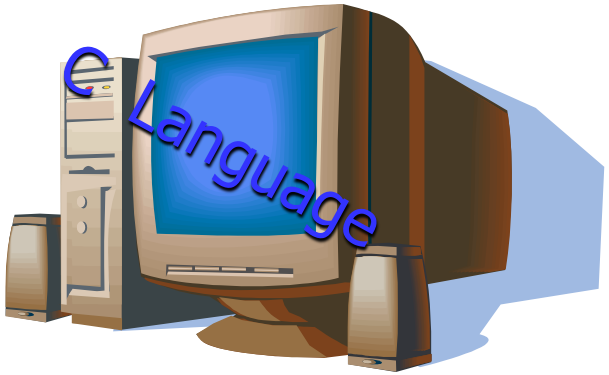
Il linguaggio C

- I tipi di dati scalari
  - Il casting
  - Le dichiarazioni di tipo
  - I puntatori



I tipi di dati scalari

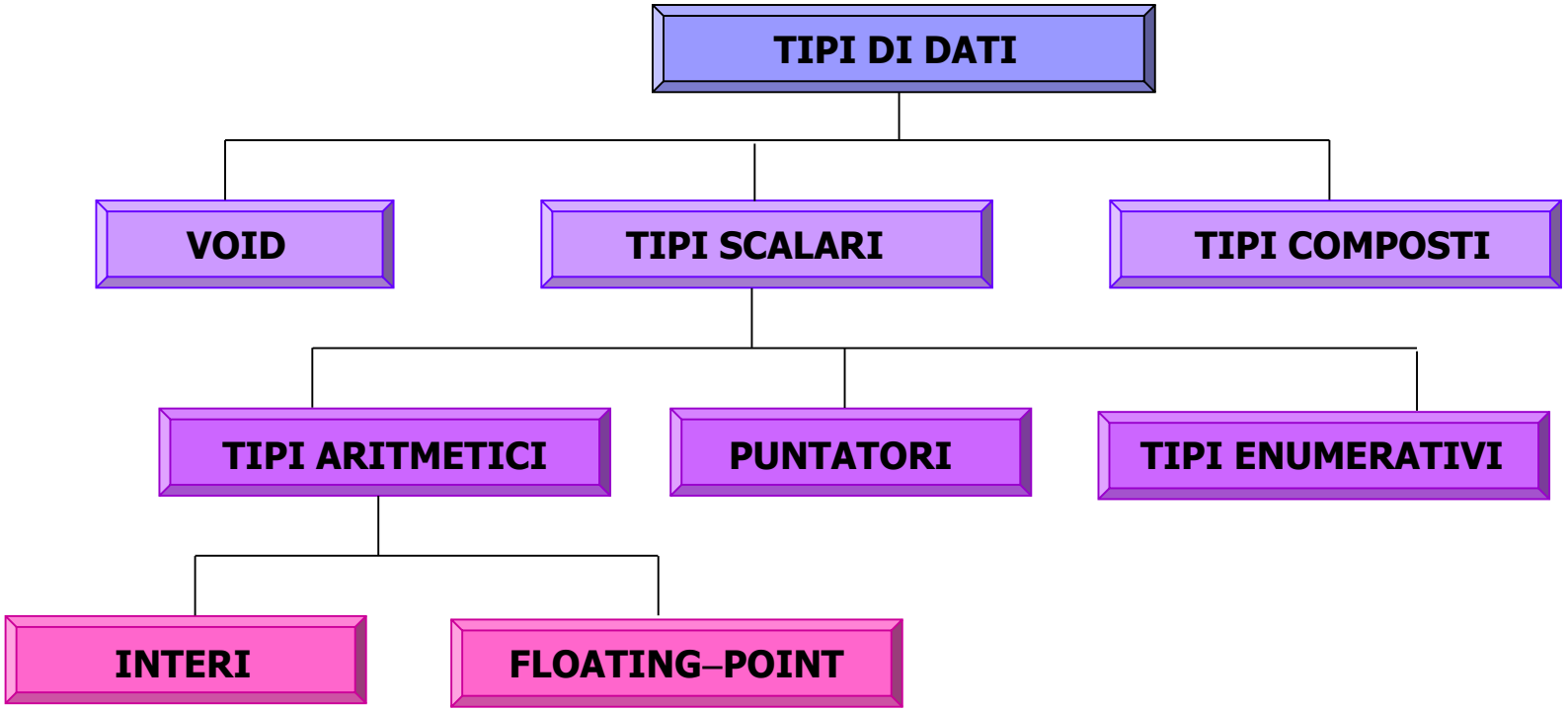
I tipi di dati scalari – 1

- Una delle caratteristiche più importanti dei linguaggi di alto livello è la capacità di classificare i dati in tipi
- È compito del compilatore assicurare che il calcolatore manipoli i bit e i byte in modo consistente con un tipo di dato, che rappresenta solo un'interpretazione applicata a stringhe di bit
- Il linguaggio C rende disponibili otto diversi tipi di numeri interi e tre tipi di numeri floating–point che, complessivamente, costituiscono i tipi aritmetici

