ООП 2021
Лекция 5

Здравый смысл и критическое мышление в программировании.

Введение в программирование оконных (Windows) приложений

oopCpp@yandex.ru

Здравый смысл

Приобретение ЗС – есть ключевая цель всякого образовательного процесса. ЗС включает в себя: приобретение знаний и умение с ними обращаться в любой возникающей жизненной ситуации, а также готовность нести ответственность за свой выбор, совершаемый поступок.

Обретение ЗС осуществляется путем многократных многообразных опытных попыток:

- **решить** поставленную задачу,
- оценить **принципиальную** возможность или невозможность решения,
- **самостоятельно** поставить задачу и определить ее технические условия.

Здравый смысл в программировании

1. Осознание своего действия. Ответ на вопрос: **Зачем**.

Зачем я написал этот класс, **зачем** я вызываю эту функцию здесь и сейчас, **зачем** мне нужно скрыть данные и надо ли это делать вообще.

В ООП это имеет отношение в первую очередь к:

- Нахождению и выбору лучшей (верной, красивой, простой) **абстракции**,
- Осознанию важности **сокрытия** или **разделения** информации,
- Преимущества **полиморфизма** и виртуализации поведения.

2. Понимание к чему приведут сделанные шаги. **Стратегия** последовательных решений.

Выработка такого навыка позволит соотнести написание кода или разработки проекта с предполагаемым результатом.

Это приведет к более **осмысленным** и **ответственным** решениям.

Написание кода и дизайн

1. Код должен нравиться автору!
2. Код должен быть адекватным задаче.
3. Код должен привносить **смысл** в решение.
4. Лучший код – понятный код. (пусть даже за счет повторов и многословия)
5. Стремление к выразительной простоте приводит к лучшему коду.
6. Хорошая читаемость кода – признак его качества.

Технологии программирования или доверие

1. Доверие определяет возможность и скорость научения (доверие книге, коду, учителю)
2. Технологии никогда не способны гарантировать хороший результат, если не опираются на фактор **личного доверия**.
3. Доверие возникшей в сознании идеи, решению, методу должно **перевешивать** технологические требования.

Навык критического мышления и интуиция

1. Критическое отношение к собственному коду позволит искать наилучшее решение. Наилучшее решение – это решение, отвечающее личному эстетическому образцу.
2. Развитие интуитивного взгляда на решение (проект, схема или код) позволит соотнести стратегию (планирование), решение (реализация) и смысл (красоту).

Создание **windows** приложений c++ в visual studio

Виды windows приложений c++ :

1. Приложение Win32 — это классическое приложение Windows на языке C++, которое может использовать встроенные API -Интерфейсы C Windows и (или) API CRT и API стандартной библиотеки, а также сторонние библиотеки.

Приложение Win32, выполняемое в окне, требует, чтобы разработчик работал явно с **сообщениями Windows** внутри **оконной процедуры** Windows. Несмотря на имя, приложение Win32 можно скомпилировать как 32-разрядный (x86) или 64-разрядный (x64) двоичный файл. В интегрированной среде разработки Visual Studio термины x86 и Win32 являются синонимами.

2. Модель COM — это спецификация, которая позволяет программам, написанным на разных языках, взаимодействовать друг с другом. Многие компоненты Windows реализуются как COM-объекты и следуют стандартным правилам COM для создания объектов, обнаружения интерфейса и уничтожения объектов. Использование объектов COM из классических приложений C++ относительно просто, но написание собственного COM-объекта является более сложным. Библиотека активных шаблонов (**ATL**) предоставляет макросы и вспомогательные функции, упрощающие разработку COM .

3. Приложение **MFC** — это классическое приложение Windows, которое использует Microsoft Foundation Classes для создания **пользовательского интерфейса**. Приложение MFC также может использовать компоненты COM, а также API CRT и библиотеки стандартных библиотек. MFC предоставляет **объектно-ориентированную оболочку** с тонким C++ для **циклов оконных сообщений** и API Windows. MFC предоставляет удобные вспомогательные классы для управления окнами, сериализации, обработки текста, печати и современных элементов пользовательского интерфейса, таких как лента. Для эффективной работы с MFC вы должны быть знакомы с Win32.

4. Приложение или компонент C++/CLI использует расширения для синтаксиса C++ (как это разрешено стандартом C++), чтобы обеспечить взаимодействие между **.NET** и машинным кодом C++.

CLI (англ. Common Language Infrastructure) — спецификация общеязыковой инфраструктуры. Наиболее известными реализациями этого стандарта являются Microsoft .NET Framework, Mono, DotGNU Portable.NET. Спецификация CLI определяет, в частности, архитектуру исполнительной системы .NET — **CLR** и сервисы, предоставляемые CLR выполняемым программам, классы, предоставляемые библиотекой **BCL**, синтаксис и мнемонику общего промежуточного языка (CIL).

Base Class Library, или так называемая .NET FCL (англ. Framework Class Library), сокращённо **BCL** — стандартная библиотека классов платформы «.NET Framework». Программы, написанные на любом из языков, поддерживающих платформу .NET, могут пользоваться классами и методами BCL — создавать объекты классов, вызывать их методы, наследовать необходимые классы BCL и т. д.

Не все языки, поддерживающие платформу .NET, предоставляют или обязаны предоставлять одинаково полный доступ ко всем классам и всем возможностям BCL — это зависит от особенностей реализации конкретного компилятора и языка.

В отличие от многих других библиотек классов, например, MFC, ATL/WTL или SmartWin, библиотека BCL не является некоей «надстройкой» над функциями операционной системы или над каким-либо API. Библиотеки BCL является органической частью самой платформы .NET Framework, её «родным» API. Её можно рассматривать как API виртуальной машины .NET.

Приложение C++/CLI может содержать части, работающие в собственном коде, и части, которые выполняются в **.NET Framework** с доступом к библиотеке базовых классов .NET. C++/CLI является предпочтительным вариантом при наличии машинного кода C++, который должен работать с кодом, написанным на C# или Visual Basic. Он предназначен для использования в библиотеках DLL .NET, а не в коде пользовательского интерфейса.

Любое классическое приложение в C++ может использовать среду выполнения C (CRT), классы и функции стандартной библиотеки, COM-объекты и открытые функции Windows.

Классическое приложение Win32

Классическое приложение на C++ — это приложение, которое имеет доступ к полному набору интерфейсов API Windows и запускается в окне или в системной консоли.

Win32 API (далее WinAPI) – это набор функций (API – Application Programming Interface), работающих под управлением ОС Windows. Их объявления содержатся в заголовочном файле windows.h.

С помощью WinAPI можно создавать различные оконные процедуры, диалоговые окна, программы и даже игры. Этот интерфейс лежит в основе Windows Forms, MFC, других библиотек.

```
#include <windows.h> // заголовочный файл, содержащий функции API
 // Основная функция - аналог int main() в консольном приложении:
int WINAPI WinMain(HINSTANCE hInstance, // дескриптор экземпляра приложения
          HINSTANCE hPrevInstance, // в Win32 не используется
          LPSTR lpCmdLine, // нужен для запуска окна в режиме командной строки
               int nCmdShow) // режим отображения окна
{
      // Функция вывода окна с кнопкой "ОК" на экран
      MessageBox(NULL, L"Привет, мир!!!", L"Оконная процедура", MB_OK);
      return NULL; // возвращаем значение функции
}
```

В начале мы подключаем заголовочный файл windows.h. В нём содержатся все необходимые API- функции.

WinMain – название функции. Она имеет четыре параметра. Первый из них – HINSTANCE hInstance. hInstance является дескриптором экземпляра окна (это некий код программы, идентификатор, по которой ОС будет отличать её от прочих программ).

