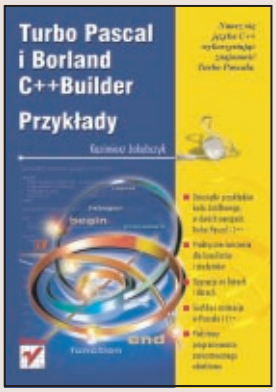
Turbo Pascal i Borland C++Builder. Przykłady



Autor: Kazimierz Jakubczyk
ISBN: 83-7197-873-1
Format: B5, stron: 234
Zawiera dyskietkę

Książka ta adresowana jest do czytelników, którzy rozpoczęli lub właśnie rozpoczynają swoją przygodę z programowaniem. Zawiera ona szereg przykładowych programów napisanych zgodnie z zasadami dobrego stylu programowania. Wszystkie stworzone zostały w dwóch wersjach: w Turbo Pascalu i C++. Dlatego książka ta jest szczególnie polecana osobom, które już potrafią programować w Pascalu i chcą zapoznać się z językiem C++.

W książce omówiono:

- Proste operacje na liczbach
- Działania na datach
- Tworzenie grafiki z wykorzystaniem BGI
- Animacje
- Tworzenie list jednokierunkowych
- Programowanie zorientowane obiektowo

Wymagania stawiane czytelnikowi są niewielkie – wystarczy podstawowa umiejętność programowania, najlepiej w Pascalu.

Do książki dołączona jest dyskietka z przykładami.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 3.

Grafika

Jednym ze sposobów tworzenia grafiki na ekranie monitora komputerowego jest składanie obrazu z podstawowych elementów graficznych, takich jak punkty, odcinki proste, wielokąty i okręgi, oraz wypełnianie obszarów zadanym kolorem lub wzorcem. Wszystkie programy zawarte w tym rozdziale działają na tej zasadzie, korzystają z biblioteki *BGI* (ang. *Borland Graphics Interface* — interfejs graficzny firmy Borland) i pracują w 16-kolorowym trybie graficznym VGA o rozdzielczości ekranu 640×480 pikseli. Dotyczą one następujących zagadnień:

- rysowanie trójkąta Sierpińskiego (gra w chaos),
- kreślenie wielokąta foremnego i gwiazdki,
- wyznaczanie najmniejszego wielokąta wypukłego zawierającego dany zbiór punktów,
- zliczanie liter w pliku tekstowym i zobrazowanie wyniku w formie wykresu słupkowego,
- tworzenie wykresu funkcji drgań harmonicznych tłumionych.

Gra w chaos

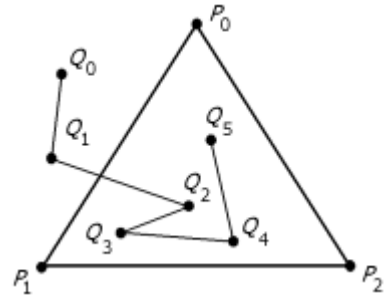
Niekiedy bardzo proste algorytmy prowadzą do zaskakujących wyników. Jednym z nich jest zaprezentowany poniżej algorytm generowania obrazu, zwany *grą w chaos*[1]. Rysunek utworzony na ekranie monitora zaskakuje szczególnie, ponieważ całe postępowanie jest oparte na *losowości*, czyli niemożności przewidywania, chaosie.

[1] Właściwie jest to jedna z gier polegających na losowym wyborze jednej z kilku prostych reguł postępowania. Przykład zaczerpnięto z książki [8].

Algorytm gry w chaos

Algorytm gry w chaos jest rzeczywiście bardzo prosty. Na początku wybieramy na ekranie trzy punkty P_0, P_1 i P_2 tak, by stanowiły wierzchołki trójkąta, np. równobocznego, oraz dowolny punkt Q_0. Jest to tzw. *punkt wiodący*. Następnie losujemy jeden z wierzchołków trójkąta i w środku odcinka łączącego ten wierzchołek z punktem Q_0 zaznaczamy nowy punkt wiodący Q_1. Ponownie losujemy wierzchołek trójkąta i w środku odcinka o końcach w wylosowanym wierzchołku i punkcie Q_1 zaznaczamy kolejny punkt wiodący Q_2. Losowanie wierzchołka i zaznaczanie następnego punktu wiodącego dokładnie w połowie odcinka łączącego wylosowany wierzchołek z poprzednim punktem wiodącym powtarzamy wiele razy. Na rysunku 3.1 pokazano pięć pierwszych kroków, w których kolejnymi wylosowanymi wierzchołkami były P_1, P_2, P_1, P_2, P_0.

Rysunek 3.1. *Pięć pierwszych kroków gry w chaos*



Jaki będzie rezultat wykonania dużej liczby powtórzeń wyznaczania następnego punktu wiodącego i rysowania go na ekranie? Wydaje się oczywiste, że jeśli punkt wiodący Q_0 został wybrany na zewnątrz trójkąta $P_0P_1P_2$, to po niewielkiej liczbie kroków kolejny wyznaczony punkt wiodący dostanie się do wnętrza tego trójkąta. Ponadto gdy jakiś punkt wiodący znajdzie się we wnętrzu trójkąta, każdy następny tam pozostanie. Można więc przypuszczać, że po dużej liczbie iteracji punkty wiodące zapełnią losowo wnętrze trójkąta. Aby się przekonać, czy rzeczywiście tak będzie, wystarczy napisać i uruchomić prosty program.

Symulacja zdarzeń losowych na komputerze wymaga odpowiedniego podprogramu generującego *liczby losowe*. Należy jednak sobie uświadomić, że generator taki działa według ściśle określonego algorytmu, a dostarczone przez niego liczby jedynie sprawiają wrażenie losowości. Bardziej stosowne jest więc nazywanie ich *liczbami pseudolosowymi*.

Program Chaos w Pascalu

W Turbo Pascalu liczby losowe generuje funkcja `Random`, którą można wywoływać bez parametru lub z jednym parametrem reprezentującym wartość całkowitą dodatnią. Wartością wywołania funkcji w pierwszym przypadku jest losowa liczba rzeczywista z przedziału [0, 1), zaś w drugim losowaniu — nieujemna liczba całkowita mniejsza od zadanego parametru, np. wartością wywołania `Random(49)` jest liczba całkowita od 0 do 48. Iteracyjne użycie funkcji `Random` daje ciąg liczb o rozkładzie jednostajnym.

Zazwyczaj przed wykorzystaniem generatora inicjuje się go za pomocą bezparametrowej procedury `Randomize`. Generator niezainicjowany tworzy taki sam ciąg liczb za każdym uruchomieniem programu, co można wykorzystać podczas testowania programu. Pełny kod źródłowy programu w Turbo Pascalu realizującego grę w chaos jest przedstawiony na wydruku 3.1.

Wydruk 3.1. *Program Chaos.pas realizujący grę w chaos*

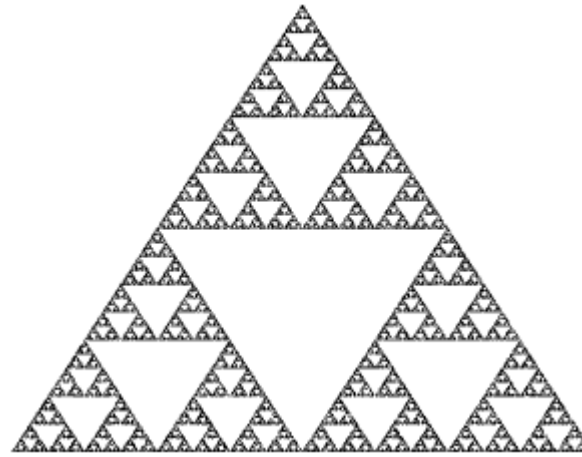
```
program Gra_w_chaos;

uses Crt, Graph;

const
  Karta: integer = VGA;
  Rozdz: integer = VGAHi;

  x: array[0..2] of integer = (320,  30, 610);
  y: array[0..2] of integer = ( 15, 460, 460);

var
  a, b, w: integer;
  k      : longint;

begin
  Randomize;
  InitGraph(Karta, Rozdz, 'C:\TP\BGI');
  a := Random(640);
  b := Random(480);
  for k:=-10 to 50000 do
  begin
    w := Random(3);
    a := (a + x[w]) div 2;
    b := (b + y[w]) div 2;
    if k>0 then PutPixel(a, b, Cyan);
  end;
  ReadKey;
  CloseGraph;
end.
```

Tablice inicjalizowane *x* i *y* zawierają współrzędne ekranowe wierzchołków trójkąta, zmienne *a* i *b* — współrzędne bieżącego punktu wiodącego, *w* — indeks wylosowanego wierzchołka, a *k* — numer kolejny iteracji. Program (bez wejścia) inicjuje generator liczb losowych, przełącza kartę graficzną w tryb graficzny `VGA` o rozdzielczości `VGAHi` (640×480 pikseli)[2], rysuje 50 000 punktów wiodących w kolorze `Cyan` (turkusowy) za pomocą procedury `PutPixel`, a po wybraniu przez użytkownika dowolnego znaku na klawiaturze przełącza kartę graficzną w tryb tekstowy i kończy działanie. Faktycznie program wykonuje iterację ponad 50 000 razy. Dodatkowe 11 początkowych kroków, w których punkt wiodący nie jest rysowany, ma na celu pominięcie punktów nienależących do wnętrza trójkąta.

[2] Ścieżka dostępu podana w wywołaniu procedury `InitGraph`, określająca miejsce sterownika karty graficznej (plik *egavga.bgi*), może być inna w przypadku innego komputera.

