Esercizi di verifica

3 – Approfondimento sui sistemi aritmetici di un computer: tipo numerico reale floating-point

P2_03_02_AT

12. [liv.1] Scrivere una *function C* per visualizzare la rappresentazione binaria (s,e,m) di un numero *float*. Verificare che il valore del numero ottenuto coincida con il dato iniziale.

13. [liv.3] Scrivere una *function C* di conversione di un numero reale da base 10 alla rappresentazione floating-point IEEE Std 754. L'input è una stringa di *char* del tipo `[±]X.Y` dove `X` e `Y` rappresentano rispettivamente la parte intera e la parte frazionaria del numero.

P2_03_03_AT

14. [liv.1] Scrivere delle *function C* per calcolare rispettivamente l'*epsilon macchina* del tipo `float`, del tipo `double` e del tipo `long double`, visualizzando ad ogni passo i singoli bit. Confrontare i risultati ottenuti con i valori delle variabili predefinite FLT_EPSILON, DBL_EPSILON e LDBL_EPSILON.

15. [liv. 3] Scrivere una *function C* per calcolare dalla definizione l'ULP(x) dove x è il parametro reale float di input.

16. [liv.2] Generando in modo random i bit* di un numero reale `x` (`double x`) [* vedere: uso di `rand()` in Materiale di supporto], determinare i bit della corrispondente rappresentazione float `flx` (`float flx; flx=(float) x`). Se il numero `x` è rappresentabile nel tipo `float`, calcolarne l'errore assoluto E_A e l'errore relativo E_R di rappresentazione (considerando come esatto `double x` e come approssimante `float flx`) dalle formule:

$$E_A = |\mathtt{x} - \mathtt{flx}|, \qquad E_R = \frac{|\mathtt{x} - \mathtt{flx}|}{|\mathtt{x}|}$$

P2_03_04_AT

Per gli esercizi che seguono, visualizzare l'errore di *roundoff* nel risultato di tipo `float` e stabilire se esso sia *di massima accuratezza* oppure no. L'errore di *roundoff* viene calcolato confrontando il risultato dell'algoritmo in aritmetica `float` con un valore di riferimento considerato "esatto": quest'ultimo è proprio la soluzione esatta del problema, quando questa sia nota, altrimenti il valore di riferimento viene calcolato mediante lo stesso algoritmo eseguito in una precisione maggiore (aritmetica `double` oppure `long double`).

17. [liv.2] Scrivere una *function C* per valutare un polinomio $P(x)$ in x_0 mediante *algoritmo di Horner*. Applicare l'algoritmo ai dati dell'esempio 3 nelle dispense calcolando l'errore relativo. Usare una versione dell'algoritmo a precisione estesa per ottenere il valore di riferimento.

18. [liv.1] Scrivere una *function C* per calcolare la somma di molti addendi a_k dello stesso ordine di grandezza

$$S = \sum_{k=1}^{N} a_k$$

mediante *algoritmo di somma a blocchi* (*Pairwise summation algorithm*) implementato in versione iterativa o ricorsiva (a scelta). Applicare l'algoritmo al seguente particolare *problema test* di cui è nota la soluzione (100):

$$N = 10^8, \quad a_k = 10^{-6}, \quad \forall k = 1, \ldots, N \qquad \Rightarrow \qquad \sum_{k=1}^{N} a_k = \sum_{k=1}^{10^8} 10^{-6} = 100$$

19. [liv.1] Scrivere una *function C* per calcolare iterativamente la somma $\sum_{k=1}^{n} \frac{x^k}{k!} \approx e^x$ con il *criterio di arresto naturale*.

20. [liv.1] Scrivere una *function C* per calcolare la somma di addendi ordinati; sommare gli addendi una volta in ordine crescente ed un'altra in ordine decrescente. Mostrare qual è il modo migliore di sommare in questo caso.

21. [liv.2] Scrivere una *function C* per calcolare la somma di addendi di segno alternato, evitando l'eventuale *cancellazione catastrofica*.