**Создание консольных приложений с помощью мастера в Visual C++ 6 - 2**

Помнится, в прошлой статье «Первая программа в Microsoft Visual C++ 6.0» я пообещал рассказать, как создавать консольные приложения в Visual C++ с помощью мастера. Обещания я привык всегда выполнять. Значит, этим сейчас и займёмся. В прошлый раз мы сами писали программный код в текстовом редакторе, среда же нам его компилировала, компоновала и запускала на выполнение. Да, это её работа, но мы её ещё кое-чем загрузим. Я уже говорил, что среда для того и нужна, чтобы выполнять за нас рутинную работу. Ну что, ща продемонстрируем!

Запускаем великий и могучий Visual C++. Жмём FileNew. Далее, поскольку мы будем создавать не просто файл исходного кода в текстовом редакторе, а проект консольного приложения (классно звучит?!), жмём на вкладку Projects, выбираем Win32 Console Application (консольное приложение для Win32). Жмём Ok. Теперь мастер нас спрашивает: What Kind of Console Application do you want to create? (Какое консольное приложение следует создать?) и предлагает вот какие варианты:

An empty project (Пустой проект).

A simple application (Простое приложение)

A “Hello World!” application (Приложение “Hello World!”)

An application that supports MFC (Приложение с поддержкой MFC)

Выбираете необходимое, жмёте Finish, и среда сама за вас кое-что делает. А именно, создаёт заготовку консольного приложения. Вот дальше будете программировать сами, так как заготовки, хотя обычно и являются вполне рабочими приложениями, по сути ничего не делают. Разве что интерфейс, но обработки информации - никакой. Это уже будет наша задача - напрограммировать мозг программы. Заготовка же - это, по сути, её рожа. Да, конечно же, интимные места программы тоже в нашей власти (если потребуется) :) Тем не менее, создание и организация интерфейса может оказаться очень долгим, муторным и неприятным занятием. Действительно, в консольных приложениях это почти не ощутимо (да какой там на фиг интерфейс??!), а вот в случае, скажем, многодокументного приложения создание и организация интерфейса вручную - действительно огромнейший геморрой! Но даже в этом за нас многое может сделать среда, а в том, что надо будет делать нам самим, она тоже в состоянии оказать хорошую помощь и поддержку. Это я просто к тому, что среда халявы нам предоставляет немеряно! Однако, многодокументные приложения - дело весьма непростое, и их мы будем создавать и разбирать значительно позже. Сейчас же консоль, консоль и ещё раз консоль!

Ну что, давайте перепробуем, разберём и изучим все 4 варианта, благо их всего лишь 4! По порядку, начинаем с пустого проекта. Назовём соответственно Listing1, Listing2, Listing3 и Listing4. Хотя можете, конечно, как понравится называть.

**An empty project**

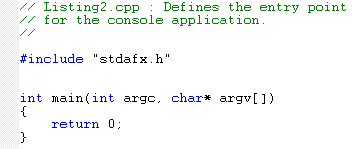
Итак, если мы выбрали An empty project, то что же нам сделает среда? Да по сути ничего! Вот зараза, да! :) Ни тебе программного кода, ни даже файл .cpp не создала - нифига… Ну, на самом деле, кое-что она конечно сделала: создала нам файлы .dsw (workspace - рабочее пространство) и .dsp (project - проект) и некоторые другие, пустую папочку Debug для отладочной версии приложения, либо Release для рабочей версии.

Кстати! Для выбора активной конфигурации проекта (обычно это Debug (отладочная версия) или Release (рабочая версия)) заходите в BuildSet Active Configuration и выбирайте. Кто не в курсе, это определяет режим компиляции проекта.

В самой среде в нашем рабочем пространстве (окно workspaces, вкладка FileView) появились пустые папочки Source Files, Header Files, Resource Files - для файлов исходного кода программы, заголовочных файлов и файлов ресурсов соответственно. Но что нам толку то с того? Мы ведь могём заставить ленивую среду и побольше работы выполнить!