Wynik utworzony przez program na ekranie monitora jest pokazany na rysunku 3.2. Uświadamia on, jak zawodna może być intuicja. Wygenerowany obraz przedstawia tzw. *trójkąt Sierpińskiego*. Jest on zbiorem punktów wyjątkowo uporządkowanym, niemającym — wydawałoby się — nic wspólnego z chaosem i losowością.

Rysunek 3.2. *Trójkąt Sierpińskiego po 50 000 krokach gry w chaos*



Program Chaos w C++

W środowisku Borland C++ jest również dostępny generator liczb losowych. Realizują go, podobnie jak w Pascalu, funkcje `random` i `randomize`, z tym że pierwsza istnieje tylko w wersji dla liczb całkowitych. Ich prototypy znajdują się w pliku *stdlib.h*. Pełny program źródłowy w Borland C++, tworzący trójkąt Sierpińskiego metodą gry w chaos, jest przedstawiony na wydruku 3.2.

Wydruk 3.2. *Program Chaos.cpp realizujący grę w chaos*

```
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>

int Karta = VGA,  Rozdz = VGAHI,

    x[] = {320, 30, 610},  y[] = {15, 460, 460};

void main()
{
   int a, b, w;
   long k;
   randomize();
   initgraph(&Karta, &Rozdz, "C:\\BC\\BGI");
   a = random(640);
   b = random(480);
   for (k=-10; k<=50000; k++)
   {
      a = (a + x[w = random(3)])/2;
      b = (b + y[w])/2;
```

```
        if (k>0) putpixel(a, b, CYAN);
    }
    getch();
    closegraph();
}
```

Podwójne znaki \\ (*Backslash*) w tekście określającym ścieżkę dostępu do sterownika karty graficznej w wywołaniu funkcji `initgraph` wynikają z faktu, iż \ jest znakiem specjalnym. Każda sekwencja \\ reprezentuje w C i C++ pojedynczy znak \.

Wielokąt foremny i gwiazdka

Zadanie rysowania łamanej, a w szczególności wielokąta, pojawia się w wielu praktycznych zagadnieniach grafiki komputerowej. Na przykład proces rysowania krzywej, nawet bardzo złożonej, sprowadza się do narysowania łamanej o odpowiednio dużej liczbie wierzchołków leżących na tej krzywej. Technika rysowania łamanej o wierzchołkach $P_0, P_1, ..., P_n$ polega na przeniesieniu pisaka (pióra) do wierzchołka początkowego P_0, a następnie wykonaniu pętli rysującej kolejne odcinki $P_{k-1}P_k$ dla $k = 1,2,...,n$. Tak samo postępuje się w przypadku wielokąta, który jest łamaną zamkniętą: $P_0 = P_n$.

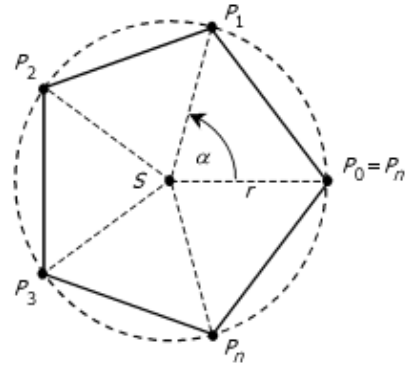
Podstawowymi narzędziami do kreślenia są:

- procedura ustawienia pisaka w dowolnym punkcie ekranu,
- procedura kreślenia odcinka od aktualnej pozycji pisaka do jego nowej pozycji.

Wyznaczanie wierzchołków wielokąta foremnego

Wierzchołki wielokąta foremnego są równomiernie rozłożone na okręgu (por. rysunek 3.3), a ich współrzędne spełniają równania parametryczne:

$$x = r \cdot \cos\varphi + p, \quad y = r \cdot \sin\varphi + q, \quad 0 \le \varphi \le 2\pi$$



Rysunek 3.3. *Wielokąt foremny wpisany w okrąg*

gdzie r jest promieniem, a p i q współrzędnymi środka S tego okręgu.

Niech α oznacza kąt między promieniami łączącymi punkt S z dowolnymi dwoma sąsiednimi wierzchołkami:

$$\alpha = \frac{2\pi}{n}$$

Jeżeli promień okręgu łączący punkty S i P_0 jest równoległy do osi y, to współrzędne wierzchołków $P_k = (x_k, y_k)$ dla $k = 0,1,...,n$ można wyznaczyć, podstawiając do powyższych równań parametrycznych w miejsce parametru φ wartości:

$$\varphi_k = k \cdot \alpha$$

czyli korzystając ze wzorów:

$$x_k = r \cdot \cos\varphi_k + p, \quad y_k = r \cdot \sin\varphi_k + q$$

Rysowanie wielokąta foremnego w Pascalu

W Turbo Pascalu procedura `MoveTo` ustawia pisak w określonym punkcie ekranu, zaś `LineTo` rysuje odcinek od bieżącej pozycji pisaka do jego nowej pozycji. Ich parametrami są liczby całkowite określające współrzędne ekranowe punktu, do którego pisak ma być przesunięty. Zatem proces rysowania wielokąta foremnego można w tym języku zaprogramować następująco:

```
procedure Wielokat(n, p, q: integer; r: real);
var
  k: integer;
  alfa: real;
begin
  alfa := 2*Pi/n;
  MoveTo(Round(r)+p, q);
  for k:=1 to n do
    LineTo(Round(r*cos(k*alfa))+p, q-Round(r*sin(k*alfa)));
end;
```

Zmiana znaku (minus zamiast plus) w wyrażeniu występującym w roli drugiego parametru w wywołaniu procedury `LineTo` wynika z dziwnego założenia przyjętego przez projektantów komputera i twórców środowiska programowania, iż początek układu współrzędnych znajduje się w lewym górnym rogu ekranu, a oś y jest skierowana w dół[3]. Funkcja `Round` została wykorzystana w celu zaokrąglenia wartości rzeczywistej do najbliższej wartości całkowitej, użycie typu rzeczywistego spowodowałoby błąd (*type mismatch* — niezgodność typu). Równie dobrze można tu wykorzystać funkcję `Trunc`, która obcina wartość rzeczywistą do części całkowitej.

[3] W tym przypadku zmiana znaku nie ma znaczenia z uwagi na symetrię wielokąta względem prostej równoległej do osi x i przechodzącej przez jego środek S.

Rysowanie wielokąta foremnego w C++

Języki C i C++ nie są tak rygorystyczne. Zawierają szereg mechanizmów automatycznego przekształcania (konwersji) typów, które są wykorzystywane, gdy jest to konieczne i ma sens. Ogólnie mówiąc, są one bardzo użyteczne i dają wyniki zgodnie z oczekiwaniami. Jednym z takich przekształceń jest zamiana wartości rzeczywistej na całkowitą, odpowiadająca pascalowskiej funkcji `Trunc`. Powyższą procedurę można więc w Borland C++ zapisać nieco prościej:

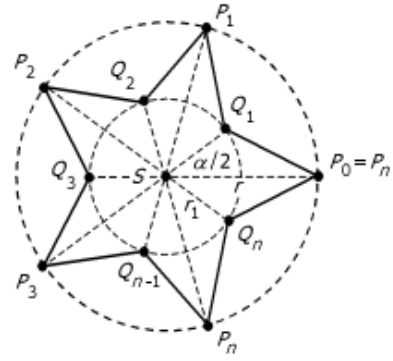
```
void wielokat(int n, int p, int q, double r)
{
   int k;
   double alfa = 2*M_PI/n;
   moveto(r+p, q);
   for (k=1; k<=n; k++)
      lineto(r*cos(k*alfa)+p, q-r*sin(k*alfa));
}
```

Oczywiście użycie symbolu `M_PI` (wartość π) w wyrażeniu inicjalizującym zmienną *alfa* oraz funkcji trygonometrycznych *sin* i *cos* wymaga włączenia pliku nagłówkowego *math.h*.

Wyznaczanie wierzchołków gwiazdki równoramiennej

Nietrudno teraz przejść do narysowania gwiazdki równoramiennej. Jej $2n$ wierzchołków leży przemiennie na dwóch koncentrycznych okręgach (por. rysunek 3.4).

Rysunek 3.4. *Gwiazdka równoramienna i dwa okręgi*



Niech $P_0, P_1, ...P_n = P_0$ będą wierzchołkami gwiazdki leżącymi, podobnie jak w przypadku wielokąta foremnego, na okręgu o promieniu r i środku $S = (p, q)$, niech również promień okręgu łączący punkty S i P_0 będzie równoległy do osi y. Natomiast niech $Q_1, Q_2, ..., Q_n$ będą wierzchołkami gwiazdki leżącymi na drugim okręgu o promieniu r_1 i środku S. Współrzędne punktów $Q_k = (u_k, v_k)$ dla $k = 1,2,...,n$ można także wyznaczyć z równań parametrycznych okręgu. Promienie, których końcami są punkty Q_k, są obrócone wokół środka S zgodnie z ruchem wskazówek zegara (kierunek ujemny) o kąt $\alpha/2$ względem promieni, których końcami są punkty P_k. Zatem spełnione są równości

$$u_k = r_1 \cos\psi_k + p, \quad v_k = r_1 \sin\psi_k + q$$

w których

$$\psi_k = \varphi_k - \frac{\alpha}{2} = \varphi_k - \frac{\pi}{n}$$

Narzucający się algorytm rysowania gwiazdki sprowadza się do przeniesienia pisaka do punktu P_0 i wykonania pętli, która dla $k = 1,2,...,n$ rysuje dwa odcinki $P_{k-1}Q_k$ i Q_kP_k zamiast jednego odcinka $P_{k-1}P_k$ w przypadku wielokąta foremnego.

Program Gwiazdka w Pascalu

Opisany wyżej algorytm rysowania gwiazdki jest wykorzystany w programie w Turbo Pascalu przedstawionym na wydruku 3.3.

