Anna Dellepiane (5565836), Marco Zoratti (5562866).

Relazione Labo 1 ALAN

Il codice del laboratorio è suddiviso in 3 parti: funzioni ausiliare, esercizi e main.

Il file .cpp può essere visionato al seguente link: Uni-Informatica/SecondoAnno/ALAN/labo1/labo.cpp at master · marchacio/Uni-Informatica (github.com)

Funzioni ausiliarie

Le seguenti funzioni sono state sviluppate ed utilizzate per semplificare e riutilizzare più volte il codice all'interno del programma:

- ***void stampaElemTabella* e *void stampaDivisoreTabella***
  Funzioni ausiliarie per migliorare l'output e renderlo più leggibile e facilitarne la comprensione.

- ***double fattoriale(double n)***
  Funzione ausiliaria per calcolare il fattoriale del numero passato come argomento.

- ***double f (double x)***
  Rinominazione della funzione esponenziale exp (e^x) ottenuta dalla libreria cmath.

- ***double fNx(double n, double x)***
  Funzione ricorsiva per ottenere il Polinomio di Taylor:
  $$fN\ (x) = \sum_{n=0}^{N} \frac{x^n}{n!}$$
  La ricorsività di questa funzione è dovuta alla presenza della sommatoria:
  - Il caso base della ricorsione avviene quando il grado del polinomio (n) è uguale a 0, infatti quando questa condizione è vera $x^0 = 1\ \forall x \in R\{0\}$ e $0! = 1$, ovvero $\frac{1}{1} = 1$.
  - Altrimenti il calcolo della frazione è seguito dalla chiamata ricorsiva della funzione con argomenti *n-1* e *x*.

Esercizio 1

Definiamo d0 e d1 come l'ultima e la penultima cifra del numero matricola 5565836, quindi rispettivamente 6 e 3.

Dato 0 < i <= 6:

- a = (d0+1) * 10^i
- b = (d1+1) * 10^{20}
- c = -b

Implementazione algoritmo

Una volta definito il numero di matricola come stringa, si ottiene il carattere corrispondente all'ultima e penultima cifra grazie all'operatore [].

In c++, il tipo di dato char fa riferimento alla codifica Unicode dove ogni carattere è definito da un numero minore di 2^8 che si può ottenere grazie al casting dinamico.

Inoltre in Unicode i numeri da 0 a 9 sono definiti sequenzialmente quindi per ottenere il numero intero rappresentato da un carattere basta sottrarre a quest'ultimo il numero rappresentato dal carattere '0'; in questo modo convertiamo i due caratteri del codice matricola in numeri interi.

Una volta ottenute le cifre, il programma esegue un ciclo for da 0 a 6 (compresi) ed esegue i calcoli.

Analisi algoritmica

Algoritmo 1: $(a + b) + c$

$s := (a + b)$

$y_1 := s + c$

$$\varepsilon_{y_1} = \frac{s}{s+c}\varepsilon_s$$

Quindi l'algoritmo è instabile quando $s \approx -c$, ovvero $(a + b) \approx -c$, in quanto comporterebbe una divisione per zero e quindi un incremento infinito dell'errore ε_s.

Questo caso si verifica finché *a* non diventa abbastanza grande da influenzare la somma *a + b* in modo considerevole e quindi rende questo algoritmo instabile.

Algoritmo 2: $a + (b + c)$

$s := (b + c)$

$y_1 := a + s$

$$\varepsilon_{y_2} = \frac{s}{a+s}\varepsilon_s$$

L'algoritmo è instabile quando $s \approx -a$ ma questo caso non può accadere perché *s* è la somma di due numeri discordi e quindi la loro somma vale 0.

Dato che $a \neq 0$ per definizione, l'algoritmo è stabile.

Output programma:

Numero matricola: 5565836; ultime due cifre: 6, 3.

i	Algoritmo 1: (a+b)+c	Algoritmo 2: a+(b+c)
0	0	7
1	0	70
2	0	700
3	0	7000
4	65536	70000
5	720896	700000
6	7.01235e+06	7e+06

Considerazioni

Come si può notare dai dati prodotti in output, il primo algoritmo è instabile mentre il secondo riporta i risultati corretti.
Questa differenza è dovuta alla precedenza di addizioni e cancellazioni.

