ERRORI

***AUTORI**:*	*Alberti Nicolo'*	*matr. 4086006*
	Canepa Andrea	*matr. 4185248*

***INDICE**:*

Laboratorio n° 1 di Calcolo Numerico
Anno accademico 2016/2017

Esercizio 1 – Proprieta' Associativa

Output:

```
Per i = 0:
(a + b) + c = 8
a + (b + c) = 7

Per i = 1:
(a + b) + c = 72
a + (b + c) = 70

Per i = 2:
(a + b) + c = 704
a + (b + c) = 700

Per i = 3:
(a + b) + c = 7000
a + (b + c) = 7000

Per i = 4:
(a + b) + c = 70000
a + (b + c) = 70000

Per i = 5:
(a + b) + c = 700000
a + (b + c) = 700000

Per i = 6:
(a + b) + c = 7e+06
a + (b + c) = 7e+06
```

/***/

Dato che l'addizione gode della proprieta' associativa i risultati dovrebbero essere uguali, ma, poiche' stiamo eseguendo i calcoli su un computer, essi verranno influenzati dall'ordine di grandezza degli operandi. Quando eseguiamo un'associazione da sinistra sommiamo un termine piccolo con uno molto grande (in termini di ordini di grandezza) percio' il risultato comincia ad essere attendibile solo quando i due ordini sono paragonabili: questo e' un caso in cui la cancellazione crea problemi al calcolatore.
D'altro canto eseguendo un'associazione da destra osserviamo che, essendo la cancellazione eseguita prima, il risultato avra' una maggior precisione; si noti anche che al crescere dell'esponente '*i*' la prima delle due somme assumera' un valore sempre piu' coerente rispetto a quello che ci aspettiamo.

Esercizio 2 – Polinomio di Taylor

Output:

Valori di e^x calcolati con la funzione exp di cmath:

x = 0.5 e^x = 1.64872
x = 30 e^x = 1.06865e+13

x = -0.5 e^x = 0.606531
x = -30 e^x = 9.35762e-14

Per calcolare errore relativo ed errore assoluto nel seguito utilizzeremo le seguenti formule:
€[ass] = fn(x) - f(x)
€[rel] = |fn(x) - f(x)|/f(x)

Per N = 3:

---Algoritmo 1:

>> x = 0.5 fn(x) ~= 1.625 f(x) = 1.64872

€[ass] = -0.0237213
€[rel] = 0.0143877

>> x = 30 fn(x) ~= 481 f(x) = 1.06865e+13

€[ass] = -1.06865e+13
€[rel] = 1

>> x = -0.5 fn(x) ~= 0.625 f(x) = 0.606531

€[ass] = 0.0184693
€[rel] = 0.0304508

>> x = -30 fn(x) ~= 421 f(x) = 9.35762e-14

€[ass] = 421
€[rel] = 4.49901e+15

---Algoritmo 2:

>> x = -0.5 fn(x) ~= 0.615385 f(x) = 0.606531

€[ass] = 0.00885396
€[rel] = 0.0145977

>> x = -30 fn(x) ~= 0.002079 f(x) = 9.35762e-14

€[ass] = 0.002079
€[rel] = 2.22172e+10

Per N = 10:

---Algoritmo 1:

>> x = 0.5 fn(x) ~= 1.64872 f(x) = 1.64872

€[ass] = -2.81877e-10
€[rel] = 1.70967e-10

>> x = 30 fn(x) ~= 7.61064e+07 f(x) = 1.06865e+13

€[ass] = -1.06864e+13
€[rel] = 0.999993

>> x = -0.5 fn(x) ~= 0.606531 f(x) = 0.606531

€[ass] = -2.57373e-10
€[rel] = 4.24336e-10

>> x = -30 fn(x) ~= -4.14684e+07 f(x) = 9.35762e-14

€[ass] = -4.14684e+07
€[rel] = 4.43151e+20

---Algoritmo 2:

>> x = -0.5 fn(x) ~= 0.606531 f(x) = 0.606531

€[ass] = 1.03697e-10
€[rel] = 1.70967e-10

>> x = -30 fn(x) ~= 1.31395e-08 f(x) = 9.35762e-14

€[ass] = 1.31394e-08
€[rel] = 140414

Per N = 50:

---Algoritmo 1:

>> x = 0.5 fn(x) ~= 1.64872 f(x) = 1.64872

€[ass] = -4.44089e-16
€[rel] = 2.69354e-16

>> x = 30 fn(x) ~= 1.06809e+13 f(x) = 1.06865e+13

€[ass] = -5.54512e+09
€[rel] = 0.000518891

>> x = -0.5 fn(x) ~= 0.606531 f(x) = 0.606531

€[ass] = -1.11022e-16
€[rel] = 1.83045e-16

>> x = -30 fn(x) ~= -1.48218e+09 f(x) = 9.35762e-14

€[ass] = -1.48218e+09
€[rel] = 1.58393e+22

---Algoritmo 2:

>> x = -0.5 fn(x) ~= 0.606531 f(x) = 0.606531

€[ass] = 1.11022e-16
€[rel] = 1.83045e-16

>> x = -30 fn(x) ~= 9.36248e-14 f(x) = 9.35762e-14

€[ass] = 4.85811e-17
€[rel] = 0.000519161

Per N = 100:

---Algoritmo 1:

>> x = 0.5 fn(x) ~= 1.64872 f(x) = 1.64872

€[ass] = -4.44089e-16
€[rel] = 2.69354e-16

>> x = 30 fn(x) ~= 1.06865e+13 f(x) = 1.06865e+13

€[ass] = 0.00390625
€[rel] = 3.65532e-16

>> x = -0.5 fn(x) ~= 0.606531 f(x) = 0.606531

€[ass] = -1.11022e-16
€[rel] = 1.83045e-16

>> x = -30 fn(x) ~= -3.42134e-05 f(x) = 9.35762e-14

€[ass] = -3.42134e-05
€[rel] = 3.65621e+08

---Algoritmo 2:

>> x = -0.5 fn(x) ~= 0.606531 f(x) = 0.606531

€[ass] = 1.11022e-16
€[rel] = 1.83045e-16

>> x = -30 fn(x) ~= 9.35762e-14 f(x) = 9.35762e-14

€[ass] = 3.78653e-29
€[rel] = 4.04647e-16

Per N = 150:

---Algoritmo 1:

>> x = 0.5 fn(x) ~= 1.64872 f(x) = 1.64872

€[ass] = -4.44089e-16
€[rel] = 2.69354e-16

```
>> x = 30          fn(x) ~= 1.06865e+13        f(x) = 1.06865e+13

        €[ass] = 0.00390625
        €[rel] = 3.65532e-16

>> x = -0.5        fn(x) ~= 0.606531           f(x) = 0.606531

        €[ass] = -1.11022e-16
        €[rel] = 1.83045e-16

>> x = -30         fn(x) ~= -3.42134e-05       f(x) = 9.35762e-14

        €[ass] = -3.42134e-05
        €[rel] = 3.6562e+08
```

---Algoritmo 2:

```
>> x = -0.5        fn(x) ~= 0.606531           f(x) = 0.606531

        €[ass] = 1.11022e-16
        €[rel] = 1.83045e-16

>> x = -30         fn(x) ~= 9.35762e-14        f(x) = 9.35762e-14

        €[ass] = -3.78653e-29
        €[rel] = 4.04647e-16
```

/**/

L'output e' molto vario a seconda dell'input percio' lo analizzeremo caso per caso: indicheremo con N il grado della serie, con $f_N(x)$ il risultato perturbato e con f(x) il valore atteso.

-------> x = 0.5

elaborato dall'algoritmo 1:

Gia' per un N piccolo otteniamo un valore abbastanza accurato, entrambi gli errori sono piccoli e piu' sviluppiamo la serie (cioe' al crescere di N) piu' il valore di $f_N(x)$ si avvicina a quello di f(x) calcolato con la funzione *pow* della libreria *cmath* con conseguente diminuzione degli errori. Questo e' anche dovuto al fatto che l'algoritmo non prevede cancellazioni in faso di calcolo.

-------> x = 30.0

elaborato dall'algoritmo 1:

Questo e' un valore interessante: abbiamo notato che per piccoli valori di N l'algoritmo produce risultati che si discostano di molto dal valore atteso (cioe' quello prodotto dalla funzione *exp* di *cmath*) mentre per N maggiori o uguali a 100 il risultato inizia a essere accurato. Per N=3, $f_N(x)$ e' talmente piccolo rispetto al valore reale di f(x) che nel calcolo degli errori non viene neanche percepito: l'errore relativo infatti e' 1, cioe' il

risultato prodotto dall'algoritmo e' sbagliato al 100%. Per N=10 il discorso riguardo all'errore e' analogo alla situazione precedente, infatti l'ordine di grandezza di $f_N(x)$ e' molto minore (di circa 10^6) rispetto a quello di f(x). Per N=50 l'output subisce un miglioramento: benche' l'errore assoluto sia nell'ordine del miliardo, quello ottenuto non e' un pessimo risultato in quanto sia f(x) che $f_N(x)$ sono nell'ordine di 10^{13}. Per valori di N superiori l'errore relativo si avvicina alla precisione di macchina percio' il risultato di $f_N(x)$ e' circa uguale a f(x).

