Artur Poznański

PROGRAMOWANIE
W WINDOWS 95
DLA CHĘTNYCH

Drobne modyfikacje i format PDF : GDR! <gdr@go2.pl>

Kraków 2000

1. WPROWADZENIE

Postanowiłem napisać kolejny tekst o moich przygodach programistycznych, tym razem z systemem Windows 95. Aby skorzystać z zawartych tutaj informacji będziesz potrzebował trzech rzeczy:

- znajomości języka C
- kompilatora tworzącego 32-bitowy kod (np. Visual C++ 5.0)
- chęci i cierpliwości

Jeżeli nie masz którejś z tych rzeczy, to dalsze czytanie niestety nie ma wielkiego sensu. Napisałem też tekst pt. „Programowanie w języku C dla chętnych" i jeżeli nie spełniasz pierwszego warunku to gorąco zachęcam, aby wpierw go przeczytać. Przedstawione poniżej przykładowe programy będą działały na systemach Windows 95, 98 i NT. Ja używałem kompilatora MS Visual C++ 5.0 firmy Microsoft z pakietu Visual Studio 97 (w skrócie VC++).

2. KOMPILACJA I URUCHAMIANIE

W VC++ wpierw tworzymy projekt wybierając **File** -> **New** -> **Projects** -> **Win 32 Application** i przed wciśnięciem **OK** w pole **Project name** wpisujemy „api" zaś w pole **Location** wpisujemy „C:\VCPP\API". W tym momencie powstaną nam pliki api.dsp (projekt) i api.dsw (obszar roboczy). Kolejny krok to stworzenie pliku. Klikamy na **File** ->**New** -> **Files** -> **C++ Source File**. W polu **File name** wpisujemy „ap1.c". Teraz powstanie plik o nazwie ap1.c, do którego można wpisać poniższy kod.

Przykład1 – ap1.c

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
szCmdLine, int iCmdShow)
{
      MessageBox(NULL, "Witaj mistrzu!", "ap1",MB_OK);
      return 0;
}
```

Kompilacja następuje po naciśnięciu klawisza **F7**. Jeśli nigdzie nie zrobiłeś błędu, to na dole pojawi się tekst „api.exe - 0 error(s), 0 warning(s)" i będzie można nasz program uruchomić kombinacją **Ctrl+F5**. Ukaże się okienko z jednym przyciskiem OK i tekstem „Witaj mistrzu!". Okno można zamknąć naciskając przycisk ok, lub kombinacją **Alt+F4** jak każdą aplikację Windows 95. Nie zapomnij zapisać pliku (**Ctrl+S).** Spróbuj teraz zamknąć kompilator i wejść do katalogu z gotowym plikiem. Będzie to **api.exe** w katalogu C:\VCPP\API\Debug\. Uruchom go i sprawdź jego objętość (ok. 80 kb). Moje gratulacje, właśnie stworzyłeś program pod Windows 95.

3. DEBUG I RELEASE

Kompilator może stworzyć dwie wersje programu. Większą, z informacjami dla debuggera w katalogu **Debug** lub mniejszą - bez tych informacji - o nazwie **Release**.

Otwórz jeszcze raz ten sam projekt i plik.(**File** ->**Open Workspace**->**api.dsw** lub **File**->**Recent Workspaces**->**api**). Następnie wybierz **Build**->**Set Active Configuration** ->**api – Win32 Release** i daj na OK. Skompiluj i zajrzyj do powstałego katalogu Release. Będzie tam mniejsza wersja api.exe o wielkości ok. 20 kb. Tak więc sam się przekonałeś, dlaczego gotowy program znajduje się w katalogu Debug bądź Release.

4. FUNKCJA MESSAGEBOX

Przyjrzyjmy się naszemu jedynemu programowi. Ma on zamiast **main** funkcję **WinMain.** Na razie całkiem pominę pierwszą linię tej funkcji i omówię **MessageBox**. Jej pierwszy argument to tzw. uchwyt. Jest to jakby dowód osobisty tej funkcji, który stwierdza do jakiego okna ona przynależy. W związku z tym, że nasza funkcja nie ma właściciela, daliśmy NULL. Drugi i trzeci argument to tekst w okienku i na pasku tytułowym. Ostatni argument określa typ okienka. Nasz następny program pokaże inną stałą MB (MB to skrót od Message Box).

Przykład2 – ap2.c

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
   PSTR szCmdLine, int iCmdShow)
{
      MessageBox(NULL, "Czy chcesz wyj•• z programu?","Pytanie",
            MB_YESNO|MB_ICONQUESTION);
      return 0;
}
```

Po otwarciu poprzedniego projektu, kliknij na zakładkę **FlieView**, po czym skasuj plik „ap1.c" (**Del**). Następnie dodaj plik „ap2.c" (**Project** -> **Add To Project** ->**Files** -> **ap2.c**). Po uruchomieniu pojawiają się dwa przyciski (Tak i Nie) oraz ikona ze znakiem zapytania. Na razie obojętne co naciśniesz, a i tak program się zakończy, ale w przyszłości użyjemy tej funkcji do bardziej praktycznych zastosowań. Jeszcze jedna ważna rzecz. Funkcja **MessageBox** zwraca nam wartość typu **int** w zależności od tego, jaki przycisk naciśniemy.

Przykład3 – ap3.c

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
   PSTR szCmdLine, int iCmdShow)
{
      int i;
      while(i!=IDYES)
      {
            i=MessageBox(NULL, "Czy chcesz wyj•• z programu?","Pytanie",
            MB_YESNO|MB_ICONQUESTION);
      }
      return 0;
}
```

Jak widać, tym razem program chodzi w kółko dzięki pętli **while** i sprawdzaniu zwracanej wartości przez funkcję **MessageBox**.

5. ŚCIEŻKA POLECEŃ

Jeszcze jedna przeróbka pierwszego programu, która daje ciekawy efekt.

Przykład4 –ap4.c

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
   PSTR szCmdLine, int iCmdShow)
{
      MessageBox(NULL, szCmdLine,"ap4",MB_OK);
      return 0;
}
```

Co program umie? Gdy go uruchomisz, żaden napis się nie pojawi, lecz gdy po skompilowaniu uruchomisz z argumentami, to zostaną one wyświetlone jako tekst w oknie dialogowym . Możesz też wpisać argumenty programu w kompilatorze (**Project** -> **Settings** -> **Debug** pole **Program arguments**). Wpisz tam na przykład tekst „ala ma kota” i uruchom program. Podsumowując rozszyfrowaliśmy, że szCmdLine jest to wskaźnik do ścieżki poleceń o typie PSTR (Pointer to STRing = wskaźnik do łańcucha).

6. SZKIELET PROGRAMU

Przejdziemy do rzeczy nieco trudniejszych więc proszę o wyrozumiałość i cierpliwość. Przejdę od razu do przykładu, który mam nadzieję, że Cię nie przerazi.

Przykład 5 - ap5.c

```
#include <windows.h>

LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);

char szNazwaOkna[] = ”Moje okno";

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszArgs, int nWinMode)
{
      HWND hwnd;
      MSG msg;
      WNDCLASSEX wc;

      wc.hInstance =  hInstance;
      wc.lpszClassName = szNazwaOkna;
      wc.lpfnWndProc = ProcOkna;
      wc.style = 0;
      wc.cbSize = sizeof(WNDCLASSEX);
      wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // du•a ikona
      wc.hIconSm = LoadIcon(NULL, IDI_WINLOGO);  // ma•a ikona
      wc.hCursor = LoadCursor(NULL,IDC_ARROW); // styl kursora
      wc.lpszMenuName = NULL;  // brak menu
      wc.cbClsExtra=0;
      wc.cbWndExtra=0;
      wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH); // t•o na
bia•o
```

```
      // rejestracja klasy okna
      if (!RegisterClassEx(&wc))
      {
   MessageBox(NULL, „Nieudana rejestracja klasy
okna”,"B••d",MB_OK|MB_ICONEXCLAMATION);
            return 0;
      }

      hwnd = CreateWindow(
            szNazwaOkna,             // nazwa okna
            "szkielet Windows 95", // tytu•
            WS_OVERLAPPEDWINDOW,  // styl - normalny
            CW_USEDEFAULT, // pocz•tkowe x
            CW_USEDEFAULT, // pocz•tkowe y
            CW_USEDEFAULT, // szeroko••
            CW_USEDEFAULT, // wysoko••
            HWND_DESKTOP, // brak okien-rodziców
            NULL,         // brak menu
            hInstance,    // uchwyt instancji
            NULL          // brak dodat. argumentów
   );
      ShowWindow(hwnd, nWinMode);   // wy•wietlenie okna
      UpdateWindow(hwnd);

      // p•tla komunikatów
      while(GetMessage(&msg, NULL, 0, 0))
      {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
      }
      return msg.wParam;
}

LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      switch(message)
      {
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
      }
      return 0;
}
```

