TECNICHE DI PROGRAMMAZIONE, A.A. 2021/2022

Esercitazione di Laboratorio 8

Valutazione: **l'esercizio 1** sarò oggetto di valutazione.
Scadenza: caricamento di quanto valutato - entro le 23:59 del 3/6/2022: andranno caricati insieme i laboratori 7 e 8.

Obiettivi

- Risolvere problemi di verifica/selezione/ordinamento, iterativi, utilizzando vettori e matrici (*Dal problema al programma: Cap. 4 e 5*),

Contenuti tecnici

- Basi di Input Output
- Utilizzo di funzioni
- Costrutti condizionali e iterativi
- Manipolazioni elementari di vettori e matrici (di int e float)

Da risolvere durante il laboratorio oppure prima/dopo il laboratorio stesso

Esercizio 1. (Esercizio da consegnare per il bonus-laboratorio)

Competenze: lettura/scrittura di file, manipolazioni di matrici; puntatori e passaggio di parametri per riferimento
Categoria: problemi di verifica e selezione (Dal problema al programma: 4.5) – Problemi complessi (Dal problema al programma: 5)

Individuazione di regioni

Re-implementare l'esercizio 1 del Lab7 apportandovi le seguenti modifiche:

- Supponendo di avere dichiarato nel main una matrice di interi `M` di dimensioni `NR x NC`, con `NR` e `NC` pari a 50, si acquisisca la matrice da un file di testo mediante una funzione (`leggiMatrice`) che restituisce il numero di righe e di colonne della matrice acquisita, come parametri "by reference" (o meglio, "by pointer"). La funzione deve avere il seguente prototipo:

  ```
  void leggiMatrice(int mat[][NC], int maxR, int *nrp, int *ncp);
  ```

  e deve essere chiamata nel main usando la seguente istruzione:

  ```
  leggiMatrice(M,NR,&nr,&nc);
  ```

  Come risultato della chiamata alla funzione, le variabili intere `nr` e `nc` dovrebbero contenere il numero effettivo di righe e colonne della matrice acquisita, `M`.
- Il riconoscimento delle regioni deve essere gestito da una funzione `riconosciRegione` che, data una casella della matrice identificata dagli indici di riga e colonna `r` e `c`, deve determinare se

questa casella rappresenti l'estremo superiore sinistro di una regione rettangolare. La funzione deve restituire un numero intero booleano come valore di ritorno: rispettivamente, `1` se la regione è stata trovata, `0` altrimenti. Anche le dimensioni (larghezza e altezza) della regione rettangolare devono essere restituite "by reference", come è stato fatto per nr e nc al punto precedente. La funzione deve essere chiamata come segue:

```
if (riconosciRegione(M,nr,nc,r,c,&b,&h)) {
  // stampa messaggio per rettangolo con
  // estremo in (r,c), base b e altezza h
  ...
}
```

Esercizio 2.

Competenze: puntatori, codifica dell'informazione, rappresentazioni numeriche
Categoria: il tipo di dato puntatore

Puntatori e rappresentazione dati

Si realizzi un programma che permetta di visualizzare la codifica interna (binaria) di un numero reale, realizzato in `C` sia con un dato di tipo `float` che di tipo `double`.
In `C` i tipi `float` e `double` (vedere la definizione su https://en.wikipedia.org/wiki/C_data_types) implementano le specifiche `IEEE-754` per i tipi di dati reali in precisione singola e doppia, rispettivamente. Dovresti ricordare lo standard IEEE per l'aritmetica floating-point (IEEE 754) dal corso di Informatica. Maggiori dettagli a questi link:
https://en.wikipedia.org/wiki/IEEE_754
https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/

Il programma in C deve:

- dichiarare 2 variabili per numeri reali (`af` e `ad`, rispettivamente di tipo `float` e `double`)
- determinare se il computer usa la codifica little Endian o big Endian ed in base a questo assegnare il valore 0 o 1 ad una variabile denominata `bigEndian` (rispettivamente, 1 se la codifica è big Endian, 0 altrimenti). Vedi la sezione `AIUTO` per alcuni suggerimenti su come eseguire questa operazione.
- visualizzare (mediante l'operatore `sizeof`) la dimensione (espressa in byte e in bit) delle due variabili `af` e `ad`
- acquisire da tastiera un numero decimale (con virgola e, eventualmente, esponente in base 10), assegnandolo alle due variabili `af` e `ad`
- chiamare la funzione `stampaCodifica`, stampare a schermo la rappresentazione interna del numero nelle due variabili `af` e `ad`.