**A simple application**

Выберем A simple application. Теперь уже Visual C++ удосужился-таки сделать нам файл исходного кода (у меня проект назван Listing2, поэтому и файл будет называться Listing2.cpp), причём следующего содержания вышел файл:



Ну что же, это уже на что-то похоже! Давайте разберём.

Сверху он нам даже прокомментировал то, что сделал. Дальше идёт #include “stdafx.h” - это подключается файл stdafx.h . А он в свою очередь относится к созданному всё тем же мастером файлу stdafx.cpp . Файл stdafx.cpp ответственен за создание перекомпилированных заголовков. Что, совсем мозги запудрил? Ничего, стряхивайте пудру с башки, объясню человеческим языком. К нашей программе может подключаться много различных файлов заголовков. Это могут быть как стандартные файлы, которые любезно предоставляет нам, к примеру, Microsoft, или же наши собственные. Так вот, если каждый раз в проекте компилировать все файлы заново - очень долго получается. Но мы ведь не все файлы изменять будем! Те, что не изменяются или изменяются очень редко - компилируются один раз при первой компиляции, а в последующем будут обрабатываться в уже скомпилированном виде. Заново будут компилироваться только часто изменяемые файлы, чтобы внесённые изменения отражались на работе программы. Это значительно сокращает время компиляции проекта. Просекаете логику? То-то! Так вот в файле stdafx.h и будут указываться (подключаться) те файлы, которые изменяются редко (вообще не изменяются) и которые будут использоваться как уже перекомпилированные.

Теперь еще, наверное, возник у вас вопрос: когда в директиве #include использовать кавычки “ ”, а когда угловые скобки < >, когда указывать расширение .h или .hpp, а когда нет? - Для указания заголовочных файлов стандартной библиотеки не требуется расширение. Они распознаются как заголовочные файлы, потому что вместо синтаксиса #include “…” используется #include <…> . И расширение .h в таком случае не указывается. Для каждого заголовочного файла стандартной библиотеки языка С существует соответствующий стандартный заголовочный файл С++. В первом случае это будет прописываться #include <\*.h>, во втором #include <\*> . Как правило, для включения библиотечных файлов пользуйтесь угловыми скобками < >, а для своих собственных - кавычками “ ”. Например:

#include <iostream> // из стандартного каталога включаемых файлов

#include “myheader.h” // из текущего каталога

Наконец, замечу по этому поводу, что пробелы внутри < > или “ “ имеют значение!

#include < iostream > // не найдёт <iostream>

Смотрим дальше, точнее, ниже. Ниже у нас уже знакомая функция main. «Но что это? Чё за фигня у неё в скобочках? Как же нас так обманули? Вроде же функция без параметров была!» - Ничего я никого не обманул, я уже говорил, что в нашем примере в прошлой статье параметров у неё не было только для простоты. Однако в реале там чаще всего указываются аргументы, необходимые для доступа к параметрам командной строки и их использования в программе. Вот это они и есть. Так что я никого не обстебал, наоборот, я предупреждал об этом! И пообещал в этой статье о них подробнее рассказать. Что ж, выполняю.

Умная среда нам по всем правилам и канонам сгенерировала такую сигнатуру для функции main (тем, кто не в курсах: сигнатура на человеческом языке - это имя функции со списком её параметров - приучайтесь к грамотной терминологии, потом окупится!):

int main(int argc, char\* argv[])

Знаете, часто консольные программы организуются так, что после того, как мы осуществим вызов программы из командной строки, программа выдаёт нам приглашение ввести необходимые данные. Это бывает весьма неудобно для пользователя, напрягает… Пользователю то охота побыстрее результат получить! Так вот, если таких данных немного, то было бы неплохо передать их программе ещё при её запуске. Создатели операционок об этом позаботились: большинство операционных систем, те же DOS и UNIX, например, позволяют передать программе при запуске несколько параметров (или один) (или ни одного). Консоль, естественно, тоже такую возможность нам предоставляет. Как я уже говорил в предыдущей статье, эти параметры называются параметрами командной строки и пишутся при вызове программы после имени самой программы через пробел. Например так:

C:>Stebatrix.exe steb 34

Итак, функция main, как правило, имеет два параметра: целый argc и массив строк argv. Хотя они могут называться и иначе, обычно они называются именно так. Между прочим, поскольку argv у нас является массивом строк, он может быть объявлен по-разному, например:

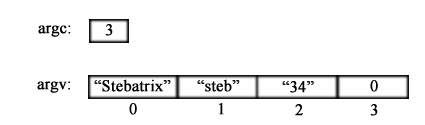
char\* argv[] или char\*\* argv - извращаться над языком можно как угодно (в рамках стандарта языка С++ естественно). Кто не понял, смотрите раздел «Язык программирования С++», постараюсь там об этом (массивах и строках) написать.

Первый параметр argc (argument counter - счётчик аргументов (параметров)) содержит в себе количество параметров командной строки.

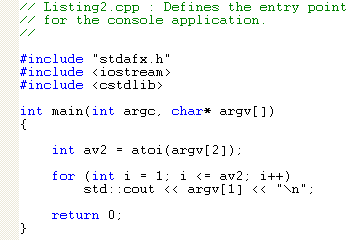
Второй argv (argument vector - вектор (массив) аргументов) содержит в себе массив параметров командной строки. Поскольку параметры являются символьными строками, это будет массив строк, причём смысл здесь такой: argv имеет тип char\* [argc+1] (не забыли про нулевой элемент массива?). Нулевым элементом массива argv будет имя программы. Список аргументов ограничен нулём, то быть argv[argc] == 0. В случае, когда командная строка выглядит так:

C:>Stebatrix.exe steb 34

запустив программу вот, что мы получим:



Теперь про параметры командной строки, думаю, всё понятно. Тем не менее, охота небось программку сделать? Давайте сделаем. Полезно будет. Сейчас и посмотрите, как в простейшем случае можно параметры командной строки использовать. Итак, заготовка исходного файла у нас уже есть (Listing2.cpp), дополните его, чтобы он выглядел так:

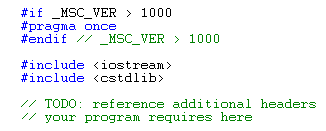


Сразу хочу обратить ваше внимание на то, что подключение iostream и cstdlib можно (пожалуй, уже нужно) объявить не в файле Listing2.cpp, а в stdafx.h. Эти файлы мы изменять не будем, а значит, имеем полное право. Тогда проект компилироваться будет быстрее, хотя в случае нашего проекта-малютки это и не будет заметно. В таком случае уберите эти две строчки

#include <iostream>

#include <cstdlib>

из Listing2.cpp и вставьте их в stdafx.h вот в это место:



Эти две строки должны быть только в одном файле: или Listing2.cpp, или stdafx.h - два раза одно и то же подключать не надо!

Пока не компилируйте и не запускайте. Давайте сейчас разберемся, что же мы напрограммировали. Наша программа будет «продвинутой версией» программы из прошлой статьи, тоже строчку текста выводить будет, только несколько иначе :) При вызове из консоли программа будет принимать два параметра: первый - это строка текста, которую надо будет вывести, а второй - сколько раз надо будет её вывести.

Про iostream вы уже знаете, а вот cstdlib подключается для того, чтобы воспользоваться функцией atoi. Эта функция имеет во какую сигнатуру (см. также файл stdlib.h):

int atoi (const char\* p);

