**Лекція 25**

## Прості фігури. Малювання із застосуванням класів.

Коли ми малюємо лініями, то вони можуть мати різні параметри: товщину, стиль і т.д., тому спочатку необхідно визначити «перо» яким буде виконуватись малювання.

Поточне перо є частиною структури контексту вікна. щоб лінію намалювати необхідним пером, спочатку ви повинні створити об'єкт типу HPEN

**HPEN** Pen = ***CreatePen*** (PS_SOLID, 3, RGB (255,0,0));

Перший аргумент функції ***CreatePen*** визначає чи буде лінія суцільний (PS_SOLID), пунктирною, штрих – пунктирною і т.д. Можна використовувати значення PS_DOT, PS_DASH, PS_DASHDOT і деякі інші. Другий аргумент визначає товщину лінії, третій – задає колір. аргументи макросу RGB є цілими числами зі значеннями від 0 до 255 і задають колір, складений з відтінків / яркостей трьох базових кольорів: червоного (Red), зеленого (Green) і синього (Blue). Нульове значення виключає відповідний базовий колір, максимальне допустиме значення яскравості базових кольорів 255. За умовчанням встановлено чорне суцільне перо товщиною в 1 піксель.

Після створення пера (HPEN) його потрібно завантажити в контекст (HDC) функцією ***SelectObject*** (HDC, HPEN). Після цього всі лінії будуть малюватися створеним пером до тих пір, поки в контексті не буде завантажено нове перо

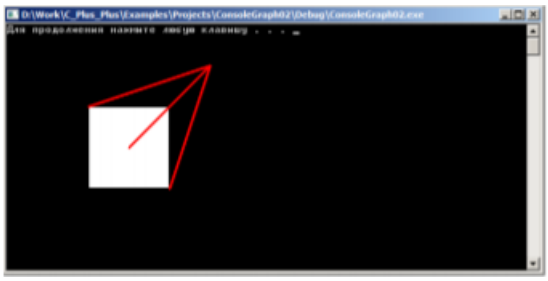
Є ще один момент. В контексті зберігається положення так званого поточного покажчика, який містить координати точки від якої наступна графічна функція починає малювати свою фігуру. При запуску програми цей покажчик встановлюється в точку (0,0). його місце розташування на екрані невидимо, він означає лише позицію у вікні для деяких функцій.

Не всі графічні функції використовують цей покажчик. Функція ***MoveToEx*** ((HDC, X, Y, NULL); переносить поточну позицію покажчика в точку (X, Y) і не запам'ятовує стару позицію (якщо останній аргумент дорівнює NULL). У наступному прикладі ми використовуємо функцію ***LineTo***(HDC, X, Y) яка малює

відрізок прямої від поточного положення покажчика до точки (X, Y) і переміщає покажчик в цю точку.

**Приклад 2.**

```
#include <windows.h>
void main ()
{// отримуємо дескриптор контексту консольного вікна
HDC hdc = GetDC (GetConsoleWindow ());
Rectangle (hdc, 100,100,200,200); // Малюємо квадрат
HPEN Pen = CreatePen (PS_SOLID, 3, RGB (255,0,0)); // створюємо перо
SelectObject (hdc, Pen); // завантажуємо створене перо в контекст
// переміщуємо покажчик в центр квадрата
MoveToEx (hdc, 150,150, NULL);
// малюємо лінію від поточного положення покажчика
LineTo (hdc, 250, 50);
// малюємо лінію з кінця попереднього відрізка
LineTo (hdc, 200, 200);
// переміщаємо покажчик в ліву верхню вершину квадрата
MoveToEx (hdc, 100, 100, NULL);
// малюємо лінію від поточного положення курсору до точки (250,50)
LineTo (hdc, 250, 50);
system ( "pause");
DeleteObject (Pen);
ReleaseDC (NULL, hdc);}
```



Графічна область вікна являє собою масив пікселів. Кожен піксель відповідає одній точці на екрані і може мати свій колір. Встановити колір пікселя в точці екрану з координатами (X, Y) можна з допомогою функції

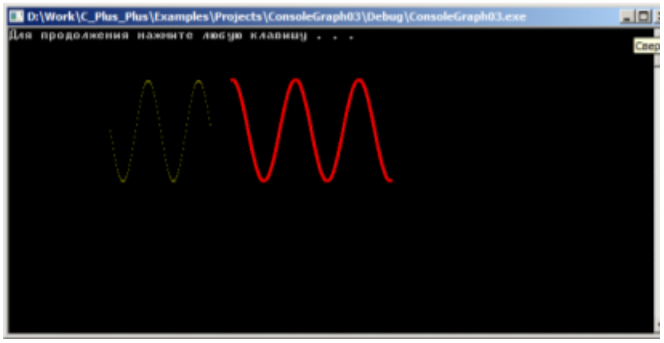
**SetPixel** (HDC dc, int X, int Y, COLORREF Color);

Тут ***COLORREF*** представляє спеціальний тип (структуру), яку створює макрос RGB. Функція **SetPixel** відображає точку заданого кольору за вказаними координатами і повертає колір, який був встановлений в дійсності (іноді дисплей не в змозі точно відтворити заданий колір).

