Puntatori e funzioni

Alessandra Giordani
agiordani@disi.unitn.it
Lunedì 14 maggio 2012
http://disi.unitn.it/~agiordani/

Puntatori - 1

- Un puntatore é una variabile che rappresenta un indirizzo di memoria
- Usati principalmente per: memorizzare indirizzi di variabili, passaggio per riferimento, manipolazione di array
- I puntatori in C sono "tipati": portano con sé informazione relativa al tipo di dato a cui puntano
- Dichiarazione di un puntatore: operatore unario "*" (solo in dichiarazione!!)

```
int a, b, c; /* dichiaro tre variabili intere */
int * x;     /* dichiaro un puntatore a intero */
double * y;  /* dichiaro un puntatore a double */
```

- Una variabile puntatore occupa in memoria un numero di byte sufficiente a rappresentare tutti gli indirizzi di memoria indirizzabili dal bus $\Rightarrow$ 4 byte in una architettura a 32 bit
- Lo spazio occupato da un puntatore non dipende dal tipo!

Puntatori - 2

- N.B.: * nella dichiarazione é associato a destra (al nome di variabile) e non a sinistra (al tipo). Se voglio dichiarare due puntatori ad intero sulla stessa riga devo scrivere

```
int * j, * k;
```

- Se invece scrivessi:

```
int * j,k;
```

  dichiarerei:

  - Un puntatore ad intero (j)
  - Una variabile intera (k)

- Domanda: quale é l'effetto della seguente istruzione?

```
double a, * b, c, d;
```

Manipolazione dei puntatori

- Operatore indirizzo di (unario): &
  - Variabile ⇒ Puntatore
  - Resituisce l'indirizzo (reference) di una variabile (e ci permette di salvarla in un puntatore!)
  - Da non confondere con l'AND logico (binario, &&) e l'AND bit a bit (binario, &)
- Operatore di dereferenziazione (unario, nelle espressioni): *
  - Puntatore ⇒ Valore
  - Restituisce il valore della variabile che si trova ad un certo indirizzo (e ci permette di manipolare il valore di una variabile di cui conosciamo l'indirizzo!)
  - Da non confondere con l'operatore di moltiplicazione (binario) o con l'operatore * nella dichiarazione di una variabile di tipo puntatore

Attenzione:

- La dichiarazione:

```
int * a;
```

- Significa: dichiaro un puntatore ad intero e lo chiamo a
- Il nome del puntatore é solo a, senza * !!!

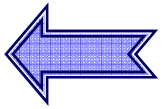
- a rappresenta un indirizzo

- *a é il valore memorizzato all'indirizzo a

L'inizializzazione

- Una dichiarazione consente di allocare la memoria necessaria per una variabile, ma alla variabile non viene automaticamente associato nessun valore
  - ⇨ Se il nome di una variabile viene utilizzato prima che sia stata eseguita un'assegnazione esplicita, il risultato non è prevedibile
- Esempio:

```
#include<stdio.h>
#include<stdlib.h>

main()
{
    int m;

    printf("Il valore di m è: %d\n", m);
    exit(0);
}
```



Il risultato del programma non è "certo": **m** assume il valore la-sciato nella locazione di memoria dall'esecuzione di un programma precedente

L'inizializzazione dei puntatori

- I puntatori possono essere inizializzati: il valore iniziale deve essere un indirizzo

```
int   j;
int   *ptr_to_j=&j;
```

- Non è possibile fare riferimento ad una variabile prima di averla dichiarata; la dichiarazione seguente non è corretta…

```
int   *ptr_to_j=&j;
int   j;
```

Esempio:

```
int a = 100, b = 50, *p;

p = &a;          // ora p punta ad a

int c = *p;      // assegno a c il valore puntato da p: 100

p = &b;          // ora p punta a b

c = *p + 10;     // assegno a c il valore puntato da p
                 // aumentato di 10, cioé 60

*p = *p + 5;     // equivalente a: b = b + 5, b vale 55

int * q = &a;

int d = *p * *q; // cioé: d = (*p) * (*q)
                 // d value 55*100 = 5500
```