In un primo momento, l'algoritmo 1 esegue una somma tra due numeri di ordine molto differente, *a e b*, fintantoché il valore di *i* è piccolo. Infatti, se *i < 3*, *a* è al massimo *n^3*, mentre *b* è sempre *m^20*.
L'influenza di *a* nella somma inizia a farsi sentire quando la differenza di ordine tra le due variabili è minore o uguale a 16 gradi. Pertanto, per i valori di $i \leq 3$, l'output dell'algoritmo è 0.
Quando *a* inizia a "modificare" *b*, in ogni caso è ancora troppo piccolo per renderlo significativamente diverso da *c*. Di conseguenza, si verifica una cancellazione tra *(a+b)* e *c* che fa degenerare i risultati.

Il secondo algoritmo invece restituisce sempre il risultato corretto a prescindere dal valore dell'indice *i*, in quanto la cancellazione viene sempre eseguita come prima operazione (*b + c*), evitando quindi l'amplificazione degli errori (questo perché una cancellazione amplifica tutti gli errori fatti fino al momento della sua esecuzione).

Esercizio 2

Implementazione algoritmo

L'algoritmo prende in input un valore *double x* utilizzato per effettuare i calcoli con *f* e *fN* definite precedentemente; Inoltre, è richiesto un parametro *bool reciproco* per gestire la stampa del reciproco di *fN(x)* (utile nella seconda parte dell'esercizio).

Dopo aver calcolato *f(x)*, l'algoritmo stampa l'intestazione della tabella e successivamente, per ogni $N \in \{3, 10, 50, 100, 150\}$ stampa i risultati di *fN(x)*, dell'*errore relativo*, *assoluto* e opzionalmente anche il reciproco di *fN(x)*.

Questa funzione viene eseguita nel *main* del programma numerose volte con parametri diversi per effettuare misurazioni e confronti tra i risultati ottenuti.

Output programma:

Per motivi di spazio l'output è stato inserito con l'ausilio di immagini ma può essere visionato per intero al file output.txt nel Repository Github.

Algoritmo 1

---> Valore x in input: 0.5:

output di f(x): 1.64872

N	fN(x)	Err Ass	Err Rel
3	1.64583	-0.00288794	-0.00175162
10	1.64872	-1.27627e-11	-7.74096e-12
50	1.64872	-4.44089e-16	-2.69354e-16
100	1.64872	-4.44089e-16	-2.69354e-16
150	1.64872	-4.44089e-16	-2.69354e-16

---> Valore x in input: 30:

output di f(x): 1.06865e+13

N	fN(x)	Err Ass	Err Rel
3	4981	-1.06865e+13	-1
10	2.3883e+08	-1.06862e+13	-0.999978
50	1.06833e+13	-3.18471e+09	-0.000298013
100	1.06865e+13	0.00390625	3.65532e-16
150	1.06865e+13	0.00390625	3.65532e-16

---> Valore x in input: -0.5:

output di f(x): 0.606531

N	fN(x)	Err Ass	Err Rel
3	0.604167	-0.00236399	-0.00389757
10	0.606531	1.17416e-11	1.93586e-11
50	0.606531	-1.11022e-16	-1.83045e-16
100	0.606531	-1.11022e-16	-1.83045e-16
150	0.606531	-1.11022e-16	-1.83045e-16

---> Valore x in input: -30:

output di f(x): 9.35762e-14

N	fN(x)	Err Ass	Err Rel
3	-4079	-4079	-4.35901e+16
10	1.21255e+08	1.21255e+08	1.29579e+21
50	8.78229e+08	8.78229e+08	9.38517e+21
100	-4.82085e-06	-4.82085e-06	-5.15179e+07
150	-4.82086e-06	-4.82086e-06	-5.1518e+07

