LAB 1

Esercizio 1

Lo scopo di questo esercizio è quello di analizzare un problema che coinvolge l'aritmetica in virgola mobile e l'errore di arrotondamento associato alle operazioni matematiche.
L'idea a livello implementativo è quella di interpretare **a** come un array di lunghezza 7 che ci permette di contenere tutti i valori di **a** in base a **i**. In questo modo possiamo svolgere i calcoli all'interno di un ciclo for per ottenere tutti i risultati per diversi valori di **i.**
Per ogni calcolo del valore di **i**, è presente il calcolo della differenza tra il risultato 1 e il risultato 2 in modo da osservare quanto la precisione della macchina influisce sui risultati.

Problema
Quando sommiamo i numeri in virgola mobile, non possiamo sempre aspettarci che l'ordine delle operazioni non influenzi il risultato. In particolare, qui si verificano due ordini di somma:

1. (a + b) + c
2. a + (b + c)

La somma di numeri con ordini di grandezza molto diversi può causare problemi di precisione, e questo accade in entrambi i casi.

Richiami teorici visti a lezione :
Teoria dell'Aritmetica in Virgola Mobile
L'aritmetica in virgola mobile è una rappresentazione numerica utilizzata nei computer per gestire numeri reali. Tuttavia, essa presenta limitazioni di precisione. Le operazioni in virgola mobile non sempre rispettano le proprietà matematiche classiche, come l'associatività. Questo significa che l'ordine in cui eseguiamo le operazioni può influenzare il risultato finale.
Quando sommiamo numeri di grandezze molto diverse (ad esempio, un numero molto grande come **b** è un numero molto piccolo come **a**), possiamo incorrere in **errori di arrotondamento.**

Alcuni Esempi

1) i = 1

Dati

$a = 1, b = 10^{20}, c = -10^{20}$

L'errore di approssimazione avviene all'interno della parentesi (a + b) + c in quanto il computer approssima il calcolo $(1 + 10^{20})$ ignorando l'1, di conseguenza otteniamo come risultato 0.

Il computer decide di ignorarlo perché 1 è troppo piccolo per essere rappresentato con precisione nella somma con un numero così grande.
Mentre il calcolo di a + (b + c) usando gli stessi dati non presenta nessun tipo di errore di approssimazione in quanto la somma b + c si annulla prima, quindi **a** non viene influenzato e mantiene il suo valore originale.

2) i = 5

Dati

$$a = 10^5 , b = 10^{20} , c = -10^{20}$$

Come nel caso precedente in (a+b)+c, **a** è trascurato rispetto a **b**, e il risultato finale è 0 anche se **a** dovrebbe contribuire. Mentre in a+(b+c), la somma b + c è 0, e il risultato finale rimane **a = 100000**, senza errori di approssimazione.

Spiegazione Codice

Come spiegato all'inizio, abbiamo deciso di considerare **a** come un array di lunghezza 7. In questo modo, tramite un ciclo for, possiamo calcolare tutti i valori che **a** assume per ogni valore di **i**. Sarebbe stato possibile svolgere manualmente ogni singolo calcolo separatamente, dato che stiamo studiando casi diversi e quindi abbiamo valori di **a** diversi. Tuttavia, per comodità, abbiamo eseguito tutto all'interno di un unico ciclo.

Nel codice è presente un ulteriore ciclo for che mi permette di svolgere i calcoli delle due espressioni (a + b) + c , a + (b + c).

Ouput ottenuti:

```
Per i = 0:
(a+b)+c = 0
a+(b+c) = 1
Differenza: 1
--------------------
Per i = 1:
(a+b)+c = 0
a+(b+c) = 10
Differenza: 10
--------------------
Per i = 2:
(a+b)+c = 0
a+(b+c) = 100
Differenza: 100
--------------------
Per i = 3:
(a+b)+c = 0
a+(b+c) = 1000
Differenza: 1000
--------------------
Per i = 4:
(a+b)+c = 16384
a+(b+c) = 10000
Differenza: 6384
--------------------
Per i = 5:
(a+b)+c = 98304
a+(b+c) = 100000
Differenza: 1696
--------------------
Per i = 6:
(a+b)+c = 999424
a+(b+c) = 1e+06
Differenza: 576
--------------------
```

Esercizio 2

In questo esercizio dobbiamo confrontare il risultato di e^x con quello ottenuto tramite l'approssimazione del Polinomio di Taylor.
Le formule richieste per questo tipo di confronto sono:

- L'*errore assoluto* è la differenza fra valore ottenuto e valore atteso: $\delta = \tilde{x} - x$
- L'*errore relativo* invece è $\varepsilon = \frac{\tilde{x}-x}{x}$ ovvero $\tilde{x} = x(1+\varepsilon)$.

