МИНЕСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

Отчет к лабораторной работе № 7

Дисциплина: «Системное программирование»

Михаил Чуворкин
11.12.2021

Лабораторная работа 7

Битовые карты и метафайлы

Подготовка к лабораторной работе

Для выполнения лабораторной работы необходимо ознакомиться (по литературе или в сети) и сделать краткое описание следующих функций и параметров, а также используемых событий.

1). Работа с битовыми картами в Win API:

- дескриптор **HBITMAP**;

- **CreateCompatibleBitmap**;

Функция **CreateCompatibleBitmap** создает точечный рисунок, совместимый с устройством, которое связано с заданным контекстом устройства.

```
HBITMAP CreateCompatibleBitmap(
  HDC hdc,         // дескриптор DC
  int nWidth,      // ширина рисунка, в пикселях
  int nHeight      // высота рисунка, в пикселях
);
```

Если функция завершается успешно, возвращаемое значение - дескриптор совместимого точечного рисунка (аппаратно-зависимая точечная картинка (**DDB**)). Иначе – возвращается NULL.

- **CreateCompatibleDC**;

Функция **CreateCompatibleDC** создает контекст устройства в памяти (**DC**), совместимый с заданным устройством.

```
HDC CreateCompatibleDC(
  HDC hdc    // дескриптор DC
);
```

Если функция завершается успешно, возвращаемое значение - дескриптор контекста устройства в памяти. Иначе – возвращается NULL.

- **BitBlt** (режимы копирования: SRCCOPY, SRCINVERT,SRCPAINT и т.д.

Функция **BitBlt** выполняет передачу битовых блоков данных о цвете, соответствующих прямоугольнику пикселей из заданного исходного контекста устройства в целевой контекст устройства.

```
BOOL BitBlt(
  HDC hdcDest, // дескриптор целевого DC
  int nXDest,  // x-коорд. левого верхнего угла целевого прямоугольника
  int nYDest,  // y-коорд. левого верхнего угла целевого прямоугольника
  int nWidth,  // ширина целевого прямоугольника
  int nHeight, // высота целевого прямоугольника
```

```
  HDC hdcSrc,  // дескриптор исходного DC
  int nXSrc,   // x-коорд. левого верхнего угла исходного прямоугольника
  int nYSrc,   // y-коорд. левого верхнего угла исходного прямоугольника
  DWORD dwRop  // код растровой операции
);
```

Список ниже показывает некоторые общие коды растровых операций.

Значение	Описание
BLACKNESS	Заполняет целевой прямоугольник, используя цвет, связанный с индексом **0** в физической палитре. (Этот цвет является черным для заданной по умолчанию физической палитры.)
CAPTUREBLT	**Windows 98/Me, Windows 2000/XP:** Включает любые окна, которые наслоены поверх вашего окна в результирующем изображении. По умолчанию, изображение содержит только ваше окно.
DSTINVERT	Инвертирует целевой прямоугольник.
MERGECOPY	Объединяет цвета исходного прямоугольника с кистью в текущий момент выбранной в ***hdcDest***, при помощи использования булева оператора **И** (**AND**).
MERGEPAINT	Объединяет цвета инвертированного исходного прямоугольника с цветами целевого прямоугольника при помощи использования булева оператора **ИЛИ** (**OR**).
NOMIRRORBITMAP	**Windows 98/Me, Windows 2000/XP:** Препятствует точечному рисунку быть зеркалируемым.
NOTSRCCOPY	Копирует инвертированный исходный прямоугольник в целевой.
NOTSRCERASE	Комбинирует цвета исходных и целевых прямоугольников при помощи использования булева оператора **ИЛИ** (**OR**) и затем инвертирует получающийся в результате цвет.
PATCOPY	Копирует кисть, в текущий момент выбранную в ***hdcDest***, в целевой точечный рисунок.
PATINVERT	Комбинирует цвета кисти, в текущий момент выбранной в ***hdcDest***, с цветами целевого прямоугольника при помощи использования булева оператора исключающее **ИЛИ** (**XOR**).
PATPAINT	Комбинирует цвета кисти, в текущий момент выбранной в ***hdcDest***, с цветами инвертированного исходного прямоугольника при помощи использования булева оператора **ИЛИ** (**OR**). Результаты этой операции объединяются с цветами целевого прямоугольника при помощи использования булева оператора **ИЛИ** (**OR**).
SRCAND	Комбинирует цвета исходных и целевых прямоугольников при помощи использования булева оператора **И** (**AND**).
SRCCOPY	Копирует исходный прямоугольник непосредственно в целевой прямоугольник.
SRCERASE	Комбинирует инвертированные цвета целевого прямоугольника с цветами исходного прямоугольника при помощи использования булева оператора **И** (**AND**).