-------> x = -0.5

elaborato da entrambi gli algoritmi:

Questo valore viene calcolato con molta precisione sia dall'algoritmo 1 che dall'algoritmo 2, infatti per valori di N maggiori di 3 (di quelli presi in esame) l'approssimazione e' buona. Mentre viene computato il risultato avvengono delle cancellazioni, però, essendo x compreso tra -1 e 0, ogni volta che calcoliamo una potenza sempre crescente di x essa sara' un numero sempre piu' piccolo e vicino allo 0; ad un certo punto, quando l'ordine di grandezza di questo valore si avvicina alla precisione di macchina allora non verra' piu' percepito nei calcoli e questo non influisce sugli errori sia relativo che assoluto. Tuttavia per N = 3 il risultato subisce una perturbazione (errore) di un'ordine di grandezza inferiore rispetto a f(x), infine si noti che l'algoritmo 2, in questo caso, e' leggermente piu' preciso dell'algoritmo 1.

-------> x = -30.0

elaborato dall'algoritmo 1:

Questo algoritmo prevede molte cancellazioni: essendo $x < 0$ e, in particolare, $x < -1$ per valori contenuti di N gli errori tenderanno a crescere poiche' i termini calcolati con Taylor avranno il numeratore molto maggiore del denominatore e questo implica una grande approssimazione. A un certo punto, al crescere di N, il denominatore diventera' maggiore del numeratore e quindi questi valori tenderanno a 0; infatti entrambi gli errori inizieranno a diminuire notevolmente anche se $f_N(x)$ non riuscira' mai ad approssimare f(x) poiche' il risultato e' stato perturbato durante le prime iterazioni.

elaborato dall'algoritmo 2:

In questo caso, invece, l'algoritmo funziona abbastanza bene: i risultati sono sempre piu' accurati al crescere di N, come succedeva nel caso x = 30.0 proprio perche' lo sviluppo di Taylor viene calcolato con questo valore. Addirittura l'errore assoulto e' molto minore della precisione di macchina, questo ci suggerisce che per il calcolatore i due valori di f(x) e $f_N(x)$ sono esattamente la stessa cosa.

- Osservazione 1: l'errore assoluto ci indica, inoltre, la prima cifra che cambia nella soluzione perturbata rispetto al valore atteso (p.e. ε_{ass} = 0.1 vuol dire che la prima cifra che ci aspettiamo diversa e' quella corrispondente ai decimi).

- Osservazione 2: abbiamo utilizzato la doppia precisione per calcolare il fattoriale poiche' se usassimo variabili di tipo *int* o *float* a causa di problemi di approssimazione nella rappresentazione del numero il risultato per N grandi non sarebbe corretto.

Esercizio 3 – Precisione di Macchina

Output:

Il valore di d0 e' 23

Il valore di d1 e' 52

La precisione di macchina calcolata in singola precisione vale eps = 1.19209e-07
La precisione di macchina calcolata in doppia precisione vale eps = 2.22045e-16

/***/

L'esecuzione del programma e' avvenuta su un'architettura a *64 bit (x64)*.

Per calcolare la precisione di macchina in singola precisione abbiamo eseguito i calcoli dichiarando le variabili in *float* (23 bit di mantissa) mentre per la precisione doppia abbiamo utilizzato i *double* (52 bit di mantissa). Per calcolare con maggior precisione le potenze abbiamo utilizzato le funzioni *pow*, la quale lavora con numeri *double*, e *powf*, che utilizza i *float*, entrambe contenute nella libreria <*cmath*>.

Abbiamo osservato che e' molto importante eseguire un cast, cioe' una conversione di tipo, in *float* o *double* a seconda dei casi, altrimenti il computer esegue i calcoli su dei registri che potrebbero non essere quelli predisposti al tipo che vorremmo trattare (per es. se volessimo fare un calcolo in doppia precisione ma il numero può essere rappresentato con accuratezza anche in singola allora il calcolatore per risparmiare spazio converte i valori in *float*): forzando questo cast il risultato che otterremo sara' piu' preciso.

Un'altro fatto rilevante e' che $d_0 = 23$ (cosi' come $d_1 = 52$) sono i bit di mantissa che ha il registro *float* (*double*) per memorizzare i numeri in virgola mobile. Infatti 23 e' il piu' grande numero intero d tale che 2^{-23} e' memorizzabile all'interno di un registro float e quindi percepibile dal calcolatore. Da $d_0 = 24$ la quantita' 2^{-d} e' talmente piccola da non essere piu' sentita durante i calcoli.