Tipi di dati scalari (casting e puntatori)

Alessandra Giordani
agiordani@disi.unitn.it
Lunedì 9 maggio 20110
http://disi.unitn.it/~agiordani/

I tipi di dati scalari

- I tipi aritmetici, i tipi enumerativi ed i puntatori vengono detti tipi scalari, poiché i valori che li compongono sono distribuiti su una *scala lineare*, su cui si può stabilire una relazione di ordine totale
- La dichiarazione fornisce al compilatore le informazioni relative al numero di byte da allocare e alle modalità di interpretazione di tali byte
- Le parole chiave char, int, float, double, ed enum descrivono i tipi base; short, long, signed, unsigned sono i qualificatori che modificano i tipi base

Le tipologie di costanti intere – 1

- Oltre alle costanti decimali, il C permette la definizione di costanti ottali ed esadecimali
- Le costanti ottali vengono definite anteponendo al valore ottale la cifra 0
- Le costanti esadecimali vengono definite anteponendo la cifra 0 e x o X

Decimale	Ottale	Esadecimale
3	03	0x3
8	010	0X8
15	017	0xF
16	020	0x10
21	025	0x15
–87	–0127	–0x57
187	0273	0xBB
255	0377	0Xff

Le tipologie di costanti intere – 2

- Esempio: Leggere un numero esadecimale da terminale e stampare gli equivalenti ottale e decimale

```
/* Stampa gli equivalenti ottale e decimale
 * di una costante esadecimale
 */
#include<stdio.h>

int main()
{
    int num; char a;

    printf("Digitare una costante esadecimale: ");
    scanf("%x", &num);
    printf("L'equivalente decimale di %x e' %d\n", num, num);
    printf("L'equivalente ottale di %x e' %o\n", num, num);
    return 0;
}
```

Le costanti floating-point

- Le costanti floating–point sono, per default, di tipo double
- Lo standard ANSI consente tuttavia di dichiarare esplicitamente il tipo della costante, mediante l'uso dei suffissi f/F o l/L, per costanti float e long double, rispettivamente

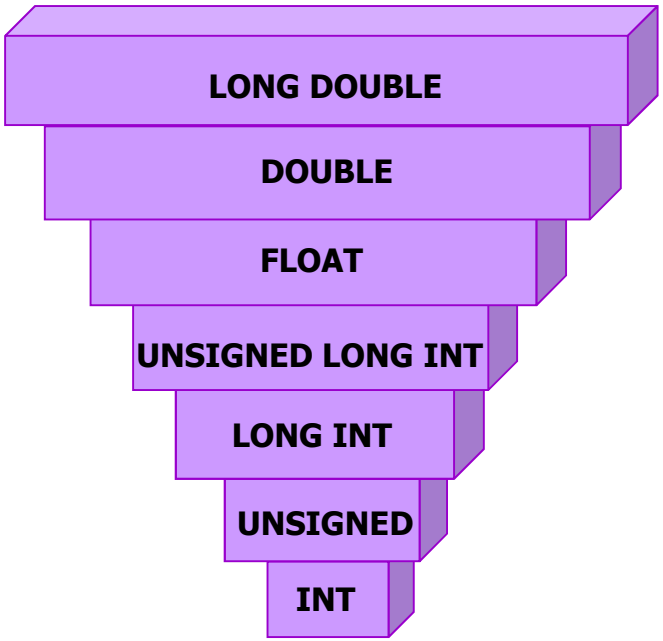
```
//Calcolo dell'area di un cerchio dato il raggio

#define PI  3.14159

float area_of_circle(radius)
float radius;
{
    float area;

    area = PI * radius * radius;
    return area;
}
```

Le combinazioni di tipi – 1

- Le conversioni implicite vengono effettuate in quattro circostanze:
  - Conversioni di assegnamento — nelle istruzioni di assegnamento, il valore dell'espressione a destra viene convertito nel tipo della variabile di sinistra
  - Conversioni ad ampiezza intera — quando un char od uno short int appaiono in un'espressione vengono convertiti in int; unsigned char ed unsigned short vengono convertiti in int, se int può rappresentare il loro valore, altrimenti sono convertiti in unsigned int
  - In un espressione aritmetica, gli oggetti sono convertiti per adeguarsi alle regole di conversione dell'operatore
  - Può essere necessario convertire gli argomenti di funzione

Le combinazioni di tipi – 2

- L'analisi di un'espressione da parte del compilatore ne comporta la suddivisione in sottoespressioni; gli operatori binari impongono operandi dello stesso tipo: l'operando il cui tipo è "gerarchicamente inferiore" viene convertito al tipo superiore:



Esempio**:** La somma fra un int e un double (1+2.5) viene valutata come (1.0+2.5)

Le conversioni di tipo esplicite: cast

- In C, è possibile convertire esplicitamente un valore in un tipo diverso effettuando un cast
- Per realizzare una conversione di tipo esplicita di un'espressione, si pone tra parentesi tonde, prima dell'espressione, il tipo in cui si desidera convertire il risultato

```
#include<stdio.h>

int main()
{
    int j, k;
    float f;

    printf("Inserire due valori j,k. Calcorero j/k con e senza casting\n");
    scanf("%d %d", &j, &k);
    f=j/k;
    printf("valore di f=%d/%d senza casting= %f\n",j,k,f);
    f=(float)j/k; //conversione esplicita
    printf("valore di f=%d/%d con casting= %f\n",j,k,f);
    f=(j*1.0)/k; //conversione implicita
    printf("valore di f=%d/%d con 1.0= %f\n",j,k,f);
    return 0;
}
```

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 * !!!