Puntatori e printf

- La funzione printf permette di stampare un indirizzo mediante l'indicatore di formato %p:

```
int a = 100;
printf("a: %d, indirizzo: %p\n", a, &a);

int *b = &a;
printf("b: %p, indirizzo: %p\n", b, &b);
return 0;
```

- L'indirizzo viene stampato in formato esadecimale

L'accesso a una variabile puntata – 1

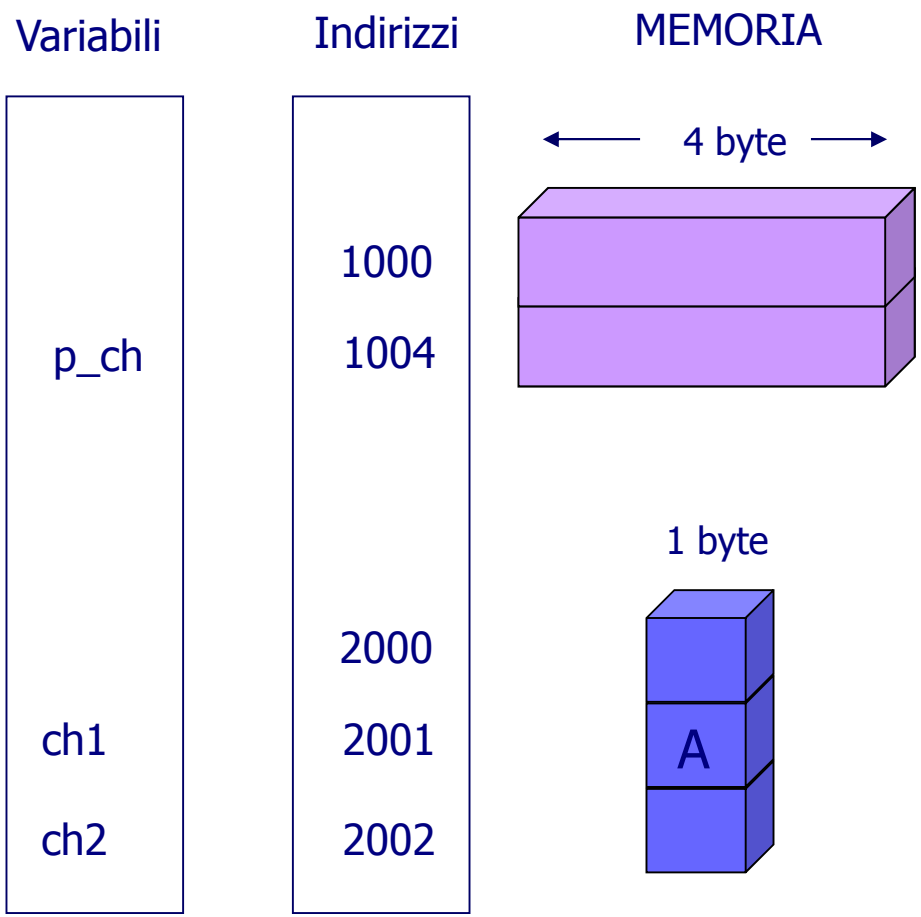
- Si usa l'asterisco * anche per accedere al valore che è memorizzato all'indirizzo di memoria contenuto in una variabile puntatore

```
#include<stdio.h>
#include<stdlib.h>

main()
{
    char *p_ch;
    char ch1='A', ch2;

    printf("L'indirizzo di p_ch è: %p\n", &p_ch);
    p_ch = &ch1;
    printf("Il valore contenuto in p_ch è %p\n, p_ch);
    printf("Il valore contenuto all'indirizzo \
            memorizzato in p_ch è: %c\n", *p_ch);
    ch2 = *p_ch;
}
```

