Corso di

Fondamenti di Programmazione

canale E-O

Tiziana Calamoneri

Introduzione alla programmazione in C

DD Cap. 2, pp.23-48
Cap. 3 pp. 74-90
Cap. 4 pp. 112-116
Cap. 13 pp. 505-506

Abbiamo detto che:

Per scrivere un programma C corretto bisogna:

1. Analizzare il problema (input, output, casi estremali)
2. Progettare una soluzione (algoritmo) tramite pseudocodice o diagramma di flusso (a blocchi)
3. Accertarsi della correttezza del progetto
4. Perseguire l'efficienza — siamo arrivati qui…
5. Scrivere il codice C (spezzando in blocchi ed usando i principi della "buona programmazione") — ora andiamo qui!
6. Accertarsi della correttezza dell'esecuzione

Introduzione

- L'intento è quello di mostrare gli elementi essenziali del linguaggio per mezzo di programmi completi.
- Eviteremo di trattare dall'inizio troppi dettagli, regole ed eccezioni.
- Talvolta sacrifichiamo la completezza e precisione per cominciare a scrivere programmi semplici nel minor tempo possibile.

Un semplice programma C p.23

```
/* un primo programma C */
#include <stdio.h>
int main()
{
   printf("Primo programma in C\n");
   return 0;
}
```

commento

direttiva del preprocessore

funzione principale: ogni programma ne ha una

istruzione di stampa

terminatore

rientro

indica che il programma è terminato con successo

E ora concentriamoci sul comando `printf`...

Cenni su printf() (1)

- printf() è usata per l'output formattato
- non appartiene al C ma è presente nella libreria standard stdio.h
- la stringa da stampare è tra apici " "
- la stringa può contenere caratteri speciali:
  - \n invio (newline)
  - \t tabulazione
  - \" doppi apici
  - \\ backslash
  - eccetera
- la stringa può contenere anche caratteri di controllo per stampare le variabili (più avanti)

Cenni su printf() (2)

- Attenzione: per andare a capo è necessario usare il carattere speciale \n.
- Analizziamo l'output dei seguenti programmi:

```
#include <stdio.h>
int main()
{
   printf("Primo programma in C\n");
   return 0;
}
```

Primo programma in C
_

Errore di compilazione:
missing terminating " character

```
#include <stdio.h>
int main()
{
   printf("Primo programma in C
         ");
   return 0;
}
```

```
#include <stdio.h>
int main()
{
   printf("Primo");
        printf(" programma in C");
   return 0;
}
```

Primo programma in C_

I commenti

- Il compilatore ignora tutti i caratteri compresi tra /* e */
- I commenti possono essere inseriti ovunque siano ammessi spazi bianchi, tabulazioni e caratteri invio
- I commenti possono essere usati liberamente per scrivere un programma di più facile comprensione
- I commenti vanno usati! Sono utili a se stessi e agli altri.

Cenni sul preprocessore (1) p. 505

Abbiamo detto che #include <stdio.h> è una direttiva del preprocessore.

Il preprocessore interviene prima della compilazione e alcune delle sue possibili azioni sono:

- l'inclusione di altri files in quello da compilare (#include)
- la definizione di costanti simboliche (#define).

Cenni sul preprocessore (2)

#include nome_file

indica al preprocessore di includere nel programma il contenuto del file nome_file.

nome_file può essere racchiuso tra <> (cerca nella directory dei files di intestazione della libreria standard) o tra "" (cerca nella directory corrente, se il file è stato scritto dal programmatore)

Alcuni files di intestazione della libreria standard:

<stdio.h> input/output

<math.h> funzioni matematiche

<string.h> funzioni su stringhe

<time.h> manipolazione data e ora

Cenni sul preprocessore (3)

#define identificatore testo_da_sostituire

- Dal momento in cui compare questa direttiva tutte le occorrenze di identificatore (purché non tra virgolette o come parte di un altro identificqtore) vengono sostituite fisicamente con testo_da_sostituire.
- La direttiva #define permette di creare le costanti simboliche, cioè di attribuire un nome a una costante ed usarlo in tutto il programma.
- Le costanti simboliche evitano i "numeri magici": numeri che hanno un significato per il programmatore ma potrebbero non averne per il lettore: meglio scrivere:

```
#define CATETO1 17
#define CATETO2 14
float area;
area= CATETO1*CATETO2/2;
```

piuttosto che scrivere:

```
float area;
area= 17*14/2;
```

Cenni sul preprocessore (4)

#define identificatore testo_da_sostituire

- Qualora una costante debba essere modificata, si può cambiare solo testo_da_sostituire invece che tutte le sue occorrenze.

