Лекция № 10

Использование функций на примере Windows GDI

Вывод графической информации

Microsoft Windows

DirectDraw
OpenGL
GDI

GDI (Graphics Device Interface) -
подсистема Windows,
отвечает за вывод графики и текста.

Вывод графической информации

Контекст устройства **DC (device context)** – структура данных, содержит параметры и атрибуты вывода графики на устройство.

- палитра доступных цветов;
- параметры пера (рисование линий);
- параметры кисти (закраска и заливка);
- параметры шрифта.

Вывод графической информации

5 типов контекста устройства:

- дисплей **(Display DC)**
- принтер **(Printer DC)**
- память **(Memory DC)**
- метафайл **(Metafile DC)**
- информационный **(Information DC)**

Вывод графической информации

Графические объекты :

перо **(pen)**
  задает режим вывода линий
  (цвет, толщина, стиль);

кисть **(brush)**
  регулирует режим закраски фигур
  (цвет, стиль);

шрифт **(font)**
  задает свойства шрифта,
  которым выводится текст;

…

Вывод графической информации

Работа с графическими объектами при помощи дескрипторов **(handles)**.

HDC,
HPEN,
HBRUSH,
HFONT и т.д.

Создание и удаление объектов производится с помощью соответствующих функций.

Вывод графической информации

---------- Создание пера ----------

```
hPen = CreatePen(PS_SOLID,
                 WIDTH,
                 RGB(R, G, B));
```

PS_SOLID – сплошная линия
PS_DASH – штрихи
PS_DOT – пунктир
PS_DASHDOT – штрих пунтктир

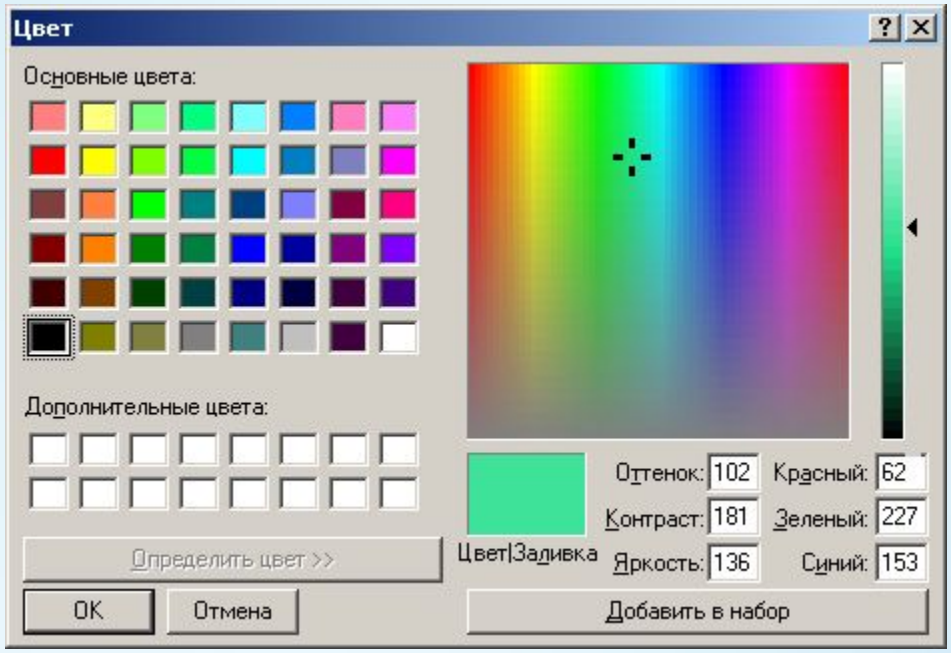
WIDTH – толщина, 0 – один пиксел

R, G, B – интенсивность цвета 0..255

Вывод графической информации

-------- Создание заливки ----------

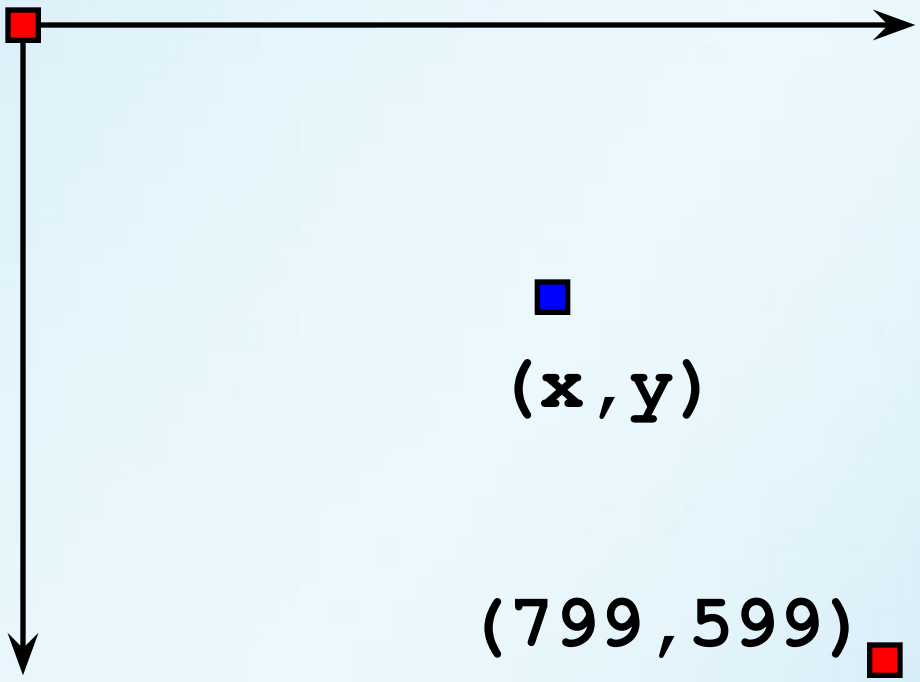
```
hBrush = CreateSolidBrush(RGB(R, G, B));
```



