Un pò di puntatori

Un puntatore è una variabile che memorizza un indirizzo in memoria.

```
...
int x;
int * x_ptr;

x = 10;
x_ptr = &x;
...
```

Come funzionano?

La variabile `x` è memorizzata nella cella di memoria con indirizzo 12000 e il suo contenuto è il numero intero 10. La variabile `x_ptr` è memorizzata nella cella di memoria con indirizzo 12050 e il suo contenuto è il valore 12000, ovvero l'indirizzo della variabile `x`.

* ed &

Il simbolo * serve a definire una variabile di tipo puntatore, in combinazione al tipo di dati puntati:

```
int * x_ptr, x;
char * c_ptr, c;
float * f_ptr, f;
...
```

Per ogni tipo di dati, è possibile definire variabili del corrispondente tipo puntatore, così `x_ptr` sarà un puntatore ad `int`, `c_ptr` sarà un puntatore a `char` e `f_ptr` sarà un puntatore a `float`, mentre `x`, `c` e `f` saranno rispettivamente delle varibili di tipo `int`, `char` e `float`.

* ed &

L'operatore `&` restituisce l'indirizzo della variabile che lo segue. `& x` restituisce l'indirizzo di memoria della variabile intera `x`. Pertanto

```
x_ptr = & x;
```

assegna alla variabile di tipo puntatore ad intero `x_ptr` l'indirizzo di memoria della variabile `x`.
L'operatore `*` permette di accedere al contenuto della cella di memoria puntata da una variabile di tipo puntatore.

Esempio

Se eseguiamo questo programma

```
...
int * x_ptr, x;
x = 10;
x_ptr = &x;
printf ("L'indirizzo di x e' %p, il suo valore e' %d\n",
          &x, x);
printf ("Il valore di x_ptr e' %p, il valore puntato da"
" x_ptr e' %d\n",
 x_ptr, *x_ptr);
...
```

otteniamo

```
L'indirizzo di x e' 0x7fff441b352c, il suo valore e' 10
Il valore di x_ptr e' 0x7fff441b352c, il valore puntato
   da x_ptr e' 10
```

A che servono i puntatori?

Ad esempio, ci permettono di modificare il valore di una variabile tramite una funzione!

```
void incrementa (int * val){
  (*val)++;
}
```

Come chiamiamo la funzione?

```
int x = 0, * x;
while (x < 10) {
  printf("%2d", x);
  incrementa (&x);
}
printf("\n");
int x = 0;
x_ptr = &x;
while (x < 10) {
  printf("%2d", *x_ptr);
  incrementa (x_ptr);
}
```

Come chiamiamo la funzione?

```
int x = 0, * x;
while (x < 10) {
  printf("%2d", x);
  incrementa (&x);
}
printf("\n");
int x = 0;
x_ptr = &x;
while (x < 10) {
  printf("%2d", *x_ptr);
  incrementa (x_ptr);
}
```

stampa

```
 0 1 2 3 4 5 6 7 8 9
 0 1 2 3 4 5 6 7 8 9
```

Esercizio

Scrivere una funzione che scambi il valore di due variabili intere ricevute in input.

Swap di due valori

```
void int_swap (int * e1, int * e2){
  static int t;
  t = *e1;
  *e1 = *e2;
  *e2 = t;
}
```

E con i vettori?

Se definiamo un vettore "`char stringa[50]`" sappiamo che le locazioni in memoria delle celle del vettore sono in sequenza. `stringa` diventa una variabile di tipo puntatore a `char`

```
...
char stringa[50] = "Aritmetica di puntatori";
int i;
char * stringa_ptr;
printf ("L'indirizzo di stringa e' %p\n", stringa);
stringa_ptr = &stringa[0];
printf ("Stampiamo la stringa usando un puntatore"
        " a char:\n");
while ( *stringa_ptr != '\0')
  printf ("%c", *(stringa_ptr++));
printf("\n");
...
```

Nel file `aritmetica_puntatori.c` trovate altri esempi.

Popolamento di un vettore

Innanzitutto, facciamo in modo di poter popolare vettori con dei valori casuali per non dover digitare troppi numeri ;)

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

void randomize ( int vect [], int size, int range) {
  static int i;
  for(i=0; i < size; i++)
    vect [ i ] = rand() % range;
}
```

La funzione `int rand( void )`, definita in stdlib.h, restituisce un numero pseudocasuale. Per inizializzare il tutto si deve fornire un seme (seed) chiamando la funzione `srand( unsigned )`. I numeri restituiti dalle chiamate a `rand`, sono in una sequenza che dipende dal seed fornito (a parità di seed, la sequanza rimane la stessa ad ogni esecuzione).

Stampa di un vettore

Poi scriviamo una funzioncina che lo stampi

```
void print_vect(  int vect [], int size){
  static int i;
  for (i=0; i < size; i++){
    printf("%3d", vect [ i ]);
  }
  printf("\n");
}
```

“Esercizio”

Scrivere un programma C che popoli un vettore di interi di 20 elementi con valori casuali fra 0 e 99. Usare una costante simbolica per definire il numero di elementi.

Ordinamento di vettori

Ora che abbiamo un programma in grado di popolare e stampare vettori di interi, vediamo come possiamo fare per ordinare questi vettori.
Vedremo ~~(se facciamo in tempo)~~ il Selection Sort~~, l'Insertion Sort e il Bubble Sort~~.

Selection Sort

```
per i=1 ad n-1
  trova il valore minimo a partire dall'indice i
  scambia v[i] con il minimo
```

Esempio: