Przykładowe programiki pokazujące problemy numeryczne

Wojciech Myszka

26 października 2008

Poniżej załączam kod użytych na wykładzie przykładowych programów pokazujących problemy numeryczne związane z arytmetyką komputerów.

1. Najprostsze dodawanie (źródło programu strona MSDN zatytułowana „Why Floating-Point Numbers May Lose Precision")

```
// Floating-point_number_precision.c
// Compile options needed: none. Value of c is printed with a decimal
// point precision of 10 and 6 (printf rounded value by default) to
// show the difference
#include <stdio.h>

#define EPSILON 0.0001    // Define your own tolerance
#define FLOAT_EQ(x,v) (((v - EPSILON) < x) && (x <( v + EPSILON)))

int main() {
   float a, b, c;

   a = 1.345f;
   b = 1.123f;
   c = a + b;

   // if (FLOAT_EQ(c, 2.468)) // Remove comment for correct result
   if (c == 2.468)            // Comment this line for correct result
      printf("Liczby sa rowne.\n");
   else
      printf("Liczby nie sa rowne!  Wartosc c wynosi %13.10f "
             "or %f\n",c,c);
}
```

Komentarz: *Program dodaje dwie liczby, a wynik zapisuje w zmiennej o nazwie* `c`*, a następnie sprawdza czy wynik jest taki jak oczekiwano. Niestety nie jest.*

Wyniki:

```
Liczby nie sa rowne!  Wartosc c wynosi  2.4679999352
```

Poniższy wariant programu definuje swoją własną operację porównania `FLOAT_EQ` która sprawdza czy dwie liczby są równe z dokładnością ε:

```c
// Floating-point_number_precision.c
// Compile options needed: none. Value of c is printed with a decimal
// point precision of 10 and 6 (printf rounded value by default) to
// show the difference
#include <stdio.h>

#define EPSILON 0.0001    // Define your own tolerance
#define FLOAT_EQ(x,v) (((v - EPSILON) < x) && (x <( v + EPSILON)))

int main() {
   float a, b, c;

   a = 1.345f;
   b = 1.123f;
   c = a + b;

   if (FLOAT_EQ(c, 2.468)) // Remove comment for correct result
   // if (c == 2.468)            // Comment this line for correct result
      printf("Liczby sa rowne.\n");
   else
      printf("Liczby nie sa rowne!  Wartosc c wynosi %13.10f "
             "or %f\n",c,c);
}
```

Wyniki:

```
Liczby sa rowne.
```

2. Program w Pascalu dokonujący wielokrotnie prostej dosyć operacji. Na podstawie Some issues on floating-point precision under Linux

```
program caos;

{$n+}        { you need to activate hardware floating-point calculation
               in order to use the extended type }

uses
   crt;

var
   s : single;     { 32-bit real }
   r : real;       { 48-bit real }
   d : double;     { 64-bit real }
   e : extended;   { 80-bit real }

   i : integer;

begin
   clrscr;
```

```
    s := 0.5;
    r := 0.5;
    d := 0.5;
    e := 0.5;

    for i := 1 to 100 do begin
        s := 3.8 * s * (1 - s);
        r := 3.8 * r * (1 - r);
        d := 3.8 * d * (1 - d);
        e := 3.8 * e * (1 - e);

        if (i/10 = int(i/10)) then begin
            writeln (i:10, s:16:5, r:16:5, d:16:5, e:16:5);
        end;
    end;

    readln;
end.
```

Program wykonuje dość proste obliczenia: sto razy wylicza wyrażenie:

$$x = 3.8 * x * (1 - x)$$

Drukowana jest co dziesiąta wartość:

10	0.18510	0.18510	0.18510	0.18510
20	0.23951	0.23963	0.23963	0.23963
30	0.88423	0.90200	0.90200	0.90200
40	0.23013	0.82493	0.82493	0.82493
50	0.76654	0.53714	0.53714	0.53714
60	0.42039	0.66878	0.66878	0.66879
70	0.93075	0.53190	0.53190	0.53203
80	0.28754	0.93557	0.93557	0.93275
90	0.82584	0.69203	0.69203	0.79884
100	0.38775	0.41983	0.41983	0.23138

Widać wyraźnie, że precyzja liczb ma wpływ na wyniki obliczeń...