**Приклад 3.** Малювання синусоїди.

У цьому прикладі демонструється можливість малювання графіка функції в командному вікні двома способами – поточково і суцільною лінією.

```
#include <windows.h>
#include <math.h>
const double Pi = 3.141592;
void main ()
{HDC hdc = GetDC (GetConsoleWindow ());
double x, y;
COLORREF penColor = RGB (255, 255, 0);
// Малюємо графік синусоїди по точкам
for (x = 0; x <100; x ++) {
y = sin (Pi * x / 25);
SetPixel (hdc, 100 + int (x), 100 + int (50 * y), penColor);}
// створюємо перо для малювання
HPEN Pen = CreatePen (PS_SOLID, 3, RGB (255,0,0));
SelectObject (hdc, Pen);
// малюємо синусоїду по відрізках
int xScale = 10, yScale = 50; // масштаб по осях X і Y
int xO = 300, yO = 100; // положення початку координат
// діапазон зміни абсциси
float xBeg = -8.0f, xEnd = 8.0f;
// покажчик в початкову точку кривої
MoveToEx (hdc, xO + xScale * xBeg, yO + yScale * sin (xBeg), NULL);
// малюємо відрізки від точки до точки
for (x = xBeg; x <= xEnd; x + = 0.01f)
LineTo (hdc, xO + xScale * x, yO + yScale * sin (x));
system ("pause");
DeleteObject (Pen);
ReleaseDC (NULL, hdc);}
```




Зверніть увагу, як використовується покажчик поточної позиції при малюванні суцільний синусоїди. Спочатку він встановлюється в початкову точку функцією **MoveToEx**. Потім в тілі циклу функція **LineTo** (hdc, Xx, Yx) малює відрізок від положення покажчика до точки Xx, Yx і встановлює покажчик в цю точку. На наступному кроці циклу функція **LineTo** з'єднує відрізком поточне положення вказівника (тобто стару точку Xx, Yx) з новою точкою кривої Xx, Yx і т.д.

Зверніть також увагу на те, що для відображення точки функцією **SetPixel** потрібно задавати тільки колір пікселя, в той час як для малювання відрізка

функцією **LineTo** потрібно створювати перо і завантажувати його в контекст вікна. Якщо в цьому прикладі ви не створите перо, то другий графік ви не побачите, оскільки перо за замовчуванням є чорним (як колір фону).
Константу Pi ми створили самостійно const double Pi = 3.141592;

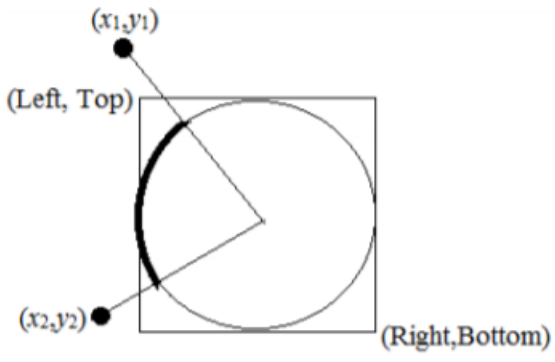
При компіляції ви повинні отримати два попередження (warning) «Conversion from 'float' to 'int', possible loss of data ». Вони означають, що при побудові другого графіка виконувалося автоматичне приведення типу з float в int в аргументах функції **LineTo**. Щоб такого повідомлення не було ви повинні самостійно виконати перетворення типу так, як ми це зробили в функції **SetPixel** при побудові графіка по точках.

**Функції малювання**

**BOOL *LineTo*** (hDC, int x, int y); – малює лінію від поточної позиції до точки, зазначеної в аргументах, і переводить покажчик позиції в цю точку.

**BOOL *Ar*** (hDC, int left, int top, int right, int bottom, int x1, int y1, int x2, int y2); малює дугу еліпса / окружності. Аргументи left, top і right, bottom визначають лівий верхній і правий нижній кути прямокутника, в який вписаний еліпс. Наступні значення – координати точок, від яких будуть проведені прямі до центру еліпса. У місцях їх перетину з еліпсом, починається і закінчується дуга (самі прямі не малюються). Дуга малюється в напрямку проти годинникової стрілки. Використані позначення пояснені на наступному малюнку.

