

Temat 2. Różnice dokładności między zmiennoprzecinkowymi typami danych: float, double i long double.

Wiktor Gut, 411 761 Sprawozdanie powstało razem z prezentacją przedstawioną na zajęciach

Środowisko

Programy wykonywane w systemie Windows x64, na procesorze AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx, za pomocą programu CLion w języku C.

Treść zadania:

Wykonać obliczenia (dla zmiennych typu float, double, long double) wg podanych poniżej wzorów dla 101 równoodległych wartości x z przedziału $[0.99, 1.01]$:

1. $f(x) = x^8 - 8x^7 + 28x^6 - 56x^5 + 70x^4 - 56x^3 + 28x^2 - 8x + 1$
2. $f(x) = ((((((x - 8)x + 28)x - 56)x + 70)x - 56)x + 28)x - 8)x + 1$
3. $f(x) = (x - 1)^8$
4. $f(x) = e^{(8 \ln(\text{abs}(x-1)))}$, $x \neq 1$

Porównać wyniki. Objaśnić różnice w wynikach. (Potęgowanie w poleceniu przedstawiłem tutaj znakiem „^”)

Uwagi

Spostrzeżenie 1: Korzystając z przekształceń matematycznych jesteśmy w stanie dojść do wniosku, że wszystkie 4 spośród przedstawionych funkcji sprowadzają się do tej samej, przedstawionej w podpunkcie 3: $(x-1)^8$; W przypadku pierwszych dwóch funkcji jest to kwestia np. tabelki Hornera (po wymnożeniu nawiasów w 2), natomiast w 4 funkcji trzeba skorzystać z własności parzystych potęg tj. $|x|^8 = x^8$, ponieważ wynik potęgowania parzystego jest zawsze nieujemny. Następnie włączamy 8 do logarytmu jako potęgę i skracamy całe wyrażenie wiedząc, że $e^{\ln(x)} = x$, tutaj otrzymując $(x-1)^8$.

W przypadku każdego typu danych skorzystałem z funkcji jak najmniej ograniczających dokładność (math.h), a w przypadku braku wystarczających funkcji, napisałem własne:

- W przypadku podnoszenia do potęg i logarytmowania istniejące funkcje: powf, pow, powl do potęgowania oraz logf, log i logl do logarytmu naturalnego odpowiednio dla float, double i long double.
- Również przy zwracaniu wartości została zachowana dokładność z zastosowaniem zapisu %f, %lf i %Lf odpowiednio dla float, double i long double.

- W przypadku wyciągania wartości absolutnej istnieje tylko funkcja „fabs(double x)”, dlatego napisałem dodatkowo funkcje (Screenshot1) z zamiarem utrzymania możliwej dokładności przy każdym przekształceniu:

```
float flabs(float x) {
    if (x < 0)
        x *= (-1);
    return x;
}

long double ldabs(long double x) {
    if (x < 0)
        x *= (-1);
    return x;
}
```

(Screenshot1)

Skala wyświetlania wyników

Aby w pełni zauważyć różnice pomiędzy wynikami musimy patrzeć na przynajmniej 35-40 miejsc po przecinku, co wynika z obliczeń:

$$(0.0002)^8 = 2^8 * 10^{(-4 * 8)} = 256 * 10^{(-32)},$$

czyli potrzebujemy minimum 32 miejsca po przecinku, jednak na dużej części obliczeń założyłem 60, żeby zobaczyć co dzieje się po końcu liczby. Założona tutaj liczba (0.0002) jest najmniejszym niezerowym argumentem jaki przyjmuje funkcja 3 (już po odjęciu wartości od 1), dlatego łatwo nam na nim zauważyć gdzie będzie ostatnia cyfra szukanych liczb.

Dokładność typów zmiennoprzecinkowych: float, double i long double

Po wykonaniu się funkcji otrzymałem oczywiście mnóstwo wyników na które nie ma tutaj miejsca, lecz zauważając że nie chodzi tyle o ilość wyników co wynikającą z ilości argumentów dokładność argumentów, do zauważenia różnic wystarczą nam 2 przypadki: przypadek liczby która daje dużo zerowych cyfr oraz taka z dużą ilością niezerowych.