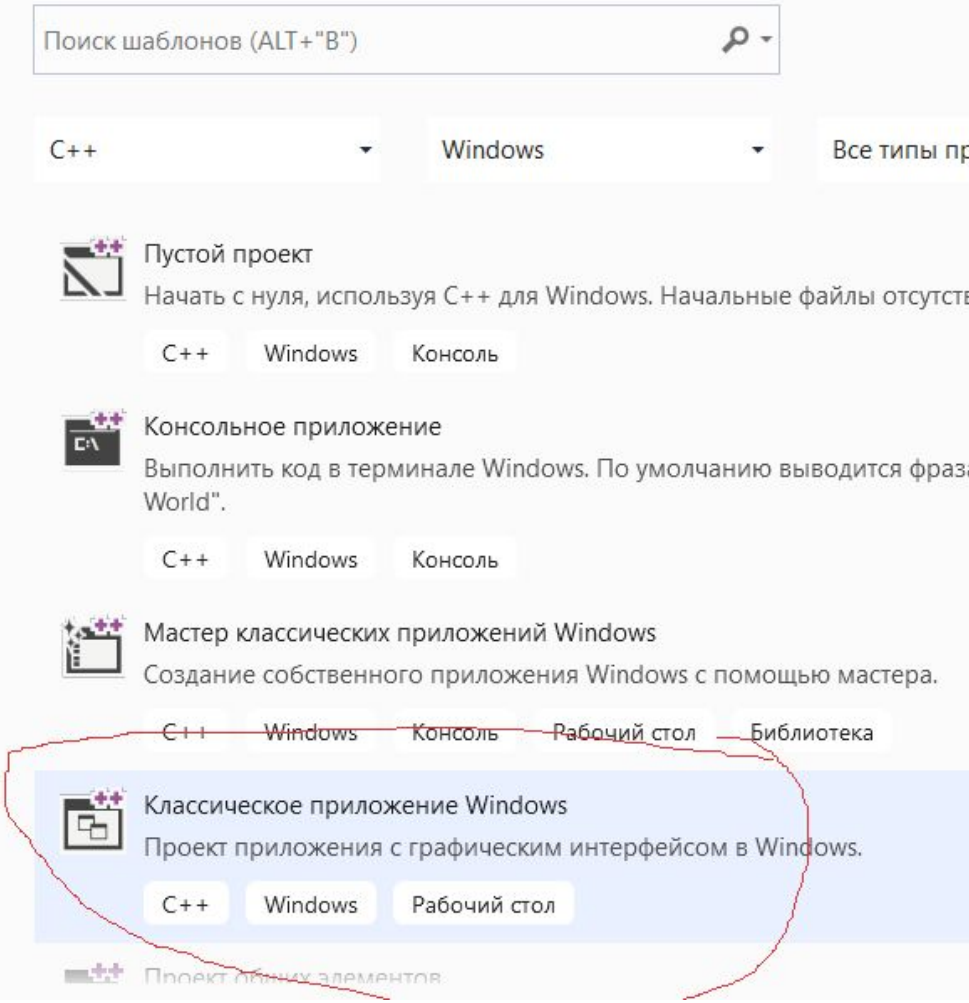
Переменная типа LPSTR с именем lpCmdLine используется в том случае, если мы запускаем окно через командную строку с явным указанием дополнительных параметров.

Параметр nCmdShow - целочисленный, определяет способ показа окна. Нужен для функции ShowWindow. Например, с помощью него мы можем развернуть окно на весь экран, сделать его определённой высоты, прозрачным или поверх остальных.

Создание проекта

Последние шаблоны проектов

- Консольное приложение CLR (.NET Framework) — C++
- Пустой проект CLR (.NET Core) — C++
- Пустой проект CLR (.NET Framework) — C++
- Мастер классических приложений Windows — C++
- Консольное приложение — C++



```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
       // Инициализация глобальных строк
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_WINDOWSPROJECT1, szWindowClass,
     MAX_LOADSTRING);
    MyRegisterClass (hInstance);

    // Выполнить инициализацию приложения:
    if (!InitInstance (hInstance, nCmdShow))     {
        return FALSE;
    }
    HACCEL hAccelTable = LoadAccelerators(hInstance,
     MAKEINTRESOURCE(IDC_WINDOWSPROJECT1));
```

```
MSG msg;
        // Цикл приложения:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}  //  конец wWinMain
```

```
ATOM MyRegisterClass (HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance,
      MAKEINTRESOURCE(IDI_WINDOWSPROJECT1));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_WINDOWSPROJECT1);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance,
      MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}
```

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {
   hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

   HWND hWnd = CreateWindowW (szWindowClass, szTitle,
      WS_OVERLAPPEDWINDOW,
       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow (hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}
```

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT message, WPARAM wParam,
    LPARAM lParam) {
   switch (message)     {
   case WM_COMMAND:         {
         int wmId = LOWORD(wParam);
        switch (wmId)           {
         case IDM_EXIT:
     break;
         default:
             return DefWindowProc(hWnd, message, wParam, lParam);
         }
      }
      break;
   case WM_PAINT:         {
         PAINTSTRUCT ps;
         HDC hdc = BeginPaint(hWnd, &ps); // дальше начинает что-то рисовать
          EndPaint(hWnd, &ps);
      }
      break;
   case WM_DESTROY:
      PostQuitMessage(0);         break;
   default:
      return DefWindowProc(hWnd, message, wParam, lParam);     }
   return 0; }
```

Типы данных в Win32 API

BOOL – этот тип данных аналогичен bool. Он также имеет два значения – 0 или 1. Только при использовании WINAPI принято использовать вместо 0 спецификатор NULL.

BYTE – байт, ну или восьмибитное беззнаковое целое число – unsigned char.

DWORD — 32-битное беззнаковое целое: unsigned long int, UINT.

INT – 32-битное целое – long int.

LONG – 32-битное целое – также long int.

NULL – нулевой указатель: void *NULL=0;

UINT – 32-битное беззнаковое целое - unsigned long int, DWORD.

Дескрипторы — это идентификатор какого-либо объекта. Для разных типов объектов существуют разные дескрипторы. Есть дескрипторы кисти, курсора мыши, шрифта и т.д. С их помощью мы можем при инициализации или в процессе работы приложения поменять какие-нибудь настройки, чего, например, мы не могли сделать в консольном приложении. Используются они в описательных функциях, управляющих типа: CreateProcess(), ShowWindow() и т.д. или как возвращаемое значение некоторых функций :

HANDLE h = GetStdHandle(DWORD x);

В этой получаем дескриптор считывания потоков std_in и std_out.

HANDLE – дескриптор объекта.
HBITMAP – дескриптор растрового изображения.
HBRUSH – дескриптор кисти.
HCURSOR – дескриптор курсора.
HDC – дескриптор контекста устройства.
HFONT – дескриптор шрифта.
HICONS – дескриптор пиктограммы.
HINSTANCE – дескриптор приложения.
HMENU – дескриптор меню.
HPEN – дескриптор пера.
HWND – дескриптор окна.

Строковые типы данных

Есть два вида кодировок символов: ANSI и UNICODE. Однобайтные символы относятся к ANSI, двухбайтные — к кодировке UNICODE.

```
// создаём строку из 10 элементов:
wchar_t str[10]; // UNICODE
```

Если же мы хотим использовать кодировку ANSI, то мы традиционно напишем:

```
char str[10];   // ANSI
```

LPCSTR – указатель на константную строку

LPCTSTR – указатель на константную строку, без UNICODE.

LPCWSTR – указатель на константную UNICODE строку.

LPSTR – указатель на строку, заканчивающуюся нуль- символом.

LPTSTR – указатель на неконстантную строку, без UNICODE.

LPWSTR – указатель на UNICODE строку.

TCHAR – символьный тип — аналог char и wchar_t.

Прочие типы

LRESULT CALLBACK MyWndProc (HWND hWnd, UINT uMSg,
WPARAM wParam, LPARAM lParam);

LPARAM – тип для описания lParam (long parameter).

LRESULT – значение, возвращаемое оконной процедурой имеет тип long.

WPARAM – тип для описания wParam (word parameter). Используются вместе с lParam в некоторых функциях.

Разработка MFC – приложений

MFC – ООП c++ библиотека.

Библиотека MFC содержит большую иерархию классов. В ее вершине находится класс **CObject**, который содержит различные функции, используемые во время выполнения программы и предназначенные, в частности, для предоставления информации о текущем типе во время выполнения, для диагностики, и для сериализации.

Если указатель или ссылка ссылается на объект, производный от класса CObject, то в этом случае предусмотрен механизм определения реального типа объекта с помощью макроса RUNTIME_CLASS().

Каждый класс, производный от CObject, может по запросу проверить свое внутреннее состояние и выдать диагностическую информацию. Это интенсивно используется в MFC при отладке.