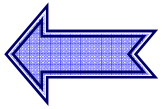
- a rappresenta un indirizzo

- *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

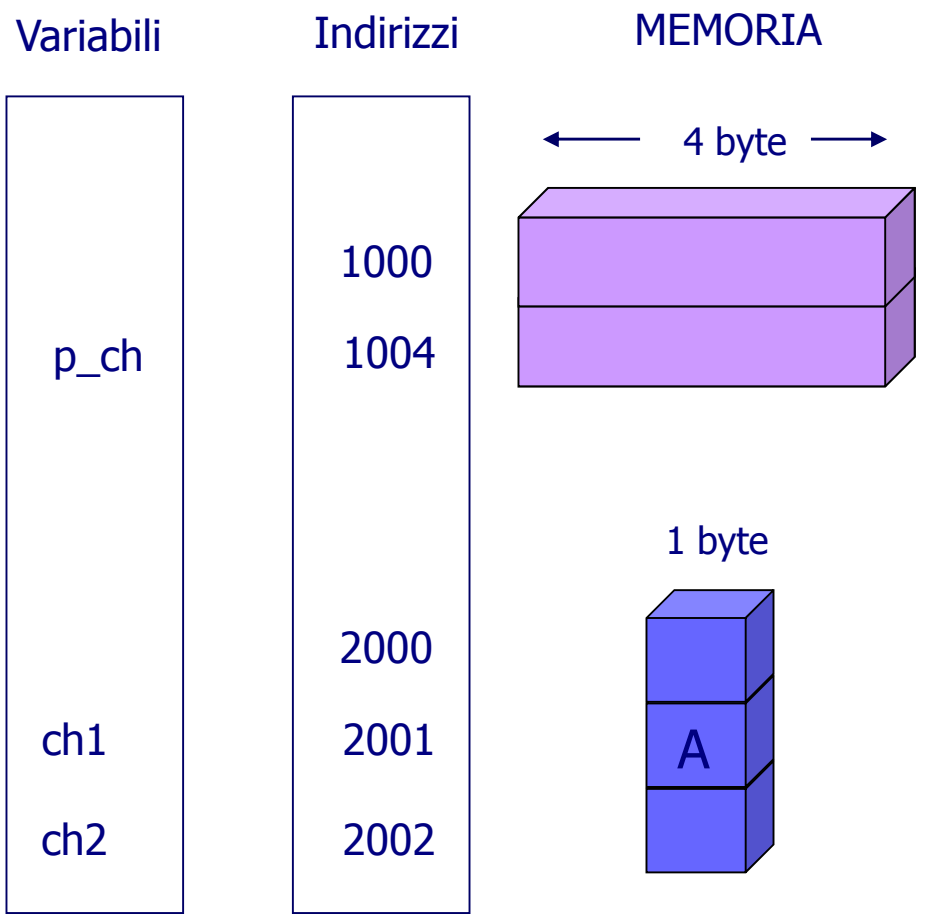
- Si usa l'asterisco * anche per accedere al valore che è memorizzato all'indirizzo di memoria contenuto in una variabile puntatore

```
#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

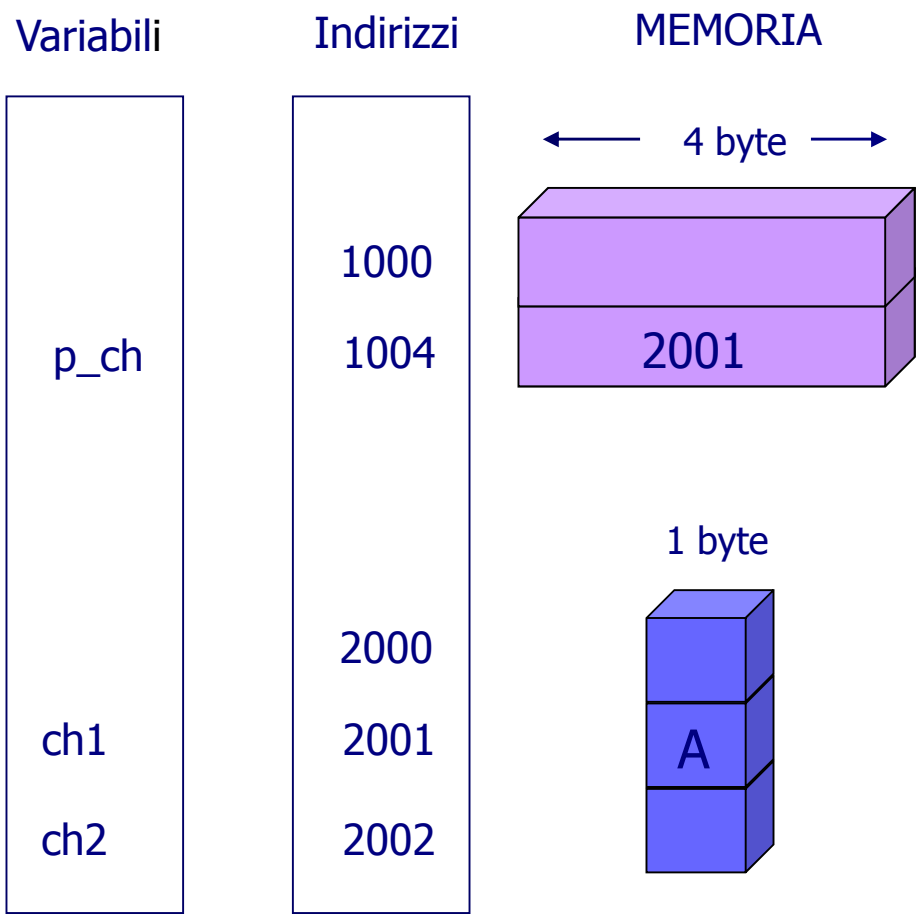
```
char *p_ch;
char ch1='A', ch2;
```



L'accesso a una variabile puntata – 3

CODICE

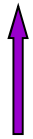
```
p_ch = &ch1;
```



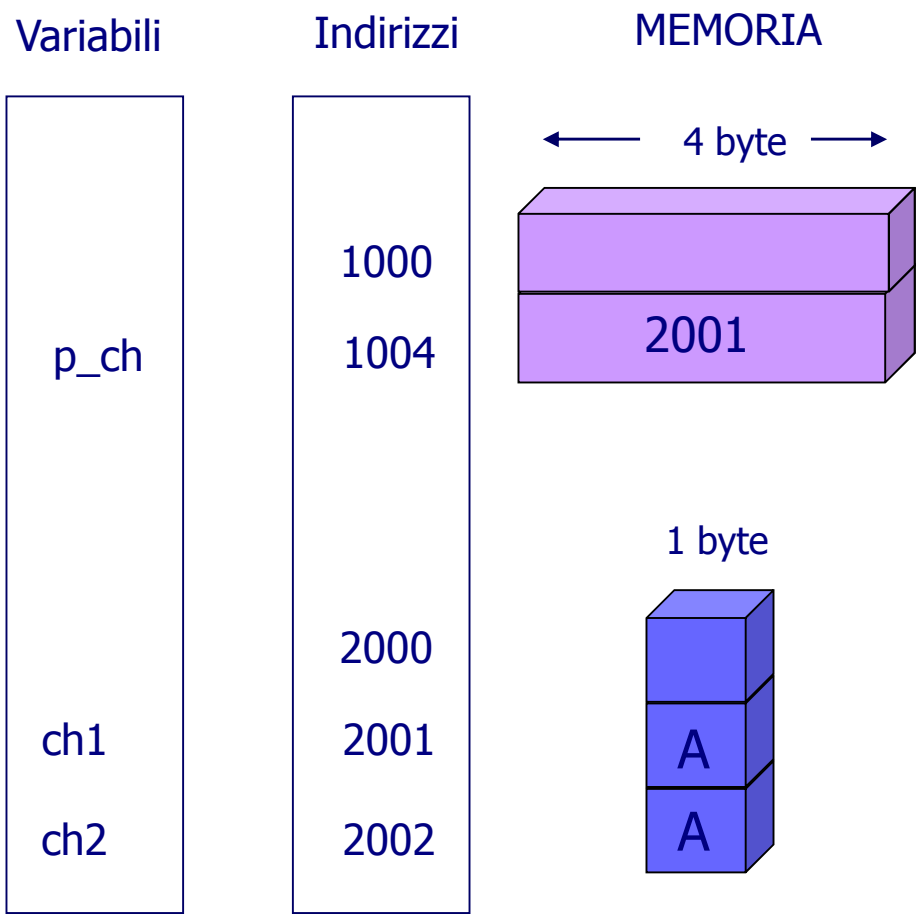
L'accesso a una variabile puntata – 4

CODICE

```
ch2 = *p_ch;
```



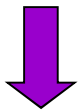
Prendi l'indirizzo memorizzato in **`p_ch`** e leggi il valore contenuto nella locazione di memoria identificata da tale indirizzo; copialo in **`ch2`**



La dichiarazione di tipo: typedef

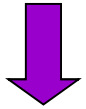
- La dichiarazione di tipo deve apparire in un programma prima che il tipo venga adoperato per la dichiarazione di variabili
- Le dichiarazioni di tipo sono particolarmente utili nella definizione di tipi composti
- Avvertenza: typedef e #define non sono equivalenti…

```
#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;
```