L'accesso a una variabile puntata – 2

CODICE

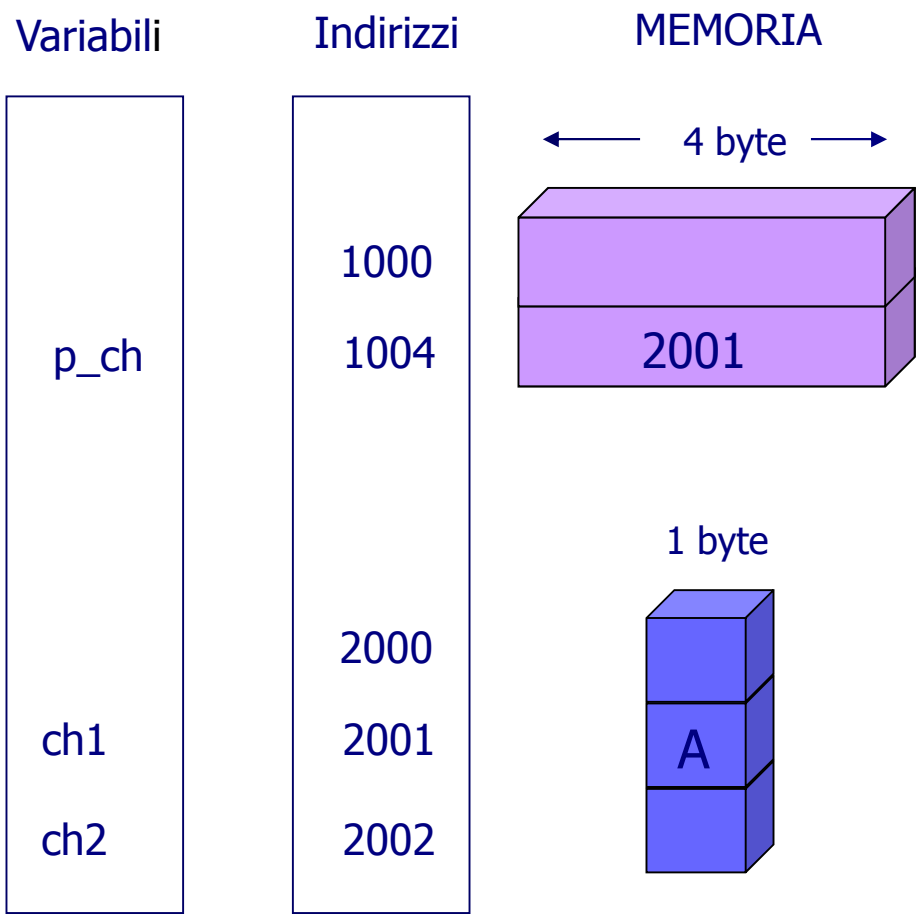
```
char *p_ch;
char ch1='A', ch2;
```



L'accesso a una variabile puntata – 3

CODICE

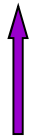
```
p_ch = &ch1;
```



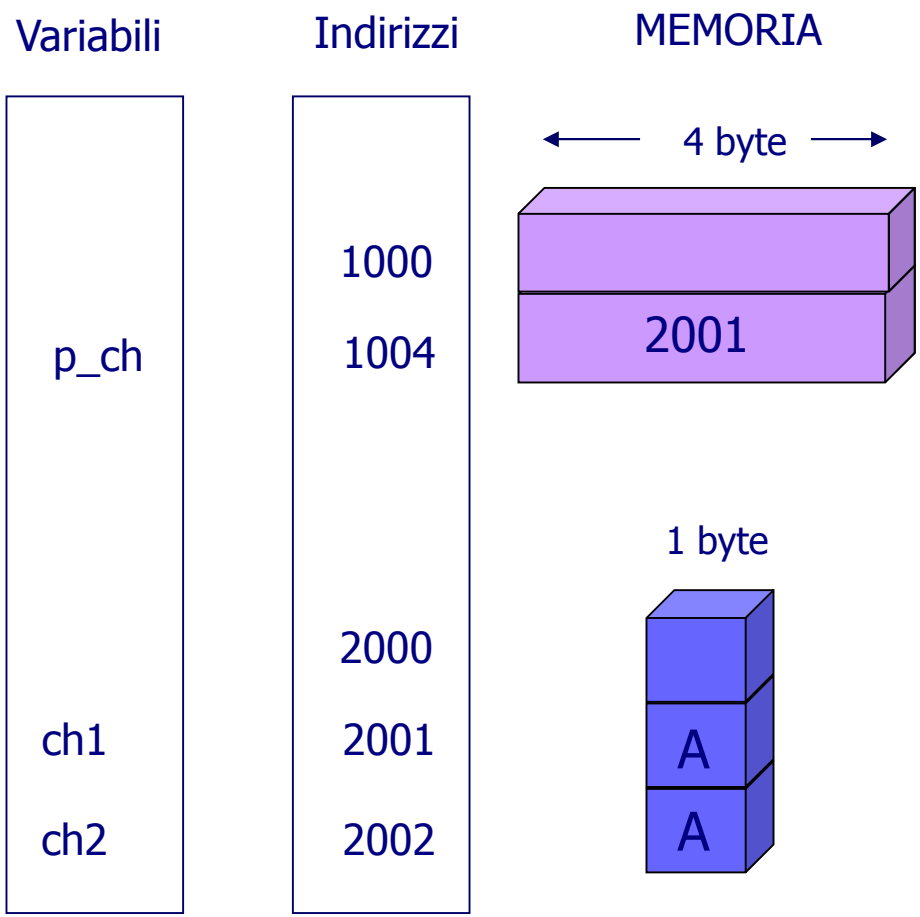
L'accesso a una variabile puntata – 4

CODICE

```
ch2 = *p_ch;
```



Prendi l'indirizzo memorizzato in **`p_ch`** e leggi il valore contenuto nella locazione di memoria identificata da tale indirizzo; copialo in **`ch2`**



Passaggio di parametri per indirizzo

- Le stringhe, ed in generale i vettori e matrici, non possono essere passate per valore come parametro di una funzione!