Сериализация - это механизм, позволяющий преобразовать текущее состояние объекта в последовательный поток байт, который обычно затем записывается на диск, и восстановить состояние объекта из последовательного потока, обычно при чтении с диска. Это позволяет сохранять текущее состояние приложения на диске, и восстанавливать его при последующем запуске.

Некоторые классы порождаются непосредственно от CObject. Наиболее широко используемыми среди них являются CCmdTarget, CFile, CDC, CGDIObject и CMenu. Класс CCmdTarget предназначен для обработки сообщений.

Класс CFile предназначен для работы с файлами.

Класс CDC обеспечивает поддержку контекстов устройств. В этот класс включены практически все функции графики GDI.

CGDIObject является базовым классом для различных GDI-объектов, таких как перья, кисти, шрифты и другие.

Класс CMenu предназначен для манипуляций с меню.

От класса CCmdTarget порождается очень важный класс CWnd. Он является базовым для создания всех типов окон, включая масштабируемые ("обычные") и диалоговые, а также различные элементы управления. Наиболее широко используемым производным классом является CFrameWnd.

От класса CCmdTarget, через класс CWinThread, порождается CWinApp. Это один из фундаментальных классов, поскольку предназначен для создания самого приложения. В каждой программе имеется один и только один объект этого класса. Как только он будет создан, приложение начнет выполняться.

Функции-члены в MFC

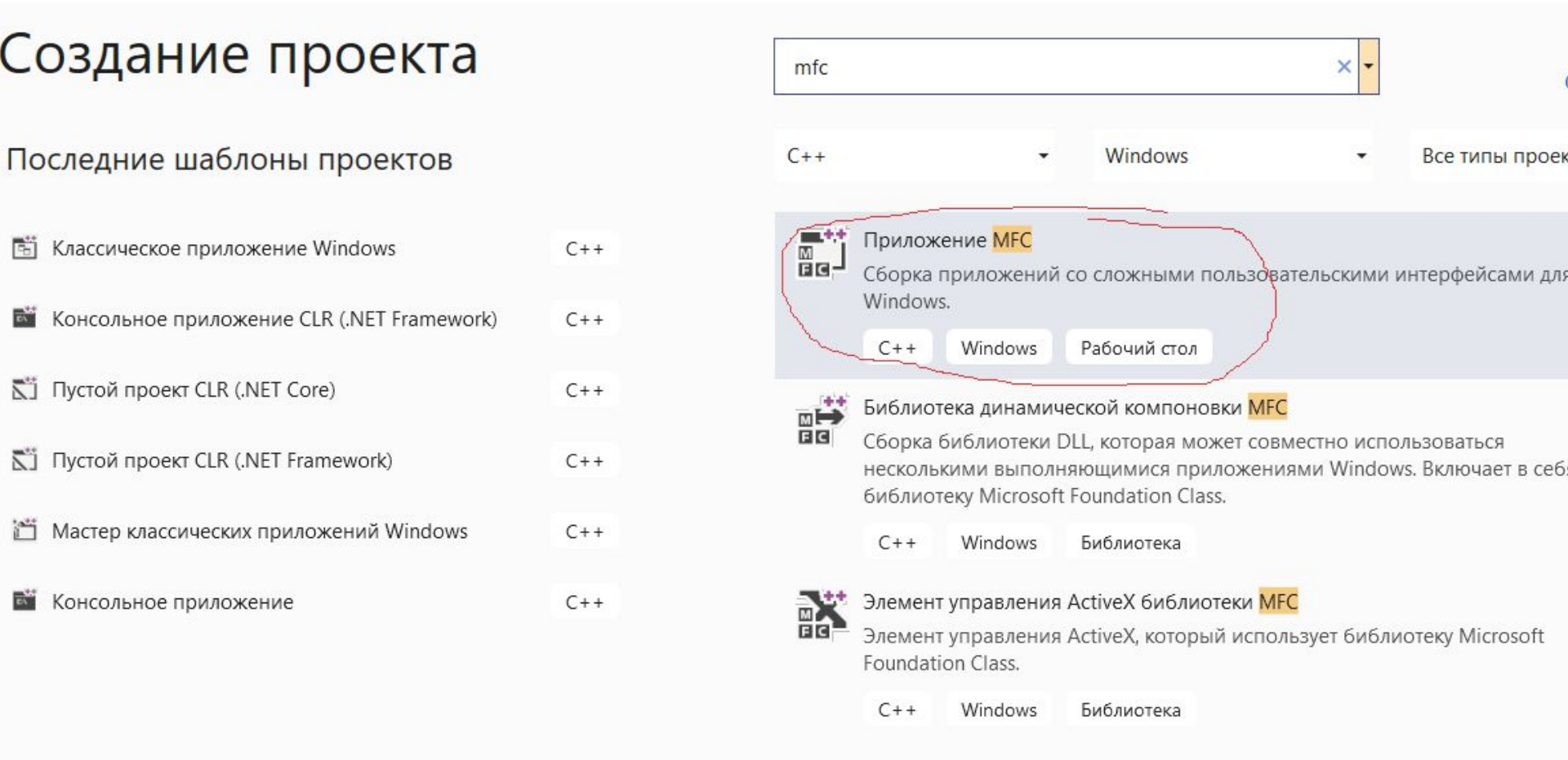
Большинство функций, вызываемых в MFC-программе, являются членами одного из классов, определенных в библиотеке. Большинство функций API доступны через функции-члены MFC. Тем не менее, всегда можно обращаться к функциям API напрямую.

