TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA
Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Linguaggio C: Variabili

Valeria Cardellini

Corso di Calcolatori Elettronici
A.A. 2019/20

Variabili

- In C una variabile rappresenta una locazione di memoria usata per la memorizzazione di un valore ed è caratterizzata da:
  - Nome
    - ✓ Sequenza di caratteri che permette di identificare la variabile
    - ✓ Regole precise per la definizioni di variabili
  - Tipo
    - ✓ Specifica il tipo di dato che la variabile può memorizzare
    - ✓ Es: intero, reale, carattere, „
  - Indirizzo
    - ✓ Locazione di memoria nella quale la variabile è memorizzata
    - ✓ La dimensione (uno o più byte) dipende dal tipo
  - Valore
    - ✓ Valore memorizzato nella variabile ad un certo istante

Identificatori di variabili

- Sequenza di caratteri composta da lettere, cifre e da _ (stesse regole in Python)
  - Inizia per lettera o per _
  - Qualsiasi lunghezza (ma 31 caratteri per C ANSI)
- C è case sensitive
  - Lettere maiuscole e minuscole sono considerate caratteri diversi
- Alcuni identificatori sono riservati perché parole chiave del linguaggio
  - Es: `include`, `return`, `main`, `function`
- Esempi di identificatori non validi
  - `4game`, `un gatto`, `return`

Dichiarazione di variabili

- Il C prevede che le variabili siano sempre esplicitamente dichiarate
- Sintassi
  - `tipo identificatore1, …, identificatoreN`
    - ✓ `tipo`: tipo della variabile (predefinito o definito nel programma)
    - ✓ `identificatoreK`: nome della variabile
  - Osservazione: è possibile dichiarare più variabili dello stesso tipo in un'unica dichiarazione
    - ✓ Es: `int x, y;`
- Semantica
  - La dichiarazione riserva spazio in memoria per la variabile
  - Rende la variabile disponibile nella parte del programma dove appare la dichiarazione

Inizializzazione di variabili

- Inizializzare una variabile significa specificare un valore che viene assegnato ad essa prima di qualsiasi suo utilizzo
  - Una variabile non inizializzata ha valore indefinito
  - Attenzione: se il valore di una variabile non inizializzata viene acceduto prima che essa sia oggetto di una assegnazione, ciò può comportare un errore nell'esecuzione del programma
    - ✓ Errore semantico, non segnalato dal compilatore
    - ✓ La locazione di memoria assegnata alla variabile contiene un valore (non prevedibile e non significativo), usato in caso di mancata inizializzazione
- L'inizializzazione può essere contestuale alla dichiarazione:

  tipo nomevariabile = espressione;

Esempio: inizializzazione di variabili

Codice sorgente: var_init.c

```
/* Errore: mancata inizializzazione della variabile */

#include <stdio.h>

int main(void)
{
        int x;

        printf("x = %d\n", x);

        return 0;
}
```

Lettura: funzione scanf

- Legge da stdin (standard input, ovvero tastiera) i valori di una sequenza di variabili
- E' definita in `stdio.h` e ha la forma:

  `scanf(stringa di controllo del formato, altri argomenti)`
  - In realtà il prototipo è:

    ```
    $> man scanf
    int scanf(const char *format, ...);
    ```
  - La stringa di controllo del formato descrive il formato dell'input, tramite una serie di specifiche di conversione:
    - ✓ tipo del dato da leggere
    - ✓ (opzionale) massimo numero di caratteri da leggere
    - ✓ (opzionale) formato nel quale andrà letto
  - Gli altri argomenti specificano le variabili da leggere
  - Restituisce il numero di variabili lette con successo

Lettura: funzione scanf

- In `scanf` le variabili da leggere devono essere indicate non con il nome ma con l'indirizzo

  ```
  int x;
  scanf("%d", &x);
  ```
  - Operatore di indirizzo &
    - ✓ Applicato ad una variabile, restituisce l'indirizzo della locazione in cui è contenuta la variabile
- Cosa accade se invece di &x avessi scritto x?
  - Poiché x è di tipo intero, il suo valore viene preso come indirizzo della variabile da leggere
  - Nell'esempio, non essendo x inizializzata, il risultato è imprevedibile
    - ✓ Possibile errore di segmentazione (segmentation fault)

