Array e puntatori

Laboratorio di Informatica I

Gli array

• un array è un insieme di elementi (valori) avente le seguenti caratteristiche:

- **un array è ordinato:** agli elementi dell'array è assegnato un ordine (c'è un primo elemento, un secondo elemento, e così via
- **un array è omogeneo:** tutti i valori immagazzinati nell'array sono dello stesso tipo

Gli array

• proprietà di un array:

1. **Tipo degli elementi:** tipo dei valori che possono essere immagazzinati nell'array
2. **Dimensione dell' array:** numero di elementi che possono essere immagazzinati nell'array

• dichiarazione di un array:

- nome dell'array
- tipo degli elementi
- dimensione dell'array

Esempio: `int a[10];` dichiara un array di nome `a` con 10 elementi di tipo *int*

Gli array

•Ciascun elemento dell'array è identificato da un intero detto *indice*

- gli indici partono da 0
- gli indici di un array di 5 elementi sono: 0,1,2,3,4

• L'identificazione di un elemento di un array è detta *selezione.* Per selezionare un elemento di un array bisogna specificare il nome dell'array e l'indice dell'elemento a cui si vuole accedere.

- Esempio: l'espressione `a[4]` permette di accedere all'elemento con indice 4 dell'array **a**

Classi di memorizzazione per array

• *extern*

– tutti gli elementi dell'array inizializzati per default a zero

• *static*

– tutti gli elementi dell'array inizializzati per default a zero

• *auto*

– gli elementi non sono necessariamente inizializzati a zero

Inizializzazione esplicita degli array

• Avviene elencando gli inizializzatori per ciascun elemento

– esempio: `int a[4]= {2, 5 , 3, 1};`
pone `a[0]=2, a[1]=5, a[2]=3, a[3]=1`

•Se gli inizializzatori elencati sono in numero inferiore a quello degli elementi dell'array allora i restanti elementi vengono inizializzati a zero

– esempio: `float b[3]={4.0, 2.5};`
pone `b[0]=4.0, b[1]=2.5, b[2]=0`

Dimensionamento dell'array tramite l'inizializzazione

• Se un array viene dichiarato senza specificarne la dimensione, allora si assume come dimensione il numero dei valori di inizializzazione

– esempio: int a[ ]= {2, 5 , 3, 1}; è equivalente a
int a[4]= {2, 5 , 3, 1};

•Nel caso di array di caratteri è possibile utilizzare le costanti stringhe come valori di inizialiazzazione

– esempio: char s[ ]="ciao"; è equivalente a
char s[ ]={'c', 'i', 'a', 'o', '\0'};
(\0 è il marcatore di fine stringa)

Gli array come parametri di funzione

• Gli array possono essere passati come parametri ad una funzione

• Ogni array ha un *indirizzo* di memoria a partire dal quale è memorizzato (*indirizzo di base*)

• L'indirizzo a partire dal quale è memorizzato un array è specificato dal nome dell'array

• Se ad una funzione viene passato un array come argomento

– viene passato per valore *l'indirizzo di base dell'array*

– le modifiche effettuate nella funzione sono effettuate sull'array e **non su una copia**

I puntatori

• Sono variabili che assumono indirizzi come valori

– dichiarazione tipica: int *p;

p è un puntatore a *int* e quindi può contenere indirizzi di variabili di tipo *int*

• Esempio:
```
int *p;          /*p è un puntatore a int*/
int a;
p = &a;          /* a p viene assegnato
                    l'indirizzo di a */
```

L'operatore *

• operatore di indirizzamento indiretto

– *p; rappresenta il valore della locazione di memoria il cui indirizzo è contenuto in p

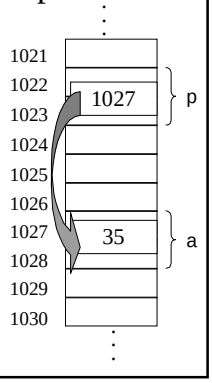
– esempio:
```
int *p;
int a=35,b;
p = &a;
b= *p;     /* assegna a b il valore di a  (35) */
```