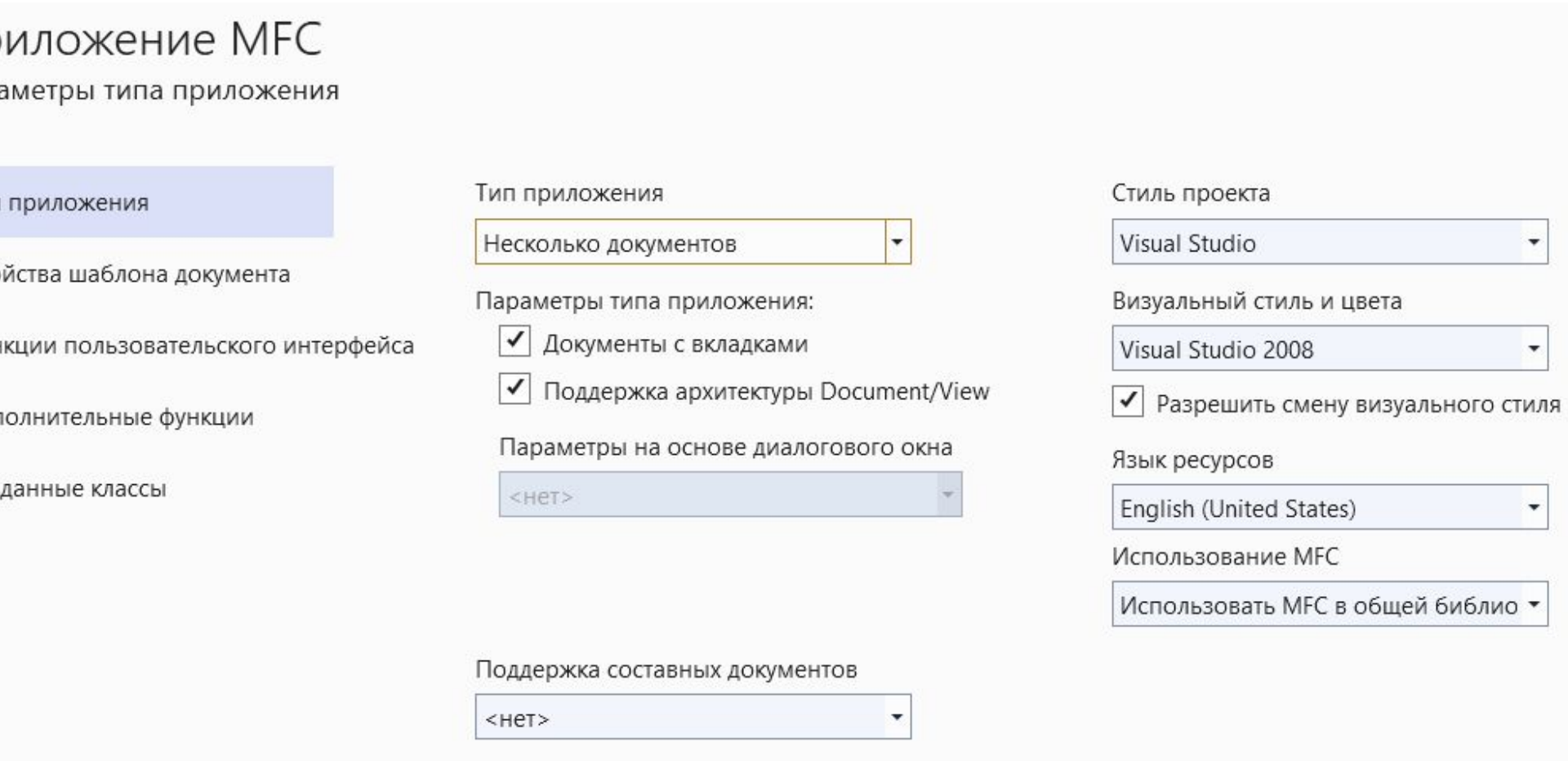
Глобальные функции в MFC

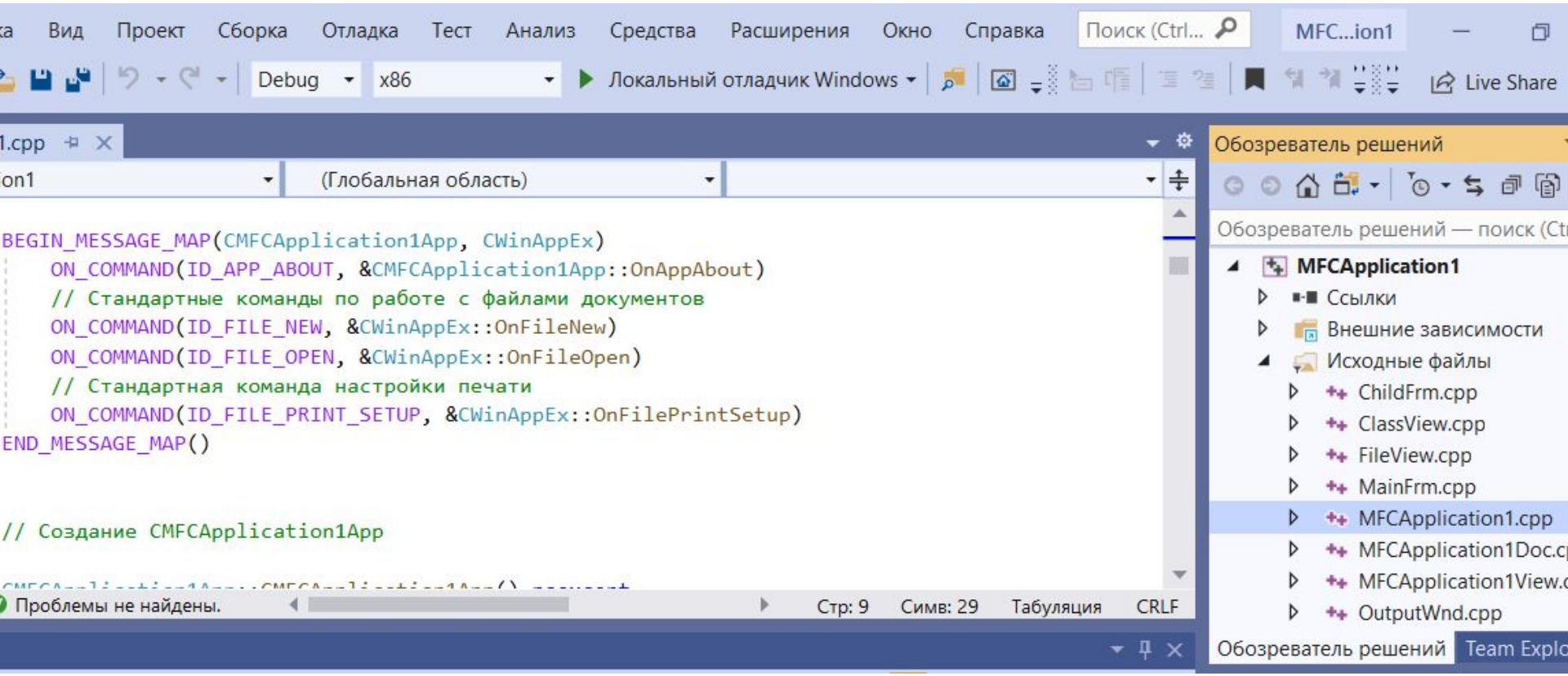
В библиотеке есть ряд глобальных функций. Все они начинаются с префикса Afx. (Когда MFC только разрабатывалась, то проект назывался AFX, Application Framework. После ряда существенных изменений AFX была переработана в MFC, но прежнее название сохранилось во многих идентификаторах библиотеки и в названиях файлов.) Например, очень часто используется функция AfxMessageBox(), отображающая заранее определенное окно сообщения. Но есть и член-функция MessageBox(). Таким образом, часто глобальные функции перекрываются функциями-членами.

Файл AFXWIN.H

Все MFC-программы включают заголовочный файл AFXWIN.H. В нем, а также в различных вспомогательных файлах, содержатся описания классов, структур, переменных и других объектов MFC. Он автоматически подключает большинство заголовочных файлов, относящихся к MFC, в том числе и WINDOWS.H, в котором определены все функции Windows API и другие объекты.


Создание проекта
mfc
Последние шаблоны проектов
C++
Windows
Все типы проек
Классическое приложение Windows
C++
Консольное приложение CLR (.NET Framework)
C++
Пустой проект CLR (.NET Core)
C++
Пустой проект CLR (.NET Framework)
C++
Мастер классических приложений Windows
C++
Консольное приложение
C++
Приложение MFC
Сборка приложений со сложными пользовательскими интерфейсами для
Windows.
C++
Windows
Рабочий стол
Библиотека динамической компоновки MFC
Сборка библиотеки DLL, которая может совместно использоваться
несколькими выполняющимися приложениями Windows. Включает в себ
библиотеку Microsoft Foundation Class.
C++
Windows
Библиотека
Элемент управления ActiveX библиотеки MFC
Элемент управления ActiveX, который использует библиотеку Microsoft
Foundation Class.
C++
Windows
Библиотека


иложение MFC
аметры типа приложения
приложения
йства шаблона документа
кции пользовательского интерфейса
олнительные функции
данные классы
Тип приложения
Несколько документов
Параметры типа приложения:
Документы с вкладками
Поддержка архитектуры Document/View
Параметры на основе диалогового окна
<нет>
Поддержка составных документов
<нет>
Стиль проекта
Visual Studio
Визуальный стиль и цвета
Visual Studio 2008
Разрешить смену визуального стиля
Язык ресурсов
English (United States)
Использование MFC
Использовать MFC в общей библио


