# Laboratorio di informatica

Ingegneria meccanica

Lezione 5 – 29 ottobre 2007



## Interi in complemento a 2 (1)

- Con N bit sono rappresentabili tutti gli interi nell'intervallo [-2^(N-1), 2^(N-1)-1]
- Esempio: 16 bit, [-2^15, 2^15-1]
- Valori non negativi/negativi: msb = 0/1
- Esempi: 8 bit, 0111 1111 rappresenta 127, 1000 0000 rappresenta -128,
- Data la rappresentazione R(X) di X, R(-X) si può ottenere così: 1. *invertire* il valore di ogni bit in R(X) (0⟶1, 1⟶0), 2. sommare 1



## Interi in complemento a 2 (2)

- Esempio su 8 bit: 0011 1101 (61)

  1. 0011 1101 ⟶ 1100 0010

  2. 1100 0010 + 1 ⟶ 1100 0011 (-61)
- E' una forma di rappresentazione posizionale in cui, su N bit, il valore dell'msb è -2^(N-1)
- *Estensione del segno*: attenzione alla rappresentazione di numeri negativi quando si aumenta il numero di bit

  61 ⟶ 0000 0000 0011 1101 su 16 bit

  -61 ⟶ 1111 1111 1100 0011 su 16 bit



## Interi in complemento a 2 (3)

- *Se* si resta nei limiti di rappresentazione previsti per N bit, il risultato di somme/sottrazioni tra numeri in complemento a 2 può essere ottenuto con la somma ordinaria, troncando il risultato su N bit

- Es. 8 bit: -61 + 43 = -18 (solo 8 bit)

  1100 0011 +
  0010 1011 =
  1110 1110

- Es. 8 bit: 61 + (-61) = 0 (nono bit scartato)

  0011 1101 +
  1100 0011 =
  1 0000 0000

## Interi in complemento a 2 (4)

- Es. 8 bit: 61 + (-43) = 18
  (nono bit scartato)

```
  0011 1101 +
  1101 0101 =
1 0001 0010
```

- Es. 8 bit: -128 + (-1) = -129
  (fuori limite! (overflow))

```
  1000 0000 +
  1111 1111 =
1 0111 1111
```

- Es. 8 bit: 127 + 1 = 128
  (fuori limite! (overflow))

```
  0111 1111 +
  0000 0001 =
1 1000 0000
```



## C: qualificatori di tipo

- Tipi base: char (caratteri), int (interi), float e double (reali)
- E' possibile modificare alcune caratteristiche dei tipi base mediante i *qualificatori* short, long, signed e unsigned
- short e long possono modificare il numero di bit utilizzato per la rappresentazione
- signed e unsigned possono modificare l'insieme dei valori rappresentati
- Esempio:
  long int (interi, intervallo rappresentato include quello rappresentato con int)



## C: Tipo short int

- Intervallo *minimo*: [-32767, 32767]
- Intervallo *vero*: dipende dall'implementazione; è specificato da SHRT_MIN e SHRT_MAX in "limits.h"
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 16 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella in complemento a 2
- Nelle dichiarazioni, short int o short
- Specifica di conversione: %hd



## C: Tipo int

- Intervallo *minimo*: [-32767, 32767]
- Intervallo *vero*: dipende dall'implementazione; è specificato da INT_MIN e INT_MAX in "limits.h"; deve includere l'intervallo usato per il tipo short int
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 16 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella in complemento a 2
- Nelle dichiarazioni int
- Specifica di conversione: %d

## C: Tipo long int

- Intervallo *minimo*: [-2147483647, 2147483647]
- Intervallo *vero*: dipende dall'implementazione; è specificato da LONG_MIN e LONG_MAX in "limits.h"; deve includere l'intervallo usato per il tipo int
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 32 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella in complemento a 2
- Nelle dichiarazioni: long int o long
- Specifica di conversione: %ld



## C: Tipo unsigned short int

- Intervallo *minimo*: [0, 65535]
- Intervallo *vero*: dipende dall'implementazione, è specificato da USHRT_MAX in "limits.h" (valore minimo è sempre 0)
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 16 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella posizionale
- Nelle dichiarazioni: unsigned short int o unsigned short
- Specifica di conversione: %hu, %ho (ottale), %hx o %hX (esadecimale, lettere minuscole o maiuscole)



