Introduzione ai puntatori in C

Definizione

- Un puntatore è una variabile che contiene l'indirizzo di un'altra variabile
- Tramite i puntatori si può quindi accedere a un oggetto indirettamente (si usa il suo indirizzo contenuto nel puntatore)

Perché i puntatori

- Problemi:
  - Realizzare funzioni che **modificano** il valore dei parametri che ricevono in ingresso (ovvero realizzare il *passaggio per riferimento)*
  - Gestire insiemi di dati di dimensione non nota a priori

Esempio: Passaggio parametri

- Esempio di funzione che vorremmo fosse in grado di agire sui parametri che vengono passati:

```
void scambia(int a, int b);
```

- Vorremmo che questa funzione scambiasse i valori delle variabili `a` e `b`
  - Col passaggio per valore, se anche li scambiassimo all'interno della funzione, la modifica da fuori non sarebbe visibile

Esempio: Dimensione non nota

- Immaginiamo di dover gestire una sequenza di dati, la cui lunghezza non sia nota a priori
- Quando dichiariamo un array, questo occupa in memoria lo stesso spazio
  - Dimensionarlo in base alla massima dimensione porta a degli sprechi di utilizzo della memoria
  - Se la dimensione che gli diamo non è sufficiente non siamo in grado di gestire tutti i dati

Puntatori

- Il C risponde a queste esigenze consentendo l'uso dei puntatori
- Non è l'unica soluzione in informatica: altri linguaggi, come Java, ad esempio, propongono soluzioni diverse
- La soluzione del C offre la massima libertà al programmatore
- Lo svantaggio è la facilità nel commettere errori!

Concetti base

- Ogni variabile è conservata in memoria
- Possiamo vedere la memoria come una grande tabella, in cui ogni riga corrisponde ad un byte
- Ogni dato in memoria è individuato dal *numero di riga (il suo indirizzo)* che contiene il primo byte del dato

Esempio

```
int a=2;
int c[3]={1,2,3};
char b='a';
```

1	00
2	00
3	00
...	
1000	2
1001	1
1002	2
1003	3
1004	'a'

Concetti base

- In C è possibile definire ed utilizzare delle variabili che invece di contenere valori, contengono indirizzi di memoria!
- E' possibile cioè definire variabili che *fanno riferimento* ad altre variabili.

Chiariamo ulteriormente

- Alcune variabili possono essere dei *puntatori* ad altre variabili, nel senso che ne contengono *l'indirizzo in memoria;*
- Il C offre degli operatori che consentono di:
  - Ottenere l'indirizzo di memoria (il *riferimento* ad una variabile);
  - Dato il riferimento ad una variabile (di qualunque tipo), consentono di ottenere il valore della variabile a cui ci si riferisce;

Dichiarazione di un puntatore

```
int *a;
```

- La '*' prima del nome della variabile significa che **a** è una variabile che contiene *un riferimento* (indirizzo di memoria) ad un intero; **non** contiene quindi un intero essa stessa!
- "**a**" è un puntatore a una variabile di tipo intero

Uso dei puntatori

- L'operatore **'&'** consente di ottenere l'indirizzo di memoria di una variabile. Es.:
  `int c=1;`
- `c` è una variabile di tipo intero che vale `1`
  `int *a;`
- `a` è l'indirizzo di una cella di memoria che contiene una variabile di tipo intero
  `a=&c;`
- Ad `a` viene assegnato l'indirizzo della variabile `c: a` punta alla una cella di memoria che contiene la variabile intera `c`

Esempio

- Immaginiamo che il compilatore abbia allocato i dati in memoria secondo la tabella a fianco.
- La variabile `c` è stata memorizzata nella riga 1001 della memoria.

```
c==1;
&c==1001;
a=&c==1001;
```

1	00
2	00
3	00
...	
1000	0
1001	1
1002	0
1003	0
1004	0

Come utilizzare un riferimento

- L'operatore * è detto di *dereferenziazione*: premettere * ad un puntatore vuol dire operare sul dato puntato. Esempio:

```
int c=1;
int *a=&c;
```

- `a` è l'indirizzo di una cella di memoria che contiene una variabile di tipo intero: questo indirizzo è l'indirizzo di `c`

  `*a=2;` oppure `c=2;`
- L'elemento puntato da `a` è 2: questo equivale a dire che `c` vale 2 perché `c` e `*a` sono lo stesso dato

Altre operazioni sui puntatori

```
int a; int *b; int *c; […]
```

- I puntatori possono:
  - Essere confrontati:
    `if (b==c)` ⇒ `(*b==*c)`…
  - Essere incrementati/decrementati:
    `b++; c--; b=b+2;`
    (bisogna fare **molta attenzione!**)
  - In generale, possono essergli sommati numeri interi;
  - **Non** possono essere sommati tra di loro (`b+c` non ha senso!)