SRCINVERT	Комбинирует цвета источникового и целевого прямоугольников при помощи использования булева оператора исключающее **ИЛИ** (**XOR**).
SRCPAINT	Комбинирует цвета источникового и целевого прямоугольников при помощи использования булева оператора **ИЛИ** (**OR**).
WHITENESS	Заполняет целевой прямоугольник, используя цвет, связанный с индексом ***1*** в физической палитре. (Этот цвет является белым для заданной по умолчанию физической палитры.)

- **StretchBlt**;

Функция **StretchBlt** копирует точечный рисунок из исходного прямоугольника в целевой прямоугольник, растягивая или сжимая его, чтобы, в случае необходимости, подогнать под размеры целевого прямоугольника. Система растягивает или сжимает точечный рисунок согласно режиму растяжения, который в текущий момент установлен в приемном контексте устройства.

```
BOOL StretchBlt(
  HDC hdcDest,          // дескриптор приемного DC
  int nXOriginDest,     // x-коорд. верхнего левого угла приёмника
  int nYOriginDest,     // y-коорд. верхнего левого угла приёмника
  int nWidthDest,       // ширина приёмного прямоугольника
  int nHeightDest,      // высота приёмного прямоугольника.
  HDC hdcSrc,           // дескриптор исходного DC
  int nXOriginSrc,      // x-коорд.верхнего левого угла источника
  int nYOriginSrc,      // y-коорд.верхнего левого угла источника
  int nWidthSrc,        // ширина исходного прямоугольника
  int nHeightSrc,       // высота исходного прямоугольника
  DWORD dwRop           // код растровой операции
);
```

- **DeleteDC**;

Функция **DeleteDC** удаляет заданный контекст устройства (**DC**).

```
BOOL DeleteDC(
  HDC hdc    // дескриптор DC
);
```

- **LoadBitmap**(hInstance, MAKEINTRESOURCE(IDB_BITMAP));

Функция **LoadBitmap** загружает заданный ресурс растрового изображения из модуля исполняемого файла.

```
HBITMAP LoadBitmap(
  HINSTANCE hInstance,  // дескриптор экземпляра приложения
  LPCTSTR lpBitmapName  // имя ресурса рисунка
);
```

lpBitmapName - Указатель на символьную строку с нулем в конце, которая содержит название ресурса загружаемого растрового изображения. Или же, этот параметр может состоять из идентификатора ресурса в младшем слове и обнуленного старшего слова. Для создания этого значения может быть использована макрокоманда MAKEINTRESOURCE.

2). Работа с битовыми картами на C#:

Класс System.Drawing.Bitmap

- Конструкторы

  Bitmap(Image) – инициализирует новый экземпляр класса Bitmap из указанного существующего изображения.

  Bitmap(Image, Int32, Int32) – инициализирует новый экземпляр класса Bitmap из указанного существующего изображения, масштабированного до заданного объекта.

  Bitmap(Int32, Int32) – инициализирует новый объект класса Bitmap с заданным размером.

  Bitmap(Int32, Int32, Graphics) - инициализирует новый объект класса Bitmap с заданным размером и с разрешением указанного объекта Graphics.

- методы

DrawImage (<Bitmap>, x, y,Width, Height)

Рисует заданный объект Bitmap в заданном месте, используя указанный размер.

DrawImage(<Bitmap>,<Rect>,<Rect>, GraphicsUnit.Pixel);

Рисует часть объекта Bitmap, определенную с помощью параметра srcRect(третий параметр) в прямоугольник destRect(второй параметр). Последний параметр задает единицы измерения.

- **GetPixel и SetPixel;**

```
public System.Drawing.Color GetPixel (int x, int y);
```

- Возвращает цвет указанного пикселя в этом изображении Bitmap.

```
public void SetPixel (int x, int y, System.Drawing.Color color);
```

Задает цвет указанного пикселя в этом объекте Bitmap.

- **MakeTransparent;**

Делает прозрачным прозрачный цвет по умолчанию (если вызывается без параметров, иначе – переданный цвет) для этого элемента Bitmap.

- **Dispose;**

Освобождает все ресурсы, используемые этим объектом Bitmap.

- **Save(<имя файла>);**

Сохраняет объект Bitmap в указанный файл или поток.