Wydruk 3.3. *Program Gwiazdka.pas rysowania gwiazdki równoramiennej*

```
program Rysowanie_gwiazdki;

uses Crt, Graph;

const
  Karta: integer = VGA;
  Rozdz: integer = VGAHi;

procedure Gwiazdka(n, p, q: integer; r, r1: real);
var
  k: integer;
  alfa, alfa2, fi, psi: real;
begin
  alfa2 := Pi/n;
  alfa := 2*alfa2;
  MoveTo(Round(r)+p, q);
  for k:=1 to n do
  begin
    fi := k*alfa;
    psi := fi-alfa2;
    LineTo(Round(r1*cos(psi))+p, q-Round(r1*sin(psi)));
    LineTo(Round(r*cos(fi))+p, q-Round(r*sin(fi)));
  end;
  FloodFill(p, q, GetColor);
end;

var
  n: integer;
  r: real;

begin
  Write('Liczba ramion: ');
  ReadLn(n);
  Write('Promień      : ');
  ReadLn(r);
  InitGraph(Karta, Rozdz, 'C:\TP\BGI');
```

```
  SetColor(LightRed);
  SetFillStyle(SolidFill, Yellow);
  Gwiazdka(n, 320, 240, r, r/2);
  ReadKey;
  CloseGraph;
end.
```

Program wczytuje z klawiatury liczbę *n* ramion gwiazdki i promień *r* okręgu określającego jej wielkość, przełącza kartę graficzną w tryb graficzny, ustawia kolor rysowania linii (`LightRed` — jasnoczerwony) i styl wypełniania obszarów (`SolidFill` — pełny, kolor `Yellow` — żółty). Następnie rysuje pośrodku ekranu gwiazdkę o $2n$ wierzchołkach leżących na przemian na okręgach o promieniach *r* i *r*/2, a w końcu zapełnia ją, poczynając od środka. Gdy użytkownik wybierze dowolny znak na klawiaturze, program powraca do trybu tekstowego i kończy działanie.

Program Gwiazdka w C++

Analogiczny program w języku Borland C++, wykorzystujący inicjalizowanie zmiennych lokalnych wyrażeniami zbudowanymi z uprzednio określonych wartości oraz umieszczanie przypisań w wyrażeniach (funkcja *gwiazdka*), został przedstawiony na wydruku 3.4.

Wydruk 3.4. *Program Gwiazdka.cpp rysowania gwiazdki równoramiennej*

```
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

int Karta = VGA,  Rozdz = VGAHI;

void gwiazdka(int n, int p, int q, double r, double r1)
{
   int k;
   double alfa2 = M_PI/n, alfa = 2*alfa2, fi, psi;
   moveto(r+p, q);
   for (k=1; k<=n; k++)
   {
      psi = (fi = k*alfa)-alfa2;
      lineto(r1*cos(psi) + p, q - r1*sin(psi));
      lineto(r*cos(fi) + p, q - r*sin(fi));
   }
   floodfill(p, q, getcolor());
}

void main()
{
   int n;
   float r;
   cout << "Liczba ramion: ";
   cin >> n;
   cout << "Promień      : ";
   cin >> r;
```

```
    initgraph(&Karta, &Rozdz, "C:\\BC\\BGI");
    setcolor(LIGHTRED);
    setfillstyle(SOLID_FILL, YELLOW);
    gwiazdka(n, 320, 240, r, r/2);
    getch();
    closegraph();
}
```

Zazwyczaj przejście od środowiska Turbo Pascala do Borland C++ nie sprawia kłopotów, gdy wykorzystuje się równoważne biblioteki standardowe. Nazwy zdefiniowanych w bibliotekach stałych i podprogramów są w obu środowiskach takie same, z tym że w C++ pierwsze pisane są dużymi, a drugie małymi literami. Zdarzają się jednak niespodzianki, czego przykładem są stałe `SolidFill` i `SOLID_FILL` (pełne wypełnianie obszarów) użyte w dwóch ostatnich programach (por. wydruki 3.3 i 3.4).

Najmniejszy wielokąt wypukły

Zbiór punktów, który dla dowolnych należących do niego dwóch punktów A i B zawiera łączący je odcinek AB, nazywamy *zbiorem wypukłym*. Najmniejszy zbiór wypukły zawierający skończoną liczbę punktów P_1, P_2,..., P_n jest wielokątem, a zbiór jego wierzchołków stanowi podzbiór tego zbioru n punktów. Nie zajmiemy się tu jednak prezentacją znanych algorytmów wyznaczania najmniejszego wielokąta wypukłego[4], lecz ograniczymy się do jego narysowania. Metoda polegać będzie na wyszukiwaniu i rysowaniu tylko tych odcinków P_iP_j dla $1 \le i < j \le n$, które leżą na brzegu wielokąta. W rezultacie wierzchołki najmniejszego wielokąta wypukłego zostaną wyznaczone. Nasuwa się więc pytanie: na czym polega wada takiego postępowania? Otóż istotne jest również określenie kolejności wierzchołków. Brak uporządkowania wierzchołków wielokąta może w przypadku współliniowości trzech lub większej ich liczby prowadzić do ponownego rysowania narysowanych wcześniej fragmentów brzegu.

Algorytm wyznaczania brzegu najmniejszego wielokąta wypukłego

Przejdźmy teraz do sformułowania algorytmu umożliwiającego sprawdzenie, czy odcinek P_iP_j określony przez dwa dowolne punkty P_i i P_j, wybrane spośród punktów $P_k = (x_k, y_k)$ dla $k = 1,2,...,n$, leży na brzegu najmniejszego wielokąta wypukłego zawierającego wszystkie te punkty. Oczywiste jest, że odcinek P_iP_j ma taką właściwość wtedy i tylko wtedy, gdy wszystkie one leżą po jednej stronie prostej przechodzącej przez końce tego odcinka lub na niej. Rozpatrzmy przypadek $x_i \neq x_j$. Prosta spełnia wówczas równanie

[4] Można je znaleźć np. w książce [2].

$$y = p(x) = \frac{y_j - y_i}{x_j - x_i}(x - x_i) + y_i$$

Położenie punktu (x_k, y_k) względem prostej zależy od znaku wyrażenia $v = p(x_k) - y_k$, mianowicie punkt leży na prostej, gdy $v = 0$, nad prostą, gdy $v < 0$, albo pod nią, gdy $v > 0$. Zatem sprawdzenie, czy punkty o indeksach i, j określają odcinek leżący na brzegu wielokąta, można opisać za pomocą następującej funkcji w Turbo Pascalu:

```
function Brzeg(i,j: integer): Boolean;
var
  k, s, z: integer;
  v: real;
begin
  Brzeg := false;
  s := 0;
  for k:=1 to n do
    if (k<>i) and (k<>j) then
    begin
      v := (y[j]-y[i])/(x[j]-x[i])*(x[k]-x[i])+y[i]-y[k];
      if v<>0 then
      begin
        if v>0 then z := 1 else z := -1;
        if s=0 then s := z else if s<>z then Exit;
      end;
    end;
  Brzeg := true;
end;
```

Pętla `for` przebiega kolejno punkty i bada ich położenie względem prostej wyznaczonej przez punkty o indeksach i i j. Istotne działania są podejmowane, gdy rozpatrywany punkt nie leży na prostej ($v \neq 0$). Wtedy zmiennej z przypisana zostaje wartość 1, gdy punkt leży poniżej prostej, albo –1, gdy powyżej. Dalsze postępowanie zależy od wartości zmiennej s. Wartość zerowa s oznacza, że napotkany punkt jest pierwszym, który nie leży na prostej, a wtedy jego położenie względem niej (liczba 1 lub –1) zostaje zapamiętane w zmiennej s. Z kolei niezerowa wartość s określa położenie wszystkich dotąd uwzględnionych punktów. Jeśli jest ona różna od wartości z, leżą one po innej stronie prostej niż rozpatrywany punkt. W takim przypadku pętla zostaje przerwana, wykonanie podprogramu zakończone, a rezultatem wywołania funkcji jest wartość `false` oznaczająca, że odcinek wyznaczony przez punkty P_i i P_j nie należy do brzegu wielokąta. Jeżeli wszystkie punkty leżą po jednej stronie prostej lub na niej, pętla zostaje zakończona w sposób naturalny, a do miejsca wywołania funkcji zostaje przekazana wartość `true` oznaczająca, że odcinek P_iP_j stanowi fragment brzegu wielokąta.

Funkcja *Brzeg* wymaga udoskonalenia, ponieważ nie zadziała prawidłowo w przypadku, gdy punkty P_i i P_j wyznaczają prostą pionową, tj. gdy $x_i = x$ (ang. *divide by zero* — dzielenie przez zero). Można temu zaradzić, wyliczając wartości $dx = x_i - x_j$ i $dy = y_i - y_j$, które posłużą do wybrania jednego z dwóch równań prostej, w zależności od jej kąta nachylenia względem osi x. I tak, dla $|dx| \geq |dy|$, tj. gdy nachylenie prostej nie przekracza kąta 45^o, korzystamy ze wzoru

$$y = \frac{dy}{dx}(x - x_i) + y_i$$

natomiast dla $|dx| < |dy|$, czyli dla nachyleń większych, traktujemy zmienną *x* jako zależną od zmiennej *y*, co prowadzi do wzoru

$$y = \frac{dy}{dx}(x - x_i) + y_i$$

Program WielWyp w Pascalu

Zmodyfikowana funkcja *Brzeg* jest wykorzystana w przedstawionym na wydruku 3.5 programie w Turbo Pascalu, który wczytuje współrzędne punktów z pliku tekstowego, wykreśla te punkty w postaci małych krzyżyków i rysuje najmniejszy zawierający je wielokąt wypukły.

