Corso di

Fondamenti di Programmazione

canale E-O

Tiziana Calamoneri

Le variabili in C

KP Cap. 3, pp. 95-129
Cap. 5, pp. 190-193
DD Cap. 9, pp.352-355

Tipi di dato

Abbiamo già parlato delle variabili, ed abbiamo fin ora usato il tipo `int`, ma abbiamo molte altre possibiltà…

I tipi di cui parleremo sono:

`char`
`int (short, long, unsigned)`
`float`
`double (long)`

Ricordiamo che queste parole chiave NON possono essere usate come nomi di variabili.

Tipo char

- Una variabile di tipo `char` occupa 1 byte in memoria.
- In C, i caratteri sono sempre delimitati da ' ' (ad es.'a')
  Esempio: `#define LETTERA 'a'`
- In realtà, in C, i caratteri sono interi, poiché si memorizzano tramite la loro codifica (ASCII o EBCDIC).
- N.B. Non c'è relazione tra il valore di un `char` ed il carattere corrispondente (il valore del carattere '2' NON è 2 ma 50!)
- Alcuni caratteri non stampabili richiedono una sequenza di escape (ad es. tab=`\t`). Tali sequenze occupano comunque un solo carattere.
- Le funzioni di I/O (ad es. `scanf()` e `printf()`) gestiscono i caratteri tramite `%c`

Un esempio

Problema: dato in input un carattere, se esso è una lettera minuscola, trasformarla in maiuscola.

```
#include<stdio.h>
int main(void)
{ char c;

 scanf("%c", &c);
 if (c>='a' && c<='z')
    printf("%c", c+'A'-'a');
 else putchar(c);
 return 0;
}
```

N.B. i caratteri sono ordinati secondo il loro codici, quindi possono essere trattati come degli interi (confrontati e addi-zionati).

'A'-'a'

…abc…xyzABC…XYZ…

Un altro esempio

Problema: dato in input un carattere, se esso rappresenta una cifra, trasformarla nel suo equivalente numerico.

```
#include<stdio.h>
int main(void)
{ char c; int n;

 scanf("%c", &c);
 if (c>='0' && c<='9')
    {
          n=c-'0';
          printf("%d", n);
    }
 else printf("il carattere inserito non è una cifra");
 return 0;
}
```

isdigit(c) in <ctype.h>

'6'-'0'

…'0' '1'…..'6'…'9' …

48 49 54 57

L'intestazione standard <ctype.h> definisce un gruppo di funzioni che forniscono test e conversioni legate al set dei caratteri. Ad es. tolower(c) restituisce la minuscola se c è maiuscola.

Tipo int (1)

- Una variabile di tipo int occupa di solito 2 o 4 bytes in memoria. Questo significa che non è possibile rappresentare tutti gli interi ma solo un numero finito di essi. Quindi: attenzione all'overflow!
- Se si ha bisogno di valori grandi si può usare long (4 byte). Il valore di LONG_MAX e LONG_MIN non è sempre uguale, ma varia da macchina a macchina ed è in limits.h
- Se la quantità di memoria usata è critica e non si usano interi grandi si può usare short (2 byte)
- Se i valori necessari sono solo non negativi, si può usare unsigned, che permette di memorizzare lo stesso numero di interi, ma tutti non negativi. Il valore di ULONG_MAX è in limits.h

Tipo int (2) DD p. 352

Le funzioni di I/O gestiscono gli interi tramite %d, %u, %o, %x, %h, %l. Vediamoli uno ad uno:

- %d intero decimale con segno; solo il - viene visualizzato, il + è sottinteso (es. -35, 24, 0, …)
- %u intero decimale senza segno; se un intero negativo viene letto con %u la lettura risulta errata! (es. -465 diventa 4294966841)
- %o %x interi senza segno ottali (cifre da 0 a 7; es. $256=2x8^2+5x8^1+6x8^0$) o esadecimali (cifre da 0 a 9 e lettere da a ad f; es.$1c7=1x16^2+12x16^1+7x16^0$)
- %h %l posti davanti ad uno degli indicatori precedenti indicano rispettivamente short e long; sono detti modificatori di lunghezza