Она преобразует строку p, представляющую собой численное значение (только из цифирей состоящую) в само численное значение. То быть строка ‘12345’ будет преобразована в 12345. Весьма полезная функция. Есть ещё две: atof и atol, преобразующие строчку в double и long соответственно. Если же строчка не представляет собой число (например ‘123g4a’) то будет возвращён 0. Вот как раз с помощью функции atoi мы преобразуем символьную строчку argv[2] в целое av2. Иначе ведь не прокатит! Так вот, преобразуем, а потом циклом for выводим av2 раз строку argv[1]. Кто не знает циклы - во-первых пора бы уже знать, а во вторых - идите в раздел «Язык программирования С++», постараюсь найти время и написать о них. Теперь попробуйте скомпилировать и запустить на выполнение программу. - Что такое? Вроде всё скомпилировалось, скомпоновалось, а когда начала запускаться программа нам вдруг сообщают об ошибке и предлагают отправить отчёт к какой-то там матери!... Неужели Matrix has us?! - Спокуха! Давайте разберёмся. Программа должна принять два параметра, а откуда она их возьмёт, когда мы запускаем её из среды Visual C++, никаких параметров не указывая? Вот и именно, что ниоткуда. Отсюда и лажа. Мы сейчас вот что сделаем: зайдём в консоль и запустим нашу прогу оттуда. Кто не знает, как в консоли работать (в MS-DOS), я здесь объяснять не буду ес-сно, почитайте там Фигурнова какого-нибудь… Итак давайте запустим теперь прогу из консоли, что называется, вручную и с параметрами. Командная строка при этом будет иметь, к примеру, такой вид:

D:Program FilesMicrosoft Visual StudioMyProjectsListing2Release>listing2 We\_have\_Matrix! 3

Только обратите внимание на то, что We\_have\_Matrix! пишем не через пробел, а через символ подчёркивания, т.к. пробелами разделяются параметры и прога так не поймёт (поймёт неправильно). Запускаем и упиваемся тожеством очередной маленькой победы над Матрицей! Но на самом то деле это ещё не победа, боя то никакого почти не было: так, три раза на заборе написали что-то, да и всё. К бою ещё подготавливаться и подготавливаться надо! А посему гружу дальше. Выходит, программа прекрасно работает из консоли. А как же быть с VC++? Да, я мог (и должен был), конечно, предусмотреть в программе возможность получения не только двух параметров командной строки, но и другого их числа. Но не сделал этого намеренно. Чтобы попроще. А если не понарошку, то учитывать такие нюансы необходимо. Ведь и из консоли мы можем программу вызвать, не указав параметры или указав непредвиденное число параметров. Поэтому надо в программе это дело обслужить: просто условным оператором или через исключения организовать выход проги из такого неудобного положения. Но на этом останавливаться не будем: нам ещё 2 варианта заготовок консольных приложений надо рассмотреть и разобрать.

**A “Hello World!” application**

Этот вариант, сразу скажу, практически бесполезен - чисто демонстрационная вещь (показуха одна!). Эта заготовка точно такая же, как и предыдущая (A simple application), с той лишь разницей, что она ещё и выводит с помощью функции printf строчку “Hello World!” на экран. Это различие заключается в одной строчке кода:

printf("Hello World!n");

Мы это с вами уже умеем, только с помощью не printf, а используя cout. Давайте, что ли, про функцию printf маленько и поговорим. Чтобы её использовать, нам надо подключить . Но не беспокойтесь об этом: посмотрите файл stdafx.h - среда уже сделала это за нас! Ну ещё бы монитор тряпочкой протирала и вообще цены бы ей не было! :) Функция printf осталась от языка С. Тем не менее, её очень часто можно встретить. Тем более что, как я уже говорил, хороший код на С будет прекрасно компилироваться и работать в компиляторе для С++. Даже вместе с кодом на С++ (правда в таком случае проблема совместимости всё же время от времени возникает, но она легко решаема). Из этого всего следует, что хотя бы в общих чертах рассмотреть функцию printf нам надо. Сигнатура у неё такая:

int printf(const char\* , …)

Возвращает она нам количество выведенных символов. В случае строки “Hello World!” их будет 13 (не забываем про NULL в конце строки). В качестве параметра она принимает строку в стиле С (массив символов) (или же непосредственно строку в кавычках), причём их может быть несколько, тогда просто через запятую перечислим:

printf("Hello”, “ World!”, “n");

Это также как в cout можете несколько раз использовать << , а в cin >>.

Кстати, кто ещё не в курсе, для ввода в языке С++ используется такая хреновина: std::cin >> и дальше имя переменной, в которую будет записано то, что мы вводим. В С этим занимается функция gets() и др.

