MOWNiT- laboratorium 1

Michał Bert grupa 4 (śr. 16:40-18:10)

Spis treści

Cel laboratorium (Zadanie nr 1)

Celem laboratorium było zapoznanie się z arytmetyką komputerową oraz sposobem reprezentacji liczb zmiennoprzecinkowych.

W przydzielonym zadaniu należało sprawdzić konkretne parametry reprezentacji liczb zmiennoprzecinkowych (rozmiar, ilość bitów na mantysę oraz cechę), a także występowanie wartości specjalnych.

Konfiguracja sprzętowa

Zadanie zostało wykonane na komputerze z systemem Windows 11 oraz procesorem Intel Core i5-12400f.

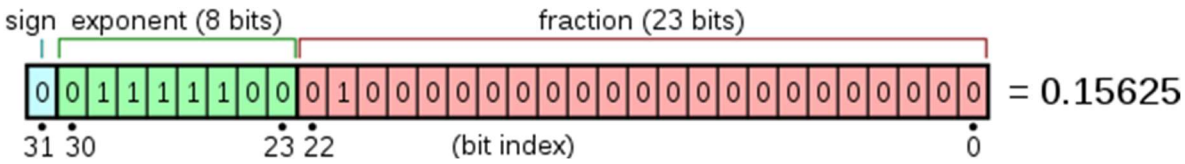
Dla porównań wykorzystane zostały 2 maszyny wirtualne z systemem Ubuntu: 64-bitowy Ubuntu 22.04 oraz 32-bitowy Ubuntu 14.04. Do wirtualizacji zostało wykorzystane oprogramowanie VirtualBox firmy Oracle.

Kod źródłowy zadania został napisany w języku C++. W przypadku Windowsa porównane zostały 2 kompilatory – g++ oraz MSVC (Microsoft Visual C++). Na maszynach z Ubuntu wykorzystany został tylko kompilator g++.

Standaryzacja IEEE 754

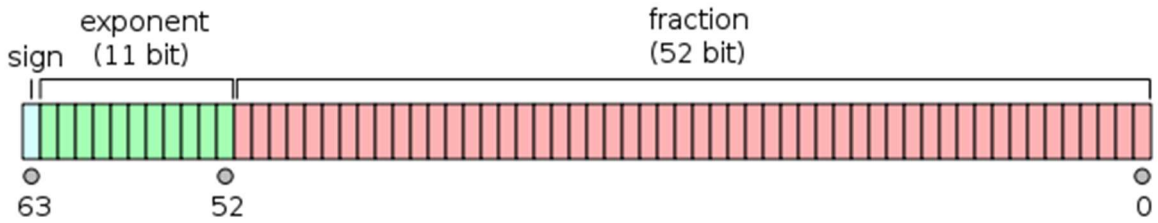
Sposób reprezentacji liczb zmiennoprzecinkowych określa standard IEEE 754. Najważniejsze określane przez niego parametry to:

- Określenie ilości bitów przypadających na znak, cechę oraz mantysę
  - W przypadku reprezentacji 32-bitowej podział wygląda następująco:
  - Znak – 1 bit
  - Cecha – 8 bitów
  - Mantysa – 23 bity



Rysunek 1: Reprezentacja liczby 32-bitowej

- W przypadku reprezentacji 64 bitowej stosowany jest poniższy podział:
- Znak – 1 bit
- Cecha – 11 bitów
- Mantysa – 52 bity



Rysunek 2: Reprezentacja liczby 64-bitowej

- Występowanie wartości szczególnych:
  - 0 oraz -0
  - inf oraz -inf (wyniki działań 1.0/0.0 oraz -1.0/0.0)
  - *NaN – Not A Number*, wartość określana jako (+inf) * 0

- Reguły zaokrągleń

Exponent bias – przesunięcie wykładnika w celu zapisania go jako liczby nieujemnej. Ułatwia to wykonywanie porównań. Przesunięcie jest obliczane ze wzoru $2^{n-1} - 1$, gdzie n jest liczbą bitów przeznaczoną na wykładnik.