I tipi di dati scalari – 2

- 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
- Combinando i tipi scalari si ottengono i **tipi composti** che comprendono **array**, **strutture** ed **unioni**: servono per raggruppare variabili logicamente correlate in insiemi di locazioni di memoria fisicamente adiacenti
- Il **tipo void**, introdotto nello standard **ANSI**, non è né scalare, né composto e si applica, ad esempio, alle funzioni che non restituiscono valori: indica che il dominio della variabile è l'insieme vuoto

I tipi di dati scalari – 3



Gerarchia dei tipi di dati in C

Le dichiarazioni – 1

- Ogni variabile deve essere dichiarata prima di poter essere usata
- La dichiarazione fornisce al compilatore le informazioni relative al numero di byte da allocare e alle modalità di interpretazione di tali byte

char	double	short	signed
int	enum	long	unsigned
float			

Le parole chiave per i tipi scalari

- 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 dichiarazioni – 2

- Per aumentare la concisione, è possibile dichiarare variabili dello stesso tipo separando i loro nomi con virgole

```
int j, k;
float x, y, z;
```

- All'interno di un blocco, tutte le dichiarazioni devono apparire prima delle istruzioni eseguibili; l'ordine relativo delle dichiarazioni non è significativo
- **Nota**: Per i nomi di variabili viene adottata una convenzione mutuata dal **FORTRAN**:
  - I nomi i, j, k, m, n sono utilizzati per contatori e variabili temporanee intere
  - I nomi x, y, z sono utilizzati per variabili temporanee floating–point
  - Il nome c è utilizzato per variabili temporanee carattere
  - Non usare l ed o che si confondono con 1 e 0, rispettivamente

La dichiarazione del tipo di funzione

- Analogamente alla dichiarazione di tipo per le variabili, è possibile dichiarare il tipo del valore restituito da una funzione
- Diversamente dalle variabili, alle funzioni viene associato un tipo di default (**int**), in assenza di dichiarazione esplicita di tipo
- Anche nel caso delle funzioni intere, è comunque buona norma di programmazione dichiarare esplicitamente il tipo

```
float f1(arg)
int arg;
{
   … …
}
```

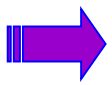
int

```
f2()
{
   … …
}
```

Le tipologie di numeri interi

- Al tipo **int** possono essere assegnate dimensioni diverse su architetture distinte (tipicamente 4 o 8 byte)
- Il tipo **int** rappresenta il formato "naturale" per il calcolatore, ossia il numero di bit che la CPU manipola normalmente in una singola istruzione
- Supponiamo che **int** corrisponda a celle di memoria di 4 byte:
  - Il tipo **short int** corrisponde generalmente a 2 byte
  - Il tipo **long int** a 4/8 byte
- Nelle dichiarazioni di interi **short**/**long** la parola **int** può essere omessa

```
short int j;
long int k;
```



```
short j;
long k;
```

Gli interi senza segno

- Si possono individuare casi in cui una variabile può assumere solo valori positivi (ad es., i contatori)
- Il linguaggio C permette la dichiarazione di **interi senza segno**
- Il bit più significativo non viene interpretato come bit di segno
- Esempio: una variabile **short int** può contenere i numeri interi compresi fra –32768 e 32767, mentre una variabile dichiarata **unsigned short int** può contenere valori da 0 a 65535
- Per dichiarare una variabile intera senza segno deve essere specificato il qualificatore unsigned

```
unsigned (int) p;
```

Gli interi con segno

- Lo standard ANSI prevede anche la parola chiave **signed**
- Lo specificatore **signed** consente di definire esplicitamente una variabile che può assumere valori sia positivi che negativi
- Normalmente **signed** è superfluo, perché i numeri interi sono con segno *per default*
- Fa eccezione il tipo carattere che, per default, è senza segno

Caratteri e interi – 1

- La maggior parte dei linguaggi distingue i caratteri dai dati numerici: 5 è un numero mentre 'A' è un carattere
- In C, la differenza tra carattere e numero è sfumata: il tipo di dati char è un valore intero rappresentato con un byte, che può essere utilizzato per memorizzare sia caratteri che interi
- Per esempio, dopo la dichiarazione

char c;

i seguenti assegnamenti sono corretti ed equivalenti:

c='A';

c=65;

