# Модуль №1

## Занятие №1

Версия 1.0.1

**План занятия:**

1. UNICODE
   1.1. Понятие кодировки. Unicode кодировка.
   1.2. Unicode и библиотека C++ (C/C++ Run-Time Library).
   1.3. Макросы для работы с Unicode.
   1.4. Строковые функции Windows.
   1.5. Перекодировка строк из ANSI в Unicode.
   1.6. Перекодировка строк из Unicode в ANSI.
2. Программирование Windows-приложений. Введение.
   2.1. Графический интерфейс пользователя.
   2.2. Многозадачность и многопоточность.
   2.3. Управление памятью.
   2.4. Независимость от аппаратных средств.
3. Событие. Сообщение. Очередь сообщений.
4. Окно.
5. Подведение итогов. Домашнее задание.

## 1. UNICODE

### 1.1. Понятие кодировки. Unicode кодировка

Рассмотрение данного вопроса необходимо начать с понятия кодировки. Привести следующее определение кодировки:

**Кодировка – это определённый набор символов (англ. *character set*), в котором каждому символу сопоставляется последовательность длиной в один или несколько байтов.**

Следует отметить, что в обычной кодировке **ANSI** (*American National Standards Institute*) каждый символ определяется восемью битами, т.е. всего можно закодировать 256 символов. Однако существуют такие языки и системы письменности (например, японские иероглифы), в которых столько знаков, что однобайтового набора символов недостаточно. Для поддержки таких языков и для облегчения «перевода» программ на другие языки была создана кодировка **Unicode**. Каждый символ в Unicode состоит из двух байтов, что позволяет расширить набор символов до 65536. Существенное отличие от 256 символов, доступных в ANSI–кодировке! Таким образом, стандарт Unicode позволяет представить символы практически всех языков мира, что даёт возможность легко обмениваться данными на разных языках, а также распространять единственный двоичный EXE- или DLL- файл, поддерживающий все языки.

## 1.2. Unicode и библиотека C++ (C/C++ Run-Time Library)

Продолжая рассмотрение UNICODE-кодировки, ознакомить слушателей с «новым» типом данных, который был введён специально для использования Unicode-строк:

| wchar_t — тип данных для Unicode-символа<br>typedef unsigned short wchar_t |
|---|

Подчеркнуть, что для работы с Unicode-строками существует набор Unicode-функций, которые эквивалентны строковым функциям ANSI C. Привести следующую таблицу соответствий:

| Строковая функция ANSI C | Эквивалентная Unicode-функция |
|---|---|
| char * strcat( char *, const char *); | wchar_t * wcscat( wchar_t *, const wchar t *); |
| char * strchr(const char *, int); | wchar_t * wcschr(const wchar_t *, wchar_t); |
| int strcmp(const char *, const char *); | int wcscmp(const wchar_t *, const wchar_t *); |
| char * strcpy(char *, const char *); | wchar_t * wcscpy(wchar_t *, const wchar_t *); |
| size_t strlen(const char *); | size_t wcslen(const wchar_t *); |

Из таблицы видно, что имена всех новых функций начинаются с **wcs** — это аббревиатура **wide character set** (набор широких символов). Таким образом, имена Unicode-функций образуются простой заменой префикса **str** соответствующих ANSI-функций на **wcs**.

В качестве примера работы с Unicode-функциями привести следующий код:

```
#include <iostream>
using namespace std;

void main()
{
	// ANSI-кодировка
	char szBuf1[15] = "Hello,";
	strcat(szBuf1, " world!");
	cout << sizeof(szBuf1) << " bytes\n"; // 15 байт

	// UNICODE-кодировка
	wchar_t szBuf2[15] = L"Hello,";
	wcscat(szBuf2, L" world!");
	cout << sizeof(szBuf2) << " bytes\n"; // 30 байт
}
```

При разборе данного примера отметить, что буква **L** перед строковым литералом указывает компилятору, что строка состоит из символов Unicode.