Вывод графической информации

-------- Рисование пиксела ----------

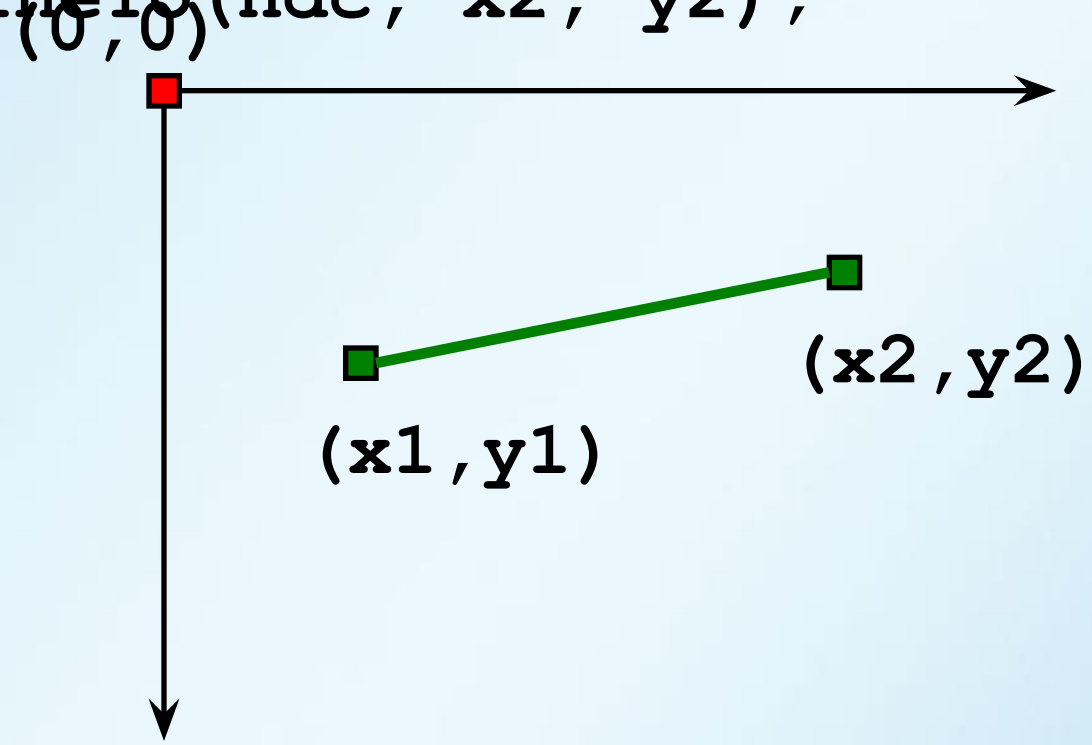
SetPixel(hdc,x,y,RGB(R, G, B))



