**Детальный разбор по файлам и классам (с фокусом на задании):**

**1. CMatrix.cpp / CMatrix.h (Класс Матрицы)**

* **Назначение:** Предоставляет базовые операции над матрицами (создание, удаление, доступ к элементам, сложение, вычитание, умножение, транспонирование, изменение размера и т.д.). **Это основа для выполнения аффинных преобразований.**
* **Конструкторы:**
  + CMatrix(): Создает матрицу 1x1 по умолчанию.
  + CMatrix(int Nrow, int Ncol): Создает матрицу заданного размера Nrow x Ncol, заполненную нулями. **Важно для создания матриц преобразований 3x3.**
  + CMatrix(int Nrow): Создает вектор-столбец размера Nrow x 1. **Важно для представления координат точек (x, y, 1).**
  + CMatrix(const CMatrix &M): Конструктор копирования. **Критически важен** для корректной передачи матриц по значению и инициализации.
  + CMatrix(ifstream &file): Конструктор для загрузки матрицы из файла (в данном проекте, похоже, не используется).
* **Деструктор ~CMatrix():** Освобождает память, выделенную под двумерный массив array. **Важно для предотвращения утечек памяти.**
* **Операторы:**
  + operator()(int i, int j) и operator()(int i): Перегрузка оператора () для удобного доступа к элементам матрицы/вектора M(i, j) или V(i). Включает проверку выхода за границы.
  + operator=: Оператор присваивания. **Критически важен** для копирования содержимого одной матрицы в другую (M1 = M2). Также проверяет соответствие размерностей.
  + operator\*: **Очень важный оператор для задания!** Реализует умножение матриц. Используется для применения матриц преобразований к векторам координат (НовыеКоординаты = МатрицаПреобразования \* СтарыеКоординаты). Проверяет совместимость размеров матриц для умножения.
  + Другие операторы (+, -, унарный -, + со скаляром, - со скаляром): Стандартные математические операции.
* **Методы:**
  + rows(), cols(): Возвращают размеры матрицы.
  + Transp(): Транспонирование матрицы.
  + GetRow(), GetCol(): Получение строки или столбца матрицы.
  + RedimMatrix(), RedimData(): Изменение размера матрицы (с удалением или сохранением данных).
  + MaxElement(), MinElement(): Поиск максимального/минимального элемента.

**2. LibBlade.cpp / LibPlanets.h (Библиотека графики и класс CBlade)**

* **CRectD, CSizeD:**
  + **Назначение:** Аналоги стандартных CRect и CSize из MFC, но используют double вместо int. **Это необходимо для задания мировых координат**, которые могут быть нецелыми.
  + CRectD: Хранит координаты left, top, right, bottom прямоугольной области в мировых координатах.
  + CSizeD: Хранит ширину cx и высоту cy в double.
* **CreateTranslate2D(double dx, double dy) (КЛЮЧЕВАЯ ФУНКЦИЯ ИЗ ЗАДАНИЯ):**
  + **Назначение:** Формирует матрицу 3x3 для **переноса (смещения)** объекта.
  + **Реализация:** Создает матрицу TM размера 3x3.
  + [[1, 0, dx],
  + [0, 1, dy],

[0, 0, 1]]

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).

* + **Объяснение:** Эта матрица используется в **однородных координатах**. Чтобы перенести точку (x, y), ее представляют как вектор-столбец [x, y, 1]. Умножение матрицы TM на этот вектор дает:
  + [[1, 0, dx], [[x], [[1\*x + 0\*y + dx\*1], [[x + dx],
  + [0, 1, dy], \* [y], = [0\*x + 1\*y + dy\*1], = [y + dy],

[0, 0, 1]] [1]] [0\*x + 0\*y + 1\*1]] [ 1 ]]

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).

Результат - новый вектор [x+dx, y+dy, 1], соответствующий точке, смещенной на dx по оси X и dy по оси Y.

* **CreateRotate2D(double fi) (КЛЮЧЕВАЯ ФУНКЦИЯ ИЗ ЗАДАНИЯ):**
  + **Назначение:** Формирует матрицу 3x3 для **поворота** объекта вокруг начала координат (0, 0) на угол fi (в градусах). Положительный fi - против часовой стрелки.
  + **Реализация:**
    1. fg = fmod(fi, 360.0): Приводит угол к диапазону [0, 360) или (-360, 0).
    2. ff = (fg / 180.0) \* pi: **Переводит угол из градусов в радианы**, так как тригонометрические функции cos и sin в C++ работают с радианами.
    3. Создает матрицу RM размера 3x3:
    4. [[cos(ff), -sin(ff), 0],
    5. [sin(ff), cos(ff), 0],

[ 0, 0, 1]]

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).