- Bisogna passarle per riferimento, utilizzando il puntatore al primo elemento dell'array (nome array!)

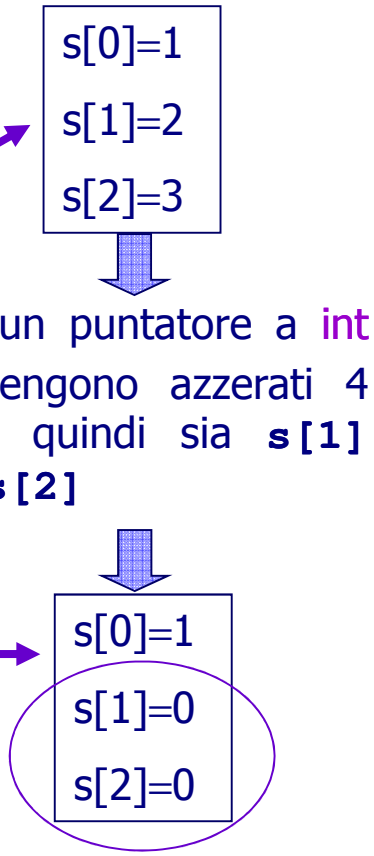
```
char  stringa[] = "prova";
int lunghezza = conta(stringa);
```

- La funzione sarà così dichiarata

```
int conta(char *stringa)
{
   …
}
```

Il passaggio di puntatori come argomenti di funzione - 1

```
#include <stdio.h>
#include <stdlib.h>

void clr(p)
int *p;
{
   *p = 0  /* Memorizza 0 alla locazione p */
}
main()
{
   static short s[3] = {1,2,3};

   clr(&s[1]);  /* Azzera l'elemento 1 di s[] */
   printf("s[0]=%d\ns[1]=%d\ns[2]=%d\n", s[0],s[1],s[2]);
   exit(0);
}
```



