranslateMessage (&lpMsg);
        DispatchMessage  (&lpMsg);
    }
    return (lpMsg.wParam);
}

//=== ФУНКЦИЯ-ОБРАБОТЧИК СООБЩЕНИЙ ГЛАВНОГО ОКНА ===
LRESULT CALLBACK <ИмяОбработчикаОкна>(HWND hWnd, UINT Mess,
                                      WPARAM wParam, LPARAM lParam)
{
    HDC         hdc;
    PAINTSTRUCT ps;
    switch (Mess)
    {
        case WM_PAINT:
            hdc = BeginPaint (hWnd, &ps);
            //< ФРАГМЕНТ ПОЛЬЗОВАТЕЛЯ >
            ValidateRect (hWnd,NULL);
            EndPaint (hWnd, &ps);
            break;
        //case < КодСообщения > :
            //hdc = BeginPaint (hWnd, &ps);
            //< ФРАГМЕНТ ПОЛЬЗОВАТЕЛЯ >
            //ValidateRect (hWnd,NULL);
            //EndPaint (hWnd, &ps);
            //break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, Mess, wParam, lParam);
    }
    return 0;
}
```

СОДЕРЖАНИЕ

Учебное издание

Муравьев Геннадий Леонидович

методическое пособие

“ ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ . Процедурный стиль”, Часть 1

Ответственный за выпуск: *Муравьев Г.Л.*
Редактор: *Строкач Т.В.*
Компьютерный набор: *Муравьев Г.Л.*
Компьютерная вёрстка: *Кармаш Е.Л.*
Корректор: *Никитчик Е.В.*

Подписано в печать 06.02.2007 г. Формат 60x80 1/16
Бумага писч. Усл. п. л. 2,79. Уч. изд. л. 3. Тираж 100 экз. Заказ № 124
Отпечатано на ризографе УО “Брестский государственный технический университет”
224017, Брест, ул. Московская, 267