# WINAPI

# WINAPI. Определение, характеристики/описание;

**WINAPI** (Windows API) — это набор интерфейсов программирования приложений (API), которые предоставляются операционной системой Microsoft Windows. Эти интерфейсы позволяют приложениям взаимодействовать с операционной системой, выполняя такие задачи, как работа с окнами, управление памятью, доступ к аппаратному обеспечению, управление процессами и потоками, взаимодействие с файловой системой и многие другие.

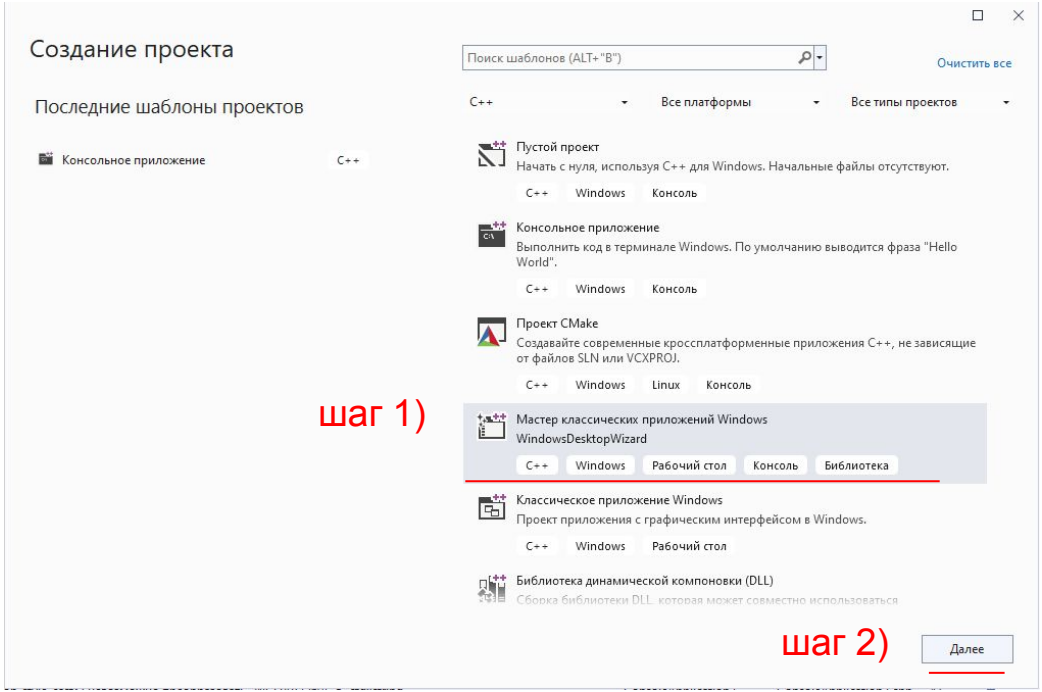
**Характеристики**

1. **Модульность**: API разделены на модули, такие как GDI (Graphics Device Interface), User (пользовательский интерфейс), Kernel (ядро), Network (сеть), и другие. Это позволяет разработчикам использовать только те части API, которые необходимы для их приложений.
2. **Совместимость**: Сохраняет обратную совместимость с предыдущими версиями Windows, что позволяет старым приложениям работать на новых версиях операционной системы.
3. **Многофункциональность**: Обеспечивает широкий спектр функциональных возможностей, включая создание и управление окнами, обработку сообщений, работу с графикой, доступ к сети и устройствам ввода-вывода.
4. **Прямой доступ к системным ресурсам**: Позволяет приложениям выполнять низкоуровневые операции, такие как управление памятью и доступ к аппаратному обеспечению.
5. **Языковая независимость**: Несмотря на то, что API в основном используются с языками C и C++, они также могут быть использованы с другими языками программирования, такими как Pascal, Python, и другие, через специальные интерфейсы и библиотеки обёрток.

