Puntatori in C

Programmazione I e Laboratorio

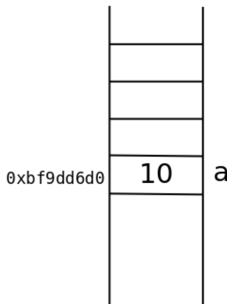
Daniele Sgandurra
daniele@di.unipi.it

Università di Pisa

03/12/2009

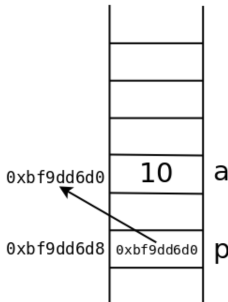
Proprietà di una Variabile

- Es.: `int a = 10;`
  - tipo: `int`
  - nome: `a`
  - valore: `10`
  - indirizzo di memoria: 0xbf9dd6d0
- Per ottenere l'indirizzo in memoria di una variabile:
  - operatore di indirizzo "`&`" (pronunciato "ampersand");
  - applicato alla variabile `a` (cioè facendo `&a`) ritona il valore `0xbf9dd6d0`.



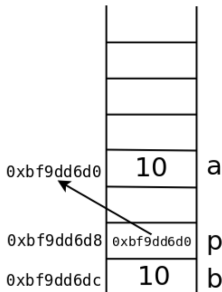
Variabili di Tipo Puntatore

- È possibile assegnare l'indirizzo di una variabile ad un'altra variabile, utilizzando le variabili di tipo puntatore.
- Es.: `int *p = &a;`
  - tipo: `int *` (puntatore ad intero)
  - nome: `p`
  - valore: `&a` (l'indirizzo in cui è salvato il valore di `a`)
  - indirizzo di memoria: 0xbf9dd6d0
- Nella dichiarazione di prima, l'operatore "`*`" specifica che `p` è una variabile puntatore (una variabile che contiene un indirizzo di memoria), mentre `int` che la memoria puntata contiene una variabile intero.



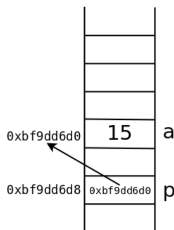
Operatore di Dereferenziamento

- È possibile risalire da una variabile puntatore al valore contenuto nella memoria puntata?
- Sì, utilizzando l' operatore di dereferenziamento "$\star$".
- Es.: `int b = *p;`
- Nell'istruzione precedente, abbiamo assegnato a `b` il valore della variabile puntata da `p` (cioè `a`).



Operatore di Dereferenziamento

- È anche possibile modificare il valore di una variabile puntata utilizzando l'operatore di dereferenziamento sul puntatore.
- Es.: `*p = 15;`
- Nell'istruzione precedente, abbiamo assegnato alla variabile puntata da `p` (cioè `a`) il valore `15`.



- Attenzione a non confondere le varie occorrenze di "$\star$":
  - in una dichiarazione serve per dichiarare una variabile di tipo puntatore:
    - `int *p;` (dichiaro una variabile di tipo puntatore a intero)
  - in un' espressione indica l'operatore di dereferenziamento:
    - `int b = *p;` (assegno a `b` il valore della variabile puntata da `p`)
    - `*p = 15;` (assegno il valore `15` alla variabile puntata da `p`)

UNIVERSITÀ DI PISA

Stampare un Puntatore

- Per stampare il valore di una variabile puntatore con la `printf`, si usa il segnaposto "`%p`", ad esempio:

```
#include <stdio.h>

int main()
{
  int a = 10;
  int *p = &a;

  printf(valore di p = %p\n, p);
  printf(valore della variabile puntata da p = %d\n, *p);
}
```

Compilando ed eseguendo si ottiene:

```
$ gcc stampa.c -o stampa
$ ./stampa
valore di p = 0xbfaca850
valore della variabile puntata da p = 10
```

Esercizio

Senza uso di puntatori, scrivere un programma che scambi il valore di due variabili intere `a` e `b`, lette con `scanf`, stampando con `printf` i valori prima e dopo aver effettuato lo scambio.

```
#include<stdio.h>

int main()
{
  int a, b, temp;

  scanf(%d \n %d, &a, &b);
  printf(valori di a=%d, di b=%d prima\n, a, b);

  temp = a;
  a = b;
  b = temp;

  printf(valori di a=%d, di b=%d dopo\n, a, b);
}
```

Compilando ed eseguendo si ottiene:

```
$ gcc swap.c -o swap
$ ./swap
10
5
valori di a=10, di b=5 prima
valori di a=5, di b=10 dopo
```

Provate a fare lo stesso esercizio usando i puntatori, introducendo due variabili puntatore ad intero `int *pa` e `int *pb` che puntano ad `a` e `b`.