Щоб намалювати контур всього еліпса / кола досить останніх 4 - х параметрів передати нулі. наприклад,
Arc (hdc, 220,100,280,160,0,0,0,0);




Функція Arc покажчик поточної позиції не використовує. Її аналог функція
BOOL ArcTo(hDC, int left, int top, int right, int bottom, int x1, int y1, int x2, int y2); використовує і оновлює положення поточного покажчика. Спочатку від

поточного покажчика до початку дуги функція малює відрізок, потім малює дугу і встановлює поточний покажчик в її кінцеву точку.

**BOOL *AngleArc***(HDC hdc, int X, int Y, DWORD R, FLOAT startAngle, FLOAT sweepAngle); – малює дугу окружності і відрізок від поточної позиції до початку цієї дуги. Дуга є частиною кола радіуса R з центром в точці X, Y. Дуга задається двома кутами, вимірюваними в градусах і відлічуваними проти годинникової стрілки від позитивного напрямку осі X. Параметр startAngle задає початок дуги, а sweepAngle її кутову міру. Функція AngleArc починає малювати від положення поточного покажчика і встановлює його в кінцеву точку своєї дуги. Дуга малюється поточним пером.

**BOOL *Polyline***(hdc, POINT * pt, int cPoints); – малює ламану, що проходить через точки, координати яких знаходяться в масиві ***pt***. Структура POINT визначена в бібліотеці у такий спосіб.

```
typedef struct tagPOINT {
  LONG x;
  LONG y;} POINT;
```

В аргументі cPoints функції Polyline передається кількість точок масиву (> = 2). Наприклад, такі команди малюють трикутник

```
POINT ptr [4] = {{350,90}, {400,140}, {300,140}, {350,90}};
Polyline (hdc, ptr, 4);
```

Функція Polyline не використовує і не оновлює покажчик поточної позиції.

**BOOL *Polyline***To (hdc, POINT * pt, int cPoints); – використовує і оновлює покажчик поточної позиції. ламана починає малюватися від поточного положення покажчика до першої точки масиву pt, і далі проходить через інші точки цього масиву. функція переводить покажчик поточної позиції в кінцеву точку ламаної.

**BOOL *PolyPolyline*** (HDC hdc, constPOINT * lppt, constDWORD * lpdwPolyPoints, DWORD cCount); – малює відразу кілька ламаних. Другий аргумент приймає масив координат вершин всіх шматків ламаної, третій аргумент lpdwPolyPoints - це масив кількостей вершин в кожному відрізку, четвертий передає кількість відрізків.

**BOOL** *TextOut* (HDC hdc, int xStart, int yStart, LPCTSTR lpString, int countChars); – відображає текстову рядок в заданому місці вікна, використовуючи поточні шрифт, колір фону і символів. Аргументи xStart і yStart визначають точку прив'язки текстового рядка lpString. Рядок не зобов'язана закінчуватися нульовим символом, оскільки аргумент countChars містить кількість символів, яке слід надрукувати. З огляду на, що ми говорили про завершальний символ A функцій Windows, фрагмент виклику цієї функції може мати наступний вигляд

```
char str [] = "Три ламаних";    TextOutA (hdc, 360, 40, str, strlen(str));
```

Функція TextOutA не повинна бути останньою в ланцюжку графічних функцій (інакше текст не відображається).

Щоб змінити колір тексту і колір фону, на якому він відображений, використовують функції **SetBkColor** і **SetTextColor**. Функція **SetTextColor** «малює» текст заданим кольором. Функція **SetBkColor** зафарбовує фон між буквами заданим кольором, а також проміжки пунктирних ліній пера, створюваного функцією **CreatePen**.

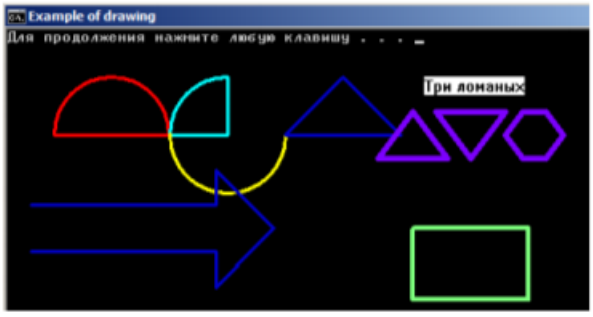
У наступному фрагменті коду показано як можна їх використовувати

```
COLORREF bk = SetBkColor (hdc, RGB (255, 255, 0));
if (bk == CLR_INVALID) cout << "Color error \ n";
SetTextColor (hdc, RGB (255, 0, 0));
TextOutA (hdc, 260, 20, "Червоний текст на жовтому фоні", 28);
```

