Fondamenti di Informatica A
Laboratorio 8
Array e puntatori in C

Danilo Pianini
danilo.pianini@unibo.it
Mirko Viroli
mirko.viroli@unibo.it

Alma Mater Studiorum—Università di Bologna

8 maggio 2013

Operazioni preliminari

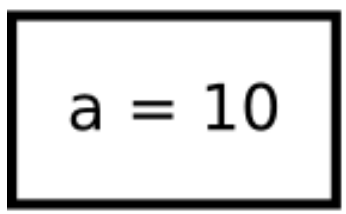
- Accendere il computer
- Effettuare il log-in con le proprie credenziali istituzionali
  - Normalmente sono simili a:
    `nome.cognome@studio.unibo.it`
- Aprire Google Chrome, o altro browser
- Collegarsi al sito `http://bit.ly/finfa2013-e8`
- Scaricare il pacchetto `http://bit.ly/finfa2013-e8-code`
- Scompattare il pacchetto in una cartella a piacimento: questa sarà la vostra directory di lavoro di oggi.

Comprensione di un esempio con puntatori

- Aprire il file `e2.c` contenente la funzione `swap` e un `main` che effettua un semplice test di essa: se ne analizzi il codice cercando di capirne il funzionamento.
- Prima di compilare il codice, si provi a capire quale dovrebbe essere l'output prodotto sulla console.
- Compilare ed eseguire. Si verifichi che l'output del programma corrisponde a quanto previsto.

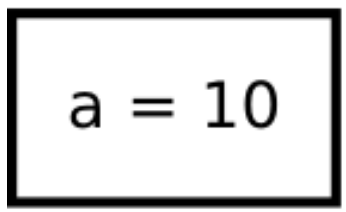
Cosa succede? 1/7

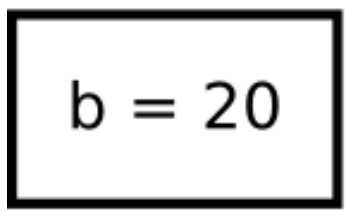
```
void swap(int *i1,int *i2){
    int temp=*i1;
    *i1=*i2;
    *i2=temp;
}
int main(void){
    int a=10; // Il controllo è qui
    int b=20;
    swap(&a,&b);
}
```



Cosa succede? 2/7

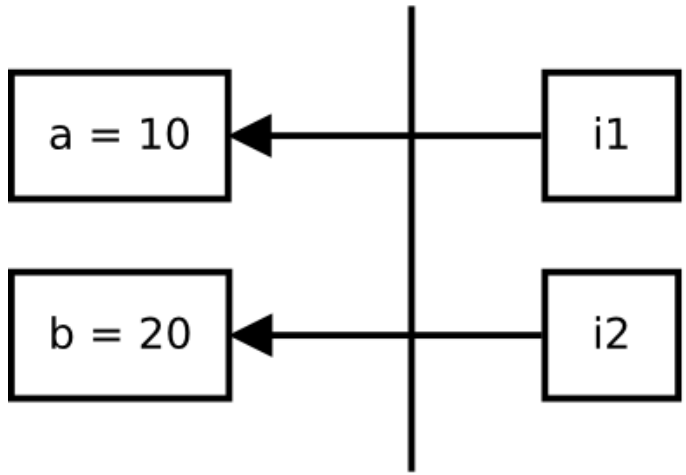
```
void swap(int *i1,int *i2){
    int temp=*i1;
    *i1=*i2;
    *i2=temp;
}
int main(void){
    int a=10;
    int b=20; // Il controllo è qui
    swap(&a,&b);
}
```





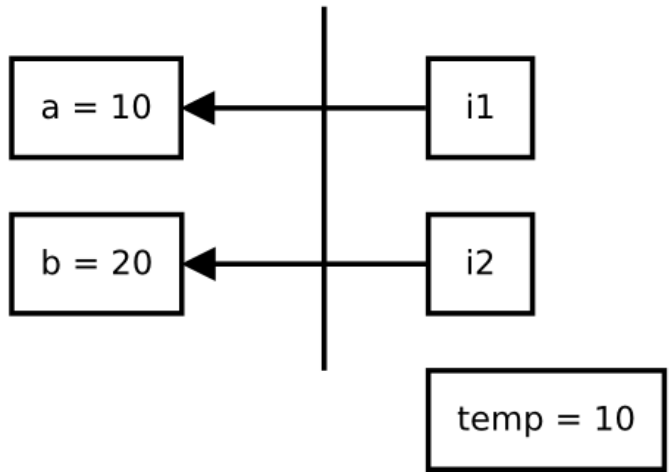
Cosa succede? 3/7

```
void swap(int *i1,int *i2){  // Il controllo è qui
    int temp=*i1;
    *i1=*i2;
    *i2=temp;
}
int main(void){
    int a=10;
    int b=20;
    swap(&a,&b);
}
```