Вывод графической информации

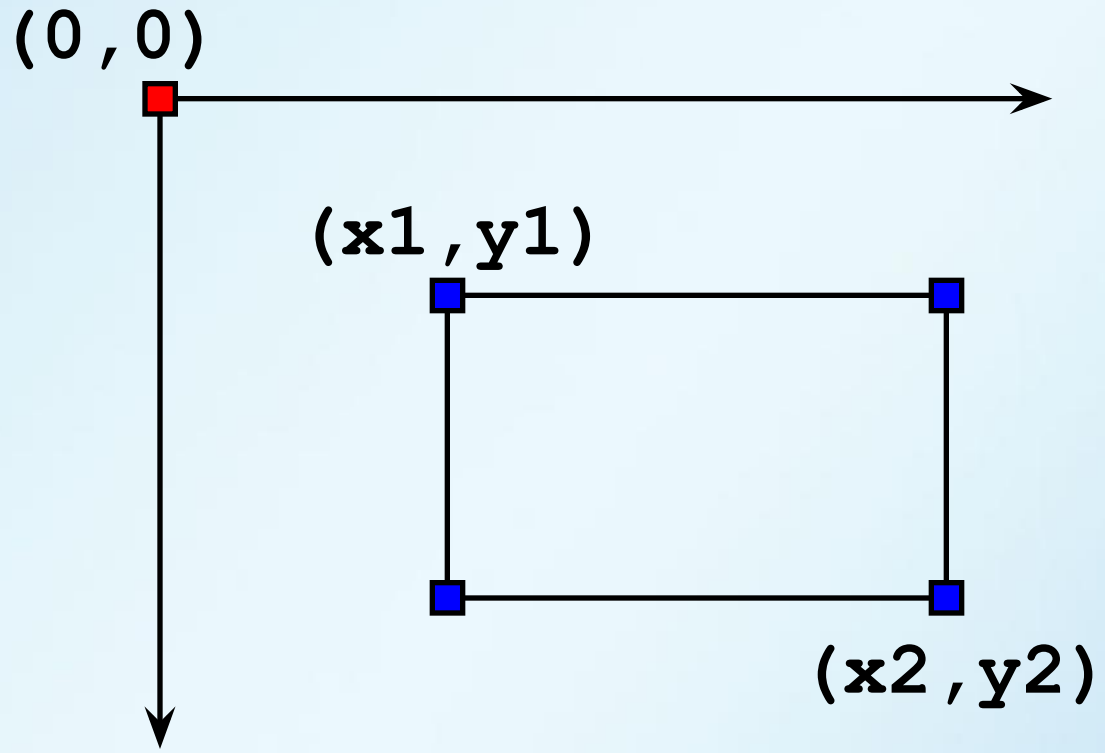
-------- Рисование отрезка ----------

```
MoveToEx(hdc, x1, y1,
NULL);
   LineTo(hdc, x2, y2);
```



Вывод графической информации

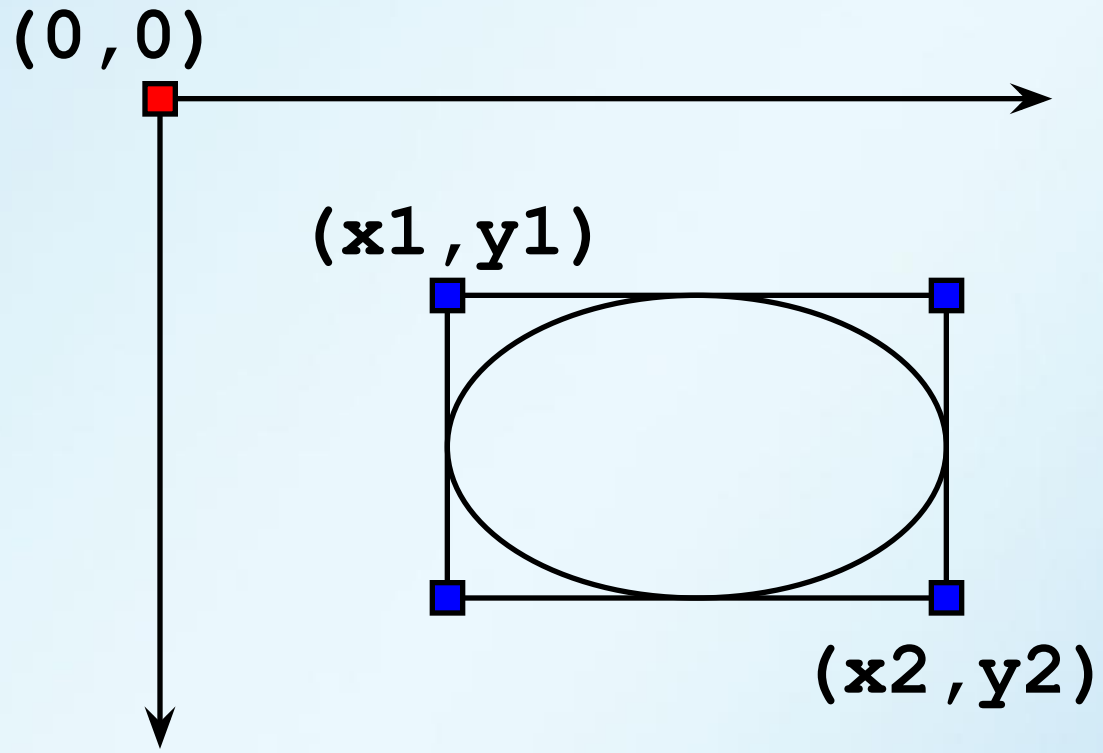
```
----- Рисование прямоугольника -----
b=Rectangle(hdc,x1,y1,x2,y2)
;
```