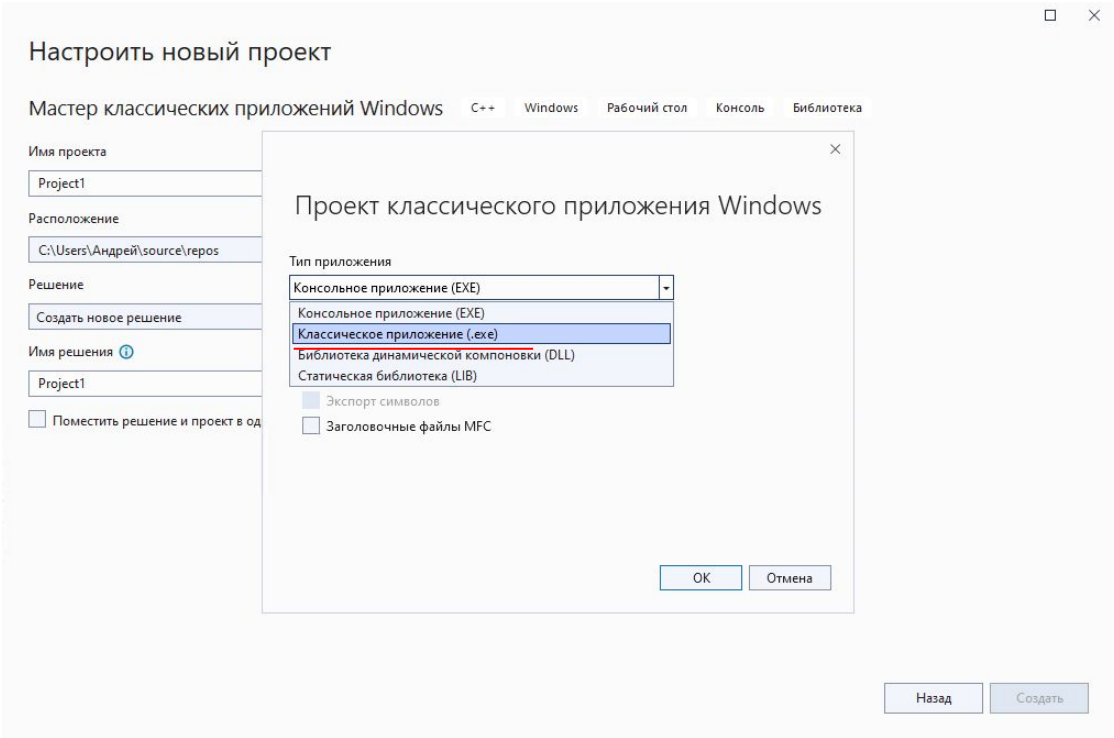
# WINAPI. Задачи/Цели.

- **Разработка пользовательских приложений**: Предоставление разработчикам инструментов для создания приложений с графическим интерфейсом пользователя, которые могут взаимодействовать с элементами управления Windows, такими как кнопки, меню и окна.
- **Управление ресурсами системы**: Обеспечение механизмов для управления системными ресурсами, включая память, процессоры, файловую систему и устройства ввода-вывода.
- **Работа с графикой и мультимедиа**: Предоставление функций для работы с графикой (через GDI и DirectX), аудио и видео, что важно для разработки игр, мультимедийных приложений и графических редакторов.
- **Обеспечение безопасности и управления доступом**: Поддержка механизмов безопасности, таких как управление правами доступа, аутентификация и шифрование, для защиты данных и ресурсов.
- **Поддержка многозадачности и многопоточности**: Обеспечение API для создания и управления процессами и потоками, что позволяет приложениям выполнять несколько задач одновременно и эффективно использовать ресурсы системы.
- **Интеграция с сетью и Интернетом**: Предоставление функций для сетевого программирования, включая поддержку TCP/IP, HTTP и других протоколов, что важно для создания сетевых и веб-приложений.
- **Обратная совместимость**: Поддержка старых приложений на новых версиях Windows, обеспечивая стабильность и непрерывность использования программного обеспечения.