Esempio: funzione scanf

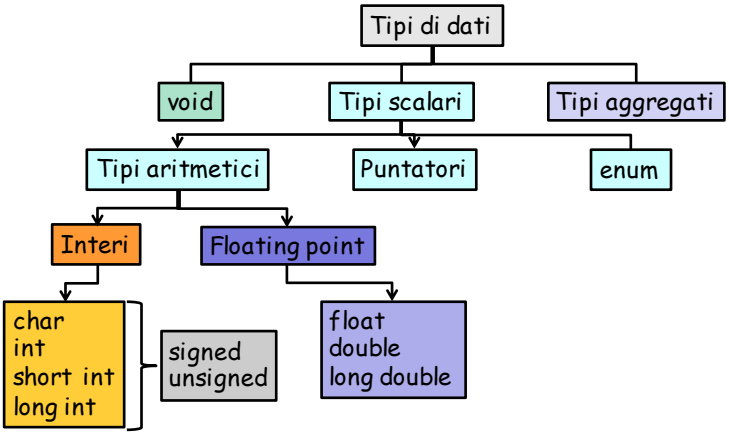
```
#include <stdio.h>
int main()
{
        int x, code;

        printf("Inserisci un intero: ");
        code = scanf("%d", &x);
        printf("Ho letto %d\n", x);
        printf("Codice di ritorno: %d\n", code);
        return 0;
}
```

Codice sorgente: lettura.c

- Possibile leggere più variabili con la stessa scanf
- Cosa succede se l'utente digita una lettera?
- Cosa succede se nella scanf non specifichiamo l'indirizzo della variabile?

Tipi di dato

- Tipi scalari e aggregati (o strutturati)
  - Tipi aggregati (ad es. array e struct) costruiti a partire dagli scalari



Tipi di dato: quantificatori

- Quantificatori: short e long
  - Influiscono sullo spazio in memoria richiesto per l'allocazione del dato
- short: la rappresentazione della variabile in memoria può utilizzare un numero di bit minore rispetto alla norma
- long: la rappresentazione della variabile in memoria può utilizzare un numero di bit maggiore rispetto alla norma
- I quantificatori possono influire sull'intervallo di rappresentazione
  - In generale: dimensione(short int) <= dimensione (int) <= dimensione(long int)
  - In generale: dimensione(float) <= dimensione(double) <= dimensione (long double)

Tipi di dato: qualificatori

- Qualificatori: signed e unsigned
  - unsigned: numeri naturali (interi senza segno)
    - ✓ Intervallo di rappresentazione per unsigned int (n bit): $[0, +2^n-1]$
  - signed: interi con segno
    - ✓ Intervallo di rappresentazione per int (n bit): $[-2^{n-1}, +2^{n-1}-1]$ assumendo rappresentazione in complemento a 2

Domini dei principali tipi scalari aritmetici

- Attenzione: le dimensioni (e quindi gli intervalli) dipendono dal sistema (hw/sw)
  - Di seguito i valori tipici per architetture a 32 bit

Tipo	*Dimensione (bit)*	*Intervallo*	*Specificatori*
char	8	$[-2^7, 2^7-1]$	%c
unsigned char	8	$[0, 2^8-1]$	%c
short int	16	$[-2^{15}, 2^{15}-1]$	%d *
int	32	$[-2^{31}, 2^{31}-1]$	%d *
unsigned int	32	$[0, 2^{32}-1]$	%u
long	32	$[-2^{31}, 2^{31}-1]$	%ld
unsigned long int	32	$[0, 2^{32}-1]$	%lu
float	32	$[3{,}4*10^{-38}, 3{,}4*10^{+38}]$	%f
double	64	$[1{,}7*10^{-308}, 1{,}7*10^{+308}]$	%lf
long double	80	$[3{,}4*10^{-4932}, 1{,}7*10^{+4932}]$	%Lf

* %d per rappresentazione in base 10, %o base 8, %x base 16

Quanti bit sono usati per un tipo?