Вывод графической информации

-------- Рисование эллипса ----------

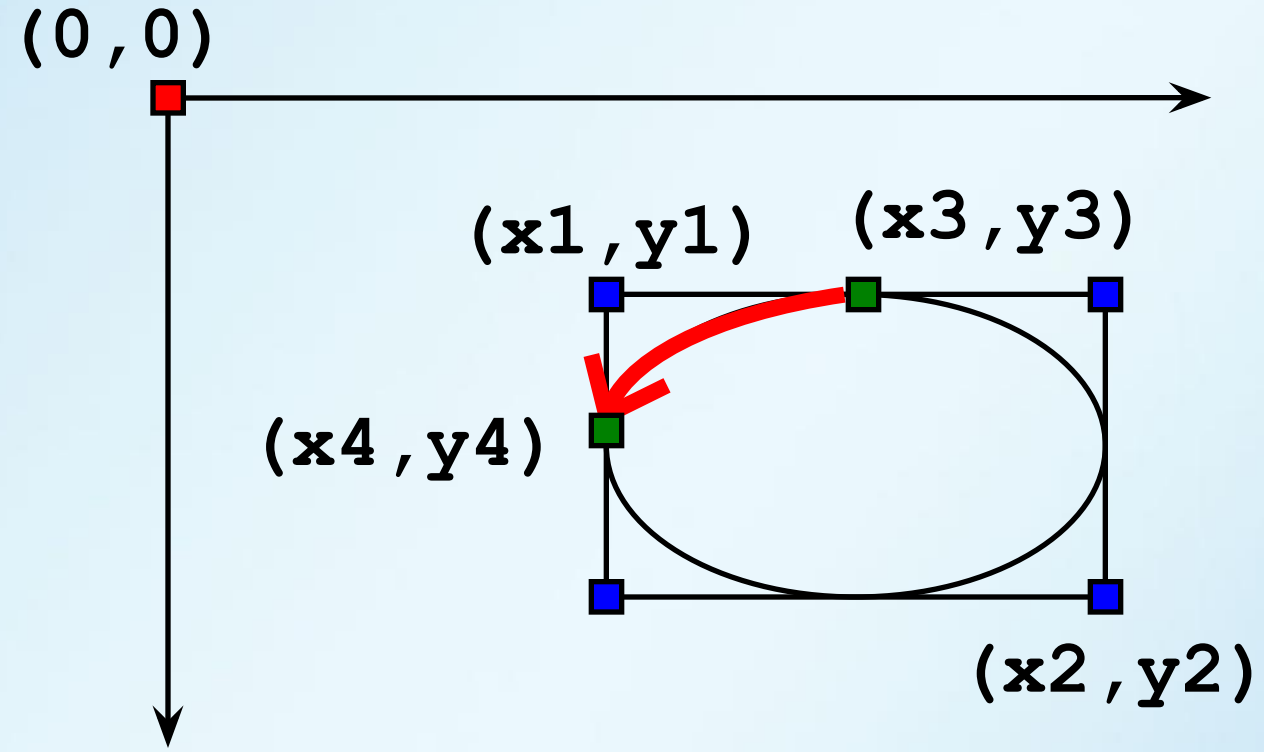
```
b=Ellipse(hdc,x1,y1,x2,y2)
;
```



Вывод графической информации

------- Рисование дуги эллипса -------

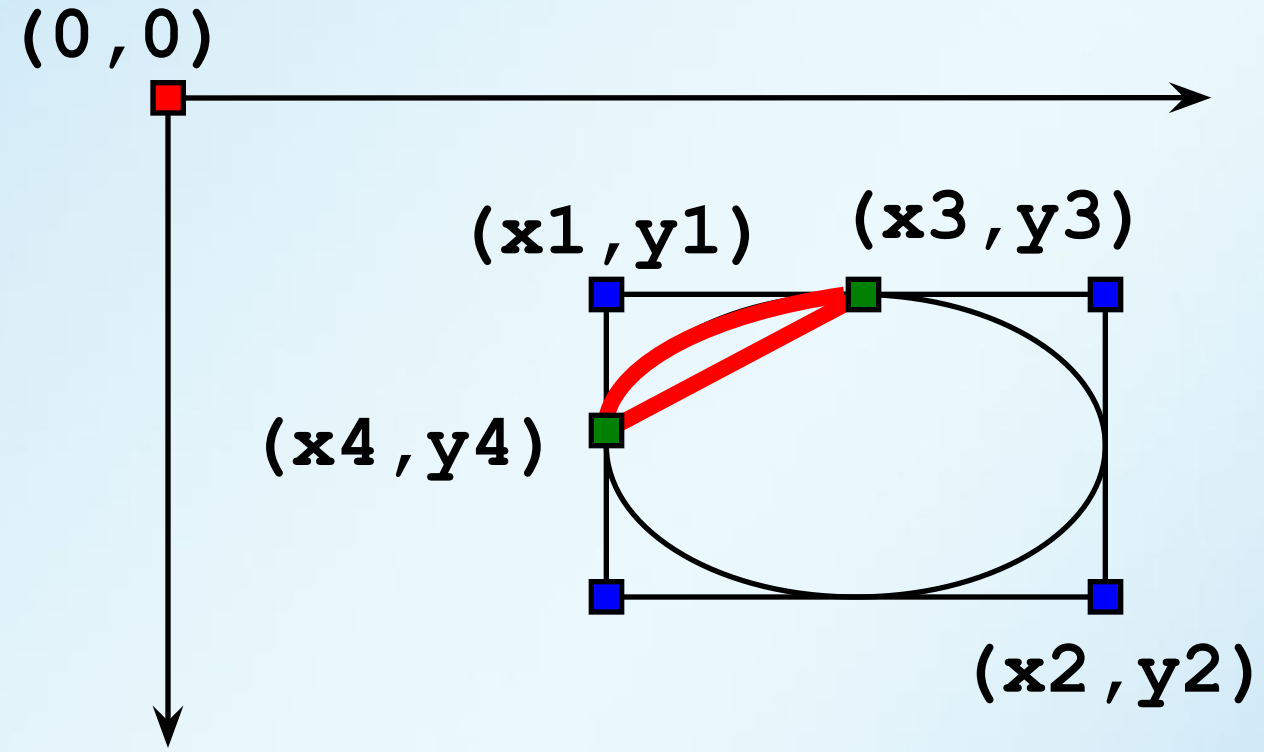
```
b=Arc(hdc,x1,y1,x2,y2,x3,y3,x4,y4)
;
```