Wydruk 3.5. *Program WielWyp.pas rysowania najmniejszego wielokąta wypukłego*

```
program Wielokat_wypukly;

uses Crt,Graph;

const
  Karta: integer = VGA;
  Rozdz: integer = VGAHi;

var
  n: integer;
  x,y: array[1..1000] of integer;

procedure CzytajDane;
var
  Dane: Text;
  Nazwa: string;
begin
  Write('Nazwa pliku: ');
  ReadLn(Nazwa);
  Assign(Dane, Nazwa);
  Reset(Dane);
  n := 0;
  while not Eof(Dane) do
  begin
    Inc(n);
    ReadLn(Dane, x[n], y[n]);
  end;
  Close(Dane);
end;

function Brzeg(i,j: integer): Boolean;
var
  k, dx, dy, s, z: integer;
  v: real;
begin
  Brzeg := false;
```

```
  dx := x[j]-x[i];
  dy := y[j]-y[i];
  if (dx=0) and (dy=0) then Exit;
  s := 0;
  for k:=1 to n do
    if (k<>i) and (k<>j) then
    begin
      if Abs(dx)>Abs(dy)
         then v := dy/dx*(x[k]-x[i])+y[i]-y[k]
         else v := dx/dy*(y[k]-y[i])+x[i]-x[k];
      if v<>0 then
      begin
        if v>0 then z := 1 else z := -1;
        if s=0 then s := z else if s<>z then Exit;
      end;
    end;
  Brzeg := true;
end;

procedure UtworzRysunek;
var
  i, j: integer;
begin
  for i:=1 to n do
  begin
    Line(x[i]-2, y[i]-2, x[i]+2, y[i]+2);
    Line(x[i]-2, y[i]+2, x[i]+2, y[i]-2);
  end;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if Brzeg(i,j) then Line(x[i], y[i], x[j], y[j]);
end;

begin
  CzytajDane;
  InitGraph(Karta, Rozdz, 'C:\TP\BGI');
  UtworzRysunek;
  ReadKey;
  CloseGraph;
end.
```

Nazwa pliku jest podawana z klawiatury. Przyjęto założenie, że plik zwiera co najwyżej 1000 wierszy, tj. $n \leq 1000$, oraz że wiersze pliku zawierają po dwie liczby całkowite, oddzielone co najmniej jedną spacją, określające współrzędne ekranowe kolejnego punktu:

$$0 \leq x_k < 640, \quad 0 \leq y_k < 480, \quad (k = 1,2,...,n)$$

Tworząc plik, np. w edytorze Turbo Pascala, należy uważać, aby po ostatnim wierszu nie umieścić wiersza pustego. Liczba punktów (wierszy) jest obliczana w pętli w trakcie czytania pliku według schematu:

podczas gdy (dopóki) nie napotkano jeszcze końca pliku, zwiększ liczbę punktów (wstępnie zero) o jeden i wczytaj współrzędne kolejnego punktu.

W funkcji *Brzeg* dodatkowo zabezpieczono się przed błędem w patologicznym przypadku, gdy parametry określają punkty o takich samych współrzędnych: dla $dx = dy = 0$ obliczenia są przerywane, wynikiem wywołania funkcji jest wartość `false`. Korzystający z funkcji *Brzeg* podprogram *UtworzRysunek* najpierw rysuje wszystkie punkty, a następnie sprawdza, czy punkty o indeksach $1 \le i < j \le n$ określają odcinki na brzegu najmniejszego wielokąta wypukłego, i w przypadku spełnienia warunku odcinki takie rysuje. Tym razem wygodniejszym narzędziem rysowania jest procedura `Line`, która wymaga czterech parametrów reprezentujących współrzędne początku i końca odcinka. W przeciwieństwie do `LineTo`, nie przenosi ona pisaka do końca rysowanego odcinka.

Program WielWyp w C++

Na wydruku 3.6 jest przedstawiony podobny program w Borland C++, ale o bardziej zwartym w porównaniu z Pascalem kodzie dotyczącym czytania danych i sprawdzania, czy odcinek określony przez dwa punkty leży na brzegu wielokąta wypukłego.

Wydruk 3.6. *Program WielWyp.cpp rysowania najmniejszego wielokąta wypukłego*

```
#include <stdio.h>
#include <math.h>
#include <graphics.h>
#include <conio.h>

int Karta = VGA, Rozdz = VGAHI, n, x[1000], y[1000];

int czytajdane()
{
   FILE *dane;
   char nazwa[256];
   printf("Nazwa pliku: ");
   gets(nazwa);
   if ((dane = fopen(nazwa, "rt")) != NULL)
   {
      for (n=0; fscanf(dane, "%d %d", x+n, y+n) != EOF; n++);
      fclose(dane);
      return 1;
   }
   else
      return 0;
}

int brzeg(int i, int j)
{
   int k, dx = x[j]-x[i], dy = y[j]-y[i], s = 0, z;
   double v;
   if (dx==0 && dy==0) return 0;
   for (k=0; k<n; k++)
      if (k != i && k != j)
         if ((v = (fabs(dx)>fabs(dy))
            ? double(dy)/dx*(x[k]-x[i])+y[i]-y[k]
            : double(dx)/dy*(y[k]-y[i])+x[i]-x[k]) != 0)
```

```
            {
               z = (v>0) ? 1 : -1;
               if (s==0) s = z; else if (s!=z) return 0;
            }
      return 1;
}

void utworzrysunek()
{
   int i, j;
   for (i=0; i<n; i++)
   {
      line(x[i]-2, y[i]-2, x[i]+2, y[i]+2);
      line(x[i]-2, y[i]+2, x[i]+2, y[i]-2);
   }
   for (i=0; i<n-1; i++)
      for (j=i+1; j<n; j++)
         if (brzeg(i,j)) line(x[i], y[i], x[j], y[j]);
}

void main()
{
   if (czytajdane())
   {
      initgraph(&Karta, &Rozdz, "C:\\BC\\BGI");
      utworzrysunek();
      getch();
      closegraph();
   }
   else
   {
      printf("Nieudane otwarcie pliku\n");
      getch();
   }
}
```

Program również rozpoczyna działanie od wczytania danych z pliku tekstowego, wykorzystując funkcję *czytajdane*. Jednakże w przypadku podania przez użytkownika niewłaściwej nazwy pliku, informuje o nieudanym otwarciu pliku, a nie kończy się błędem (ang. *file not found* — plik nieznaleziony). Parametrami funkcji `fopen` są nazwa pliku i tryb jego otwarcia (*r* oznacza odczyt, *t* — plik tekstowy). Wartością funkcji jest wskaźnik do struktury typu `FILE`, gdy plik został otwarty, bądź tzw. *wskaźnik pusty* `NULL`, gdy z różnych przyczyn pliku nie udało się otworzyć. Funkcja `fclose` zamyka plik.

Do wczytania dwóch współrzędnych kolejnego punktu użyto funkcji `fscanf`. Jej wywołanie, zamieszczone w warunku kontynuacji pętli `for`, daje wartość równą liczbie wczytanych pól, gdy operacja czytania powiodła się, bądź wartość `EOF` (ang. *end of file* — koniec pliku), gdy napotkano koniec pliku. Cztery parametry wywołania oznaczają kolejno: wskaźnik do pliku, format czytanych danych (symbole formatujące `%d` odnoszą się do liczb całkowitych dziesiętnych) i dwa wskaźniki określające elementy tablic, których wartości mają być wczytane.

Wskaźniki a tablice w C i C++

Wskaźnik do zmiennej jest zazwyczaj tworzony za pomocą operatora `&`, np. zapis `&n` oznacza wskaźnik do zmiennej *n*. Wyrażeniami określającymi wskaźniki do elementów `x[n]` i `y[n]` są zatem `&x[n]` i `&y[n]`, można je było zastosować w wywołaniu funkcji `fscanf`. Nazwa tablicy jest jednak traktowana jako wskaźnik do jej początkowego elementu (skutkiem tego parametrem wywołania funkcji `gets`, wczytującej nazwę pliku z klawiatury, jest nazwa tablicy znakowej). Ponadto poprzez dodanie wartości całkowitej do wskaźnika otrzymuje się wskaźnik do elementu przesuniętego o tę wartość. Zapisy `x+n` i `y+n` oraz `&x[n]` i `&y[n]` są więc całkowicie równoważne.

Ścisła odpowiedniość między tablicami i wskaźnikami przejawia się w dowolności używania indeksów i wskaźników w kodowaniu operacji dotyczących przetwarzania tablic. I tak, stosując jednoargumentowe operatory `&` i *, otrzymujemy równoważne formy wyrażeń:

```
x      i  &x[0]     oraz  x[0]    i  *x
x+1    i  &x[1]           x[1]    i  *(x+1)
x+2    i  &x[2]           x[2]    i  *(x+2)
.................         ...................
x+k    i  &x[k]           x[k]    i  *(x+k)
.................         ...................
```

Wyrażenia warunkowe w C i C++

Niespodziankę dla Czytelnika podejmującego się przejścia od Pascala do języków C i C++ mogą stanowić *wyrażenia warunkowe* występujące w funkcji *brzeg*, określające sposób obliczania wartości zmiennych *v* i *z*, utworzone za pomocą operatora `?:` wymagającego aż trzech argumentów. Ich składnia jest następująca:

```
wyrażenie1 ? wyrażenie2 : wyrażenie3
```

Najpierw obliczane jest pierwsze wyrażenie. Jeśli jego wartość jest różna od zera, obliczana jest wartość drugiego wyrażenia i ona zostaje potraktowana jako wartość całego wyrażenia warunkowego. W przeciwnym przypadku obliczana jest wartość trzeciego wyrażenia i ona zostaje przyjęta jako wartość wynikowa. Tak więc spośród dwóch wyrażeń, drugiego i trzeciego, obliczane jest tylko jedno.