* + **Объяснение:** Эта матрица также используется в однородных координатах. Умножение RM на вектор [x, y, 1] дает вектор [x', y', 1], где (x', y') - это координаты точки (x, y), повернутой на угол ff (в радианах) вокруг начала координат. Формулы поворота:
    1. x' = x \* cos(ff) - y \* sin(ff)
    2. y' = x \* sin(ff) + y \* cos(ff)  
       Эти формулы точно соответствуют результату умножения матрицы RM на вектор [x, y, 1].
* **SpaceToWindow(CRectD& RS, CRect& RW):**
  + **Назначение:** Вычисляет **матрицу преобразования координат** из мировой системы (RS, double) в оконную (RW, int, пиксели). Эта матрица объединяет масштабирование и перенос.
  + **Реализация:**
    1. Получает размеры областей в мировых (dsx, dsy) и оконных (dwx, dwy) координатах.
    2. Вычисляет коэффициенты масштабирования kx = dwx / dsx и ky = dwy / dsy.
    3. Формирует матрицу M, которая:
       - Масштабирует по X на kx.
       - Масштабирует по Y на ky и **инвертирует** ось Y (так как в мировых координатах Y растет вверх, а в оконных - вниз), поэтому M(1, 1) = -ky.
       - Смещает начало координат так, чтобы левый нижний угол RS (RS.left, RS.bottom) отобразился в левый нижний угол RW (RW.left, RW.bottom). Это достигается членами M(0, 2) и M(1, 2).
  + **Важность:** Эта матрица позволяет вычислить пиксельные координаты для любой точки, заданной в мировых координатах, но в коде она используется не напрямую для точек, а для настройки режима MM\_ANISOTROPIC через SetMyMode. В функции CBlade::Draw она используется для преобразования координат вершин треугольников перед вызовом dc.Polygon, что немного **нестандартно** при использовании SetMyMode. Обычно при SetMyMode рисуют прямо в мировых координатах.
* **SetMyMode(CDC& dc, CRectD& RS, CRect& RW) (КЛЮЧЕВАЯ ФУНКЦИЯ ДЛЯ ОТОБРАЖЕНИЯ):**
  + **Назначение:** Настраивает контекст устройства (dc) так, чтобы можно было рисовать, используя мировые координаты, а система сама преобразовывала их в пиксельные координаты окна.
  + **Реализация:**
    1. dc.SetMapMode(MM\_ANISOTROPIC): Устанавливает **анизотропный режим**. Это самый гибкий режим, позволяющий задать произвольное соответствие между логическими (мировыми) и физическими (оконными) координатами, включая разный масштаб по осям и инверсию оси Y.
    2. dc.SetWindowExt((int)dsx, (int)dsy): Задает "размер" логического окна в мировых единицах (ширина dsx, высота dsy).
    3. dc.SetViewportExt(dwx, -dwy): Задает "размер" окна устройства в пикселях (ширина dwx, высота dwy). **Знак минус у dwy инвертирует ось Y.**
    4. dc.SetWindowOrg((int)xsL, (int)ysL): Задает **логические** координаты (мировые) точки (xsL, ysL - обычно RS.left, RS.bottom), которая будет соответствовать началу координат устройства.
    5. dc.SetViewportOrg(xwL, ywH): Задает **физические** координаты (пиксели) точки (xwL, ywH - обычно RW.left, RW.bottom), которая будет соответствовать началу координат логического окна.
  + **Результат:** После вызова этой функции можно использовать функции рисования GDI (например, dc.MoveTo, dc.LineTo, dc.Ellipse, dc.Polygon), передавая им **мировые координаты**, а они будут корректно отображаться в указанной области окна RW.
* **CBlade (Класс Лопасти - КЛЮЧЕВОЙ КЛАСС ИЗ ЗАДАНИЯ):**
  + **Назначение:** Инкапсулирует все данные и логику, связанные с фигурой лопасти: ее геометрию, положение, вращение и отрисовку.
  + **Поля-члены:**
    1. MainPoint, FirstTop, ... RightBlade (CRect): Прямоугольники, вероятно, для определения *начальных* размеров или областей компонентов лопасти в **мировых координатах**. Используются для рисования центра (MainPoint) и, возможно, должны были использоваться для лопастей, но логика рисования опирается на координаты.