**Приклад 4.** Демонстрація функцій малювання кривих. У наступному прикладі ми використовуємо описані вище функції для малювання декількох простих контурів.

```
#include <windows.h>
#include <iostream>
using namespace std;
BOOL Line (HDC hdc, int x1, int y1, int x2, int y2);
void main ()
{SetConsoleTitle (L "Example of drawing");
HDC hdc = GetDC (GetConsoleWindow ());
HPEN Pen = CreatePen (PS_SOLID, 3, RGB (255, 0, 0)); // перо 1
SelectObject (hdc, Pen); // поміщаємо перо 1 в контекст
Arc (hdc, 40,40,140,140,200,90,0,90); // Дуга півкола
Line (hdc, 40,90,140,90); // діаметр півкола
HPEN Pen2 = CreatePen (PS_SOLID, 3, RGB (0,255,255)); // перо 2
```

```
SelectObject (hdc, Pen2); // поміщаємо перо 2 в контекст
LineTo (hdc, 190,90); // горизонтальний відрізок
// вертикальний радіус і дуга чверті кола
ArcTo (hdc, 140,40,240,140,190,0,0,90);
HPEN Pen3 = CreatePen (PS_SOLID, 3, RGB (255,255,0)); // перо 3
SelectObject (hdc, Pen3); // поміщаємо перо 3 в контекст
AngleArc (hdc, 190,90,50,180,180); // жовта півколо
HPEN Pen4 = CreatePen (PS_SOLID, 3, RGB (0,0,200)); // перо 4
SelectObject (hdc, Pen4); // поміщаємо перо 4 в контекст
POINT ptr [3] = {{340,90}, {290,40}, {240,90}};
PolylineTo (hdc, ptr, 3); // малюємо трикутник
POINT tArrow [7] = {{20,150}, {180,150}, {180,120}, {230,170},
{180,220}, {180,190}, {20,190}};
Polyline (hdc, ptArrow, 7); // малюємо стрілку
char str [] = "Три ламаних";
TextOutA (hdc, 360,40, str, strlen (str)); // виводимо текст
SelectObject (hdc, CreatePen (PS_SOLID, 5, RGB (125,0,255)));
POINT ptFigs [15] = {{350,70}, {320,110}, {380,110}, {350,70},
{370,70}, {400,110}, {430,70}, {370,70}, {445,70}, {430,90},
{445,110}, {465,110}, {480,90}, {465,70}, {445,70}};
DWORD PolyPoints [3] = {4,4,7}; // малюємо відразу
PolyPolyline (hdc, ptFigs, PolyPoints, 3); // три ламаних
SelectObject (hdc, CreatePen (PS_SOLID, 4, RGB (125,255,125)));
POINT ptRect [5] =
{{350,170}, {450,170}, {450,230}, {350,230}, {350,170}};
Polyline (hdc, ptRect, 5); // малюємо прямокутник
system ("pause");}
BOOL Line (HDC hdc, int x1, int y1, int x2, int y2)
{MoveToEx (hdc, x1, y1, NULL);
return LineTo (hdc, x2, y2);}
```




**Малювання геометричних фігур**

**Пензель**

Є багато функцій, які малюють геометричні фігури (зафарбовані плоскі області). Але також, як для ліній потрібно спершу призначити перо, для фігур треба встановити пензель, яка буде зафарбовувати їх внутрішність.

Є два способи оголосити пензель. Перший – поставити пензель із суцільною заливкою, другий – вказати пензель із заданим стилем заливки. Для цього існують дві функції: **CreateSoldBrush**() і **CreateHatchBrush**(). після створення пензель повинен бути завантажений в контекст пристрою функцією **SelectObject**. Наприклад, в наступному фрагменті ми створюємо суцільний червоний пензель і робимо його активним.

```
HBRUSH hBrush; // створюємо об'єкт – пензель
hBrush = CreateSolidBrush (RGB (255,0,0)); // створюємо суцільний пензель
SelectObject (hdc, hBrush); // робимо пензель активним
```

Після цього графічні функції, які малюють фігури, будуть зафарбовувати їх суцільним червоним пензлем. Цей пензель активний та тих пір, поки у контексті не буде завантажений інший пензель.

Функція оголошення несуцільного пензля має вигляд:

```
HBRUSH CreateHatchBrush (int fnStyle, RGB (r, g, b));
```

Аргумент fnStyle може набувати таких значень:

HS_DIAGONAL – штрихує по діагоналі

HS_CROSS – клітинка

HS_DIAGCROSS – діагональна сітка

HS_FDIAGONAL – по діагоналі в іншу сторону

HS_HORIZONTAL – горизонтальний "тільник"

HS_VERTICAL – вертикальний "паркан"

Ось приклад установки такої кисті

```
HBRUSH hBrush;
hBrush = CreateHatchBrush (HS_CROSS, RGB (255,0,120));
SelectObject (hdc, hBrush); // робимо кисть активної
```

При завершенні роботи програми бажано в контексті консолі відновити вихідний пензель. Наступний фрагмент коду показує правильний спосіб такого відновлення

```
HBRUSH hBrush = CreateSolidBrush (RGB (0, 160, 160));
HBRUSH hBrush2 = (HBRUSH) SelectObject (hdc, hBrush);
.....
// Виклик графічних функцій
.....
DeleteObject (SelectObject (hdc, hBrush2));
```

Функція **SelectObject** зазвичай повертає об'єкт, який був нею замінений. У нашому фрагменті коду пензель hBrush2 – це пензель, який був в констексті до заміни. Останній рядок фрагмента повертає пензель hBrush2 в контекст, функція SelectObject повертає колишній пензельь hBrush, і функція DeleteObject звільняє ресурси від цього пензля.