Также в printf могут указываться спецификаторы преобразования (это например когда мы значение какой-нибудь переменной хотим вывести), но подробно разговор о функции printf и потоках ввода вывода пойдёт в разделе «Язык программирования С++», там этой теме будет посвящена отдельная статья. Здесь же замечу еще, что использование функции printf нежелательно. Хотя, возможно с ней и проще обращаться, чем с cout и cin, но она имеет большой недостаток: плохой контроль типов данных (при неявных преобразованиях типов, например), что даже у опытных программистов становится иногда причиной досадных ошибок. Поэтому совет: пользуйтесь для ввода/вывода средствами C++!

Переходим к последнему варианту: консольное приложение с поддержкой MFC.

**An application that supports MFC**

Для непосвящённых: MFC - Microsoft Foundation Class (базовые классы Microsoft). И более ничего об MFC писать сейчас не буду, т.к. MFC мы посвятим не просто статью, а целый раздел! Ладно, так и быть - напишу одной фразой: пакет MFC даёт возможность разрабатывать GUI-приложения (GUI - Graphic User Interface - графический пользовательский интерфейс) для Windows на языке С++ с использованием богатого набора библиотечных классов, причём эти библиотеки являются объектно-ориентированными (помимо инкапсуляции поддерживают наследование и полиморфизм). Однако, поскольку MFC в значительной степени ориентирована на работу с окнами, документами, представлениями и т.д., большая её часть остаётся невостребованной консольным приложением. Тем не менее, она всё равно может оказаться очень полезной!

Итак, выбираем теперь четвёртый вариант и смотрим, что нам сотворила среда Visual C++. Да! На этот раз видно, что поработала малышка! Много чего непонятного (и, надеюсь, понятного тоже) написала! Давайте разбираться…

Во-первых, новые файлы появились: Listing4.h, Listing4.rc, Resource.h.

Listing4.h - файл заголовка для нашего исходника. Пока что у нас здесь вот что: первые две строчки и самая последняя - предупреждение повторного подключения, это чтобы не вышло так, что мы бы включили дважды этот файл в какой-нибудь другой (можете почитать в разделе «Язык программирования С++» про стражи включения). Дальше три строчки - проверка версии MFC. Ну и, наконец, подключение файла Resource.h.

Практически каждое приложение, разрабатываемое в Visual C++, содержит один файл ресурсов, одноимённый с самим приложением и имеющий расширение .rc . В нашем случае это Listing4.rc. В нём находятся описания всех ресурсов нашего приложения. У нас здесь будет только таблица строковых ресурсов. В этом и есть высокое предназначение нашей заготовки - она выводит на экран строчку «Hello from MFC!», оформленную как строковый ресурс. Среда ведь на самом деле может сама (с нашей помощью конечно) объявить все необходимые ресурсы, дать им идентификаторы (уникальные имена), связать идентификаторы с конкретными числовыми значениями и т.д. На самом деле это очень удобно, например, для русификации приложения не понадобиться искать в программном коде места где выводятся строчки, а просто в таблице взять и поменять текст с английского на русский, сохранив при этом все идентификаторы и численные значения. В программном коде-то у нас прописаны идентификаторы. Так что умная среда, обеспечив нам такую политику, может избавить нас от значительного геморроя. И это лишь один пример халявы, которую дарит нам Visual C++. При попытке открыть файл Listing4.rc у нас открывается окошко ResourceView. Тут у нас и есть таблица строк. В ней мы можем изменить строку соответствующую данному идентификатору, или создать новый строковый ресурс, поставив ему в соответствие новый идентификатор. Просекаете логику? То-то!

В файле Resource.h просто, используя директиву #define , каждому идентификатору ставится в соответствие численное значение. Здесь же мы можем его поменять или указать его для нового ресурса.

Попробуйте запустить программу, поздоровайтесь с MFC и убедитесь, что приложение работает. Если поздоровались, значит работает. Между тем в нашем рабочем пространстве появилась папочка «External Dependencies» с файлом basetsd.h внутри. В нём, если хотите знать, нам показывают определения базовых типов данных. Это на случай, если нам вздумается (или придётся) использовать 64-битные типы данных.