• ha la stessa priorità e associatività da destra degli altri operatori unari

– esempio: se q è un puntatore ad una variabile puntatore allora ****q è equivalente a *(*q),** ovvero rappresenta il valore della variabile il cui indirizzo è nella variabile puntata da q

Rappresentazione dei puntatori

```
int *p;
int a= 35;
p = &a;
```



• La variabile puntatore p è memorizzata a partire dal byte 1022

•La variabile *int* a è memorizzata in 2 byte: il primo ha indirizzo 1027 e il secondo ha indirizzo 1028

• La variabile p punta ad a (contiene 1027 che è l'indirizzo a partire dal quale a è memorizzata)

Inizializzazione in dichiarazioni

• Esempio :

– int i, *p = &i;

p è un puntatore ad una variabile di tipo *int* e il suo valore iniziale è l'indirizzo di i

• Attenzione!

– *p = &i; inizializza p e **non** *p

– i deve essere dichiarata prima di p

Assegnamenti a puntatori

• I valori assegnabili ad un puntatore includono l'indirizzo speciale 0 e un insieme di interi che rappresentano indirizzi della macchina

– esempi:

```
p = 0;
p = NULL;       /* stessa cosa di p = 0 ;*/
p = &i;
p = (int *) 1123   /* p contiene l'indirizzo 1123 */
```

Assegnamenti illegali

• non tutti i valori vengono memorizzati in locazioni accessibili attraverso i puntatori

– esempi:

```
p = &5;
p = &(k * 10);
register int i;
p = &i;
```

Alcuni esempi di espressioni con puntatori

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

espressione	espressione equivalente	valore
p == &i	p == (&i)	1
*&p	* (&p)	&i
**&p	* (* (&p))	3
r = &x	r = (& x)	illegale
7 * * p / * q + 7	((7 * (*p))/ (* q)) + 7	11

Chiamata per indirizzo

• In altri linguaggi alle funzioni vengono passati gli indirizzi delle variabili passate come argomento

– in questo modo una funzione modificherà le variabili passate come parametri

• In C il programmatore può effettuare una chiamata per indirizzo usando come parametri della funzione gli indirizzi delle variabili

– i parametri devono essere indicati come puntatori

- esempio di prototipo: double f (double *p);
- esempio di chiamata: f(&p);

Esempio di chiamata per indirizzo

• vogliamo costruire una funzione che scambia il valore di due variabili

• prima vedremo una soluzione sbagliata che fa uso della chiamata per valore

• successivamente vedremo la soluzione corretta che fa uso della chiamata per indirizzo

SWAP
una soluzione sbagliata

```
void swap (int, int);
main()
{
int i=3, j=5;
printf ("i=%d, j=%d\n");   ← stampa i = 3, j = 5
swap (i, j);
printf ("i=%d, j=%d\n");   ← stampa di nuovo i = 3, j = 5
}
void swap (int a, int b)
{
int temp;
temp = a;
a = b;
b = temp;
}
```

SWAP
una soluzione sbagliata

- swap (i, j);

  – Quando s*wap* viene chiamata all'interno della funzione main i valori degli argomenti i e j vengono copiati nei parametri formali a e b, rispettivamente

- ```
  void swap (int a, int b)
  {
  int temp;
  temp = a;
  a = b;
  b = temp;
  }
  ```

  – La funzione s*wap* scambia i valori di a e b e non quelli di i e j

SWAP
una soluzione corretta

```
void swap (int *, int *);
main()
{
int i=3, j=5;
printf ("i=%d, j=%d\n");
swap (&i, &j);
printf ("i=%d, j=%d\n");
}
void swap (int *a, int *b)
{
int temp;
temp = *a;
*a = *b;
*b = temp;
}
```

- i parametri sono puntatori ad interi
- stampa i= 3, j=5
- a *swap* vengono passati gli indirizzi di *i* e *j*
- stampa i = 5, j =3
- mette in temp il valore puntato da a
- mette nella variabile puntata da a il valore puntato da b
- mette nella variabile puntata da b il valore di temp

SWAP
una soluzione corretta

- swap (&i, &j);

