Puntatori e array

Tipo puntatore
Vettori

I puntatori

- un puntatore si definisce scrivendo il tipo cui si punta, l'asterisco e il nome della variabile

  esempi di dichiarazione:

```
int *ptr1, *ptr2;
```

- un puntatore rappresenta un indirizzo di memoria
  - dal momento che gli indirizzi della memoria sono interi positivi, in pratica un puntatore è un numero intero senza segno

I puntatori

Esempio di dichiarazione:

```
int *p, num = 42;
```

- **La variabile p è un puntatore destinato a contenere l'indirizzo di memoria di un intero**
- **num è una variabile che memorizza un numero intero a cui viene assegnato il valore 42**

I puntatori

Esempio di dichiarazione:

```
int *p, num = 42;
```

- **come ogni altra variabile, anche un puntatore, finché non viene inizializzato, contiene una sequenza di bit casuali che difficilmente corrisponde a un indirizzo di memoria valido**
- **tale situazione può essere causa di errori difficilmente identificabili**

I puntatori

- un puntatore non contiene il valore di una variabile "tradizionale"
- nell'esempio precedente:
  - la variabile p non è destinata a contenere un numero di tipo int
  - contiene l'indirizzo di una locazione in memoria
  - i bit puntati sono interpretati come un numero int quando tali bit vengono letti o scritti per mezzo del puntatore p
- il valore memorizzato in un puntatore può essere utilizzato per accedere al valore memorizzato nell'indirizzo puntato dal puntatore stesso

I puntatori

- come tutte le variabili, i puntatori sono caratterizzati da un proprio indirizzo e da una dimensione
- i puntatori sulla stessa macchina sono tutti della stessa dimensione
  - sono a 16, 32 o 64 bit a seconda del processore (o sistema operativo) su cui si lavora
- su tutte le piattaforme UNIX un unsigned long e un puntatore hanno la stessa dimensione

Indirizzo di una variabile

- L'indirizzo di una variabile può essere ottenuto utilizzando l'operatore unario &

- Con l'assegnamento seguente:

```
p = &num;
```

in p viene memorizzato l'indirizzo della variabile num

Indirizzo di una variabile

- il puntatore p occupa 4 byte a partire dall'indirizzo 10204
- contiene il valore 4574, che corrisponde ad un indirizzo
- che contiene 32 bit da interpretare come numero int quando venga acceduto mediante il puntatore p
- tali bit rappresentano il numero 42, accessibili anche mediante l'identificatore num

Accesso al valore puntato

- con i valori dell'esempio, p punta all'indirizzo della variabile num
- è possibile fare il seguente assegnamento:

```
*p = 3;
```

- l'operatore unario * permette di accedere al valore contenuto in un particolare indirizzo di memoria (sia in lettura che in scrittura)
- si applica l'operatore unario * al puntatore p per accedere alla locazione di memoria da esso puntata al fine di modificarne il valore, impostandolo pari a 3
- dal momento che p punta all'indirizzo di num, l'effetto è quello di assegnare a num il valore 3

Accesso al valore puntato

- un assegnamento come il seguente:

```
int num2;
num2 = *p;
```

- accede in lettura al valore memorizzato all'indirizzo puntato da p e lo assegna alla variabile num2
- in pratica, dopo l'assegnamento effettuato precedentemente, alla variabile num2 viene assegnato il valore 3

Il puntatore nullo

- il "puntatore nullo" vale zero e non è un puntatore valido, ovvero non può essere utilizzato per effettuare un accesso alla memoria
- la macro NULL viene utilizzata per identificare il puntatore nullo
- NULL vale 0
- la costante 0 è confrontabile con qualunque puntatore

Il puntatore nullo

```
char *p;
if (p == NULL) {
    /* istruzioni */
}
```

- Le istruzioni vengono eseguite solo se il valore del puntatore p è nullo
  - è sintatticamente scorretto confrontare numeri interi con puntatori; la costante 0 è un caso speciale
- Si noti che si poteva in maniera equivalente anche scrivere:

```
if (!p) { ...
```

Gli array

- Gli array permettono di memorizzare in aree contigue di memoria un numero fissato di elementi di tipo omogeneo (tutti dello stesso tipo)
- A seconda del numero di dimensioni dell'array, essi vengono chiamati vettori (dim. 1), matrici (dim. 2), o array multi-dimensionali (dim. >2)

I vettori

- I vettori, o array monodimensionali, permettono di allocare un insieme di elementi dello stesso tipo in zone contigue della memoria
- La sintassi per la dichiarazione di un vettore è la seguente:

```
nome-tipo identificatore [ cardinalità ] ;
```

- nome-tipo è un tipo di dato, sia semplice che derivato
- identificatore è il nome che identifica il vettore
- cardinalità è una costante intera che indica da quanti elementi è costituito il vettore