Prostym przykładem użycia wyrażenia warunkowego jest wyznaczenie wartości *c* jako większej z *a* i *b*. Można oczywiście zastosować instrukcję warunkową

```
if (a>b)
   c = a;
else
   c = b;
```

ale prościej jest napisać

```
c = (a>b) ? a : b;
```

Wyrażenie warunkowe może występować w innych wyrażeniach, czego przykładem jest wyrażenie określające wartość zmiennej *v* w funkcji *brzeg*. Zostało ono użyte w warunku sterującym wykonaniem instrukcji `if`.

Częstotliwość występowania liter w pliku

Zajmiemy się obecnie zliczaniem liter w pliku tekstowym. Warto wspomnieć, że zagadnienie wyznaczenia częstotliwości występowania liter w tekście miało duże znaczenie praktyczne już wiele lat przed pojawieniem się komputerów. Gdy konstruowano pierwsze maszyny do pisania, należało rozmieścić klawisze jej klawiatury tak, aby znaki najczęściej używane były łatwiej dostępne niż znaki używane rzadziej. Również alfabet Morse'a został tak zbudowany, aby znakom częściej używanym odpowiadały krótsze sygnały.

Program Litery w Pascalu

Zakładamy, że program ma uwzględniać tylko litery alfabetu angielskiego, oraz że ma utożsamiać litery duże i małe. Wynikiem jego działania nie mają być jednak liczby wystąpień poszczególnych liter, lecz wykres słupkowy obrazujący częstotliwości ich występowania. Naturalne wydaje się wyodrębnienie dwóch składowych programu, z których pierwsza zlicza litery w pliku, a druga tworzy grafikę. Program taki w języku Turbo Pascal został przedstawiony na wydruku 3.7.

Wydruk 3.7. *Program Litery.pas obrazujący częstotliwość występowania liter w pliku*

```
program Liczba_liter_pliku_tekstowego;

uses Crt, Graph;

const
  Driver: integer = VGA;
  Mode  : integer = VGAHi;

var
  L: array['A'..'Z'] of integer;

procedure Liczenie;
var
  Plik : Text;
  Nazwa: string;
  c    : char;
begin
  for c:='A' to 'Z' do
    L[c] := 0;
  Write('Nazwa pliku: ');
  ReadLn(Nazwa);
  Assign(Plik, Nazwa);
```

```
  Reset(Plik);
  while not Eof(Plik) do
  begin
    Read(Plik, c);
    if c in ['A'..'Z','a'..'z'] then Inc(L[UpCase(c)]);
  end;
  Close(Plik);
end;

procedure Rysowanie;
const
  N     = Ord('Z') - Ord('A') + 1;   { Liczba słupków   }
  dx    = 640 div N;                 { Rozstaw słupków  }
  Width = 3*dx div 5;                { Szerokość słupka }
  Depth = (dx - Width) div 2;        { Głębokość słupka }
var
  Max, x: integer;
  c: char;
begin
  Max := 1;
  for c:='A' to 'Z' do
    if L[c] > Max then Max := L[c];
  x := (640 - N*dx) div 2;
  SetFillStyle(SolidFill, DarkGray);
  for c:='A' to 'Z' do
  begin
    Bar3D(x, 450-Round(L[c]/Max*440), x+Width, 450, Depth, true);
    OutTextXY(x+Depth, 460, c);
    Inc(x, dx);
  end;
end;

begin
  Liczenie;
  InitGraph(Driver, Mode, 'C:\TP\BGI');
  Rysowanie;
  ReadKey;
  CloseGraph;
end.
```

Procedura *Liczenie* oblicza liczby wystąpień poszczególnych liter w pliku tekstowym, umieszczając wynik w tablicy globalnej *L*. Wartościami indeksów elementów tej tablicy są duże litery od A do Z, typ indeksu jest więc podzbiorem typu znakowego `char`, a element `L[c]` przedstawia liczbę wystąpień znaku (litery) *c* w tekście. Najpierw zerowane są wszystkie elementy tablicy *L*. Następnie w pętli pobierany jest z pliku kolejny znak *c* i w przypadku, gdy jest to litera, wartość elementu `L[c]` jest zwiększana o 1. Wcześniej jednak, gdy *c* jest małą literą, indeks elementu jest zamieniany na dużą literę za pomocą funkcji `UpCase`.

Procedura *Rysowanie* rysuje dla każdej litery słupek o wysokości proporcjonalnej do liczby wystąpień tej litery w pliku. W celu uproszczenia kodu procedury i poprawienia jego czytelności zdefiniowano kilka stałych: *N* — liczba słupków (26, tj. liczba liter alfabetu angielskiego), *dx* — rozstaw słupków przy szerokości ekranu 640 pikseli, *Width* — szerokość słupka, *Depth* — głębokość słupka. Najwyższy słupek, o wy-

sokości 440 pikseli, odpowiada literze, która pojawiła się w pliku najczęściej. Licznik wystąpień tej litery jest maksymalną wartością wszystkich elementów tablicy *L*. Wyznacza się go w zmiennej *Max* według schematu:

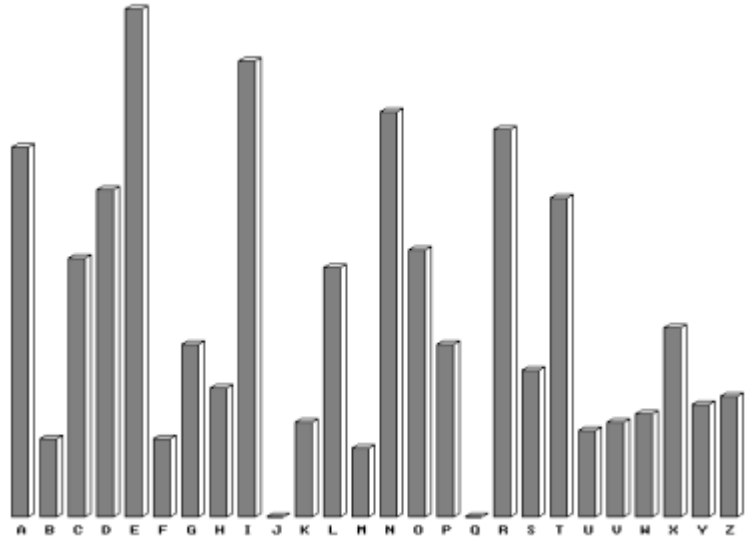
jeżeli kolejny element ma wartość większą od dotychczasowej wartości zmiennej Max, przypisz zmiennej Max wartość tego elementu.

Elementy są nieujemne, na wstępie należało więc wyzerować zmienną *Max*. Jednakże z uwagi na obliczanie wysokości słupka litery *c* ze wzoru

$$h(c) = \frac{L[c]}{Max} 440$$

zmiennej *Max* przypisuje się na początku wartość 1. Unika się w ten sposób błędu dzielenia przez zero, gdy plik nie zawiera żadnej litery.

Słupki są rysowane za pomocą bibliotecznej procedury `Bar3D`. Pierwsze cztery parametry jej wywołania określają współrzędne lewego górnego i prawego dolnego rogu prostokąta stanowiącego ściankę przednią słupka, piąty — głębokość słupka, szósty zaś (`true`) wymusza rysowanie górnego denka (dla `false` denko nie byłoby rysowane). Wspólna wartość *y* = 450 dla dolnego boku ścianek przednich sprawia, że słupki „wyrastają" w górę z jednej płaszczyzny poziomej (por. rysunek 3.5). Jednoliterowe napisy pod słupkami, od *y* = 460, są tworzone za pomocą procedury `OutTextXY`.



Rysunek 3.5. *Częstotliwość występowania liter w pliku Litery.pas*

Program Litery w C++

Przystępując do omówienia analogicznego programu w Borland C++ należy przypomnieć, że elementy tablic są w C i C++ zawsze indeksowane liczbami całkowitymi od zera wzwyż. Zatem element `L[0]` będzie zawierał liczbę wystąpień litery A, `L[1]` — liczbę wystąpień litery B, ..., `L[25]` — liczbę wystąpień litery Z. Krótko mówiąc, licz-

ba wystąpień litery będącej wartością zmiennej *c* zostanie zapamiętana w elemencie `L[c-'A']`. Zmienne i stałe typu `char` są arytmetycznie traktowane identycznie jak zmienne typu `int`, wyrażenie `c-'A'` jest więc poprawne. Jego wartością jest liczba całkowita od 0 do 25, która odpowiada jednej z liter od A do Z zawartej w zmiennej *c*. W celu przekształcenia małej litery na dużą wygodnie jest posłużyć się funkcją `toupper`. Pełna wersja tego programu w Borland C++ została przedstawiona na wydruku 3.8.