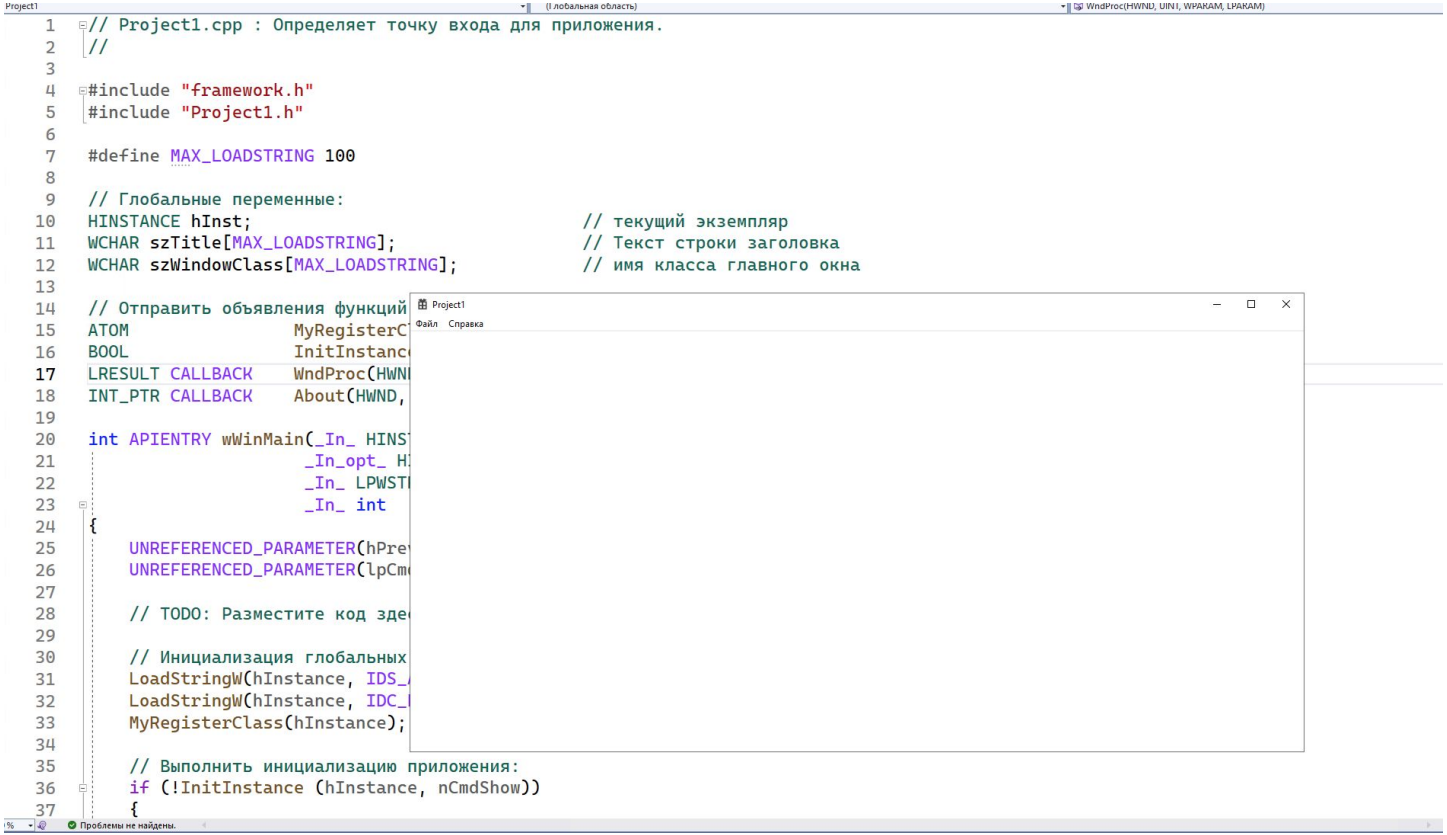
# WINAPI. Создание проекта в Visual Studio.



