Informatica

in linguaggio C

Stefano Lodi

16 novembre 2005

Algoritmi, linguaggi, programmi, processi

trasformazione di un insieme di dati iniziali in un insieme di risultati finali mediante istruzioni	algoritmo
strumento o formalismo per rappresentare le istruzioni di un algoritmo e la loro concatenazione	linguaggio di programmazione
algoritmo scritto in un linguaggio di programmazione al fine di comunicare al calcolatore elettronico le azioni da intraprendere	programma
programma in esecuzione su un calcolatore	processo

Struttura di un programma C

- Un programma C esegue *operazioni* in successione su un insieme di *variabili*
- Dichiarazioni di variabili e descrizione delle operazioni compaiono all'interno di funzioni
- Un programma C comprende una o piú funzioni, delle quali una è sempre la funzione principale (*main*)
- La struttura di un programma C è la seguente

```
main() {
  <dichiarazioni di dati>
  <istruzioni>
}
```

Programma sorgente

- Programma sorgente $\Longleftrightarrow$ sequenza di caratteri che soddisfa determinate regole
- Si distinguono regole lessicali e regole sintattiche
- Lessico $\Longleftrightarrow$ insieme di vocaboli utilizzabili per la costruzione del programma
- Il lessico fornisce i "mattoni da costruzione" da combinare secondo regole sintattiche per formare programmi formalmente corretti
- Solo i programmi formalmente corretti sono accettati dal compilatore

Lessico del C

parole riservate	significato speciale; non utilizzabili in modi diversi da quanto previsto, es. `main`, `while`, `for`
identificatori	nomi univoci attribuiti agli oggetti con cui il programma opera
costanti	valori numerici o sequenze di caratteri da utilizzare nelle operazioni
simboli speciali	caratteri non alfanumerici usati per comporre i vari oggetti del programma, es. ; [ ] ( ) ==
separatori	caratteri che determinano la fine di una parola riservata, di un identificatore o di una costante; comprendono i simboli speciali, lo spazio bianco, il fine linea

Tipi di dato

- I valori manipolabili da un programma si raggruppano in alcune categorie, con operazioni specifiche della categoria
  - interi, reali, sequenze di caratteri
- Un tipo di dato è definito da
  - dominio di valori D
  - funzioni $f_1, \ldots, f_n$ e predicati $p_1, \ldots, p_m$ *definiti sul dominio* (operatori)
  - costanti $c_1, \ldots, c_q$
- Se $v_1, \ldots, v_{k_i} \in D$ allora $f_i(v_1, \ldots, v_{k_i}) \in D$, dove k_i è il numero di argomenti di f_i (per $i = 1, \ldots, n$).
- Se $v_1, \ldots, v_{k_i} \in D$ allora $p_i(v_1, \ldots, v_{k_i})$ è vero oppure falso, dove k_i è il numero di argomenti di p_i (per $i = 1, \ldots, m$).

Tipi di dato

- rappresentazione di un tipo di dato in un linguaggio $\Longleftrightarrow$ descrizione con strutture linguistiche per *definirlo* e *manipolarlo*
- Esistono
  - Tipi semplici e strutturati
    - Nei tipi strutturati il dominio è un prodotto cartesiano di domini
  - Tipi predefiniti e definiti dal programmatore
- Qualche convenzione terminologica
  - Dati $\Longleftrightarrow$ sia valori che variabili
  - Operatori $\Longleftrightarrow$ simboli di predicato o funzione
  - Operandi $\Longleftrightarrow$ dati a cui sono applicati operatori

Il tipo `int`

- Numeri interi tra un minimo e un massimo
- Può essere qualificato come
  - `signed` oppure `unsigned`, rappresentando cosí interi con segno o senza segno, e
  - `short` oppure `long`, per rappresentare due differenti campi di variazione corrispondenti all'impiego di 16 bit o 32 bit per la rappresentazione
  - La dimensione di `int` senza qualificatori non è fissa e varia a seconda del compilatore (16 o 32 bit)