Wydruk 3.8. *Program Litery.cpp obrazujący częstotliwość występowania liter w pliku*

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <ctype.h>

#define  N      ('Z'-'A'+1)         // Liczba słupków
#define  DX     (640/N)             // Rozstaw słupków
#define  WIDTH  (3*DX/5)            // Szerokość słupka
#define  DEPTH  ((DX-WIDTH)/2)      // Głębokość słupka

int Driver = VGA,  Mode = VGAHI,  L[N];

int liczenie()
{
   FILE *plik;
   char nazwa[256];
   int  c;
   printf("Nazwa pliku: ");
   gets(nazwa);
   if ((plik = fopen(nazwa, "rt")) != NULL)
   {
      for (c=0; c<N; L[c++]=0);
      while ((c = fgetc(plik)) != EOF)
         if ((c = toupper(c)) >= 'A' && c <= 'Z') L[c-'A']++;
      fclose(plik);
      return 1;
   }
   else
      return 0;
}

void rysowanie()
{
   int Max = 1,  x = (640 - N*DX)/2;
   char *s = "z", c;
   for (c=0; c<N; c++)
      if (L[c] > Max) Max = L[c];
   setfillstyle(SOLID_FILL, DARKGRAY);
   for (c=0; c<N; c++)
   {
      bar3d(x, 450-(L[c]*440.0)/Max, x+WIDTH, 450, DEPTH, 1);
      *s = 'A'+c;
      outtextxy(x+DEPTH, 460, s);
      x += DX;
   }
}
```

```
void main()
{
   if (liczenie())
   {
      initgraph(&Driver, &Mode, "C:\\BC\\BGI");
      rysowanie();
      getch();
      closegraph();
   }
   else
   {
      printf("Nieudane otwarcie pliku\n");
      getch();
   }
}
```

Występująca w funkcji *liczenie* definicja zmiennej *c*, służącej do pamiętania kolejnego znaku pobranego z pliku, może rodzić pytanie: dlaczego zmienna ta jest typu `int`, a nie typu `char`? Powodem użycia typu `int` zamiast `char` jest typ wartości zwracanych przez funkcję `fgetc`, która czyta znak z pliku i zwraca go jako wartość wynikową, a w przypadku napotkania końca pliku zwraca wartość odpowiadającą nazwie symbolicznej `EOF`, określoną w pliku nagłówkowym *stdio.h* jako –1. Tak więc wartościami zwracanymi przez funkcję `fgetc` są wszystkie możliwe znaki i dodatkowa wartość –1 typu `int`, która binarnie jest ciągiem samych jedynek. Przekształcenie jej do typu `char` polega na obcięciu tego ciągu do 8 bitów, co daje wartość 255[5]. Zatem gdyby zmienna *c* była typu `char`, program zawieszałby się, ponieważ wykrycie końca pliku powodowałoby przypisywanie zmiennej *c* wartości 255, w efekcie warunek kontynuacji pętli `while` byłby zawsze spełniony.

Konwersja znaku na łańcuch w C i C++

Wyjaśnienia może wymagać użycie zmiennej wskaźnikowej *s* w podprogramie *rysowanie*, która nie ma swojego odpowiednika w jego wersji pascalowskiej. Jest ona zainicjalizowana jednoznakową stałą tekstową, czyli dwuelementową tablicą znakową. Wartość pierwszego elementu tablicy nie ma tu znaczenia, istotna jest zerowa wartość jej drugiego elementu oznaczająca koniec tekstu (symbol `\0`). W pętli `for` pierwszemu elementowi tablicy jest przypisywana kolejna duża litera od A do Z, a powstały w ten sposób tekst jednoznakowy jest następnie wykorzystywany w wywołaniu funkcji `outtextxy` tworzącej napis pod słupkiem.

Zabieg ten jest właściwie konwersją między typem znakowym a typem tekstowym. W językach C i C++ nie ma kompatybilności między tymi typami, nawet istnieje różnica między stałą znakową i stałą tekstową zawierającą jeden znak. Na przykład `'A'` to nie to samo co `"A"`. Pierwsza stała oznacza znak o wartości numerycznej 65 (kod litery A), zaś druga — tekst o długości jednego znaku, zawierający literę A i zakończony znakiem `\0`. W Turbo Pascalu nie ma takiego problemu, zmiennej łańcuchowej (typu `string`) można przypisać wartość znakową.

[5] Przy włączonej opcji *Uunsigned characters* kompilator Borland C++ traktuje wartości typu `char` jak `unsigned char` (liczby całkowite bez znaku). Domyślnie opcja ta jest włączona.

Definiowanie stałych symbolicznych w C++

Zamieszczone na początku programu cztery dyrektywy `#define`, właściwe językowi C, są odpowiednikiem definicji stałych w pascalowskiej procedurze *Rysowanie*. Definiują one nazwy symboliczne: *N*, *DX*, *WIDTH* i *DEPTH*, z którymi wiążą określone ciągi znaków. Każde wystąpienie takiej nazwy w tekście programu, z wyjątkiem nazw w cudzysłowach, jest przez preprocesor zamieniane na odpowiadający jej ciąg znaków. Stosowanie dyrektyw `#define` prowadzi niekiedy do błędów. Np. gdyby ciąg znaków odpowiadający nazwie *N* nie zawierał nawiasów, ciąg odpowiadający nazwie *DX* określałby nieprawidłowy rozstaw słupków.

W języku C++ istnieje możliwość definiowania stałych za pomocą słowa kluczowego `const`, co pozwala na unikanie dyrektywy `#define` i nieprzyjemnych problemów wynikających z jej stosowania. Omawiane dyrektywy lepiej jest zastąpić następującymi deklaracjami stałych:

```
const int N = 'Z'-'A'+1;          // Liczba słupków
const int DX = 640/N;             // Rozstaw słupków
const int WIDTH = 3*DX/5;         // Szerokość słupka
const int DEPTH = (DX-WIDTH)/2;   // Głębokość słupka
```

Wykres funkcji drgań harmonicznych tłumionych

Sporządzenie wykresu funkcji ciągłej jednej zmiennej sprowadza się do obliczania wartości tej funkcji w odpowiednio dużej liczbie punktów rozpatrywanego przedziału i narysowania linii łamanej o wyznaczonych w ten sposób wierzchołkach — punktach płaszczyzny. W celu zobrazowania wykresu na ekranie monitora komputerowego konieczne jest przejście od rzeczywistego układu współrzędnych, w którym zdefiniowana jest funkcja, do układu ekranowego. Operacja ta, zwana *okienkowaniem* (ang. *windowing*), dotyczy dwóch obszarów prostokątnych o bokach równoległych do osi współrzędnych: *okna* w układzie rzeczywistym i *widoku* (ang. *viewport*) w układzie ekranowym [2].

Okno i widok przy tworzeniu wykresu funkcji

Załóżmy, że potrafimy wyznaczać wartości funkcji ciągłej $y = f(x)$ dla wartości zmiennej x należących do przedziału $[xR_{\min}, xR_{\max}]$. Niech $yR_{\min}$ oznacza minimalną, a $yR_{\max}$ maksymalną wartość f w tym przedziale. Wykres funkcji f mieści się wtedy w prostokątnym oknie

$$[xR_{\min}, xR_{\max}] \times [yR_{\min}, yR_{\max}]$$

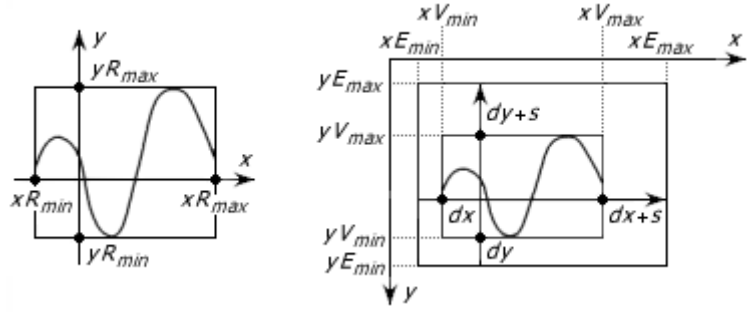
którego odpowiednikiem na ekranie monitora jest widok

$$[xV_{\min}, xV_{\max}] \times [yV_{\min}, yV_{\max}]$$

Zazwyczaj widok nie zajmuje całego ekranu. Na przykład oprócz niego może być wyświetlana dodatkowa informacja tekstowa, tabelka lub inny element graficzny. W przedstawionym na rysunku 3.6 przypadku pokazane jest okno w układzie rzeczywistym (po lewej) i widok w układzie ekranowym (po prawej), zamieszczony w większym prostokącie pozwalającym na dorysowanie osi układu współrzędnych:

$$[xE_{\min}, xE_{\max}] \times [yE_{\min}, yE_{\max}]$$

Rysunek 3.6.
Układ rzeczywisty (po lewej) i ekranowy (po prawej)



Takie rozwiązanie pozwala na wygodne określanie rozmiarów rysunku i jego umiejscowienie na ekranie[6]. Wystarczy ustalić parametry tego prostokąta i wyliczyć granice widoku ze wzorów:

$$xV_{\min} = xE_{\min} + dx, \quad xV_{\max} = xE_{\max} - dx - s$$
$$yV_{\min} = yE_{\min} - dy, \quad yV_{\max} = yE_{\max} + dy + s$$

gdzie *dx* i *dy* są odstępami (poziomym i pionowym) granic prostokątów równymi długościom odcinków osi wystających poza widok, a *s* — długością strzałki kończącej te odcinki. Wartości *dx*, *dy* i *s* można dobrać eksperymentalnie, kierując się względami estetycznymi. Użycie znaku - (minus) zamiast + (plus) we wzorach określających współrzędne *y* widoku jest konsekwencją faktu, iż na ekranie oś *y* jest skierowana w dół. Jest oczywiste, że gdy oś nie jest rysowana, stosowne brzegi obu prostokątów pokrywają się.

Odwzorowanie opisujące przejście od współrzędnych (xR, yR) okna w układzie rzeczywistym do współrzędnych (xV, yV) widoku w układzie ekranowym ma postać:

$$xV = xV_{\min} + sx \cdot (xR - xR_{\min}), \quad yV = yV_{\min} + sy \cdot (yR - yR_{\min})$$

gdzie *sx* i *sy* są czynnikami skalującymi spełniającymi zależności

$$sx = \frac{xV_{\max} - xV_{\min}}{xR_{\max} - xR_{\min}}, \quad sy = \frac{yV_{\max} - yV_{\min}}{yR_{\max} - yR_{\min}}$$

[6] Urządzeniem kreślącym może być również inne urządzenie, np. drukarka i ploter.

Program Drgania w Pascalu

Wykorzystamy teraz powyższe rozważania do sporządzenia wykresu funkcji opisującej drgania harmoniczne tłumione [13]:

$$y = A \cdot e^{-bx} \cos(\omega \cdot x + \varphi)$$

Wzór przedstawia amplitudę (przemieszczenie) y drgającego punktu materialnego w zależności od czasu x. Stała A oznacza amplitudę w chwili początkowej, b — czynnik tłumiący, ω — częstotliwość drgań, φ — fazę początkową. Wartości A, b i ω są liczbami dodatnimi. Wielkość amplitudy zawiera się pomiędzy dwoma asymptotami

$$A_{\min}(x) = -A \cdot e^{-bx} \le y \le A \cdot e^{-bx} = A_{\max}(x)$$

i zanika w czasie, dążąc do zera.

Na wydruku 3.9 przedstawiony jest pełny program w Turbo Pascalu tworzący wykres drgań harmonicznych dla parametrów $A = 3$, $b = 1/2$, $\omega = 5$ i $\varphi = 0$ w przedziale czasowym od 0 do 2π. Podstawowymi składowymi programu są:

- funkcje *xV* i *yV* opisujące przejście od układu rzeczywistego do ekranowego,
- funkcje *Amax* i *Amin* stanowiące deklaracje obydwu asymptot,
- funkcja *Ruch* opisująca drgania harmoniczne tłumione,
- procedura *Ustaw_Paramery* nadająca wartości wszystkim zmiennym globalnym,
- procedura *Rysuj_Uklad* rysująca osie układu współrzędnych,
- procedura *Rysuj_Wykres* rysująca wykres funkcji.

Wydruk 3.9. *Program Drgania.pas tworzący wykres funkcji drgań harmonicznych tłumionych*

