Основы визуального программирования

Графический интерфейс

Графический интерфейс пользователя (Graphical User Interface, GUI) еще называют «визуальный интерфейс» или «графическая оконная среда».

GUI делает возможным использование графики на растровом экране. Графика дает лучшее восприятие элементов управления на экране, визуально богатую среду для передачи информации.

В GUI экран становится устройством ввода и показывает различные графические объекты в виде картинок и конструкций для ввода информации, таких как кнопки или полосы прокрутки.

Графические объекты можно перетаскивать, кнопки можно нажимать, полосы прокрутки можно прокручивать. Взаимодействие между пользователем и программой становится более тесным.

Графический интерфейс

Любая программа для Windows имеет **окно** — прямоугольную область на экране, в котором приложение отображает информацию и получает реакцию от пользователя. Окно идентифицируется заголовком.

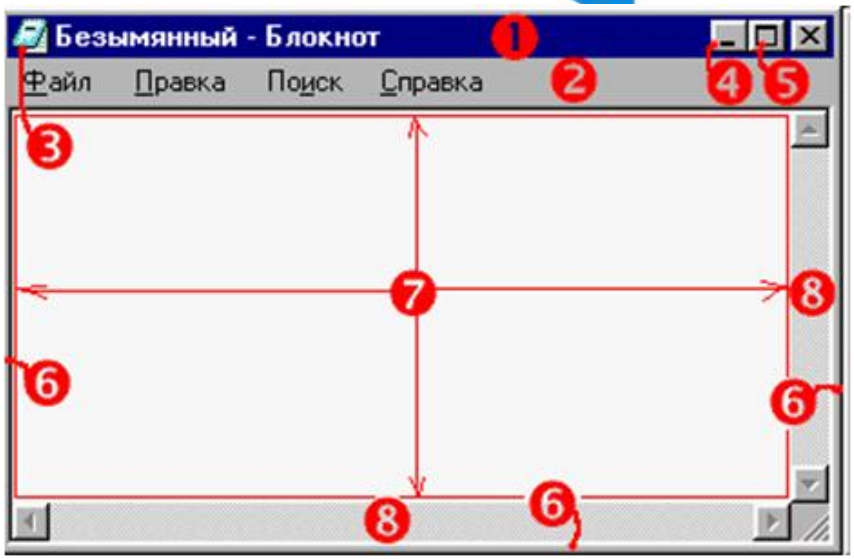
Большинство функций программы запускается посредством **меню**. Слишком большой для экрана объем информации может быть просмотрен с помощью **полос прокрутки**. Некоторые пункты меню вызывают появление окон диалога, в которые пользователь вводит дополнительную информацию.

Программирование Windows-приложений тесно связано с понятиями объектно-ориентированного программирования.

Активное окно – окно, получающее реакцию от пользователя в данный момент.

Графический интерфейс

Основными элементами окна являются



1 — строка заголовка title bar

2 — строка меню menu bar

3 — системное меню system menu

4 — кнопка сворачивания окна minimize box

5 — кнопка разворачивания окна maximize box

6 — рамка изменения размеров sizing border

7 — клиентская область client area

8 — горизонтальная и вертикальная полосы прокрутки scroll bars

Графический интерфейс

Многозадачность (multitasking) — свойство операционной системы обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких процессов.

Программа пассивна, после запуска она ждет, когда ей уделит внимание операционная система. Операционная система делает это посылкой специально оформленных групп данных, называемых **сообщениями**. Сообщения могут быть разного типа, они функционируют в системе достаточно хаотично, и приложение не знает, какого типа сообщение придет следующим.

Логика построения Windows-приложения должна обеспечивать корректную и предсказуемую работу при поступлении сообщений любого типа.

Операционная система не сможет реализовать многозадачность без управления памятью. Так как одни программы запускаются, а другие завершаются, память фрагментируется. Операционная система Windows имеет средства управления фрагментами памяти.

Процессы и потоки

Процессом (process) называется экземпляр программы, загруженной в память. Экземпляр программы может создавать **потоки** (thread), которые представляют собой последовательность инструкций на выполнение.

Выполняются не процессы, а именно потоки. Любой процесс имеет хотя бы один поток. Этот поток называется **главным** (основным) потоком приложения.

Потоки на самом деле выполняются не одновременно, а по очереди. Распределение процессорного времени происходит между потоками, но переключение между ними происходит так часто, что кажется будто они выполняются параллельно.

Все потоки ранжируются по приоритетам. Приоритет потока обозначается числом от 0 до 31, и определяется исходя из приоритета процесса, породившего поток, и относительного приоритета самого потока. Таким образом, достигается наибольшая гибкость, и каждый поток в идеале получает столько времени, сколько ему необходимо.

Дескрипторы

Дескриптор (описатель) объекта — служебная структура данных, представляющая собой беззнаковое целое число и служащая для идентификации различных объектов. **Дескриптор представляет собой указатель на некоторую системную структуру или индекс в некоторой системной таблице.**

Примеры дескрипторов, описанных в заголовочном файле windows.h

```
typedef void *HANDLE;  // абстрактный дескриптор (например, файла)
typedef void *HMODULE;  // дескриптор модуля
typedef void *HINSTANCE; // дескриптор экземпляра программы
typedef void *HKEY;  // дескриптор ключа в реестре
typedef void *HGDIOBJ;  // дескриптор граф. примитива (перо, кисть)
typedef void *HWND;  // дескриптор окна
typedef void *HMENU;  // дескриптор меню
typedef void *HICON;  // дескриптор иконки
typedef void *HBITMAP;  // дескриптор картинки
typedef void *HFONT;  // дескриптор шрифта
```

Дескрипторы

Контекст устройства

GDI – графический интерфейс устройства. Функции системной библиотеки GDI32.dll используются для вывода графики на экран.

Дескриптор контекста устройства — это паспорт конкретного окна для функций GDI. Контекст устройства фактически является структурой данных, которая внутренне поддерживается GDI. Он связан с конкретным устройством вывода информации (принтер, дисплей). Что касается дисплея, то в данном случае контекст устройства обычно связан с конкретным окном на экране.

Структура оконного приложения

Оконные приложения строятся по принципам **событийно-управляемого программирования (event-driven programming)** — стиля программирования, при котором поведение компонента системы определяется набором возможных внешних событий и ответных реакций компонента на них. Такими компонентами в Windows являются окна.

С каждым окном в Windows связана определенная функция обработки событий – **оконная функция**.

События для окон называются **сообщениями**. Сообщение относится к тому или иному типу, идентифицируемому определенным кодом (32-битным целым числом), и сопровождается парой 32-битных параметров (**WPARAM и LPARAM**), интерпретация которых зависит от типа сообщения.

Задача любого оконного приложения — создать главное окно и сообщить Windows функцию обработки событий для этого окна. Все самое интересное для приложения будет происходить именно в функции обработки событий главного окна.

Функции оконного приложения

Классическое оконное приложение, как правило, состоит по крайней мере из двух функций:

- ✓ стартовая функция, создающая главное окно WinMain();
- ✓ функция обработки сообщений окна (оконная функция).

```
int WINAPI WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    PSTR szCmdLine,
    int iCmdShow) {...}
```

Эта функция использует последовательность вызовов API и при завершении возвращает операционной системе целое число.

Аргументы функции

hInstance – дескриптор процесса (instance handle) – число, идентифицирующее программу, когда она работает под Windows. Если одновременно работают несколько копий одной программы, каждая копия имеет свое значение hInstance.

hPrevInstance — предыдущий дескриптор процесса (previous instance) — в настоящее время устарел, всегда равен NULL.

szCmdLine — указатель на оканчивающуюся нулем строку, в которой содержатся параметры, переданные в программу из командной строки. Можно запустить программу с параметром командной строки, вставив этот параметр после имени программы в командной строке.

iCmdShow — целое константное значение, показывающее, каким должно быть выведено на экран окно в начальный момент. Задается при запуске программы другой программой. В большинстве случаев число равно 1 (SW_SHOWNRMAL).

Аргументы функции

ИМЯ	Значение	Описание
SW_HIDE	0	Скрывает окно и делает активным другое окно
SW_SHOWNORMAL	1	Отображает и делает активным окно в его первоначальном размере и положении.
SW_SHOWMINIMIZED	2	Активизирует окно и отображает его в свернутом виде
SW_SHOWMAXIMIZED	3	Активизирует окно и отображает его в полноэкранном виде
SW_SHOWNOACTIVATE	4	Отображает окно аналогично **SW_SHOWNORMAL**, но не активизирует его
SW_SHOW	5	Отображает и делает активным окно с текущим размером и положением.
SW_MINIMIZE	6	Сворачивает текущее окно и делает активным следующее окно в порядке очереди.

