Esercizi
Programmazione I e Informatica II

3 Dicembre 2015

Esercizio 1 Valutazione espressioni con puntatori

Valutare e definire il tipo delle seguenti espressioni. Inserire le parentesi per sotolineare l'ordine della valutazione in base alla priorità degli operatori.

```
int i = 3, j = 5, *p = &i, *q = &j, *r;
double x;

p == & i;
* * & p;
r = &x;
7 * * p / * q + 7;
* (r = & j) *= *p;

```

Soluzione dell'esercizio 1

Le espressioni sono tutte di tipo intero:

- $p == (\&i)$, valore 1
- $^*(^*(\&p))$, valore 3
- $r = (\&x)$, il compilatore restituisce un *warning*, conversione implicita da * *double* a *int* *
- $(((7^*(^*p)))/(^*q) + 7$, valore 11
- $(^*(r = (\&j)))* = (^*p)$, valore 15

Ricordarsi $^*r$ è un *lvalue* dato che è equivalente alla variabile a cui punta, cioè j.

Esercizio 2 Assegnamenti legali e illegali con i puntatori

Valutare quali dei seguenti assegnamenti sono legali (nessun errore/warning) e quali sono illegali.

```
int *p;
float *q;
void *v;

p = 0;
p = 1;
p = (int*) 1;
p = v = q;
v = 1;
p = q;
p = (int*) p;

```

Soluzione dell'esercizio 2

- Legale (a differenza dell'esempio sotto, 0 è un intero che può essere assegnato ad un puntatore, per esempio, anche *NULL* è definito come 0 in *stdlib.h*).
- Illegale (da *int* a **int*)

- Legale
- Legale
- Illegale (da *int* a **void*)
- Illegale (da **float* a **int*)
- Legale

Esercizio 3 Chiamata per riferimento

Scrivere una funzione *swap* che scambia i valori di due variabili di tipo *int*. Testare la funzione chiamandola da *main()*, su due variabili *int* $i = 3$ e *int* $j = 5$. Stampare infine le due variabili i e j per controllare che il loro valore sia scambiato. Scrivere le due funzioni nell0 stesso file *mainfile.c*.

Come ulteriore esercizio, spostare la definizione della funzione *swap* in un altro file *myswaplib.c*. Includere la dichiarazione della funzione nel file *myswaplib.h*, ed importarlo *#include "myswaplib.h"*. Fare riferimenti all'esercizio 4 del 28 Ottobre. Compilare separatamente i due file: `gcc -c myswaplib.c` e `gcc -c mainfile.c`. Infine linkare i due file `gcc -o mainfile mainfile.o myswaplib.o` per ottenere un solo eseguibile.

Soluzione dell'esercizio 3

```c
#include <stdio.h>

void swap(int *p, int *q)
{
  int tmp;
  tmp = *p;
  *p = *q;
  *q = tmp;
}


int main(void)
{
  int i =3, j =5;
  swap(&i, &j);
  printf("%d %d\n", i, j);
  return 0;
}

```

Esercizio 4 Chiamata per riferimento

Scrivere cosa stampa il seguente programma.

```c
 #include <stdio.h>

 int main() {
 double a[2]= {2.3, 4.7} , *p= NULL, *q= NULL;
 int *r= NULL;
 int res= 0;

 p= a;
 q = p + 1;
 printf("%ld\n", q - p);
 printf("%d\n" , (int) q - (int) p);
 printf("%d\n", q == &a[1]); // A cosa punta q quindi?

 r= (int*) a + 1;  // Se non si sa cosa stampa in questo caso dire dove punta r
 printf("%d\n", *r);
 printf("%d\n", (int) r - (int) a);
 printf("%f\n", *p);

 return 0;
 }

```

Soluzione dell'esercizio 4

Ecco cosa stampa:

```
1
8
1
1073899110
4
2.300000
```

Il valore di $q - p$ è il numero di elementi di distanza nell'array tra l'elemento puntato da p e quello puntato da q. $(int)q - (int)p$ è invece la distanza in byte tra di essi: in questo caso, essendo un array di *double* la distanza è 8 byte, visto che ogni elemento *double* dell'array occupa 8 byte.

Nel quarto caso non è facile dire cosa stampa. Con quell'operazione vengono letti i secondi 4 byte del primo elemento dell'array (un tipo *double* in C occupa 8 byte), e si stampano come se fossero un valore di tipo *int*, come mostra il quinto valore stampato.