## 1.3. Макросы для работы с Unicode

Обратить внимание слушателей на возможность создания универсального кода, способного задействовать как ANSI-кодировку, так и Unicode-кодировку. Для этого необходимо подключить файл **tchar.h**, в котором имеются макросы, заменяющие явные вызовы str- или wcs-функций. При этом если в программе определена символическая константа **_UNICODE**, то макросы будут ссылаться на wcs-функции, в противном случае макросы будут ссылаться на str-функции.

```
// определение символической константы _UNICODE
#define _UNICODE
#include <iostream>
#include <tchar.h>
```

```
using namespace std;

void main()
{
	_TCHAR szBuf3[15] = _TEXT("Hello,");
	_tcscat(szBuf3, _TEXT(" world!"));
	wcout << szBuf3 << '\n';
	cout << "The size of array: " << sizeof(szBuf3) << " bytes\n"; //30 байт
}
```

```
// отсутствие символической константы _UNICODE
#include <iostream>
#include <tchar.h>
using namespace std;

void main()
{
	_TCHAR szBuf3[15] = _TEXT("Hello");
	_tcscat(szBuf3, _TEXT(" world!"));
	wcout << szBuf3 << '\n';
	cout << "The size of array: " << sizeof(szBuf3) << " bytes\n"; //15 байт
}
```

Детально разбирая вышеприведенные примеры, подчеркнуть, что для объявления символьного массива универсального назначения (ANSI/Unicode) применяется тип данных **_TCHAR**. Если макрос **_UNICODE** определен, **_TCHAR** объявляется так:

```
typedef wchar_t _TCHAR;
```

В ином случае **_TCHAR** объявляется следующим образом:

```
typedef char _TCHAR
```

Макрос **_TEXT** избирательно ставит заглавную букву **L** перед строковым литералом. Если **_UNICODE** определен, **_TEXT** определяется так:

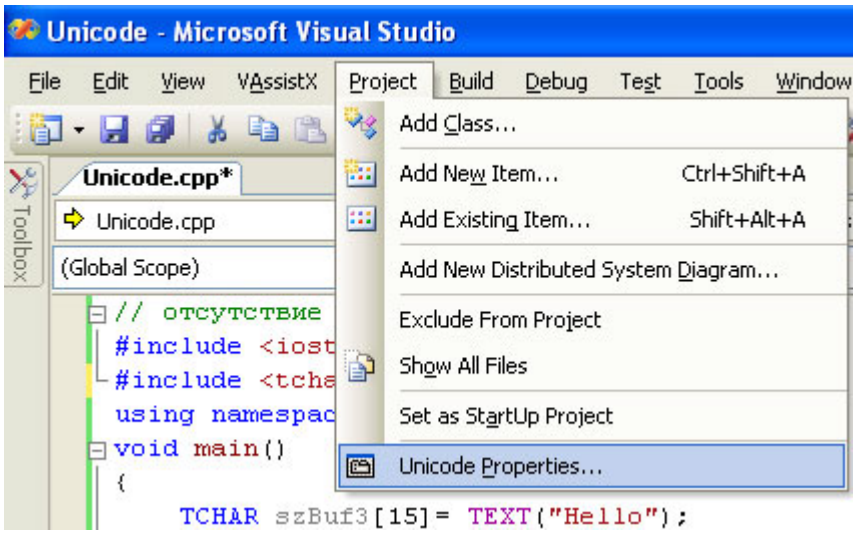
```
#define _TEXT(x)  L##x
```

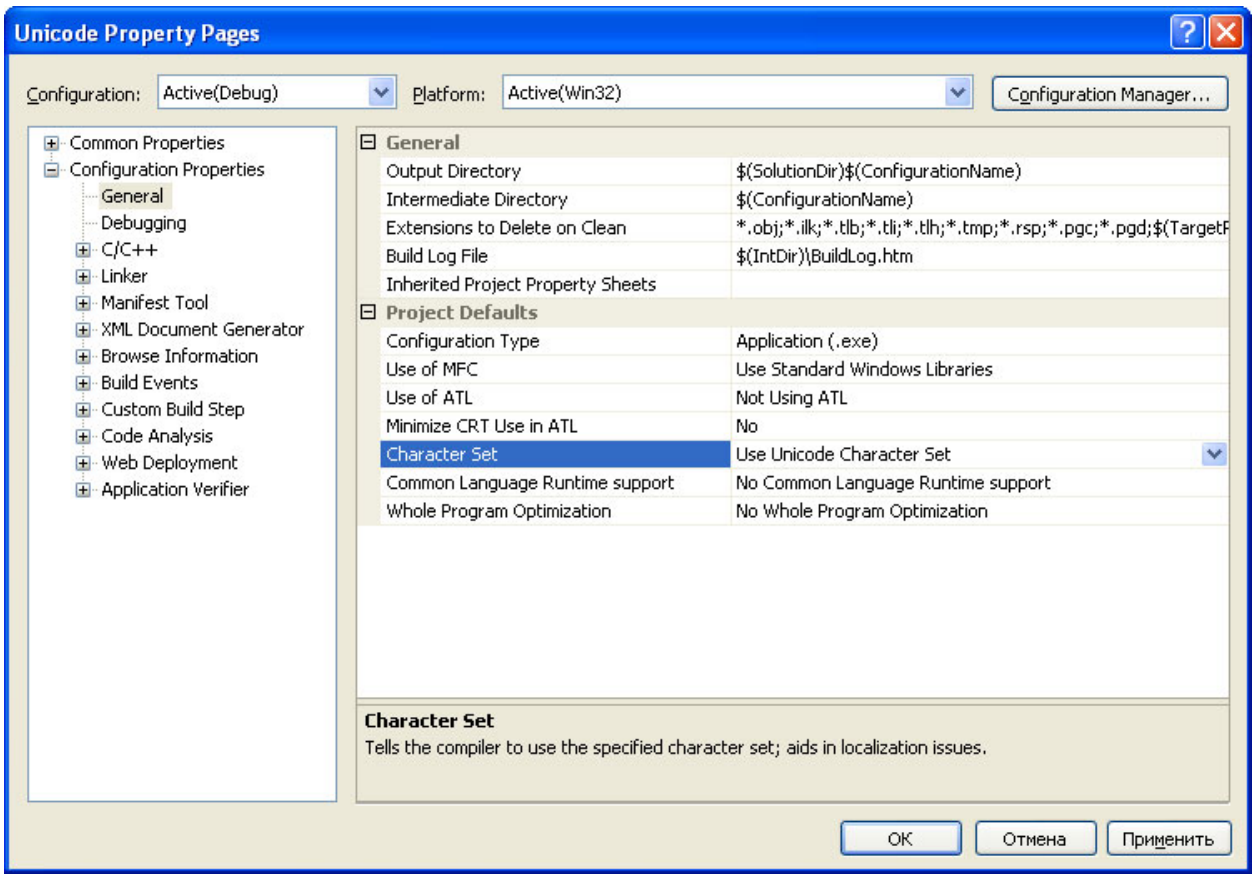
В ином случае **_TEXT** определяется следующим образом:

```
#define _TEXT(x)  x
```

где x – строка символов.

Следует отметить, что существует альтернативный способ задания Unicode-кодировки через свойства проекта.



## 1.4. Строковые функции Windows

В продолжении разговора о Unicode-кодировке ознакомить слушателей со строковыми функциями Windows. Подчеркнуть, что эти функции похожи на строковые функции из библиотеки C/C++, например на **strcpy** и **wcscpy**. Однако функции Windows являются частью операционной системы, и многие её компоненты используют именно их, а не аналоги из библиотеки C/C++. Рекомендуется отдать предпочтение функциям операционной системы. Это немного повысит быстродействие программы.

| Функция | Описание |
|---|---|
| lstrcat | Выполняет конкатенацию строк |
| lstrcmp | Сравнивает две строки с учетом регистра букв |
| lstrcmpi | Сравнивает две строки без учета регистра букв |
| lstrcpy | Копирует строку в другой участок памяти |
| lstrlen | Возвращает длину строки в символах |

Акцентировать внимание слушателей на том, что функции Windows реализованы как макросы, вызывающие либо Unicode-, либо ANSI-версию функции в зависимости от того, определен ли макрос **UNICODE** (аналог **_UNICODE** из библиотеки C/C++) при компиляции исходного модуля. Например, если **UNICODE** не определен, **lstrcat** раскрывается в **lstrcatA**, определен — в **lstrcatW**.