## C: Tipo unsigned int

- Intervallo *minimo*: [0, 65535]
- Intervallo *vero*: dipende dall'implementazione, è specificato da UINT_MAX in "limits.h" (valore minimo è sempre 0)
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 16 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella posizionale
- Nelle dichiarazioni: unsigned int o unsigned
- Specifica di conversione: %u, %o (ottale), %x o %X (esadecimale, lettere minuscole o maiuscole)



## C: Tipo unsigned long int

- Intervallo *minimo*: [0, 4294967295]
- Intervallo *vero*: dipende dall'implementazione, è specificato da ULONG_MAX in "limits.h" (valore minimo è sempre 0)
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 32 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella posizionale
- Nelle dichiarazioni: unsigned long int o unsigned long
- Specifica di conversione: %lu, %lo (ottale), %lx o %lX (esadecimale, lettere minuscole o maiuscole)

## Specifiche di conversione per tipi interi

| short int | int | long int |
|---|---|---|
| %hd | %d | %ld |

| unsigned short int | unsigned int | unsigned long int |
|---|---|---|
| %hu | %u | %lu |
| %ho | %o | %lo |
| %hx | %x | %lx |
| %hX | %X | %lX |



## C: Operandi interi (1)

- Variabili di uno dei tipi int
- Costanti usate direttamente nelle espressioni
- Esempio: b = 2 * a + 33 * b - c / 19 ;
- Una costante viene specificata con

  segno: + o -, opzionale

  sequenza di cifre: in base 10, 8 (prefisso 0, [0-7]), 16 (prefisso 0x o 0X, [0-9], [a-f] o [A-F] )

  suffisso: u o U per unsigned, l o L per long
- Esempi: -165438L, 0xFFFFFFFFl, -0765, 0XaAaA1, +2147483647L



## C: Operandi interi (2)

- Costanti introdotte con #define
- Forma: #define nome valore
- Effetto: *ogni* occorrenza successiva di nome sarà rimpiazzata con valore (*qualunque* esso sia!)
- Nome: stesse regole date per il nome di variabili
- #define è una direttiva per il compilatore (elaborata dal preprocessore a *tempo di compilazione*)
- Uso tipico: per modificare valori di costanti *senza interventi pesanti* sul testo del programma (si ricompila il programma dopo aver aggiornato solo il valore che compare nella #define)



## C: Tipo di una costante

- Il *tipo* di una costante intera *dipende* da come viene specificata
- Base 10, senza suffisso: primo tipo possibile tra int, long int e unsigned long int
- Base 8 o 16, senza suffisso: primo tipo possibile tra int, unsigned int, long int e unsigned long int
- Con suffisso u o U: primo tipo possibile tra unsigned int e unsigned long int
- Con suffisso l o L: primo tipo possibile tra long int e unsigned long int

## C: Operatori aritmetici per interi (1)

- **Operatori binari** (due operandi): somma (+), sottrazione (-), prodotto (*), quoziente (/), resto (%)
- **Operatori unari** (un operando): segno (+), inversione segno (-)
- **Attenzione**: se il risultato di un'operazione eccede i limiti della rappresentazione, il comportamento *dipende* dall'implementazione
- **Attenzione**: se almeno uno degli operandi è negativo, il comportamento di / e % *dipende* dall'implementazione



## C: Operatori ++ --

- Operatori *unari*: incremento (++) e decremento (--)
- ++ e -- sono applicabili solo a variabili e provocano incremento o decremento di 1
- Possono apparire come suffisso (p.es. x++) o come prefisso (p.es. --k)
- Suffisso definisce postincremento (++) o postdecremento (--) della variabile: si usa il valore corrente della variabile e poi lo si incrementa o decrementa di 1
- Prefisso definisce preincremento (++) o predecremento (--) della variabile: si incrementa o decrementa di 1 il valore corrente della variabile e poi si usa il valore



## Esempio

```
int main ( ) {
    int c = 2 ;                 /* stampati */
    printf( "%d\n", c ) ;       /*    2     */
    printf( "%d\n", c++ ) ;     /*    2     */
    printf( "%d\n", c ) ;       /*    3     */
    c = 2 ;
    printf( "%d\n", c ) ;       /*    2     */
    printf( "%d\n", ++c) ;      /*    3     */
    printf( "%d\n", c ) ;       /*    3     */
    return 0 ;
}
```



## C: Operatori di assegnamento

- Operatore = usato nella forma
  variabile = espressione
- Operatore +=, -=, *=, /=, %= (indicato qui con op=) usato nella forma
  variabile op= espressione
- Significato
  variabile = variabile op (espressione)
  Esempi:
  a += 3*b + c → a = a + (3*b + c)
  x *= 3 + b - 2*c → x = x * (3 + b - 2*c)