Esercizio 5 Matrici 1

Data una matrice `int a[3][5]` e due cicli *for* indicizzati da i (righe) e j (colonne), dire quali delle seguenti espressioni è corretta per recuperare l'elemento `a[i][j]` (cioè, dire quali di queste espressionisono equivalenti ad essa):

```
*(a[i] + j)
(*(a + i))[j]
*((*(a + i)) + j)
*(&a[0][0] + 5*i + j)
```

Soluzione dell'esercizio 5

Sono tutte e quivalenti a `a[i][j]`.

Esercizio 6 Matrici 2

Dichiarare una matrice di *int* di dimensione 4×4 e inizializzarla con i valori da 1 a 16. Utilizzare due cicli *for* per stampare tutta la matrice, stampando tutti gli elementi di una riga separati da uno spazio e poi andando a capo per ogni nuova riga:

```
Riga 1
1 2 3 4
Riga 2
5 6 7 8
Riga 3
9 10 11 12
Riga 4
13 14 15 16
```

Soluzione dell'esercizio 6

```c
#include <stdio.h>
int main() {
  const int rows= 4, cols= 4;


  int matrix[rows][cols] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};

  for (int i= 0; i < rows; i++) {
    printf("Riga %d\n", i+1);
    for (int j= 0; j < cols; j++)
      printf("%d ", matrix[i][j]);
    puts("");
  }

  return 0;
}

```

Esercizio 7 Matrici 3

Nella funzione *main()* prendere da tastiera (*scanf()*) il numero di righe e colonne di una matrice. Sempre dentro *main()* chiamare una funzione *int** mallocMatrix(int, int)* che crea alloca con *malloc()* una matrice con un numero di righe e colonne passato per parametro (ritorna il puntatore alla matrice). Chiamare poi una funzione *void inputMatrix(int**, int, int)*, passando il puntatore alla matrice ed il numero di righe e colonne; tale funzione legge da tastiera (*scanf()*) tutti i valori della matrice (utilizzare due cicli *for*). Infine chiamare una funzione *void printMatrix(int**, int, int)*, che stampa tutti i valori della matrice. Un esempio dell output e dell'input è dato qui sotto. Scrivere tutto il codice in un solo file (le definizioni di tutte e tre le funzioni e la funzione *main()*). Infine, come nell'esercizio 3, spostare le definizioni delle tre funzioni *mallocMatrix()*, *inputMatrix()* e *printMatrix()* in un file *mylib.c* separato, e importare le dichiarazioni delle tre funzioni utilizzando *mylib.h*.

```
Dammi il numero di righe della matrice: 2
Dammi il numero di colonne della matrice: 3
Dammi elemento [1],[1]: 1
Dammi elemento [1],[2]: 2
Dammi elemento [1],[3]: 3
Dammi elemento [2],[1]: 4
Dammi elemento [2],[2]: 5
Dammi elemento [2],[3]: 6
Riga 1
1 2 3
Riga 2
4 5 6
```

Soluzione dell'esercizio 7

```c
#include <stdio.h>
#include <stdlib.h>


int** mallocMatrix(int rows, int cols) {

  int **mat = (int **) malloc(rows * sizeof(int*));

  for(int i = 0; i < rows; i++)
    mat[i] = (int *)malloc(cols * sizeof(int));

  return mat;
}



void inputMatrix(int ** matrix, int rows, int cols) {

  for (int i= 0; i < rows; i++)
    for (int j= 0; j < cols; j++) {
      printf("Dammi elemento [%d],[%d]: ", i+1, j+1);
      scanf("%d", &matrix[i][j]);
  }

  return;
}

void printMatrix(int ** matrix, int rows, int cols) {

  for (int i= 0; i < rows; i++) {
    printf("Riga %d\n", i+1);
    for (int j= 0; j < cols; j++)
      printf("%d ", matrix[i][j]);
    puts("");
  }

  return;
}

```

```c
int main() {
  int rows= 0;
  int cols = 0;

  printf("Dammi il numero di righe della matrice: ");
  scanf("%d", &rows);
  printf("Dammi il numero di colonne della matrice: ");
  scanf("%d", &cols);

  int** matrix= mallocMatrix(rows, cols);
  inputMatrix(matrix, rows, cols);
  printMatrix(matrix, rows, cols);

  free(matrix);

  return 0;
}
```

Esercizio 8 Equazioni di secondo grado

Si realizzi un programma in linguaggio C per risolvere equazioni di secondo grado. In particolare, data una generica equazione di secondo grado nella forma

$$ax^2 + bx + c = 0$$

dove a, b, c sono coefficienti reali noti e x rappresenta l'incognita, il programma determini le due radici x_1 ed x_2 dellequazione data, ove esse esistano. Si identifichino tutti i casi particolari ($a = 0$, $\Delta \leq 0$, ...) e si stampino gli opportuni messaggi informativi.