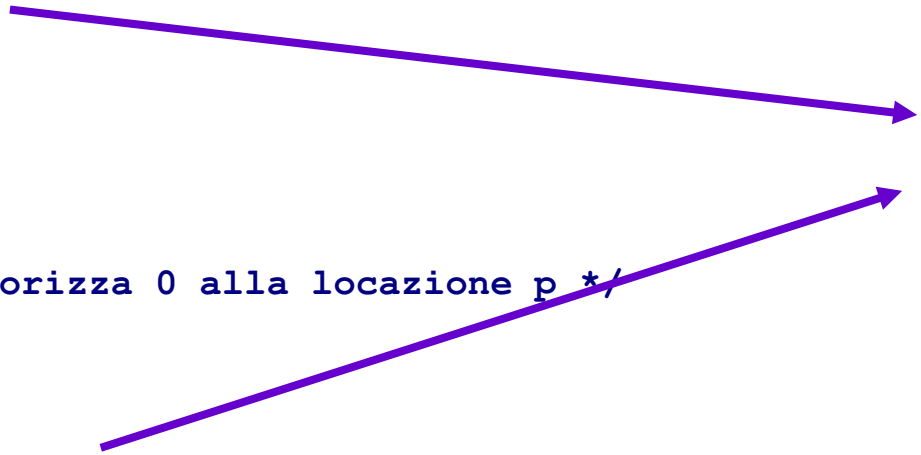
Il passaggio di puntatori come argomenti di funzione – 2

- Il compilatore segnala i tentativi di utilizzo congiunto di puntatori di tipi diversi
- Un'eccezione alla regola è costituita dall'uso di puntatori come argomenti di funzione: in mancanza di un prototipo, il compilatore non effettua controlli per verificare la corrispondenza di tipo fra parametro attuale e parametro formale

  ⇨ si possono produrre risultati inconsistenti nel caso di parametri di tipo disomogeneo
- Il prototipo di una funzione è una dichiarazione di funzione antecedente alla sua definizione: permette al compilatore di compiere il controllo sui tipi degli argomenti che vengono passati alla funzione

Prototipo di una funzione

```
#include <stdio.h>
#include <stdlib.h>

void clr(int *);

void clr(p)
int *p;
{
   *p = 0  /* Memorizza 0 alla locazione p */
}
main()
{
   static short s[3] = {1,2,3};

   clr(&s[1]);  /* Azzera l'elemento 1 di s[] */
   printf("s[0]=%d\ns[1]=%d\ns[2]=%d\n", s[0],s[1],s[2]);
   exit(0);
}
```



p è un puntatore a int

incompatibile con un puntatore a short int!

Implementare la funzione conta()!

- Completare l'implementazione della funzione che conta la lunghezza della stringa

```
int conta(char *stringa)
{
    int l = 0;
   …
   return
}
```

Le funzioni di libreria per le stringhe `strlen()`

- La funzione **`strlen()`**, restituisce il numero di caratteri che compongono una stringa (escluso il carattere nullo)
- Poiché nell'espressione **`*str++`** i due operatori hanno la stessa precedenza ed associatività destra, l'espressione viene analizzata dal compilatore nel modo seguente:
  - Valutazione dell'operatore di incremento postfisso; il compilatore passa **`str`** all'operatore successivo e lo incrementa solo al termine della valutazione dell'espressione
  - Valutazione dell'operatore *, applicato a **`str`**
  - Completamento dell'espressione, con l'incremento di **`str`**

```
int strlen(str)
char *str;
{
  int i;

  for (i=0; *str++; i++)
       ; /* istruzione vuota */
  return i;
}
```

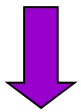
printf() di stringhe

- All'interno della stringa template dobbiamo usare %s.
- Come altro parametro dobbiamo passare il nome della variabile, che usato da solo è il puntatore al primo elemento della stringa
- printf() stamperà quel elemento fino al carattere `'\0'`
  - `char str[] = "stringa";`
  - `printf("%s",str); //stampa` **`stringa`**
  - `printf("%s",str+2); // stampa` **`ringa`**
  - `printf("%s",str[2]); // non corretto!`
  - `printf("%c",str[2]); // stampa` **`r`**

La dichiarazione di tipo: typedef

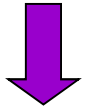
- La dichiarazione di tipo deve apparire in un programma prima che il tipo venga adoperato per la dichiarazione di variabili
- Le dichiarazioni di tipo sono particolarmente utili nella definizione di tipi composti
- Avvertenza: typedef e #define non sono equivalenti…

```
#define PT_TO_INT  int *

PT_TO_INT p1, p2;
```



```
int  *p1, p2;
```

```
typedef  int *  PT_TO_INT;

PT_TO_INT p1, p2;
```



```
int  *p1, *p2;
```