In entrambi i casi, viene assegnato alla variabile **c** il valore 65, corrispondente al codice ASCII della lettera A

Caratteri e interi – 2

- Le costanti di tipo carattere sono racchiuse tra apici singoli
- Esempio: Leggere un carattere da terminale e visualizzarne il codice numerico

```
/* Stampa del codice numerico di un carattere */
#include<stdio.h>
#include<stdlib.h>

main()
{
    char ch;

    printf("Digitare un carattere: ");
    scanf("%c", &ch);
    printf("Il codice numerico corrispondente è %d\n", ch);
    exit(0);
}
```

Caratteri e interi – 3

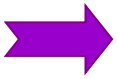
- Dato che in C i caratteri sono trattati come interi, su di essi è possibile effettuare operazioni aritmetiche

```
int  j;
j = 'A'+'B';
```

j conterrà il valore 131, somma dei codici ASCII 65 e 66

- Esempio: Scrivere una funzione che converte un carattere da maiuscolo a minuscolo

Funziona per la codifica ASCII



```
char to_lower(ch)
char ch;
{
    return ch+32;
}
```

- In C, esistono le routine di libreria ***toupper*** e ***tolower*** in grado di convertire anche nel caso di codifiche diverse dall'ASCII

I tipi interi

Tipo	Byte	Rango
int	4	da -2^{31} a $2^{31}-1$
short int	2	da -2^{15} a $2^{15}-1$
long int	4	da -2^{31} a $2^{31}-1$
	8	da -2^{63} a $2^{63}-1$
unsigned int	4	da 0 a $2^{32}-1$
unsigned short int	2	da 0 a $2^{16}-1$
unsigned long int	8	da 0 a $2^{64}-1$
signed char	1	da -2^{7} a $2^{7}-1$
unsigned char	1	da 0 a $2^{8}-1$

Dimensione e rango dei valori dei tipi interi sulla macchina di riferimento

Le tipologie di costanti intere – 1

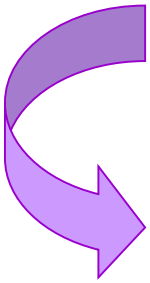
- Oltre alle costanti decimali, il C permette la definizione di costanti ottali ed esadecimali
- Le costanti ottali vengono definite anteponendo al valore ottale la cifra **0**
- Le costanti esadecimali vengono definite anteponendo la cifra **0** e **x** o **X**

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

La costante può essere inserita senza prefisso 0x



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

main()
{
    int num;

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

Le tipologie di costanti intere – 3

- Il numero di byte allocati per una costante intera varia su architetture diverse, in dipendenza delle dimensioni dei tipi interi
- Lo standard ANSI prevede che il tipo di una costante intera sia il primo degli elementi della lista dei tipi associata alla costante, in cui il valore può essere rappresentato

Tipologia delle costanti	Lista dei tipi
Decimale senza suffisso	int, long int, unsigned long int
Ottale o esadecimale	int, unsigned int, long int
Senza suffisso	unsigned long int
Con suffisso u o U	unsigned int, unsigned long int
Con suffisso l o L	long int, unsigned long int

Le tipologie di costanti intere – 4

- Se una costante è troppo grande per il tipo più ampio contenuto nella lista, il valore della costante viene troncato e si produce un messaggio di errore
- È possibile indicare esplicitamente una costante di tipo **long**, aggiungendo **l** o **L** alla costante stessa
- È possibile infine applicare alla costante il qualificatore **unsigned**, postponendo **u** o **U**

55L 55u

077743U 0777777L

–0XAAAB321L 0xfffu

Le sequenze di caratteri di escape

Le sequenze di escape

\a	alert	Produce una segnalazione visiva o sonora
\b	backspace	Muove il cursore di una posizione all'indietro
\f	form feed	Muove il cursore alla pagina successiva
\n	newline	Stampa un carattere di ritorno a capo
\r	carriage return	Stampa un carattere di ritorno carrello
\t	horizontal tab	Stampa un carattere di tabulazione orizzontale
\v	vertical tab	Stampa un carattere di tabulazione verticale

- Il C rende disponibili anche le sequenze **\numero–ottale** e **\numero–esadecimale** che vengono tradotte nel carattere con codifica ASCII pari al numero
- I numeri ottali devono essere espressi senza prefisso, gli esadecimali con il solo prefisso **x**

I tipi floating-point – 1

- Per dichiarare il tipo di dati floating–point devono essere usate le parole chiave **float** o **double**
- La parola **double** significa "doppia precisione": un numero **double** ha infatti una precisione circa doppia rispetto ad un **float** (raddoppia il numero di cifre della mantissa)
- Per il tipo **float** si utilizzano normalmente 4 byte, 8 per i **double**

```
float pi;
double pi_squared;
pi=3.141;
pi_squared=pi*pi;
```