3). Win API. Работа с метафайлами:

- Дескриптор **HMETAFILE**

Дескриптор используется для ссылки на метафайл, после того как он был создан и закрыт.

- **CreateMetaFile(NULL);**

```
HDC WINAPI CreateMetaFile(LPCSTR lpszFileName);
```

Параметр lpszFileName должен указывать на строку, содержащую путь к имени файла, в который будут записаны команды GDI, или NULL. В последнем случае создается метафайл в оперативной памяти.

- **CloseMetaFile;**

```
HMETAFILE WINAPI CloseMetaFile(HDC hdc);
```

Эта функция закрывает метафайл для контекста hdc и возвращает идентификатор метафайла. Идентификатор закрытого метафайла использовать нельзя, так как он не содержит никакой полезной информации.

- **PlayMetaFile;**

```
BOOL WINAPI PlayMetaFile(HDC hdc, HMETAFILE hmf);
```

Эта функция проигрывает метафайл в контексте отображения или контексте устройства

- **CopyMetaFile**(<метафайл>,<имя файла.wmf>);

```
HMETAFILE WINAPI CopyMetaFile(HMETAFILE hmf, LPCSTR lpszFileName);
```

Эта функция копирует метафайл в обычный дисковый файл. Параметр hmf определяет метафайл, параметр lpszFileName содержит путь к имени файла, в который будет записан метафайл .

- **DeleteMetaFile;**

```
BOOL WINAPI DeleteMetaFile(HMETAFILE hmf);
```

Удаление метафайла с помощью функции DeleteMetaFile делает недействительным идентификатор метафайла hmf и освобождает оперативную память, занятую метафайлом. Если метафайл был создан как обычный дисковый файл, функция DeleteMetaFile не удаляет его с диска.

- **GetMetaFile (**<имя файла.wmf>).

```
HMETAFILE WINAPI GetMetaFile(LPCSTR lpszFileName);
```

Эта функция загружает метафайл с диска. Параметр – путь к файлу

Задание

Для выполнения работы создать два приложения: программа Win API и программа на языке C#.

1. Перед началом работы приложения Win API создать две картинки (битовые карты) и включить их в ресурсы программы. Программный код должен обеспечивать вывод на рабочую поверхность окна большой картинки, а вторая (небольшая) картинка должна медленно перемещаться по ее поверхности (анимация). Картинки должны составлять единый сюжет.

   Создадим необходимые глобальные переменные:

```
// задание 1 - битовые карты
HBITMAP hBliss;
HBITMAP hMower;
HWND btnPlayAnimation;
const int idBtnPlayAnimation = 0;
bool isPlayingAnimation = false;
const int idTimer = 1;
static int timerTick = 0;

HDC clientHDC;
HDC blissHDC;

HDC mowerHDC;
```

Кнопку для вызова анимации:

```
btnPlayAnimation = CreateWindow(L"BUTTON", L"Animation", WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 2, 2, 100, 30, hWnd, (HMENU)idBtnPlayAnimation, hInstance, 0);
```