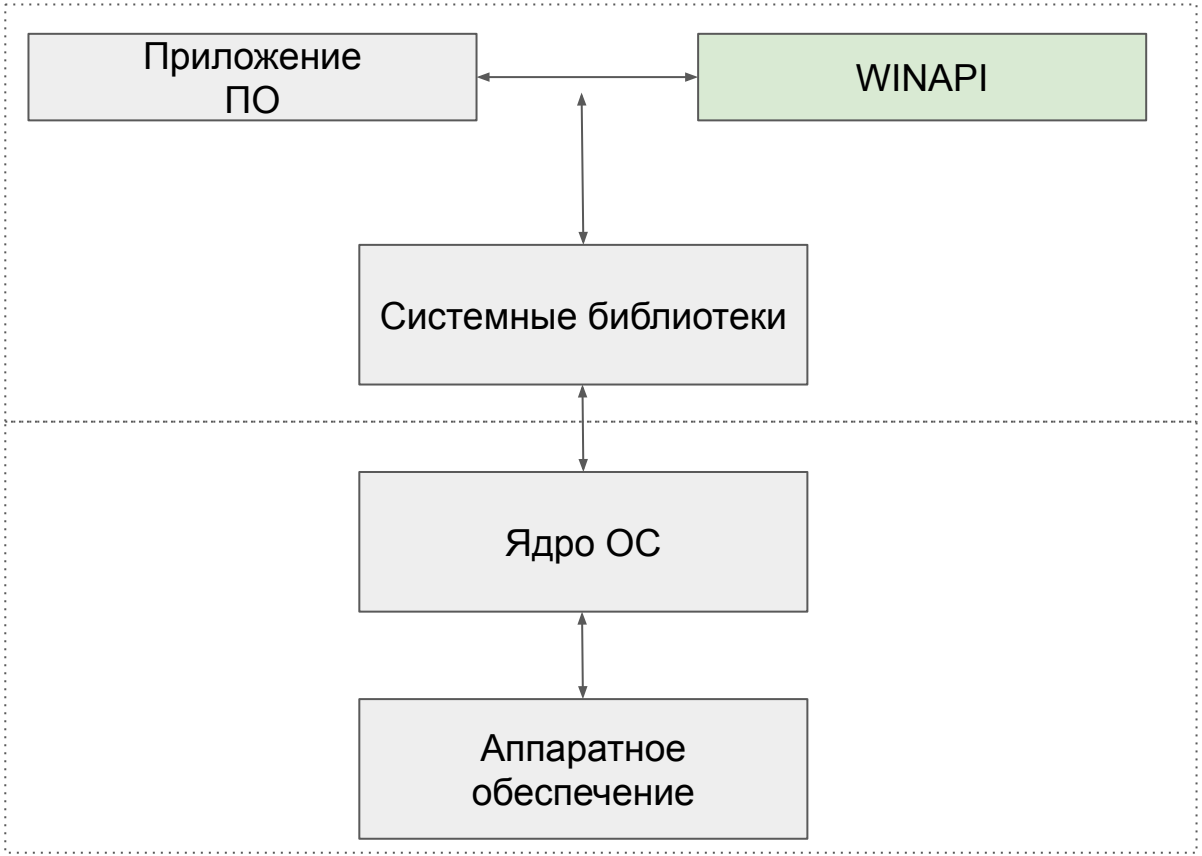
# WINAPI. Настройка проекта.



# WINAPI. Первый запуск.



# WINAPI. Схема работы.




1. **Приложение (Application)**: Исходная точка взаимодействия, где пользователь или разработчик создает программу.
2. **WINAPI (API Functions)**: Приложение вызывает функции WINAPI для выполнения различных задач, таких как создание окон, работа с файлами и сетями, управление памятью и процессами.
3. **Системные библиотеки (System Libraries)**: Функции WINAPI используют системные библиотеки, которые содержат реализацию необходимых операций. Эти библиотеки обеспечивают более высокоуровневый доступ к функциональности операционной системы.
4. **Ядро ОС (Operating System Kernel)**: Системные библиотеки взаимодействуют с ядром операционной системы, которое предоставляет низкоуровневые службы, такие как управление памятью, планирование задач и обработка аппаратных прерываний.
5. **Аппаратное обеспечение (Hardware)**: Ядро ОС напрямую управляет аппаратными ресурсами компьютера, такими как процессор, память, дисковые устройства и сетевые интерфейсы, чтобы выполнить запросы приложения.

# Хендлы (handles). Типы handles.

**Хендл (Handle) в WinAPI** — это абстрактный идентификатор, который используется для представления различных объектов операционной системы. Хендлы позволяют приложениям взаимодействовать с этими объектами, не раскрывая внутреннюю структуру данных, которую они представляют. Они используются для управления ресурсами и обеспечивают уровень абстракции, позволяя разработчикам работать с объектами через единообразный интерфейс. *Типы handles:*

- **HWND (Handle to a Window)**: Представляет окно или элемент управления. Используется для выполнения операций над окнами, таких как изменение размера, перемещение, обновление и обработка сообщений.
- **HINSTANCE (Handle to an Instance)**: Представляет экземпляр приложения или модуля (например, DLL). Используется для получения ресурсов, загрузки библиотек и других операций, связанных с модулем.
- **HMODULE**: Эквивалентен HINSTANCE и также представляет экземпляр приложения или загруженной библиотеки. Эти два типа могут использоваться взаимозаменяемо в большинстве случаев.
- **HANDLE**: Универсальный тип хендла, который может представлять любой объект, такой как файл, процесс, поток или синхронизирующий объект (например, событие или мьютекс).
- **HDC (Handle to a Device Context)**: Представляет контекст устройства, который используется для графического вывода. HDC используется при рисовании графики в окне или другом устройстве вывода.
- **HBITMAP**: Представляет битмап-изображение. Используется для работы с изображениями в памяти, включая создание, изменение и отображение битмапов.
- **HPEN, HBRUSH**: Представляют графические объекты, такие как перо (HPEN) и кисть (HBRUSH), которые используются для рисования линий, контуров и заливки областей.
- **HFONT**: Представляет шрифт. Используется для работы с различными типами шрифтов в графическом выводе текста.
- **HRGN (Handle to a Region)**: Представляет регион. Используется для определения сложных областей для рисования и клиппинга.
- **HICON, HCURSOR**: Представляют иконки и курсоры соответственно. Используются для работы с иконками и курсорами, их загрузки и отображения.
- **HANDLE**:Общий тип хендла, который используется для работы с различными объектами, такими как файлы, процессы, потоки, события, мьютексы и другие.