I tipi floating-point – 2

- Esempio: Conversione da gradi Fahrenheit a Celsius

```
/* Conversione di un valore da Fahrenheit
 * in Celsius
 */

double fahrenheit_to_celsius(temp_fahrenheit)
double temp_fahrenheit;
{
    double temp_celsius;

    temp_celsius = (temp_fahrenheit – 32.0) *
                   100.0/(212.0 – 32.0);
    return temp_celsius;
}
```

- Esempio: Calcolo dell'area di un cerchio

```
/* 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 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

Costanti corrette

3.141
.3333333
0.3
3e2
5E–5
3.5f
3.7e12
3.5e3L

Costanti scorrette

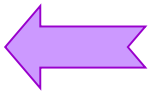
35	Mancano il punto decimale o l'esponente
3,500.45	La virgola non è ammessa
4e	Il simbolo esponente deve essere seguito da un numero
4e3.6	L'esponente deve essere intero

L'inizializzazione – 1

- 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 lasciato nella locazione di memoria dall'esecuzione di un programma precedente

L'inizializzazione – 2

- Il C fornisce una sintassi speciale per inizializzare una variabile, scrivendo un'espressione di assegnamento all'interno di una dichiarazione
- Esempio:

```
char ch='A';
```

alloca un byte per la variabile ch e le assegna il valore 'A'; si ottiene un risultato identico con la coppia di istruzioni

```
char ch;
ch='A';
```

- L'istruzione di inizializzazione è una scorciatoia per combinare una dichiarazione ed un assegnamento in un'unica istruzione

Le combinazioni di tipi – 1

- Nelle espressioni, il C ammette la combinazione di tipi aritmetici:

```
num=3*2.1;
```

l'espressione è la combinazione di un **int** ed un **double**; inoltre num potrebbe essere di qualunque tipo scalare, eccetto un puntatore

- Per associare un significato alle espressioni contenenti dati di tipi diversi, il C effettua automaticamente un insieme di *conversioni implicite*:

```
3.0+1/2
```

verrebbe valutata 3.0 anziché 3.5, dato che la divisione viene effettuata in aritmetica intera

Le combinazioni di tipi – 2

- 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 – 3

- Per le conversioni di assegnamento, sia j un **int** e si consideri…

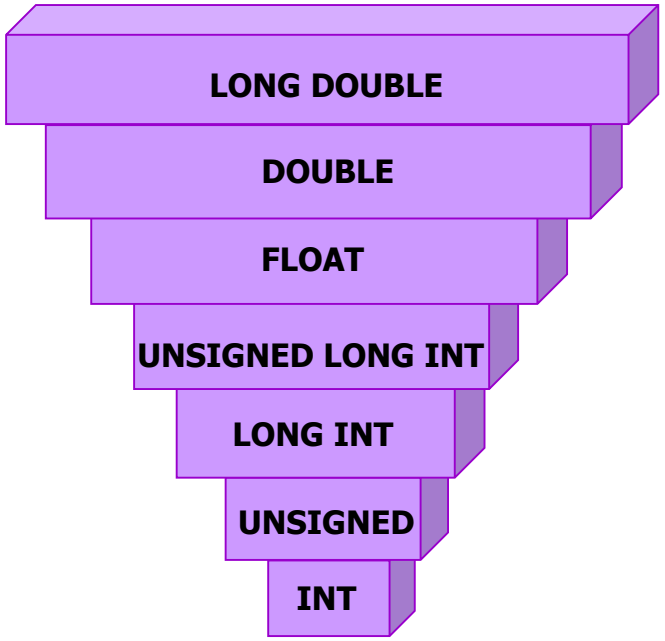
  j=2.6;

  Prima di assegnare la costante di tipo **double**, il compilatore la converte in **int**, per cui j assume il valore intero 2 (agisce per troncamento, non per arrotondamento)

- La conversione ad ampiezza intera o ***promozione ad intero***, avviene generalmente in modo trasparente

Le combinazioni di tipi – 4

- 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)

La combinazione di interi – 1

- I quattro tipi interi (**char**, **short**, **int** e **long**) possono essere combinati liberamente in un'espressione: il compilatore converte i **char** e gli **short** in **int** prima di valutare l'espressione
- La conversione di un intero positivo **short** si riduce all'aggiunta di due ulteriori byte di zero (nelle posizioni più significative)

  5 **short** 00000000 00000101

  5 **int** 00000000 00000000 00000000 00000101

- Per i numeri negativi, si effettua invece un'estensione in segno, aggiungendo byte di uno

  –5 **short** 11111111 11111011

  –5 **int** 11111111 11111111 11111111 11111011

La combinazione di interi – 2

- Si possono verificare errori in fase di assegnamento, quando una conversione implicita riduce la dimensione degli oggetti

- Esempio: se c è una variabile **char**, l'assegnazione a c di 882 non può essere eseguita correttamente, in quanto la rappresentazione dell'intero 882 richiede 2 byte: 00000011 01110010; in c verrebbe memorizzato il byte meno significativo e si otterrebbe c=114 (è il codice ASCII di 'r')

La combinazione di tipi con e senza segno

- La differenza fra tipi interi con e senza segno è la modalità di interpretazione del dato

$$11101010 = \begin{cases} 234 \text{ unsigned} \\ -22 \text{ signed} \end{cases}$$

- L'ANSI C prevede che, se uno degli operandi di un'espressione binaria è **unsigned**, anche il risultato dell'espressione è **unsigned**

$$10u + (-15) = 4294967291 \text{ (su 4 byte)}$$

che, comunque, corrisponde alla stessa sequenza di bit relativa a -5 (ma non viene interpretato in tal modo)

La combinazione di floating-point – 1

- L'uso congiunto di **float**, **double** e **long double** nella stessa espressione fa sì che il compilatore, dopo aver diviso l'espressione in sottoespressioni, ampli l'oggetto più corto di ogni coppia associata ad un operatore binario

- In molte architetture, i calcoli effettuati sui **float** sono molto più veloci che quelli relativi a **double** e **long double**…
  - I tipi di numeri più ampi dovrebbero essere impiegati solo quando occorre una grande precisione o occorre memorizzare numeri molto grandi

- Possono esserci problemi quando si effettuano conversioni da un tipo più ampio ad uno meno ampio
  - Perdita di precisione
  - Overflow

La combinazione di floating-point – 2

- Esempio: Perdita di precisione

  Se f è una variabile di tipo **float** e si esegue l'assegnamento

  f=1.0123456789;

  il calcolatore arrotonda la costante **double** prima di assegnarla ad f ad 1.0123456, se i **float** occupano 4 byte

- Esempio: Overflow

  Se il più grande numero **float** rappresentabile fosse $\approx 7 \times 10^{38}$, l'istruzione

  f=2e40;

  provocherebbe un comportamento non standard (in ANSI), con probabile emissione di un messaggio di errore a *run–time*

La combinazione di interi e floating-point – 1

- La combinazione di valori interi e floating–point è lecita, così come è permesso l'assegnamento di un floating–point ad una variabile intera, o di un valore intero ad una variabile floating–point

- Quando si assegna un intero ad una variabile floating–point, il valore intero viene implicitamente convertito in floating–point prima dell'assegnamento

- Se il tipo **float** non è sufficiente per rappresentare l'intero, si può avere perdita di precisione

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

main()
{
    long int j=2147483600;
    float x;

    x=j;
    printf("j è %d\n x è %f\n", j, x);
    exit(0);
}
```

La combinazione di interi e floating-point – 2

- Se valori interi e floating–point sono usati congiuntamente in un'espressione, il compilatore converte tutti gli interi nel tipo floating–point più ampio

- Esempio: sia j un **int** ed f un **float**, allora…

  j + 2.5 + f

  viene valutata complessivamente in doppia precisione (il tipo della costante), ovvero prima j viene convertito in **double** e, di conseguenza, anche f viene convertito in **double**

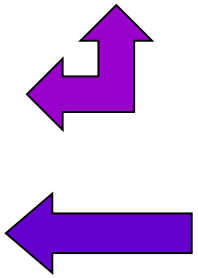
- Nel caso di assegnamento di un valore **float** ad un intero, avviene il troncamento della parte frazionaria (con grave perdita di precisione); inoltre può verificarsi l'overflow

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
- **Esempio**:

```
int j=2, k=3;
float f;

f=k/j;

f=(float)k/j;
```

Assegna 1.0 ad f: la divisione viene effettuata fra interi



Assegna 1.5 ad f: k viene convertito esplicitamente in **float**, j implicitamente; la divisione viene effettuata sui **float**

I tipi enumerativi

- I tipi enumerativi sono utili quando si vuole definire un insieme preciso di valori che possono essere associati ad una variabile

- Esempio:

```
enum {RED, BLUE, GREEN, YELLOW} color;
enum {BRIGHT, MEDIUM, DARK} intensity;
```

- La sintassi per dichiarare i tipi enumerativi è introdotta dalla parola chiave enum, seguita dall'elenco dei nomi delle costanti fra parentesi graffe, seguito dal nome delle variabili

- Ai nomi delle costanti viene associato un valore intero di default, basato sulla loro posizione nell'elenco (a partire da 0)