При старті програми пензлем за замовчуванням є WHITE_BRUSH.

Функція **Rectangle** малює зафарбований прямокутник, тип заливки якого визначається поточним пензлем.

```
BOOL Rectangle (hDC, left, top, right, bottom);
```

Аргументами цієї функції є контекст консолі і координати лівого верхнього і правого нижнього кутів прямокутника. контур цього прямокутника і інших фігур малюється поточним пером.

Функція **RoundRect** малює округлений прямокутник.

```
BOOL RoundRect (hDC, left, top, right, bottom, width, height);
```

Перші п'ять параметрів збігаються з параметрами попередньої функції. Для заокруглення кутів використовуються дуги еліпса, ширина і висота якого задаються в аргументах width і height.

Функція **FillRect** зафарбовує прямокутну область вікна заданої пензлем (без контуру). Прямокутник визначається другим аргументом через покажчик на

структуру RECT. Пензель, що передається третім аргументом, може задаватися дескриптором кисті або константою, що ідентифікує системний колір.

int FillRect (HDC hDC, const RECT * lprc, HBRUSH hbr);

Це може бути корисним для завдання фону консольного вікна, якщо передати їй у другому аргументі розміри консолі.

У нашій наступній програмі ми будемо зафарбовувати все вікно. Для отримання розмірів його прямокутника можна використовувати функцію **GetClientRect**. Вона має наступний сітнтаксис

BOOL GetClientRect (HWND hWnd, LPRECT lpRect);

Тут першим аргументом є ідентифікатор вікна HWND, а другим – покажчик на змінну типу RECT, яка отримає координати лівого верхнього і правого нижнього кутів клієнтської області вікна. наприклад,

```
RECT rectConsole;
GetClientRect (hwnd, & rectConsole);
cout << rectConsole.left << "" << rectConsole.top << "" <<
rectConsole.right << "" << rectConsole.bottom << "\ n";
```

Функція **FrameRect**(HDC, RECT *, HBRUSH) малює контур навколо прямокутної області, заданої другим аргументом, використовуючи пензель, переданий в третьому аргументі. Товщина контуру завжди дорівнює одній логічній одиниці (зазвичай один піксель). наприклад,

```
RECT rct = {300,40,400,180};
FrameRect (hdc, & rct, CreateSolidBrush (RGB (255,0,0)));
```

Функція **InvertRect**(HDC, RECT *) інвертує кольору пік селів всередині прямокутної області, заданої другим аргументом. Звернення складається в застосуванні операції NOT до бітам значення кольору кожного пікселя. наприклад,

```
RECT rc3 = {40,40,120,140};
InvertRect (hdc, & rc3); // інвертування кольорів
```

Функція **Ellipse** малює еліпс або коло.

BOOL Ellipse (HDC hdc, int left, int top, int right, int bottom); Її аргументи задають координати лівого верхнього і правого нижнього кутів прямокутника, в який вписаний еліпс.

Функція **Chord** малює сегмент (область між дугою еліпса і її хордою). Контур сегмента малюється поточним пером, а його нутро зафарбовується поточним пензлем.

BOOL Chord (hDC, int left, int top, int right, int bottom,

int x1, int y1, int x2, int y2);

Аргументи left, top і right, bottom визначають лівий верхній і правий нижній кути прямокутника, в який вписаний еліпс. Наступні значення – координати точок, від яких будуть проведені прямі до центру еліпса. У місцях їх перетину з еліпсом, починається і закінчується дуга.

Дуга малюється в напрямку проти годинникової стрілки, а її кінці з'єднуються хордою. Сенс аргументів x1, y1, x2, y2 такий же, як у функції Arc.

Функція **Pie** малює сектор (область між дугою еліпса і двома її радіальними відрізками). Контур сектора малюється поточним пером, а його внутрішність зафарбовується поточним пензлем.

BOOL Pie (hDC, int left, int top, int right, int bottom,

int x1, int y1, int x2, int y2);

Аргументи функції Pie мають такий же зміст, як у функції Chord.