Tipi reali (1)

- C'è una grossa differenza tra reali "matematici" e reali "digitali": i primi sono continui, gli altri discreti.
  Anche qui: attenzione all'overflow!
- Per memorizzare un intero come reale bisogna scriverlo con il . (ad es. 2.0)
- I reali possono scriversi con il punto decimale o tramite notazione esponenziale che, quando non c'è, è sottintesa (ad es. 1.342e3 vale 1342.3, mentre 3.14 è come se fosse 3.14e0)
- In C sono previsti 3 tipi reali: float, double e long double
- float numero in virgola mobile, precisione singola
- double numero in virgola mobile, precisione doppia
- long double numero in virgola mobile, precisione multipla
- ATTENZIONE: anche qui, il num. dei byte usati dipende dalla macchina, quindi le dimensioni distinte potrebbero essere 1, 2 o 3.

Tipi reali (2)

Le funzioni di I/O gestiscono i reali tramite %e, %f, %L.

- %e visualizza in forma esponenziale (es. 1,502876e+02=1,502876 x 10^2=150,2876)
- %f visualizza con la virgola (es. 345,872000); visualizza sempre almeno una cifra a sx della virgola
- N.B. i valori visualizzati tramite %e ed %f sono visualizzati di default con 6 cifre di precisione. %L posto davanti ad uno degli indicatori precedenti indica long double

Costanti

- Abbiamo detto che le costanti si definiscono tramite una direttiva del preprocessore.
- Esse non vengono definite con un tipo, che deve essere ricavato dal processore.
- La costante 1234 è un int
- Una costante long si scrive con una L finale: 123456789L
- Le costanti in virgola mobile si scrivono con il punto decimale (123.45) oppure in forma esponenziale (1e-2) ed il loro tipo è double
- I suffissi F ed L indicano i tipi float e long double
- Una costante carattere è un intero, scritto come carattere tra apici. Quindi le costanti carattere possono prendere parte alle operazioni numeriche anche se si usano di solito in confronto ad altri caratteri
- N.B. attenzione alla differenza tra 0 e '0'!!

Esercizi 12. (tipi interi e reali)

- scrivere un programma C che stampi i numeri +875 e -875 con tutti gli indicatori di conversione disponibili
- scrivere un programma C che stampi i numeri +1234567.89 e -1234567.89 con tutti gli indicatori di conversione disponibili
- scrivere un programma C che determini la gamma di valori possibili dei tipi short, int, long, con e senza segno, visualizzando valori appropriati dalle intestazioni standard.

sizeof

- Il C fornisce l'operatore unario sizeof per determinare il numero di byte usati per memorizzare un oggetto.
- Esempio. sizeof(oggetto)
  dove oggetto può essere un tipo (come int o float), una variabile (come x) o un'espressione (come a+b).
- N.B. i risulati di sizeof possono variare da una macchina all'altra, ma è sempre vero che:
  sizeof(char)=1
  sizeof(char)≤sizeof(short)≤sizeof(int)≤sizeof(long)
  sizeof(float)≤sizeof(double)≤sizeof(long double)

Esercizi 13. (tipi di var., sizeof)

- Stampare una variabile `char` come carattere e come intero
- Provare ad eseguire operazioni tra interi e memorizzare il risultato in un `int` ed in un `float`, e poi stampare entrambi
- Provare a memorizzare in un `int` un numero troppo grande
- Stampare il numero di byte usati da tutti i tipi descritti

Conversioni implicite (1) p. 115

- Ogni espressione aritmetica, come `x+y`, ha un valore ed un tipo.
- Se sia `x` che `y` sono `int`, `x+y` è un `int`, ma se `x` ed `y` sono `short`, `x+y` è comunque un `int`, e non `short`, perché nelle espressioni i valori `short` vengono convertiti in `int`.
- In generale, se un'espressione coinvolge tipi diversi, i tipi inferiori vengono promossi ai tipi superiori. Ad esempio, se `x` è `int` e `y` è `float`, `x+y` sarà `float`.
- Questa operazione si chiama conversione implicita o automatica.