Аргументы функции

ИМЯ	Значение	Описание
SW_SHOWMINNOACTIVE	7	Сворачивает окно аналогично **SW_SHOWMINIMIZED**, но не активизирует его.
SW_SHOWNA	8	Отображает окно в текущей позиции аналогично **SW_SHOW**, но не активизирует его.
SW_RESTORE	9	Отображает и активизирует окно. Если окно было свернуто или развернуто во весь экран, оно отображается в своем первоначальном положении и размере.
SW_SHOWDEFAULT	10	Отображает окно способом, заданным по умолчанию.
SW_FORCEMINIMIZE	11	Применяется для минимизации окон, связанных с различными потоками.

Операции стартовой функции

В структуре стартовой функции Windows можно выделить следующие операции, образующие «скелет» программы:

- регистрация класса окна;
- создание главного окна;
- отображение и перерисовка главного окна;
- цикл обработки очереди сообщений.

Регистрация класса окна

Регистрация класса окна осуществляется функцией

ATOM WINAPI RegisterClass(_In_ const WNDCLASS *lpWndClass);

Прототип функции находится в **файле библиотеки user32.dll.**
Единственным аргументом функции является указатель на структуру

```
typedef struct _WNDCLASS {
  UINT      style;
  WNDPROC   lpfnWndProc;
  int       cbClsExtra;
  int       cbWndExtra;
  HINSTANCE hInstance;
  HICON     hIcon;
  HCURSOR   hCursor;
  HBRUSH    hbrBackground;
  LPCTSTR   lpszMenuName;
  LPCTSTR   lpszClassName; } WNDCLASS;
```

Члены структуры функции регистрации класса окна

style — устанавливает стиль(и) класса. Этот член структуры может быть любой комбинацией стилей класса.

Имя	Значение	Описание
CS_VREDRAW	0x01	Вертикальная перерисовка: осуществлять перерисовку окна при перемещении или изменении высоты окна.
CS_HREDRAW	0x02	Горизонтальная перерисовка: осуществлять перерисовку окна при перемещении или изменении ширины окна.
CS_KEYCVTWINDOW	0x04	В окне будет выполняться преобразование виртуальных клавиш.
CS_DBLCLKS	0x08	Окну будут посылаться сообщения о двойном щелчке кнопки мыши.
CS_OWNDC	0x20	Каждому экземпляру окна присваивается собственный контекст изображения.
CS_BYTEALIGNWINDOW	0x2000	Выравнивание окна: bспользование границы по байту по оси x.
CS_PUBLICCLASS CS_GLOBALCLASS	0x4000	Определяется глобальный класс окон.

Члены структуры функции регистрации класса окна

style — устанавливает стиль(и) класса. Этот член структуры может быть любой комбинацией стилей класса.

Имя	Значение	Описание
CS_CLASSDC	0x40	Классу окна присваивается собственный контекст изображения,который можно разделить между копиями.
CS_PARENTDC	0x80	Классу окна передается контекст изображения родительского окна.
CS_NOKEYCVT	0x100	Отключается преобразование виртуальных клавиш.
CS_NOCLOSE	0x200	Незакрываемое окно: в системном меню блокируется выбор пункта закрытия окна.
CS_SAVEBITS	0x800	Часть изображения на экране, закрытая окном, сохраняется.
CS_BYTEALIGNCLIENT	0x1000	Выравнивание клиентской области окна: использование границы по байту по оси x.

Члены структуры функции регистрации класса окна

lpfnWndProc — указатель на оконную процедуру.

cbClsExtra — устанавливает число дополнительных байт, которые размещаются вслед за структурой класса окна. Система инициализирует эти байты нулями, в большинстве случаев равен 0.

cbWndExtra— устанавливает число дополнительных байтов, которые размещаются вслед за экземпляром окна. Система инициализирует байты нулями.

hInstance — дескриптор экземпляра, который содержит оконную процедуру для класса.

hIcon — дескриптор значка класса, дескриптор ресурса значка. Если этот член структуры — NULL, система предоставляет заданный по умолчанию значок.

Члены структуры функции регистрации класса окна

hCursor — дескриптор курсора класса, дескриптор ресурса курсора. Если этот член структуры — NULL, приложение устанавливает форму курсора всякий раз, когда мышь перемещается в окно прикладной программы.

hbrBackground — дескриптор кисти фона класса, дескриптор физической кисти, которая используется, чтобы красить цветом фона, или код цвета, преобразованный к типу HBRUSH.

lpszMenuName — указатель на символьную строку с символом конца строки (‘\0’), которая устанавливает имя ресурса меню класса. Можно использовать целое число, чтобы идентифицировать меню с помощью макроса MAKEINTRESOURCE(int). Если этот член структуры — NULL, окна, принадлежащие этому классу, не имеют заданного по умолчанию меню.

lpszClassName — указатель на символьную строку с именем класса, оканчивающуюся ‘\0’.

Создание окна

Создание окна осуществляется функцией

```
HWND WINAPI CreateWindow(
_In_opt_  LPCTSTR lpClassName,
_In_opt_  LPCTSTR lpWindowName,
_In_      DWORD dwStyle,
_In_      int x,
_In_      int y,
_In_      int nWidth,
_In_      int nHeight,
_In_opt_  HWND hWndParent,
_In_opt_  HMENU hMenu,
_In_opt_  HINSTANCE hInstance,
_In_opt_  LPVOID lpParam );
```

Прототип функции находится в **файле библиотеки user32.dll.**

Возвращаемое значение – **дескриптор создаваемого окна**. В случае невозможности создать окно возвращается NULL.

Аргументы функции создания окна

lpClassName – указывает на строку с ‘\0’ в конце, которая определяет имя класса окна. Имя класса может быть зарегистрированным функцией RegisterClass или любым из предопределенных имен класса элементов управления.

lpWindowName — указывает на строку с ‘\0’ в конце, которая определяет имя окна.

x — определяет координату левой стороны окна относительно левой стороны экрана. Измеряется в единицах измерения устройства, чаще всего в точках (pt). Для дочернего окна определяет координату левой стороны относительно начальной координаты родительского окна. Если установлен как CW_USEDEFAULT, Windows выбирает заданную по умолчанию позицию окна.

y – определяет координату верхней стороны окна относительно верхней стороны экрана. Измеряется в единицах измерения устройства, чаще всего в точках (pt). Для дочернего окна определяет координату верхней стороны относительно начальной координаты родительского окна.

Аргументы функции создания окна

dwStyle — определяет стиль создаваемого окна.

Имя	Значение	Описание
WS_BORDER	0x00800000	Окно имеет тонкую границу в виде линии.
WS_CAPTION	0x00C00000	Окно имеет строку заголовка.
WS_CHILD	0x40000000	Окно является дочерним.
WS_DISABLED	0x08000000	Окно является изначально неактивным.
WS_GROUP	0x00020000	Окно группирует другие управляющие элементы.
WS_HSCROLL	0x00100000	Окно содержит горизонтальную полосу прокрутки.
WS_MAXIMIZE	0x01000000	Исходный размер окна – во весь экран.
WS_MINIMIZE	0x20000000	Исходно окно свернуто.
WS_OVERLAPPED	0x00000000	Окно может быть перекрыто другими окнами.
WS_POPUP	0x80000000	Всплывающее окно.
WS_SYSMENU	0x00080000	Окно имеет системное меню в строке заголовка.
WS_VISIBLE	0x10000000	Окно изначально видимое.
WS_VSCROLL	0x00200000	Окно имеет вертикальную полосу прокрутки.

Аргументы функции создания окна

Width – определяет ширину окна в единицах измерения устройства. Если параметр соответствует CW_USEDEFAULT, Windows выбирает заданную по умолчанию ширину и высоту для окна.

nHeight – определяет высоту окна в единицах измерения устройства.

hWndParent – дескриптор родительского окна.

hMenu – идентифицирует меню, которое будет использоваться окном. Этот параметр может быть NULL, если меню класса будет использовано.

hInstance — идентифицирует экземпляр модуля, который будет связан с окном.

lpParam — указывает на значение, переданное окну при создании.

Отображение и перерисовка окна

Отображение окна осуществляется функцией

```
BOOL WINAPI ShowWindow(
  _In_  HWND hWnd,
  _In_  int nCmdShow);
```

Прототип функции находится в **файле библиотеки user32.dll.**
Возвращаемое значение: 1 – успешное отображение окна, 0 – ошибка.

Аргументы функции:
hWnd – дескриптор отображаемого окна.
nCmdShow – константа, определяющая, как будет отображаться окно согласно таблице.

Перерисовка окна осуществляется функцией

```
BOOL UpdateWindow(_In_  HWND hWnd);
```

Прототип функции находится в **файле библиотеки user32.dll.**
Возвращаемое значение: 1 – успешная перерисовка окна, 0 – ошибка.
Аргумент функции hWnd – дескриптор окна.