Esempio:

```
#define PI 3.14159
#define IPOTENUSA 20
#define CATETO 17
```

- Per distinguere le variabili dalle costanti, per convenzione, si usano caratteri minuscoli per i nomi delle variabili e caratteri maiuscoli per i nomi delle COSTANTI.
- Attenzione: le costanti NON sono variabili, quindi non si può fare ad es.: IPOTENUSA=7;
- N.B. non c'è ; alla fine di una riga che inizia con #define

Cenni sulla sintassi del C

- Alla fine di ogni istruzione va messo ;
- se l'istruzione è composta, cioè è costituita da più azioni (istruzioni), queste vanno racchiuse tra {}
- dopo la } non va il ;
- per aumentare la leggibilità tutti i programmi dovrebbero avere:
  - una indentazione chiara e coerente
  - dei commenti chiari ed esplicativi
- Entrambe le cose servono solo a chi legge, non al calcolatore, ma sono fondamentali!

Un altro semplice programma p.28

```
/* somma di due interi */
   int main()
   {
      int a=3;
      int b=4;
      int somma;

      somma=a+b;
      return 0;
   }
```

variabili intere

Ma che cos'è una variabile?

Variabili (1) p.29

In un programma C possiamo usare variabili per salvare dei valori che vogliamo usare.

```
int a=3;
int b=4;
```

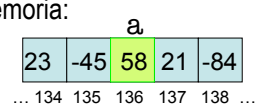
Una variabile è un oggetto che ha:
- un nome (a e b sono nomi di variabili)
- un tipo (int rappresenta il tipo intero)
- un valore (3 è il valore di a e 4 di b)
- un indirizzo (locazione di memoria assegnata)

Prima di usare una variabile, cioè prima di assegnarle un valore, dobbiamo dichiararla, vale a dire dobbiamo fare in modo che il programma sappia che quella variabile esiste e che le assegni una locazione di memoria. Per dichiararla, dobbiamo specificarne il nome ed il tipo.

Variabili (2)

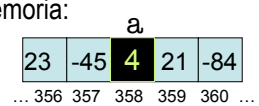
`int a;` permette di dichiarare una variabile intera di nome a

in memoria:



`int a=4;` permette di dichiarare una variabile intera di nome a il cui valore è 4

in memoria:



Variabili (3)

int (intero), float (reale), char (carattere), ecc. sono parole chiave, cioè parole riservate dal C che non possono essere utilizzate altrimenti (ad esempio per chiamare le variabili).

La dichiarazione di tipo serve:

- per informare il compilatore su quanto spazio in memoria debba essere riservato per i valori associati alle variabili
- per "leggere" correttamente la codifica binaria della variabile
- per permettere al compilatore di scegliere le istruzioni macchina appropriate (ad es. somma di interi e di reali diversa)

Variabili (4)

Il nome della variabile viene detto identificatore.

Esso consiste di lettere o numeri.

Esempio: a, var1, num_tel, ecc.

Attenzione:

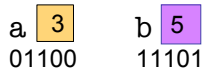
- un identificatore non può iniziare con un numero o con _
- in un identificatore le maiuscole vengono distinte dalle minuscole, quindi a e A sono due variabili diverse
- un identificatore non può essere una parola chiave

Buona regola: assegnare alle variabili nomi significiativi e non troppo lunghi (ad esempio, per una variabile che contiene la somma di altre, è meglio somma piuttosto che x, a, o pippo).

Variabili (5)

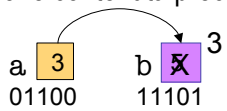
La lettura di una variabile NON comporta alcuna variazione in memoria.

Esempio: b=a;



per copiare a su b, devo prima leggere a…

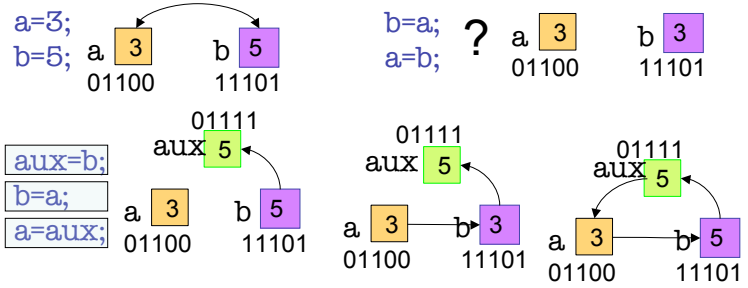
L'assegnamento di un valore ad una variabile implica la perdita dell'informazione contenuta precedentemente.