Uff. Po uruchomieniu pojawia się okienko, które można przesuwać, chować do paska, zmieniać rozmiar itp. Teraz będzie minimum wyjaśnień, byśmy mogli ruszyć dalej. Program w Windows 95 jest sterowany zdarzeniami. Zdarzenia to np. naciśnięcie przycisku, przesunięcie myszy itd... Każdemu zdarzeniu odpowiada wysyłany przez system do twojego programu komunikat. Komunikat to pewna struktura typu MSG z sześcioma polami. Zauważ, że w kodzie są dwie funkcje: **WinMain** i **ProcOkna.** Ta druga nie jest wywoływana z **WinMain**, lecz przez system operacyjny. Robi on to prawie bez przerwy przekazując jej jako parametry cztery pola ze struktury MSG. Zatem w ten sposób Windows wysyła do okna

komunikaty. Mają one nazwę zaczynającą się WM (np. WM_DESTROY). Oto co się dzieje, gdy uruchomisz program i klikniesz na ikonę X, czyli go zamkniesz.

1. Wywołana jest procedura okna, bo nastąpiło zdarzenie.
2. Parametr stanowi komunikat WM_DESTROY, bo tak działa Windows
3. Wywołana zostaje funkcja **PostQuitMessage** (wysyła WM_QUIT)
4. Komunikat WM_QUIT sprawia, że **GetMessage** w **WinMain** zwraca 0
5. Kończy się działanie pętli komunikatów w **WinMain**

Reszty programu nie tłumaczę, bo nie ma takiej potrzeby. Jeszcze tylko dodam, że wszytkie pozostałe komunikaty obsługuje funkcja **DefWindowProc**.

7. FUNKCJA OUTTEXT

Następny program wypisze tekst w naszym oknie.

Przykład 6 – ap6.c

```
#include <windows.h>
#include <stdio.h>

LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);

char szNazwaOkna[] = "Moje okno";
char napis[255];

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszArgs, int nWinMode)
{
   // tu jest to samo co w poprzednim przyk•adzie czyli w ap5.c
}

LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      HDC hdc;
      PAINTSTRUCT ps;
      switch(message)
      {
      case WM_PAINT:
            hdc = BeginPaint(hwnd,&ps);
            sprintf(str,"Witaj mistrzu!");
            TextOut(hdc,0,0,napis,strlen(napis));
            EndPaint(hwnd,&ps);
            break;
      case WM_DESTROY:
            PostQuitMessage(0);
            break;
      default:
            return DefWindowProc(hwnd, message, wParam, lParam);
      }
      return 0;
}
```

Dodaliśmy obsługę nowego komunikatu o nazwie WM_PAINT. Funkcje **BeginPaint** i **EndPaint** muszą być takie jak są napisane, więc ich nie będziemy ruszać. Powiem tylko, że **BeginPaint** daje uchwyt do kontekstu urządzenia (hdc = handle of a device context). Jest to kolejna głupia nazwa będąca jakby furtką umożliwiającą pisanie w naszym okienku. Funkcja **sprintf** umieszcza nasz tekst w tablicy **napis** po to, aby później móc podać jego długość dla

funkcji **TextOut**. Funkcja **TextOut** ma pięć argumentów: pierwszy to wspominany uchwyt, drugi i trzeci to współrzędne x i y początku napisu, czwarty to wskaźnik do łańcucha (napis), zaś piątym jest długość łańcucha. Niestety nie wygląda to tak prosto jak w DOS'ie, ale takie są wymagania trybu graficznego.

8. KOLOR TEKSTU I TŁA

Powiedzmy, że nasz napis ma być niebieski na czerwonym tle. Do poprzedniego przykładu dodaj pomiędzy **sprintf** , a **TextOut** takie dwie linijki:

```
SetTextColor(hdc,RGB(0,0,255));
SetBkColor(hdc,RGB(255,0,0));
```

Pierwsza funkcja ustawia kolor tekstu, zaś druga kolor tła. Trzy liczby w makrze RGB to składowe: czerwona, zielona i niebieska. Są nimi liczby z przedziału 0-255. Spróbuj samemu umieścić wiele napisów, w różnych miejscach i kolorach na ekranie. Jeśli napis nie jest widoczny to znaczy, że prawdopodobnie przekroczyłeś zakres przy podawaniu współrzędnych. Ważną zaletą systemu jest obecność polskich czcionek, dzięki czemu można napisać np. tekst „górny róg".

9. WYZNACZENIE ŚRODKA OBSZARU ROBOCZEGO

Chcemy umieścić nasz napis mniej więcej na środku obszaru okna. Jednak to, jakie wartości wpisać dla początkowych współrzędnych napisu, zależy od tego jaka jest rozdzielczość ekranu oraz rozmiar okna. Zatem wprowadzimy zmienne **cxClient** i **cyClient**, które będą określały wielkość dostępnego obszaru. Nasze wywołanie będzie wyglądać:

```
Outtext(hdc,cxClient/2, cyClient/2,"Witaj mistrzu!");
```

Lecz trzeba jeszcze skąś wziąć wartości **cxClient** i **cyClient**. W tym celu skorzystamy z komunikatu WM_SIZE.

Przykład 7 – ap7.c

```
#include <windows.h>
#include <stdio.h>

LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);

char szNazwaOkna[] = "Moje okno";
char napis[255];

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszArgs, int nWinMode)
{
      HWND hwnd;
      MSG msg;
      WNDCLASSEX wc;

      wc.hInstance =  hInstance;
      wc.lpszClassName = szNazwaOkna;
      wc.lpfnWndProc = ProcOkna;
      wc.style = CS_HREDRAW | CS_VREDRAW;
      wc.cbSize = sizeof(WNDCLASSEX);
      wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // du•a ikona
      wc.hIconSm = LoadIcon(NULL, IDI_WINLOGO);  // ma•a ikona
```

```
      wc.hCursor = LoadCursor(NULL,IDC_ARROW); // styl kursora
      wc.lpszMenuName = NULL;  // brak menu
      wc.cbClsExtra=0;
      wc.cbWndExtra=0;
      wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH); // t•o na
bia•o

      // rejestracja klasy okna
      if (!RegisterClassEx(&wc))
      {
      MessageBox(NULL, "Nieudana rejestracja klasy okna" , "B••d" ,
      MB_OK|MB_ICONEXCLAMATION);
         return 0;
      }

      hwnd = CreateWindow(
            szNazwaOkna,             // nazwa okna
            "Przyk•ad z napisem", // tytu•
            WS_OVERLAPPEDWINDOW,  // styl - normalny
            CW_USEDEFAULT, // pocz•tkowe x
            CW_USEDEFAULT, // pocz•tkowe y
            CW_USEDEFAULT, // szeroko••
            CW_USEDEFAULT, // wysoko••
            HWND_DESKTOP, // brak okien-rodziców
            NULL,         // brak menu
            hInstance,    // uchwyt instancji
            NULL          // brak dodat. argumentów
);
      ShowWindow(hwnd, nWinMode);   // wy•wietlenie okna
      UpdateWindow(hwnd);

      // p•tla komunikatów
      while(GetMessage(&msg, NULL, 0, 0))
      {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
      }
      return msg.wParam;
}

LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static int cxClient, cyClient;
      HDC hdc;
      PAINTSTRUCT ps;
      switch(message)
      {
      case WM_SIZE:
            cxClient = LOWORD(lParam);
            cyClient = HIWORD(lParam);
            break;
      case WM_PAINT:
            hdc = BeginPaint(hwnd,&ps);
            sprintf(napis,"Witaj mistrzu!");
            TextOut(hdc,cxClient/2,cyClient/2,napis, strlen(napis));
            EndPaint(hwnd,&ps);
```

```
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
          default:
            return DefWindowProc(hwnd, message, wParam, lParam);
        }
        return 0;
}
```

W tej chwili początek napisu znajduje się na środku obszaru roboczego okna, niezależnie od jego wielkości. W funkcji **WinMain** zaszła drobna zmiana. Polu **style** przypisaliśmy CS_HREDRAW | CS_VREDRAW co powoduje, że po zmianie rozmiaru okna jest ono odświeżane. System wysyła także komunikat WM_SIZE. W mniej znaczącym słowie zmiennej **lParam** jest szerokość, a bardziej znaczącym – wysokość obszaru roboczego. Obie wartości umieszczamy w zmiennych statycznych **cxClient** i **cyClient**.