Вид
Проект
Сборка
Отладка
Тест
Анализ
Средства
Расширения
Окно
Справка
Поиск (Ctrl...
MFC...ion1
Debug
x86
Локальный отладчик Windows
Live Share
(Глобальная область)
BEGIN_MESSAGE_MAP(CMFCApplication1App, CWinAppEx)
ON_COMMAND(ID_APP_ABOUT, &CMFCApplication1App::OnAppAbout)
// Стандартные команды по работе с файлами документов
ON_COMMAND(ID_FILE_NEW, &CWinAppEx::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, &CWinAppEx::OnFileOpen)
// Стандартная команда настройки печати
ON_COMMAND(ID_FILE_PRINT_SETUP, &CWinAppEx::OnFilePrintSetup)
END_MESSAGE_MAP()
// Создание CMFCApplication1App
Проблемы не найдены.
Стр: 9
Симв: 29
Табуляция
CRLF
Обозреватель решений
Обозреватель решений — поиск (Ct
MFCApplication1
Ссылки
Внешние зависимости
Исходные файлы
ChildFrm.cpp
ClassView.cpp
FileView.cpp
MainFrm.cpp
MFCApplication1.cpp
MFCApplication1Doc.c
MFCApplication1View.
OutputWnd.cpp
Обозреватель решений
Team Explo

```
int CMainFrame::OnCreate (LPCREATESTRUCT lpCreateStruct) {
 // обработка сообщения WM_CREATE при создании главного окна программы
     if (CMDIFrameWndEx::OnCreate(lpCreateStruct) == -1)
     return -1;
 // ….
if (!m_wndMenuBar.Create(this))
{
TRACE0("Не удалось создать строку меню\n");
return -1;      // не удается создать
}

m_wndMenuBar.SetPaneStyle(m_wndMenuBar.GetPaneStyle() |
     CBRS_SIZE_DYNAMIC | CBRS_TOOLTIPS | CBRS_FLYBY);

if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
     CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
     CBRS_SIZE_DYNAMIC) ||
!m_wndToolBar.LoadToolBar(theApp.m_bHiColorIcons ? IDR_MAINFRAME_256 :
     IDR_MAINFRAME))
{
TRACE0("Не удалось создать панель инструментов\n");
return -1;      // не удалось создать
}
```

```
// …
// Разрешить операции с пользовательскими панелями инструментов:
InitUserToolbars(nullptr, uiFirstUserToolBarId, uiLastUserToolBarId);

if (!m_wndStatusBar.Create(this))
{
TRACE0("Не удалось создать строку состояния\n");
return -1;      // не удалось создать
}
m_wndStatusBar.SetIndicators(indicators, sizeof(indicators)/sizeof(UINT));

// TODO: удалите эти пять строк, если панель инструментов и строка меню не
     должны быть закрепляемыми
m_wndMenuBar.EnableDocking(CBRS_ALIGN_ANY);
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockPane(&m_wndMenuBar);
DockPane(&m_wndToolBar);
```

```
// включить режим работы закрепляемых окон стилей Visual Studio 2005
CDockingManager::SetDockingMode(DT_SMART);

// создать закрепляемые окна
if (!CreateDockingWindows())
{
TRACE0("Не удалось создать закрепляемые окна\n");
return -1;
}
 /// ….
 // включить быструю (Alt+перетаскивание) настройку панелей инструментов
CMFCToolBar::EnableQuickCustomization();

if (CMFCToolBar::GetUserImages() == nullptr)
{
// загрузить изображения пользовательских панелей инструментов
if (m_UserImages.Load(_T(".\\UserImages.bmp")))
{
CMFCToolBar::SetUserImages(&m_UserImages);
}
}
```

```
// включить персонализацию меню (последние использованные команды)
// TODO: определите свои основные команды так, чтобы каждое
     раскрывающееся меню содержало по крайней мере одну основную
     команду.

CMFCToolBar::SetBasicCommands(lstBasicCommands);



// повышает удобство использования панели задач, так как на эскизе
     отображается имя документа.
ModifyStyle(0, FWS_PREFIXTITLE);

return 0;
} // конец OnCreate
```

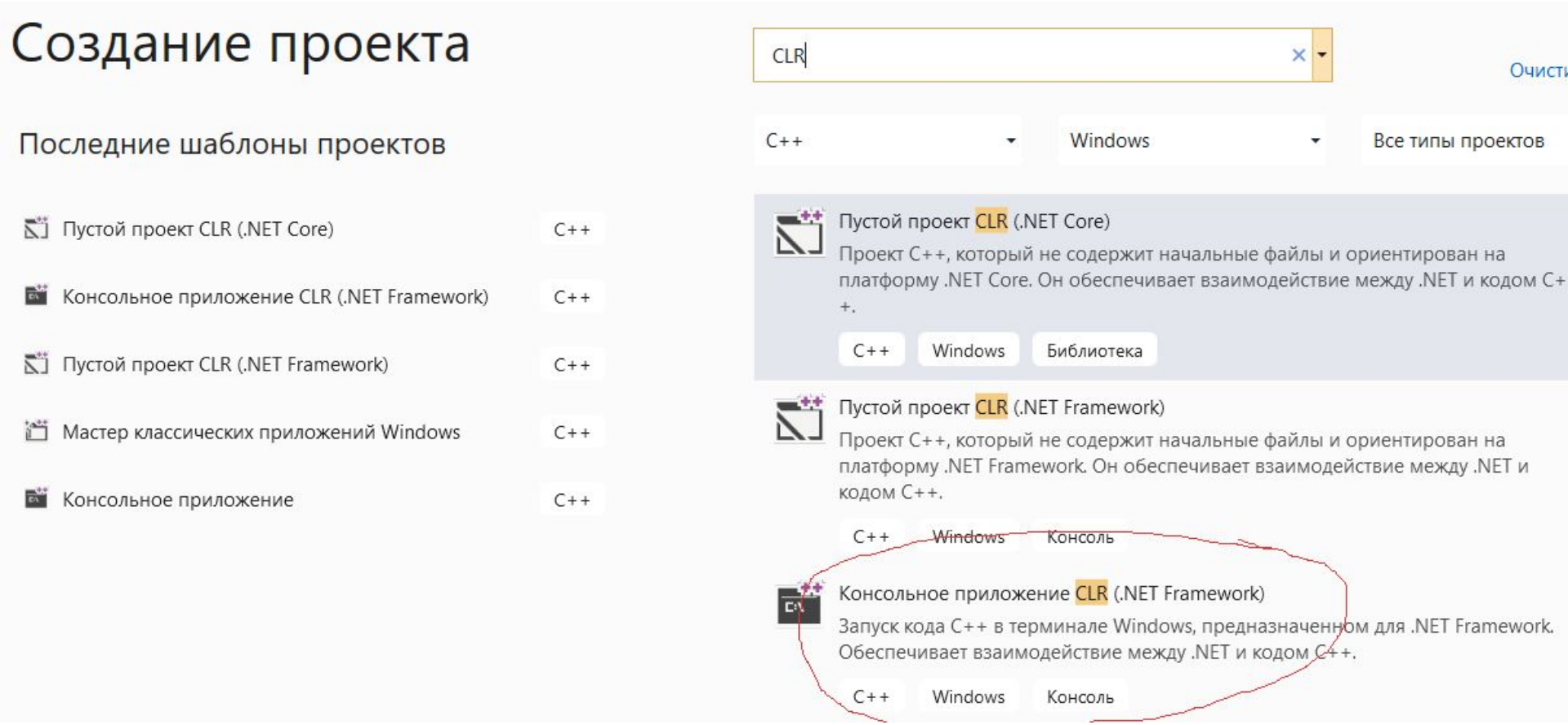
Разработка Windows Form – приложений (CLR)

Шаг-1: Файл - > Новый - > Проект - > Шаблоны - > Visual C++ - > CLR - > CLR Консольное Приложение - > OK.

Шаг-2: Исходные файлы - > Добавить новый элемент - > Visual C++ - > UI -> Windows формы (MyForm.h) - > добавить.

Шаг 3: В дереве слева выберите: Свойства проекта = > Компоновщик = > система. А справа от окна свойств установите "SubSystem" как " Windows (/SUBSYSTEM:WINDOWS)

Шаг 4: Указать точку входа: функцию main.


Создание проекта
CLR
Очисти
Последние шаблоны проектов
C++
Windows
Все типы проектов
Пустой проект CLR (.NET Core) C++
Консольное приложение CLR (.NET Framework) C++
Пустой проект CLR (.NET Framework) C++
Мастер классических приложений Windows C++
Консольное приложение C++
Пустой проект CLR (.NET Core)
Проект C++, который не содержит начальные файлы и ориентирован на платформу .NET Core. Он обеспечивает взаимодействие между .NET и кодом C++.
C++ Windows Библиотека
Пустой проект CLR (.NET Framework)
Проект C++, который не содержит начальные файлы и ориентирован на платформу .NET Framework. Он обеспечивает взаимодействие между .NET и кодом C++.
C++ Windows Консоль
Консольное приложение CLR (.NET Framework)
Запуск кода C++ в терминале Windows, предназначенном для .NET Framework. Обеспечивает взаимодействие между .NET и кодом C++.
C++ Windows Консоль

Настроить новый проект

Консольное приложение CLR (.NET Framework)

C++ Windows Консоль

Имя проекта

WF1test

Расположение

C:\PR1 ...