Variabili (6)

Esempio:

Supponiamo di avere 2 variabili e di doverne scambiare il contenuto:



L'aritmetica nel C (1) p.34

Spesso i programmi eseguono calcoli. Gli operatori aritmetici, tutti operatori binari, servono per eseguirli:

addizione + sottrazione -
moltiplicazione * divisione /
resto %

Attenzione a:

- La divisione tra interi restituisce un intero e tronca la parte frazionaria, mentre la divisione tra reali restituisce un reale; Esempio: 10/4=2 ma 10/4.0=2.5
- la divisione per 0 non è definita!
- l'operatore resto può essere applicato solo ad operandi interi.

L'aritmetica nel C (2)

- Si usano le parentesi per raggruppare i termini nelle espressioni (come nelle espr. algebriche, ma solo tonde!)
- Il C valuta le espressioni seguendo le seguenti regole di priorità (come nelle espr. algebriche):
  - Prima vengono moltiplicazione, divisione e resto, nell'ordine in cui si trovano
  - Poi vengono addizione e sottrazione, nell'ordine in cui si trovano

Esempio: (a+b+c+d+e)/5
1 2

Esempio: a*b+c-d/e
1 3 4 2

Operatori di uguaglianza e relazionali (1) p.37, p.115

- Operatori di uguaglianza:
  uguale ==
  diverso !=
- Operatori relazionali:
  maggiore >
  minore <
  maggiore o uguale >=
  minore o uguale <=
- Gli operatori aritmetici hanno TUTTI la precedenza sugli operatori relazionali

Attenzione a:

- non interporre uno spazio quando l'operatore è formato da due caratteri
- non confondere = e == `(if (voto==18) printf("pochino");)`

Operatori di uguaglianza e relazionali (2)

Confondere l'operatore di uguaglianza e quello di assegnamento

Caso 1: volevamo scrivere
`x==4` ma abbiamo scritto `x=4`;

Soluzione: scrivere la costante a sx e la variabile a destra: `4==x;` se dimentichiamo un =, l'istruzione `4=x;` dà un errore di compilazione!

Caso 2: volevamo scrivere
`x=4` ma abbiamo scritto `x==4`;

Non ci sono errori in compilazione, ma in esecuzione, perché la variabile non viene mai modificata.

Operatori logici (1) p.112

- `&&` (AND logico)
  Ritorna true se entrambe le condizioni sono vere
- `||` (OR logico)
  Ritorna true se almeno una delle due condizioni è vera
- `!` (NOT logico, negazione logica)
  Inverte la verità/falsità della condizione
  N.B. E' un operatore unario

Gli operatori logici sono utili nelle condizioni dei cicli

Operatori logici (2)

Esempio:

Espressione	Risultato
true && false	false
true || false	true
!false	true

Attenzione: se vi sono più condizioni in AND o in OR il C le valuta da sx a dx, interrompendo la valutazione se può già dedurre il risultato; quindi bisognerebbe anteporre le condizioni che variano di più e posporre quelle più dispendiose computazionalmente.

Operatori logici (3)

- Gli operatori aritmetici e relazionali hanno TUTTI la precedenza rispetto agli operatori logici.
- && ha la precedenza su ||

Tabella di verità

x	y	x AND y (x && y)	x OR y (x || y)	NOT x (!x)
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Esempi:
x=2; y=4; (x==2)&&(x==y/2) ha valore vero
x=2; (x==0)||(x==2) ha valore vero

Esempio

Riprendiamo un esempio già studiato:

> Problema: Si leggano due interi *a* e *b*, e si risolva l'equazione *ax+b=0*.

La soluzione che avevamo proposto era:

```
Leggi a; Leggi b;
Se (a=0) allora
    se (b≠0) allora
        stampa:"non esiste soluzione"
    altrimenti
        stampa:"infinite soluzioni"
altrimenti
    x=-b/a
    stampa x
```

Ora possiamo scrivere:

```
Leggi a; Leggi b;
se (a=0) AND (b≠0) allora
    stampa:"non esiste soluzione"
altrimenti
    se (a=0) AND (b=0) allora
        stampa:"infinite soluzioni"
    altrimenti /* cioè se a≠0
        x=-b/a
        stampa x
```

Esercizi 5. (oper. aritmetici)

- Siano a, b e c degli interi che valgono, rispettivamente, 8, 3 e -5. Determinare il valore delle seguenti espressioni:
  - a+b+c
  - 2*b+3*(a-c)
  - a/b
  - a%b
  - a/c
  - a%c
  - a*b/c
  - a*(b/c)
  - (a*c)%b
  - a*(c%b)
- Scrivere un programma C per verificare le risposte date.

