Tipi di dato: conversioni e stringhe

Alessandra Giordani
agiordani@disi.unitn.it
Lunedì 22 aprile 2013
http://disi.unitn.it/~agiordani/

Ripasso: il linguaggio C è

- *esplicitamente tipato*: occorre esplicitamente associare ad ogni variabile il tipo di dato che essa rappresenta
- *staticamente tipato*: il tipo di una variabile non puo cambiare durante il suo ciclo di vita;
- *fortemente/debolmente tipato*: non è possibile mischiare tipo di dato diversi nella stessa espressione, o se è possibile è richiesto che le **conversioni** siano **esplicite** (<op> casting). Però, in alcuni casi questa conversione esplicita non è necessaria (**conversioni implicite**).

Tipi di dato fondamentali

- int e' il tipo di dato che consente di rappresentare numeri interi all'interno di un programma.
  - Su un architettura a 32 bit, un int è rappresentato in complemento a 2 utilizzando 4 byte;
- float permette di rappresentare numeri in virgola mobile
  - usando 4 byte con una precisione di circa 7 cifre decimali;
- double permette di rappresentare numeri decimali in virgola mobile in doppia precisione
  - usando 8 byte con una precisione di circa 15 cifre decimali;
- char permette di rappresentare caratteri alfabetici.
  - usando un byte. La sequenza di bit che rappresenta un carattere puo anche essere interpretata come un intero senza segno

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

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	4	da 0 a $2^{32}-1$
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

Operatore sizeof()

```
#include <stdio.h>
int main()
{
    printf("Size of (in bytes):\n");
    printf("int: %d\n", sizeof(int));
    printf("short int: %d\n", sizeof(short int));
    printf("long int: %d\n", sizeof(long int));
    printf("char: %d\n", sizeof(char));
    printf("float: %d\n", sizeof(float));
    printf("double: %d\n", sizeof(double));
    printf("long double: %d\n", sizeof(long double));
    return 0;
}
```

Le combinazioni di tipi

- 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

Conversioni implicite

- 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

Conversioni implicite di assegnamento

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

  j=2.6;

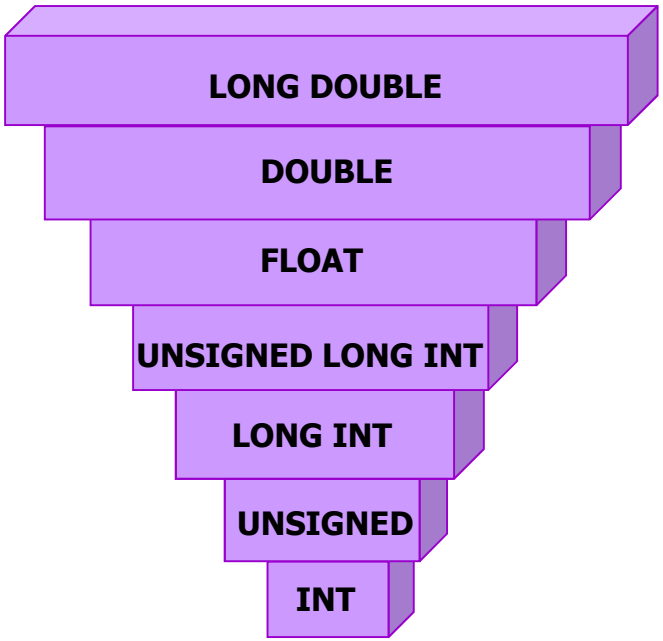
  Prima di assegnare la costante di tipo double, il compilatore la converte in int, per cui j assume il valore intero 2 (agisce per troncamento, non per arrotondamento)

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

  int ch='a'+32

Conversioni implicite - gerarchia

- 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;
}
```

La combinazione di floating-point

- 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

Il “tipo” stringa *char[]*

- Non esiste un tipo dedicato
- Si implementano con particolari array:
  - Ogni elemento del vettore è di tipo char
  - La stringa è terminata dal carattere speciale `‘\0’`
- Quindi è una sequenza di caratteri delimitata dal terminatore di stringa e inclusa tra doppi apici (").
- Ad esempio:
  - `“ciao” é ‘c’ ‘i’ ‘a’ ‘o’ ‘\0’`
  - `“ciao mondo"`
  - `“questa è una stringa”`

Sequenze di Escape

- Alcuni caratteri non sono rappresentabili con un singolo carattere (es. carattere di fine riga)
- Delle sequenze speciali (escape sequences) ci consentono di inserire questi caratteri.
- Ad esempio, il carattere di fina riga puo essere rappresentato con `'\n'` (non `"\n"`!)
- Quindi, la stringa "questa e una stringa\n" termina con un carattere di fine riga.
- In ogni caso una stringa termina sempre con il carattere di fine stringa `'\0'` (non `"\0n"`!)

Input di stringhe

- La sequenza %s all'interno di un template consuma qualunque sequenza di caratteri che non contenga spazi
- Per leggere una stringa, dobbiamo prima avere allocato un buffer (cioe aver dichiarato un array di caratteri) abbastanza capiente. Ad esempio:

```
char buf [256]; // crea un buffer
scanf("%s", buf);
```

- Se la stringa immessa dall'utente è piu lunga del buffer (nell'esempio la lunghezza massima è 255, tenendo conto del null character '\0') abbiamo un errore di **segmentation fault** (violazione di accesso in memoria)

printf() di stringhe

- printf può anche essere usata per stampare il contenuto di una variabile un array di caratteri o parte di esso
- All'interno della stringa template dobbiamo usare %s.
- Come altro parametro dobbiamo passare il nome della variabile, che usato da solo è il puntatore al primo elemento della stringa
- printf() stamperà quel elemento fino al carattere `'\0'`
  - `char str[] = "stringa";`
  - `printf("%s",str); //stampa` **`stringa`**
  - `printf("%s",str+2); // stampa` **`ringa`**
  - `printf("%s",str[2]); // non corretto!`
  - `printf("%c",str[2]); // stampa` **`r`**

scanf() di stringhe

- Allo stesso modo per leggere una stringa non carattere per carattere ma per intero
  - template “%s”
  - indirizzo del primo elemento della stringa che è proprio il nome della stringa! (no &)

Inversione di una stringa

- Leggo una parola *a* di lunghezza sconosciuta (dim max 20)
- Inserisco in *b* i caratteri di *a* in ordine inverso
- Stampo stringhe

```c
#include <stdio.h>
#define MAX 20
int main()
{
    char a[MAX], b[MAX];
    int i, dim;
    /*leggo parola*/
    printf("Inserire una parola (dim.max %d): ",MAX-1);
        scanf("%s", a);
    /*stampo parola - e calcolo lunghezza*/
    printf("Ho letto: '");
    for(i=0;a[i]!='\0';i++)
        printf("%c", a[i]);
    dim=i;
    printf("' lunga %d caratteri\n",dim);
    /*inverto caratteri*/
    for(i=0;i<dim;i++)
        b[i]=a[dim-1-i];
    b[dim]='\0';
    /*stampo parola invertita*/
    printf("Ecco la parola invertita '%s'\n",b);
    return 0;
}
```

Inversione di una frase?!

- Se proviamo ad immettere una stringa con spazi, quando si incontra uno spazio la stringa di input si considera finita…

scanf() di stringhe con spazi

- Nel template possiamo specificare…
  - quanti caratteri leggere:
    - “%20s” leggere massimo 20 caratteri (no spazi)
  - quali carattere accettare
    - “%[A-Za-z]” accettare solo caratteri alfabetici maiuscolo o minuscolo e lo spazio
    - “%[0-9-]” accettare solo caratteri numerici e il –
  - quali carattere non accettare
    - “%[^\n]” accetta tutti i caratteri tranne l’a-capo quindi legge una riga intera, compresi caratteri alfabetici maiuscolo o minuscolo, numeri, punteggiatura

Conversione da stringa ad intero

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

main()
{
     char anno_nascita[5], anno_corrente[5];
     int anni;

      printf("Inserire l'anno di nascita: ");
     scanf("%s", anno_nascita);
     printf("Inserire l'anno corrente: ");
     scanf("%s", anno_corrente);

     /* atoi() converte una stringa in un intero */
     anni = atoi(anno_corrente) - atoi(anno_nascita);
     printf("Eta': %d\n", anni);
     exit(0);
}
```

Stringa MAIUSCOLA

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

main()
{
          char s[100], t[100];
          int i;

          printf("Inserisci una stringa: ");
          scanf("%[A-Za-z ]", s);
          for (i=0; i<strlen(s); i++)
          {
               if ((s[i] >= 'a') && (s[i] <= 'z'))
                  t[i] = s[i] - 32;
               else
                  t[i] = s[i];
          }
          t[i]='\0';
          printf("Stringa maiuscola: %s\n", t);
          exit(0);
}
```

Esercizio per la prova pratica

- Scrivere un main che
  - Legga una frase e un carattere
  - Conti quante volte il carattere appare nella stringa
  - Non sia case sensitive (non tenga conto della differenza tra maiuscole e minuscole)
- Esempi
  - “pratica” ‘a’ → 2 occorrenze
  - “Prova pratica” ‘p’ → 2 occorrenze
  - “Prova pratica” ‘A’ → 3 occorrenze

soluzione

```c
#include<stdio.h>
#include<ctype.h>

int main()
{
    char str[30], ch;
    int i, k=0;

    printf("Inserire una stringa\n");
    scanf("%[A-Za-z ]", str);
    getchar();
    printf("Inserire un carattere\n");
    scanf("%c", &ch);
    for (i=0; str[i]!='\0'; i++) {
        if (toupper(str[i])==toupper(ch))
         k++;
        }
    printf("la lettera %c appare in \"%s\" %d volte\n",ch,str,k);
    return 0;
}
```

NOTA!

- La getchar() mi serve per eliminare il carattere \n dal buffer di lettura
- La scanf infatti legge finchè non trova un terminatore: può essere l'acapo, ma anche uno spazio, nel caso in cui non mettiamo lo spazio nel range dei caratteri accettati
- Può accadere che debba servire un ciclo di getchar() per pulire i caratteri non alfabetici presenti nel buffer…