Функція **BOOL Polygon** (hdc, POINT * pt, int cPoints); малює багатокутник, координати вершин якого розташовані в масиві, ім'я якого передається в другому аргументі. Елементи масиву мають тип POINT. Контур багатокутника малюється поточним пером, а його нутро зафарбовується поточним пензлем. Кількість вершин багатокутника передається в третьому аргументі. Функція Polygon не використовує і не оновлює покажчик поточної позиції. багатокутник замикається автоматично. Ось фрагмент коду побудови трикутника.

POINT ptTriangle [3] = {{310,200}, {280,240}, {340,240}};

Polygon (hdc, ptTriangle, 3);

Зверніть увагу, що аргументи функції Polygon такі ж, як і у функції Polyline.

Функція **PolyPolygon** малює відразу кілька багатокутників, аналогічно тому, що робить функція PolyPolyline.

**BOOL PolyPolygon** (HDC hdc, const POINT * lppt,

const DWORD * lpdwPolyPoints, DWORD cCount)

Другий аргумент приймає масив координат вершин всіх багатокутників, третій аргумент lpdwPolyPoints – це масив кількостей вершин в кожному багатокутнику, четвертий передає кількість багатокутників. Як бачите, аргументи функції PolyPolygon такі ж, як і у функції PolyPolyline.

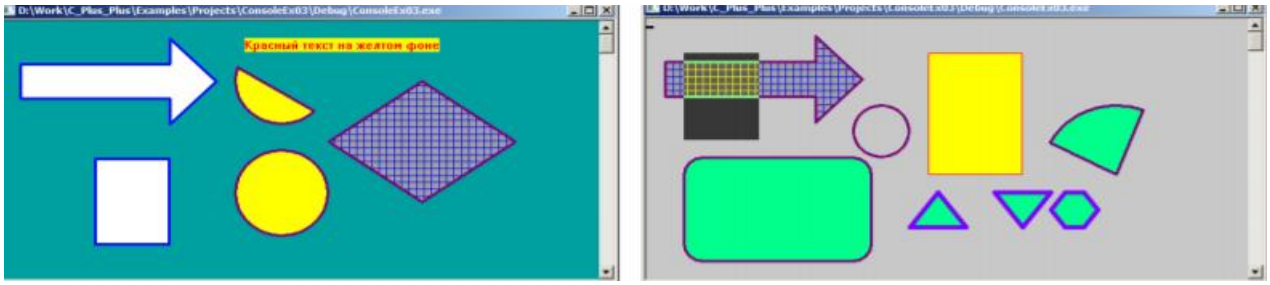
У наступному прикладі ми використовуємо описані вище функції для малювання декількох областей.

**Приклад.** Демонстрація графічних функцій малювання областей.

```
#include <windows.h>
#include <iostream>
#include <conio.h>
using namespace std;
void main ()
{HWND hwnd = GetConsoleWindow (); // ідентифікатор вікна консолі
HDC hdc = GetDC (hwnd); // дескриптор контексту консольного вікна
RECT clientRect; // координати прямокутника
GetClientRect (hwnd, & clientRect); // отримуємо розміри вікна
// заповнюємо фон вікна
FillRect (hdc, & clientRect, CreateSolidBrush (RGB (0,160,160)));
Sleep (100);
COLORREF bk = SetBkColor (hdc, RGB (255,255,0)); // колір фону тексту
if (bk == CLR_INVALID) cout << "color error \ n";
SetTextColor (hdc, RGB (255,0,0)); // Установка кольору тексту
TextOutA (hdc, 260, 20, "Червоний текст на жовтому фоні", 28);
COLORREF penColor = RGB (0,0,255); // синє перо
SelectObject (hdc, CreatePen (PS_SOLID, 3, penColor));
Rectangle (hdc, 100,160,180,260); // кисть за замовчуванням WHITE_BRUSH
POINT ptArrow [7] = {{20,50}, {180,50}, {180,20}, {230,70},
{180,120}, {180,90}, {20,90}};
Polygon (hdc, ptArrow, 7); // використання колір фону за замовчуванням
// Коричневе перо
SelectObject (hdc, CreatePen (PS_SOLID, 3, RGB (120,0,120)));
// Жовтий пензель
SelectObject (hdc, CreateSolidBrush (RGB (255,255,0)));
Ellipse (hdc, 250,150,350,250);
Chord (hdc, 250,20,350,120,150,20,350,120); // сегмент
```

```
SetBkColor (hdc, RGB (160,160,160)); // міняємо задній фон кисті
SelectObject (hdc, CreateHatchBrush (HS_CROSS, RGB (0,0,255)));
POINT romb [4] = {{450,70}, {550,140}, {450,210}, {350,140}};
Polygon (hdc, romb, 4); // малюємо ромб
_getch (); // ---------- Перший екран програми -------
/ Міняємо фон робочого вікна. Стираємо попередні фігури
FillRect (hdc, & clientRect, CreateSolidBrush (RGB (200, 200,200)));
Polygon (hdc, ptArrow, 7); // Полігональна область стрілки
RECT rct = {300,40,400,180};
// заповнений прямокутник без контуру кордону
FillRect (hdc, & rct, CreateSolidBrush (RGB (255,255,0)));
// контур прямокутника малюємо пензлем товщиною 1 піксель
FrameRect (hdc, & rct, CreateSolidBrush (RGB (255, 0, 0)));
RECT rc3 = {40,40,120,140};
InvertRect (hdc, & rc3); // звернення квітів усередині прямокутника
HBRUSH brsh = CreateSolidBrush (RGB (0,255,140));
SelectObject (hdc, brsh);
RoundRect (hdc, 40,160,240,280,40,40); // округлений прямокутник
Arc (hdc, 220,100,280,160,0,0,0,0); // коричневе коло
Pie (hdc, 420, 100, 580, 260, 550, 50, 250, 50); // сектор
SelectObject (hdc, CreatePen (PS_SOLID, 5, RGB (125,0,255)));
POINT ptFigs [15] = {{310,200}, {280,240}, {340,240}, {310,200},
{370,200}, {400,240}, {430,200}, {370,200}, {445,200}, {430,220},
{445,240}, {465,240}, {480,220}, {465,200}, {445,200}};
INT PolyPoints [3] = {4,4,7}; // малюємо відразу 3 багатокутника
PolyPolygon (hdc, ptFigs, PolyPoints, 3);
_getch (); // ---------- Другий екран програми -------}
```