```
program Wykres_funkcji_drgan_tlumionych;

uses Crt, Graph;

const
  Karta: integer = VGA;
  Rozdz: integer = VGAHi;

var
  xRmin,xRmax,yRmin,yRmax,sx,sy: real;
  xEmin,xEmax,yEmin,yEmax,xVmin,xVmax,yVmin,yVmax: integer;

type
  Funkcja = function(x: real): real;

function xV(xR: real): integer;
begin
  xV := Round(sx*(xR-xRmin)) + xVmin;
end;

function yV(yR: real): integer;
```

```
begin
  yV := Round(sy*(yR-yRmin)) + yVmin;
end;

function Amax(x: real): real; far;
begin
  Amax := 3*exp(-0.5*x);
end;

function Amin(x: real): real; far;
begin
  Amin := -Amax(x);
end;

function Ruch(x: real): real; far;
begin
  Ruch := Amax(x)*cos(5*x);
end;

procedure Ustaw_Parametry;
var
  d: integer;
begin
  xEmin := 0;            xEmax := GetMaxX;
  yEmin := GetMaxY;      yEmax := 0;
  xRmin := 0;            xRmax := 2*Pi;
  yRmin := Amin(xRmin);  yRmax := Amax(xRmin);
  d := (xEmax-xEmin) div 25;
  xVmin := xEmin+d;      xVmax := xEmax-d-8;
  d := (yEmax-yEmin) div 25;
  yVmin := yEmin+d;      yVmax := yEmax-d+8;
  sx := (xVmax-xVmin)/(xRmax-xRmin);
  sy := (yVmax-yVmin)/(yRmax-yRmin);
end;

procedure Rysuj_Uklad;
var
  d: integer;
begin
  d := yV(0);
  MoveTo(xEmin, d);  LineTo(xEmax, d);
  LineRel(-8, -3);   MoveTo(xEmax, d);   LineRel(-8, 3);
  OutTextXY(xEmax-8, d-14, 'x');
  d := xV(0);
  MoveTo(d, yEmin);  LineTo(d, yEmax);
  LineRel(-3, 8);    MoveTo(d, yEmax);   LineRel(3, 8);
  OutTextXY(d-14, yEmax, 'y');
end;

procedure Rysuj_Wykres(f: Funkcja);
var
  xV: integer;
begin
  MoveTo(xVmin, yV(f(xRmin)));
  for xV:=xVmin+1 to xVmax do
     LineTo(xV, yV(f(xRmin+(xV-xVmin)/sx)));
end;
```

```
begin
  InitGraph(Karta, Rozdz, 'C:\TP\BGI');
  Ustaw_Parametry;
  SetColor(Cyan);
  Rysuj_Uklad;
  Rysuj_Wykres(Amax);
  Rysuj_Wykres(Amin);
  SetColor(White);
  Rysuj_Wykres(Ruch);
  ReadKey;
  CloseGraph;
end.
```

Procedura *Ustaw_Parametry* wykonuje szereg ważnych operacji pomocniczych. Najpierw określa usytuowanie i wielkość prostokąta obejmującego rysunek, przypisując zmiennym *xEmin*, *xEmax*, *yEmin* i *yEmax* stosowne wartości krańcowe. Proponowany prostokąt zajmuje cały ekran, ponieważ jego lewy górny róg jest umiejscowiony w punkcie (0, 0), a prawy dolny w punkcie (`GetMaxX`, `GetMaxY`). Funkcja `GetMaxX` podaje maksymalną wartość współrzędnej *x*, a `GetMaxY` maksymalną wartość współrzędnej *y* ekranu graficznego (dla trybu `VGAHi` są to liczby 639 i 479). Następnie procedura *Ustaw_Parametry* wyznacza okno w układzie rzeczywistym, przypisując zmiennym *xRmin* i *xRmax* wartości końców rozpatrywanego przedziału czasowego, a zmiennym *yRmin* i *yRmax* minimalną i maksymalną amplitudę. Kolejne obliczenia dotyczą ustalenia krańców widoku w układzie ekranowym. Ponieważ na rysunku mają być pokazane odcinki obu osi układu współrzędnych, granice widoku są przesunięte względem granic prostokąta obejmującego cały rysunek o 1/25 jego szerokości i 1/25 wysokości, z uwzględnieniem strzałek o długości 8 pikseli. Wartości wynikowe zostają zapamiętane w zmiennych *xVmin*, *xVmax*, *yVmin* i *yVmax*. Na koniec procedura wylicza w zmiennych *sx* i *sy* czynniki skalujące, które są wykorzystywane przez funkcje *xV* i *yV*.

Uproszczenia wynikające z dbałości o niewielki i czytelny kod programu sprawiły, że procedura *Ustaw_Parametry* nie umożliwia określania parametrów drgań harmonicznych (A, b, ω i φ) i zakresu czasowego (*xRmax*), ani ustalania rozmiaru i położenia rysunku na ekranie (*xEmin*, *xEmax*, *yEmin*, *yEmax*). Czytelnik może takie udoskonalenia wprowadzić, modyfikując kod programu (podprogramy *Amax*, *Ruch* i *Ustaw_Parametry*).

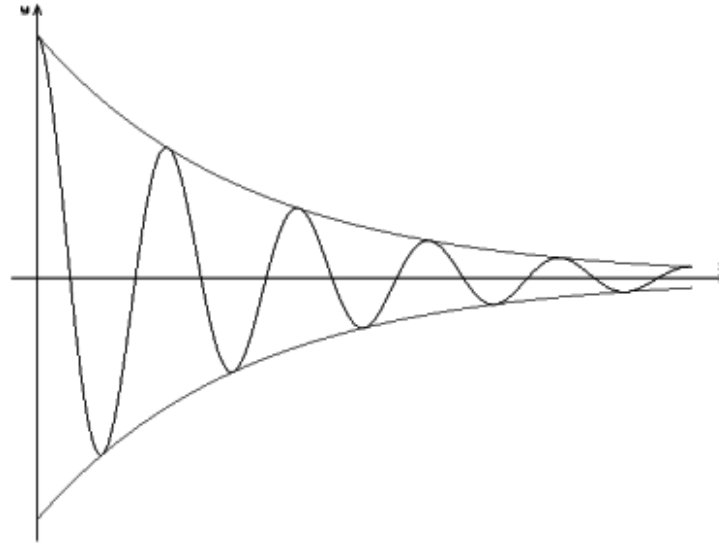
Procedura *Rysuj_Uklad* jest bardzo prosta. Rysuje odcinki osi układu współrzędnych, kończąc je strzałkami i opisując tradycyjnymi jednoliterowymi nazwami *x* i *y*. Strzałki są składane z dwóch krótkich odcinków za pomocą procedury `LineRel`, która rysuje odcinek od bieżącej pozycji pisaka do jego nowej pozycji przesuniętej o określoną liczbę pikseli.

Nazwa funkcji parametrem podprogramu w Pascalu

Szczególnie istotną rolę w powyższym programie odgrywa krótka procedura *Rysuj_Wykres*, która rysuje wykres funkcji *f* określonej w parametrze. Zmienna lokalna *xV* przebiega kolejno wszystkie wartości całkowite na osi *x* widoku w układzie ekranowym, te zaś są przeliczane na wartości osi *x* okna w układzie rzeczywistym. W konse-

kwencji uzyskuje się ciąg równoodległych punktów wypełniających na całej szerokości przedział określoności funkcji *f*, w których wylicza się wartości $y = f(x)$. Otrzymane w ten sposób punkty (x, y) w układzie rzeczywistym są wierzchołkami łamanej przybliżającej funkcję. Odwzorowanie ich do układu ekranowego (tylko współrzędnej *y*, bo ekranowe *x* jest wartością zmiennej *xV*) umożliwia rysowanie na ekranie łamanej stanowiącej graficzne przedstawienie funkcji *f*.

Wynik wykonania programu jest pokazany na rysunku 3.7. Uzyskany obraz przedstawia przebiegi trzech funkcji: amplitudy górnej, amplitudy dolnej i drgania harmonicznego tłumionego w czasie. Potrójny wykres jest efektem trzykrotnego wywołania procedury *Rysuj_Wykres*: dla funkcji *Amax*, *Amin* i *Ruch*. Wystąpienie nazw tych funkcji w roli parametru procedury *Rysuj_Wykres* wymagało zdefiniowania nowego typu funkcyjnego *Funkcja* i użycia słowa kluczowego `far` w ich deklaracjach[7].



Rysunek 3.7. *Wykres drgań harmonicznych tłumionych w czasie*

Nazwa funkcji a wskaźnik w C i C++

Przetłumaczenie omówionego wyżej programu pascalowskiego na język Borland C++ jest czynnością niemal automatyczną. Jedyny kłopot może sprawić podprogram rysowania wykresu funkcji. Nazwa funkcji, podobnie jak nazwa tablicy, jest w C i C++ wskaźnikiem. Przekonało się o tym dobitnie wielu studentów programujących w Turbo Pascalu, którzy na początku swojej przygody z Borland C++ próbowali wyczyścić ekran za pomocą instrukcji