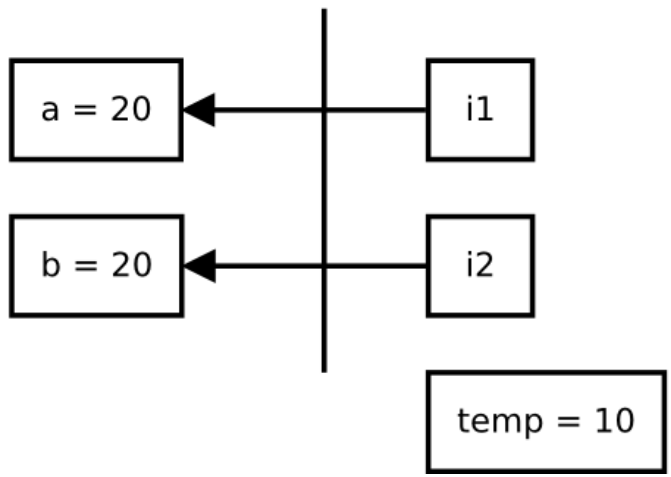
Cosa succede? 4/7

```
void swap(int *i1,int *i2){
    int temp=*i1; // Il controllo è qui
    *i1=*i2;
    *i2=temp;
}
int main(void){
    int a=10;
    int b=20;
    swap(&a,&b);
}
```



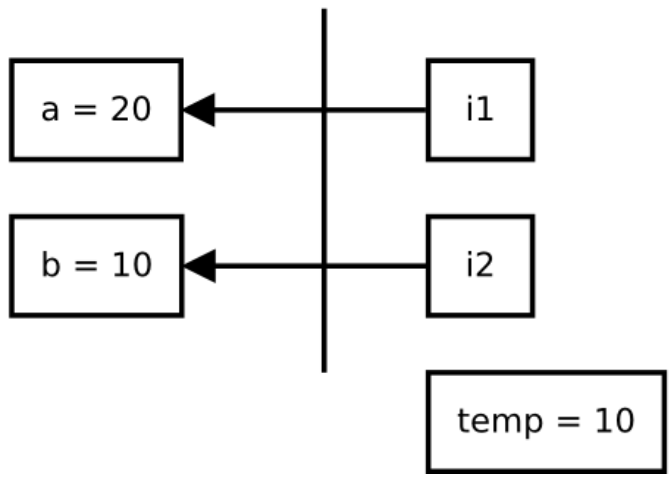
Cosa succede? 5/7

```
void swap(int *i1,int *i2){
    int temp=*i1;
    *i1=*i2; // Il controllo è qui
    *i2=temp;
}
int main(void){
    int a=10;
    int b=20;
    swap(&a,&b);
}
```



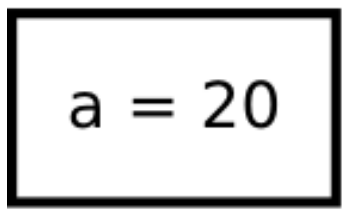
Cosa succede? 6/7

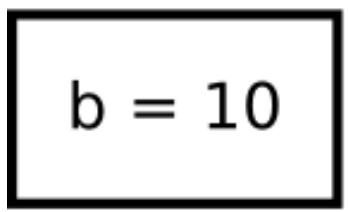
```
void swap(int *i1,int *i2){
    int temp=*i1;
    *i1=*i2;
    *i2=temp; // Il controllo è qui
}
int main(void){
    int a=10;
    int b=20;
    swap(&a,&b);
}
```



Cosa succede? 7/7

```
void swap(int *i1,int *i2){
    int temp=*i1;
    *i1=*i2;
    *i2=temp;
}
int main(void){
    int a=10;
    int b=20;
    swap(&a,&b); // Il controllo è qui
}
```





Modifica dell'esempio

Si modifichi `e2.c` come segue:

- Si elimini il contenuto del `main`.
- Si crei sullo stack (sintassi `{10, 20, 30, 40}`) un nuovo array di interi, contenente i valori 10, 20, 30 e 40.
- Si stampi il valore dei quattro elementi dell'array tramite apposita `printf`
- Si invochi la funzione `swap` passando come primo argomento il puntatore al primo valore dell'array e come secondo argomento il puntatore al secondo elemento dell'array (suggerimento: l'accesso all'i-esimo elemento dell'array a si effettua tramite `a[i]`, mentre il puntatore all'i-esimo elemento di a si può ottenere con `&a[i]`)
- Si stampi nuovamente il valore dei quattro elementi dell'array tramite apposita `printf`
- Si compili ed esegua: il risultato è conforme alle attese?

`void *malloc(unsigned int size)`

La funzione `void *malloc(unsigned int size)` consente di allocare per il programma una porzione di memoria nello heap grande `size` bytes.

- Si osservi il contenuto di `create_array.c`. Cosa fa? Una volta compreso il funzionamento, si compili e si esegua.
- Si modifichi `create_array.c` affinché crei un nuovo array il cui contenuto sia numerato progressivamente (e.g. `create_array(5)` deve tornare `{1,2,3,4,5}`. Si modifichi il `main` perché stampi a video l'intero contenuto dell'array con gli elementi separati da spazi, quindi vada a capo.

Stringhe di caratteri in C

Diversamente da Java, C non ha supporto per il tipo `String` a livello di linguaggio. In C, le Stringhe sono in realtà dei `char *` (ossia anche dei `char []`), terminati dal carattere speciale `\0`.

- Utilizzando anche la funzione malloc vista in precedenza, e facendo riferimento ai lucidi visti a lezione, si realizzi una funzione che concatena due stringhe, separandole con uno spazio, ossia col carattere `' '` (si noti che la concatenazione semplice è già risolta nei lucidi!)
- Si scriva un `main` che concateni le stringhe "ottimo" e "lavoro". E le stampi a video. Si ricordi che è possibile stampare con `printf` un `char *` direttamente.
- Si scriva una funzione `char scanToLast(char *s)` che, data una stringa di lunghezza non nota, torna l'ultimo elemento. Non è consentito l'uso di `strlen`. La si testi nel `main`.

Output negli argomenti

I puntatori consentono di utilizzare argomenti come output: si passa un puntatore all'area in cui vogliamo il risultato, la funzione computa e scrive in quell'area il risultato, in questo modo il chiamante può accedervi. Si faccia riferimento alla slide 10 del pacchetto `06d-C-array_puntatori2.pdf`.

- Si realizzi la funzione
  `void sum(int a, int b, int *res)`
  che dati due interi ne calcola la somma e la mette nell'area di memoria puntata da res. Si verifichi nel `main` il funzionamento.
- Si implementi nel file `array_creator.c` la funzione
  `void arrayCreator(int size, int **res)`
  che crea un array di interi lungo `size` nell'area di memoria puntata da res, utilizzando la `malloc`, quindi lo riempie con valori crescenti a partire da 0.

Esercizi aggiuntivi

Chi ha terminato gli esercizi di base, può dar libero sfogo alle proprie abilità svolgendo quelli aggiuntivi.

Riconoscitori di grammatiche in C

Si risolvano gli esercizi proposti nei file `e3.c` ed `e4.c`, sull'implementazione di un semplice riconoscitore di grammatiche e della sua versione ricorsiva.

Matrici in C

- Si osservi il file `matr.c`. Se ne comprenda il funzionamento, e solo dopo averlo compreso si compili e si esegua.
- Sulla base di matr.c, si crei un nuovo programma in `matrBorder.c`, che crea una matrice della dimensione desiderata, quindi ne resetta il bordo esterno (prima e ultima riga, prima e ultima colonna).

Output negli argomenti

Perché fermarsi ai puntatori a puntatore quando si possono fare puntatori a puntatori a puntatori? (e via per induzione...)

- Si realizzi all'interno di matr.c la funzione
  `void matrixCreator(int size, int ***res)`
  che crea una matrice quadrata di interi grande `size` nell'area di memoria puntata da res, utilizzando la `malloc`, quindi lo riempie con valori crescenti a partire da 0. Si usi nel `main` questa funzione al posto di
  `int **matr(int size,int elem)`.