## 1.5. Перекодировка строк из ANSI в Unicode

Для преобразования мультибайтовых символов строки в Unicode-строку используется следующая функция:

```
int MultiByteToWideChar(
  UINT CodePage,          // кодовая страница
  DWORD dwFlags,          // дополнительные настройки, влияющие на
//преобразование букв с диакритическими знаками
```

```
  LPCSTR lpMultiByteStr, // указатель на преобразуемую строку
  int cbMultiByte,       // длина строки в байтах
  LPWSTR lpWideCharStr,  // указатель на буфер, куда запишется Unicode-строка
  int cchWideChar        // размер буфера
);
```

Для демонстрации работы функции привести следующий пример:

```
#include <windows.h>
#include <iostream>
using namespace std;

void main()
{
      char buffer[] =
      "MultiByteToWideChar converts ANSI-string to Unicode-string";
      // определим размер памяти, необходимый для хранения Unicode-строки
      int length = MultiByteToWideChar(CP_ACP /*ANSI code page*/, 0, buffer,
      -1, NULL, 0);
      wchar_t *ptr = new wchar_t[length];
      // конвертируем ANSI-строку в Unicode-строку
      MultiByteToWideChar(CP_ACP, 0, buffer, -1, ptr, length);
      wcout << ptr << endl;
      cout << "Length of Unicode-string: " << wcslen(ptr) << endl;
      cout << "Size of allocated memory: " << _msize(ptr) << endl;
      delete[] ptr;
}
```

При разборе данного примера следует отметить, что первый вызов функции **MultiByteToWideChar** определяет размер памяти, необходимый для хранения Unicode-строки, в то время как второй вызов этой функции выполняет непосредственно перекодировку. Кроме того, следует обратить внимание слушателей на использование объекта **wcout** вместо **cout** для вывода Unicode-строки.

Подчеркнуть, что аналогичные преобразования выполняет функция из C/C++ Run-Time Library:

```
size_t mbstowcs(
   wchar_t * wcstr,//преобразованная Unicode-строка
   const char * mbstr,//исходная ANSI-строка
   size_t count //максимальное число символов исходной строки
);
```

Для демонстрации работы функции привести следующий пример:

```
#include <iostream>
using namespace std;

void main()
{
	char buffer[] = "mbstowcs converts ANSI-string to Unicode-string";
	// определим размер памяти, необходимый для хранения Unicode-строки
	int length = mbstowcs(NULL, buffer, 0);
	wchar_t *ptr = new wchar_t[length];
	// конвертируем ANSI-строку в Unicode-строку
	mbstowcs(ptr, buffer, length);
	wcout << ptr;
	cout << "\nLength of Unicode-string: " << length << endl;
	cout << "Size of allocated memory: " << _msize(ptr) << " bytes" << endl;
	delete[] ptr;
}
```

## 1.6. Перекодировка строк из Unicode в ANSI

Для преобразования Unicode-строк в ANSI используется следующая функция:

```
int WideCharToMultiByte(
  UINT CodePage,          // кодовая страница
  DWORD dwFlags,          // дополнительные настройки, влияющие на
//преобразование букв с диакритическими знаками
  LPCWSTR lpWideCharStr,  // указатель на преобразуемую Unicode-строку
  int cchWideChar,        // количество символов в строке.
  LPSTR lpMultiByteStr,  // указатель на буфер, куда запишется новая строка
  int cbMultiByte,          // размер буфера
  LPCSTR lpDefaultChar,     // функция использует символ по умолчанию, если
//преобразуемый символ не представлен в кодовой странице
  LPBOOL lpUsedDefaultChar  // указатель на флаг, сигнализирующий об успешном
//преобразовании всех символов (в этом случае - FALSE)
);
```