Limiti di indicizzazione di un vettore

- si consideri un vettore di cardinalità N

```
int vett[N];
```

  - l'identificatore associato al vettore è vett
  - ciascun elemento del vettore è di tipo int
  - N deve essere una costante
  - valori validi dell'indice sono limitati all'intervallo [0, N-1]
  - il primo valore ha indice 0 e l'ultimo ha indice N-1

I vettori

- **Esempio di definizione:**

```
int num[5];
```

- **Definisce il vettore num di 5 interi**
  - **il vettore è indicizzato da 0 a 4, ovvero il primo elemento è num[0] mentre l'ultimo è num[4]**

- **Esempio di utilizzo:**
  - **Il terzo elemento è la somma dei primi due elementi del vettore:**

```
num[2] = num[0] + num[1];
```

I vettori

- Il vettore viene allocato in un'area di memoria contigua a partire dall'indirizzo 1024
- Ciascun elemento del vettore è di tipo int e occupa 4 byte
- Il vettore può essere indicizzato nell'intervallo [0, 4]
- Sono memorizzati rispettivamente i valori 640, 231, 100, 91 e 1003

I vettori

- Gli elementi di un vettore vengono sempre memorizzati in aree contigue della memoria
- In questo modo l'indirizzo dell'elemento di indice i può essere ricavato con un semplice calcolo
  - se base è l'indirizzo del primo elemento del vettore e dim la dimensione in byte di ciascun elemento
  - allora l'indirizzo dell'elemento di indice i è pari a

$$base + i * dim$$

- considerando l'esempio precedente, essendo base uguale a 1024 e dim uguale 4, l'indirizzo dell'elemento di indice 2 (num[2]) è

$$1024 + 2 * 4 = 1032$$

Nome del vettore e indirizzo

- Il nome di un vettore è un puntatore costante il cui valore corrisponde all'indirizzo del primo elemento del vettore

  `num == &num[0]`
- È vera anche la seguente espressione:

  `*num == num[0]`
- Entrambe le notazioni indicano il primo elemento del vettore

Violazione dei limiti di un vettore

- Nel caso si tenti di accedere ad un elemento del vettore utilizzando un indice con valore al di fuori dell'intervallo ammesso, nel migliore dei casi il programma termina con un errore di "segmentation fault" (violazione di memoria), ovvero di accesso ad una area di memoria alla quale non è permesso accedere
  - l'area di memoria cui si fa accesso può essere di pertinenza di un altro programma oppure essere riservata al sistema operativo

Violazione dei limiti di un vettore

- In altri casi l'accesso errato va a leggere/scrivere una porzione di memoria che appartiene al programma corrente, allocata nella memoria presente oltre la dimensione del vettore
- è difficile prevedere quale errore verrà causato
  - si pensa di accedere ad un elemento del vettore
  - invece si sta accedendo ad un indirizzo che non ha relazione con il vettore stesso
  - in caso di accesso in lettura verrà letto un valore che dipende dal contenuto della memoria, che però è collocato ad un indirizzo non significativo
  - in caso di scrittura, non è possibile sapere quale variabile verrà modificata dalla scrittura stessa
- Questa situazione è molto pericolosa per il funzionamento del programma, in quanto possono verificarsi comportamenti indesiderati, imprevedibili, e spesso molto difficili da diagnosticare

Inizializzazione di un vettore

- L'inizializzazione di un vettore può essere effettuata senza specificarne la dimensione, ma semplicemente elencandone gli elementi
- non viene specificata la cardinalità del vettore
- si racchiude tra parentesi graffe una lista di elementi separati da virgola
- il compilatore dimensiona automaticamente il vettore sulla base del numero di valori utilizzati per l'inizializzazione

Inizializzazione di un vettore

- **Ad esempio:**

```
int dxm[] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

- **Il vettore dxm (days per month) viene inizializzato con la durata in giorni di ciascun mese dell'anno**
  - **in questo caso la dimensione del vettore è automaticamente impostata dal compilatore a 12**

Inizializzazione di un vettore

- È possibile dichiarare un vettore di una determinata dimensione e inizializzarne esplicitamente solo i primi elementi
  - i rimanenti elementi verranno inizializzati a 0

```
int n[10] = {1, 5, 3};
```

  - i primi tre elementi vengono inizializzati rispettivamente ai valori 1, 5 e 3, mentre i rimanenti a 0
- Un modo semplice per inizializzare a 0 tutti gli elementi di un vettore è:

```
int n[10] = {0};
```

Uso di macro con vettori e cicli

```
#define MAX 10

int vet[MAX];
for (i = 0; i < MAX; i++) {
    /* ... */
}
```

- In ogni punto del programma viene usata la macro MAX per fare riferimento alla dimensione del vettore
- Il valore della macro MAX è definito in un unico punto e quindi esiste un unico punto di aggiornamento
- Dopo la ricompilazione, il valore corretto viene utilizzato sia per dimensionare il vettore e che per controllare il ciclo

Operatore sizeof

- l'operatore sizeof si applica ad un tipo, ad un nome di variabile o ad un'espressione
- restituisce la dimensione in byte dell'oggetto indicato
- il calcolo viene effettuato durante la fase di compilazione in base al tipo di dato che viene passato a sizeof
- alcuni esempi:

```
int i, v[10], *p;
i = sizeof(char); // 1
i = sizeof(int);  // normalmente 4, ma anche 2
i = sizeof(v[0]); // 4 o 2
i = sizeof(*p);   // 4 o 2
i = sizeof(p);    // 8 o 4 (dimensione del puntatore)
i = sizeof(v);    // 40 o 20
i = sizeof(v)/sizeof(*v); /* 10: numero di elementi
    del vettore v */
```

Una macro utile per i vettori

```
#define ARRAY_SIZE(x) (sizeof(x)/sizeof(*x))
```

- restituisce il numero di elementi di un vettore
- si utilizza passando come argomento l'identificatore di un vettore

- ad esempio:

```
int dim, v[10];
dim = ARRAY_SIZE(v);
```

- a dim viene assegnato il valore 10
  - si noti che il risultato non dipende dalla dimensione del tipo di dato

Vettori di caratteri

- I vettori di caratteri non differiscono ovviamente dagli altri vettori

  esempi di dichiarazione:

```
char vet[4],
     vet1[] = { 'a', 'b', 'c', '\n' },
     vet2[4] = { 'a' },
     vet3[] = { '0', 0 };
```

  – vet non è inizializzato, vet1 è un vettore di 4 caratteri inizializzato, vet2 è ancora di 4 caratteri (il primo 'a', poi riempito di zero, vet3 memorizza 2 valori: il codice del carattere 0, il valore 0 (sono due costanti diverse)

Le stringhe

- Una stringa costante è rappresentata da una sequenza di 0 o più caratteri racchiusi fra doppi apici

- per esempio:

```
"Questa è una stringa"

"Hello World!\n"

"max=%d"
```

Le stringhe

- È possibile scrivere due stringhe consecutive separate da un 'a capo' oppure da uno o più spazi
  - in questo caso, il compilatore le concatena e le considera come fosse una unica stringa

- Nell'esempio seguente:

```
"Le due stringhe in fase di compilazione"
" saranno concatenate"
```

- Le due stringhe sono separate da un a capo
  - sono considerate dal compilatore una unica stringa costituita dalla concatenazione delle due

Le stringhe

- La memorizzazione di una stringa comprende i caratteri che effettivamente la compongono, più un carattere di terminazione che delimita l'ultimo carattere della stringa
- Il carattere di terminazione è il byte di valore numerico 0 (zero)
  - esso non corrisponde ad un carattere ASCII stampabile

Una stringa è un vettore di caratteri il cui contenuto è terminato dal carattere '\0' che delimita l'insieme di caratteri "validi" della stringa

Variabili stringa

- La dimensione di una variabile stringa deve prevedere lo spazio sufficiente per includere anche il carattere zero di terminazione, oltre ai caratteri "effettivi" che compongono la stringa
- È possibile che una stringa sia vuota, ovvero che non contenga alcun carattere
- La stringa vuota è rappresentata da "" (due doppi apici consecutivi)
- Occupa un byte, che memorizza il carattere di terminazione

Variabili stringa

stringhe.c

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    char str1[] = "Questa è una stringa";
    char str2[100] = "Hello World!\n";
    char str3[5] = "123456";

    printf("%d %s\n", sizeof(str1), str3);
    str3[4] = 0;
    printf("%s\n", str3);
    return 0;
}
```

20 + 1

```
user@ubuntu:~
user@ubuntu:~$ gcc -Wall stringhe.c
stringhe.c: In function ‘main’:
stringhe.c:7:17: warning: initializer-string for array
of chars is too long [enabled by default]
stringhe.c:6:7: warning: unused variable ‘str2’ [-Wunus
ed-variable]
user@ubuntu:~$ ./a.out
21 12345
1234
user@ubuntu:~$
```

Risultato non riproducibile

Visualizzazione di stringhe

- È possibile visualizzare delle stringhe a terminale utilizzando la funzione puts:

```
char testo[] = "Una stringa";
puts(testo);
```

- La funzione puts stampa la stringa e aggiunge sempre un carattere newline alla fine
- In alternativa si può usare anche la printf con `"%s"`

```
printf("%s", testo);
```

```
char nome[] = "Donald";
char cognome[] = "Duck";
printf("Il nome di %s e` %s.\n", cognome, nome);
```

Lettura di stringhe

- In maniera duale alla scrittura, per la lettura si possono usare sia la funzione gets, che la scanf

```
char buffer[100];
gets(buffer);
scanf("%s", buffer);
```

- La funzione gets elimina il carattere di newline, ma aggiunge il carattere di fine stringa \0
  - anche la scanf aggiunge il fine stringa
  - occorre prevedere abbastanza spazio per memorizzare i caratteri e il fine stringa
  - nell'esempio se vengono immessi più di 100 caratteri l'esecuzione diventa imprevedibile (e a volte anche pericolosa)