Attenzione

È un errore comune dereferenziare una variabile puntatore non inizializzata: si sta scrivendo/leggendo su un indirizzo non definito!!!

```
int main()
{
  int *p; //variabile non inizializzata
  *p = 10; //ERRORE!!|
}
```

Compilando ed eseguendo si ottiene:

```
$ gcc errore.c -o errore
$ ./errore
Segmentation fault
```

Il sistema operativo termina il programma con l'errore `Segmentation fault`, perché abbiamo provato a scrivere su un indirizzo di memoria non valido! Dobbiamo sempre ricordarci di inizializzare la variabile, ad es.:

```
int main()
{
  int a;
  int *p = &a; //variabile inizializzata
  *p = 10; //corretto
}
```

Tipi di Variabile Puntatore

- Per ogni tipo primitivo del C esiste un puntatore:
  - `int a; int *pi = &a;`
  - `float f; float *pf = &f;`
  - `char c; char *pc = &c;`
  - etc.
- Nota bene: due variabili di tipo puntatore a tipi diversi non sono compatibili tra di loro.
- Esiste anche un puntatore generico, il cui tipo è `void *` :
  - compatibile con tutti i tipi puntatore (può puntare a qualsiasi variabile);
  - non può essere dereferenziato (occorre fare il cast esplicito).

Dimensione in Memoria delle Variabili e dei Tipo

- L'operatore `sizeof()` restituisce il numero di byte occupati in memoria da una variabile o da un tipo, ad es.:

```
#include<stdio.h>

int main()
{
  int a = 10;
  printf(dimensione di a = %d\n, sizeof(a));
  printf(dimensione di un int = %d\n, sizeof(int));
}
```

Compilando ed eseguendo si ottiene:

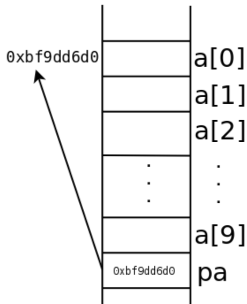
```
$ gcc sizeof.c -o sizeof
$ ./sizeof
dimensione di a = 4
dimensione di un int = 4
```

Esercizio

Usando `sizeof()` e `printf` stampare la dimensione in memoria di tutti i tipi di dato primitivo che avete incontrato finora in C, e di tutte le variabili puntatore per ogni tipo di dato.

Array e Puntatori

- Array: insieme contiguo di celle in memoria tutte contenenti lo stesso tipo di dato, ad es.: `int a[10];`
- Nel caso degli array, il nome è una variabile puntatore, cioè contiene l'indirizzo della prima locazione dell'array:
  - ad es., possiamo scrivere `int *pa = a;` (`pa` è una variabile puntatore che punta alla prima locazione dell'array definito prima)



Array e Puntatori

- È possibile ottenere l'indirizzo corrispondente ad ogni posizione dell'array con l'operatore "`&`", ad es.:
  - `int *pa = &a[0];` (analogo a `int *pa = a;`)
  - `int *pa = &a[9];` (`pa` punta alla decima locazione)
- Nelle variabili puntatori ad array, l'operatore di dereferenziamento `*` funziona così (supponiamo di aver dichiarato `int *pa = a;`):
  - `*pa` corrisponde a `a[0]`;
    - ad es.: `int b = *pa;` (assegniamo a `b` il valore presente in `a[0]`)
  - `*(pa+1)` corrisponde a `a[1]`;
    - ad es.: `*(pa+1) = 5;` (scriviamo in `a[1]` il valore `5`)
  - `*(pa+2)` corrisponde a `a[2]`
    - ad es.: `int b = *(pa+2);` (assegniamo a `b` il valore presente in `a[2]`)
  - e così via.

Esercizio

- Dichiarare un array di 10 elementi e, usando i puntatori: (i) inizializzarlo tramite `scanf`; (ii) stampare gli elementi; (iii) stampare l'indice e l'indirizzo dell'elemento maggiore e minore.

Malloc

- A tempo di esecuzione è possibile richiedere una nuova area di memoria (ad es., per contenere un array di 10 posizioni) e ottenerne l'indirizzo.
- Si usa la funzione `malloc` che ha questa firma:
  `void *malloc(size_t size);`
- `size` indica quanti byte vogliamo allocare: ad es. con `sizeof()`.
  - es.: `float *f = malloc(sizeof(float));` (alloca un float)
  - es.: `int *a = malloc(10 * sizeof(int));` (alloca un array di 10 interi: successivamente, l'uso di `a` è del tutto analogo ad averla dichiarata con `int a[10];`)
- Prima della fine del programma, tutte le zone di memoria allocate con `malloc` vanno liberate con la funzione `free` che ha questa firma:
  `void free(void *ptr);`
  - es.: `free(f);` (libera la memoria puntata da `f`)
  - es.: `free(a);` (libera la memoria puntata da `a`)

Esercizio

- Modificare l'esercizio precedente in maniera da chiedere all'utente la dimensione dell'array e quindi creare l'array di quella dimensione tramite `malloc`. Ricordarsi di eseguire la `free` alla fine.