Цикл обработки сообщений

После вызова функции UpdateWindow, окно окончательно выведено на экран. Теперь программа должна подготовить себя для получения информации от пользователя через клавиатуру и мышь. Windows поддерживает «очередь сообщений» (message queue) для каждой программы, работающей в данный момент в системе Windows. Когда происходит ввод информации, Windows преобразует ее в «сообщение», которое помещается в очередь сообщений программы. Программа извлекает сообщения из очереди сообщений, выполняя блок команд, известный как «цикл обработки сообщений» (message loop):

```
while(GetMessage(&msg,NULL,0,0))
{
  TranslateMessage(&msg);
  DispatchMessage(&msg);
}
```

Для получения сообщения из очереди используется функция:

```
BOOL WINAPI GetMessage(
_Out_     LPMSG lpMsg,
_In_opt_  HWND hWnd,
_In_      UINT wMsgFilterMin,
_In_      UINT wMsgFilterMax);
```

Прототип функции находится в файле библиотеки user32.dll.

В случае получения из очереди сообщения, отличного от WM_QUIT, возвращает ненулевое значение.

Аргументы функции

lpMsg — указатель на структуру сообщения.

```
typedef struct MSG {
  HWND hwnd; // дескриптор окна, очередь сообщений которого просматривается
  UINT message; // идентификатор сообщения
  WPARAM wParam; // дополнительная информация о сообщении,
  LPARAM lParam; // зависит от идентификатора сообщения
  DWORD time; // время помещения сообщения в очередь
  POINT pt; // структура, содержащая координаты курсора в момент помещения сообщения в очередь
} MSG;
```

Структура POINT имеет вид

```
typedef struct POINT
{
  LONG x; // координата x
  LONG y; // координата y
} POINT;
```

Аргументы функции

hWnd — дескриптор окна, очередь для которого просматривается.

wMsgFilterMin — нижняя граница фильтра идентификаторов сообщений.

wMsgFilterMax — верхняя граница фильтра идентификаторов сообщений.

Функция
BOOL WINAPI TranslateMessage(_In_ const MSG *lpMsg);

передает аргумент — структуру msg обратно в Windows для преобразования какого-либо сообщения с клавиатуры. Возвращает ненулевое значение в случае успешной расшифровки сообщения, 0 – ошибка.

Функция
LRESULT WINAPI DispatchMessage(_In_ const MSG *lpmsg);

передает аргумент — структуру msg обратно в Windows. Windows отправляет сообщение для его обработки соответствующей оконной процедуре — таким образом, Windows вызывает соответствующую оконную функцию, указанную при регистрации класса окна.

Аргументы функции

После того, как оконная функция обработает сообщение, оно возвращается в Windows, которая все еще обслуживает вызов функции **DispatchMessage**.

Когда Windows возвращает управление в стартовую функцию **WinMain**() к следующему за вызовом **DispatchMessage** коду, цикл обработки сообщений в очередной раз возобновляет работу, вызывая **GetMessage**.

Возвращает значение, определяемое оконной функцией, которое чаще всего игнорируется.

Прототипы функций находятся в **файле библиотеки user32.dll.**

Пример стартовой функции, создающей и выводящей окно размером 500x300 точек:

```
#include <windows.h>
LONG WINAPI WndProc(HWND, UINT, WPARAM,LPARAM);
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
 HWND hwnd; // дескриптор окна
 MSG msg; // структура сообщения
 WNDCLASS w; // структура класса окна
 // Регистрация класса окна
 memset(&w,0,sizeof(WNDCLASS));
 w.style = CS_HREDRAW | CS_VREDRAW;
 w.lpfnWndProc = WndProc; // имя оконной функции
 w.hInstance = hInstance;
 w.hbrBackground = (HBRUSH)(WHITE_BRUSH);
 w.lpszClassName = "My Class";
 RegisterClass(&w);
 // Создание окна
 hwnd = CreateWindow("My Class", "Окно пользователя",
 WS_OVERLAPPEDWINDOW, 500, 300, 500, 380, NULL, NULL, hInstance, NULL);
 ShowWindow(hwnd,nCmdShow); // отображение
 UpdateWindow(hwnd);         // перерисовка
```

Продолжение

```
 // Цикл обработки сообщений
  while(GetMessage(&msg,NULL,0,0))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return msg.wParam;
}

  ShowWindow(hwnd,nCmdShow); // отображение
  UpdateWindow(hwnd);          // перерисовка
  // Цикл обработки сообщений
  while(GetMessage(&msg,NULL,0,0))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return msg.wParam;
}
```

Примечание: Для корректной сборки приложения используется многобайтовая кодировка.

Оконная функция — обработка сообщений окна

Оконная функция предназначена для обработки сообщений окна. Функция обработки сообщений окна организована по принципу ветвления, состоящего из последовательной проверки типа сообщения. При совпадении типа сообщения, переданного в структуре Message с соответствующей веткой, осуществляется его обработка. Минимальный вид оконной функции представлен ниже.

```
LONG WINAPI WndProc(HWND hwnd, UINT Message, WPARAM wparam, LP
ARAM lparam)
{
  switch (Message)
  {
    case WM_DESTROY:
      PostQuitMessage(0);
      break;
    default:
      return DefWindowProc(hwnd, Message, wparam, lparam);
  }
  return 0;
}
```

Оконная функция — обработка сообщений окна

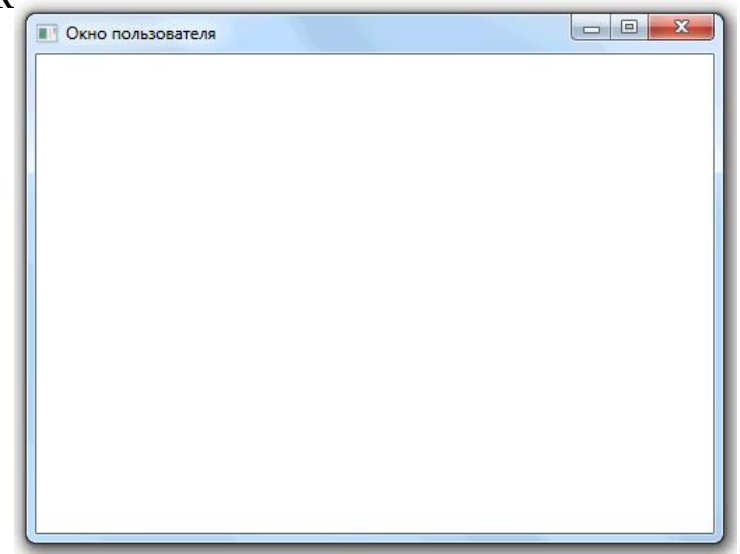
4 аргумента оконной функции идентичны первым четырем полям структуры сообщения MSG.

В примере обрабатывается только один тип сообщения WM_DESTROY, которое передается оконной функции при закрытии окна.

Вызов функции DefWindowProc() обрабатывает по умолчанию все сообщения, которые не обрабатывает оконная процедура.

Функция PostQuitMessage() сообщает Windows, что данный поток запрашивает завершение. Аргументом является целочисленное значение, которое функция вернет операционной системе.

Результат выполнения программы, выводящей окно:



Элементы управления окна

Главным элементом программы в среде Windows является окно. Окно может содержать элементы управления: кнопки, списки, окна редактирования и др. Эти элементы также являются окнами, но обладающими особым свойством: события, происходящие с этими элементами (и самим окном), приводят к приходу сообщений в процедуру окна.

Системный класс	Предназначение
BUTTON	Кнопка.
COMBOBOX	Комбинированное окно (окно со списком и поля выбора).
EDIT	Окно редактирования текста.
LISTBOX	Окно со списком
SCROLLBAR	Полоса прокрутки
STATIC	Статический элемент (текст)

Создание элементов управления окна

Создание элементов управления окна осуществляется функцией

```
HWND WINAPI CreateWindow(
  _In_opt_ LPCTSTR lpClassName, // имя предопределенного класса
  _In_opt_ LPCTSTR lpWindowName, // текст
  _In_ DWORD dwStyle, // стиль
  _In_ int x, // координата x
  _In_ int y, // координата y
  _In_ int nWidth, // ширина
  _In_ int nHeight, // высота
  _In_opt_ HWND hWndParent, // дескриптор родительского окна
  _In_opt_ HMENU hMenu, // номер пункта меню
  _In_opt_ HINSTANCE hInstance, // дескриптор приложения
  _In_opt_ LPVOID lpParam ); // NULL
```

Функция возвращает дескриптор элемента управления окна, который может быть впоследствии использован для анализа элемента управления, с которым связано обрабатываемое событие.

Таблицу стилей элементов управления окна можно устанавливать в параметре dwStyle как и для создания родительского окна. При этом обязательно указывается, что создаваемое окно является дочерним — WS_CHILD.