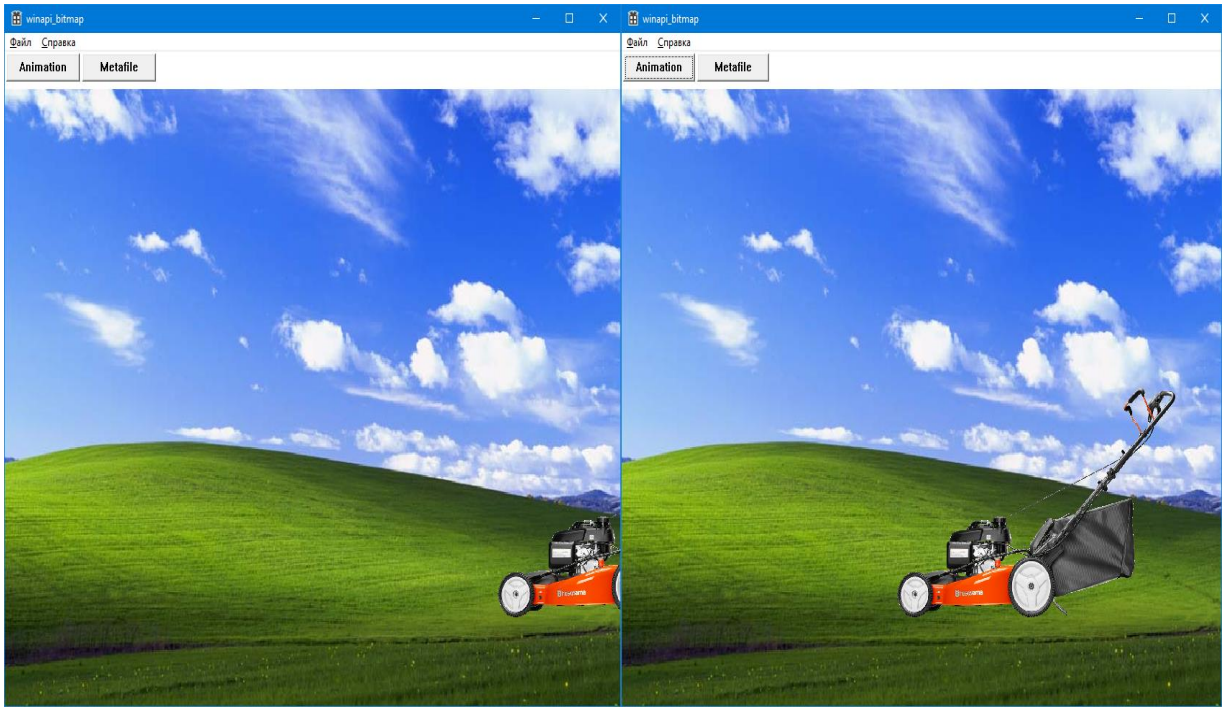
В WM_COMMAND добавим следующее (будем при нажатии изменять состояние флажка, если он true, то загрузим картинки, обнулим таймер и запустим его, получим контекст устройства для рисования на клиентской области и создадим в памяти совместимые устройства и выберем туда карты, при повторном нажатии на кнопки будем освобождать ресурсы):

```
case idBtnPlayAnimation:
            {
                isPlayingAnimation = !isPlayingAnimation;
                isPlayingMetafile = false;
                if (isPlayingAnimation) {
                    // загрузка картинок
                    hBliss = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BLISS));
                    hMower = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_MOWER));

                    timerTick = 0; // обнулить таймер
                    SetTimer(hWnd, idTimer, 33, (TIMERPROC)NULL); // таймер с периодом
33мс
                    clientHDC = GetDC(hWnd); // контекст - клиентская область
                    blissHDC = CreateCompatibleDC(clientHDC); // совместимый контекст в
памяти
```

```
                SelectObject(blissHDC, hBliss); // выбор карты в контекст в памяти

                mowerHDC = CreateCompatibleDC(clientHDC);
                SelectObject(mowerHDC, hMower);

            }
            else {
                // освобождаем ресурсы
                DeleteDC(blissHDC);
                DeleteObject(hBliss);

                DeleteDC(mowerHDC);
                DeleteObject(hMower);

                ReleaseDC(hWnd, clientHDC);
                KillTimer(hWnd, idTimer); // остановить таймер
                InvalidateRect(hWnd, NULL, true);
            }

        }

    break;
```

В WM_PAINT будем рисовать наши картинки в соответствии со значениями таймера и обнулять таймер, если движущаяся картинка ушла за пределы окна:

```
if (isPlayingAnimation) {
    RECT rect;
    GetWindowRect(hWnd, &rect); // размер окна для растяжения
    StretchBlt(clientHDC, 0, 40, rect.right - rect.left, rect.bottom - rect.top - 40,
blissHDC, 0, 0, 800, 600, SRCCOPY);

    int mowerX = rect.right - rect.left - timerTick * 5;
    int mowerY = rect.bottom - rect.top - 400;
    TransparentBlt(clientHDC, mowerX, mowerY, 400, 256, mowerHDC, 0, 0, 893, 574,
RGB(0, 0, 0));
    if (mowerX + 400 < 0) timerTick = 0;
}
```



Замечание: для создания эффекта прозрачности я загрузил 32 битный bmp файл (с альфа каналом) и средствами Visual Studio преобразовал его в 24битный формат. При преобразовании «прозрачные» пиксели стали черными. После этого в функции TransparentBlt я указал цвет, который должен становиться прозрачным как черный.

2. В приложении C# создать слайд-презентацию. На слайде должны постепенно проявляться (двигаться, приближаться, удаляться и т.д.) различные картинки – битовые карты (графики, диаграммы, объекты и т.д.), а также выводиться соответствующий текст.