Вывод графической информации

----- Рисование сегмента эллипса -----

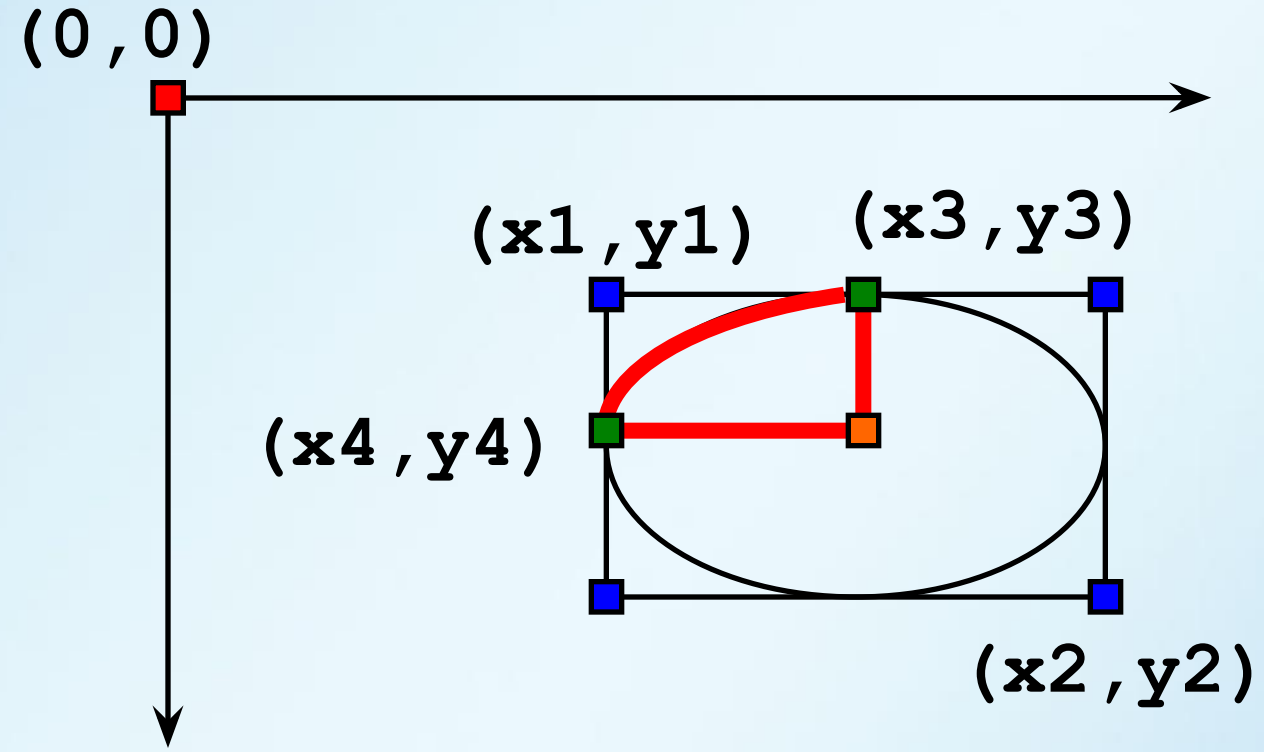
```
b=Chord(hdc,x1,y1,x2,y2,x3,y3,x4,y4)
;
```



