Практическая работа №10

**Тема: Использование таймера**

**Цель работы:** закрепление знаний полученных при изучении функций WinAPI для работы с таймерами.

**Ход работы**

1. #include <windows.h>
2. #include <cmath>
3. #define IDC\_EDIT\_INTERVAL 101
4. #define IDC\_BUTTON\_START\_STOP 102
5. #define IDC\_BUTTON\_DIRECTION 103
6. const int WINDOW\_WIDTH = 400;
7. const int WINDOW\_HEIGHT = 400;
8. const float PI = 3.14159265358979323846f;
9. // Прямое объявление функции DrawVectorCursor
10. void DrawVectorCursor(HDC hdc, float rotationAngle);
11. // Глобальные переменные
12. float rotationAngle = 0.0f;
13. bool isClockwise = true;
14. UINT\_PTR timerId = 0;
15. HWND hwndEditInterval;
16. // Обработчик сообщений для главного окна
17. LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
18. int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR pCmdLine, int nCmdShow) {
19. const wchar\_t CLASS\_NAME[] = L"RotatingCursorWindow";
20. WNDCLASSW wc = {};
21. wc.lpfnWndProc = WindowProc;
22. wc.hInstance = hInstance;
23. wc.lpszClassName = CLASS\_NAME;
24. RegisterClassW(&wc);
25. HWND hwnd = CreateWindowExW(
26. 0,
27. CLASS\_NAME,
28. L"Rotating Vector Cursor",
29. WS\_OVERLAPPEDWINDOW | WS\_VISIBLE,
30. CW\_USEDEFAULT, CW\_USEDEFAULT, WINDOW\_WIDTH, WINDOW\_HEIGHT,
31. NULL,
32. NULL,
33. hInstance,
34. NULL
35. );
36. if (hwnd == NULL) {
37. return 0;
38. }
39. ShowWindow(hwnd, nCmdShow);
40. UpdateWindow(hwnd);
41. MSG msg;
42. while (GetMessage(&msg, NULL, 0, 0)) {
43. TranslateMessage(&msg);
44. DispatchMessage(&msg);
45. }
46. return (int)msg.wParam;
47. }
48. [22:14]
49. LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
50. switch (uMsg) {
51. case WM\_CREATE:
52. // Создание элемента управления для ввода интервала таймера
53. hwndEditInterval = CreateWindowExW(0, L"EDIT", L"10",
54. WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_NUMBER,
55. 10, 10, 100, 25, hwnd, (HMENU)IDC\_EDIT\_INTERVAL,
56. (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);
57. // Создание кнопки для запуска и остановки вращения
58. CreateWindowExW(0, L"BUTTON", L"Start/Stop",
59. WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,
60. 120, 10, 100, 25, hwnd, (HMENU)IDC\_BUTTON\_START\_STOP,
61. (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);
62. // Создание кнопки для изменения направления вращения
63. CreateWindowExW(0, L"BUTTON", L"Change Direction",
64. WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,
65. 230, 10, 150, 25, hwnd, (HMENU)IDC\_BUTTON\_DIRECTION,
66. (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);
67. return 0;
68. case WM\_DESTROY:
69. // Остановка таймера перед закрытием окна
70. KillTimer(hwnd, timerId);
71. PostQuitMessage(0);
72. return 0;
73. case WM\_TIMER:
74. // Обновление угла поворота в зависимости от направления
75. if (isClockwise) {
76. rotationAngle += 0.01f;
77. }
78. else {
79. rotationAngle -= 0.01f;
80. }
81. // Обновление окна для перерисовки
82. InvalidateRect(hwnd, NULL, TRUE);

return 0;