Mały i duży przedział cyfr znaczących-mała i duża mantysa

[illegible]

(Screenshot 2 i 3)

[illegible][illegible]

(Screenshot 4)

Takie same obliczenia zostały wykonane dla potęg 1-12 liczby 0.0088 jako jednego z argumentów o możliwie długim ciągu niezerowych cyfr; 88^8 daje ok.16 cyfr, a 88^{12} ok.23. (Screenshot 4)

W obu przypadkach widać widać pasmo zer występujące po liczbach, która w końcu zanika, przytłoczona liczbą cyfr koniecznych do zapamiętania przez zmienną, co wynika z ilości bitów przeznaczonych na mantysę i cechę w tych typach zmiennych. Przy próbie z 0.0088 widać tylko dwa bloki danych, ponieważ blok floatowy niemal od razu przestał tracić dokładność i nie został załączony.

W przypadku liczby 0.0002 rząd wielkości zmieniał się dynamicznie, ale liczba cyfr doszła ledwie do 4 w przypadku potęgi 12, dlatego nie zmienna nie zgubiła dokładności nawet przy takim obliczeniu, jednak w przypadku 0.0088 już przy 7 potędze ilość cyfr liczby przebiła dokładność zmiennej. Prowadzi nas to do wniosku 1:

Wniosek 1: Budowa typów zmiennych (mantysa i cecha) wpływa na zapamiętywanie danych kolokwialnie w taki sposób, że jeśli ułamek jest nawet bardzo mały, to odpowiednio o tyle miejsc po przecinku przesuwają się nasze dokładności: zmienna danego typu jest w stanie dokładnie zapamiętać rząd wielkości oraz określony przedział cyfr liczby więc nie tylko ilość miejsc po przecinku, a rozstrzał pierwszej i ostatniej niezerowej cyfry ma znaczenie.

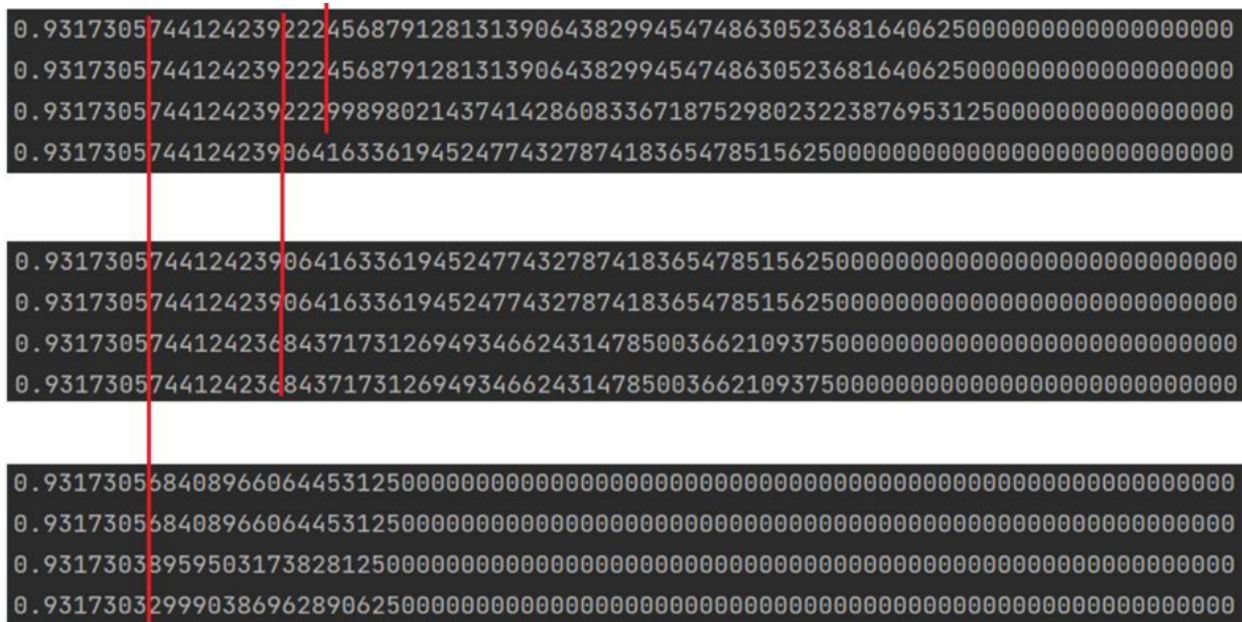
Zostało to również uwidocznione przy porównaniu 3 wartości: 0.0088^8 jako float(1), double(2), long double(3) oraz dwóch różnic: (3)-(2) jako double i (3)-(2) jako long double (4 i 5):