КНОПКА

Кнопка — маленькое прямоугольное дочернее окно, которое представляет собой кнопку, по которой пользователь может щелкать мышью, чтобы включить или выключить ее. Кнопки управления могут использоваться самостоятельно или в группах, и они могут или быть подписаны или появляться без текста. Кнопки управления обычно изменяют свой вид, когда пользователь щелкает мышью по ним.

При нажатии кнопки операционная система генерирует сообщение **WM_COMMAND** с параметром **lParam**, соответствующим дескриптору кнопки.

Обработка нажатия кнопки:

```
LONG WINAPI WndProc(HWND hwnd, UINT Message, WPARAM wparam, LP
ARAM lparam) {
  ...
  switch (Message) {
    case WM_COMMAND:
      if(lparam == (LPARAM)hBtn) {
        //обработка нажатия кнопки
      }
      break;
    ...  }}
```

ПОЛЕ РЕДАКТИРОВАНИЯ

Поле редактирования — прямоугольное дочернее окно, внутри которого пользователь может напечатать с клавиатуры текст.

Для считывания информации из поля редактирования используется функция

```
int WINAPI GetWindowText(
  _In_   HWND hWnd,   // дескриптор поля
  _Out_  LPTSTR lpString,   // указатель на текстовую строку
  _In_   int nMaxCount );       // максимальное количество символов
```

Возвращаемое значение – длина считанной текстовой строки.

Для установки текста в поле редактирования используется функция

```
BOOL WINAPI SetWindowText(
  _In_      HWND hWnd,   // дескриптор поля
  _In_opt_  LPCTSTR lpString ); // указатель на текстовую строку
```

В случае успешного завершения функция возвращает ненулевое значение.

Пример на C++ Найти сумму двух чисел

Программа создает два поля редактирования для ввода чисел и кнопку «Рассчитать», при нажатии на которую выводится статический текст, соответствующий сумме.

Для перевода строки в целое число и обратно использованы функции StrToInt и IntToStr, описанные в статье Преобразование строки в число и обратно. Также использовалась многобайтовая кодировка.

```
#include <windows.h>
LONG WINAPI WndProc(HWND, UINT, WPARAM, LPARAM); // функция
обработки сообщений окна
// Функция преобразования строки в число
int StrToInt(char *s)
{
  int temp = 0; // число
  int i = 0;
  int sign = 0; // знак числа 0- положительное, 1 - отрицательное
  if (s[i] == '-')
```

Пример на C++ Найти сумму двух чисел

```
{
    sign = 1;
    i++;
  }
  while (s[i] >= 0x30 && s[i] <= 0x39)
  {
    temp = temp + (s[i] & 0x0F);
    temp = temp * 10;
    i++;
  }
  temp = temp / 10;
  if (sign == 1)
    temp = -temp;
  return(temp);
}
```

Пример на C++ Найти сумму двух чисел

```
// Функция преобразования числа в строку
char* IntToStr(int n)
{
  char s[40], t, *temp;
  int i, k;
  int sign = 0;
  i = 0;
  k = n;
  if (k<0)
  {
    sign = 1;
    k = -k;
  }
  do {
    t = k % 10;
    k = k / 10;
    s[i] = t | 0x30;
    i++;
  } while (k>0);
  if (sign == 1)
  {
    s[i] = '-';
    i++;
  }
  temp = new char[i];
  k = 0;
  i--;
  while (i >= 0) {
    temp[k] = s[i];
    i--; k++;
  }
  temp[k] = '\0';
  return(temp);
}
```

Пример на C++ Найти сумму двух чисел

```
// Стартовая функция
int  WINAPI  WinMain(HINSTANCE  hInstance,
  HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
  HWND hwnd; // дескриптор окна
  MSG msg;   // структура сообщения
  WNDCLASS w; // структура класса окна
  memset(&w, 0, sizeof(WNDCLASS)); // очистка памяти для структуры
  w.style = CS_HREDRAW | CS_VREDRAW;
  w.lpfnWndProc = WndProc;
  w.hInstance = hInstance;
  w.hbrBackground = CreateSolidBrush(0x00FFFFFF);
  w.lpszClassName = "MyClass";
  RegisterClass(&w); // регистрация класса окна
  // Создание окна
  hwnd = CreateWindow("MyClass", "Сумма двух чисел",
    WS_OVERLAPPEDWINDOW,
    500, 300, 500, 380,
    NULL, NULL, hInstance, NULL);
  ShowWindow(hwnd, nCmdShow); // отображение окна
  UpdateWindow(hwnd);         // перерисовка окна
```

Пример на C++ Найти сумму двух чисел

```
// Цикл обработки сообщений
  while (GetMessage(&msg, NULL, 0, 0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
 return msg.wParam;
}
// Функция обработки сообщений
LONG WINAPI WndProc(HWND hwnd, UINT Message,
  WPARAM wparam, LPARAM lparam) {
  HDC hdc;
  HINSTANCE hInst;
  PAINTSTRUCT ps;
  static HWND hBtn; // дескриптор кнопки
  static HWND hEdt1, hEdt2; // дескрипторы полей редактирования
  static HWND hStat; // дескриптор статического текста
  TCHAR StrA[20];
  int a, b, sum, Len;
  switch (Message) {
  case WM_CREATE: // сообщение создания окна
    hInst = ((LPCREATESTRUCT)lparam)->hInstance; // дескриптор
приложения
```

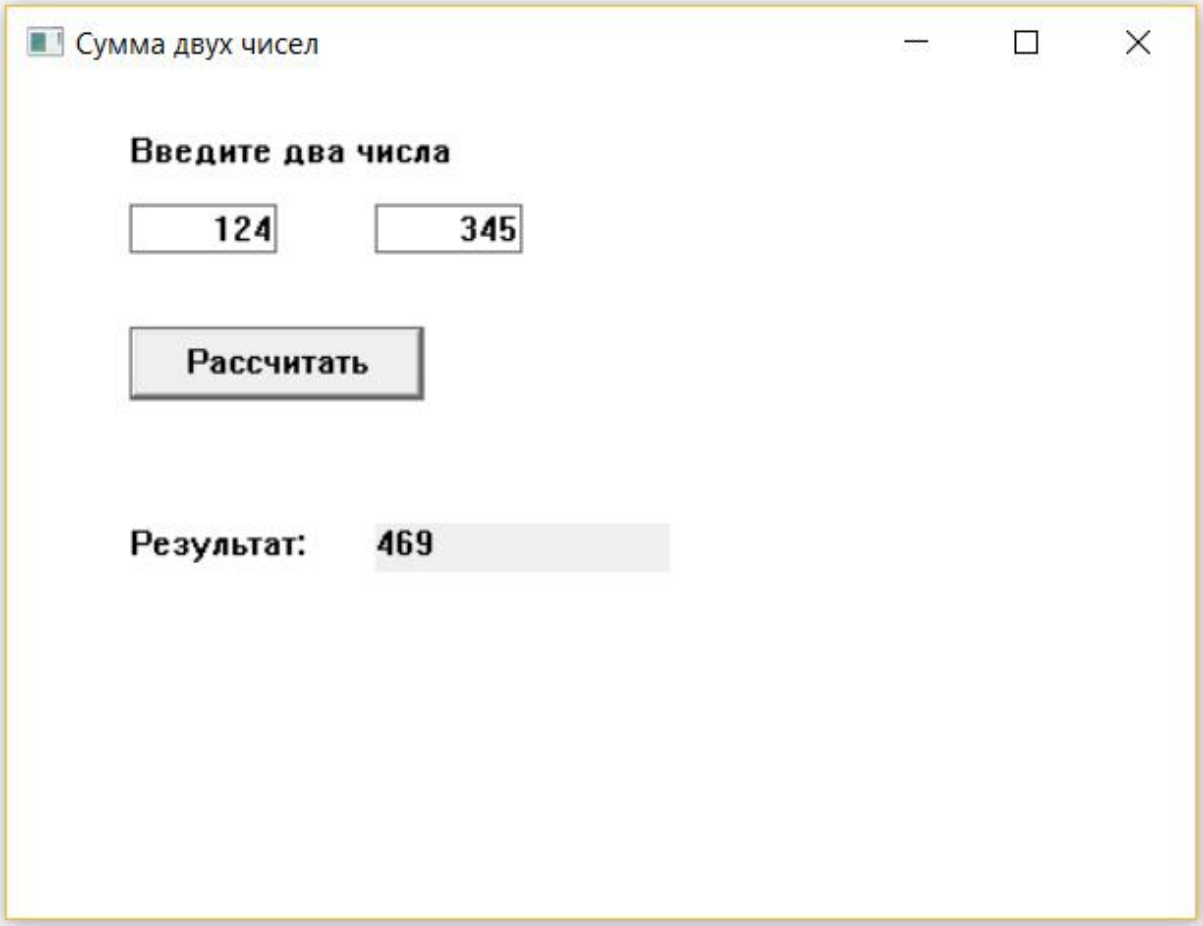
Пример на C++ Найти сумму двух чисел

```
// Создаем и показываем первое поле редактирования
  hEdt1 = CreateWindow("edit", "0",
    WS_CHILD | WS_VISIBLE | WS_BORDER | ES_RIGHT, 50, 50, 60, 20,
    hwnd, 0, hInst, NULL);
  ShowWindow(hEdt1, SW_SHOWNORMAL);
  // Создаем и показываем второе поле редактирования
  hEdt2 = CreateWindow("edit", "0",
    WS_CHILD | WS_VISIBLE | WS_BORDER | ES_RIGHT, 150, 50, 60,
    20, hwnd, 0, hInst, NULL);
  ShowWindow(hEdt2, SW_SHOWNORMAL);
  // Создаем и показываем кнопку
  hBtn = CreateWindow("button", "Рассчитать",
    WS_CHILD | WS_VISIBLE | WS_BORDER,
    50, 100, 120, 30, hwnd, 0, hInst, NULL);
  ShowWindow(hBtn, SW_SHOWNORMAL);
  // Создаем и показываем поле текста для результата
  hStat = CreateWindow("static", "0", WS_CHILD | WS_VISIBLE,
    150, 180, 120, 20, hwnd, 0, hInst, NULL);
  ShowWindow(hStat, SW_SHOWNORMAL);
  break;
```