   Необходимые переменные:

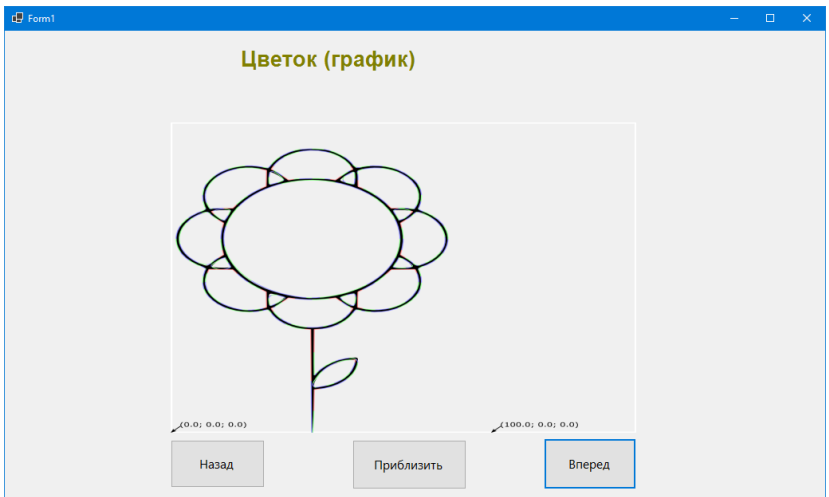
```
Font font = new Font("Arial", 20, FontStyle.Bold); // шрифт для надписи
// имена файлов с картинками
List<string> bitmapsNames = new List<string>{ "FL.png", "genTestSet.png",
"spectrogram.png", "mcm2.png"};
// надписи, соответствующие картинкам
List<string> names = new List<string> {"Цветок (график)", "Сгенерированные цифры",
"Спектрограмма", "Матрица ошибок"};
// список для картинок
List<Bitmap> bitmaps = new List<Bitmap> { };
int currentBitmap = 0;
bool zoomed = false; // флаг состояния, когда приближена картинка
bool isZooming = false; // флаг состояния, когда картинка приближается (отдаляется)
int currentZoom = 0; // текущее значение приближения
int zoomDirection = -1; // направление [приближение / отдаление]

RectangleF textRect; // прямоугольник для текста
```

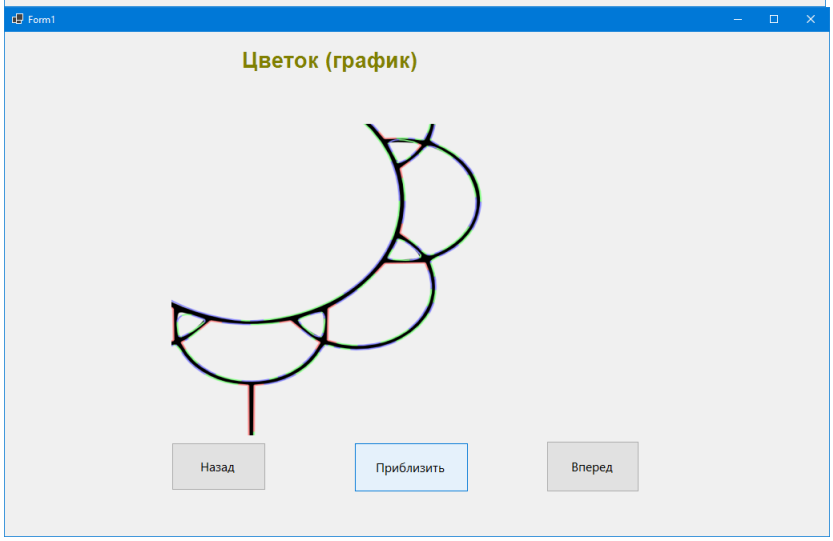
Создадим три кнопки (следующий слайд, предыдущий слайд, приблизить слайд) и напишем обработчики нажатия на них:

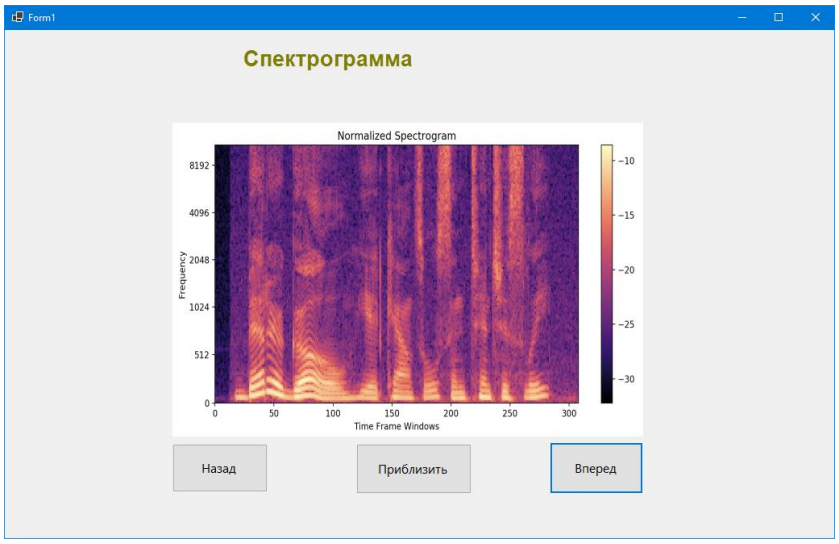
```
private void btnPrev_Click(object sender, EventArgs e)
{
	// уменьшаем индекс текущей картинки (с закольцовыванием)
	if (currentBitmap <= 0) currentBitmap = bitmaps.Count - 1;
	else currentBitmap--;
	zoomed = false;
	Refresh(); // для обновления надписи
}

private void btnZoom_Click(object sender, EventArgs e)
{
	isZooming = true;
	zoomDirection *= -1;
	if (zoomed) currentZoom = 200;
	else currentZoom = 0;
	pictureBox1.Invalidate();
}
private void btnNext_Click(object sender, EventArgs e)
{
	// увеличиваем индекс текущей картинки (с закольцовыванием)
	if (currentBitmap >= bitmaps.Count - 1) currentBitmap = 0;
	else currentBitmap++;
	zoomed = false;
	Refresh(); // для обновления надписи
}
```

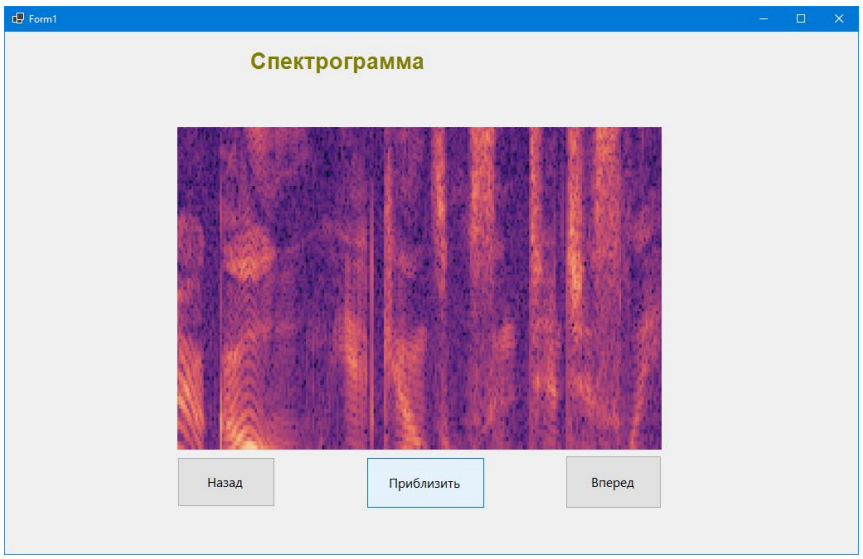
В обработчиках событий Paint формы и picturebox будем показывать (в соответствии с состоянием) картинки и текст:

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
	pictureBox1.Invalidate(); // будем перерисовывать область pictureBox1
	if (isZooming)
	{
		// обработка текущего приближения (выход за пределы приближения)
		if (currentZoom > 200 || currentZoom < 0) {
			isZooming = false;
			zoomed = !zoomed;
		}

		currentZoom += zoomDirection * 5;
		e.Graphics.DrawImage(bitmaps[currentBitmap], 0 - currentZoom, 0 -
currentZoom, pictureBox1.Width + 2 * currentZoom, pictureBox1.Height + 2 * currentZoom);

	}
	else {
		// обработка состояний, когда приближена картинка и нет
		if (zoomed)  e.Graphics.DrawImage(bitmaps[currentBitmap], -200, -200,
pictureBox1.Width + 400, pictureBox1.Height + 400);
		else e.Graphics.DrawImage(bitmaps[currentBitmap], 0, 0, pictureBox1.Width,
pictureBox1.Height);
	}