10. CENTROWANIE TEKSTU

A co zrobić, gdy tekst jest dłuższy? Nasz napis będzie niesymetrycznie przesunięty w prawo, gdyż w centrum okna znajduje się początek tekstu, a nie środek. Możemy temu zaradzić dodając przed **TextOut** taką oto linijkę:

```
SetTextAlign(hdc, TA_CENTER | TA_BASELINE);
```

Powoduje ona, że możemy bez obawy wyświetić nawet taki napis:

```
sprintf(napis,"Oto tekst na •rodku ekranu, nieco d•u•szy od ostatniego");
```

Funkcja **SetTextAlign** wyrównuje tekst do lewej (TA_LEFT), do prawej (TA_RIGHT) bądź centruje (TA_CENTER). Druga stała powoduje zmianę wysokości w położeniu tekstu.

11. RYSOWANIE LINII

Tym razem naszym celem będzie stworzenie gry w kółko i krzyżyk. Wpierw musimy przygotować planszę w postaci czterech linii, które podzielą obszar na dziewięć części.

Przykład 8 – ap8.c

```
   #include <windows.h>
#include <stdio.h>

   LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);

   char szNazwaOkna[] = "Moje okno";

   int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                      LPSTR lpszArgs, int nWinMode)
   {
      // to samo co w ap7.c
   }

   LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
   {
      static int cxClient, cyClient;
```

```
      HDC hdc;
      PAINTSTRUCT ps;
      switch(message)
      {
      case WM_SIZE:
            cxClient = LOWORD(lParam);
            cyClient = HIWORD(lParam);
            break;
      case WM_PAINT:
            hdc = BeginPaint(hwnd,&ps);
         MoveToEx(hdc, 0, cyClient/3, NULL);
               LineTo(hdc, cxClient, cyClient/3);// 1 linia pozioma
         MoveToEx(hdc, 0, cyClient*2/3, NULL);
               LineTo(hdc, cxClient, cyClient*2/3);// 2 linia pozioma
         MoveToEx(hdc, cxClient/3, 0 , NULL);
               LineTo(hdc, cxClient/3, cyClient);// 1 linia pionowa
         MoveToEx(hdc, cxClient*2/3, 0, NULL);
            LineTo(hdc, cxClient*2/3, cyClient);// 2 linia pionowa
            EndPaint(hwnd,&ps);
            break;
      case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
      }
      return 0;
}
```

Narysowaliśmy cztery linie: dwie poziome i dwie pionowe. Funkcja **MoveToEx** określa początkowe współrzędne dla funkcji rysujących takich jak **LineTo**, która rysuje linię do współrzędnych końcowych punktu podanych jako jej argumenty.

12. OBSŁUGA MYSZY

Chwilowo zostawmy kółko i krzyżyk. Wrócimy do niego nieco później. Następny przkład zademonstruje wykorzystanie myszy.

Przykład 9 - ap9.c

```
   #include <windows.h>
#include <stdio.h>

   LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);

   char szNazwaOkna[] = "Moje okno";
   char napis[255];
   int ilosc = 0;

   int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                      LPSTR lpszArgs, int nWinMode)
   {
     // to samo co w ap7.c
   }

   LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
```

```
   {
static int cxClient, cyClient, cxMyszy, cyMyszy;
HDC hdc;
PAINTSTRUCT ps;
switch(message)
{
      case WM_SIZE:
            cxClient = LOWORD(lParam);
            cyClient = HIWORD(lParam);
            break;
      case WM_LBUTTONDOWN:
            cxMyszy = LOWORD(lParam);
            cyMyszy = HIWORD(lParam);
            ilosc++;
            InvalidateRect(hwnd,NULL,TRUE);
            break;
      case WM_PAINT:
            hdc = BeginPaint(hwnd,&ps);
            sprintf(napis,"%d",ilosc);
            TextOut(hdc, cxMyszy, cyMyszy, napis, strlen(napis));
            EndPaint(hwnd,&ps);
            break;
      case WM_DESTROY:
            PostQuitMessage(0);
            break;
      default:
            return DefWindowProc(hwnd, message, wParam, lParam);
      }
return 0;
}
```

Jest to krótki przykład przedstawiający wykorzystanie małego gryzonia. Po kliknięciu lewym przyciskiem na obszarze roboczym, pojawia się liczba, która za każdym razem się zwiększa. Przedstawia ona ilość kliknięć myszką i przechowuje ją zmienna całkowita **ilosc**. Pojawił się również nowy komunikat WM_LBUTTONDOWN, który system wysyła, gdy tylko naciśniemy lewy przycisk myszki. Podobnie jak w przypadku komunikatu WM_SIZE, w słowach zmiennej **lParam** znajdują się interesujące nas rzeczy. Są to współrzędne położenia kursora w momencie kliknięcia i przechowują je zmienne statyczne **cxMyszy** i **cyMyszy**. Ponieważ funkcje rysujące działają dopiero po wysłaniu WM_PAINT, musimy to wymusić funkcją **InvalidateRect**, która każdorazowo odświeży nam cały obszar roboczy. W identyczny sposób można dodać obsługę WM_RBUTTONDOWN, czyli prawego przycisku myszy, który np. może zmniejszać wartość zmiennej **ilosc** o jeden.

13. WYŚWIETLANIE BITMAP

Chcemy wyświetlić dwie bitmapy, jedną przedstawiającą czerwony krzyżyk, a drugą – niebieskie kółko. Bitmapy będą miały wymiary 128 na 128 pikseli. Zrobimy to w czterech etapach.

Etap 1 – Przygotowanie bitmap
Etap 2 – Przygotowanie pliku zasobów
Etap 3 - Napisanie kodu
Etap 4 - Połączenie wszystkiego razem

Ad 1.

Wpierw włącz będący częścią Windows program **Paint**. Następnie kliknij na opcję **Obraz** ->**Atrybuty** i wpisz w pola **Szerokość** i **Wysokość** po 128 (pikseli). Narysuj duży czerwony krzyżyk i zapisz plik pod nazwą „krzyzyk.bmp". W taki sam sposób przygotuj plik „kolko.bmp". Przegraj oba pliki do katalogu z źródłami (np. C:\VCPP\API).

Ad 2.

Stwórz w katalogu, w którym są bitmapy, plik o nazwie „zasoby1.rc" i wpisz do niego:

Przykład 10
- plik „zasoby1.rc"

```
KRZYZYK BITMAP KRZYZYK.BMP
KOLKO   BITMAP KOLKO.BMP
```

Ad 3.

W pliku ap10.c umieść poniższy kod.