- Il compilatore ha il compito di allocare la memoria necessaria per un tipo enumerativo: a *color* dovrebbe essere allocato un singolo byte (solo quattro possibili valori)

Il tipo void

- Il tipo di dati **void** viene utilizzato per dichiarare funzioni che non restituiscono un valore

```
void func(a, b)
int a, b;

{
    … …
}
```

- Una funzione di tipo **void** non può essere utilizzata in un'espressione, ma solo richiamata

```
func(x,y);
d = func(x,y);      \* errore *\
```

- Il tipo **void** viene inoltre utilizzato per dichiarare puntatori generici

La dichiarazione di tipo: typedef – 1

- Il C consente di associare ai tipi di dati nomi definiti dal programmatore, mediante la parola chiave typedef
  - Dal punto di vista sintattico, la dichiarazione di tipo è analoga alla dichiarazione di variabile
  - Dal punto di vista semantico, il nome definito diviene un sinonimo di un tipo di dati e la dichiarazione non produce allocazione immediata di memoria

- Esempio:

```
typedef  long int  EIGHT_BYTE_INT;
```

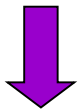
rende EIGHT_BYTE_INT un sinonimo di **long int**

- Per convenzione, i nomi di tipo sono scritti con lettere maiuscole, per non confonderli con i nomi di variabile

La dichiarazione di tipo: typedef – 2

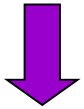
- La dichiarazione di tipo deve apparire in un programma prima che il tipo venga adoperato per la dichiarazione di variabili
- Le dichiarazioni di tipo sono particolarmente utili nella definizione di tipi composti
- Avvertenza: **typedef** e #define non sono equivalenti…

```
#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;
```

Il reperimento dell'indirizzo di un oggetto – 1

- Per ottenere l'indirizzo di una variabile si usa l'operatore **&**
- **Esempio**: Se j è una variabile **long int** con indirizzo 2486, allora l'istruzione…

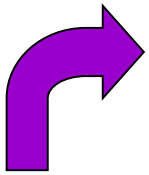
ptr=&j;

memorizza l'indirizzo 2486 nella variabile ptr

- **Esempio**:

L'indirizzo che si ottiene varia per esecuzioni diverse dello stesso programma

%p è lo specificatore per stampare l'indirizzo di un dato



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

main()
{
    int j=1;

    printf("Il valore di j è: %d\n", j);
    printf("L'indirizzo di j è: %p\n", &j);
    exit(0);
}
```

Il reperimento dell'indirizzo di un oggetto – 2

- L'operatore **&** non è utilizzabile nella parte sinistra di un'istruzione di assegnamento
- Non è possibile cambiare l'indirizzo di un oggetto, pertanto…

```
&x = 1000;   \* ILLEGALE *\
```

- È il compilatore — che sfrutta i servizi offerti dal sistema operativo — l'unico gestore della memoria allocata all'esecuzione di un programma

Introduzione ai puntatori – 1

- Nell'istruzione di assegnamento

  ptr = &j;

  la variabile che contiene l'indirizzo di j non può essere una normale variabile intera, ma un tipo speciale di variabile, chiamato puntatore: memorizzando un indirizzo, esso "punta" ad un oggetto

- Per dichiarare una variabile puntatore, si fa precedere al nome un asterisco:

  long *ptr;

  il tipo di dati **long** fa riferimento al tipo di variabile a cui ptr può puntare

Introduzione ai puntatori – 2

ESEMPI

```
/* CORRETTO */
long *ptr;
long long_var;
ptr = &long_var;
```

```
/* NON CORRETTO */
long *ptr;
float float_var;
ptr = &float_var;
```

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

main()
{
    int j=1;
    int *pj;

    pj = &j;  /*Assegna l'indirizzo di j a pj */
    printf("Il valore di j è: %d\n", j);
    printf("L'indirizzo di j è: %p\n", pj);
    exit(0);
}
```