Между тем и в уже известных нам файлах произошли значительные изменения. Давайте начнем теперь с файла StdAfx.h. В начале всё те же, что и в Listing4.h стражи включения, далее всё та же проверка версии MFC, дальше идёт строчка:

#define VC\_EXTRALEAN // Exclude rarely-used stuff from Windows headers

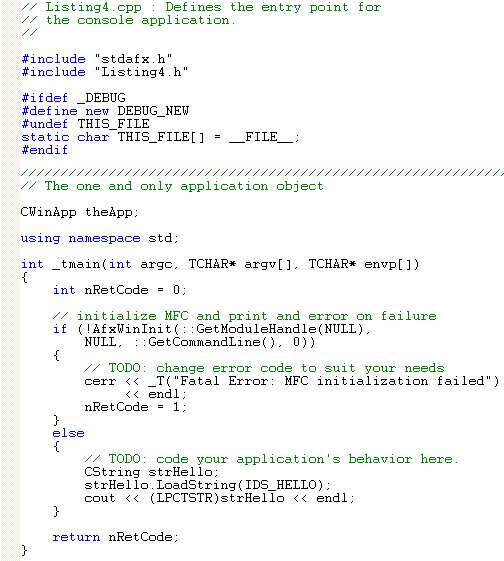
- она для ускорения процесса компиляции (за счёт отказа от некоторых возможностей)

далее идут строчки с подключением основных MFC-шных файлов: afx.h, afxwin.h, afxext.h, afxdtctl.h, afxcmn.h.

Кстати, если будете программировать, используя MFC, очень советую посмотреть эти заголовочные файлы. Если вы знаете язык С++, то должны увидеть там не фигу, а полезную информацию по MFC-классам и функциям и т.д. Бывает ничуть не хуже справочников. А в качестве справочника очень советую MSDN Library.

Ниже расположено место, куда мы подключаем заголовочные файлы, которые редко изменяются и будут использоваться как перекомпилированные (я говорил об этом выше, в начале статьи). Zabot’ливая среда Visual C++ 6.0 уже подключила там <iostream>.

А вот теперь давайте смотреть самое интересное - кончено же Listing4.cpp. Тут у нас вот чего уже готово:



Разбираем по порядку. Как обычно, подключаем stdafx.h, а вместе с ним и Listing4.h . Он подключается здесь, а не в файле stdafx.h, потому что ожидается, что мы этот файл (Listing4.h) будем достаточно часто ковырять. Следующий блок из пяти строчек может оказаться малопонятным. Пожалею я вас и не буду сейчас мозги пудрить, подробно разъясняя эту запись, просто скажу, что она, можно сказать, каноническая, и менять её, как правило, не стоит. Применяется в основном в отладочных целях и во избежание некоторых трудностей с ключевыми словами и именами. Дальше как раз интересное и начинается!

Итак, дальше у нас идёт объявление объекта класса нашего приложения. А именно объявление объекта theApp класса CWinApp. Класс CWinApp является производным от CWinThread. Поэтому класс CWinApp представляет и основной поток выполнения программы, и само приложение. Таким образом, в любом MFC-приложении существует только один объект класса CWinApp. Нас об этом в комментарии уже предупредили.

Строчка

using namespace std;

говорит нам о том, что в данном блоке программы будет использоваться стандартное пространство имён std. Поясняю. В программе может подключаться очень большое число разных файлов заголовков. В каждом файле заголовка используется большое количество различных имен: переменные, функции, классы, структуры и т.д. Разные библиотеки разрабатывают разные люди, и, конечно же, получается так, что в разных библиотеках могут использоваться одинаковые имена. Если мы будем пользоваться сразу несколькими библиотеками, то может сложиться такая ситуация, когда при попытке использовать имя, которое дублируется и в другом подключаемом файле, компоновщик выдаст сообщение об ошибке: «Identifier multiply defined» (Идентификатор определён несколько раз», где идентификатор - это имя какого-либо элемента). Такая ситуация называется конфликтом имён. Для того, чтобы избежать таких проблем и придумали пространства имён. Пространства имён используются для разделения глобального пространства имён, что позволяет пусть и не всегда устранить, но, по крайней мере, уменьшить количество конфликтов имён.