# Тип DWORD.

**DWORD** (Double Word) — это тип данных в Windows API, представляющий собой 32-битное беззнаковое целое число. Он используется для хранения числовых значений, флагов и других данных, которые помещаются в 32-битное пространство.

**Ключевые характеристики**

- **Размер**: 32 бита (4 байта).
- **Диапазон значений**: от 0 до 4,294,967,295 (0x00000000 до 0xFFFFFFFF).
- **Тип данных**: беззнаковое целое число (unsigned int).

**использования:**

DWORD широко используется в различных частях Windows API для представления размеров, флагов, идентификаторов, временных меток и других числовых данных.

# Тип BOOL.

**BOOL** — это тип данных, представляющий логическое значение (истина или ложь).

- **Размер**: 4 байта (32 бита).
- **Значения**: TRUE (истина, обычно 1) и FALSE (ложь, обычно 0).

```
BOOL success = TRUE;

if (success) {
   std::cout << "Operation was successful!" <<
std::endl;
} else {
   std::cout << "Operation failed!" << std::endl;
}
```

# Тип WCHAR.

**WCHAR** — это тип данных, представляющий символ Unicode. Используется для работы с текстом в кодировке Unicode.

- **Размер**: 2 байта (16 бит).
- **Диапазон значений**: от 0 до 65,535

```
WCHAR text[] = L"Hello, World!";
std::wcout << text << std::endl;
```

# Тип CHAR.

**CHAR** — это тип данных, представляющий один символ в кодировке ANSI.

- **Размер**: 1 байт (8 бит).
- **Диапазон значений**: от -128 до 127 (signed) или от 0 до 255 (unsigned).

```
CHAR letter = 'A';
std::cout << "Character: " << letter << std::endl;
```

# Тип LPCTSTR.

**LPCTSTR** (Long Pointer to a Constant String) — это указатель на строку в кодировке ANSI или Unicode, в зависимости от настроек компиляции.

- **ANSI**: const char*
- **Unicode**: const wchar_t*

```
LPCTSTR text =  T("Hello, World!");
std::wcout << text <<  std::endl;
```

# Использование типов.

- **BOOL**: Используется для представления логических значений (истина или ложь).
- **WCHAR**: Используется для представления символов Unicode.
- **CHAR**: Используется для представления символов в кодировке ANSI.
- **HANDLE**: Абстрактный указатель на объект системы.
- **DWORD**: Беззнаковое 32-битное целое число.
- **LPCTSTR**: Указатель на строку, который может быть ANSI или Unicode в зависимости от настроек компиляции.

# WINAPI. Функции #1.

**Работа с окнами и сообщениями:**

- **CreateWindowEx**: Создание окна с расширенными параметрами.
- **ShowWindow**: Отображение окна на экране.
- **UpdateWindow**: Обновление окна на экране.
- **GetMessage**: Получение сообщения из очереди сообщений.
- **TranslateMessage**: Преобразование сообщения клавиатуры в соответствующие символы.
- **DispatchMessage**: Отправка сообщения в оконную процедуру.
- **DefWindowProc**: Обработка сообщений по умолчанию в оконной процедуре.

# WINAPI. Функции #2.

**Работа с ресурсами и классами:**

- **LoadIcon**, **LoadCursor**: Загрузка иконок и курсоров из ресурсов.
- **LoadImage**: Загрузка изображения из файла или ресурса.
- **RegisterClass**, **UnregisterClass**: Регистрация и удаление классов окон.
- **LoadString**: Загрузка строки из ресурсов.

