I/O da tastiera + costrutti while e if

Alessandra Giordani
agiordani@disi.unitn.it
Lunedì 2 maggio 2011
http://disi.unitn.it/~agiordani/

Ripasso funzione printf()

- Usata per stampare il contenuto di una o più variabile
- All'interno della stringa **template** possiamo inserire delle sequenze di caratteri speciali a cui, in fase di stampa, l'interprete sostituirà il valore attuale di una variabile.
- Nel **template** dobbiamo dire alla funzione di quale variabile vogliamo stampare il valore:
  - per stampare una variabile int, la sequenza speciale è %d
  - per stampare una variabile float, la sequenza speciale è %f
  - per stampare una variabile char, la sequenza speciale è %c
  - per stampare una variabile stringa, la sequenza speciale è %s

  (stringa terminata da *null*)

La funzione scanf()

- La funzione scanf (definita in stdio.h) consente di leggere interattivamente l'input immesso dall'utente di un programma
- La sintassi sfrutta il passaggio di parametri per riferimento
- La funzione ha una sintassi simile a printf:

```
int scanf (const char* template , ...) ;
```

  - template è una stringa che specifica che cosa si vuole leggere
  - ... sta per un numero variabile di puntatori alle variabili in cui vogliamo salvare i valori letti
- La funzione ritorna il numero di assegnamenti (valori salvati in variabili) effettuati

scanf() - Esempio

- Chiedi all'utente di immettere un numero e calcolane la radice:

```
#include <stdio.h>
#include <math.h>
int main() {
    double number;
    printf("Inserisci un numero decimale: ");
    scanf("%lf", &number);
    printf("La radice quadrata di %f é %f\n",
       number, sqrt(number));
    return 0;
}
```

- All'interno del template:

%d Legge un intero (si aspetta un `int*`)

%f Legge un float (si aspetta un `float*`)

%lf Legge un double (si aspetta un `double*`)

%s Legge una stringa (si aspetta un `char*`, vedi slide successive)

- Uno spazio consuma qualunque sequenza di white-space characters (spazi, tabulazioni, a capo)

Il costrutto if

Sintassi

```
if (<condizione>) { /* blocco_if ... */ }
[ else { /* blocco_else ... */ } ]
```

- Esegui `blocco_if` solo se `<condizione> != 0`
- Se presente, esegui `blocco_else` solo se `<condizione> == 0`

Il costrutto if (cont)

- Blocchi `if` possono essere combinati per gestire piú di due casi, es:

```
if (x == 0) {
    //eseguito solo se x == 0
} else if (x > 5) {
    //eseguito solo se x > 5
} else if (x > 0) {
    //eseguito solo se x > 0 e x <= 5
} else {
    //eseguito in tutti i restanti casi (x < 0)
}
```

Il costrutto if (cont)

- L'ordine é importante!
- Confronta questo esempio con il precedente:

```
if (x == 0) {
    //eseguito solo se x == 0
} else if (x > 0) {
    //eseguito solo se x > 0
} else if (x > 5) {
    //eseguito solo se x > 5 e x <= 0, cioé MAI!!!
} else {
    //eseguito in tutti i restanti casi (x < 0)
}
```

scanf() - matching multiplo

- La funzione scanf puo' essere usata per leggere più di una variabile in un colpo solo:

```
#include <stdio.h>
int main()
{
    float peso, altezza;
    printf("Immetti il tuo peso in kg e la tua altezza in metri:\n");
    scanf("%f %f",&peso,&altezza);
    float imc=peso/(altezza*altezza);    //indice di massa corporea
    printf("Il tuo IMC e' %f. ",imc);
    if (imc<18.5)
       printf("Sei sottopeso, mangia!\n");
    else if(imc<25)
         printf("Alla grande!\n");
    else if (imc<30)
         printf("Sei un po' in carne, datti una regolata!\n");
    else printf("A dieta, subito!\n");
    return 0;
}
```

Differenza tra due interi

```c
#include<stdio.h>

int main()
{
    //Dichiarazione delle variabili
    int a, b, c;

    //Inserimento dati
    printf("Inserire primo intero: \n");
    scanf("%d", &a);
    printf("Inserire secondo intero: \n");
    scanf("%d", &b);

    //Determinazione del numero maggiore
    if (a>b){
        c=a-b;
        //Stampa risultato
        printf("Risultato di %d - %d: %d\n", a, b, c);
             }
    else{
        c=b-a;
        //Stampa risultato
        printf("Risultato di %d - %d: %d\n", b, a, c);
           }
    return 0;
}
```

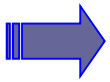
I tipi di dati scalari

- 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

Qualificatori short/long

- 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;
```

Qualificatori unsigned/signed

- Si possono individuare casi in cui una variabile può assumere solo valori positivi (ad es., i contatori)
- 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

unsigned (int) p;

- 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*

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;
}
```

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>

int main()
{
    char ch;

    printf("Digitare un carattere: ");
    scanf("%c", &ch);
    printf("Il codice numerico corrispondente e' %d\n", ch);
    return 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 = '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

```
#include<stdio.h>

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

int main()
{
    char ch;

    printf("Digitare un carattere MAIUSCOLO [A-Z]: ");
    scanf("%c", &ch);
    printf("Ecco lo stesso carattere minuscolo: %c\n", to_lower(ch));
    return 0;
}
```

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…

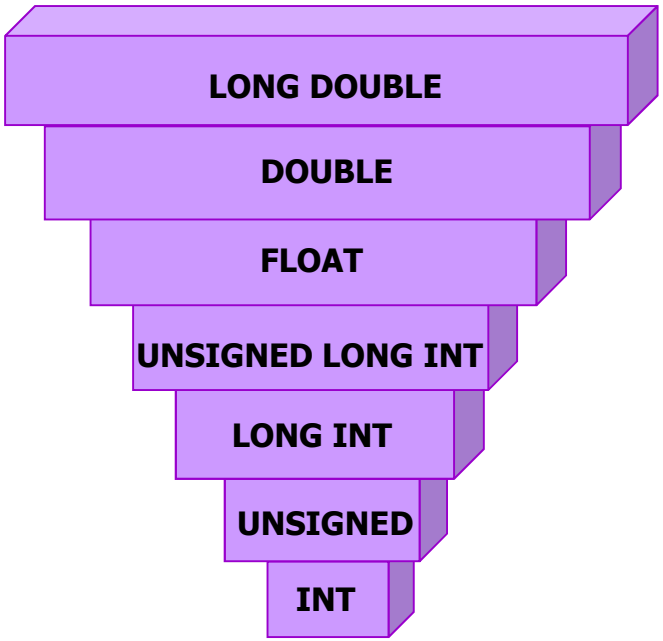
  j=2.6;

  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)

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;
}
```