Agnieszka Kurzajewska
Nr indeksu: 244994

Zadanie 1.

Wyznaczenie machespu (epsilonu maszynowego), czyli liczby spełniającej własność:

$$fl(1.0 + macheps) > 1.0$$
$$fl(1.0+macheps) = 1+macheps.$$

Polega to na znalezieniu za pomocą programu w języku Julia najmniejszej możliwej liczby większej od liczby 1 dla konkretnego typu zmiennopozycyjnego w standardzie IEEE 754.
„fl" jest oznaczeniem arytmetyki zmiennopozycyjnej.

Type	Precision	Number of bits
`Float16`	half	16
`Float32`	single	32
`Float64`	double	64

W programie występuje zmienna h – której wartość na początku ustawiona na 1.0, docelowo dąży do wartości machepsu

W programie utworzono pętlę, która sprawdza, czy wartość h+1 w podanym typie zmiennopozycyjnym jest większa od 1.0 oraz czy (h/2)+1 nie jest równe 1.0. Jeśli podane warunki zostają spełnione, liczbę „h" dzielimy przez 2. Jeśli podana zmienna nie spełnia warunków, czyli h+1 jest większe od 1.0, ale następna liczba (h/2)+1 zostanie zaokrąglona do zera, program drukuje ostatnią wartość liczby h, która jest ostateczną wartością machepsu. Podane wyniki sprawdzone zostają z wbudowaną funkcją systemową eps(…) w zależności od danego typu zmiennopozycyjnego.
W programie istnieją 3 różne pętle, w których każda operuje na innym typie danych. Obliczone wartości porównane z wynikami wbudowanych funkcji wyglądają następująco:

Half:
Float16: obliczony macheps: 0.000977, macheps podany przez funkcję eps(): 0.000977

Single:
Float32: obliczony macheps: 1.1920929e-7, macheps podany przez funkcję eps(): 1.1920929e-7

Double:
Float64: obliczony macheps: 2.220446049250313e-16, macheps podany przez funkcję eps(): 2.220446049250313e-16

Jak widać, otrzymane wyniki zgadzają się z rzeczywistymi danymi.

Druga część zadania polega na wyznaczeniu wartości „eta" spełniającej zależność:

$$eta > 0.0$$

Zadanie jest wykonane dokładnie w ten sam sposób, zmienione zostają tylko warunki pętli: do h nie zostaje dodana wartość 1.0, a wartość porównywana jest nie z 1.0 a 0.0.

W następnej części wyznaczana jest największa wartość liczby w poszczególnych typach zmiennoprzecinkowych. Zmienna, której początkowa wartość równa się 1, jest zwiększana w każdej iteracji pętli. Warunek pętli sprawdza, czy następna liczba nie jest nieskończona. Jeśli jest, to wynikiem zostaje wartość pomnożona razy dwa, od której zostaje odjęta najmniejsza możliwa wartość (macheps), ponieważ istnieje warunek, że wartość pomnożona razy 2 jest już nieskończona.

Wyniki przedstawiają się w następujący sposób:
Half:
Float16: floatmax obliczone: 6.55e4, floatmax z funkcji wbudowanej: 6.55e4
Single:
Float32: floatmax obliczone: 3.4028235e38, floatmax z funkcji wbudowanej: 3.4028235e38
Double:
Float64: floatmax obliczone: 1.7976931348623157e308, floatmax z funkcji wbudowanej: 1.7976931348623157e308

Zadanie 2.

Zadanie polega na sprawdzeniu słuszności twierdzenia Kahana dla 3 typów zmiennoprzecinkowych:

$$\text{macheps} = 3(\frac{4}{3} - 1)\text{-}1$$

Wyniki działania programu:
Half:
Float16: obliczone z wyrażenia: -0.000977, obliczone przez funkcję eps(): 0.000977
Single:
Float32: obliczone z wyrażenia: 1.1920929e-7, obliczone przez funkcję eps(): 1.1920929e-7
Double:
Float64: obliczone z wyrażenia: -2.220446049250313e-16, obliczone przez funkcję eps(): 2.220446049250313e-16

Wyniki obliczone za pomocą wyrażenia,jak i funkcji eps() są takie same, gdyby wziąć ich wartość bezwzględną. Wynika z tego, że do wzoru Kahana powinna zostać dodana wartość bezwzględna biorąc pod uwagę fakt, że macheps musi być większy od zera.

Zadanie 3.

Zadanie polega na badaniu rozmieszczenia kolejnych liczb w arytmetyce Float64 w przedziałach [1,2], $[\frac{1}{2},1]$, [2,4] korzystając z funkcji bitstring.

Funkcja bitstring zamienia podaną wartość liczbową na jej reprezentację bitową, z której możemy odczytać poszczególne odstępy liczbowe między kolejnymi liczbami.