Пространство имен (namespace) по сути представляет собой логическую группу имен, в пределах которой имена не дублируются. Например, пространство имен стандартной библиотеки std. При обращении к идентификатору (имени), входящему в какое-либо пространство имен надо явно указывать это пространство. Поэтому, обращаясь к cout пространства имен std, мы и пишем std::cout. То быть, пишется название пространства имён (std) и через два двоеточия само имя (cout). Вот почему, пользуясь именами из заголовочных файлов стандартной библиотеки (<iostream>, <string>, <list> и др.) мы должны указывать, что всё это относится пространству имён std. Вы можете создавать и свои пространства имён (ну если вам, например, потребуется назвать свою функцию именем cout), но это разговор отдельной статьи.

Запись using namespace std; говорит о том, что в пределах текущей области действия (обычно в пределах фигурных скобок) будет по умолчанию использоваться пространство имён std (не надо будет перед каждым именем писать std:: , если оно принадлежит к этому пространству). Однако этой штукой надо пользоваться очень осторожно - только если вы уверены, что будете использовать только имена указанного пространства, в противном случае могут возникнуть трудности. Если не уверены - лучше не ленитесь и явно указывайте, к какому пространству имен принадлежит данное имя.

Дальше у нас идет функция \_tmain. А почему \_tmain? И почему там тип TCHAR\* , а не char\* ? И что ещё за параметр TCHAR\* envp[] ?

Ну что же, для интересующихся растолкую. Функция называется \_tmain и тип параметров TCHAR\* потому что среда генерирует нам такой код, для того чтобы обеспечить совместимость с различными кодировками текста, разных типов char - одно и многобайтных и т.д. и т.п. Не буду вдаваться во все подробности, да и что нам с того: main или \_tmain, char или TCHAR? Для нас сейчас, в общем-то, смысл не меняется: \_tmain - главная функция в программе, TCHAR\* - строка текста. Обо всех проблемах со стандартами уже поZabot’илась среда. А вот касательно envp ?

Как видите, TCHAR\* envp[] - тоже массив строк. Он необходим для работы с переменными окружения. Кто не в курсе про переменные окружения - читайте что-нибудь про MS-DOS (напр., Фигурнов «IBM PC для пользователя»). Поскольку работа с переменными окружения и процессами в консоли (а оно часто для того и надо) не так проста, и не слишком часто используется, то более подробно в этой статье я разъяснять про них не буду. Хватит с нас пока аргументов командной строки.

Ниже идет объявление переменной nRetCode - она будет возвращаться функцией \_tmain как код ошибки. Напоминаю, если 0 - выполнение программы завершилось успешно, если не 0 - то неуспешно :)

Затем следует собственно инициализация MFC. Обычно (в GUI-приложениях) это происходит так: при инициализации объекта класса CWinApp (или производного от него) функцией WinMain, являющейся частью библиотеки MFC, вызывается функция AfxWinInit и проверяется возвращаемое ею значение. Но, поскольку консольные приложения не используют функцию WinMain, нам приходится вызывать функцию AfxWinInit непосредственно. А она у нас в таком случае просит четыре параметра:

HINSTANCE hInstance

- дескриптор текущего модуля;

HINSTANCE hPrevInstance

- дескриптор предыдущей копии приложения; для Win32-приложений этот параметр всегда NULL;

LPTSTR lpCmdLine

- указатель на командную строку текущего процесса;

int nCmdShow

- определяет, как должно выглядеть основное окно GUI-приложения (поскольку у нас не GUI-приложение, то этот параметр тоже 0);

Что, типы данных странные? Ну да, странные… Но не буду я вам сейчас о них рассказывать - слишком много будет. Пожалею вас - не буду сейчас мозги вам этим пудрить. Всё что на данный момент вам необходимо знать, я пояснил.