Вывод графической информации

----- Рисование сектора эллипса -----

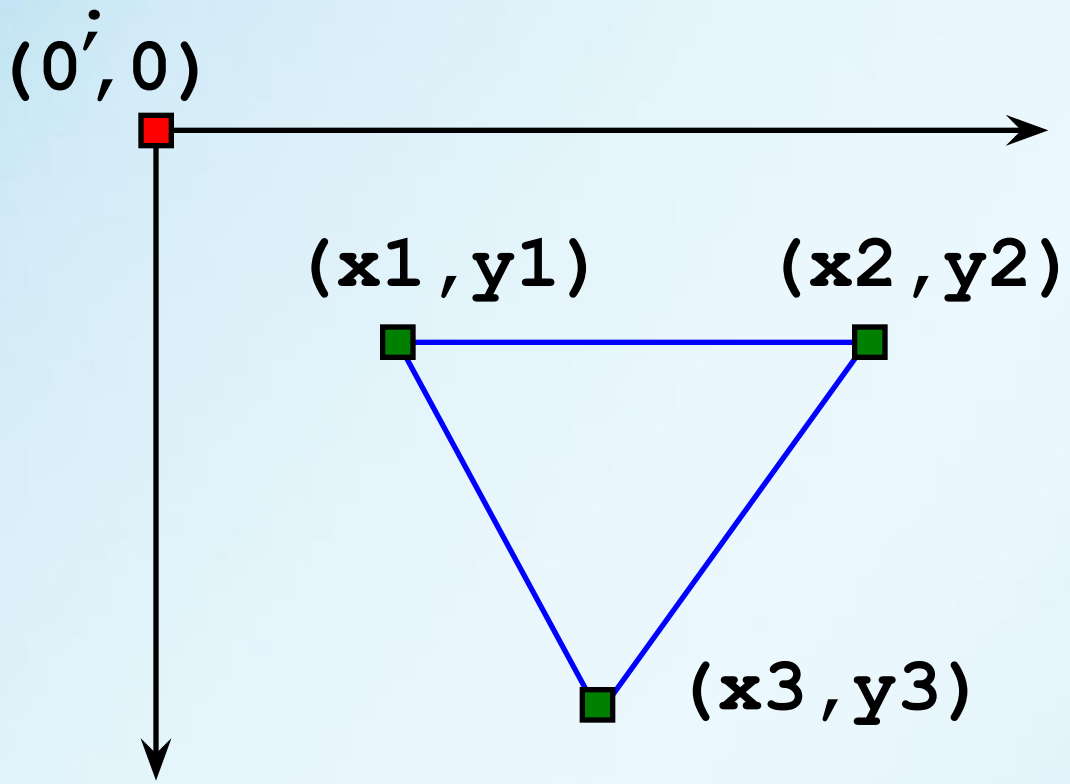
```
b=Pie(hdc,x1,y1,x2,y2,x3,y3,x4,y4)
;
```



Вывод графической информации

------ Рисование многоугольника -----

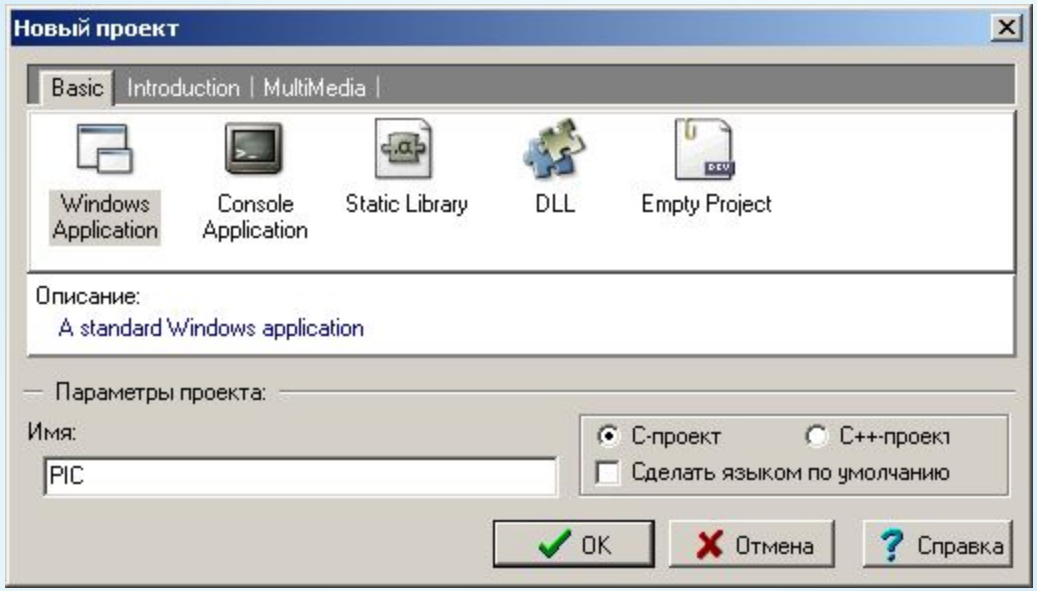
```
b=Polygon(hdc,pt,N)
```




```
const int N=3;
POINT pt[3];

pt[0].x=x1;
pt[0].y=y1;

pt[1].x=x2;
pt[1].y=y2;

pt[2].x=x3;
pt[2].y=y3;
```