**Работа с графикой и изображениями:**

- **BitBlt**: Копирование изображения из одного контекста устройства в другой.
- **StretchBlt**: Масштабирование и копирование изображения.
- **CreateCompatibleDC**, **DeleteDC**: Создание и удаление контекста устройства.
- **CreateCompatibleBitmap**: Создание совместимого битмапа

# WINAPI. Функции #3.

**Функции работы с файлами и директориями:**

- **CreateFile**, **ReadFile**, **WriteFile**: Создание, чтение и запись файлов.
- **FindFirstFile**, **FindNextFile**, **FindClose**: Поиск файлов в директории.
- **CreateDirectory**, **RemoveDirectory**: Создание и удаление директорий.

**Работа с памятью и строками:**

- **HeapAlloc**, **HeapFree**: Выделение и освобождение памяти из кучи.
- **GlobalAlloc**, **GlobalFree**: Выделение и освобождение глобальной памяти.
- **strcpy**, **strcat**, **strcmp**: Операции с копированием, конкатенацией и сравнением строк.

**Функции для работы с потоками и процессами:**

- **CreateProcess**: Создание нового процесса.
- **CreateThread**, **ExitThread**: Создание и завершение потоков.
- **WaitForSingleObject**: Ожидание завершения одного объекта.

# WINAPI. Функции #4.

**Функции управления ресурсами и интерфейсом:**

- **EnableWindow**, **IsWindowEnabled**: Включение и проверка состояния окна.
- **SetWindowText**, **GetWindowText**: Установка и получение текста окна.
- **SendMessage**, **PostMessage**: Отправка сообщений в оконную процедуру.

**Функции работы с временем и системой:**

- **GetLocalTime**, **GetSystemTime**: Получение текущего времени.
- **GetTickCount**, **GetTickCount64**: Получение числа миллисекунд, прошедших с запуска системы.

# window procedure

**Window Procedure (или WndProc)** - это функция обратного вызова, которая обрабатывает сообщения, отправляемые операционной системой Windows приложению. Каждое окно, созданное в WinAPI, должно иметь свою собственную процедуру окна, которая обрабатывает сообщения, такие как клики мыши, клавишные события, изменения размеров окна и другие.

*Она принимает четыре параметра:*

- **HWND hwnd**: Дескриптор окна, для которого предназначено сообщение.
- **UINT uMsg**: Идентификатор сообщения, который определяет тип сообщения (например, WM_CLOSE, WM_COMMAND).
- **WPARAM wParam**: Дополнительная информация о сообщении, зависит от типа сообщения.
- **LPARAM lParam**: Дополнительная информация о сообщении, зависит от типа сообщения.

# WINAPI. UI, GUI.

**UI** (User Interface) - пользовательский интерфейс, часть программного обеспечения, через которую пользователь взаимодействует с приложением. UI включает в себя различные элементы управления, такие как кнопки, поля ввода, списки и другие компоненты.

**GUI** (Graphical User Interface) - графический пользовательский интерфейс, представляет собой тип UI, который использует графические элементы для представления информации и взаимодействия с пользователем, включая окна, меню, иконки и т.д.

**Оконная процедура (WndProc)**:

- Центральная функция обратного вызова, которая обрабатывает сообщения от операционной системы и элементов управления UI.
- Обработка сообщений, таких как нажатия кнопок, перемещения мыши, изменения размеров окна и других событий.

# Объекты/Элементы управления UI.

- **Кнопка (button)** - это элемент управления, который пользователи могут нажимать для выполнения определенного действия. (HWND hwndButton);
- **Текстовое поле (text edit)** - это элемент управления, который позволяет пользователям вводить и редактировать текст;
- **Метка (label text)** - это элемент управления, который отображает текст или изображения и не поддерживает редактирование;
- **Флажок (checkbox)** - это элемент управления, который позволяет пользователям выбирать или снимать выбор;
- **Радиокнопка (radio button)** - это элемент управления, который позволяет пользователям выбирать один вариант из группы;
- **Полоса прокрутки (scroll bar)** - это элемент управления, который позволяет пользователям прокручивать содержимое;
- **Список (list box)** - это элемент управления, который отображает список элементов, из которых пользователь может выбрать один или несколько;
- **Выпадающий список** - это элемент управления, который комбинирует текстовое поле и список, позволяя пользователю выбирать из списка или вводить текст;

# Объект - кнопка/текст;

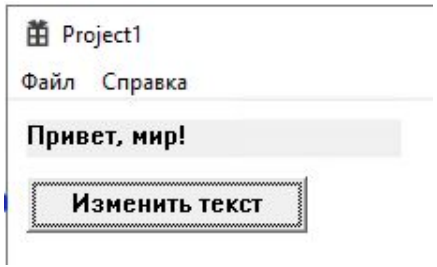
global секция:

**HWND hLabel, hButton;**

в InitInstance:

```
hLabel = CreateWindowW(L"Static", L"Текст какой-то...", WS_CHILD | WS_VISIBLE, 10, 10, 200, 20, hWnd,
nullptr, hInstance, nullptr);

hButton = CreateWindowW(L"Button", L"Изменить текст", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,10, 40, 150, 30,
hWnd, nullptr, hInstance, nullptr);
```



# Обработчик на кнопку.

обработчик добавляется в WndProc;

```
case BN_CLICKED: // Сообщение от кнопки при нажатии
   if ((HWND)lParam == hButton)
   {
       SetWindowText(hLabel,  L"Текст изменен!");
   }
break;
```

# связь текстовое поле и кнопки;

global секция:

```
#define ID_EDIT_NAME 101

#define ID_BUTTON_CHECK 102

HWND hEditName, hLabel, hButton;
```

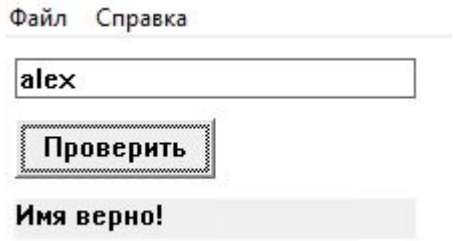
в InitInstance:

```
hEditName = CreateWindowW(L"Edit", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER,
    10, 10, 200, 20, hWnd, (HMENU)ID_EDIT_NAME, hInstance, NULL);

hButton = CreateWindowW(L"Button", L"Проверить", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
    10, 40, 100, 30, hWnd, (HMENU)ID_BUTTON_CHECK, hInstance, NULL);

hLabel = CreateWindowW(L"Static", L"", WS_CHILD | WS_VISIBLE,
    10, 80, 200, 20, hWnd, NULL, hInstance, NULL);

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



# Обработчик

обработчик добавляется в WndProc;

```
case ID_BUTTON_CHECK:
{
    wchar_t name[100];
    GetWindowText(hEditName, name,  100);
    if (wcscmp(name, L"alex") == 0)
    {
        SetWindowText(hLabel,  L"Имя верно!");
    }
    else
    {
        SetWindowText(hLabel,  L"Имя неверно!");
    }
}
```

# name correctly - new window;

в wWinMain: `MyRegisterNewClass(hInstance);`

Регистрация нового окна;

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

   wcex.cbSize =  sizeof(WNDCLASSEX);

   wcex.style = CS_HREDRAW | CS_VREDRAW;
   wcex.lpfnWndProc = NewWndProc;
   wcex.cbClsExtra =  0;
   wcex.cbWndExtra =  0;
   wcex.hInstance = hInstance;
   wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_PROJECT1));
   wcex.hCursor = LoadCursor( nullptr, IDC_ARROW);
   wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW +   1);
   wcex.lpszMenuName =  nullptr;
   wcex.lpszClassName = szNewWindowClass;
   wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

   return RegisterClassExW(&wcex);
}
```

# name correctly - new window;

в wWinMain: `MyRegisterNewClass(hInstance);`

Регистрация нового окна;

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

   wcex.cbSize =  sizeof(WNDCLASSEX);

   wcex.style = CS_HREDRAW | CS_VREDRAW;
   wcex.lpfnWndProc = NewWndProc;
   wcex.cbClsExtra =  0;
   wcex.cbWndExtra =  0;
   wcex.hInstance = hInstance;
   wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_PROJECT1));
   wcex.hCursor = LoadCursor( nullptr, IDC_ARROW);
   wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW +   1);
   wcex.lpszMenuName =  nullptr;
   wcex.lpszClassName = szNewWindowClass;
   wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

   return RegisterClassExW(&wcex);
}
```

# также..

в глобальной секции:

```
WCHAR szNewWindowClass[MAX_LOADSTRING] =  L"NewWindowClass";

LRESULT CALLBACK NewWndProc(HWND, UINT, WPARAM, LPARAM);
```

# LRESULT CALLBACK NewWndProc