Пример на C++ Найти сумму двух чисел

```
   case WM_COMMAND:  // сообщение о команде
    if (lparam == (LPARAM)hBtn)    // если нажали на кнопку
    {
      Len = GetWindowText(hEdt1, StrA, 20);
      a = StrToInt(StrA); // считываем число из первого поля
      Len = GetWindowText(hEdt2, StrA, 20);
      b = StrToInt(StrA); // считываем число из второго поля
      sum = a + b;  // находим сумму двух чисел
      SetWindowText(hStat, IntToStr(sum)); // выводим результат в статическое поле
    }
    break;
  case WM_PAINT: // перерисовка окна
    hdc = BeginPaint(hwnd, &ps); // начало перерисовки
    TextOut(hdc, 50, 20, "Введите два числа", 18); // вывод текстовых сообщений
    TextOut(hdc, 50, 180, "Результат:", 10);
    EndPaint(hwnd, &ps); // конец перерисовки
    break;
  case WM_DESTROY: // закрытие окна
    PostQuitMessage(0);
    break;
  default: // обработка сообщения по умолчанию
    return DefWindowProc(hwnd, Message, wparam, lparam);
  }
  return 0;
}
```

Пример на C++ Найти сумму двух чисел



Графический пользовательский интерфейс. Windows Forms

Основы алгоритмизации и

Основные понятия

Windows Forms — это набор средств для создания оконных windows-приложений, выполняющихся в среде CLR.

CLR- Common Language Runtime — общеязыковая исполняющая среда.

Форма и используемые с ней элементы управления представлены классом **C++/CLI** (Common Language Intermediate).

При создании проекта приложения создается как окно приложения **Windows Forms**,

Разработка приложения

Разработка приложения сводится к выполнению четырех отдельных операций:

1. ***Интерактивное создание графического интерфейса пользователя***

На вкладке ***Form Конструктор (Конструктор формы)***, отображаемой в панели ***Редактор (Редактор)***, путем выбора элементов управления в окне ***Панель элементов*** и их помещения на оконную форму мышкой.

На этом этапе можно также создавать дополнительные окна форм, то есть можно сделать так, что, например, при нажатии пользователем по кнопке, появится новое окно приложения с другими элементами.

2. ***Изменение свойств элементов управления и форм***

В окне ***Свойства*** в соответствии с потребностями приложения.

Разработка приложения

Разработка приложения сводится к выполнению четырех отдельных операций:

3. ***Обработчики событий щелчков для элементов управления***

Можно создавать, дважды щелкая на элементе управления на вкладке ***Form Design (Дизайнер формы)***.

В окне ***Properties (Свойства)*** элемента управления в качестве его обработчика события можно также определять существующую функцию (метод).

4. Для удовлетворения потребностей приложения можно изменять и расширять классы, автоматически создаваемые в результате взаимодействия с вкладкой Form Design.

Класс Form

Класс MyForm — производный от класса Form, определен в пространстве имен System::Windows::Forms.

Класс Form представляет окно приложения или диалогового окна.

Класс MyForm, который определяет окно для пространства имен, наследует все члены класса Form.

В конце класса MyForm содержит определение функции **InitializeComponent() -** эта функция вызывается конструктором для определения окна приложения и любых компонентов, добавляемых в форму.

InitializeComponent()

Первый оператор сохраняет в члене **components** дескриптор объекта **Container**, представляющего коллекцию, которая хранит компоненты графического интерфейса пользователя в списке.

Каждый новый компонент, добавляемый в форму с помощью средств **Form Design**, добавляется в этот объект **Container**.

Остальные операторы функции **InitializeComponent()** определяют свойства объекта **MyForm**. Ни одно из этих свойств не следует изменять непосредственно в коде, но их значения можно выбирать посредством окна **Свойства (Properties)**.

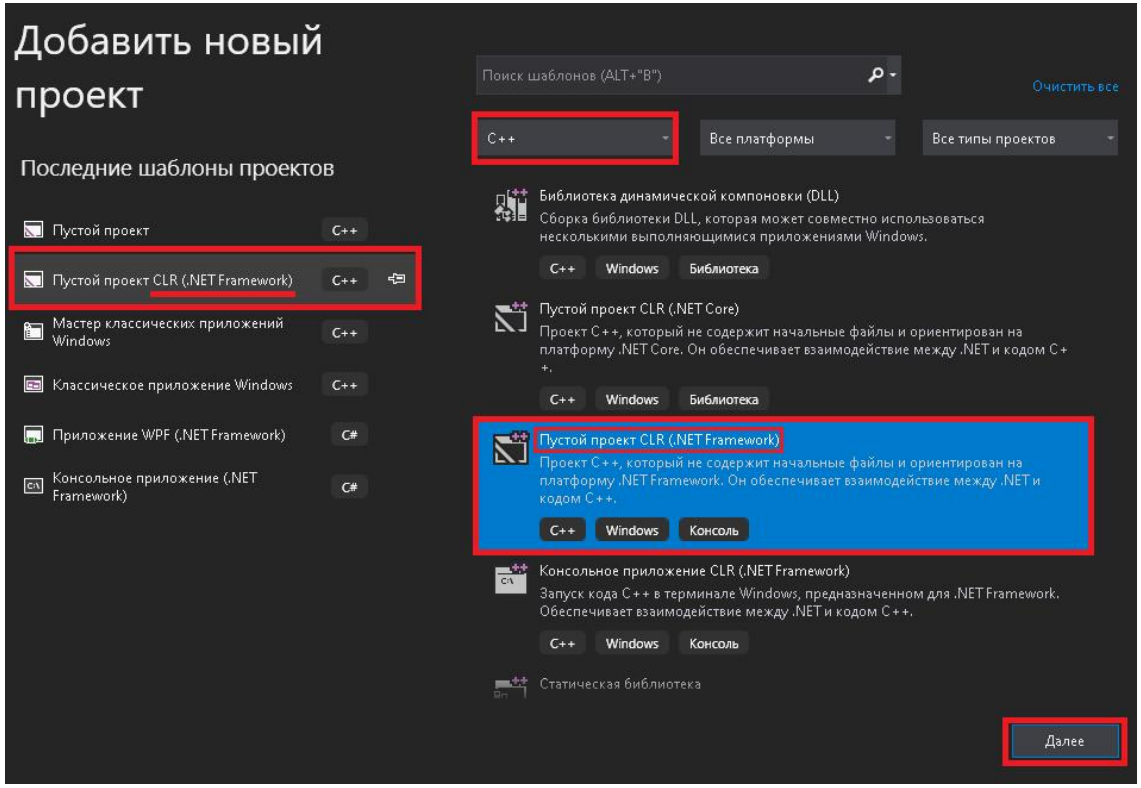
Если переключится обратно на вкладку **Form1.h[Конструктор]** в окне редактора и щелкнуть правой клавишей мыши, выбрать **Свойства**, откроется **окно свойств формы** (один из способов открыть окно Свойств).