Windows Application

Создаем проект Windows Application



Файл pic.c (1)

```
#include <windows.h>

/*  Declare Windows procedure  */
LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

/*  Make the class name into a global variable  */
char szClassName[ ] = "WindowsApp";

int WINAPI WinMain (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nFunsterStil)

{
    HWND hwnd;               /* This is the handle for our window */
    MSG messages;            /* Here messages to the application are saved */
    WNDCLASSEX wincl;        /* Data structure for the windowclass */

    /* The Window structure */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;      /* This function is called by windows */
    wincl.style = CS_DBLCLKS;                 /* Catch double-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Use default icon and mouse-pointer */
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
```

Файл pic.c (2)

```
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;                 /* No menu */
    wincl.cbClsExtra = 0;                      /* No extra bytes after the window class */
    wincl.cbWndExtra = 0;                      /* structure or the window instance */
    /* Use Windows's default color as the background of the window */
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

    /* Register the window class, and if it fails quit the program */
    if (!RegisterClassEx (&wincl))
        return 0;

    /* The class is registered, let's create the program*/
    hwnd = CreateWindowEx (
           0,                   /* Extended possibilites for variation */
           szClassName,         /* Classname */
           "Windows App",       /* Title Text */
           WS_OVERLAPPEDWINDOW, /* default window */
           CW_USEDEFAULT,       /* Windows decides the position */
           CW_USEDEFAULT,       /* where the window ends up on the screen */
           544,                 /* The programs width */
           375,                 /* and height in pixels */
           HWND_DESKTOP,        /* The window is a child-window to desktop */
           NULL,                /* No menu */
           hThisInstance,       /* Program Instance handler */
           NULL                 /* No Window Creation data */
           );
```

Файл pic.c (3)

```
    /* Make the window visible on the screen */
    ShowWindow (hwnd, nFunsterStil);

    /* Run the message loop. It will run until GetMessage() returns 0 */
    while (GetMessage (&messages, NULL, 0, 0))
    {
        /* Translate virtual-key messages into character messages */
        TranslateMessage(&messages);
        /* Send message to WindowProcedure */
        DispatchMessage(&messages);
    }
    /* The program return-value is 0 - The value that PostQuitMessage() gave */
    return messages.wParam;
}
/*  This function is called by the Windows function DispatchMessage()  */
LRESULT CALLBACK WindowProcedure (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)                  /* handle the messages */
    {
        case WM_DESTROY:
            PostQuitMessage (0);       /* send a WM_QUIT to the message queue */
            break;
        default:                      /* for messages that we don't deal with */
            return DefWindowProc (hwnd, message, wParam, lParam);
    }
    return 0;
}
```

Измененный файл pic.c (1)

```
#include <windows.h>
void MyGraph(HWND hwnd);
/*  Declare Windows procedure  */
LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

/*  Make the class name into a global variable  */
char szClassName[ ] = "WindowsApp";

int WINAPI WinMain (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nFunsterStil)

{
    HWND hwnd;               /* This is the handle for our window */
    MSG messages;            /* Here messages to the application are saved */
    WNDCLASSEX wincl;        /* Data structure for the windowclass */

    /* The Window structure */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;      /* This function is called by windows */
    wincl.style = CS_DBLCLKS;                 /* Catch double-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Use default icon and mouse-pointer */
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
```

Измененный файл pic.c (2) без изменений

```
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;                 /* No menu */
    wincl.cbClsExtra = 0;                      /* No extra bytes after the window class */
    wincl.cbWndExtra = 0;                      /* structure or the window instance */
    /* Use Windows's default color as the background of the window */
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

    /* Register the window class, and if it fails quit the program */
    if (!RegisterClassEx (&wincl))
        return 0;

    /* The class is registered, let's create the program*/
    hwnd = CreateWindowEx (
           0,                   /* Extended possibilites for variation */
           szClassName,         /* Classname */
           "Windows App",       /* Title Text */
           WS_OVERLAPPEDWINDOW, /* default window */
           CW_USEDEFAULT,       /* Windows decides the position */
           CW_USEDEFAULT,       /* where the window ends up on the screen */
           544,                 /* The programs width */
           375,                 /* and height in pixels */
           HWND_DESKTOP,        /* The window is a child-window to desktop */
           NULL,                /* No menu */
           hThisInstance,       /* Program Instance handler */
           NULL                 /* No Window Creation data */
           );
```

Измененный файл pic.c (3)