La funzione `stampaCodifica` deve avere come prototipo:

```
void stampaCodifica (void *p, int size, int bigEndian);
```

e deve essere chiamata due volte (rispettivamente, sulla variabile di tipo float e di tipo double) nel modo seguente:

```
stampaCodifica((void *)&af,sizeof(af),bigEndian);
stampaCodifica((void *)&ad,sizeof(ad),bigEndian);
```

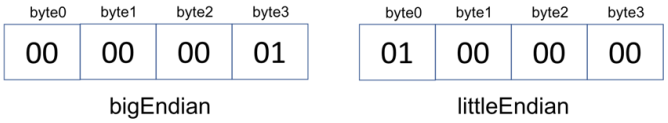
dove i tre parametri passati alla funzione sono: 1) il puntatore alla variabile (convertito ad un "generico" puntatore di tipo `void*` con una funzione di cast), 2) le dimensioni della variabile, e 3) la variabile flag `bigEndian`, che fornisce le informazioni riguardo l'endianness.

Applicando l'aritmetica dei puntatori, la funzione deve stampare a schermo la codifica binaria interna della variabile (rispettivamente: il bit di segno, i bit dell'esponente e i bit della mantissa).
Vedi la sezione seguente per alcuni suggerimenti.

AIUTO

Come verificare l'endianness del sistema?

Si può dichiarare una variabile intera `test`, inizializzata a un valore arbitrario (per esempio: il valore 1, che in base 16 è `0x00000001`). Assumendo che una variabile intera occupi 4 byte (lo si può verificare con `sizeof`), le codifiche big Endian e little Endian saranno:



dove byte0 indica la cella di 1 byte con l'indirizzo più basso. Quindi, è sufficiente verificare cosa è memorizzato nel byte0: la codifica è big Endian se il contenuto di byte0 è 0, little Endian se invece contiene 1.

Come fare per accedere ai singoli byte di una variabile intera? È necessario utilizzare l'aritmetica dei puntatori.
La variabile `test` è di tipo `int`, che è un dato di 4 byte. Di conseguenza, `&test` è un puntatore ad una variabile di tipo intero `int`, che può essere utilizzato unicamente per accedere al contenuto di un dato a 4 byte. Tuttavia, per gestire le celle di 1 byte, serve un puntatore che permetta di accedere ai singoli byte!
Il tipo di dato che in C ha dimensione 1 byte è il `char`. Quindi, è possibile dichiarare un puntatore ad una variabile di tipo char, `pchar`, ed inizializzare `pchar` con lo stesso indirizzo puntato da `&test`, nel modo seguente:

```
char *pchar = (char *)&test;
```

(si noti che la dichiarazione della variabile `pchar` richiede che i due puntatori abbiano lo stesso tipo: quindi, `&test` è convertita al tipo `char *` con un operatore di cast).

Facendo questo, si ottiene che la variabile `pchar` punta al primo byte della variabile intera `test` (byte0, nella figura), `pchar+1` punta al byte1, `pchar+2` al byte2, etc. Quindi, il contenuto del byte0 è `*pchar`, il contenuto del byte1 è `*(pchar+1)`, etc.

Suggerimenti su come risolvere gli esercizi

Si noti che NON si chiede di rappresentare, come numeri, l'esponente e la mantissa, ma solo di visualizzarne i bit. Pur essendo possibili varie soluzioni, si consiglia la strategia che segue:

- non potendo realizzare in `C` un vettore di "bit", si consiglia di leggere il numero reale come un vettore di `unsigned char`, dove un `unsigned char` in C è un valore di 8 bit non negativo, che varia tra 0 e 255. Ad esempio, una variabile `float` da 32 bit corrisponde a 4 byte e quindi ad un vettore di 4 `unsigned char`. Ogni elemento del vettore va poi decodificato mediante un algoritmo di conversione a binario (cioè, un algoritmo che riconosce i bit di un numero senza segno). In pratica, il puntatore `p` (di tipo `void *`) della funzione `stampaCodifica` andrà internamente assegnato a un puntatore a `unsigned char`

- occorre stampare i bit a partire dal più significativo (MSB), separando bit di segno, di esponente e di mantissa. In base al parametro `bigEndian`, è possibile conoscere la direzione in cui i byte del numero devono essere letti. Il parametro `size` serve per sapere dove terminano i bit di esponente ed iniziano quelli della mantissa. L'analisi dei byte può essere realizzata in maniera opportuna mediante l'utilizzo dell'aritmetica dei puntatori, in modo da percorrere tutti i byte del dato dal più significativo al meno significativo (o viceversa).