Riassumendo

- `&a` vuol dire: l'indirizzo della variabile `a`;
- `*b` vuol dire: il valore contenuto all'indirizzo `b`;
- Possiamo vedere `*` e `&` come operatori inversi: `&*a` è come scrivere `a`

```
(*&a==a)
(&*a==a)
```

Esempio

```
#include <stdio.h>
#include <stdlib.h>
void main(){
  int *a,*b;
  int c;
  c=2;
… /*istruzioni per allocare lo
  spazio per i puntatori*/
```

Esempio (2)

```
*a=c;
```

- L'elemento puntato da `a` vale `c`, cioè 2

```
*a=3;
```

- L'elemento puntato da `a` vale 3
- ATTENZIONE: Questa istruzione non modifica il valore di c

```
printf("%d\n",a);
```

- Stampa `1331088`, cioè l'indirizzo contenuto in `a`

Esempio (3)

```
printf("%d\n",*a);
```

- Stampa 3, cioè il valore che sta nella cella il cui indirizzo è contenuto in `a`, in altre parole il valore a cui "punta" `a`

```
printf("%d\n",c);
```

- Stampa 2, cioè il valore di `c`

```
a=&c;
```

- Ad `a` viene assegnato l'indirizzo di `c`, cioè `a` "punta" alla cella in cui è contenuto `c`

Esempio (4)

```
printf("%d\n",a);
```

- Stampa `1245032`, cioè l'indirizzo contenuto in `a` (l'indirizzo della cella in cui è contenuto `c`)

```
printf("%d\n",*a);
```

- Stampa 2, cioè il valore che sta nella cella il cui indirizzo è contenuto in a, in altre parole il valore a cui "punta" a

```
printf("%d\n",c);
```

- Stampa 2

Puntatori e vettori

- Vettori e puntatori sono strettamente correlati:
  `int x[10];`
- `x` è ora un puntatore ad intero!
- `x` e `&x[0]` sono la stessa cosa
- `x[0]` e `*x` sono la stessa cosa
- `x[1]` e `*(x+1)` sono la stessa cosa
- Più in generale:

`y[n]` e `*(y+n)` sono la stessa cosa...

Puntatori e vettori

- Quindi, quando dichiariamo un vettore:

`int c[10];`

il compilatore:

- Riserva una zona della memoria per la memorizzazione del vettore;
- Considera `c` come un puntatore ad intero e gli assegna l'indirizzo del primo elemento `(&c[0]);`

Puntatori e vettori

- ATTENZIONE: **dichiarare** un puntatore ad una variabile **non** è come dichiarare un vettore, perché nel primo caso **non** viene riservata (allocata) memoria per contenere i dati!
- Vedremo più avanti come sia possibile chiedere di allocare memoria durante l'esecuzione di un programma.

Perché i puntatori sono "pericolosi"

- I puntatori sono indirizzi di memoria;
- Il pericolo maggiore è di fare riferimento ad indirizzi di memoria non significativi;
- Esempio:

```
int a=1;
int *c=&a;
c++; printf ("%i",*c);
```

`c` punta da qualche parte nella memoria, e fa riferimento ad un dato non definito...

Passaggio per riferimento

- I puntatori permettono di realizzare il passaggio dei parametri alle funzioni *per riferimento*
- E' utile quando si vuole realizzare una funzione che modifichi i parametri passati. Esempio:

```
void incrementa(int *a)
{
    (*a)++; return;
}
```

- Il valore a cui a "punta" viene incrementato e tale modifica è visibile anche dopo che la funzione `incrementa` ha terminato la sua esecuzione

Passaggio per riferimento

- Se invochiamo la funzione `incrementa` dobbiamo passare un puntatore ad intero:

```
main()
{
  int a=1;
  incrementa(&a);
  printf ("%d",a);
}
```

Esercizio

- Realizzare una funzione che scambi due variabili di cui riceve il riferimento:

```
void scambia (int *a, int *b)
[…]
```

Soluzione

```
#include <stdio.h>
void scambia (int *a, int *b){
  int x;
  x=*a;
  *a=*b;
  *b=x;
}
```

Soluzione (cont.)

```
void main()
{
  int s,t;
  s=7;
  t=8;
  scambia(&s,&t);
  printf("%d\n",s); /*8*/
  printf("%d\n",t); /*7*/
}
```