Kod testujący zadane wartości

Napisany program wyznacza zadane wartości w następujący sposób:

- Rozmiar liczby:
  - Użyte zostało słowo kluczowe ***sizeof()***, zwracający rozmiar argumentu w bajtach.
- Ilość bitów na mantysę:
  - Krótki algorytm wykonuje dzielenie wartości tymczasowej *a* przez 2, dopóki 1.0 + a != 1 (jest to również wyznaczanie maszynowego epsilonu), jednocześnie inkrementując ilość bitów przeznaczonych na mantysę
  - Wynik jest porównywany z biblioteczną stałą zawierającą ilość bitów na mantysę.
  - Wynik zawiera 1 bit przeznaczony na znak

- Ilość bitów na cechę:
    - Aby wyznaczyć ilość bitów przeznaczonych na wykładnik, skorzystamy ze wzoru $\log_2(\mathrm{MAX}_{\mathrm{EXP}} - MIN_{EXP} + 1)$
- Wyznaczanie maszynowego epsilonu:
    - Dla początkowej wartości zmiennej *a* poszukujemy takiej jej wartości końcowej, że 1.0 + a != 1
    - Znajdujemy ją poprzez dzielenie jej wartości przez 2 do momentu spełnienia postawionego warunku.
- Zera:
    - +0:
        - Uzyskiwane jest poprzez dzielenie 1.0 przez stosownie dużą liczbę dodatnią.
        - Wszystkie bity są zerami
    - -0:
        - Uzyskiwane jest poprzez dzielenie 1.0 przez liczbę ujemną o odpowiednio dużym module.
        - Wszystkie bity są zerami, z wyjątkiem bitu znaku, który jest równy 1
- Nieskończoności:
    - +inf:
        - Uzyskiwane w wyniku dzielenia 1/0.
    - -inf:
        - Uzyskiwane w wyniku dzielenia -1/0.
- NaN:
    - Uzyskane w wyniku operacji sqrt(-2)

Dodatkowo w przypadku nieskończoności można znaleźć różne definicje:

- std::numeric_limits<float>infinity();
    - Korzysta ona z funkcji zależnej od środowiska ***__built_in_huge_val***, która w przypadku obsługi nieskończoności zwraca inf, a przeciwnym wypadku **DBL_MAX**.
- INFINITY
    - Zdefiniowana jako (float)(**_HUGE_ENUF** * **_HUGE_ENUF**), gdzie **_HUGE_ENUF** zdefiniowany jest jako 1e+300 (o ile podczas w systemie nie jest zdefiniowana inna wartość)

Sama wartość **NaN** zdefiniowana jest jako **–(float)(INFINITY * 0.0F)**.

Porównania wyników

Porównanie dla różnych systemów operacyjnych

g++	Windows 11			Ubuntu 22.04 LTS		
Typ danych	float	double	long double	float	double	long double
Rozmiar (B)	4	8	16	4	8	16
Rozmiar mantysy (b)	24	53	64	24	53	64
Rozmiar cechy (b)	8	11	15	8	11	15
Maszynowy epsilon	1.19209e-07	2.22045e-16	1.0842e-19	1.19209e-07	2.22045e-16	1.0842e-19
Występowanie +0/-0	tak / tak			tak / tak		
Występowanie +inf/-inf	tak / tak			tak / tak		
Wynik sqrt(-2)	nan			-nan		

Tabela 1: Porównanie otrzymanych wartości dla systemów Windows 11 oraz Ubuntu 22.04 LTS

Jak można zauważyć, otrzymane wartości są praktycznie takie same – jedyną różnicą jest występowanie wartości *-nan* na systemie Ubuntu, w przeciwieństwie do samego *nan* na systemie Windows. Wartości te są również zgodne ze standardem IEEE 754.