- Come conoscere le dimensioni di un tipo?

1. Gli header file `limits.h` e `float.h` specificano i massimi ed i minimi numeri rappresentabili per i tipi aritmetici

Codice sorgente: tipi_limiti.c

1. Per conoscere il numero di byte usato da un tipo di dato si usa l'operatore sizeof
   - Operatore speciale (e molto usato) del linguaggio C che, applicato ad un tipo, restituisce il numero di byte usati per memorizzare quel tipo
   - Es: `printf("dimensione int = %lu\n", sizeof(int));`
   - Es: `printf("dimensione double = %lu\n", sizeof(double));`

Codice sorgente: tipi_dim.c

Overflow

- Problema del trabocco (overflow) determinato dalla rappresentazione dei numeri con un numero limitato di cifre
  - Si verifica quando il risultato di un'operazione aritmetica non è rappresentabile nel dominio del tipo dato
  - Tipicamente il compilatore non lo segnala come errore, che può produrre effetti imprevedibili

Esempio: overflow

Codice sorgente: overflow.c

```
#include <stdio.h>

int main(void)
{
        short int si = 32767;
        long int li = 32767;
        printf("si prima dell'incremento: %d \n", si);
        printf("li prima dell'incremento: %ld \n", li);
        si = si+1;
        li = li+1;
        printf("si dopo l'incremento: %d \n", si);
        printf("li dopo l'incremento: %ld \n", li);
        return 0;
}
```

- Perché dopo l'incremento di si viene stampato -32768?
- Rappresentazione in complemento a 2!

Tipi di dato: booleani

- In C non esiste un tipo primitivo per rappresentare dati booleani (vero e falso)
- Come vengono rappresentati i risultati di espressioni relazionali?
  - Il C prevede che i valori logici restituiti da espressioni relazionali vengano rappresentati attraverso gli interi {0,1} secondo la convenzione:
    - ✓ 0 equivale a falso
    - ✓ 1 equivale a vero
  - Osservazione: qualsiasi valore diverso da 0 rappresenta vero

Operatori logici in C

- Il C utilizza per gli operatori logici la seguente notazione:
  - && AND logico
  - || OR logico
  - ! NOT logico
- Come per le condizioni elementari, le espressioni booleane restituiscono valore 1 per vero e 0 per falso
  - Es.: !2 è 0
  - Es.: (12 && 2) è 1
  - Es.: (0 && 2) è 0

Operatori logici in C: precedenza

- NOT ha precedenza su AND e AND ha precedenza su OR
- Il calcolo delle condizioni elementari ha precedenza sugli operatori booleani
- Esempio

```
a && !b || c;          equivale ad
(a && (!b)) || c;
```


Tipi di dato: char

- I char sono rappresentati in C da interi su 8 bit
  - Ogni valore di tipo char viene rappresentato dal suo codice (l'intero che lo rappresenta nella tabella ASCII)
  - Il dominio associato al tipo char è ordinato: l'ordine dipende dal codice associato ai vari caratteri nella tabella ASCII
- Sul tipo di dato char si possono eseguire tutte le operazioni previste per gli interi
  - Ogni operazione è applicata ai codici associati agli operandi
  - Es: `'a'>'b'` è falso perché codice(`'a'`) < codice(`'b'`)
  - Es: `!'A'` è vero perché codice(`'A'`) ≠ `0`

Esempio: char

- Quale valore viene stampato nel seguente codice?