W programie sprawdzono dla poszczególnych przedziałów i różnych interwałów, czy kolejna wyliczona liczba jest równa wartości funkcji nextFloat. Można zauważyć, że dla odstępu 2^{-52} wartości te są równe dla przedziałów [1,2], $[\frac{1}{2},1]$, natomiast nie zgadzają się w przedziale [2,4]. Po zmianie interwału na 2^{-51} Zgadzają się tylko dane dla przedziału [2,4]

Wnioski:

Dla przedziałów [1,2] i $[\frac{1}{2},1]$ odstęp wynosi 2^{-52}.

Dla przedziału [2,4] odstęp wynosi 2^{-51}.

Zadanie 4.

Zadanie polega na znalezieniu liczby x, spełniającej dane warunki:

- $x \epsilon (1, 2)$
- $x * \frac{1}{x} \neq 1$

Program ustawia początkową wartość zmiennej na 1, a następnie w pętli za każdym razem zwiększa ją o wartość funkcji nextFloat(Float64), która przy pierwszej iteracji jest równa wartości funkcji eps(Float64). Wynikiem programu i jednocześnie najmniejszą liczbą spełniającą warunki jest liczba 1.000000057228997

Zadanie 5.

Zadanie polega na obliczeniu iloczynu wektorowego dwóch wektorów w dwóch typach zmiennoprzecinkowych(single, double) zmieniając kolejność obliczeń.

Dokładna wartość: $-1{,}00657107000000 * 10^{-11}$

Sposób obliczenia	Float32	Float64
a)	1.0251881368296672e-10	1.0251881368296672e-10
b)	-0.4543457	-1.5643308870494366e-10
c)	-2.7554628740109736e6	-2.7554628740109736e6
d)	-0.4543457	-1.5643308870494366e-10

Z wyników można wywnioskować, że przy sumowaniu liczb bardzo bliskich albo bardzo odległych od siebie wyniki stają się coraz mniej wiarygodne. Można zauważyć też, że kolejność sumowania może zmienić ostateczny wynik oraz, że użycie typu większej precyzji nie oznacza otrzymanie wyniku o większej precyzji.

Zadanie 6.

Zadanie polega na policzeniu wartości dwóch funkcji:

$$f(x) = \sqrt{x^2+1} - 1$$
$$g(x) = x^2/(\sqrt{x^2+1} + 1)$$

dla różnych wartości x: liczby 8 podniesionej do potęgi -1, -2, -3 itd.

Wyniki:

x	f(x)	g(x)
8^{-1}	0.00012206286282867573	0.00012206286282875901
8^{-2}	0.3917915853514671	0.39179158535146696

8^{-3}	0.0088839708539672	0.008883970885396724
8^{-4}	0.2544844559308521	0.25448445593085206
8^{-5}	0.021198209620718167	0.02119820962071824
8^{-6}	0.19281172270387859	0.1928117227038785
8^{-7}	0.032918988140842265	0.03291898814084225
8^{-8}	0.15801588762035101	0.15801588762035088
8^{-9}	0.04314953425708756	0.04314953425708758
8^{-10}	0.13616806112927882	0.1361680611292788

Można zauważyć, że funkcja f podaje mniej dokładny wynik, ponieważ występuje tutaj odejmowanie dwóch prawie równych liczb, bo redukuje się znaczące cyfry, więc precyzja jest mała.

Zadanie 7.

Zadanie polega na obliczeniu pochodnej funkcji f(x) = sin x + cos 3x korzystając ze wzoru:

$$f'(x_0) \approx \widetilde{f}'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

dla różnych wartości h = 2^{-n}(n = 0, 1, 2, . . . , 54) w punkcie $x_0 = 1$.

n	Przybliżona wartość	Błąd obliczeniowy
0	2.0179892252685967	1.9010469435800585
4	0.3704000662035192	0.253457784514981
7	0.1484913953710958	0.03154911368255764
10	0.12088247681106168	0.0039401951225235265
13	0.11743474961076572	0.0004924679222275685
16	0.11700383928837255	6.155759983439424e-5
19	0.11694997636368498	7.694675146829866e-6
22	0.11694324295967817	9.612711400208696e-7
25	0.116942398250103	1.1656156484463054e-7
28	0.11694228649139404	4.802855890773117e-9
31	0.11694216728210449	1.1440643366000813e-7
34	0.11694145202636719	8.296621709646956e-7
37	0.1169281005859375	1.4181102600652196e-5
40	0.1168212890625	0.0001209926260381522
43	0.1162109375	0.0007313441885381522
46	0.109375	0.007567281688538152
49	0.125	0.008057718311461848
52	-0.5	0.6169422816885382

Przy wysokim h przybliżona wartość pochodnej funkcji obliczonej ze wzoru zbliża się od dwóch do zera, a od pewnego momentu jest równa 0. Wartość błędu obliczeniowego waha się w bliżej nieokreślony sposób -

błąd jest najmniejszy dla środkowych wartości h: <16, …, 47>, a przy skrajnych wartościach błąd jest największy.