TECNICHE DI PROGRAMMAZIONE

Esercitazione di Laboratorio 9

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.

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

Si riveda l'esercizio 1 del Lab08 apportandovi le seguenti modifiche:

- supponendo di avere dichiarato una matrice di interi `M` e di aver definito `MAXR` come `50`, si acquisisca la matrice mediante una funzione (`leggiMatrice`) che ne ritorna il numero di righe e di colonne effettivamente usati, come parametri "by reference" (o meglio, "by pointer", cioè con puntatori by value). La funzione deve poter essere chiamata con un'istruzione del tipo:

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

- per effettuare il riconoscimento delle regioni si utilizzi una funzione `riconosciRegione` che, data una casella della matrice, determini se si tratti o meno di estremo superiore sinistro di una regione, ritornandone "by reference" (come per la precedente) le dimensioni del rettangolo, e avente come valore di ritorno un intero booleano (vero: rettangolo trovato, falso: rettangolo non trovato). La funzione deve poter 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 una funzione che permetta di visualizzare la codifica interna (binaria) di un numero reale, realizzato, in `C` da un `float` o `double`
Premessa: i tipi `C float` e `double` (se ne veda, ad esempio, la definizione su https://en.wikipedia.org/wiki/C_data_types) realizzano le specifiche IEEE-754 (https://it.wikipedia.org/wiki/IEEE_754) per i tipi di dato reali in precisione singola e doppia.

Il programma `C`:

- usa 2 variabili per numeri reali (`af`, `ad`, rispettivamente di tipo `float` e `double`)
- determina (utilizzando un numero intero, a scelta del programmatore) se il calcolatore utilizza la codifica little endian o big endian e assegna di conseguenza il valore vero o falso (come intero) a una variabile `bigEndian`
- visualizza (mediante l'operatore `C sizeof`) la dimensione (espressa in byte e in bit) delle due variabili `af`, `ad`
- acquisisce da tastiera un numero decimale (con virgola ed eventuale esponente in base 10), assegnandolo alle due variabili `af`, `ad`
- mediante la funzione `stampaCodifica` visualizza la rappresentazione interna del numero nelle due variabili `af`, `ad`.

La funzione `stampaCodifica`, avente prototipo:

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

va chiamata due volte, ricevendo come parametri, rispettivamente il puntatore a una della due variabili (convertito a `void *`) e la dimensione della variabile:

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

La funzione `stampaCodifica`, utilizzando l'aritmetica dei puntatori, la conoscenza del tipo di codifica e la dimensione ricevuta come parametro, deve stampare il bit di segno, i bit di esponente e i bit di mantissa del numero.

Suggerimenti e/o consigli

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 di ricavare la codifica binaria, nella funzione `stampaCodifica`, con la strategia che segue:

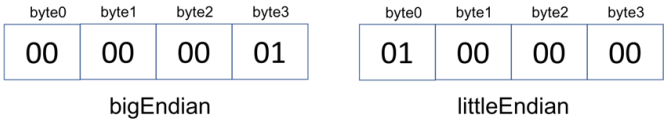
- non potendo realizzare in `C` un vettore di bit, si consiglia di leggere/riconoscere il numero ricevuto come vettore di `unsigned char` (ad esempio, un `float` da 32 bit corrisponde a un vettore di 4 `unsigned char`). Si usa l'`unsigned` per limitarsi a numeri senza segno (non negativi). Ogni elemento del vettore va poi decodificato mediante un algoritmo di conversione a

binario (riconoscimento dei bit di un numero senza segno). In pratica, il puntatore `p` (di tipo `void *`) andrà internamente assegnato a un puntatore a `unsigned char`

- occorre stampare i bit, separando bit di segno, di esponente e di mantissa, a partire dal più significativo (MSB). A seconda del parametro `bigEndian`, si stabilisce la direzione in cui percorrere i byte del numero. Il parametro `size` serve per sapere dove terminano i bit di esponente ed iniziano quelli della mantissa. Il percorso sui byte può essere realizzato mediante opportuno utilizzo dell'aritmetica dei puntatori, tale da percorrere tutti i byte del dato dal più significativo al meno significativo (o viceversa, a seconda della scelta fatta per la decodifica).

Come verificare l'endianness del sistema?

Si può dichiarare una variabile intera `test`, inizializzata a un valore arbitrario, tale da permettere di "riconoscere" l'endianness (ad esempio: il valore 1, che in esadecimale è `0x00000001`). Asumendo 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 "dove" si trovi il byte con valore 0x01, rispetto ai byte contenenti 0x00.

Come fare per accedere ai ("guardare" i) singoli byte di una variabile intera? E' sufficiente usare un opportuno puntatore, in quanto la variabile `test` di tipo `int`, è un dato di 4 byte, gestito in modo automatico, senza visione (esterna) sui singli byte. Di conseguenza, `&test` è puntatore a intero `int`, utilizzabile unicamente per accedere a un intero. Ma 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 si uò usare un puntatore a char, `pchar`, a cui si assegni l'indirizzo della variabile `test` (cioè `&test)`, come segue:

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

(si noti che il cast è necessario perchè i puntatori non sono dello stesso tipo).
A questo punto `pchar` punta al primo byte di `test` (byte0, in figura), `pchar+1` punta a byte1 (si ricorda che byte1 può essere visto indifferentemente come `pchar[1]` oppure come `*(pchar+1)`), e così per i byte successivi.