Имя решения

WF1test

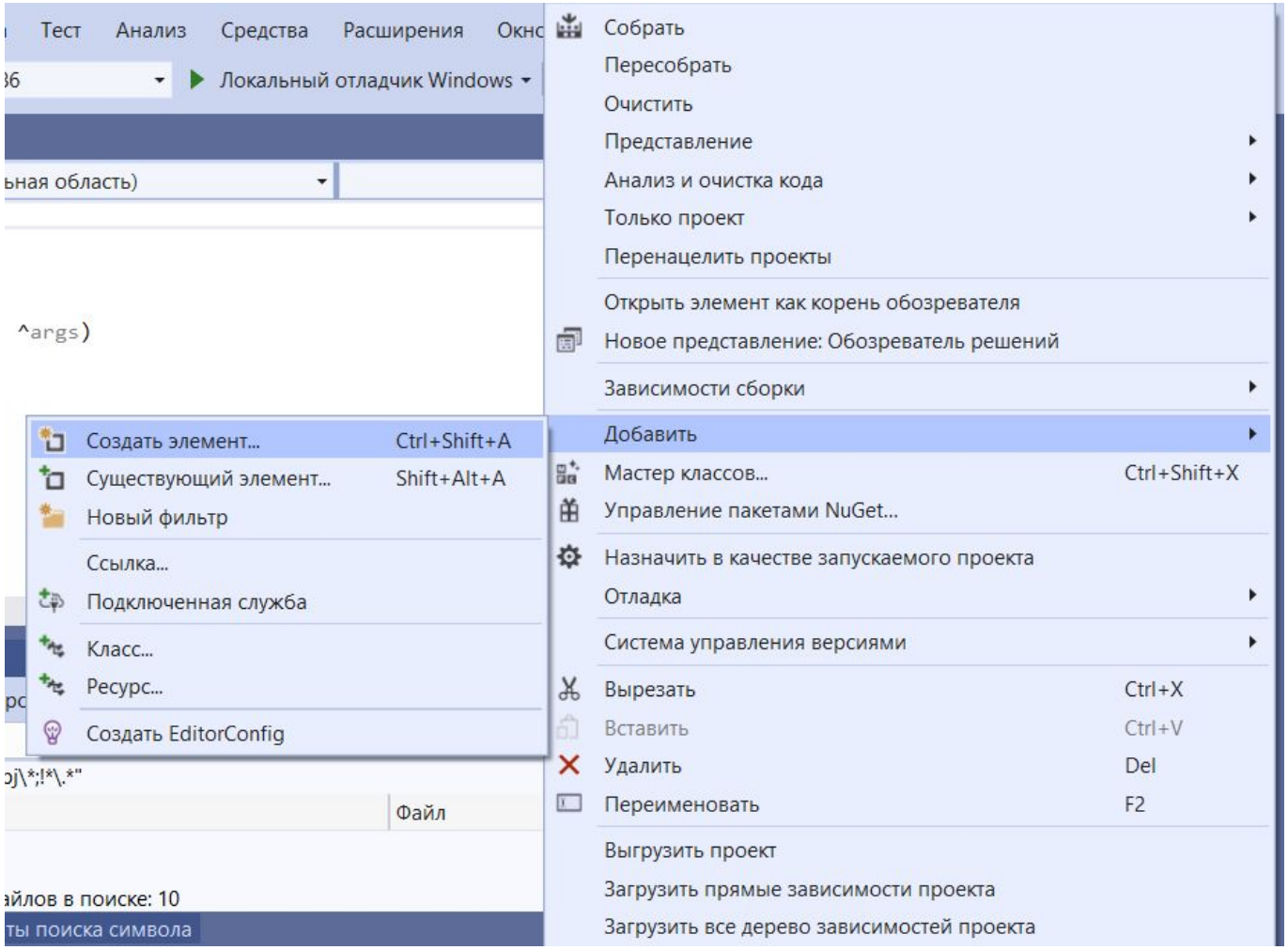
Поместить решение и проект в одном каталоге