На малюнку ліворуч показано перший кадр програми, праворуч - другий.




У Windows API є багато функцій роботи з графічними областями.

Область – це прямокутна, багатокутна, еліптична фігура або комбінація декількох таких фігур. Над її пікселями можна виконувати різні операції. Вона може бути зафарбована, обрамлена контуром, її кольору можуть бути інвертовані, її можна використовуватися для перевірки виконання клацання миші всередині неї.

Щоб працювати з областю, спочатку вона повинна бути створена. Для цього є цілий ряд функцій, наприклад, CreateRectRgn, CreateRoundRectRgn, CreateEllipticRgn, CreatePolygonRgn і ще кілька інших. Вони використовують свої параметри для опису геометрії області і повертають дескриптор області - об'єкт типу HRGN. Після створення області з нею можна виконувати різні операції.

Для створення прямокутної області використовуються функції **CreateRectRgn** і **CreateRectRgnIndirec**t.

Функція **CreateRectRgn** створює прямокутну область, використовуючи координати лівого верхнього і правого нижнього кутів прямокутника.

HRGN CreateRectRgn (int left, int top, int right, int bottom);

Функція **CreateRectRgnIndirect** створює прямокутну область, використовуючи один аргумент – покажчик на змінну типу RECT, яка містить координати тих же вершин.

HRGN CreateRectRgnIndirect (const RECT * lprc);

Функція **CreateRoundRectRgn** створює область «округленій» прямокутника.

HRGN CreateRoundRectRgn (int left, int top, int right,

int bottom, int widthEllipse, int heightEllipse);

Перші чотири параметри збігаються з параметрами функції CreateRectRgn. Для заокруглення кутів використовуються дуги еліпса, ширина і висота якого задаються в аргументах widthEllipse і heightEllipse.

Функція **CreateEllipticRgn** створює еліптичну область.

HRGN CreateEllipticRgn (int left, int top, int right, int bottom);

Її аргументи задають координати лівого верхнього і правого нижнього кутів прямокутника, в який вписаний еліпс.

Функція **CreateEllipticRgnIndirect** також створює еліптичну область, але використовує один аргумент - покажчик на змінну типу RECT, яка містить координати тих же вершин.

HRGN CreateEllipticRgnIndirect (const RECT * lprc);

Функція **CreatePolygonRgn** створює многокутну область

HRGN CreatePolygonRgn (const POINT * lppt, int cPoints, int fnPolyFillMode);

Координати вершин багатокутника розташовані в масиві, ім'я якого передається в першому аргументі. Елементи масиву мають тип POINT. Кількість вершин багатокутника передається в другому аргументі. Багатокутник замикається автоматично.

Третій аргумент задає спосіб заповнення фону багатокутника. Він може приймати два значення ALTERNATE і WINDING. В режимі ALTERNATE включаються внутрішні області, що знаходяться між першою і другий, третій і четвертій т.д. сторонами контуру. У режимі WINDING включаються всі внутрішні області. Наступний малюнок пояснює відмінність між цими способами.