1. [22:14]
2. case WM\_PAINT:
3. {
4. PAINTSTRUCT ps;
5. HDC hdc = BeginPaint(hwnd, &ps);
6. // Очистка фона белым кистью
7. HBRUSH hBackgroundBrush = CreateSolidBrush(RGB(255, 255, 255));
8. HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBackgroundBrush);
9. FillRect(hdc, &ps.rcPaint, hBackgroundBrush);
10. SelectObject(hdc, hOldBrush);
11. DeleteObject(hBackgroundBrush);
12. // Отрисовка векторного курсора
13. DrawVectorCursor(hdc, rotationAngle);
14. EndPaint(hwnd, &ps);
15. return 0;
16. }
17. case WM\_COMMAND:
18. // Обработка событий нажатия кнопок
19. if (LOWORD(wParam) == IDC\_BUTTON\_START\_STOP) {
20. // Запуск или остановка таймера в зависимости от текущего состояния
21. if (timerId == 0) {
22. int interval = GetDlgItemInt(hwnd, IDC\_EDIT\_INTERVAL, NULL, FALSE);
23. if (interval > 0) {
24. timerId = SetTimer(hwnd, 1, interval, NULL);
25. }
26. }
27. else {
28. KillTimer(hwnd, timerId);
29. timerId = 0;
30. }
31. }
32. else if (LOWORD(wParam) == IDC\_BUTTON\_DIRECTION) {
33. // Изменение направления вращения
34. isClockwise = !isClockwise;
35. // Обновление окна для перерисовки
36. InvalidateRect(hwnd, NULL, TRUE);
37. }
38. return 0;
39. }
40. return DefWindowProcW(hwnd, uMsg, wParam, lParam);

}

1. [22:15]
2. // Функция для отрисовки векторного курсора
3. void DrawVectorCursor(HDC hdc, float rotationAngle) {
4. // Определение центра окна
5. int centerX = WINDOW\_WIDTH / 2;
6. int centerY = WINDOW\_HEIGHT / 2;
7. // Определение радиусов внешнего и внутреннего кругов
8. int outerRadius = min(centerX, centerY) - 20;
9. int innerRadius = outerRadius / 2;
10. // Создание карандаша и кисти для отрисовки курсора
11. HPEN hPen = CreatePen(PS\_SOLID, 1, RGB(0, 0, 0)); // Черный карандаш
12. HBRUSH hBrush = CreateSolidBrush(RGB(0, 0, 0)); // Черный кисть
13. // Выбор старых объектов и сохранение старых объектов
14. HPEN oldPen = (HPEN)SelectObject(hdc, hPen);
15. HBRUSH oldBrush = (HBRUSH)SelectObject(hdc, hBrush);
16. // Вычисление вершин треугольника
17. POINT vertices[3];
18. for (int i = 0; i < 3; ++i) {
19. float angle = (rotationAngle + (2 \* PI \* i) / 3) - (PI / 2); // Поворот на 90 градусов, чтобы треугольник начинался сверху
20. vertices[i].x = centerX + (int)(outerRadius \* cos(angle)); // Вычисление координаты x вершины
21. vertices[i].y = centerY - (int)(outerRadius \* sin(angle)); // Вычисление координаты y вершины, учитывая инверсию оси Y в координатах окна
22. }
23. // Отрисовка треугольника

Polygon(hdc, vertices, 3);

1. [22:15]
2. // Вычисление координат внутренней точки
3. int innerX = centerX + (int)(innerRadius \* cos(rotationAngle)); // Вычисление координаты x внутренней точки
4. int innerY = centerY - (int)(innerRadius \* sin(rotationAngle)); // Вычисление координаты y внутренней точки, учитывая инверсию оси Y в координатах окна
5. // Отрисовка внутренней точки
6. Ellipse(hdc, innerX - 2, innerY - 2, innerX + 2, innerY + 2); // Отрисовка маленького эллипса в качестве внутренней точки
7. // Возврат старых объектов и удаление созданных объектов
8. SelectObject(hdc, oldPen);
9. SelectObject(hdc, oldBrush);
10. DeleteObject(hPen);
11. DeleteObject(hBrush);

}