Spiegazione Codice

Abbiamo creato tre file:

- file .h
- Taylor.cpp
- main.cpp

Funzione fact
Questa funzione ci permette di calcolare il fattoriale di un numero passato come parametro, è uno degli esercizi che avevamo nell'eserciziario di IP durante il primo anno.

Funzione Taylor

Questa funzione contiene un ciclo for che mi permette di calcolare il termine $\frac{x^n}{N!}$.
Alla fine una volta sommato tutti i termini della serie (escluso il termine costante iniziale 1) la funzione restituisce *1 + value* in quanto la somma completa della serie di Taylor per e^x è:

$$e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \ .... + \frac{x^n}{N!}$$

Funzione algoritmo 1

Questa funzione ci permette di calcolare f_N(tramite la chiamata alla funzione Taylor) e f_n (usando exp grazie alla libreria math.h).
Successivamente calcoliamo errore assoluto e relativo tramite le formule indicate qui sopra.

Funzione algoritmo 2

Questa funzione è simile alla funzione algoritmo 1, ma al posto di considerare la f(x)=e^x si prende in considerazione la f(-x)=$\frac{1}{f(x)}$ che nel nostro caso è $\frac{1}{e^x}$ ovvero e^{-x}
e lo stesso vale per la funzione di Taylor infatti si va a considerare $\frac{1}{f_N(x)}$ al posto di f_N(x)

Ouput ottenuti:

```
Algoritmo 1:

x: 0.5, N: 3
fN: 1.64583, fx: 1.64872
Errore Assoluto: 0.00288794
Errore Relativo: 0.00175162
----------------------------------------
x: 0.5, N: 10
fN: 1.64872, fx: 1.64872
Errore Assoluto: 1.27627e-11
Errore Relativo: 7.74096e-12
----------------------------------------
x: 0.5, N: 50
fN: 1.64872, fx: 1.64872
Errore Assoluto: 0
Errore Relativo: 0
----------------------------------------
x: 0.5, N: 100
fN: 1.64872, fx: 1.64872
Errore Assoluto: 0
Errore Relativo: 0
----------------------------------------
x: 0.5, N: 150
fN: 1.64872, fx: 1.64872
Errore Assoluto: 0
Errore Relativo: 0
----------------------------------------
x: 30, N: 3
fN: 4981, fx: 1.06865e+13
Errore Assoluto: 1.06865e+13
Errore Relativo: 1
----------------------------------------
x: 30, N: 10
fN: 2.3883e+08, fx: 1.06865e+13
Errore Assoluto: 1.06862e+13
Errore Relativo: 0.999978
----------------------------------------
x: 30, N: 50
fN: 1.06833e+13, fx: 1.06865e+13
Errore Assoluto: 3.18471e+09
Errore Relativo: 0.000298013
----------------------------------------
x: 30, N: 100
fN: 1.06865e+13, fx: 1.06865e+13
Errore Assoluto: 0.00390625
Errore Relativo: 3.65532e-16
----------------------------------------
x: 30, N: 150
fN: 1.06865e+13, fx: 1.06865e+13
Errore Assoluto: 0.00390625
Errore Relativo: 3.65532e-16
----------------------------------------
x: -0.5, N: 3
fN: 0.604167, fx: 0.606531
Errore Assoluto: 0.00236399
Errore Relativo: 0.00389757
----------------------------------------
x: -0.5, N: 10
fN: 0.606531, fx: 0.606531
Errore Assoluto: 1.17418e-11
Errore Relativo: 1.9359e-11
----------------------------------------
x: -0.5, N: 50
fN: 0.606531, fx: 0.606531
Errore Assoluto: 2.22045e-16
Errore Relativo: 3.6609e-16
----------------------------------------
x: -0.5, N: 100
fN: 0.606531, fx: 0.606531
Errore Assoluto: 2.22045e-16
Errore Relativo: 3.6609e-16
----------------------------------------
x: -0.5, N: 150
fN: 0.606531, fx: 0.606531
Errore Assoluto: 2.22045e-16
Errore Relativo: 3.6609e-16
----------------------------------------
x: -30, N: 3
fN: -4079, fx: 9.35762e-14
Errore Assoluto: 4079
Errore Relativo: 4.35901e+16
----------------------------------------
x: -30, N: 10
fN: 1.21255e+08, fx: 9.35762e-14
Errore Assoluto: 1.21255e+08
Errore Relativo: 1.29579e+21
----------------------------------------
x: -30, N: 50
fN: 8.78229e+08, fx: 9.35762e-14
Errore Assoluto: 8.78229e+08
Errore Relativo: 9.38517e+21
----------------------------------------
x: -30, N: 100
fN: -4.82085e-06, fx: 9.35762e-14
Errore Assoluto: 4.82085e-06
Errore Relativo: 5.15179e+07
----------------------------------------
x: -30, N: 150
fN: -4.82086e-06, fx: 9.35762e-14
Errore Assoluto: 4.82086e-06
Errore Relativo: 5.1518e+07
----------------------------------------

Algoritmo 2:

fN(0.5) = 1.64583, f(-0.5) (reciproco) = 0.607595
Errore Assoluto: 0.00106428
Errore Relativo: 0.0017547
----------------------------------------
fN(0.5) = 1.64872, f(-0.5) (reciproco) = 0.606531
Errore Assoluto: 4.69513e-12
Errore Relativo: 7.74097e-12
----------------------------------------
fN(0.5) = 1.64872, f(-0.5) (reciproco) = 0.606531
Errore Assoluto: 0
Errore Relativo: 0
----------------------------------------
fN(0.5) = 1.64872, f(-0.5) (reciproco) = 0.606531
Errore Assoluto: 0
Errore Relativo: 0
----------------------------------------
fN(0.5) = 1.64872, f(-0.5) (reciproco) = 0.606531
Errore Assoluto: 0
Errore Relativo: 0
----------------------------------------
fN(30) = 4981, f(-30) (reciproco) = 0.000200763
Errore Assoluto: 0.000200763
Errore Relativo: 2.14545e+09
----------------------------------------
fN(30) = 2.3883e+08, f(-30) (reciproco) = 4.18709e-09
Errore Assoluto: 4.18699e-09
Errore Relativo: 44744.2
----------------------------------------
fN(30) = 1.06833e+13, f(-30) (reciproco) = 9.36041e-14
Errore Assoluto: 2.78952e-17
Errore Relativo: 0.000298102
----------------------------------------
fN(30) = 1.06865e+13, f(-30) (reciproco) = 9.35762e-14
Errore Assoluto: 3.78653e-29
Errore Relativo: 4.04647e-16
----------------------------------------
fN(30) = 1.06865e+13, f(-30) (reciproco) = 9.35762e-14
Errore Assoluto: 3.78653e-29
Errore Relativo: 4.04647e-16
----------------------------------------
```