    2. FTCoords, STCoords, ... RBCoords (CMatrix): **Векторы (3x1)**, хранящие **текущие мировые координаты** вершин лопастей в формате [x, y, 1]. **Именно они изменяются при вращении.**
    3. RW (CRect): Область в окне (пиксели), где будет рисоваться лопасть. Задается в конструкторе, используется в SetMyMode и SpaceToWindow.
    4. RS (CRectD): Область в **мировой системе координат**, которая будет отображаться в RW. Задается в конструкторе (-RoE, RoE, RoE, -RoE), используется в SetMyMode и SpaceToWindow. **Определение размеров области D в МСК (задание).**
    5. wPoint (double): Угловая скорость вращения (градусы в единицу времени dt). Задана как -10, что означает **вращение по часовой стрелке** (как в задании).
    6. fiSB, fiFB, ... fiBlue4 (double): **Текущие углы** (в градусах) для различных вершин/частей лопасти. Используются для расчета координат и **обновляются при вращении**.
    7. dt (double): Интервал времени (шаг дискретизации) для расчета нового положения. Устанавливается в OnLabpBlade и используется в SetNewCoords.
    8. RedBrush, BlueBrush, SunBrush (CBrush): Кисти для заливки частей лопасти разными цветами.
  + **Конструктор CBlade():**
    1. **Назначение:** Инициализация всех параметров лопасти **по умолчанию**. **Выполняет требование задания задать параметры в конструкторе.**
    2. Задает размеры мировых (RS) и оконных (RW) координат.
    3. Задает размеры центрального круга (MainPoint).
    4. Инициализирует кисти (RedBrush, BlueBrush, SunBrush).
    5. Задает размеры CRect для разных частей лопасти (возможно, не используются в финальной отрисовке).
    6. Задает **начальные углы** (fiFT, fiST, ... fiBlue4).
    7. Задает угловую скорость (wPoint = -10).
    8. Задает начальный dt = 1 (позже изменяется).
    9. Инициализирует матрицы координат (FTCoords, и т.д.) как векторы 3x1. **Важно:** Начальные *значения* координат (x, y) вычисляются только при первом вызове SetNewCoords.
  + **SetNewCoords() (ЛОГИКА ВРАЩЕНИЯ - ОЧЕНЬ ВАЖНО):**
    1. **Назначение:** Вычисляет новые координаты всех вершин лопасти для следующего кадра анимации.
    2. **Реализация (для одной точки FTCoords):**
       - RoV = (EarthOrbit.right - EarthOrbit.left) / 2: Получает радиус "орбиты" (расстояние от центра до вершин).
       - ff = (fiFT / 90.0) \* pi: **Ошибка здесь.** Должно быть (fiFT / 180.0) \* pi для перевода из градусов в радианы. 90.0 - неверно.
       - x = RoV \* cos(ff), y = RoV \* sin(ff): Вычисляет декартовы координаты (x, y) на окружности радиуса RoV для **текущего** угла fiFT.
       - FTCoords(0) = x; FTCoords(1) = y; FTCoords(2) = 1;: Записывает эти координаты в вектор.
       - fiFT += wPoint \* dt;: **Обновляет угол** для следующего шага (прибавляет приращение, рассчитанное из угловой скорости и шага времени). wPoint отрицательный, так что угол уменьшается (вращение по часовой).
       - CMatrix P = CreateRotate2D(fiFT);: Создает матрицу поворота на **новый, уже обновленный** угол fiFT.
       - FTCoords = P \* FTCoords;: **Применяет поворот к только что вычисленным координатам.**
    3. **Проблема/Особенность:** Эта логика кажется **избыточной или некорректной** для стандартной анимации вращения. Она каждый раз пересчитывает положение точки на окружности по *старому* углу, а затем *поворачивает* эту точку на *новый* угол. Более стандартный подход:
       - **Вариант 1 (Инкрементный поворот):** Хранить текущие координаты FTCoords. На каждом шаге создавать матрицу поворота на *малый угол* delta\_fi = wPoint \* dt. Умножать *текущие* координаты на эту матрицу: FTCoords = CreateRotate2D(delta\_fi) \* FTCoords.