# Dati strutturati in vettori (array)

- Array: struttura composta da elementi omogenei (tutti dello stesso tipo)
- Ogni elemento di un dato array è individuato univocamente mediante una combinazione di indici interi
- Array monodimensionali: un indice (vettori)
- Array bidimensionali: due indici (matrici rettangolari)
- Array multidimensionali: N indici (matrici N-dimensionali)



# Array in C (1)

- Un array è individuato da nome, tipo elementi e dimensioni

  ESEMPIO

  int vect[8] ; dichiarazione dell'array vect di 8 elementi di tipo int
- Per un array di N elementi, l'indice assume i valori 0, 1, …, N-1
- L'elemento di indice i di un array di nome A si indica con la scrittura A[i] (se A ha dimensione N i suoi elementi sono A[0], A[1],…, A[N-1])

  I singoli elementi di un array possono essere trattati come variabili del tipo dichiarato



# Array in C (2)

- Inizializzazione di un array

  Gli elementi di un array possono essere inizializzati al momento della dichiarazione

  ESEMPIO

  int a[3] = {2, 7, 9}; produce

  a[0] = 2, a[1] = 7, a[2] = 9,
- Se non vengono forniti valori per tutti gli elementi, si usano quelli forniti per *i primi indici* e per i mancanti si usa il valore 0

  ESEMPIO

  int a[3] = {2, -4}; produce

  a[0] = 2, a[1] = -4, a[2] = 0,



# Array in C (3)

ESEMPIO

int bv[5] = {0}; produce

bv[0] = 0, bv[1] = 0, …, bv[4] = 0,

- Se ci sono più inizializzatori che elementi viene segnalato un errore

  ESEMPIO

  int bv[5] = {0, 1, 2, 3, 4, 5}; produce ERRORE
- Possibile assegnare valori ai singoli elementi anche dopo la dichiarazione dell'array

  ESEMPIO

  int bv[5] ; bv[0] = 0; bv[1] = 0; (attenzione: valore dei rimanenti elementi non definito)

# Array in C (4)

- Esempio di utilizzazione: acquisizione di dati omogenei da stdin da sottoporre a qualche tipo di elaborazione. Dichiarato un array di dimensione sufficiente, si può gestire l'acquisizione con un ciclo sull'indice dell'array

```
…
int vetc[DIM] , i ;
printf( "Lettura vettore di %d interi\n" , DIM ) ;
for ( i = 0 ; i < DIM ; i = i + 1 )
        scanf( "%d" , &vect[i] ) ;
…
```



# Array in C (5)

- Esempio: calcolo del massimo elemento nell'array vect di DIM interi int e dell'indice corrispondente

```
…
i_m = 0 ;
for ( i = 1 ; i < DIM ; i = i + 1 )
  if ( vect[ i ] > vect[ i_m ] )
        i_m = i ;
printf( "indice del massimo: %d" , i_m ) ;
printf( "valore del massimo: %d" , vect[ i_m ] );
…
```



# Esempio

- Programma che legge da stdin 10 numeri interi inserendoli in un vettore, copia poi questo vettore (in ordine inverso) in un secondo vettore, ed infine stampa gli elementi dei due vettori



# Soluzione

```
#define DIM 10

int main( ) {
int vet1[ DIM ] , vet2[ DIM ] , i ;
printf( "Caricamento %d elementi nel vettore\n", DIM);
for( i = 0 ; i < DIM ; i++) {
     printf( "Inserire elemento %d \n" , i );
     scanf( "%d" , &vet1[ i ] ) ;
}
for( i = 0 ; i < DIM ; i++)
  vet2[ DIM -1 -i ] = vet1[ i ];
printf( "\nElementi primo vettore \n" );
for( i = 0 ; i < DIM ; i++ )
  printf ("posizione: %d valore: %d\n" , i , vet1[ i ] );
printf( "\nElementi secondo vettore \n" ) ;
for( i = 0 ; i < DIM ; i++ )
  printf ("posizione: %d valore: %d\n" , i , vet2[ i ] ) ;
return 0;
}
```

# Esercizio

- **Programma che legge da stdin 10 numeri interi inserendoli in un vettore, inverte l'ordine degli elementi nel vettore senza usare un secondo vettore, ed infine stampa gli elementi del vettore**