- plik „ap10.c"

```
#include <windows.h>
#include <stdio.h>

LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);

char szNazwaOkna[] = "Moje okno";
char napis[255];

HBITMAP hBit1, hBit2;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        LPSTR lpszArgs, int nWinMode)
{
      HWND hwnd;
      MSG msg;
      WNDCLASSEX wc;

      wc.hInstance =  hInstance;
      wc.lpszClassName = szNazwaOkna;
      wc.lpfnWndProc = ProcOkna;
      wc.style = CS_HREDRAW | CS_VREDRAW;
      wc.cbSize = sizeof(WNDCLASSEX);
      wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // du•a ikona
      wc.hIconSm = LoadIcon(NULL, IDI_WINLOGO);  // ma•a ikona
      wc.hCursor = LoadCursor(NULL,IDC_ARROW); // styl kursora
      wc.lpszMenuName = NULL;  // brak menu
      wc.cbClsExtra=0;
```

```
	wc.cbWndExtra=0;
	wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH); // t•o na
bia•o

	// rejestracja klasy okna
	if (!RegisterClassEx(&wc))
	{
		MessageBox(NULL, "Nieudana rejestracja klasy
		okna","B••d",MB_OK|MB_ICONEXCLAMATION);
				return 0;
	}

	hwnd = CreateWindow(
		szNazwaOkna,			// nazwa okna
		"Kó•ko i krzy•yk", // tytu•
		WS_OVERLAPPEDWINDOW,  // styl - normalny
		CW_USEDEFAULT, // pocz•tkowe x
		CW_USEDEFAULT, // pocz•tkowe y
		CW_USEDEFAULT, // szeroko••
		CW_USEDEFAULT, // wysoko••
		HWND_DESKTOP, // brak okien-rodziców
		NULL,		// brak menu
		hInstance,	// uchwyt instancji
		NULL		// brak dodat. argumentów
);

	hBit1 = LoadBitmap(hInstance,"KRZYZYK");
	hBit2 = LoadBitmap(hInstance,"KOLKO");

	ShowWindow(hwnd, nWinMode);  // wy•wietlenie okna
	UpdateWindow(hwnd);

	// p•tla komunikatów
	while(GetMessage(&msg, NULL, 0, 0))
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}
	return msg.wParam;
}

LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
	HDC hdc;
	HDC memDC;
	PAINTSTRUCT ps;
	switch(message)
	{
	case WM_PAINT:
		hdc = BeginPaint(hwnd,&ps);
		memDC = CreateCompatibleDC(hdc);
		SelectObject(memDC, hBit1);
		BitBlt(hdc, 0, 0, 128, 128, memDC, 0,0,SRCCOPY);
		SelectObject(memDC, hBit2);
		BitBlt(hdc, 128, 0, 128, 128, memDC, 0,0,SRCCOPY);
		EndPaint(hwnd,&ps);
```

```
            DeleteDC(memDC);
            break;
      case WM_DESTROY:
            DeleteObject(hBit1);
            DeleteObject(hBit2);
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
      }
      return 0;
}
```

Ad. 4

Pora jeszcze połączyć wszystko ze sobą. Zatem w VC++ kliknij na zakładkę **FileView** i tym razem dodaj do projektu dwa pliki: „ap10.c" oraz „zasoby1.rc" (**Project->Add To Project->Files...**), po czym całość skompiluj. Po kompilacji pojawi się jeszcze coś o nazwie **External Dependencies** z naszymi bitmapami.Uruchom program, a na ekranie ujrzysz okno, a w nim narysowane przez siebie bitmapy.

Kilka słów o tym jak to się stało, że to zadziałało. Wpierw definiujemy dwie zmienne **hBit1** i **hBit2**, które są uchwytami do bitmap. Uzyskuje je się za pomocą funkcji **LoadBitmap**. Nazwy bitmap są umieszczone w pliku zasobów o rozszerzeniu .rc w postaci *nazwa_bitmapy* BITMAP *plik.bmp*
Kolejna rzecz, którą musimy zrobić, to dostać uchwyt do tzw. kontekstu urządzenia pamięciowego (chodzi po prostu o pamięć). Czyni to funkcja **CreateCompatibleDC.** Następnie za pomocą **SelectObject** umieszczamy w pamięci wybraną bitmapę. Możemy już wreszcie wyświetlić nasz obrazek dzięki **BitBlt**. Tu kopiuje ona piksele bit po bicie z kontekstu pamięci (**memDC**) do kontekstu urządzenia (**hdc**). Dwie pierwsze liczby w wywołaniu funkcji określają początkowe współrzędne, zaś dwie następne rozmiar kopiowanego obszaru (nasze bitmapy są 128x128). Ważne, aby zwolnić pamięć, gdy już jest po wszystkim. Usuwamy kontekst pamięci funkcją **DeleteDC** oraz uchwyty do bitmap funkcją **DeleteObject**. Kompilując plik z rozszerzeniem .rc powstaje plik .res zawierający wszystkie używane zasoby (w tym przypadku dwie bitmapy).

14. ROZCIĄGANIE BITMAP

Połączmy wiedzę zdobytą w kilku poprzednich rozdziałach. Poniższy program, łącząc fragmenty kodu kilku wcześniejszych programów, pozwala na umieszczanie w różnych polach naszych pięknych bitmapek.
Przykład 11 – ap11.c (potrzebny także zasoby1.rc i dwa pliki z bitmapami)

```
#include <windows.h>
#include <stdio.h>

#define KOLO  1
#define KRZYZ 2

LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);

char szNazwaOkna[] = "Moje okno";
char napis[255];

HBITMAP hBit1, hBit2;
```

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszArgs, int nWinMode)
{
      // to samo co w ap10.c
}

LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static int cxClient, cyClient, cxMyszy, cyMyszy;
      static int stan[3][3];
      int x,y;
      HDC hdc;
      HDC memDC;
      PAINTSTRUCT ps;

      switch(message)
      {
      case WM_SIZE:
            cxClient = LOWORD(lParam);
            cyClient = HIWORD(lParam);
            break;
      case WM_LBUTTONDOWN:
            cxMyszy = LOWORD(lParam);
            cyMyszy = HIWORD(lParam);
            x = cxMyszy/(cxClient/3);
            y = cyMyszy/(cyClient/3);
            if (x < 3 && y < 3) stan[x][y]=KOLO;
            InvalidateRect(hwnd,NULL,TRUE);
            break;
      case WM_RBUTTONDOWN:
            cxMyszy = LOWORD(lParam);
            cyMyszy = HIWORD(lParam);
            x = cxMyszy/(cxClient/3);
            y = cyMyszy/(cyClient/3);
            if (x < 3 && y < 3) stan[x][y]=KRZYZ;
            InvalidateRect(hwnd,NULL,TRUE);
            break;
      case WM_PAINT:
            hdc = BeginPaint(hwnd,&ps);
            memDC = CreateCompatibleDC(hdc);
            for (x=0; x<3; x++)
                  for (y=0; y<3; y++)
            {
                  if (stan[x][y]==KRZYZ)
                  {
                        SelectObject(memDC, hBit1);
                        StretchBlt(hdc, x*cxClient/3, y*cyClient/3,
                        cxClient/3, cyClient/3, memDC,
                        0,0,128,128,SRCCOPY);
                  }
                  if (stan[x][y]==KOLO)
                  {
                        SelectObject(memDC, hBit2);
                        StretchBlt(hdc, x*cxClient/3, y*cyClient/3,
                        cxClient/3, cyClient/3, memDC,
                        0,0,128,128,SRCCOPY);
```

```
                }

            }
            MoveToEx(hdc, 0, cyClient/3, NULL);
            LineTo(hdc, cxClient, cyClient/3);// 1 linia pozioma
            MoveToEx(hdc, 0, cyClient*2/3, NULL);
            LineTo(hdc, cxClient, cyClient*2/3);// 2 linia pozioma
            MoveToEx(hdc, cxClient/3, 0 , NULL);
            LineTo(hdc, cxClient/3, cyClient);// 1 linia pionowa
            MoveToEx(hdc, cxClient*2/3, 0, NULL);
            LineTo(hdc, cxClient*2/3, cyClient);// 2 linia pionowa
            EndPaint(hwnd,&ps);
            DeleteDC(memDC);
            break;
      case WM_DESTROY:
            DeleteObject(hBit1);
            DeleteObject(hBit2);
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
      }
      return 0;
}
```

Stworzyliśmy dziewięcioelementową tablicę **stan,** która „pamięta", czy na danym polu znajduje się kółko (wartość 1) czy krzyżyk (2). Pojawiła się również nowa funkcja. Zamiast **BitBlt** użyliśmy **StretchBlt**, której podajemy dodatkowe dwa parametry – wysokość i szerokość źródła, z którego kopiujemy. Funkcja nie tylko skopiuje nam bitmapę, ale ją także odpowiednio rozciągnie, bądź skurczy. Obsłużyliśmy zarówno komunikat lewego przycisku myszy (WM_LBUTTONDOWN) jak i prawego (WM_RBUTTONDOWN). Teoretycznie dałoby się już w „to" grać, lecz przed nami jeszcze daleka droga :).

15. KOMUNIKAT WM_CREATE

Najwyższa pora dodać część związaną ze sprawdzaniem, czy gra się zakończyła oraz wyświetlaniem odpowiednich komunikatów. Uwzględnimy także sytuację remisową, gdy żaden z graczy nie ma swoich trzech figur w jednej linii. Kółko oraz krzyżyk będą pojawiały się na zmianę. Także wielkość naszego okna nie będzie jak poprzednio domyślna, lecz w miarę kwadratowa.

Przykład 12 – ap12.c

```
#include <windows.h>
#include <stdio.h>

#define PUSTE 0
#define KOLO  1
#define KRZYZ 2
#define TAK 1
#define NIE 0

void SprawdzStan(HWND okno);
void Pytanie(HWND okno);
void NowaGra(HWND okno);
BOOL SprawdzKola();
BOOL SprawdzKrzyze();
```

