

Laboratorio di Programmazione: Linguaggio C

Lezione 13 del 23 gennaio 2014

Damiano Macedonio

Esercizio 1

Scrivere un programma C composto da:

- Una funzione `leggi` che richiede all'utente dei numeri interi non negativi e li memorizza in un array.
- Una funzione `massimo` che calcola il valore massimo dei valori inseriti e lo restituisce.
- Una funzione `minimo` che calcola il valore minimo dei valori inseriti e lo restituisce.
- Una funzione `media` che calcola la media dei valori inseriti e la restituisce.
- Una funzione `moda` che calcola la moda dei valori inseriti. Per definizione, la moda è il valore che compare più frequentemente. In caso ci siano più numeri con la stessa frequenza, è sufficiente che la funzione `moda` ritorni uno qualsiasi di essi.
- Una funzione `mediana` che calcola la mediana dei valori inseriti. Per definizione, dato un array di n elementi *ordinati*, la mediana è il valore centrale:
 - se n è dispari la mediana è il valore che occupa la posizione $(n + 1)/2$;
 - se n è pari la mediana è la media dei valori che occupano la posizione $n/2$ e $(n/2) + 1$.
- Una funzione `main` che richiama le altre funzioni e stampa i risultati ottenuti.

Definire funzioni parametriche nella dimensione dell'array.

Esercizio 2

Scrivere un programma C composto da:

- una funzione `void leggi(int [4][5])` che richiede all'utente di inserire una matrice 4×5 di interi;
- una funzione `void trasposta(int a[4][5], int b[5][4])` che riceve i parametri: `a` (matrice di dimensione 4×5) e `b` (matrice di dimensione 5×4).

La funzione calcola la trasposta di `a` e la memorizza in `b`.

Per definizione, una matrice `a` con `h` righe e `k` colonne può essere trasposta in una matrice `b` con `k` righe e `h` colonne, dove `b[j][i] = a[i][j]` per tutti gli indici `i` e `j` validi.

Esempio

Se `a` è la matrice

```
1  2  3  4  5
6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
```

allora `b` diventa

```
1  6 11 16
2  7 12 17
3  8 13 18
4  9 14 19
5 10 15 20
```

Esercizio 3

Si scriva una funzione `partiziona` che riceve un array di interi e lo modifica in modo che *tutti i numeri dispari precedano i numeri pari*. Si scriva quindi una funzione `main` che testa la funzione chiedendo all'utente i valori per riempire un array, passa tale array alla funzione `partizione` e lo stampa dopo la sua esecuzione.

Esempio

```
$ ./a.out
```

```
Inserire 10 valori interi: 59 26 53 22 31 41 80 87 78 37
```

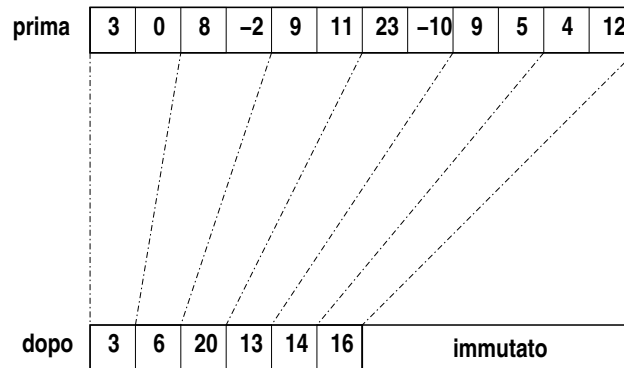
```
Ecco i valori raggruppati: 53 59 41 31 87 37 22 26 80 78
```

Esercizio 4 [Tratto dalla prova parziale del 4 febbraio 2013]

Si scriva una funzione

```
void pack(int array[], int length)
```

che riceve come parametri un array di lunghezza pari `length` e modifica l'array in modo che la sua prima metà diventi la somma degli elementi dell'array, due a due, come nel seguente disegno:



Ne caso che `length` sia dispari, l'ultimo valore deve essere semplicemente ricopiato nella cella `length / 2` (parte intera).

Esempio

L'esecuzione del seguente main:

```
int main(void) {
    int a[] = { 3, 0, 8, -2, 9, 11, 23, -10, 9, 5, 4, 12 };
    int b[] = { 3, 0, 8, -2, 9, 11, 23, -10, 9, 5, 4, 12, 26 };
    int pos;

    pack(a, 12);
    pack(b, 13);

    for (pos = 0; pos < 6; pos++)
        printf("%i ", a[pos]);
    printf("\n");

    for (pos = 0; pos < 7; pos++)
        printf("%i ", b[pos]);
    printf("\n");

    return 0;
}
```

deve stampare:

```
3 6 20 13 14 16
3 6 20 13 14 16 26
```

Esercizio 5 [Insertion Sort]

Si scriva una funzione

```
void insertion_sort(int array[], int  )
```

che implementa l'algoritmo descritto nel seguito.

Insertion sort è un algoritmo relativamente semplice per ordinare un array. Simula il modo in cui un essere umano, spesso, ordina un mazzo di carte. L'idea dell'algoritmo è la seguente.

Consideriamo l'array **a** di dimensione **size**, alla **i**-esima iterazione:

- l'array **a** è partizionato in due sottoarray: uno (dall'indice 0 all'indice **i-1**) già ordinato e uno (dall'indice **i-1** all'indice **size - 1**) ancora da ordinare;
- l'elemento **a[i]** viene rimosso dal sottoarray non ordinato e inserito nella posizione corretta del sottoarray ordinato. In tal modo, il sottoarray ordinato viene esteso di un elemento.
- Per inserire **a[i]** nella giusta posizione nel sottoarray ordinato, l'algoritmo utilizza un indice **j**:
 - inizialmente **j** viene posto a **i**;
 - vengono confrontati gli elementi **a[j-1]** e **a[j]**;
 - se **a[j]** è minore di **a[j-1]**, i due elementi vengono scambiati di posto e l'indice **j** viene decrementato di 1; altrimenti l'iterazione finisce.

Esempio. Quella che segue è un esempio di traccia di esecuzione dell'algoritmo.

```
[ 8, 5, 2, 34, 25, 3, 11 ] i = 0
[ 8, 5, 2, 34, 25, 3, 11 ] i = 1
[ 5, 8, 2, 34, 25, 3, 11 ]
[ 5, 8, 2, 34, 25, 3, 11 ] i = 2
[ 5, 2, 8, 34, 25, 3, 11 ]
[ 2, 5, 8, 34, 25, 3, 11 ]
[ 2, 5, 8, 34, 25, 3, 11 ] i = 3
[ 2, 5, 8, 34, 25, 3, 11 ] i = 4
[ 2, 5, 8, 25, 34, 3, 11 ]
[ 2, 5, 8, 25, 34, 3, 11 ] i = 5
[ 2, 5, 8, 25, 3, 34, 11 ]
[ 2, 5, 8, 3, 25, 34, 11 ]
[ 2, 5, 3, 8, 25, 34, 11 ]
[ 2, 3, 5, 8, 25, 34, 11 ]
[ 2, 3, 5, 8, 25, 34, 11 ] i = 6
[ 2, 3, 5, 8, 25, 11, 34 ]
[ 2, 3, 5, 8, 11, 25, 34 ]
```