В качестве примера, иллюстрирующего работу функции, привести следующий код:

```
#include <windows.h>
#include <iostream>
using namespace std;

void main()
{
	wchar_t buffer[] =
	L"WideCharToMultiByte converts Unicode-string to ANSI-string";
	// определим размер памяти, необходимый для хранения ANSI-строки
	int length = WideCharToMultiByte(CP_ACP /*ANSI code page*/, 0, buffer,
	-1, NULL, 0, 0, 0);
	char *ptr = new char[length];
	// конвертируем Unicode-строку в ANSI-строку
	WideCharToMultiByte(CP_ACP, 0, buffer, -1, ptr, length, 0, 0);
	cout << ptr << endl;
	cout << "Length of ANSI-string: " << strlen(ptr) << endl;
	cout << "Size of allocated memory: " << _msize(ptr) << endl;
	delete[] ptr;
}
```

Подчеркнуть, что аналогичные преобразования выполняет функция из C/C++ Run-Time Library:

```
size_t wcstombs(
   char * mbstr, //преобразованная ANSI-строка
   const wchar_t * wcstr, //исходная Unicode-строка
   size_t count //максимальное число символов исходной строки
);
```

В качестве примера, иллюстрирующего работу функции, привести следующий код:

```
#include <iostream>
using namespace std;

void main()
{
	wchar_t buffer[] = L"wcstombs converts Unicode-string to ANSI-string";
	// определим размер памяти, необходимый для хранения преобразованной
	// ANSI-строки
	int length = wcstombs(NULL, buffer, 0);
	char *ptr = new char[length + 1];
	// конвертируем Unicode-строку в ANSI-строку
	wcstombs(ptr, buffer, length + 1);
	cout << ptr;
	cout << "\nLength of ANSI-string: " << strlen(ptr) << endl;
	cout << "Size of allocated memory: " << _msize(ptr) << " bytes" << endl;
	delete[] ptr;
}
```

# 2. Программирование Windows-приложений. Введение

Переходя к рассмотрению вопроса программирования Windows-приложений, вкратце ознакомить слушателей с общими принципами функционирования операционных систем семейства Windows. Отметить, что операционная система Windows по сравнению с операционными системами типа MS-DOS обладает серьезными преимуществами и для пользователей, и для программистов. Среди этих преимуществ обычно выделяют:

- графический интерфейс пользователя;
- многозадачность и многопоточность;
- управление памятью;
- независимость от аппаратных средств.

Далее необходимо кратко охарактеризовать каждое из вышеперечисленных преимуществ Windows.

## 2.1. Графический интерфейс пользователя

Графический интерфейс пользователя **GUI (Graphical User Interface)** дает возможность пользователям работать с приложениями максимально удобным способом. Стандартизация графического интерфейса имеет очень большое значение для пользователя, потому что одинаковый интерфейс экономит его время и упрощает изучение новых приложений. С точки зрения программиста, стандартный вид интерфейса обеспечивается использованием подпрограмм, встроенных непосредственно в Windows, что также приводит к существенной экономии времени при написании новых программ.

## 2.2. Многозадачность и многопоточность

Многозадачные операционные системы позволяют пользователю одновременно работать с несколькими приложениями или несколькими копиями одного приложения. Многозадачность осуществляется в Windows при помощи про-

цессов и потоков. Любое приложение Windows после запуска реализуется как процесс. Процесс можно представить как совокупность программного кода и выделенных для его исполнения системных ресурсов. При инициализации процесса система всегда создает первичный (основной) поток, который исполняет код программы, манипулируя данными в адресном пространстве процесса. Из основного потока при необходимости могут быть запущены один или несколько вторичных потоков, которые выполняются одновременно с основным потоком. На самом деле истинный параллелизм возможен только при исполнении программы на многопроцессорной компьютерной системе, когда есть возможность распределить потоки между разными процессорами. В случае обычного однопроцессорного компьютера операционная система выделяет по очереди некоторый квант времени каждому потоку.