```
BOOL SprawdzRemis();

LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);


char szNazwaOkna[] = "Moje okno";
char napis[255];
int stan[3][3];
static BOOL Gracz1, koniec;

HBITMAP hBit1, hBit2;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszArgs, int nWinMode)
{
      HWND hwnd;
      MSG msg;
      WNDCLASSEX wc;

      wc.hInstance =  hInstance;
      wc.lpszClassName = szNazwaOkna;
      wc.lpfnWndProc = ProcOkna;
      wc.style = CS_HREDRAW | CS_VREDRAW;
      wc.cbSize = sizeof(WNDCLASSEX);
      wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // du•a ikona
      wc.hIconSm = LoadIcon(NULL, IDI_WINLOGO);  // ma•a ikona
      wc.hCursor = LoadCursor(NULL,IDC_ARROW); // styl kursora
      wc.lpszMenuName = NULL;  // brak menu
      wc.cbClsExtra=0;
      wc.cbWndExtra=0;
      wc.hbrBackground  =  (HBRUSH)  GetStockObject(WHITE_BRUSH);  //  t•o  na
bia•o

      // rejestracja klasy okna
      if (!RegisterClassEx(&wc))
      {
            MessageBox(NULL, "Nieudana rejestracja klasy
            okna","B••d",MB_OK|MB_ICONEXCLAMATION);
            return 0;
      }

      hwnd = CreateWindow(
            szNazwaOkna,            // nazwa okna
            "Kó•ko i krzy•yk", // tytu•
            WS_OVERLAPPEDWINDOW,  // styl - normalny
            20, // pocz•tkowe x
            20, // pocz•tkowe y
            400, // szeroko••
            400, // wysoko••
            HWND_DESKTOP, // brak okien-rodziców
            NULL,         // brak menu
            hInstance,    // uchwyt instancji
            NULL          // brak dodat. argumentów
);

      hBit1 = LoadBitmap(hInstance,"KRZYZYK");
      hBit2 = LoadBitmap(hInstance,"KOLKO");
```

```
	ShowWindow(hwnd, nWinMode);   // wy•wietlenie okna
	UpdateWindow(hwnd);

	// p•tla komunikatów
	while(GetMessage(&msg, NULL, 0, 0))
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}
	return msg.wParam;
}

LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
	static int cxClient, cyClient, cxMyszy, cyMyszy;
	int x,y;
	HDC hdc;
	HDC memDC;
	PAINTSTRUCT ps;

	switch(message)
	{
	case WM_CREATE:
		NowaGra(hwnd);
		break;
	case WM_SIZE:
		cxClient = LOWORD(lParam);
		cyClient = HIWORD(lParam);
		break;
	case WM_LBUTTONDOWN:
		cxMyszy = LOWORD(lParam);
		cyMyszy = HIWORD(lParam);
		x = cxMyszy/(cxClient/3);
		y = cyMyszy/(cyClient/3);
		if (x < 3 && y < 3)
		{
			if (Gracz1)
			{
				stan[x][y]=KOLO;
				Gracz1=NIE;
			}
			else
			{
				stan[x][y]=KRZYZ;
				Gracz1=TAK;
			}
		}
		InvalidateRect(hwnd,NULL,TRUE);
		break;
	case WM_PAINT:
		hdc = BeginPaint(hwnd,&ps);
		memDC = CreateCompatibleDC(hdc);
		for (x=0; x<3; x++)
			for (y=0; y<3; y++)
		{
```

```
                if (stan[x][y]==KRZYZ)
                {
                      SelectObject(memDC, hBit1);
                      StretchBlt(hdc, x*cxClient/3, y*cyClient/3,
                      cxClient/3, cyClient/3, memDC,
                      0,0,128,128,SRCCOPY);
                }
                if (stan[x][y]==KOLO)
                {
                      SelectObject(memDC, hBit2);
                      StretchBlt(hdc, x*cxClient/3, y*cyClient/3,
                      cxClient/3, cyClient/3, memDC,
                      0,0,128,128,SRCCOPY);
                }

          }

          MoveToEx(hdc, 0, cyClient/3, NULL);
          LineTo(hdc, cxClient, cyClient/3);// 1 linia pozioma
          MoveToEx(hdc, 0, cyClient*2/3, NULL);
          LineTo(hdc, cxClient, cyClient*2/3);// 2 linia pozioma
          MoveToEx(hdc, cxClient/3, 0 , NULL);
          LineTo(hdc, cxClient/3, cyClient);// 1 linia pionowa
          MoveToEx(hdc, cxClient*2/3, 0, NULL);
          LineTo(hdc, cxClient*2/3, cyClient);// 2 linia pionowa

          EndPaint(hwnd,&ps);
          DeleteDC(memDC);
          SprawdzStan(hwnd);
          if (koniec==TAK)
                DestroyWindow(hwnd);
          break;
    case WM_DESTROY:
          DeleteObject(hBit1);
          DeleteObject(hBit2);
          PostQuitMessage(0);
          break;
      default:
          return DefWindowProc(hwnd, message, wParam, lParam);
    }
    return 0;
}

void SprawdzStan(HWND okno)
{
    if (SprawdzKola())
    {
          MessageBox(okno, "Wygra•o kó•ko!" , "Koniec",MB_OK);
          Pytanie(okno);
    }
    if (SprawdzKrzyze())
    {
          MessageBox(okno, "Wygra• krzy•yk!" , "Koniec",MB_OK);
          Pytanie(okno);
    }
    if (SprawdzRemis() && !SprawdzKola() && !SprawdzKrzyze())
    {
```

```
            MessageBox(okno, "Remis!", "Koniec",MB_OK);
            Pytanie(okno);
      }
}

void Pytanie(HWND okno)
{
      int i;
      i=MessageBox(okno, "Czy grasz od nowa?", "Pytanie",
      MB_YESNO|MB_ICONQUESTION);
      if (i==IDYES)
            NowaGra(okno);
      else
            koniec=TAK;
}

void NowaGra(HWND okno)
{
      int x, y;
      Gracz1=TAK;
      koniec=NIE;
      for (x=0; x<3; x++)
            for (y=0; y<3; y++)
                  stan[x][y]=PUSTE;
      InvalidateRect(okno,NULL,TRUE);
}


BOOL SprawdzKola()
{
      if   ((stan[0][0]==KOLO) &&
            (stan[1][0]==KOLO) &&
            (stan[2][0]==KOLO) || // 1 poziom
            (stan[0][1]==KOLO) &&
            (stan[1][1]==KOLO) &&
            (stan[2][1]==KOLO) || // 2 poziom
            (stan[0][2]==KOLO) &&
            (stan[1][2]==KOLO) &&
            (stan[2][2]==KOLO) || // 3 poziom
            (stan[0][0]==KOLO) &&
            (stan[0][1]==KOLO) &&
            (stan[0][2]==KOLO) || // 1 pion
            (stan[1][0]==KOLO) &&
            (stan[1][1]==KOLO) &&
            (stan[1][2]==KOLO) || // 2 pion
            (stan[2][0]==KOLO) &&
            (stan[2][1]==KOLO) &&
            (stan[2][2]==KOLO) || // 3 pion
            (stan[0][0]==KOLO) &&
            (stan[1][1]==KOLO) &&
            (stan[2][2]==KOLO) || // 1 przek•tna
            (stan[0][2]==KOLO) &&
            (stan[1][1]==KOLO) &&
            (stan[2][0]==KOLO) ) // 2 przek•tna

            return TAK;
      else
```

```
            return NIE;
}

BOOL SprawdzKrzyze()
{
      if   ((stan[0][0]==KRZYZ) &&
            (stan[1][0]==KRZYZ) &&
            (stan[2][0]==KRZYZ) || // 1 poziom
            (stan[0][1]==KRZYZ) &&
            (stan[1][1]==KRZYZ) &&
            (stan[2][1]==KRZYZ) || // 2 poziom
            (stan[0][2]==KRZYZ) &&
            (stan[1][2]==KRZYZ) &&
            (stan[2][2]==KRZYZ) || // 3 poziom
            (stan[0][0]==KRZYZ) &&
            (stan[0][1]==KRZYZ) &&
            (stan[0][2]==KRZYZ) || // 1 pion
            (stan[1][0]==KRZYZ) &&
            (stan[1][1]==KRZYZ) &&
            (stan[1][2]==KRZYZ) || // 2 pion
            (stan[2][0]==KRZYZ) &&
            (stan[2][1]==KRZYZ) &&
            (stan[2][2]==KRZYZ) || // 3 pion
            (stan[0][0]==KRZYZ) &&
            (stan[1][1]==KRZYZ) &&
            (stan[2][2]==KRZYZ) || // 1 przek•tna
            (stan[0][2]==KRZYZ) &&
            (stan[1][1]==KRZYZ) &&
            (stan[2][0]==KRZYZ) ) // 2 przek•tna

            return TAK;
      else
            return NIE;