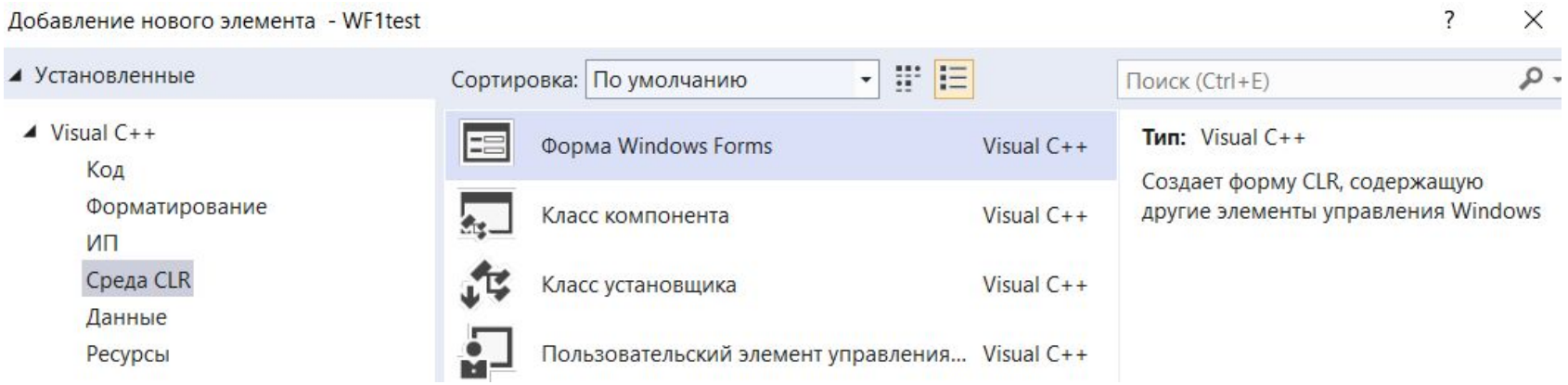
Платформа

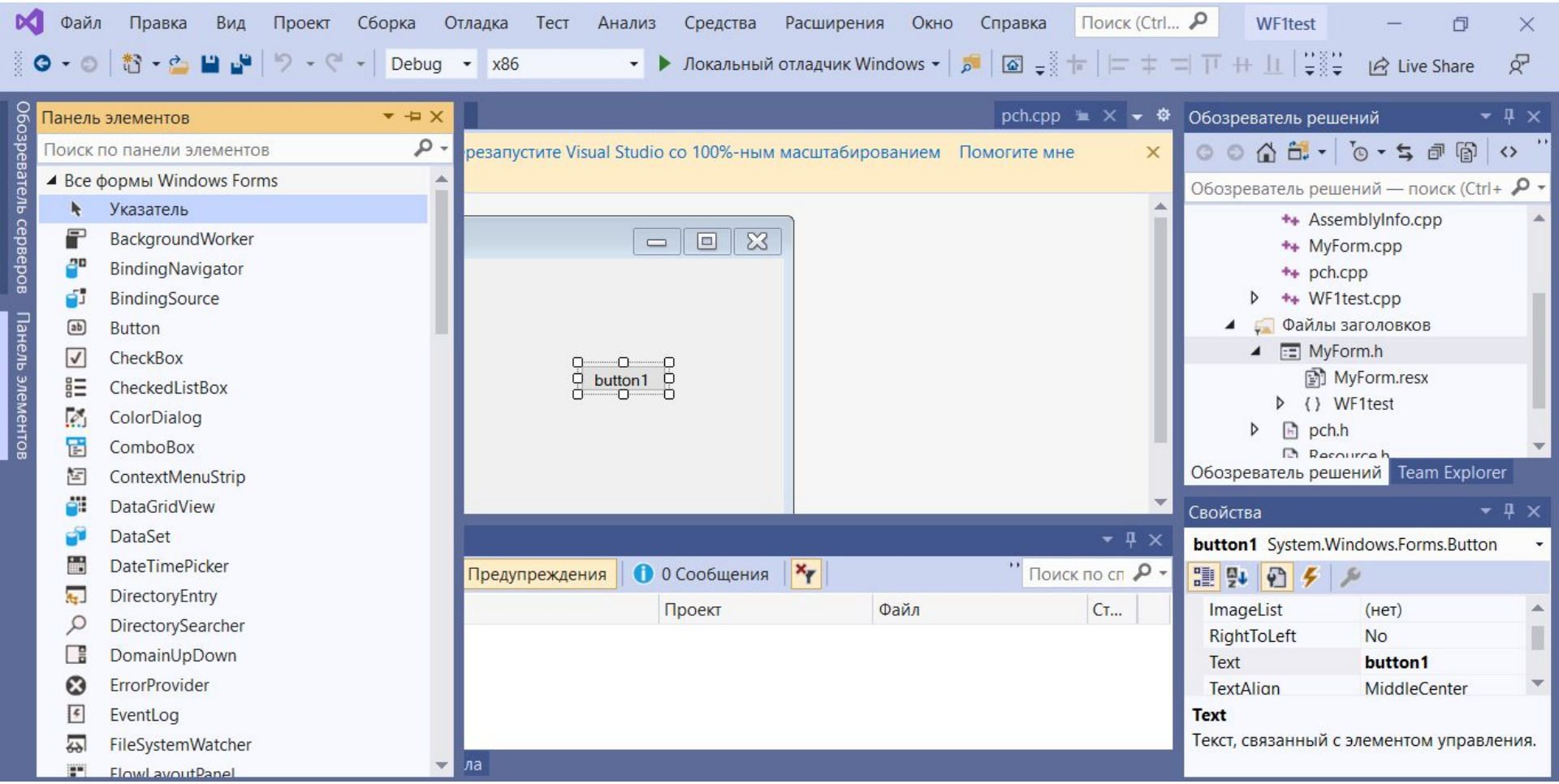
.NET Framework 4.7.2

Назад Созда

ENG 10:5 11.09.


Тест
Анализ
Средства
Расширения
Окно
Локальный отладчик Windows
^args)
Создать элемент... Ctrl+Shift+A
Существующий элемент... Shift+Alt+A
Новый фильтр
Ссылка...
Подключенная служба
Класс...
Ресурс...
Создать EditorConfig
Файл
Собрать
Пересобрать
Очистить
Представление
Анализ и очистка кода
Только проект
Перенацелить проекты
Открыть элемент как корень обозревателя
Новое представление: Обозреватель решений
Зависимости сборки
Добавить
Мастер классов... Ctrl+Shift+X
Управление пакетами NuGet...
Назначить в качестве запускаемого проекта
Отладка
Система управления версиями
Вырезать Ctrl+X
Вставить Ctrl+V
Удалить Del
Переименовать F2
Выгрузить проект
Загрузить прямые зависимости проекта
Загрузить все дерево зависимостей проекта


Добавление нового элемента - WF1test
Установленные
Visual C++
Код
Форматирование
ИП
Среда CLR
Данные
Ресурсы
Сортировка: По умолчанию
Поиск (Ctrl+E)
Форма Windows Forms Visual C++
Класс компонента Visual C++
Класс установщика Visual C++
Пользовательский элемент управления... Visual C++
Тип: Visual C++
Создает форму CLR, содержащую другие элементы управления Windows


Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl...
WF1test
Debug
x86
Локальный отладчик Windows
Live Share
Панель элементов
Поиск по панели элементов
Все формы Windows Forms
Указатель
BackgroundWorker
BindingNavigator
BindingSource
Button
CheckBox
CheckedListBox
ColorDialog
ComboBox
ContextMenuStrip
DataGridView
DataSet
DateTimePicker
DirectoryEntry
DirectorySearcher
DomainUpDown
ErrorProvider
EventLog
FileSystemWatcher
FlowLayoutPanel
Обозреватель серверов
Панель элементов
pch.cpp
резапустите Visual Studio со 100%-ным масштабированием
Помогите мне
button1
Предупреждения
0 Сообщения
Проект
Файл
Ст...
Обозреватель решений
Обозреватель решений — поиск (Ctrl+
AssemblyInfo.cpp
MyForm.cpp
pch.cpp
WF1test.cpp
Файлы заголовков
MyForm.h
MyForm.resx
WF1test
pch.h
Обозреватель решений
Team Explorer
Свойства
button1 System.Windows.Forms.Button
ImageList (нет)
RightToLeft No
Text button1
TextAlign MiddleCenter
Text
Текст, связанный с элементом управления.

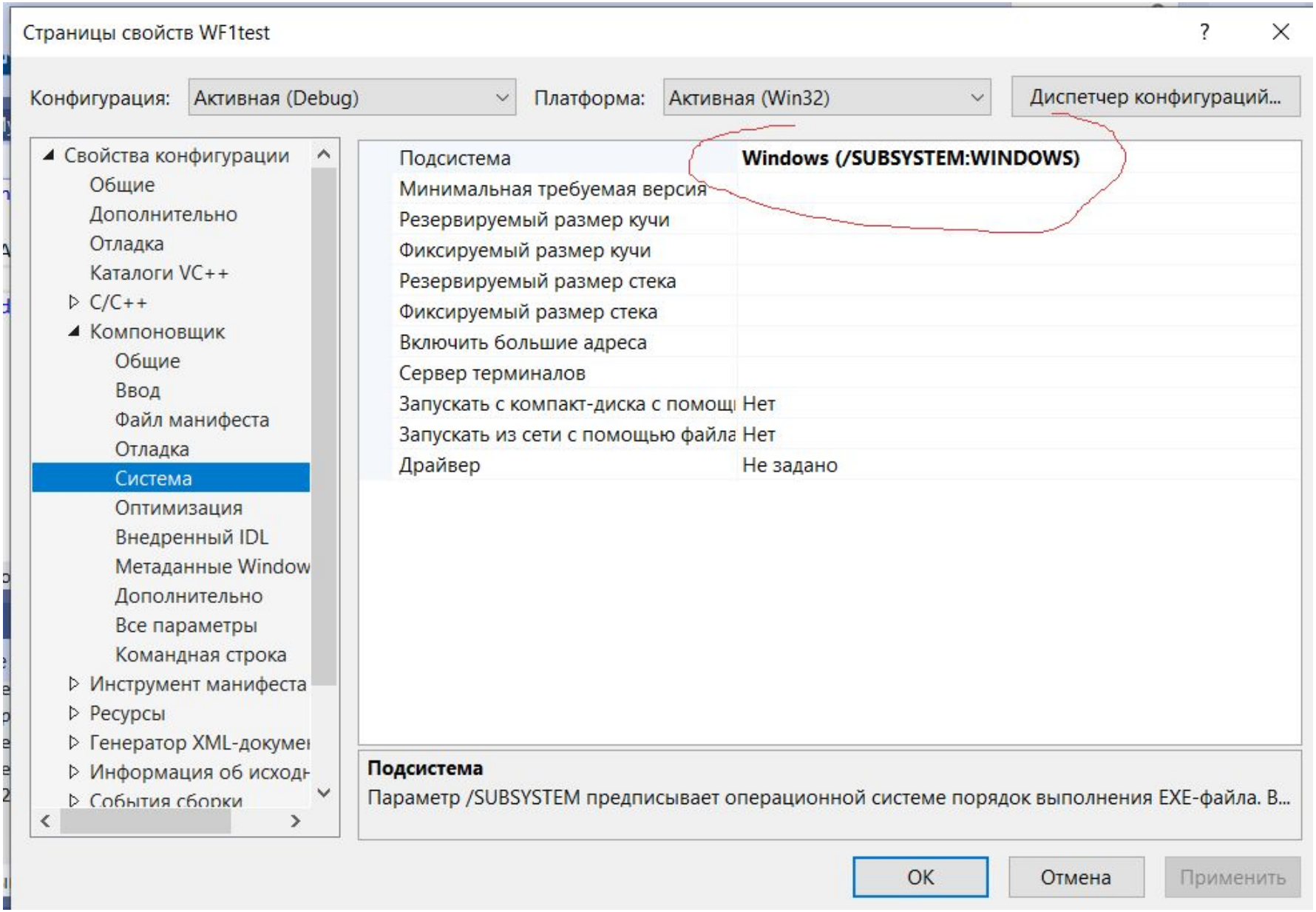
Заменить код в файле, содержащем главную функцию приложения на этот:

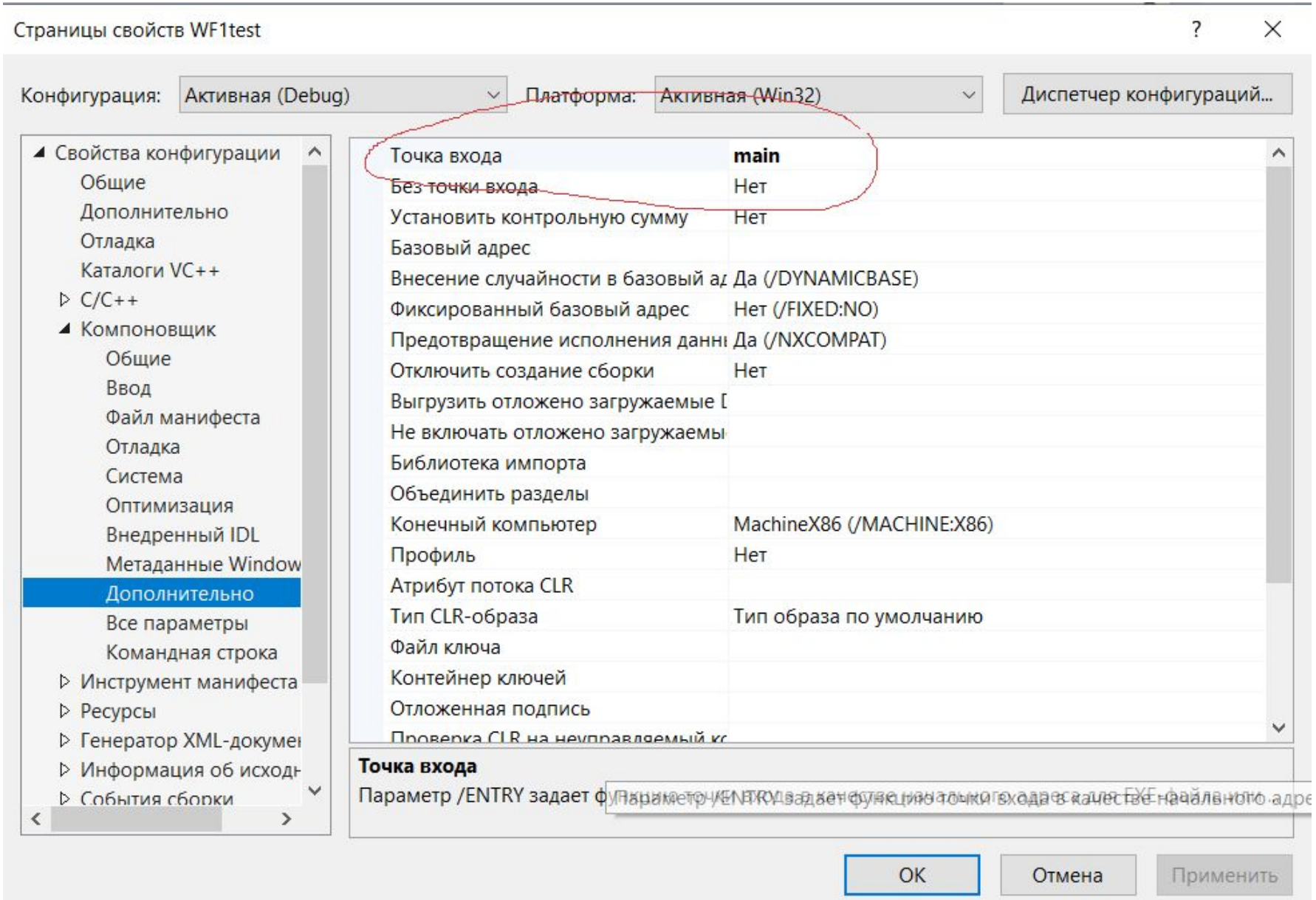
```
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThread]

void main(array<String^>^ args)
{
// обратите внимания на эти крышечки у  array<String^>^ args
// это тип указателей в системе .NET
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    WF1test::MyForm form;
    Application::Run(% form);
}
```


Страницы свойств WF1test
Конфигурация: Активная (Debug)
Платформа: Активная (Win32)
Диспетчер конфигураций...
Свойства конфигурации
Общие
Дополнительно
Отладка
Каталоги VC++
C/C++
Компоновщик
Общие
Ввод
Файл манифеста
Отладка
Система
Оптимизация
Внедренный IDL
Метаданные Window
Дополнительно
Все параметры
Командная строка
Инструмент манифеста
Ресурсы
Генератор XML-докуме
Информация об исход
События сборки
Подсистема
Windows (/SUBSYSTEM:WINDOWS)
Минимальная требуемая версия
Резервируемый размер кучи
Фиксируемый размер кучи
Резервируемый размер стека
Фиксируемый размер стека
Включить большие адреса
Сервер терминалов
Запускать с компакт-диска с помощ
Нет
Запускать из сети с помощью файла
Нет
Драйвер
Не задано
Подсистема
Параметр /SUBSYSTEM предписывает операционной системе порядок выполнения EXE-файла. В...
OK
Отмена
Применить


Страницы свойств WF1test
Конфигурация: Активная (Debug)
Платформа: Активная (Win32)
Диспетчер конфигураций...
Свойства конфигурации
Общие
Дополнительно
Отладка
Каталоги VC++
C/C++
Компоновщик
Общие
Ввод
Файл манифеста
Отладка
Система
Оптимизация
Внедренный IDL
Метаданные Window
Дополнительно
Все параметры
Командная строка
Инструмент манифеста
Ресурсы
Генератор XML-докумен
Информация об исходн
События сборки
Точка входа main
Без точки входа Нет
Установить контрольную сумму Нет
Базовый адрес
Внесение случайности в базовый ад Да (/DYNAMICBASE)
Фиксированный базовый адрес Нет (/FIXED:NO)
Предотвращение исполнения данн Да (/NXCOMPAT)
Отключить создание сборки Нет
Выгрузить отложено загружаемые D
Не включать отложено загружаемы
Библиотека импорта
Объединить разделы
Конечный компьютер MachineX86 (/MACHINE:X86)
Профиль Нет
Атрибут потока CLR
Тип CLR-образа Тип образа по умолчанию
Файл ключа
Контейнер ключей
Отложенная подпись
Проверка CLR на неуправляемый ко
Точка входа
Параметр /ENTRY задает функцию точки входа в качестве начального адре
OK
Отмена
Применить

Еще раз напомню о правиле 3-х

Правило трёх — правило в C++, гласящее, что если класс или структура определяет один из следующих методов, то они должны **явным** образом определить **все три метода**:

Деструктор

Конструктор копирования

Оператор присваивания копированием

И еще раз об операторе присваивания копированием: operator=

```
MyClass& operator= (const MyClass& obj) {

  if ( this == & obj) return *this;
 DeleteData( );
//  что-то копируется из внешнего obj  в  уже  существующий  this->obj
 CopyData (obj);

    return *this;
}
```

Обратите внимание, что в существующем объекте уже может что-то находиться. Это надо все удалить. Также присвоен может быть объект сам себе: надо проверить на самокопирование.

Домашнее задание на неделю

Проект 29. Создать абстрактный базовый класс **именем своей фамилии**, записанной латиницей. Например: **Ivanov**.

Создать 2 производных класса с именами измененного имени базового класса с суффиксами вида «_1» и «_2». Например: Ivanov_2. Первый класс наследуется от базового, а второй – от первого.

В первом классе есть член данных типа int*. Положить туда по умолчанию число 33.

Создать также отдельно от иерархии класс базы данных DB, в котором разместить член-данных типа vector<Ivanov*>.

В функции main создать в динамической памяти по 2 объекта типов созданных классов. Положить их в DB* db1= new DB . Затем, в глобальной функции change изменить значение 33 на 55 в db1 только у объектов типа _1.

Далее реализовать, чтобы правильно работал, следующий в main код:

```
DB db2 (* db1);
delete db1;
DB db3;
db3 = db2;
```

Вывести на консоль данные из db3. Убедиться, что выводит 55, а не 33. Проверить на утечку памяти.

Контрольная работа 5

Создать базовый (Base) и производный класс полиморфной иерархии **именем своей фамилии и имени**.

Создать в базовом классе член-данных типа **int* , выделить для него память и** инициализировать ее произвольным числом в конструкторе по умолчанию.

В производном классе создать член-данных типа float и положить туда число 2.61 в конструкторе по умолчанию.

В функции main создать два хранилища типа vector < Base *>.

В первое хранилище положить (создавая на ходу) по 2 объекта базового и производного типа.

А затем провести глубокое копирование из 1-го вектора во второй и не забыть освободить ресурсы.