Создание полноэкранного приложения

Основы алгоритмизации и

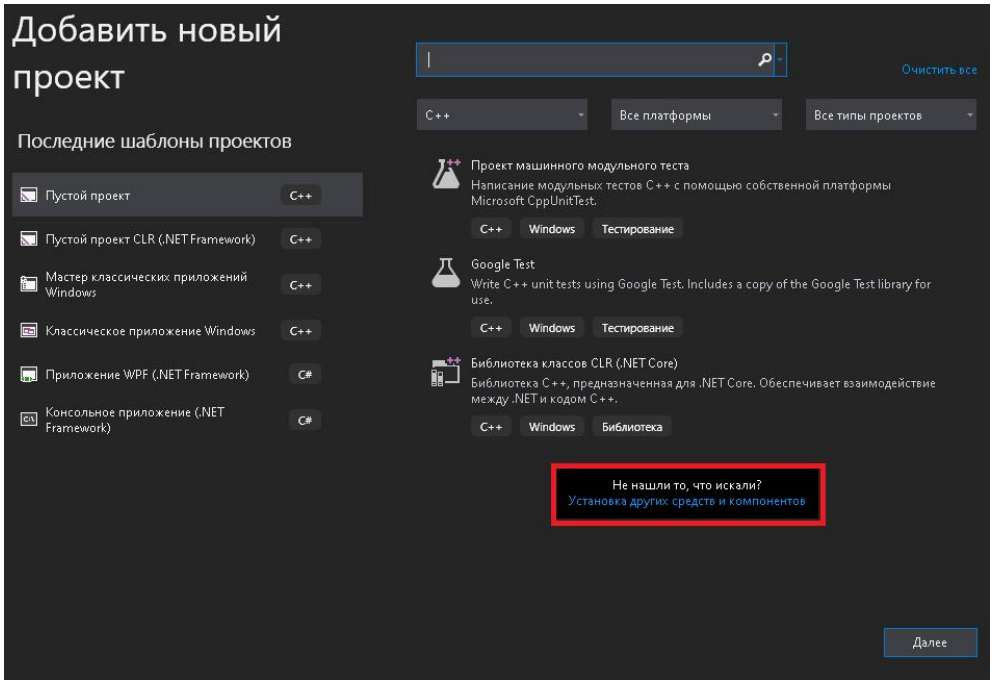
Создание приложения

Запустить MS Visual Studio 2022 и создать новый проект **Пустой проект CLR (.NET Framework)**.



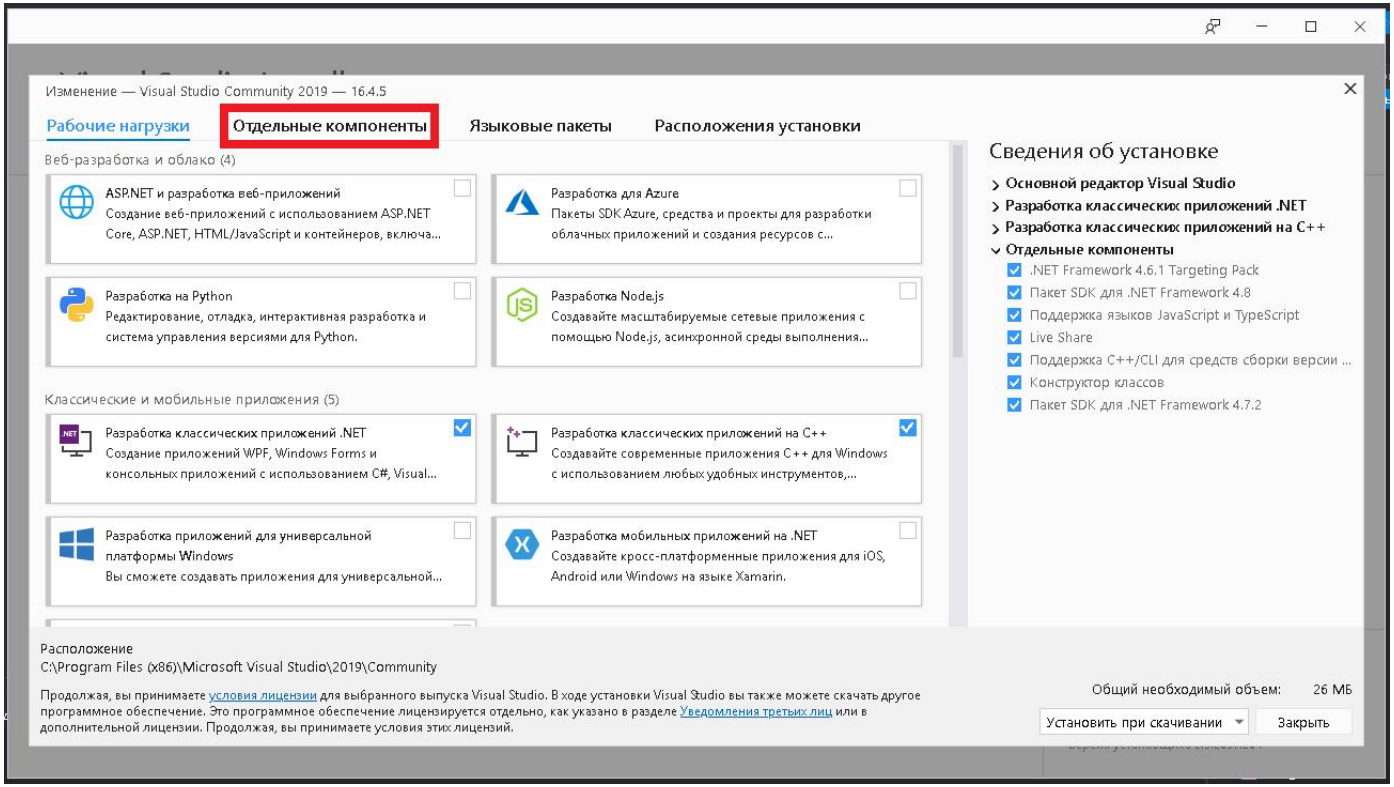
Утановка шаблона

Если не установлен нужный шаблон, то нужно скачать и установить шаблон следующим образом. Прокрутите ползунок выбора доступных проектов в самый низ, чтобы увидеть кнопку Установка других средств и компонентов, нажмите ее. Возможна загрузка обновления установщика.



Установка шаблона

По окончании обновления Установщика VS2022 кликаем на вкладку **Отдельные компоненты:**



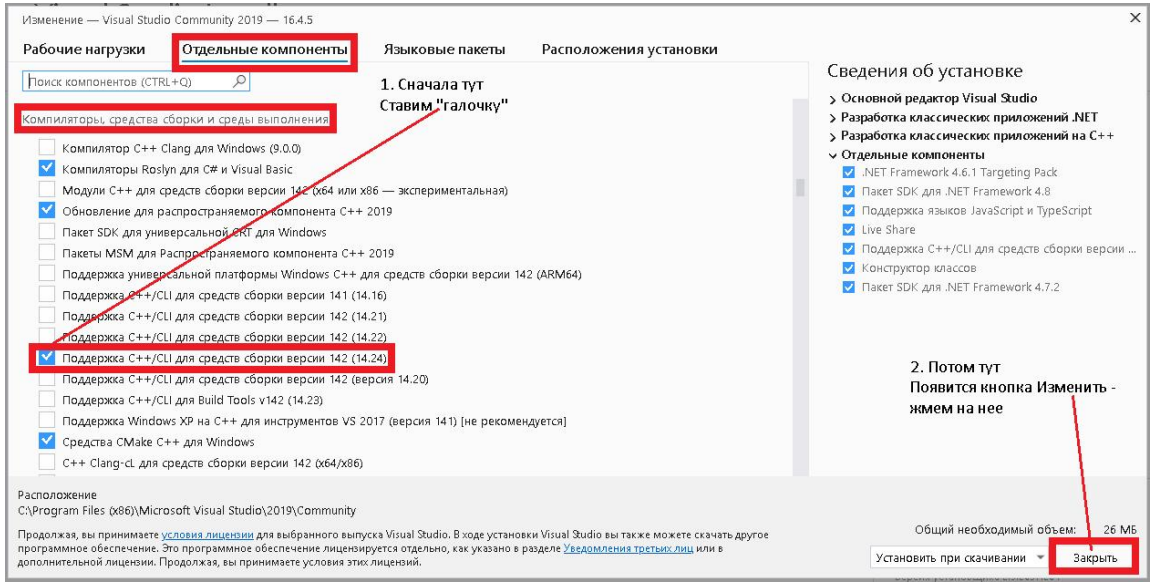
Установка шаблона

Открывается вкладка Отдельные компоненты, в которой «галочками» отображаются уже установленные компоненты и без галочек – те компоненты, которые можно скачать и установить.