}

BOOL SprawdzRemis()
{
      if   ((stan[0][0]!=PUSTE) &&
            (stan[1][0]!=PUSTE) &&
            (stan[2][0]!=PUSTE) &&
            (stan[0][1]!=PUSTE) &&
            (stan[1][1]!=PUSTE) &&
            (stan[2][1]!=PUSTE) &&
            (stan[0][2]!=PUSTE) &&
            (stan[1][2]!=PUSTE) &&
            (stan[2][2]!=PUSTE))
            return TAK;
      else
            return NIE;
}
```

Jak widać, kod zaczyna się szybko wydłużać. Będzie tu sporo wyjaśniania z mojej strony, więc zacznę od rzeczy najprostszych. W funkcji **WinMain** zmieniliśmy początkowe rozmiary okna. Zamiast CW_USEDEFAULT wpisane są odpowiednie współrzędne x i y ekranu, na którym wyświetlamy okno. Następną sprawą jest modyfikacja obsługi

WM_LBUTTONDOWN. Po kliknięciu lewym przyciskiem zmienia się wartość **Gracz1** reprezentująca pierwszego gracza. Drugi gracz ma ruch, gdy **Gracz1** otrzyma wartość 0 (stała NIE). Stworzyliśmy także sześć własnych funkcji, które będą zarządzały sprawdzaniem stanu dziewięciu pól (**SprawdzStan**, **SprawdzKrzyze**, **SprawdzKola, SprawdzRemis**) oraz ustawiały początkowe wartości (**NowaGra**). Obsłużyliśmy nowy komunikat o nazwie WM_CREATE, który system wywołuje tylko raz, zaraz po stworzeniu okna. Po skończonej grze wyświetla się okienko z pytaniem, czy gracz zamierza grać od nowa. Zajmuje się tym funkcja **Pytanie**. W razie potrzeby modyfikuje ona zmienną **koniec** sprawdzaną po narysowaniu wszystkich elementów. Ostatnią nową funkcją jest **DestroyWindow.** Powoduje ona to, że system wysyła do naszego okna komunikat WM_DESTROY.

16. WŁASNE MENU

Podczas testowania poprzedniej wersji programu wyszło na jaw, że przy kliknięciu na pole już zajęte, zmienia się jego zawartość. Musimy to poprawić. Poza tym umożliwimy graczowi rozpoczęcie gry od nowa w dowolnym momencie, tworząc menu z opcjami.

Przykład 13

- plik „menu1.h”

```
#define IDM_NOWA     100
#define IDM_WYJSCIE  101
#define IDM_O_GRZE   102
```

- plik „zasoby2.rc”

```
#include "menu1.h"

MOJEMENU MENU
{
      POPUP "&Gra"
      {
            MENUITEM "&Nowa",    IDM_NOWA
            MENUITEM "&Wyj•cie", IDM_WYJSCIE
      }
      POPUP "&Pomoc"
      {
            MENUITEM "&O grze...", IDM_O_GRZE
      }
}

KRZYZYK BITMAP KRZYZYK.BMP
KOLKO   BITMAP KOLKO.BMP
```

- plik „ap13.c”

```
#include <windows.h>
#include <stdio.h>
#include "menu1.h"

#define PUSTE 0
#define KOLO  1
#define KRZYZ 2
#define TAK 1
#define NIE 0

void SprawdzStan(HWND okno);
void Pytanie(HWND okno);
```

```
void NowaGra(HWND okno);
BOOL SprawdzKola();
BOOL SprawdzKrzyze();
BOOL SprawdzRemis();

LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);

char szNazwaOkna[] = "Moje okno";
char napis[255];
int stan[3][3];
static BOOL Gracz1;

HBITMAP hBit1, hBit2;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszArgs, int nWinMode)
{
      HWND hwnd;
      MSG msg;
      WNDCLASSEX wc;

      wc.hInstance =  hInstance;
      wc.lpszClassName = szNazwaOkna;
      wc.lpfnWndProc = ProcOkna;
      wc.style = CS_HREDRAW | CS_VREDRAW;
      wc.cbSize = sizeof(WNDCLASSEX);
      wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // du•a ikona
      wc.hIconSm = LoadIcon(NULL, IDI_WINLOGO);  // ma•a ikona
      wc.hCursor = LoadCursor(NULL,IDC_ARROW); // styl kursora
      wc.lpszMenuName = "MOJEMENU"; // nazwa menu
      wc.cbClsExtra=0;
      wc.cbWndExtra=0;
      wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH); // t•o na
bia•o

      // rejestracja klasy okna
      if (!RegisterClassEx(&wc))
      {
            MessageBox(NULL, "Nieudana rejestracja klasy
okna","B••d",MB_OK|MB_ICONEXCLAMATION);
            return 0;
      }

      hwnd = CreateWindow(
            szNazwaOkna,             // nazwa okna
            "Kó•ko i krzy•yk", // tytu•
            WS_OVERLAPPEDWINDOW,  // styl - normalny
            20, // pocz•tkowe x
            20, // pocz•tkowe y
            400, // szeroko••
            400, // wysoko••
            HWND_DESKTOP, // brak okien-rodziców
            NULL,         // brak menu
            hInstance,    // uchwyt instancji
            NULL          // brak dodat. argumentów
);
```

```
	hBit1 = LoadBitmap(hInstance,"KRZYZYK");
	hBit2 = LoadBitmap(hInstance,"KOLKO");

	ShowWindow(hwnd, nWinMode);   // wy•wietlenie okna
	UpdateWindow(hwnd);

	// p•tla komunikatów
	while(GetMessage(&msg, NULL, 0, 0))
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}
	return msg.wParam;
}

LRESULT CALLBACK ProcOkna(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
	static int cxClient, cyClient, cxMyszy, cyMyszy;
	int x,y;
	HDC hdc;
	HDC memDC;
	PAINTSTRUCT ps;

	switch(message)
	{
	case WM_CREATE:
		NowaGra(hwnd);
		break;
	case WM_COMMAND:
		switch (LOWORD(wParam))
		{
		case IDM_NOWA:
			NowaGra(hwnd);
			break;
		case IDM_WYJSCIE:
			Pytanie(hwnd);
			break;
		case IDM_O_GRZE:
			MessageBox(hwnd,"Kó•ko i krzyzyk\nwer.1.0\nautor: Artur
			Pozna•ski","O grze,,,",MB_OK);
			break;
		}
		break;
	case WM_SIZE:
		cxClient = LOWORD(lParam);
		cyClient = HIWORD(lParam);
		break;
	case WM_LBUTTONDOWN:
		cxMyszy = LOWORD(lParam);
		cyMyszy = HIWORD(lParam);
		x = cxMyszy/(cxClient/3);
		y = cyMyszy/(cyClient/3);
		if ((x < 3 && y < 3) && (stan[x][y]==PUSTE))
		{
			if (Gracz1)
			{
```

```
                  stan[x][y]=KOLO;
                  Gracz1=NIE;
            }
            else
            {
                  stan[x][y]=KRZYZ;
                  Gracz1=TAK;

            }
            InvalidateRect(hwnd,NULL,TRUE);
      }
      break;
case WM_PAINT:
      hdc = BeginPaint(hwnd,&ps);
      memDC = CreateCompatibleDC(hdc);
      for (x=0; x<3; x++)
            for (y=0; y<3; y++)
      {
            if (stan[x][y]==KRZYZ)
            {
                  SelectObject(memDC, hBit1);
                  StretchBlt(hdc, x*cxClient/3, y*cyClient/3,
                  cxClient/3, cyClient/3, memDC,
                  0,0,128,128,SRCCOPY);
            }
            if (stan[x][y]==KOLO)
            {
                  SelectObject(memDC, hBit2);
                  StretchBlt(hdc, x*cxClient/3, y*cyClient/3,
                  cxClient/3, cyClient/3, memDC,
                  0,0,128,128,SRCCOPY);
            }