Algoritmo 2

```
ALGORITMO 2:
---> Valore x in input: 0.5:

output di f(x): 1.64872

N               fN(x)           Err Ass         Err Rel         Rec
----------------------------------------------------------------------------
3               1.64583         -0.00288794     -0.00175162     0.607595
10              1.64872         -1.27627e-11    -7.74096e-12    0.606531
50              1.64872         -4.44089e-16    -2.69354e-16    0.606531
100             1.64872         -4.44089e-16    -2.69354e-16    0.606531
150             1.64872         -4.44089e-16    -2.69354e-16    0.606531


---> Valore x in input: 30:

output di f(x): 1.06865e+13

N               fN(x)           Err Ass         Err Rel         Rec
----------------------------------------------------------------------------
3               4981            -1.06865e+13    -1              0.000200763
10              2.3883e+08      -1.06862e+13    -0.999978       4.18709e-09
50              1.06833e+13     -3.18471e+09    -0.000298013    9.36041e-14
100             1.06865e+13     0.00390625      3.65532e-16     9.35762e-14
150             1.06865e+13     0.00390625      3.65532e-16     9.35762e-14
```

Considerazioni

Come si può notare dai dati prodotti dall'algoritmo 1, *fN(x)* ha una precisione linearmente dipendente da *N*, infatti più *N* è grande più il risultato della funzione è preciso e simile ad *f(x)*;
Di conseguenza gli errori assoluti e relativi diminuiscono e tendono al valore della precisione di macchina ε, che nel caso migliore viene raggiunto (si veda l'output con *x = -0.5*).

Analizzando l'output di *fN(x)* con *x = -30* si nota come i risultati siano notevolmente diversi da *f(x)* e tra loro: questi risultati sono figli di cancellazioni svolte nel calcolo della sommatoria di *fN* dovute al segno negativo di x: la ripetuta somma di valori simili ma di segno opposto dà luogo a numerose cancellazioni che fanno degenerare il calcolo.
Per ottenere il risultato corretto si può optare per seguire la relazione della funzione esponenziale:

$$f(-x) = \frac{1}{f(x)} \rightarrow f(-x) \approx \frac{1}{fN(x)} \leftrightarrow f(x) \approx \frac{1}{fN(-x)}$$

Infatti, il reciproco di *fN(30)* è *9.35762e-14* $\approx$ *f(-30).*
La differenza tra fn(-30) e il reciproco di fN(30) è dovuta al fatto che la prima esegue N-1 somme tra numeri simili e con segno opposto mente la seconda esegue una divisione sul risultato, più stabile e con meno probabilità di errore.
Lo stesso ragionamento può essere applicato ad *x = -0.5*:

$$f(-0.5) \approx \frac{1}{fN(0.5)}$$

Esercizio 3

Implementazione di un programma che determini la precisione di macchina in singola e doppia precisione.

Implementazione algoritmo

Una volta definita la variabile *float eps* inizializzata ad 1, esegue un ciclo while che la dimezza fino a quando non sarà abbastanza piccola da poter essere semplificata dal calcolatore.

Lo stesso codice è stato utilizzato per la variabile in doppia precisione *double eps2 = 1*.

Output programma:

Precisione di macchina (eps) in singola precisione: 5.96046e-08

Precisione di macchina (eps) in doppia precisione: 1.11022e-16

Considerazioni

La *precisione di macchina* è definita come il più piccolo $\varepsilon \neq 0 \in FP$ (insieme dei numeri in virgola mobile, *Floating Points*) tale che la sua somma con un qualsiasi numero $n \in FP$ sia un numero diverso da n stesso.

In altre parole, essa rappresenta la precisione minima della macchina nei calcoli, che può variare in base alla componentistica hardware e software: infatti i dati riportati in output variano in base al pc utilizzato per la compilazione e l'esecuzione del programma.

Com'era prevedibile, il valore restituito dall'algoritmo per i tipi double è minore di quello restituito in singola precisione. Tale comportamento è dovuto al fatto che il numero di bit utilizzati per la rappresentazione di valori in singola precisione è minore rispetto a quelli utilizzati per la rappresentazione in doppia precisione. Quest'ultima ci permette quindi la memorizzazione di valori molto più precisi.

Ambiente di sviluppo, compilazione e altro

L'intero laboratorio è stato sviluppato tramite *Visual Studio Code* su macchina virtuale *Linux* (*WSL2, Ubuntu 22.04*) utilizzando il compilatore *g++* aggiornato alla versione *11.4.0* .

Per compilare, utilizzare il comando:

g++ -Wall labo.cpp