}

private void Form1_Paint(object sender, PaintEventArgs e)
{
	// на форме рисуем текст
	e.Graphics.DrawString(names[currentBitmap], font, Brushes.Olive, textRect);
}
```


Form1
Цветок (график)
(0.0; 0.0; 0.0)
(100.0; 0.0; 0.0)
Назад
Приблизить
Вперед


Form1
Цветок (график)
Назад
Приблизить
Вперед


Form1
Спектрограмма
Normalized Spectrogram
Frequency
8192
4096
2048
1024
512
0
0
50
100
150
200
250
300
Time Frame Windows
−10
−15
−20
−25
−30
Назад
Приблизить
Вперед



3. В приложении Win API создать метафайл и нарисовать в нем несколько геометрических фигур (сохранить файл). Создать в любом приложении такую же картинку и сохранить в формате .bmp. Сравнить файлы по размеру.

Необходимые глобальные переменные:

```
// задание 3 - метафайлы
HWND btnMetafile;
const int idBtnMetafile = 1;

bool isPlayingMetafile = 0;
```

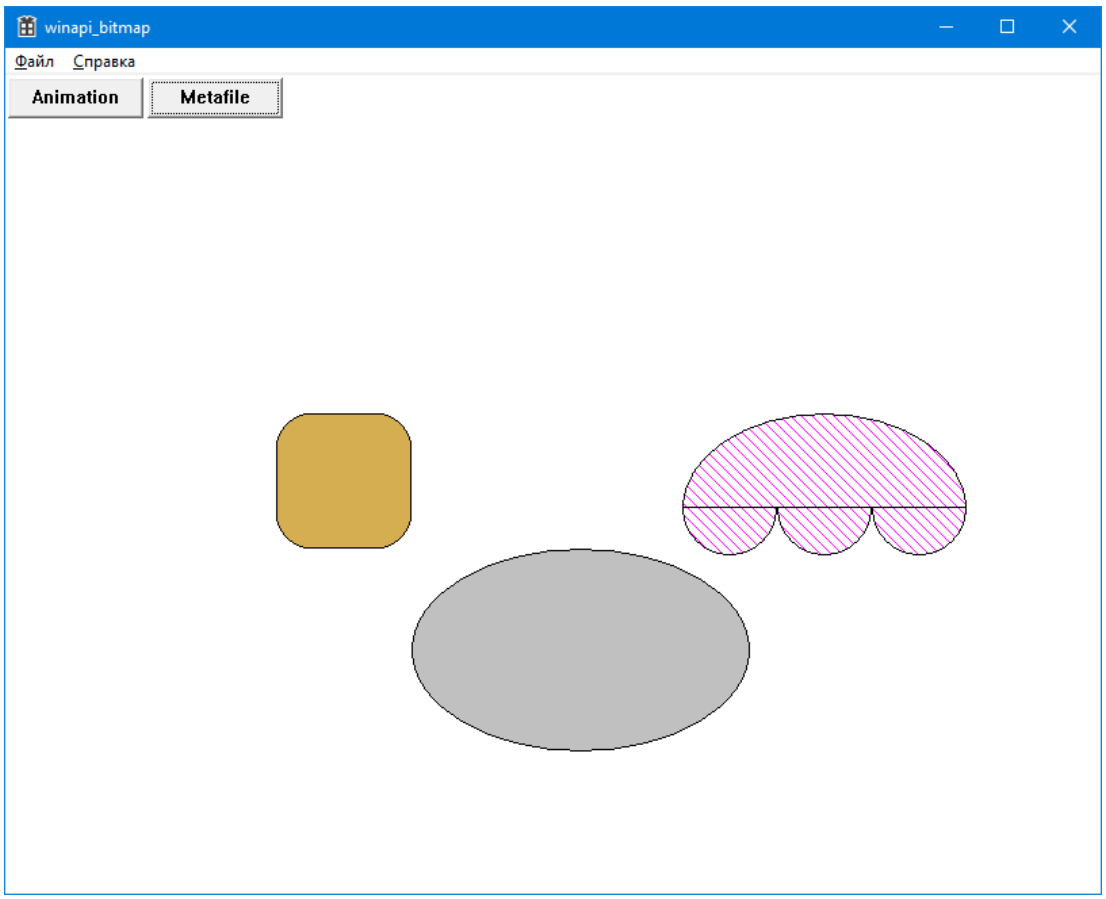
Кнопка для создания метафайла, его проигрывания и сохранения:

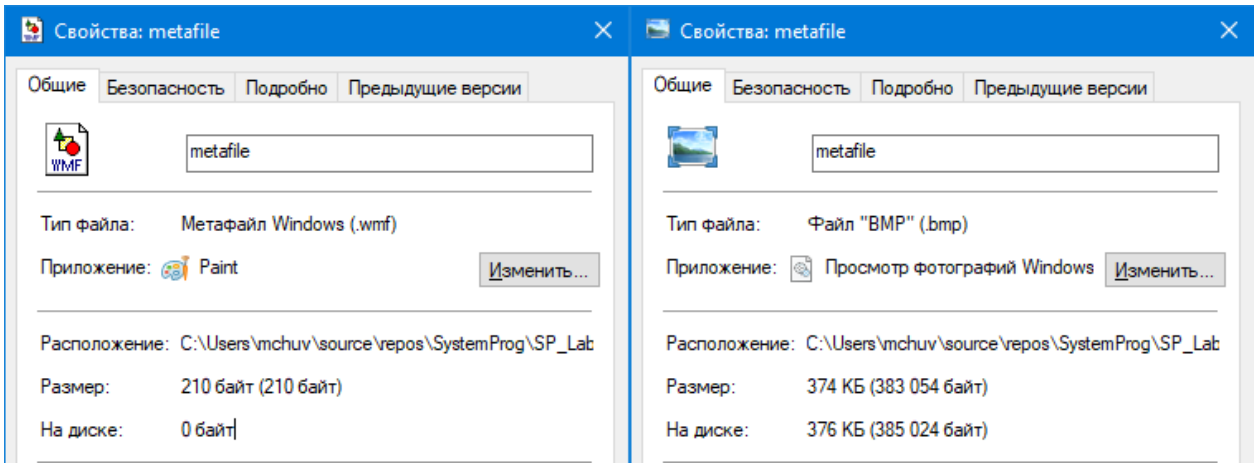
```
btnMetafile = CreateWindow(L"BUTTON", L"Metafile", WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE,
105, 2, 100, 30, hWnd, (HMENU)idBtnMetafile, hInstance, 0);
```

В WM_COMMAND добавим (при нажатии на кнопку создаем метафайл, рисуем в него, проигрываем, сохраняем):