L'accesso a variabile puntata – 1

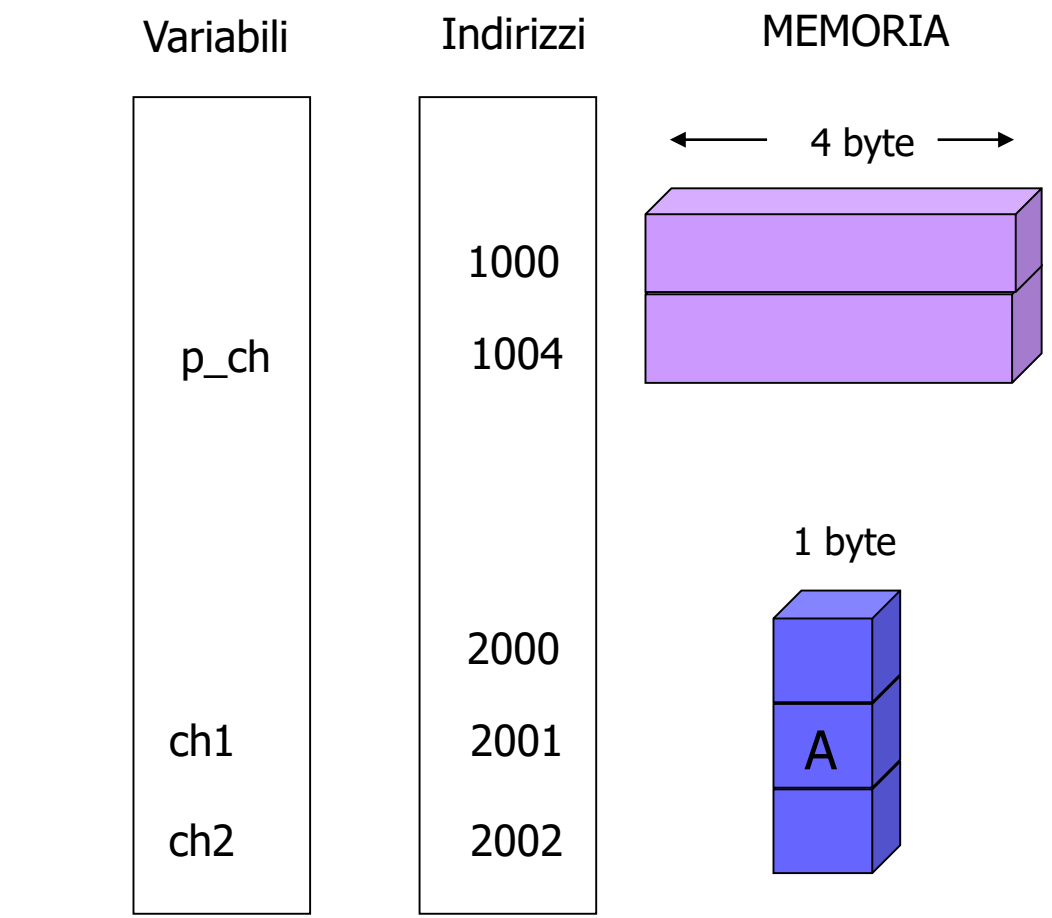
- Si usa l'asterisco * anche per accedere al valore che è memorizzato all'indirizzo di memoria contenuto in una variabile puntatore

```
#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 \
            puntato da p_ch è: %c\n", *p_ch);
    ch2 = *p_ch;
    exit(0);
}
```