Codice sorgente: caratteri.c

```
#include <stdio.h>

int main(void)
{
        char ch;
        ch = 'a';
        ch++;
        printf("%c\n", ch);
        return 0;
}
```


Operatori di assegnamento

- In C è possibile usare delle notazioni composte in quelle assegnazioni in cui la variabile a primo membro compare anche a secondo

```
sum = sum + val;
salario = salario* aumento;
```

- Si può abbreviare in:

```
sum += val;
salario *= aumento;
```

- Per ogni operatore esiste il corrispondente composto:

```
+       -       *       /       %
+=      -=      *=      /=      %=
```

Operatori di incremento e decremento

- E' frequente dover incrementare o decrementare una variabile di una unità
- Il C prevede una notazione semplificata (e più efficiente) per gli operatori di incremento e decremento unari
  - x++ e x--
- Forme equivalenti:

```
x = x+ 1;
x += 1;
x++;

x = x-1;
x -= 1;
x--;
```


Operatori di pre- e post-incremento

- Due forme di incremento in C
- Pre-incremento: l'operatore è prefisso alla variabile

```
++x;
```

- Post-incremento: l'operatore è postfisso alla variabile

```
x++;
```

- Entrambe le forme incrementano la variabile di 1
- La differenza è nell'ordine in cui avvengono le azioni di incremento e di calcolo dell'espressione
- Analogamente per il decremento
  - pre-decremento: --x;
  - post-decremento: x--;

Esempio: pre- e post-incremento

```
/* Pre-incrementing  and post-incrementing  */
#include  <stdio.h>

main()
{
   int  c;

   c = 5;
   printf("%d\n",   c);
   printf("%d\n",   c++);     /* post-increment   */
   printf("%d\n\n",   c);

   c = 5;
   printf("%d\n",   c);
   printf("%d\n",   ++c);     /* pre-increment   */
   printf("%d\n",   c);

   return  0;    /* successful  termination  */
}
```

Codice sorgente: pre-post_incr.c

Viene stampato 5

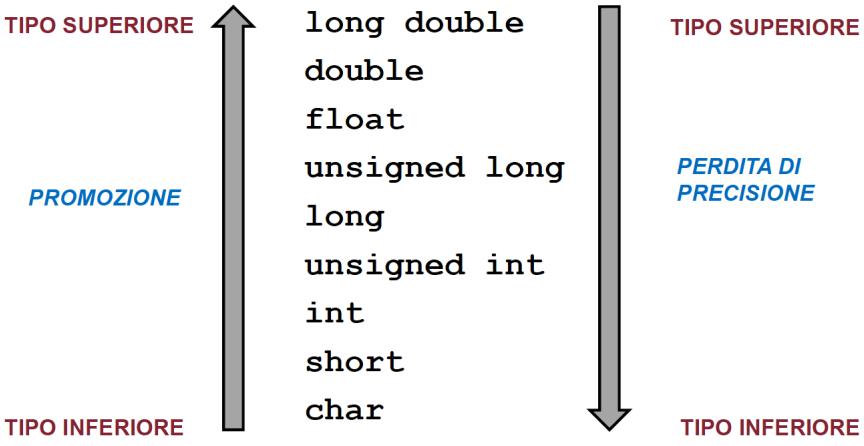
Viene stampato 6

Viene stampato 6

Conversioni di tipo

- In C è possibile combinare tra di loro operandi di tipo diverso
- Conversioni necessarie in vari casi:
  - Assegnazione tra due variabili di tipo diverso (ma compatibile)
    - ✓ Es: x = y; , con x di tipo float e y di tipo int
  - Operazione tra variabili di tipo diverso nella stessa espressione
    - ✓ Es: x = y * z; (x e y di tipo float e z di tipo long)
- In C due tipi di conversione
  - **Implicita** (promozione)
    - ✓ Il compilatore ne rileva la necessità e promuove la variabile al tipo superiore
    - ✓ Nel primo esempio y è promossa da int a float, nel secondo z da long a float
  - **Esplicita** (casting)
    - ✓ Effettuata dal programmatore

Conversione implicita: gerarchia di promozione



Conversione esplicita

- Per convertire esplicitamente dati da un tipo ad un altro si deve usare l'operatore di casting
- Sintassi

  (tipo) espressione
  - tipo: è un nome di tipo
  - espressione: è l'espressione della quale si desidera convertire il tipo
  - Es:
```
int c, d;
float x;
x =c/(float)d; /* divisione tra reali */
```
- Semantica
  - Converte il tipo di un'espressione ad un altro tipo, rendendo possibili operazioni che coinvolgono tipi altrimenti incompatibili

Codice sorgente: conversione.c