Conversioni implicite (2)

- In genere, le uniche conversioni automatiche sono quelle che consentono ad un tipo di ampliare il proprio campo di variabilità senza perdita di informazione (ad es. da `int` a `float`)
- Espressioni prive di senso, come reali usati come indici, provocano un errore
- Espressioni che potrebbero causare perdita di informazione (come assegnare un numero in virgola mobile ad un intero) NON SONO ILLEGALI, quindi ATTENZIONE!!
- Gli assegnamenti implicano conversioni: il valore del membro destro è convertito nel tipo del membro sinistro prima di essere copiato.

Cast (1)

Oltre alle conversioni implicite, che avvengono durante gli assegnamenti o nelle espressioni miste, sono possibili anche conversioni esplicite o forzate (cast).

Esempio. Se `x` è intero, allora
`(double)x;` converte il suo valore in double, ma la variabile x NON viene modificata.

Quindi il cast si può usare nelle espressioni.

Esempi. se i è un intero:

```
f=(float)(i+1);   /* i+1 sarebbe un int */
d=(double)i/3;    /* i/3 sarebbe un float */
```

L'operatore unario cast ha la stessa priorità degli altri operatori unari (ad es. segno).

Cast (2)

Attenzione: il casting modifica il valore della variabile ma il contenuto non viene modificato!

Ci sono funzioni di libreria (come sqrt) che hanno dei parametri di un certo tipo (double), e non producono cose sensate se l'argomento che si passa è di un altro tipo.

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

int main()
{
int i=4; float a=4.0;

printf("radice di interi=%d\n", sqrt(i));
printf("radice di reali=%f\n", sqrt(a));
return 0;
}
```

Output:
radice di interi=1073741824
radice di reali=2.000000

Visibilità delle variabili(1)

REGOLA BASE: le variabili sono accessibili solo all'interno del blocco nel quale sono dichiarati: al di fuori sono sconosciute.

Esempio:

```
...
{
int a=2;
printf("%d",a);
{
   int a=5;
   printf("%d",a++);
}
printf("%d",a);
...
```

output:
2 5 2

N.B. La a più interna non sovrascrive l'altra!

Per evitare confusione, meglio non usare gli stessi identificatori:

```
...
{
int a=2;
printf("%d",a);
{
   int b=5;
   printf("%d",b++);
}
printf("%d",a);
...
```

Visibilità delle variabili (2)

Le variabili definite in un blocco sono dette locali, quelle definite fuori qualunque blocco, all'inizio del programma, sono dette globali.

Le variabili globali sono visibili dappertutto, le locali solo all'interno del blocco nel quale sono definite.

Esempio:

```
int i; int a;
int main()
{  scanf("%d", &a);
   if (a==1)
   {  i=1;
      while (i <=10)
         printf("%d", i++);
   }
   else printf ("%d", a);
   printf ("%d", ++i);
}
```

Esempio:

```
int a;
int main()
{  scanf("%d", &a);
   if (a==1)
   {  int i=1;
      while (i <=10)
         printf("%d", i++);
   }
   else printf ("%d", a);
   printf ("%d", ++i); ???
}
```

Visibilità delle variabili (3)

Due blocchi possono trovarsi uno dopo l'altro; in tal caso il secondo blocco non vede nulla di quello che accade nel primo, e viceversa. Le variabili esterne ai blocchi sono viste da entrambi.

Esempio:

```
...
{  int a; int b;
   ...
   {  int c; float a;
      ...
   }
   ...
   {  int d; float b;
      ...
   }
}
```

a e b sono visibili sia in 1 che in 2

c non è visibile in 2

a intera non è visibile perché è stata definita un'altra a

blocco 1

d non è visibile in 1

b intera non è visibile perché è stata definita un'altra b

blocco 2

Visibilità delle variabili (4)

- E' buona norma usare il più possibile variabili locali, posizionando la loro definizione in modo che la loro visibilità sia ristretta ai blocchi che le usano.
- Un programma con molte variabili globali non è un buon programma.
- Ad esempio, non c'è motivo che i contatori siano variabili globali.
- La globalità dovrebbe essere riservata solo a quelle variabili che ricorrono per tutto il programma.
- Le variabili globali dovrebbero essere poche.