[illegible]

Jak dobrze widać po zrobionych odstępach, typ float jako pierwszy traci dokładność już po 7 cyfrach niezerowych. Typy double i long double rozbiegają się dopiero po 16 cyfrach, przy czym w takim porównaniu ciężko jest doświadczać wyznaczyć dokładność long double'a. Widać natomiast, że jest dokładniejszy; Wynik (4) jest równy 0, ponieważ pamięć double'a zapamiętuje dokładnie te miejsca, w których dwie porównywane liczby są identyczne, a ostatni bit mantysy zostaje zaokrąglony do 0. W przypadku (5) typ long double jest w stanie trzymać większą dokładność, więc otrzymujemy niezerowy wynik. **Można więc powiedzieć, że doświadczalnie udowodniliśmy dokładności floata i double'a oraz zwiększoną nad nimi dokładność long double'a.**

Porównanie wyników funkcji dla tych samych typów danych (Screenshot 5)

Tym razem od góry wyniki dla typów w kolejności long double, double i float.



```

0.931730574412423922245687912813139064382994547486305236816406250000000000000000
0.931730574412423922245687912813139064382994547486305236816406250000000000000000
0.931730574412423922299898021437414286083367187529802322387695312500000000000000
0.931730574412423906416336194524774327874183654785156250000000000000000000000000

0.931730574412423906416336194524774327874183654785156250000000000000000000000000
0.931730574412423906416336194524774327874183654785156250000000000000000000000000
0.931730574412423684371731269493466243147850036621093750000000000000000000000000
0.931730574412423684371731269493466243147850036621093750000000000000000000000000

0.931730568408966064453125000000000000000000000000000000000000000000000000000000
0.931730568408966064453125000000000000000000000000000000000000000000000000000000
0.931730389595031738281250000000000000000000000000000000000000000000000000000000
0.931730329990386962890625000000000000000000000000000000000000000000000000000000
  
```

(Screenshot 5)

Dla wartości x bliskich 1, różnice między wartościami $f(x)$ dla różnych typów zmiennych wynikają z niedokładności reprezentacji liczb zmiennoprzecinkowych w pamięci komputera. Ponadto, w przypadku funkcji 4, różnice w wynikach wynikają z niedokładności przybliżeń stosowanych w funkcjach eksponencjalnej i logarytmicznej, a także z różnic w implementacji tych funkcji dla różnych typów zmiennych.

Na obrazie (Screenshot 5) widać, że wyniki funkcji 4 najszybciej odbiegają od reszty, za to warto zauważyć, że wyniki funkcji 1 i 2 są takie same niezależnie od typu (parami takie same), hipotezą może być podobny charakter funkcji, lecz za to i w 1 i w 3 zostały użyte funkcje wykładnicze, które powinny przyczynić się do podobieństwa wyników funkcji 1 i 3. Nasuwa się wniosek, że dużą rolę może grać kolejność wykonywania działań. W funkcjach 1 i 2 operujemy potęgowo na liczbach znacząco większych niż w funkcji 3; W pierwszej funkcji wartości x oscylują wokół 1, natomiast w funkcji 2 skaczą często między dodatnimi a ujemnymi dziesiątkami, w porównaniu do potęgowania liczb mniejszych niż 0.01 w funkcji 3.

Wniosek 2: Funkcje arytmetyczne (+, -, *) wpływają na dokładność wyników zmiennoprzecinkowych widocznie mniej, niż funkcje logarytmiczne i wykładnicze. Widoczne znaczenie przy precyzji wyników ma również kolejność wykonywania działań i skala wartości.

Bibliografia: sprawozdanie powstało na bazie 1 wykładu z przedmiotu MOWNiT 03.03.23 oraz doświadczeń własnych wykonanych w programie CLion z użyciem języka C.