```
case idBtnMetafile:
            {
                // освобождаем ресурсы, если проигрывали анимацию
                if (isPlayingAnimation) {
                    DeleteDC(blissHDC);
                    DeleteObject(hBliss);

                    DeleteDC(mowerHDC);
                    DeleteObject(hMower);

                    ReleaseDC(hWnd, clientHDC);
                    KillTimer(hWnd, idTimer); // остановить таймер
                    isPlayingAnimation = false;
                }

                isPlayingMetafile = !isPlayingMetafile;
                // затирание всего - иначе останется все, что использовалось в анимации
```

```
                InvalidateRect(hWnd, NULL, true);
                UpdateWindow(hWnd);
                if (isPlayingMetafile){


                    HDC hdc = CreateMetaFile(NULL);

                    // Рисуем скругленный прямоугольник случайным цветом
                    SelectObject(hdc, CreateSolidBrush(RGB(rand() % 256, rand() % 256,
rand() % 256)));
                    int xr = 200, yr = 250;
                    RoundRect(hdc, xr, yr, xr + 100, yr + 100, 50, 50);

                    // Рисуем эллипс серой кистью
                    SelectObject(hdc, GetStockObject(LTGRAY_BRUSH));
                    Ellipse(hdc, xr + 100, yr + 100, xr + 350, yr + 250);

                    // Рисуем облако из четырех полукругов с диагональной штриховкой
                    SelectObject(hdc, CreateHatchBrush(HS_FDIAGONAL, RGB(255, 0, 255)));
                    int _size = 70, d1 = 3 * _size, d2 = 2 * _size;
                    int x = 500, y = 250;
                    Chord(hdc, x, y, x + d1, y + d2, x + d1, y + d2 / 2, x, y + d2 / 2);
                    int dx = _size;
                    Chord(hdc, x, y + d2 / 4, x + dx, y + 3 * d2 / 4, x, y + d2 / 2, x +
d1, y + d2 / 2);
                    Chord(hdc, x + dx, y + d2 / 4, x + 2 * dx, y + 3 * d2 / 4, x, y + d2
/ 2, x + d1, y + d2 / 2);
                    Chord(hdc, x + 2 * dx, y + d2 / 4, x + 3 * dx, y + 3 * d2 / 4, x, y +
d2 / 2, x + d1, y + d2 / 2);
                    HMETAFILE hmf = CloseMetaFile(hdc);
                    hdc = GetDC(hWnd);
                    PlayMetaFile(hdc, hmf);
                    CopyMetaFile(hmf, L"metafile.wmf");
                    DeleteMetaFile(hmf);
                }
            }

   break;
```





По свойству размер заметим, что при использовании метафайла занято 210 байт, а при использовании 24bpp bmp формата – 383054 байта (в 1824 раза больше).