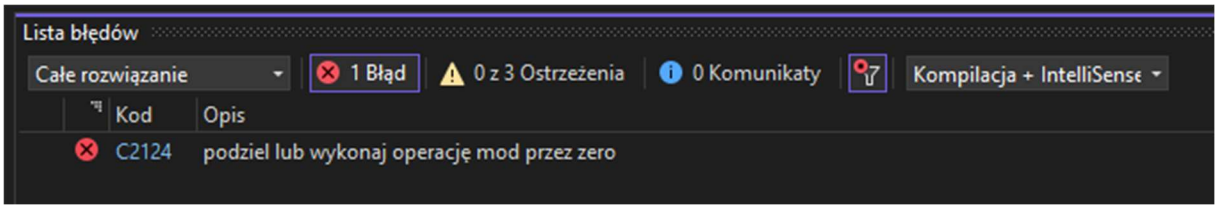
Porównanie dla różnych kompilatorów

Windows 11	g++			MSVC		
Typ danych	float	double	long double	float	double	long double
Rozmiar (B)	4	8	16	4	8	8
Rozmiar mantysy (b)	24	53	64	24	53	53
Rozmiar cechy (b)	8	11	15	8	11	11
Maszynowy epsilon	1.19209e-07	2.22045e-16	1.0842e-19	1.19209e-07	2.22045e-16	2.22045e-16
Występowanie +0/-0	tak / tak			tak / tak		
Występowanie +inf/-inf	tak / tak			tak / tak		
Wynik sqrt(-2)	nan			-nan(ind)		

Tabela 2: Porównanie wyników dla kompilatorów g++ oraz MSVC

W przypadku otrzymanych wyników można zauważyć, że w przypadku kompilatora MSVC typ *double* i *long double* to efektywnie te same typy danych.

Kolejną obserwacją jest większa restrykcyjność kompilatora, który sygnalizuje błąd kompilacji w przypadku dzielenia przez 0 (można jednak skompilować program pomimo tego).



Rysunek 3: Błąd kompilacji przy dzieleniu przez 0.

Ostatnią różnicą jest wynik **-nan(ind)** jako wynik operacji sqrt(-2). Argument **ind** oznacza w tym wypadku ***indeterminate numer***. Jest to szczególny przypadek NaN oznaczający, że liczby tej nie da się przedstawić w formacie zmiennoprzecinkowym.

Wyniki te również są zgodne ze standardem IEEE 754.

Porównanie na różnych architekturach:

Ubuntu	22.04 LTS 64-bit			14.04 LTS 32-bit		
Typ danych	float	double	long double	float	double	long double
Rozmiar (B)	4	8	16	4	8	12
Rozmiar mantysy (b)	24	53	64	24	53	64
Rozmiar cechy (b)	8	11	15	8	11	15
Maszynowy epsilon	1.19209e-07	2.22045e-16	1.0842e-19	1.0842e-19	1.0842e-19	1.0842e-19
Występowanie +0/-0	tak / tak			tak / tak		
Występowanie +inf/-inf	tak / tak			tak / tak		
Wynik sqrt(-2)	-nan			-nan		

Tabela 3: Porównanie wyników dla architektury 32- i 64- bitowej

Otrzymane wartości są bardzo zbliżone, są jednak pewne różnice.

Pierwszą z nich jest rozmiar zmiennej typu *long double* w architekturze 32-bitowej, która wynosi 12 bitów (dalej jednak jest to zgodne ze standardem, który w przypadku typu *long double* jest mniej rygorystyczny i w wielu kwestiach zakłada tylko dolne granice).

Drugą jest wartość maszynowego epsilonu, który ma taką samą wartość dla każdego typu.

Ilość bitów na mantysę została tutaj pobrana bezpośrednio z bibliotek ze względu na błędne wyniki pomiarów.

Wnioski

Otrzymane wyniki wskazują jednoznacznie, że wybór systemu, jego architektury, a także kompilatora użytego do stworzenia oprogramowania ma spore znaczenie w kwestii dokładności liczb zmiennoprzecinkowych. Niektóre z różnic są bardziej „kosmetyczne", jak np. różnica pomiędzy *nan* a *-nan(ind)*, a inne bardziej znaczące, tak jak typ *long double* będący tym samym co typ *double* w przypadku kompilatora MSVC.

Pomimo różnic możemy jednak zauważyć, że otrzymane wyniki są zgodne ze standardami IEEE 754.

Bibliografia

- https://www.geeksforgeeks.org/ieee-full-form/
- https://en.wikipedia.org/wiki/IEEE_754
- https://stackoverflow.com/questions/502022/how-to-find-mantissa-length-on-a-particular-machine
- https://www.codeproject.com/Articles/824516/Concept-of-NaN-IND-INF-and-DEN