```
clrscr;
```

[7] Turbo Pascal rozróżnia wskaźniki 16-bitowe (`near` — bliskie) i 32-bitowe (`far` — dalekie). Nazwa podprogramu jest wskaźnikiem, a gdy występuje jako parametr, musi być wskaźnikiem 32-bitowym. W środowiskach 32-bitowych słowa `far` i `near` są ignorowane.

Ku ich wielkiemu zaskoczeniu ekran pozostał zaśmiecony, ponieważ jest to formalnie poprawna instrukcja wyrażeniowa, która jedynie wyznacza wskaźnik do podprogramu `clrscr`. Pożądaną instrukcją była

```
clrscr();
```

Powróćmy do deklaracji podprogramu rysowania funkcji. Prawidłowym jego parametrem określającym funkcję, której wykres ma być tworzony, jest

```
double (*f)(double)
```

Zapis ten mówi, że *f* jest wskaźnikiem do funkcji zmiennej rzeczywistej zwracającej wynik rzeczywisty. Pierwsza para nawiasów jest tu bardzo istotna, gdyż bez nich zapis określałby funkcję zwracającą wskaźnik do wartości rzeczywistej.

Program Drgania w C++

Pełny kod programu w języku Borland C++, rysującego wykres funkcji drgań harmonicznych tłumionych w czasie, został przedstawiony na wydruku 3.10.

Wydruk 3.10. *Program Drgania.cpp tworzący wykres funkcji drgań harmonicznych tłumionych*

```
#include <graphics.h>
#include <conio.h>
#include <math.h>

int Karta = VGA,  Rozdz = VGAHI,
    xEmin, xEmax, yEmin, yEmax, xVmin, xVmax, yVmin, yVmax;

double xRmin, xRmax, yRmin, yRmax, sx, sy;

int xV(double xR)
{
   return sx*(xR-xRmin) + xVmin;
}

int yV(double yR)
{
   return sy*(yR-yRmin) + yVmin;
}

double Amax(double x)
{
   return 3*exp(-0.5*x);
}

double Amin(double x)
{
   return -Amax(x);
}

double Ruch(double x)
{
   return Amax(x)*cos(5*x);
```

```
}

void ustaw_parametry()
{
   int d;
   xEmin = 0;            xEmax = getmaxx();
   yEmin = getmaxy();    yEmax = 0;
   xRmin = 0;            xRmax = 2*M_PI;
   yRmin = Amin(xRmin);  yRmax = Amax(xRmin);
   xVmin = xEmin + (d = (xEmax-xEmin)/25);
   xVmax = xEmax - d - 8;
   yVmin = yEmin + (d = (yEmax-yEmin)/25);
   yVmax = yEmax - d + 8;
   sx = (xVmax-xVmin)/(xRmax-xRmin);
   sy = (yVmax-yVmin)/(yRmax-yRmin);
}

void rysuj_uklad()
{
   int d;
   moveto(xEmin, d=yV(0));  lineto(xEmax, d);
   linerel(-8, -3);  moveto(xEmax, d);  linerel(-8, 3);
   outtextxy(xEmax-8, d-14, "x");
   moveto(d=xV(0), yEmin);  lineto(d, yEmax);
   linerel(-3, 8);   moveto(d, yEmax);  linerel(3, 8);
   outtextxy(d-14, yEmax, "y");
}

void rysuj_wykres(double (*f)(double))
{
   moveto(xVmin, yV(f(xRmin)));
   for (int xV=xVmin+1; xV<=xVmax; xV++)
      lineto(xV, yV(f(xRmin+(xV-xVmin)/sx)));
}

void main()
{
   initgraph(&Karta, &Rozdz, "C:\\BC\\BGI");
   ustaw_parametry();
   setcolor(CYAN);
   rysuj_uklad();
   rysuj_wykres(Amax);
   rysuj_wykres(Amin);
   setcolor(WHITE);
   rysuj_wykres(Ruch);
   getch();
   closegraph();
}
```

W funkcji *rysuj_wykres* skorzystano z powszechnie stosowanego w C++ udogodnienia deklarowania zmiennej w wyrażeniu inicjującym pętli `for`. Dotyczy to zmiennej *xV* pełniącej rolę licznika pętli. Tak zadeklarowana zmienna jest dostępna od miejsca jej zadeklarowania do końca bloku obejmującego pętlę, w tym przypadku do końca funkcji *rysuj_wykres*.

Ćwiczenia

Ćwiczenie 3.1

Trójkąt Sierpińskiego można wygenerować, korzystając z układu trzech tzw. *iterowanych odwzorowań*, które punktowi (x, y) przyporządkowują punkt (x', y') na podstawie równań:

a) $x' = 0{,}5x, \quad y' = 0{,}5y$

b) $x' = 0{,}5x + 0{,}5, \quad y' = 0{,}5y$

c) $x' = 0{,}5x + 0{,}25, \quad y' = 0{,}5y + 0{,}5$

Algorytm jest równie prosty jak w przypadku omówionej na początku tego rozdziału gry w chaos, stanowi właściwie jej odmianę. Na początku wybieramy dowolny punkt (x_0, y_0). Następnie losujemy jedno z trzech odwzorowań i wyznaczamy punkt (x_1, y_1) jako obraz punktu (x_0, y_0) przekształconego przez wylosowane odwzorowanie. Losowanie odwzorowania i wyznaczanie kolejnego punktu powtarzamy, np. 50 000 razy. Otrzymany ciąg punktów, po ewentualnym pominięciu pewnej liczby początkowych elementów, tworzy trójkąt Sierpińskiego zawarty w kwadracie $[0, 1]\times[0, 1]$. Przedstawienie go na ekranie monitora wymaga odwzorowania tego kwadratu do stosownego widoku w układzie ekranowym. Napisać program rysujący w ten sposób trójkąt Sierpińskiego.

Ćwiczenie 3.2

Napisać program, który rysuje liść paprotki Barnsleya[8], wykorzystując układ czterech iterowanych odwzorowań losowanych z różnym prawdopodobieństwem p:

a) $x' = 0{,}849x + 0{,}037y + 0{,}075, \quad y' = -0{,}037x + 0{,}849y + 0{,}183, \quad p = 0{,}74$

b) $x' = 0{,}197x - 0{,}226y + 0{,}400, \quad y' = 0{,}226x + 0{,}197y + 0{,}049, \quad p = 0{,}13$

c) $x' = -0{,}150x + 0{,}283y + 0{,}575, \quad y' = 0{,}260x + 0{,}237y - 0{,}084, \quad p = 0{,}11$

d) $x' = 0{,}500, \quad y' = 0{,}160y, \quad p = 0{,}02$

Pożądany nierównomierny rozkład prawdopodobieństwa można uzyskać, rozróżniając cztery zakresy wartości losowych Random(100): 0 – 73, 74 – 86, 87 – 97 i 98 – 99.

Ćwiczenie 3.3

Zmodyfikować procedurę rysowania gwiazdki (wydruki 3.3 i 3.4) tak, aby jedno jej ramię było skierowane w górę. Napisać program przedstawiający flagę USA, który wykorzystuje tak zmodyfikowaną procedurę do rysowania 50 gwiazdek pięcioramiennych.

[8] Trójkąt Sierpińskiego i paprotka Barnsleya są tzw. *fraktalami*. Te i inne przykłady fraktali oraz ich metody generowania można znaleźć np. w książkach [6, 8, 9].

Ćwiczenie 3.4

Napisać program, który tworzy wykres funkcji

$$y = \sin x \cdot \cos 3x - \frac{1}{4}\cos 2x + \frac{1}{20}x^2 - 1, \quad x \in [-2\pi, 2\pi]$$

Przed sporządzeniem wykresu program powinien wyznaczyć wysokość okna w układzie rzeczywistym, znajdując minimalną i maksymalną wartość funkcji. Obliczenia te nie muszą być bardzo dokładne, wystarczy porównać wartości funkcji w punktach dzielących przedział np. na 100 równych odcinków.

Ćwiczenie 3.5

W obliczeniach numerycznych korzysta się niekiedy z wielomianów Czebyszewa, które można zdefiniować następująco:

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \quad (k = 2,3,...)$$

Ich wartości bezwzględne dla x należących do przedziału [-1, 1] nie przekraczają wartości 1. Napisać program, który w jednym widoku, odpowiadającemu oknu [-1, 1]×[-1, 1], rysuje wykresy kilku początkowych wielomianów Czebyszewa.