```
LRESULT CALLBACK  NewWndProc (HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   switch (message) {
       case WM_COMMAND: {
           int wmId = LOWORD(wParam);
           switch (wmId){
               case IDM_ABOUT:
                  DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
               break;
               case IDM_EXIT:
                  DestroyWindow(hWnd);
               break;
               default:
                   return DefWindowProc(hWnd, message, wParam, lParam);
           }
       }
       break;
       case WM_PAINT:
       {
           PAINTSTRUCT ps;
           HDC hdc = BeginPaint(hWnd, &ps);
           TextOut(hdc,  10, 10, L"Это новое окно!" , 15); // Пример текста в новом окне
           EndPaint(hWnd, &ps);
       }
       break;
       case WM_DESTROY:
           PostQuitMessage( 0);
       break;
       default:
           return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}
```

# Объект - checkbox;

```
hCheckbox = CreateWindowW(L"Button", L"Опция", WS_CHILD | WS_VISIBLE | BS_CHECKBOX,

10, 40, 100, 20, hWnd, (HMENU)ID_CHECKBOX, hInstance, NULL);
```

# Общий пример кода;

https://disk.yandex.ru/d/SIhOpqPBD9CdyA

# Работа с ресурсами системы. Многопоточ. программирование;

WinAPI предоставляет механизмы для доступа и управления различными ресурсами операционной системы Windows, такими как файлы, изображения, звуки и другие.

**Основные функции**:

- Загрузка ресурсов из исполняемого файла или внешнего файла с помощью LoadResource, FindResource, LoadImage, LoadIcon, LoadCursor.
- Управление ресурсами, такими как иконки, курсоры, изображения, шрифты, строковые ресурсы и т.д.
- Освобождение ресурсов с помощью FreeResource, DeleteObject.

**Многопоточное программирование в WinAPI** позволяет одновременно выполнять несколько потоков внутри одного процесса для повышения производительности и отзывчивости приложений.

- Создание потока с помощью CreateThread или BeginThread.
- Синхронизация потоков с помощью объектов синхронизации: мьютексы (CreateMutex), семафоры (CreateSemaphore), события (CreateEvent).
- Защита доступа к общим ресурсам с помощью критических секций (InitializeCriticalSection, EnterCriticalSection, LeaveCriticalSection).
- Ожидание завершения потоков с помощью WaitForSingleObject или WaitForMultipleObjects.

# Работа с DLL.

DLL (динамические библиотеки связи) - это файлы, содержащие функции и ресурсы, которые могут быть использованы несколькими приложениями одновременно. Они загружаются в память при запуске приложения или по запросу.

**Основные функции**:

- Загрузка DLL с помощью функции LoadLibrary.
- Получение адреса функций из DLL с помощью GetProcAddress.
- Выгрузка DLL с помощью FreeLibrary.

# Работа с реестром.

Реестр - централизованное хранилище информации о настройках и параметрах операционной системы Windows и приложений.

**Основные функции**:

- Открытие ключа реестра с помощью RegOpenKeyEx.
- Чтение, запись и удаление значений с помощью RegQueryValueEx, RegSetValueEx, RegDeleteValue.
- Закрытие ключа с помощью RegCloseKey.

# обработка ошибок в winapi.

**Обработка ошибок**: WinAPI предоставляет механизмы для обработки ошибок, возвращаемых функциями.

- Функция GetLastError() возвращает код последней ошибки.
- Функция FormatMessage() используется для форматирования сообщения об ошибке на основе кода ошибки.

например:

```
if (!SomeFunction()) {
   DWORD dwError = GetLastError();
   // Обработка ошибки
   // Форматирование сообщения об ошибке
}
```

# Задание-1

Написать оконное приложение исп-я средства WINAPI.

- использовать кнопки, текст, галочка запомнить меня, и др объекты;
- регистрация/вход пользователя;
- после успешного входа пользователь попадает в dashboard;
- хранение данных локально (можно в виде файла .txt, .csv);
- код должен соответствовать заявленным требованиям;

# Задание-2

Написать оконное приложение исп-я средства WINAPI.

- Создать окно приложения с кнопкой "Открыть файл".
- Реализовать функционал открытия файла с помощью стандартного диалога открытия файлов (Open File Dialog).
- Прочитать содержимое выбранного файла.
- Отобразить содержимое файла в текстовом поле или многострочном текстовом поле в окне приложения.
- Добавить кнопку "Сохранить", чтобы пользователь мог сохранить измененные данные обратно в файл.

# Задание-3

Написать оконное приложение исп-я средства WINAPI.

- реализовать простой калькулятор;
- операции +, -, *, /, %.