## 2.3. Управление памятью

**Память** — это один из важнейших разделяемых ресурсов в операционной системе. Если одновременно запущены несколько приложений, то они должны разделять память, не выходя за пределы выделенного адресного пространства. Так как одни программы запускаются, а другие завершаются, то память фрагментируется. Система должна уметь объединять свободное пространство памяти, перемещая блоки кода и данных. Операционная система Windows обеспечивает достаточно большую гибкость в управлении памятью. Если объем доступной памяти меньше объема исполняемого файла, то система может загружать исполняемый файл по частям, удаляя из памяти отработавшие фрагменты. Если пользователь запустил несколько копий, которые также называют отдельными экземплярами приложения, то система размещает в памяти только одну копию исполняемого кода, которая используется этими экземплярами совместно. Программы, запущенные в Windows, могут использовать также функции из библиотек динамической компоновки — **DLL (dynamic link libraries)**. Windows поддерживает механизм связи программ во время их работы с функциями из DLL. Даже сама операционная система Windows, по существу, является набором ди-

намически подключаемых библиотек. Эти библиотеки содержат набор функций **WinAPI (Application Programming Interface - интерфейс прикладного программирования)**, позволяющих программисту создавать приложения, работающие под управлением Windows.

## 2.4. Независимость от аппаратных средств

Еще одним преимуществом Windows является независимость от используемой платформы. У программ, написанных для Windows, нет прямого доступа к аппаратной части таких устройств отображения информации, как, например, экран и принтер. Вместо этого они вызывают функции графической подсистемы WinAPI, называемой **графическим интерфейсом устройства** (**Graphics Device Interface, GDI**). Функции GDI реализуют основные графические команды при помощи обращения к программным драйверам соответствующих аппаратных устройств. Одна и та же команда (например, LineTo — нарисовать линию) может иметь различную реализацию в разных драйверах. Эта реализация скрыта от программиста, использующего WinAPI, что упрощает разработку приложений. Таким образом, приложения, написанные с использованием WinAPI, будут работать с любым типом дисплея и любым типом принтера, для которых имеется в наличии драйвер Windows. То же самое относится и к устройствам ввода данных — клавиатуре, манипулятору «мышь» и т.д. Такая независимость Windows от аппаратных средств достигается благодаря указанию требований, которым должна удовлетворять аппаратура, в совокупности с **SDK** (**Software Development Kit** — набор разработки программ) и/или **DDK** (**Driver Development Kit** — набор разработки драйверов устройств). Разработчики нового оборудования поставляют его вместе с программными драйверами, которые обязаны удовлетворять этим требованиям.

Прежде чем перейти к рассмотрению архитектуры Windows-приложений, необходимо упомянуть о структуре и способе исполнения консольных приложений, которые слушатели разрабатывали до этого момента. Отметить, что **все консольные приложения имеют архитектуру DOS-программ (Disk Oper-**

**ating System). В таких программах применяется процедурный подход к программированию, при котором программа выполняется последовательно от начала (от первой строки функции main) до конца в предопределенном порядке**. Наиболее часто выполняемые блоки программы выделяются в подпрограммы (функции). В такой архитектуре взаимодействие между системой и программой инициирует сама программа. Например, программа запрашивает разрешение на ввод и вывод данных. Таким образом, консольные приложения, написанные с использованием подобной архитектуры DOS-программ, при необходимости сами обращаются к операционной системе. Операционная система никогда не вызывает прикладную программу.

## 3. Событие. Сообщение. Очередь сообщений

Приступая к рассмотрению архитектуры Windows-приложения, следует отметить, что взаимодействие приложения с внешним миром и операционной системой строится на основе событий и сообщений.

**Событие – это действие, инициированное пользователем, либо операционной системой, либо приложением**. Любому событию соответствует сообщение, которое однозначно идентифицирует произошедшее событие. **Сообщение - это уведомление о том, что произошло некоторое событие.** Событие может быть следствием действий пользователя (например, перемещение курсора, щелчок кнопкой мыши, изменение размеров окна, выбора пункта меню и т.д.). Иногда одно событие влечет за собой еще несколько событий и сообщений. Например, событие создания окна влечет за собой событие перерисовки окна, активизации окна, а также событие создания окна для дочерних окон (кнопок, полей ввода) и так далее.