Функція **CombineRgn** дозволяє виконувати операції теорії множин над двома областями, як множинами точок на площині.

int CombineRgn (HRGN hrgnDest, HRGN hrgnSrc1,

HRGN hrgnSrc2, int fnCombineMode);

Області hrgnSrc1 і hrgnSrc2 комбінуються, а результат поміщається в область hrgnDest. Операція комбінування задається 4-м аргументом fnCombineMode. Він може приймати наступні значення:

 RGN_AND - перетин областей;

 RGN_DIFF - різниця областей;

 RGN_OR - об'єднання областей;

 RGN_XOR - симетрична різниця областей;

 RGN_COPY - створюється копія області, відданою в аргументі hrgnSrc1;

Функція повертає значення, яке визначає тип результуючої області. Це можуть бути значення: NULLREGION (результуюча область порожня), SIMPLEREGION (результат є прямокутником), COMPLEXREGION (результат є більш складною фігурою, ніж прямокутник), ERROR (область не створена).

При виклику функції CombineRgn область результату hrgnDest вже повинна бути створена. Три області, використовувані в аргументах, не зобов'язані бути різними. Наприклад, область hrgnSrc1 може збігатися з hrgnDest.

Є ще одна корисна функція **OffsetRgn**. Вона «зрушує» область.

int OffsetRgn (HRGN hrgn, int nXOffset, int nYOffset);

Її перший аргумент - дескриптор зсувається області, другий і третій задають величину зсуву по горизонталі і вертикалі. Обидві ці величини можуть бути негативними. Зауважимо, що зміщення області не призводить до її автоматичної перемальовуванні. Якщо ви бажаєте побачити область у новому положенні, то на початку її треба стерти (наприклад, зафарбувати кольором фону), потім змістити функцією OffsetRgn, а потім зафарбувати в новому положенні.

Наступний фрагмент пояснює цю послідовність команд.

FillRgn (hdc, rgnBall, bkBrsh);

OffsetRgn (rgnBall, rght, dwn);

FillRgn (hdc, rgnBall, fBrsh);

У цьому фрагменті hdc є дескриптором консолі, rgnBall - зміщується областю, а bkBrsh і fBrsh є кистями фону і заливки області відповідно.

**Приклад.** Зміщення області за допомогою клавіш управління курсором.

```
#include <windows.h>
#include <conio.h>
#define KEY_ARROW_UP 72
#define KEY_ARROW_RIGHT 77
#define KEY_ARROW_DOWN 80
#define KEY_ARROW_LEFT 75
// Зсув кола (область rgnBall) на rght одиниць вправо і dwn одиниць вниз.
//Фарбуємо область кольором фону, зміщуємо область, заливаємо заданим
// кольором
void drawBall (HDC hdc, HRGN rgnBall, HBRUSH bkBrsh,
HBRUSH fBrsh, int rght, int dwn)
{FillRgn (hdc, rgnBall, bkBrsh);
OffsetRgn (rgnBall, rght, dwn);
FillRgn (hdc, rgnBall, fBrsh);}

void main ()
{HWND hwnd = GetConsoleWindow ();
HDC hdc = GetDC (hwnd); // дескриптор консолі
```

```
RECT clientRect;
GetClientRect (hwnd, & clientRect); // отримуємо розміри вікна
Sleep (100);
HBRUSH bkBrush = CreateSolidBrush (RGB (0, 80, 0));
HRGN bgRgn = CreateRectRgnIndirect (& clientRect);
FillRgn (hdc, bgRgn, bkBrush); // заливаємо фон консолі.
intx1 = 50, y1 = 50, r1 = 30; // координати центру кулі і його радіус
HBRUSH ballBrush1 = CreateSolidBrush (RGB (200,0,0));
HRGN rgnBall1 = CreateEllipticRgn (x1-r1, y1-r1, x1 + r1, y1 + r1);
FillRgn (hdc, rgnBall1, ballBrush1);
int iKey = 67;
while (iKey! = 27) // Вихід по клавіші ESC
  {if (_kbhit ())
          {iKey = _getch ();
           switch (iKey){
              case KEY_ARROW_UP:
              drawBall (hdc, rgnBall1, bkBrush, ballBrush1, 0, -10);break;
              case KEY_ARROW_RIGHT:
              drawBall (hdc, rgnBall1, bkBrush, ballBrush1, 10, 0);break;
              case KEY_ARROW_DOWN:
              drawBall (hdc, rgnBall1, bkBrush, ballBrush1, 0, 10);break;
              case KEY_ARROW_LEFT:
              drawBall (hdc, rgnBall1, bkBrush, ballBrush1, -10, 0);break;}
      } }   }
```

У цьому прикладі ми переміщаємо еліптичну область всередині консолі, використовуючи клавіші управління курсором. Для цього ми створили функцію drawBall, яка крім дескриптора консолі hdc і області rgnBall, в якості аргументів приймає пензлі заливки фону bkBrsh і області fBrsh, а також величини зсуву по горизонталі rght і вертикалі dwn