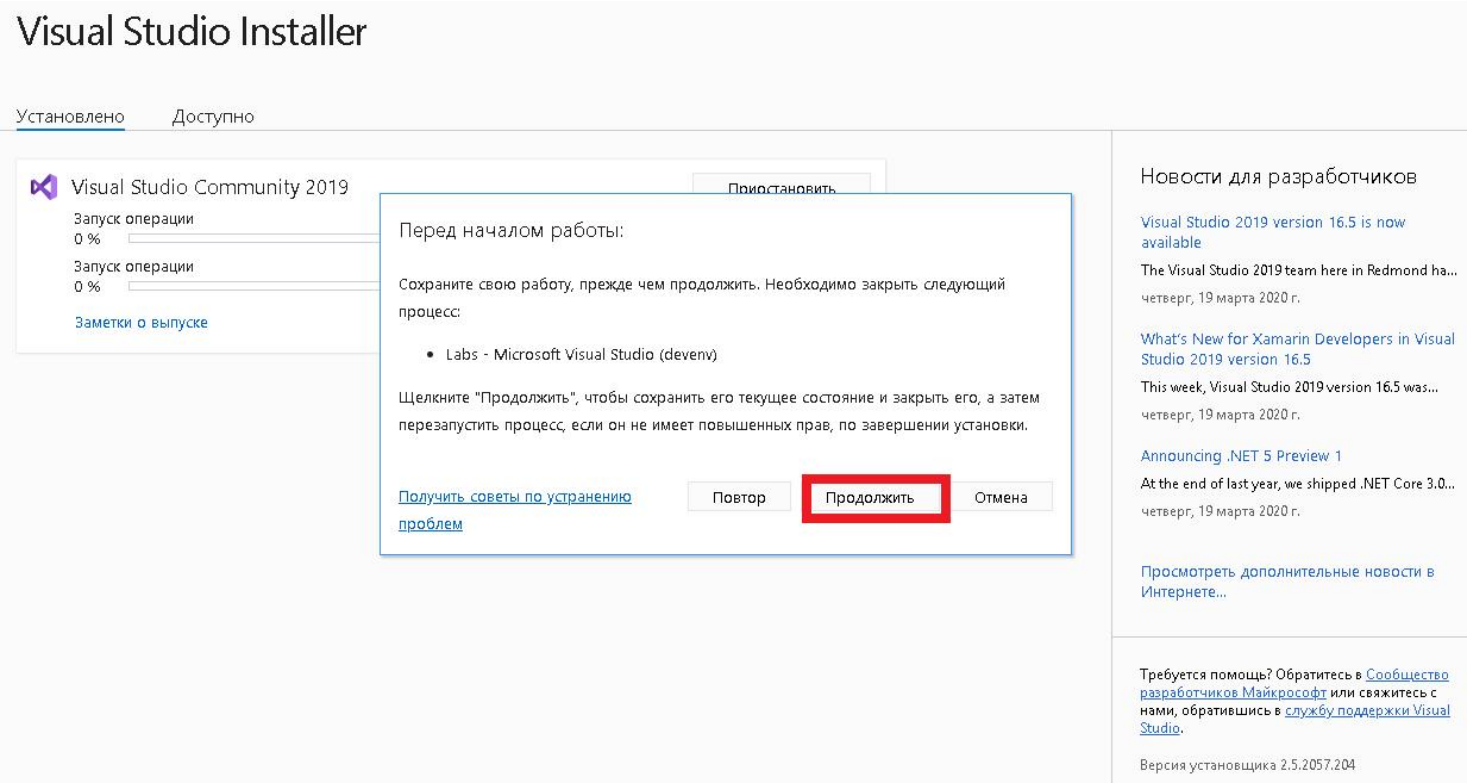
Раздел «Компиляторы, средства сборки и среды выполнения», в котором ищем компонент «Поддержка C++/CLI для средств сборки…» и ставим перед ним «галочку».

Если таких строк будет несколько, выбираем одну самую новую версию (ее номер будет больше остальных).

После этого внизу справа кнопка Закрыть должна поменяться на кнопку Изменить. Жмем на кнопку Изменить и ждем скачивания и установки нового компонента.



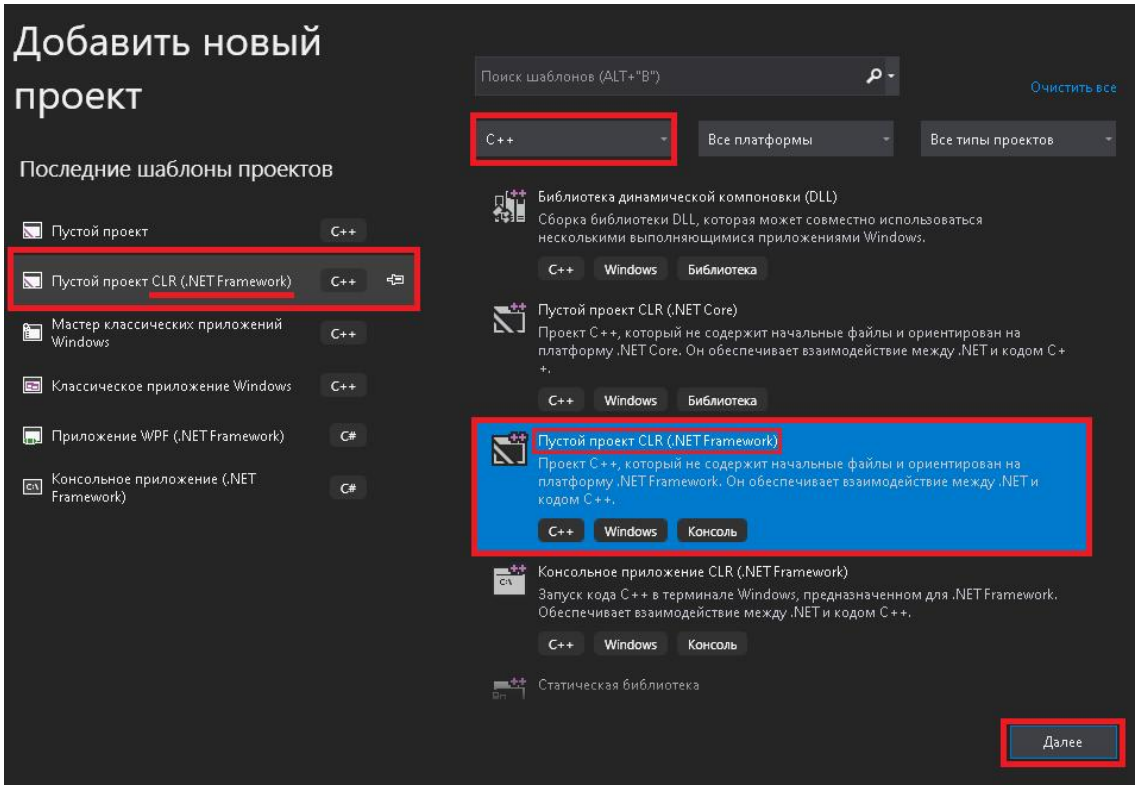
Установка шаблона



Когда MS VS2022 обновится и перезапустится можно снова создавать Пустой проект CLR (.NET Framework) [C++, консольный], который теперь уже должен быть доступен для создания.

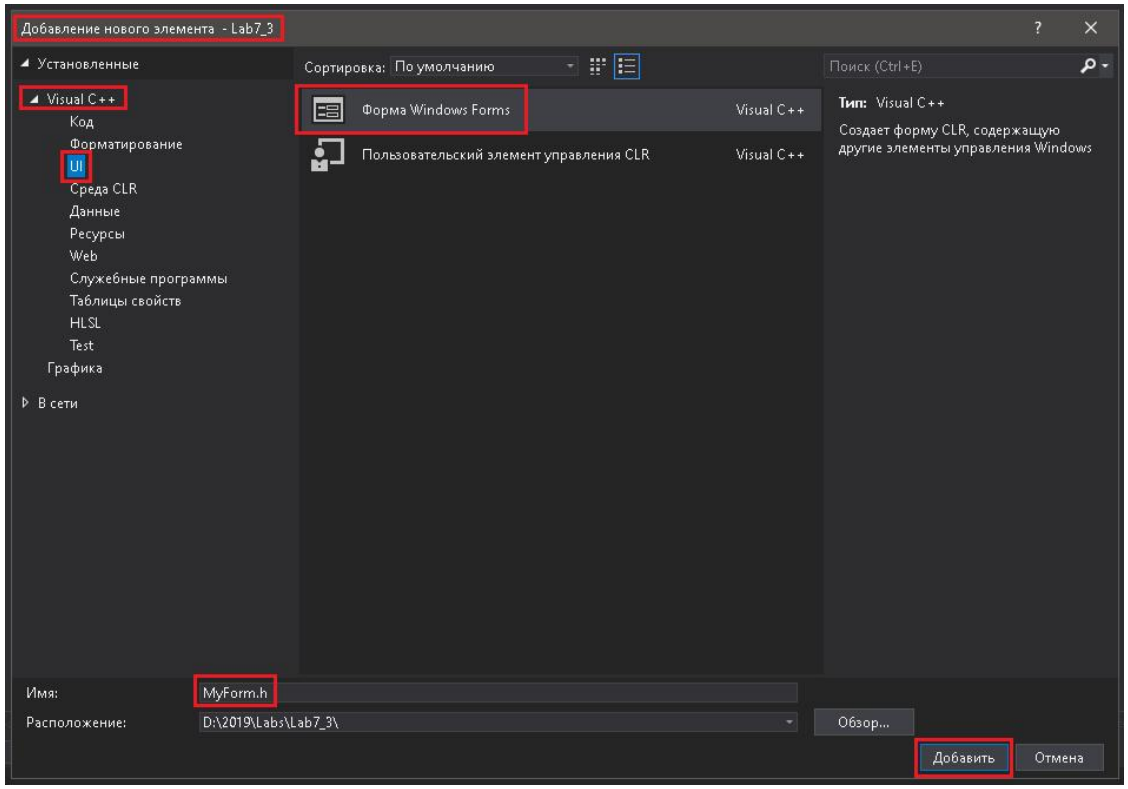
Создание проекта

Когда MS VS2022 обновится и перезапустится можно снова создавать Пустой проект CLR (.NET Framework), который теперь уже должен быть доступен для создания.