      }

      MoveToEx(hdc, 0, cyClient/3, NULL);
      LineTo(hdc, cxClient, cyClient/3);// 1 linia pozioma
      MoveToEx(hdc, 0, cyClient*2/3, NULL);
      LineTo(hdc, cxClient, cyClient*2/3);// 2 linia pozioma
      MoveToEx(hdc, cxClient/3, 0 , NULL);
      LineTo(hdc, cxClient/3, cyClient);// 1 linia pionowa
      MoveToEx(hdc, cxClient*2/3, 0, NULL);
      LineTo(hdc, cxClient*2/3, cyClient);// 2 linia pionowa

      EndPaint(hwnd,&ps);
      DeleteDC(memDC);
      SprawdzStan(hwnd);
      break;
case WM_DESTROY:
      DeleteObject(hBit1);
      DeleteObject(hBit2);
      PostQuitMessage(0);
      break;
  default:
      return DefWindowProc(hwnd, message, wParam, lParam);
}
return 0;
```

```
}

void SprawdzStan(HWND okno)
{
      if (SprawdzKola())
      {
            MessageBox(okno, "Wygra•o kó•ko!" , "Koniec",MB_OK);
            NowaGra(okno);
      }
      if (SprawdzKrzyze())
      {
            MessageBox(okno, "Wygra• krzy•yk!" , "Koniec",MB_OK);
            NowaGra(okno);
      }
      if (SprawdzRemis() && !SprawdzKola() && !SprawdzKrzyze())
      {
            MessageBox(okno, "Remis!", "Koniec",MB_OK);
            NowaGra(okno);
      }
}

void Pytanie(HWND okno)
{
      int i;
      i=MessageBox(okno, "Na pewno chcesz wyj••?", "Pytanie",
      MB_YESNO|MB_ICONQUESTION);
      if (i==IDYES)
            DestroyWindow(okno);
}

void NowaGra(HWND okno)
{
      int x, y;
      Gracz1 = TAK;
      for (x=0; x<3; x++)
            for (y=0; y<3; y++)
                  stan[x][y]=0;
      InvalidateRect(okno,NULL,TRUE);
}


BOOL SprawdzKola()
{
      // jak w ap12.c
}

BOOL SprawdzKrzyze()
{
      // jak w ap12.c
}

BOOL SprawdzRemis()
{
      // jak w ap12.c
}
```

Ważne aby skompilować w tym samym projekcie pliki „ap13.c" i „zasoby2.rc". Jakie zmiany zaszły? Pierwszą było dodanie drugiego warunku przy zmianie stanu pola (sprawdzamy czy jest ono puste). Następną stanowiła zmiana treści pytania oraz momentu, w którym się ono pojawia (nie po zakończonej partii, lecz przy próbie wyjścia z gry). Później okazało się, że wyrażenie **koniec==TAK** można zastąpić funkcją **DestroyWindow** i tym samym zmienna **koniec** nie jest już nam więcej potrzebna. Kazaliśmy odświeżać okno po postawieniu figury, a nie po kliknięciu gdziekolwiek myszą. Jednak wszystko do tej pory można nazwać jedynie kosmetyką, gdyż największą zmianę uczyniło dodanie nowego zasobu – menu. Przypatrz się jego budowie uważnie, a zauważysz hierarchiczną strukturę. Każdej opcji odpowiada pewna stała, której wartość określa się liczbą całkowitą od 100 w górę, jak to ma miejsce w pliku „menu1.h". Ważne, aby ostatnią linię w tym pliku zostawić pustą. Po wybraniu z menu jakiejś opcji, do okna wysyłany jest komunikat WM_COMMAND. W dolnym słowie zmiennej **wParam** znajduje się liczba i właśnie nią musieliśmy się zająć. Dla odróżnienia wszystkie stałe należące do menu zaczynają się od IDM. Dodałem też krótką informację o autorze. Przypominam, że w systemie Windows można wybierać opcje menu z klawiatury (Alt+podkreślona litera, a potem strzałki lub znów podkreślona litera). Aby zaznaczyć literę w kodzie, przed jej znakiem stawiamy &, pamiętając oczywiście aby nie zaznaczać tej samej litery w różnych opcjach będących na jednej liście. Natąpiła też drobna zmiana w funkcji **WinMain**. Pole **lpszMenuName** ma przypisaną nazwę menu, która została ustalona w pliku „zasoby2.rc" przed słowem MENU.

17. AKCELERATORY

Myślę, że od tej pory będzie już z górki. Dodamy dwa klawisze skrótów zwane akceleratorami. Klawisz **F2** będzie rozpoczynał grę od nowa, a kombinacja **Ctrl-X** spowoduje wyjście z gry.

Przykład 14

plik „zasoby3.rc"

```
#include <windows.h>
#include "menu1.h"

MOJEMENU MENU
{
      POPUP "&Gra"
      {
            MENUITEM "&Nowa\tF2", IDM_NOWA
            MENUITEM "&Wyj•cie\tCtrl-X", IDM_WYJSCIE
      }
      POPUP "&Pomoc"
      {
            MENUITEM "&O grze...", IDM_O_GRZE
      }
}
MOJEMENU ACCELERATORS
{
      VK_F2, IDM_NOWA, VIRTKEY
      "^X" , IDM_WYJSCIE
}
KRZYZYK BITMAP KRZYZYK.BMP
KOLKO   BITMAP KOLKO.BMP
```

plik „ap14.c (fragment)

```
//--------------------
// pliki nag•ówkowe
```

```
//--------------------
#include <windows.h>
#include <stdio.h>
#include "menu1.h"
//--------------------
// definicje sta•ych
//--------------------
#define PUSTE 0
#define KOLO  1
#define KRZYZ 2
#define TAK 1
#define NIE 0
//--------------------
// prototypy funkcji
//--------------------
void SprawdzStan(HWND okno);
void Pytanie(HWND okno);
void NowaGra(HWND okno);
BOOL SprawdzKola();
BOOL SprawdzKrzyze();
BOOL SprawdzRemis();
LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);
//--------------------
// definicje zmiennych
//--------------------
char szNazwaOkna[] = "Moje okno";
char napis[255];
int stan[3][3];
static BOOL Gracz1;
HBITMAP hBit1, hBit2;
//--------------------
//      WinMain
//--------------------
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszArgs, int nWinMode)
{
      HWND hwnd;
      MSG msg;
      WNDCLASSEX wc;
      HACCEL hAccel;

      wc.hInstance =  hInstance;
      wc.lpszClassName = szNazwaOkna;
      wc.lpfnWndProc = ProcOkna;
      wc.style = CS_HREDRAW | CS_VREDRAW;
      wc.cbSize = sizeof(WNDCLASSEX);
      wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // du•a ikona
      wc.hIconSm = LoadIcon(NULL, IDI_WINLOGO);  // ma•a ikona
      wc.hCursor = LoadCursor(NULL,IDC_ARROW); // styl kursora
      wc.lpszMenuName = "MOJEMENU"; // nazwa menu
      wc.cbClsExtra=0;
      wc.cbWndExtra=0;
      wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH); // t•o na
bia•o

      // rejestracja klasy okna
      if (!RegisterClassEx(&wc))
```

```
         {
            MessageBox(NULL, "Nieudana rejestracja klasy
            okna","B••d",MB_OK|MB_ICONEXCLAMATION);
            return 0;
      }

      hwnd = CreateWindow(
            szNazwaOkna,             // nazwa okna
            "Kó•ko i krzy•yk", // tytu•
            WS_OVERLAPPEDWINDOW,  // styl - normalny
            20, // pocz•tkowe x
            20, // pocz•tkowe y
            400, // szeroko••
            400, // wysoko••
            HWND_DESKTOP, // brak okien-rodziców
            NULL,         // brak menu
            hInstance,    // uchwyt instancji
            NULL          // brak dodat. argumentów
);
      hAccel = LoadAccelerators(hInstance, "MOJEMENU");

      hBit1 = LoadBitmap(hInstance,"KRZYZYK");
      hBit2 = LoadBitmap(hInstance,"KOLKO");

      ShowWindow(hwnd, nWinMode);   // wy•wietlenie okna
      UpdateWindow(hwnd);