       - **Вариант 2 (Абсолютный угол):** Хранить *начальные* координаты точки (например, [RoV, 0, 1]). На каждом шаге обновлять *общий* угол fiTotal += wPoint \* dt. Создавать матрицу поворота на fiTotal. Применять эту матрицу к *начальным* координатам: FTCoords = CreateRotate2D(fiTotal) \* InitialCoords.  
         Текущая реализация делает что-то среднее и, вероятно, приводит к не совсем точному вращению или накоплению ошибки, хотя визуально может выглядеть приемлемо для простых случаев. Плюс ошибка в конвертации градусов в радианы.
    4. Эта логика повторяется для всех 4 красных и 4 синих точек (вершин лопастей).
  + **Draw(CDC& dc) (ОТРИСОВКА - ОЧЕНЬ ВАЖНО):**
    1. **Назначение:** Рисует текущее состояние лопасти в контекст устройства dc. **Предполагается, что SetMyMode уже вызван до Draw (это делается в OnPaint).**
    2. **Реализация:**
       - CMatrix transformMatrix = SpaceToWindow(RS, RW);: **Получает матрицу преобразования.** Как упоминалось, ее использование здесь при активном SetMyMode немного странно.
       - CPen blackPen(...): Создает черное перо для обводки.
       - DrawTriangleWithBorder(...): Вызывает вспомогательную функцию для рисования **треугольников**, формирующих лопасти. Передает координаты вершин (FTCoords, STCoords, и т.д.), цвет заливки и transformMatrix.
       - dc.SelectObject(&SunBrush); dc.Ellipse(MainPoint);: Рисует центральный круг (MainPoint) **используя мировые координаты**, так как SetMyMode активен. Заливает его кистью SunBrush.
       - Восстанавливает старое перо.
  + **DrawTriangleWithBorder(...):**
    1. **Назначение:** Рисует треугольник, соединяющий начало координат (0,0) и две переданные точки (FCoords, SCoords), с заливкой и черной обводкой.
    2. **Реализация:**
       - CMatrix p1 = transformMatrix \* FCoords; и т.д.: **Преобразует мировые координаты вершин в оконные** с помощью матрицы.
       - CBrush brush(fillColor); CPen blackPen(...): Создает кисть и перо.
       - POINT points[3] = { {0, 0}, ... };: **Ошибка здесь.** Координаты {0, 0} - это начало координат в **мировой** системе. Но координаты FCoords(0), FCoords(1) и SCoords(0), SCoords(1) также берутся из **мировых** координат (FCoords, SCoords), хотя SetMyMode уже настроен для работы с мировыми координатами. При активном SetMyMode нужно было бы передавать мировые координаты напрямую в dc.Polygon. Если же SetMyMode не используется (или используется MM\_TEXT), то нужно было бы использовать *преобразованные* координаты (p1(0), p1(1), p2(0), p2(1), pc(0), pc(1)) в структуре POINT. Текущая реализация смешивает подходы и, скорее всего, рисует некорректно, хотя может случайно выглядеть правильно из-за того, что transformMatrix пытается сделать то же самое, что и SetMyMode. **Правильно было бы:**
       - void CBlade::DrawTriangleWithBorder(CMatrix FCoords, CMatrix SCoords, CDC& dc, COLORREF fillColor)
       - {
       - // Предполагаем, что SetMyMode УЖЕ вызван до вызова Draw, который вызывает эту функцию.
       - // Не используем transformMatrix здесь.
       - CBrush brush(fillColor);
       - CPen blackPen(PS\_SOLID, 1, RGB(0, 0, 0));
       - CPen\* pOldPen = dc.SelectObject(&blackPen);
       - CBrush\* pOldBrush = dc.SelectObject(&brush);
       - // Используем МИРОВЫЕ координаты напрямую
       - POINT points[3] = {
       - {0, 0}, // Начало мировой системы координат
       - {static\_cast<int>(FCoords(0)), static\_cast<int>(FCoords(1))},
       - {static\_cast<int>(SCoords(0)), static\_cast<int>(SCoords(1))}
       - };
       - // GDI сама преобразует их в оконные благодаря SetMyMode
       - dc.Polygon(points, 3);
       - dc.SelectObject(pOldBrush);
       - dc.SelectObject(pOldPen);