N.B. Attenzione a cosa succede con il segno -
posit. / negat.=negat. e negat./posit.=negat.
posit.%negat.=posit. ma negat.%posit.=negat.
negat./negat.=posit. e negat.%negat.=negat.

Esercizi 6. (oper. logici)

- Scrivere un'espressione C che, data una variabile intera a, restituisca true solo se a contiene un numero positivo pari oppure un numero negativo dispari
- scrivere un'espressione C che, data una variabile reale b, restituisca true solo se b contiene un valore compreso in uno dei seguenti intervalli: [2,3], (5,6)
- scrivere un'espressione C che, date due variabili intere a e b, restituisca true solo se a e b contengono entrambe valori >2 o <0
- dati due punti (a,b) e (c,d) che rappresentano i vertici superiore sinistro ed inferiore destro di un rettangolo, scrivere un'espressione C che restituisca true solo se un terzo punto (x,y) si trova dentro il rettangolo
- dati 3 punti (a,b), (c,d) ed (e,f), scrivere un'espressione C che restituisca true solo se i tre punti giacciono sulla stessa retta.
- scrivere un'espressione C che, data una variabile anno, restituisca true se l'anno è bisestile, false altrimenti (sol. a pag. 39 del KR).

Operatori di assegnamento (1) p.74

Abbiamo usato comandi come:

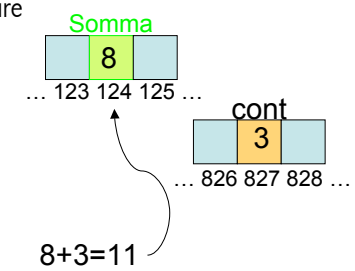
```
Somma=0
Somma=Somma+cont
Somma=Somma+1
```

che si sicrivono in C come:

```
Somma=0;
Somma=Somma+cont; oppure
Somma+=cont;
Somma=Somma+1; oppure
Somma+=1; oppure
Somma++;
```

cosa succede in memoria?

```
Somma=Somma+cont;
```



Operatori di assegnamento (2)

Ogni istruzione del tipo `variabile=espressione;` significa che alla variabile viene assegnato il valore dell'espressione.
Quando un valore viene assegnato ad una variabile, tale valore rimpiazza (e quindi distrugge) il precedente.
A sinistra dell'operatore di assegnamento c'è sempre il nome di una variabile, mentre a destra vi può essere una costante, una variabile o un'espressione.

Esempio.
```
x=3;
a=x;
b=(x+3)/a;
```

Operatori di assegnamento (3)

Sono ammesse anche assegnazioni multiple, cioè:

Esempio: `a=b=c=6;`

e queste vengono sempre eseguite da sinistra a destra.

Ogni istruzione del tipo

`var=var operatore espressione;`

può essere scritta come:

`var operatore=espressione`

dove `operatore` è un operatore binario (+, -, *, /)

Esempio: `c=c+3;` è equivalente a `c+=3;`

`x=x-y;` è equivalente a `x-=y;`

Operatori di assegnamento (4)

Se nell'istruzione

var=var operatore espres;

- operatore è + o - e
- espres è 1

si parla di incremento o decremento.

Esempio: c=c+1;

Postincremento: c++; (prima usa c poi la incrementa)

Preincremento: ++c; (prima incrementa c poi la usa)

Esempio:

```
c=3;          d=3 e c=4
d=c++;

c=3;          d=4 e c=4
d=++c;
```

Esempi

Sia x pari a 3.

Quanto vale y nell'espressione y=++x+x? 8

Le due espressioni y=x+(++x) e y=(x++)+x assegnano lo stesso valore ad y? no, 8 e 6

Quanto vale y nell'espressione y=x+++x? 6 perché le espressioni si valutano da sx verso dx

Buona regola: evitare queste ultime espressioni per ragioni di leggibilità

Un altro esempio

Osserviamo questo programma:

```
int main()
  {
    int a=1 b=1; int apiu, piub;

    apiu=a++;
    piub=++b;
  }
```

Che valore hanno le 4 variabili? a=2, apiu=1, b=2, piub=2

Esercizi 7. (assegnamento)

Valutare sulla carta i valori delle variabili in gioco alla fine dei seguenti frammenti di codice; poi scrivere il programma completo e verificare il risultato:

```
x=3; somma=2;
somma+=x++;

x=3; prodotto=2;
prodotto*=++x;

x=3; y=2;
++x += y++;

x=3; y=2;
x+=++x-y--;
```