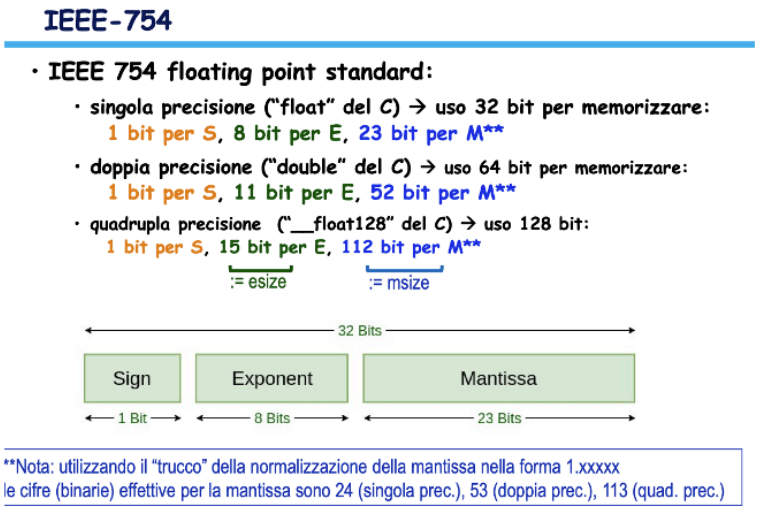
Esercizio 3

Definizione di precisione di macchina

La precisione di macchina ε descrive il livello di accuratezza con cui un computer può rappresentare i numeri in virgola mobile. Essa dipende dal numero di bit utilizzati per rappresentare la **mantissa** o **frazione** dei numeri nei formati di virgola mobile definiti dallo standard **IEEE 754**.

- In **singola precisione** (32 bit), vengono usati 23 bit per la mantissa.
- In **doppia precisione** (64 bit), vengono usati 52 bit per la mantissa.

Questo significa che, in singola precisione, la precisione di macchina sarà dell'ordine di 2^{-23} , mentre in doppia precisione sarà dell'ordine di 2^{-52}
.



Screenshoot preso dalle slide di ADC su aulaweb

Spiegazione codice

La precisione di macchina viene calcolata attraverso un ciclo while che incrementa l'esponente d finché la somma $1 + 2^{-d}$ è ancora maggiore di 1. Quando 2^{-d} diventa così piccolo da non influenzare il risultato, la somma risulta uguale a 1, indicando che abbiamo superato la precisione di macchina. Al termine del ciclo, per ottenere il valore di precisione di macchina, è sufficiente ridurre di 1 l'esponente d. In questo modo, otteniamo il valore massimo di 2^{-d} che, sommato a 1, produce ancora un risultato diverso da 1.

Ouput ottenuti:

```
Precisione singola: 1.19209e-07
Precisione doppia: 2.22045e-16
```