}

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).C++

* + DrawBladePair, DrawBlade: Похоже, альтернативные или устаревшие функции рисования, которые не вызываются из основного Draw.
  + GetRS(CRectD& RSX), GetRW(): Методы для получения копий областей RS и RW.

**3. ChildView.cpp / ChildView.h (Окно для рисования и взаимодействия)**

* **Назначение:** Отображает лопасть и обрабатывает команды пользователя (меню, клики мыши) для управления анимацией.
* **Поля-члены:**
  + IsData (int): Флаг, показывающий, была ли выбрана лопасть из меню (готовы ли данные для рисования).
  + dT\_Timer (int): Интервал таймера в миллисекундах (как часто обновлять кадр).
  + RW, RS: Копии областей из объекта Blade. RS используется в OnPaint для SetMyMode.
  + Blade (CBlade): **Экземпляр класса лопасти.** Вся логика лопасти находится внутри этого объекта.
* **BEGIN\_MESSAGE\_MAP / END\_MESSAGE\_MAP:** Связывает сообщения Windows с функциями-обработчиками.
  + ON\_WM\_PAINT(): Сообщение WM\_PAINT (нужно перерисовать окно) -> OnPaint().
  + ON\_COMMAND(ID\_LAB\_PLANETS, ...): Выбор пункта меню с ID ID\_LAB\_PLANETS -> OnLabpBlade(). **Важно для задания: выбор пункта меню.**
  + ON\_WM\_TIMER(): Сообщение WM\_TIMER (сработал таймер) -> OnTimer(). **Важно для задания: использование таймера.**
  + ON\_WM\_LBUTTONDBLCLK(): Двойной клик левой кнопкой мыши -> OnLButtonDblClk(). **Важно для задания: старт вращения.**
  + ON\_WM\_RBUTTONDBLCLK(): Двойной клик правой кнопкой мыши -> OnRButtonDblClk(). **Важно для задания: стоп вращения.**
* **PreCreateWindow(CREATESTRUCT& cs):**
  + Вызывается перед созданием окна.
  + cs.lpszClass = AfxRegisterWndClass(...): Регистрирует класс окна. Важно, что стиль CS\_DBLCLKS включен, **чтобы окно получало сообщения о двойных кликах (WM\_LBUTTONDBLCLK, WM\_RBUTTONDBLCLK)**.
* **OnPaint() (ОБРАБОТЧИК ПЕРЕРИСОВКИ - КЛЮЧЕВОЙ):**
  + **Назначение:** Вызывается системой, когда нужно перерисовать содержимое окна (после Invalidate(), при изменении размера, перекрытии и т.д.).
  + **Реализация:**
    1. CPaintDC dc(this): Создает специальный контекст устройства для рисования в ответ на WM\_PAINT.
    2. if (IsData == 1): Рисует только если лопасть была инициализирована через меню.
    3. Blade.GetRS(RS); RW = Blade.GetRW();: Получает актуальные области из объекта лопасти.
    4. SetMyMode(dc, RS, RW);: **Настраивает систему координат (мировая -> оконная). Это КЛЮЧЕВОЙ вызов перед рисованием.**
    5. Blade.Draw(dc);: **Вызывает метод рисования объекта лопасти**, передавая ему настроенный dc. Лопасть рисуется с использованием мировых координат (с учетом упомянутых выше проблем в DrawTriangleWithBorder).
    6. dc.SetMapMode(MM\_TEXT);: Возвращает режим отображения по умолчанию (пиксельный) после завершения рисования в SetMyMode. Хорошая практика.
* **OnLabpBlade() (ОБРАБОТЧИК МЕНЮ):**
  + **Назначение:** Вызывается при выборе пункта меню "ЛР\_Лопасть ► Лопасть". Инициализирует отображение лопасти.
  + **Реализация:**
    1. Blade.SetDT(0); Blade.SetNewCoords();: Устанавливает dt=0, вызывает SetNewCoords. Это вычислит *начальное* положение лопасти без смещения угла (wPoint \* dt == 0).
    2. Blade.SetDT(10);: Устанавливает реальный шаг времени dt = 10 для последующей анимации. **Внимание:** Единицы dt не ясны из кода, но wPoint в градусах/шаг, так что dt - это просто множитель шага. Если wPoint - градусы/сек, то dt должен быть в секундах.
    3. dT\_Timer = 50;: Задает интервал таймера в **миллисекундах**. Т.е. кадр будет обновляться каждые 50 мс (~20 FPS).
    4. IsData = 1;: Устанавливает флаг, что можно рисовать.
    5. Invalidate();: **Делает окно недействительным, что приводит к вызову OnPaint()** и отрисовке статического изображения лопасти.
* **OnTimer(UINT\_PTR nIDEvent) (ОБРАБОТЧИК ТАЙМЕРА - АНИМАЦИЯ):**
  + **Назначение:** Вызывается системой каждые dT\_Timer миллисекунд после запуска таймера. Двигает лопасть.
  + **Реализация:**
    1. Blade.SetNewCoords();: **Вычисляет новые координаты лопасти на основе прошедшего шага времени (dt) и угловой скорости (wPoint).**
    2. Invalidate();: **Запрашивает перерисовку окна**, чтобы отобразить лопасть в новом положении. Снова будет вызван OnPaint.
    3. CWnd::OnTimer(nIDEvent);: Вызывает обработчик базового класса.
* **OnLButtonDblClk(UINT nFlags, CPoint point) (СТАРТ ВРАЩЕНИЯ):**
  + **Назначение:** Вызывается при двойном клике левой кнопкой мыши.
  + **Реализация:**
    1. SetTimer(1, dT\_Timer, NULL);: **Запускает таймер.**
       - 1: Идентификатор таймера (может быть любым, главное использовать тот же ID для KillTimer).
       - dT\_Timer: Интервал в миллисекундах (задан в OnLabpBlade).
       - NULL: Функция обратного вызова (не используется, так как ловится сообщение WM\_TIMER).
    2. Теперь система будет посылать сообщение WM\_TIMER окну каждые dT\_Timer миллисекунд, вызывая OnTimer.
* **OnRButtonDblClk(UINT nFlags, CPoint point) (СТОП ВРАЩЕНИЯ):**
  + **Назначение:** Вызывается при двойном клике правой кнопкой мыши.
  + **Реализация:**
    1. KillTimer(1);: **Останавливает таймер** с идентификатором 1. Сообщения WM\_TIMER больше не будут приходить, и OnTimer вызываться не будет, прекращая анимацию.