Сообщение – это уникальная целочисленная константа. Для удобства программирования в программе вместо целочисленных номеров используются макроопределения. Например:

```
#define WM_PAINT 0x000F
```

Весь список мнемонических имен сообщений определен в файле **winuser.h.**

Сообщения от внешних источников, например, от клавиатуры, адресуются в каждый конкретный момент времени только одному из работающих приложений, а именно — активному окну. При этом Windows играет роль диспетчера сообщений. С момента старта операционная система создает в памяти глобальный объект, называемый **системной очередью сообщений**. Все сообщения, генерируемые как аппаратурой, так и приложениями, помещаются в эту очередь. Windows периодически опрашивает эту очередь и, если она не пуста, посылает очередное сообщение нужному адресату (окну).

Сообщения, получаемые приложением, могут поступать асинхронно из разных источников. Например, приложение может работать с системным таймером, посылающим ему сообщения с заданным интервалом, и одновременно оно должно быть готовым в любой момент получить любое сообщение от операционной системы. Чтобы не допустить потери сообщений, Windows одновременно с запуском приложения создает глобальный объект, называемый **очередью сообщений приложения**. Время жизни этого объекта совпадает со временем жизни приложения.

## 4. Окно

Рассмотрение данного вопроса необходимо начать с определения окна. Привести следующее определение окна:

**Окно – это некоторый объект, обладающий набором свойств и занимающий определенную область оперативной памяти.**

Далее следует подчеркнуть, что для любого окна Windows выделяет **дескриптор (handle)**, который уникально идентифицирует окно в пределах системы. Именно при помощи дескриптора Windows определяет какому окну нужно отправить сообщение.

Акцентировать внимание слушателей на том, что у любого окна есть **оконная процедура**, которая получает все сообщения, предназначенные окну. В ней программист может выполнить обработку поступающих сообщений. После

того как сообщение получено, оконная процедура ищет функцию-обработчик данного сообщения, и если он определен в коде программы, то выполняется его тело, иначе вызывается стандартный Windows-обработчик (стандартная функция обработки этого сообщения). После завершения обработки сообщения программа ожидает следующее сообщение. Windows может посылать программе сообщения самых различных типов. Таким образом, программирование под Windows, в сущности, сводится к обработке сообщений.

**Важно отметить, что сообщения поступают в программу неупорядоченным образом, т.е. невозможно предугадать какое сообщение придет в следующий момент времени. Отсюда следует, что нельзя заранее знать какой фрагмент кода оконной процедуры будет выполняться в конкретный момент времени. Это обуславливает нелинейный способ выполнения программы.**

Далее ознакомить слушателей с понятием класса окна. Отметить, что **оконный класс (window class), или класс окна** — это структура, определяющая основные характеристики окна. К ним относятся стиль окна и связанные с окном ресурсы, такие как пиктограмма, курсор, меню и кисть для закрашивания фона. Кроме того, одно из полей структуры содержит адрес оконной процедуры, предназначенной для обработки сообщений, получаемых любым окном данного класса.

Использование класса окна позволяет создавать множество окон на основе одного и того же оконного класса и, следовательно, использовать одну и ту же оконную процедуру. Например, все кнопки в программах Windows созданы на основе оконного класса **BUTTON**. Оконная процедура этого класса, расположенная в динамически подключаемой библиотеке, управляет обработкой сообщений для всех кнопок всех окон. Аналогичные системные классы имеются и для других элементов управления (например, списки и поля редактирования), которые представляют собой частный случай окна. В совокупности эти классы называются предопределенными или стандартными оконными классами.

Windows содержит предопределенный оконный класс также и для диалоговых окон, играющих важную роль в графическом интерфейсе пользователя.

Для главного окна приложения обычно создается собственный класс окна, учитывающий индивидуальные требования к программе.

## 5. Подведение итогов. Домашнее задание

Подвести общие итоги занятия. Еще раз акцентировать внимание слушателей на необходимости использования Unicode-кодировки. Отметить основные преимущества операционной системы Windows. Провести сравнительный анализ исполнения консольного приложения и Windows-приложения. Напомнить слушателям основные термины и понятия, связанные с архитектурой Windows-приложений. Отметить наиболее тонкие моменты изложенной темы.

Ввиду теоретической направленности данного занятия практическая часть домашнего задания отсутствует. Однако необходимо настоятельно рекомендовать слушателям тщательно проработать материал.