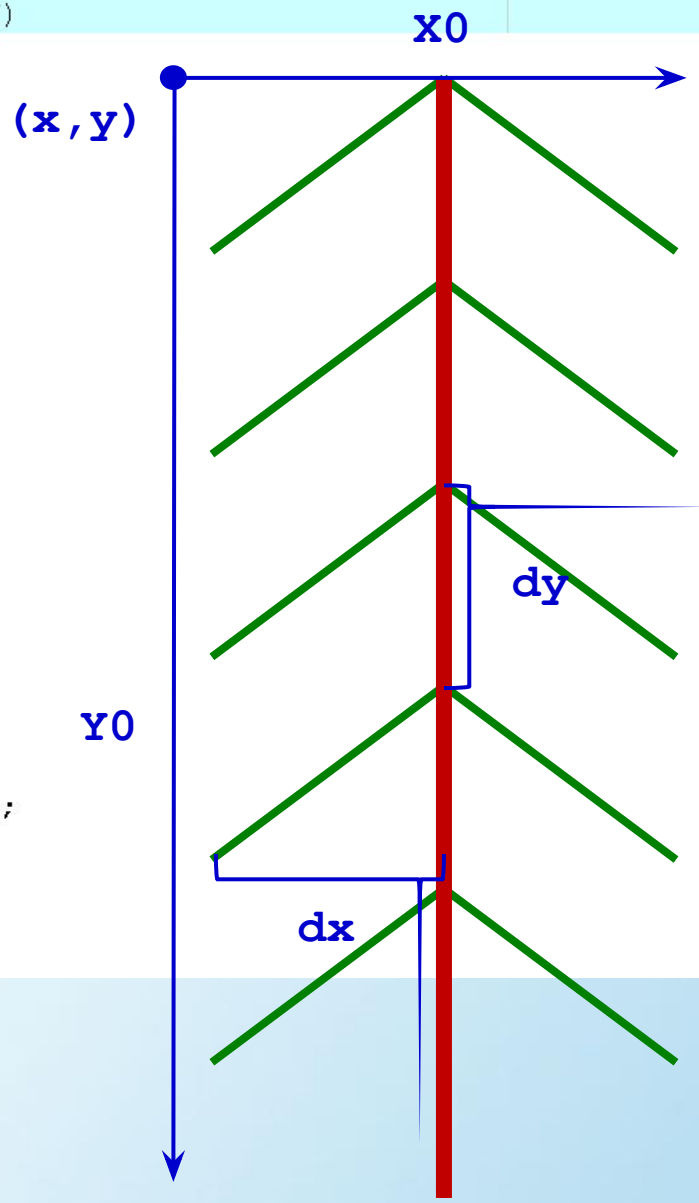
```
    ShowWindow (hwnd, nFunsterStil);

    /* Run the message loop. It will run until GetMessage() returns 0 */
    while (GetMessage (&messages, NULL, 0, 0))
    {
        /* Translate virtual-key messages into character messages */
        TranslateMessage(&messages);
        /* Send message to WindowProcedure */
        DispatchMessage(&messages);
    }
    /* The program return-value is 0 - The value that PostQuitMessage() gave */
    return messages.wParam;
}
/*  This function is called by the Windows function DispatchMessage()  */
LRESULT CALLBACK WindowProcedure (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)                  /* handle the messages */
    {
        case WM_DESTROY:
            PostQuitMessage (0);       /* send a WM_QUIT to the message queue */
            break;

        case WM_PAINT:
            MyGraph(hwnd);
            break;

        default:                      /* for messages that we don't deal with */
```

Измененный файл pic.c (4)

```
            return DefWindowProc (hwnd, message, wParam, lParam);
    }
    return 0;
}

//------------------- Функция рисования ---------------------------------
void MyGraph(HWND hwnd)
{
  PAINTSTRUCT ps;
  HDC hdc;
  HPEN hPen;

  hdc=BeginPaint(hwnd, &ps);

    hPen=CreatePen(PS_SOLID,
              4,RGB(23,80,11));

    SelectObject(hdc,hPen);

    Ellipse(hdc,100,100,
                300,300);

    DeleteObject(hPen);

  EndPaint(hwnd, &ps);
}
```

Функция Tree

```
void Tree(HDC hdc, int x, int y, int X0, int Y0, int N)
{
  HPEN hPen1, hPen2;
  hPen1=CreatePen(PS_SOLID,2,RGB(0,255,0));
  hPen2=CreatePen(PS_SOLID,3,RGB(128,64,64));

  int k; // счетчик веток
  int dx, dy, xt, yt;

  dx=X0/2; dy=Y0/N;
  xt=x+dx;
  yt=y;

  SelectObject(hdc,hPen1);
  for (k=1; k<N-1; k++) // ------- ветки
  {
    MoveToEx(hdc,xt,yt,NULL); LineTo(hdc,xt-dx,yt+dy);
    MoveToEx(hdc,xt,yt,NULL); LineTo(hdc,xt+dx,yt+dy);
    yt=yt+dy;
  }

  SelectObject(hdc,hPen2);  // ------- ствол
  MoveToEx(hdc,x+X0/2,y,NULL); LineTo(hdc,x+X0/2,y+Y0);

  DeleteObject(hPen1);
  DeleteObject(hPen2);
}
```



Функция MyGraph

```
//-------------------- Функция рисования ---------------------------------
void MyGraph(HWND hwnd)
{
  PAINTSTRUCT ps;
  HDC hdc;
  HPEN hPen;

  hdc=BeginPaint(hwnd, &ps);
  int k;

  for (k=1; k<30; k++)    Tree(hdc,rand()%1000+100,rand()%500+100,30,100,10);


  EndPaint(hwnd, &ps);
}
```