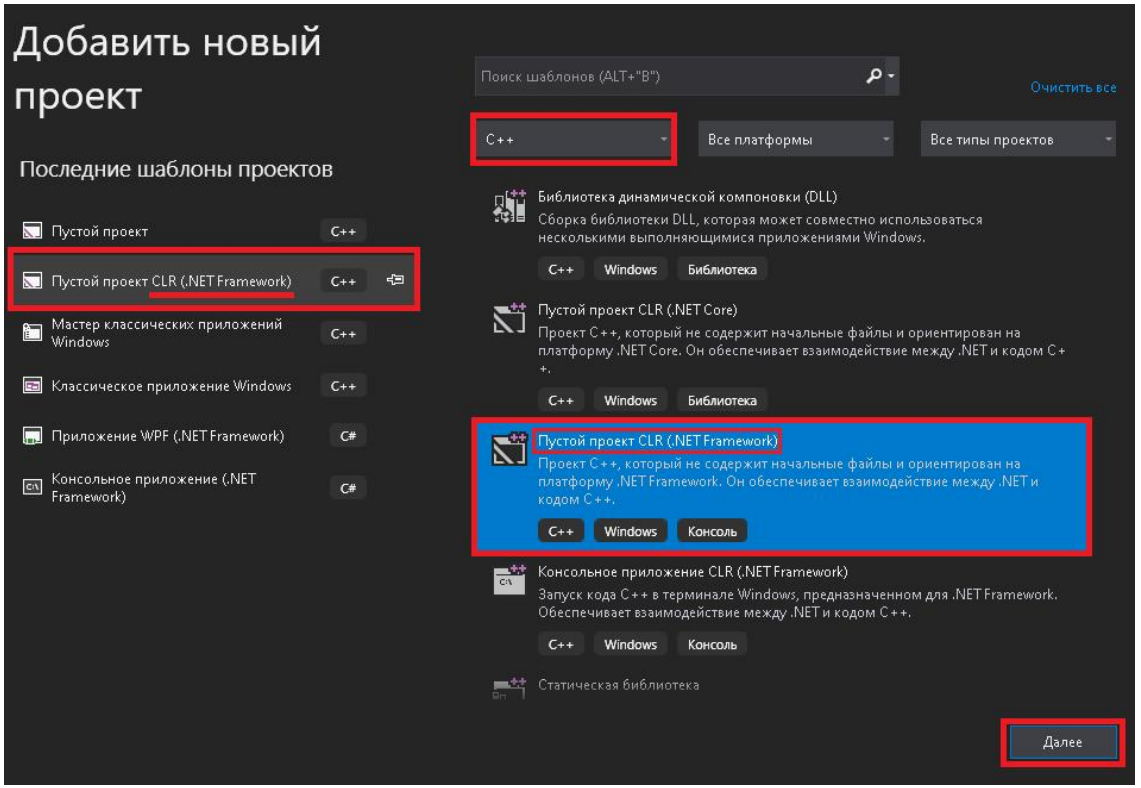
Создание проекта

Даем проекту краткое имя латинскими буквами, проверяем место его сохранения (на диске D:\\) и жмем кнопку Создать. Когда проект создался, назначаем его автозагружаемым.



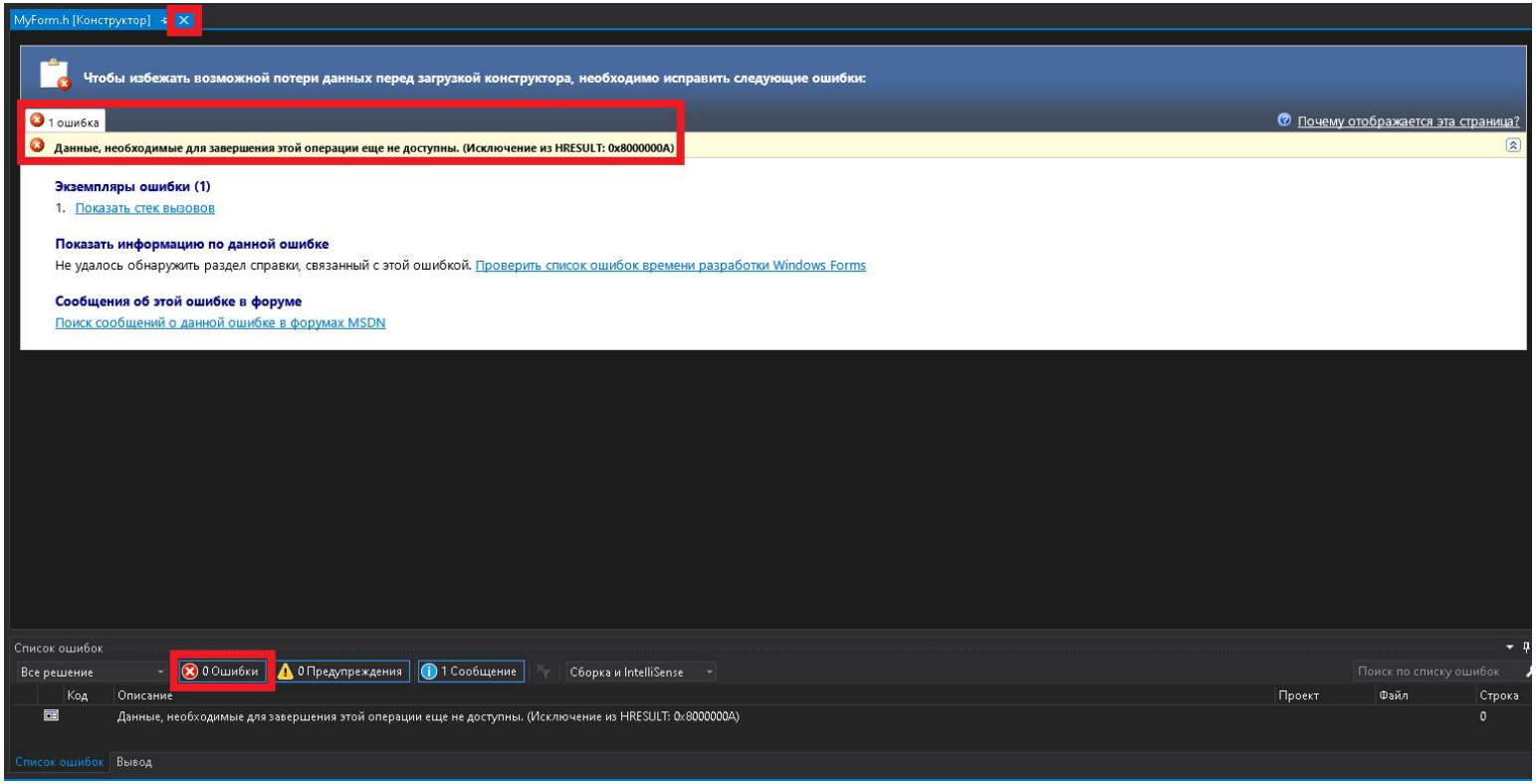
Создание проекта

Когда MS VS2022 обновится и перезапустится можно снова создавать Пустой проект CLR (.NET Framework), который теперь уже должен быть доступен для создания.



Создание проекта

Появившееся окно с сообщением об ошибке закрываем.



Создание проекта

В окне Обозреватель решений жмем мышью по файлу MyForm.cpp, добиваясь его открытия в режиме редактирования кода.
Прописываем следующий код:

```
#include "MyForm.h"
#include <Windows.h>
using namespace Project21; //название проекта
int WINAPI WinMain(HINSTANCE, HINSTANCE,
LPSTR, int) {
		Application::EnableVisualStyles();
		Application::SetCompatibleTextRenderin
gDefault(false);
		Application::Run(gcnew MyForm);
		return 0;
}
```

Проект нужно сохранить и скомпилировать, чтобы создался .exe-файл.