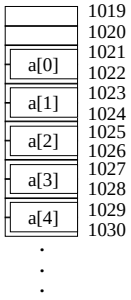
  – Quando si passano &i e &j a *swap*, gli indirizzi di i e j vengono copiati in a e b

- ```
  void swap (int *a, int *b)
  {
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
  }
  ```

  – La funzione s*wap* effettua quindi lo scambio su *a e *b. Ciò corrisponde ad effettuare lo scambio su i e j

Rappresentazione di un array

- Un array è memorizzato in locazioni consecutive a partire da un indirizzo di partenza
- Esempio: int a[5];
- Il compilatore ottiene l'indirizzo di a[2] (1025) sommando l'indirizzo di partenza al numero di byte necessari per rappresentare a[0] e a[1]:
  – 1021 + 2 * sizeof (int)



Relazioni tra array e puntatori

- Il nome di un array è di per sé un indirizzo di memoria
- Array e puntatori sono di fatto equivalenti a parte alcune differenze sottili:
  – un array rappresenta un indirizzo fissato (a differenza dei puntatori che possono cambiare)
  – con un array si alloca anche della memoria, mentre con i puntatori no.

Equivalenza di espressioni con array e puntatori

- Consideriamo un array a e un intero i

```
float a[7];
int i;
```

a[i]; è equivalente a ***(a + i);**

- Consideriamo un puntatore p e un intero i

```
int *p;
int i;
```

***(p + i);** è equivalente a **p[i];**

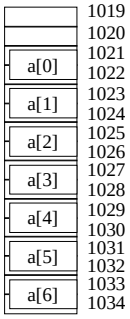
Motivo dell'equivalenza

• Il compilatore ottiene l'indirizzo di a[i] sommando l'indirizzo di partenza di a al numero di byte necessari per rappresentare a[0], a[1],…,a[i-1]:

– 1021 + i * sizeof (int)

• Per accedere a *(a+i) ci si sposta di i posizioni *intere* rispetto all'indirizzo contenuto nel puntatore a spostato

– 1021 + i * sizeof(int)



Aritmetica dei puntatori

• Se p è un puntatore ad un dato con un certo tipo:

– allora l'espressione p+i rappresenta l'indirizzo di memoria per memorizzare la successiva i-esima variabile dello stesso tipo

• Anche le seguenti espressioni sono significative:

p += i;

p--;

• Se q e p sono due puntatori che puntano ad elementi dello stesso array allora p-q rappresenta il numero di elementi dell'array tra p e q

Esempio dell'aritmetica dei puntatori

```
#include <stdio.h>
main()
{
double a[2], *p, *q;
p=a;
q= p+1;
printf("%d\n", q-p);       /* stampa 1  (differenza in termini di
                              elementi dell'array) */
printf("%d\n", (int) q – (int) p);     /* stampa 8   ( differenza
                                          in termini  di byte)  */
}
```

Esercizi

- Scrivere una funzione GetArray( ) che prende in input al più 15 interi e li mette in un array dichiarato esternamente. La funzione restituisce il numero degli elementi presi in input.
- Scrivere un programma che:
  - Prende in input mediante GetArray( ) un array di al più 15 interi.
  - Chiama una funzione che restituisce il numero di interi nell'array che sono la somma di due interi inseriti nell'array
  - Suggerimento: la funzione deve scorrere tutte le terne di indici dell'array (tre for innestati)

Esercizi

- Scrivere una funzione ***GetString(a[],n)*** che prende in input una stringa di al più ***n*** caratteri e li mette nell'array ***a[]*** dichiarato esternamente. La funzione restituisce il numero degli elementi presi in input.
- Scrivere un programma che:
  - Prende in input mediante ***GetArray(a[],n)*** due stringhe di lunghezze al più 50 e 10, rispettivamente, e le mette in due array ***str1*** e ***str2*** dichiarati esternamente.
  - Il programma utilizza una funzione ***Find(str2[],n2,str1[],n1)*** per contare quante volte ***str2*** compare in ***str1***.

Esempio: str1[] = "abcddabcbadabc", str2 = "abc"

Output = 3