L’accesso a variabile puntata – 2

CODICE

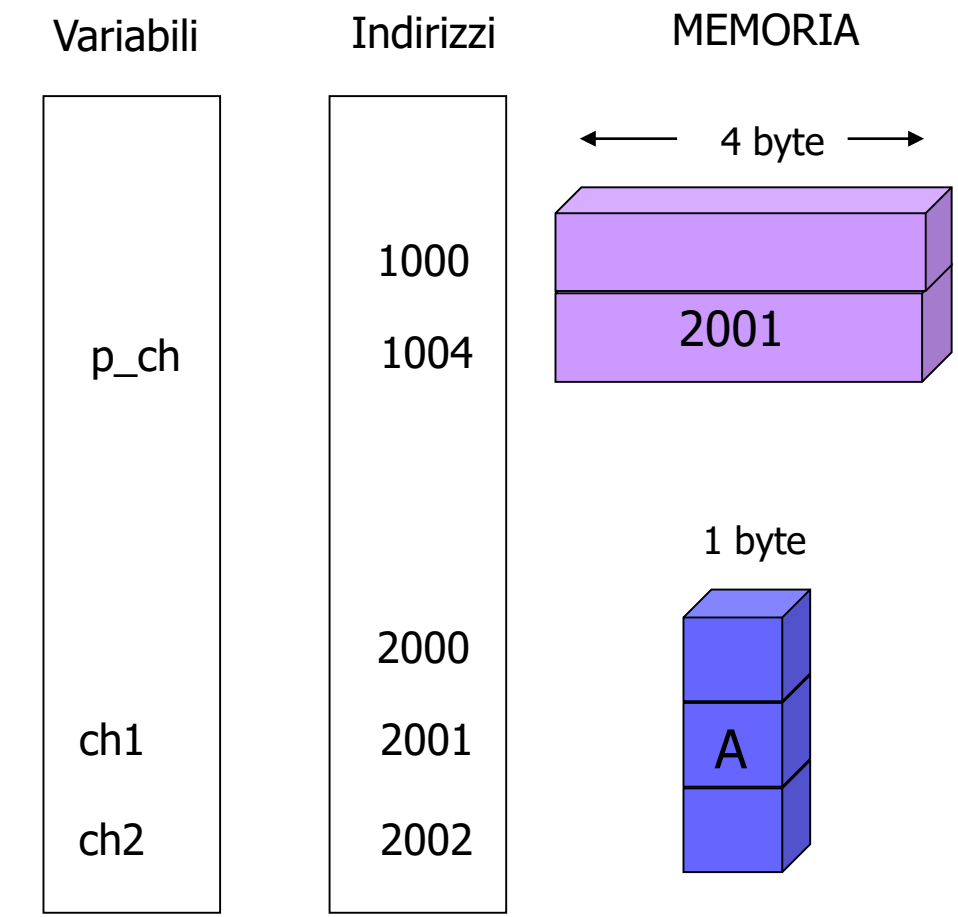
```
char *p_ch;
char ch1='A', ch2;
```



L'accesso a variabile puntata – 3

CODICE

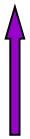
```
p_ch = &ch1;
```



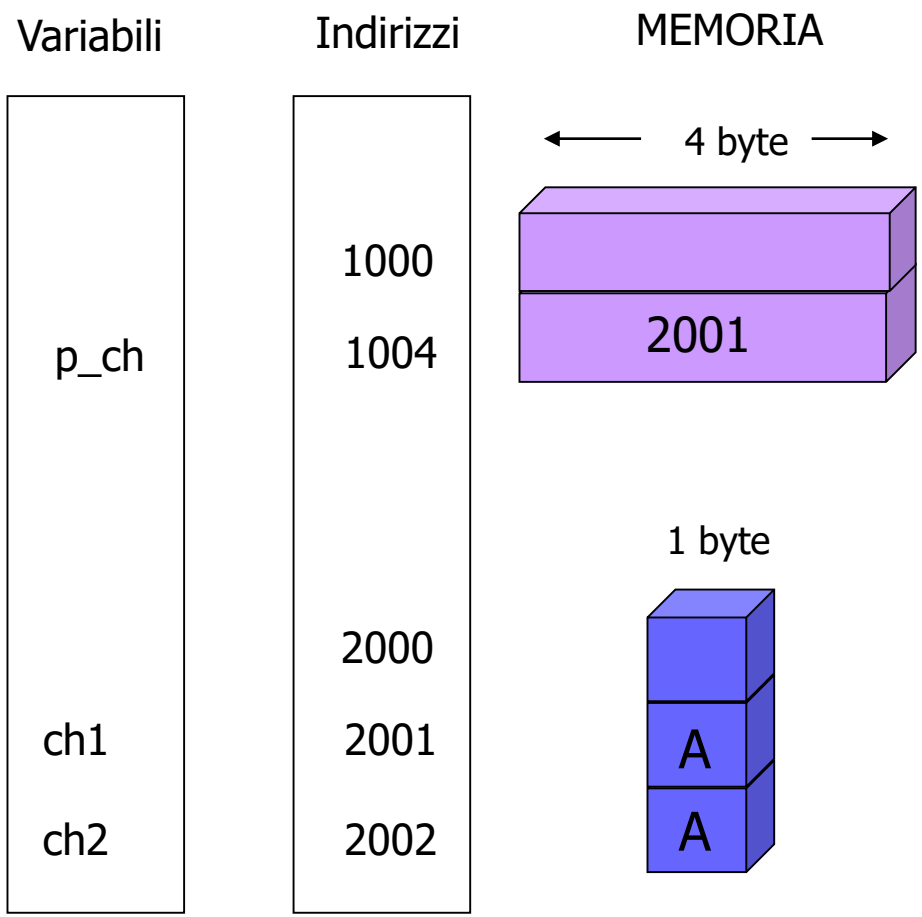
L'accesso a variabile puntata – 4

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



L'accesso a variabile puntata – 5

- Il tipo di dato contenuto nella dichiarazione del puntatore indica il tipo del risultato dell'operazione "accesso all'indirizzo contenuto in"
- Esempio: La dichiarazione

  float *fp;

  significa che quando *fp appare in un'espressione il risultato è di tipo **float**; l'espressione *fp può anche apparire alla sinistra di un'istruzione di assegnamento

  ***fp = 3.15;**

  che memorizza il valore 3.15 nella locazione di memoria puntata da fp

- Esempio: L'assegnazione

  fp = 3.15;

  è scorretta poiché gli indirizzi "non sono numeri" interi né floating–point, e non possono essere "assegnati"

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;
```