      // p•tla komunikatów
      while(GetMessage(&msg, NULL, 0, 0))
      {
            if (!TranslateAccelerator(hwnd,hAccel,&msg))
            {
                  TranslateMessage(&msg);
                  DispatchMessage(&msg);
            }
      }
      return msg.wParam;
}
// reszta taka sama jak w pliku ap13.c
```

Aby wprowadzić do programu klawisze skrótów, potrzebne były trzy rzeczy. Pierwszą stanowiło przypisanie stałym odpowiednich klawiszy w pliku zasobów („zasoby3.rc"). Zwykłe klawisze podawane są w cudzysłowiu, zaś funkcyjne zaczynają się od VK i na końcu potrzebują słowa VIRTKEY. Drugą zmianą było stworzenie uchwytu **hAccel** i użycie funkcji **LoadAccelerators** na podobnej zasadzie do ładowania bitmap. Trzecia konieczna modyfikacja polega na dodaniu funkcji **TranslateAccelerator**, która przetwarza wszystkie naciśnięte klawisze i sprawdza, czy nie są klawiszami skrótów do którejś z komend menu. Aby kod był czytelniejszy i bardziej estetyczny, dodałem komentarze dzielące go na pewne „tematyczne" części.

18. WŁASNA IKONA

Spróbujmy stworzyć własną ikonę. W tym celu konieczne będzie jej przygotowanie oraz drobna modyfikacja kodu. Najpierw zajmiemy się tym pierwszym. Włącz VC++, a następnie kliknij na **New** -> **Files** -> **Icon File**. Otworzy się wbudowany edytor ikon. Narysuj

w nim jakieś linie, parę krzyżyków i kółek w odpowiednich kolorach, po czym zapisz plik pod nazwą „ikona.ico”. Przegraj go do katalogu z projektem. Następnie stwórz następujące pliki.

Przykład 15

plik „menu2.h”

```
#define IDM_NOWA     100
#define IDM_WYJSCIE  101
#define IDM_O_GRZE   102

#define IDI_IKONA    200
```

plik „zasoby4.rc”

```
#include <windows.h>
#include "menu1.h"

MOJEMENU MENU
{
      POPUP "&Gra"
      {
            MENUITEM "&Nowa\tF2", IDM_NOWA
            MENUITEM "&Wyj•cie\tCtrl-X", IDM_WYJSCIE
      }
      POPUP "&Pomoc"
      {
            MENUITEM "&O grze...", IDM_O_GRZE
      }
}

MOJEMENU ACCELERATORS
{
      VK_F2, IDM_NOWA, VIRTKEY
      "^X" , IDM_WYJSCIE
}


KRZYZYK BITMAP KRZYZYK.BMP
KOLKO   BITMAP KOLKO.BMP

IDI_IKONA ICON IKONA.ICO
```

plik „ap15.c” (fragment)

```
//--------------------
// pliki nag•ówkowe
//--------------------
#include <windows.h>
#include <stdio.h>
#include "menu1.h"
//--------------------
// definicje sta•ych
//--------------------
#define PUSTE 0
#define KOLO  1
#define KRZYZ 2
#define TAK 1
#define NIE 0
//--------------------
```

```
// prototypy funkcji
//--------------------
void SprawdzStan(HWND okno);
void Pytanie(HWND okno);
void NowaGra(HWND okno);
BOOL SprawdzKola();
BOOL SprawdzKrzyze();
BOOL SprawdzRemis();
LRESULT CALLBACK ProcOkna(HWND, UINT, WPARAM, LPARAM);
//--------------------
// definicje zmiennych
//--------------------
char szNazwaOkna[] = "Moje okno";
char napis[255];
int stan[3][3];
static BOOL Gracz1;
HBITMAP hBit1, hBit2;
//--------------------
//      WinMain
//--------------------
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           LPSTR lpszArgs, int nWinMode)
{
      HWND hwnd;
      MSG msg;
      WNDCLASSEX wc;
      HACCEL hAccel;

      wc.hInstance =  hInstance;
      wc.lpszClassName = szNazwaOkna;
      wc.lpfnWndProc = ProcOkna;
      wc.style = CS_HREDRAW | CS_VREDRAW;
      wc.cbSize = sizeof(WNDCLASSEX);
      wc.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_IKONA)); // du•a
ikona
      wc.hIconSm = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_IKONA));  //
ma•a ikona
      wc.hCursor = LoadCursor(NULL,IDC_ARROW); // styl kursora
      wc.lpszMenuName = "MOJEMENU"; // nazwa menu
      wc.cbClsExtra=0;
      wc.cbWndExtra=0;
      wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH); // t•o na
bia•o

      // rejestracja klasy okna
      if (!RegisterClassEx(&wc))
      {
            MessageBox(NULL, "Nieudana rejestracja klasy
okna","B••d",MB_OK|MB_ICONEXCLAMATION);
            return 0;
      }

    hwnd = CreateWindow(
            szNazwaOkna,            // nazwa okna
            "Kó•ko i krzy•yk", // tytu•
            WS_OVERLAPPEDWINDOW,  // styl - normalny
            20, // pocz¦tkowe x
```

```
            20, // pocz¦tkowe y
            400, // szeroko••
            400, // wysoko••
            HWND_DESKTOP, // brak okien-rodziców
            NULL,         // brak menu
            hInstance,    // uchwyt instancji
            NULL          // brak dodat. argumentów
);
      hAccel = LoadAccelerators(hInstance, "MOJEMENU");

      hBit1 = LoadBitmap(hInstance,"KRZYZYK");
      hBit2 = LoadBitmap(hInstance,"KOLKO");

      ShowWindow(hwnd, nWinMode);   // wy•wietlenie okna
      UpdateWindow(hwnd);

      // p•tla komunikatów
      while(GetMessage(&msg, NULL, 0, 0))
      {
            if (!TranslateAccelerator(hwnd,hAccel,&msg))
            {
                  TranslateMessage(&msg);
                  DispatchMessage(&msg);
            }
      }
      return msg.wParam;
}
// reszta taka sama jak w pliku ap13.c
```

W kodzie zmieniło się dość niewiele. W pliku nagłówkowym „menu2.h" dodaliśmy jeszcze jeden identyfikator, któremu w „zasoby4.rc" przypisaliśmy nazwę naszej ikony. W funkcji **WinMain** zmieniliśmy pola **hIcon** i **hIconSm**. Funkcja **LoadIcon** zwraca uchwyt do ikony, jednak jej drugim argumentem musi być wskaźnik do łańcucha. My zaś dysponujemy stałą IDI_IKONA, którą stanowi liczba całkowita (200). Dlatego też użyliśmy makro MAKEINTRESOURCE, które nam zamieni liczbę na ciąg znaków z zasobu. Pamiętajmy, że po kompilacji programu ikona znajduje się wewnątrz pliku .exe . Tak więc dając komuś nasz program wystarczy przekazać jedynie plik wykonywalny skompilowany z opcją **Release** oraz dowolnie zmienić nazwę (np. na **kółko.exe**).

19. ZAKOŃCZENIE

W tym miejscu dobrnęliśmy do końca przygody z programowaniem w Windows 95. Oczywiście nie powiedziałem o wielu rzeczach takich jak okna dialogowe, różne kontrolki, wielozadaniowość itp. Jednak mam nadzieję, że tekst ten będzie stanowił pewne wprowadzenie do własnych eksperymentów. Programując nie trzeba wszystkiego rozumieć (do momentu gdy wszystko działa ;). Ktoś może spytać skąd nazwa projektu „API". Otóż zbiór większości użytych w przykładach funkcji stanowi tzw. interfejs programowania aplikacji (Application Programming Interface – w skrócie API). Omawiane przeze mnie API ma nazwę **Win32** i obsługuje 32-bitowe aplikacje. Inne API stanowi biblioteka klas firmy Microsoft o nazwie **MFC**. Ułatwia ona i nieraz przyspiesza proces tworzenia aplikacji, jednak programy zajmują więcej i chodzą nieco wolniej. Jeśli pasjonuje Cię programowanie, możesz wybrać jedną z wielu dróg dalszego kształcenia. Poniżej znajdziesz kilka moich propozycji.

1. Dalej poznawać „czyste" API, tworząc coraz bardziej złożone programy.
2. Zająć się biblioteką klas **MFC** (wymagana znajomość języka C++).
3. Rozpocząć przygodę z bibliotekami graficznymi takimi jak **DirectX** czy **OpenGL**.

Ja już żegnam, życząc powodzenia i wiele satysfakcji z dalszej pracy oraz zdobywanej wiedzy. Przyjemnego grania w „Kółko i krzyżyk" !