Значит так, второй и четвёртый параметр у нас нуль, я уже сказал, а вот откуда мы берём первый и третий. В качестве первого параметра мы используем функцию GetModuleHandle. Она как раз и возвращает нам дескриптор модуля (файла .dll или .exe), имя которого указывается в качестве параметра. Когда этот параметр равен нулю (как у нас), возвращается дескриптор файла, использованного для создания текущего процесса. Что крутовато звучит? Если не понятно, почитайте что-нибудь по архитектуре и основным концепциям Windows… Почитайте про процессы, дескрипторы и т.д. Хотя, быть может, и без того разберётесь? Ладно. В качестве третьего параметра мы используем функцию GetCommandLine, возвращающую указатель на командную строку (формата Unicode) текущего процесса. Кто не в курсе, два двоеточия перед именем функции ставится, так как это - глобальная функция Windows.

Что у нас там дальше идёт? Ах да, после проверки условия, если функция AfxWinInit вернёт не 0 (помните, у нас ведь записано «если не» if (!AfxWinInit … ), то будет передано сообщение об ошибке с помощью стандартного небуферизованного потока диагностики ошибок cerr. У него синтаксис такой же, как и у cout. Кому интересно, макрос \_T используется для решения всё тех же трудностей с кодировками (Unicode, ANSI …) Напоминаю, о потоках читайте в скором времени в разделе «Язык программирования С++». Ну и конечно же, нашему «сторожу» nRetCode присваивается значение 1. В противном случае (если инициализация MFC прошла успешно) будет выполняться код нашей программы. Именно сюда мы и будем писать свой программный код. В случае нашей заготовки это просто вывод на экран строкового ресурса. Объявляем strHello типа CString. Дальше строчкой

strHello.LoadString(IDS\_HELLO);

мы, используя функцию LoadString, которая является членом класса CString, загружаем строковый ресурс Windows, имеющий идентификатор IDS\_HELLO в существующий объект strHello. Затем этот объект выводится на экран с использованием стандартного потока вывода cout. Кому интересно, (LPCTSTR) - опять же для решения трудностей с кодировками.

Ну и, наконец, возвращается наш «сторож» nRetCode.

Вот, собственно, и всё! Ну что, загрузил по полной? Ничего, разбирайтесь, книжки читайте, MSDN читайте, «медитируйте» :)

Хотелось бы ещё сказать, что же всё-таки такое MSDN. MSDN - это здоровенный справочник для программирования в Visual Studio .NET. Там есть всё: и по языку С++, и по программированию в Microsoft Visual C++ и ещё очень много чего полезного. Вся эта прелесть размещается на трёх CD, и при установке на жёсткий диск весит порядка 2Гб. Да, она на английском. Но разобраться можно. Одним словом, рекомендую.

Что же вы теперь знаете и умеете? Вы теперь знаете, как сделать и использовать все 4 типа заготовок консольных приложений, знаете почти во всех деталях, что в этих заготовках и зачем нужно, а значит, можете и сами подобные вещи творить. Более того, разобравшись в этих заготовках, вы теперь знаете, как наилучшим образом на их основе начать создавать своё рабочее приложение. Но само рабочее приложение мы ещё не создавали. Я вас только «натаскал», натренировал маненько для сражений с Матрицей и её отродьями, боя же ещё не было. Но он будет! Впереди нас ждёт первый, пусть и небольшой, но реальный бой с Матрицей. А именно, мы с вами в следующей статье сделаем наше первое реально ПОЛЕЗНОЕ консольное приложение. Сегодня мы с вами подготовились. Битва впереди!

**Список литературы**

Н.Секунов «Самоучитель Visual C++ 6»;

Бьерн Страуструп «Язык программирования С++»;

Джесс Либерти «Освой С++ самостоятельно за 21 день»;

MSDN for Visual Studio .NET;

Содержимое заголовочных файлов (.h) среды Visual C++;

Юджин Олафсен, Кенн Скрайбер, Дэвид Уайт «MFC и Visual C++ 6»;

В.Э. Фигурнов «IBM PC для пользователя»

Zabot 2004