**4. MainFrm.cpp / MainFrm.h (Главное окно)**

* **Назначение:** Создает основную структуру окна приложения и дочернее окно CChildView. **Важно для задания: фиксация размера окна.**
* **OnCreate(LPCREATESTRUCT lpCreateStruct):**
  + Вызывается при создании окна.
  + int width = 623; int height = 700;: Задаются желаемые размеры окна.
  + MoveWindow(...): **Устанавливает размер и положение окна.** Центрирует окно на экране.
  + m\_wndView.Create(...): Создает экземпляр CChildView внутри CMainFrame.
  + m\_wndStatusBar.Create(...): Создает строку состояния.
* **PreCreateWindow(CREATESTRUCT& cs) (ФИКСАЦИЯ РАЗМЕРА - КЛЮЧЕВОЕ):**
  + **Назначение:** Позволяет изменить стили окна *до* его фактического создания.
  + **Реализация:**
    1. cs.style = WS\_OVERLAPPED | WS\_CAPTION | FWS\_ADDTOTITLE | WS\_MINIMIZEBOX | WS\_SYSMENU;: **Устанавливает стили окна.**
       - WS\_OVERLAPPED: Обычное окно верхнего уровня.
       - WS\_CAPTION: Окно с заголовком.
       - FWS\_ADDTOTITLE: Добавляет имя документа к заголовку (стандартно для MFC).
       - WS\_MINIMIZEBOX: Кнопка "Свернуть".
       - WS\_SYSMENU: Системное меню (иконка слева в заголовке).
       - **Важно:** В этом наборе **отсутствуют** стили WS\_THICKFRAME (рамка для изменения размера) и WS\_MAXIMIZEBOX (кнопка "Развернуть"). **Именно отсутствие WS\_THICKFRAME делает окно нереcайзабельным мышью**, выполняя требование задания.
    2. cs.dwExStyle &= ~WS\_EX\_CLIENTEDGE;: Убирает утопленную границу вокруг клиентской области.
    3. cs.lpszClass = AfxRegisterWndClass(0);: Регистрирует класс окна для фрейма.