Il tipo `int`

Tipo	abbreviazione	campo di valori
signed short int	short int	da -32768 a 32768
signed int	int	dipende dal compilatore
signed long int	long int	da -2147483648 a 2147483648
unsigned short int		da 0 a 65535
unsigned int		dipende dal compilatore
unsigned long int		da 0 a 4294967295

Il tipo char

- L'insieme dei caratteri disponibili per la comunicazione
  - A rigore, dipende dal compilatore
  - In pratica, oggi la conformità allo standard ASCII è assai comune
- 256 caratteri (numerati da 0 a 255) di cui
  - da 0 a 31 non stampabili, codici di controllo
  - da 32 a 127 set di caratteri standard
  - da 128 a 255 nazionali
- Subsequenze notevoli
  - da 48 a 57, cifre, ordine crescente di significato
  - da 65 a 90, maiuscole, in ordine alfabetico
  - da 97 a 122, minuscole, in ordine alfabetico
- $\rightarrow$ proprietà utili per le conversioni

I tipi `float`, `double`

- Per le applicazioni numeriche è indispensabile un tipo decimale
- Si utilizzano i numeri in virgola mobile (*floating point*), con un numero finito prefissato di cifre $\rightarrow$ sottoinsieme dei razionali $\mathbb{Q}$
- `float`, singola precisione
- `double`, doppia precisione
- `long double`, quadrupla precisione

Tipo	numero di byte	campo di valori	cifre signif.
`float`	4 byte	$3.4 \times 10^{-38} \div 3.4 \times 10^{38}$	6
`double`	8 byte	$1.7 \times 10^{-308} \div 1.7 \times 10^{308}$	15
`long double`	16 byte	$1.1 \times 10^{-4932} \div 1.1 \times 10^{4932}$	19

Costanti

- Costante $\Longleftrightarrow$ dato che non può cambiare valore per tutta la durata del programma
- Costante intera: `123`
- Costante reale: `3.14`, `314E-2`, `0.314E1`
- Costante di tipo carattere: un singolo carattere scritto tra apici, es. `'a'`
  - Alla costante si associa il valore nell'insieme di caratteri considerato, (es. in ASCII `'a'` $\rightarrow 65$)
  - Alcuni caratteri non stampabili si rappresentano con un carattere preceduto da barra (*escape sequence)*; ad esempio, a capo si rappresenta con `'\n'`, tabulazione con `'\t'`, a inizio riga con `'\r'`
- Costante di tipo stringa: zero o piú caratteri tra doppi apici: `""`, `"ciao"`, `"continuare? (digitare \"si\"per continuare)"`

Dichiarazioni di costanti

- Associa un identificatore a una stringa di caratteri
- Sintassi

`#define` ⟨identif⟩ ⟨stringa-di-caratteri⟩

- Tutte le occorrenze di ⟨identif⟩ sono sostituite con ⟨stringa-di-caratteri⟩
- ```
  #define Pi 3.1415926
  #define StringaVuota ""
  ```
- `#define` non è una instruzione ma una direttiva, elaborata dal preprocessore
  ```

Variabili

- Variabile $\Longleftrightarrow$ dato che può essere usato e modificato dal programma
- La dichiarazione di variabile
  - associa un identificatore ad un tipo
  - determina l'allocazione di un'area di memoria adatta a contenere valori del tipo associato
- Nella dichiarazione è possibile specificare il valore che deve essere assunto dalla variabile all'inizio dell'esecuzione del programma (valore di inizializzazione)

Dichiarazione di variabile

- Sintassi semplificata

$$\langle \text{dich-variabile} \rangle \longrightarrow \langle \text{nome-tipo} \rangle \, \langle \text{lista-variabili} \rangle$$

$$\langle \text{lista-variabili} \rangle \longrightarrow \langle \text{variabile} \rangle \mid \langle \text{variabile} \rangle \texttt{ , } \langle \text{lista-variabili} \rangle$$

$$\langle \text{variabile} \rangle \longrightarrow \langle \text{identif} \rangle \mid \langle \text{identif} \rangle = \langle \text{espr} \rangle$$

- ```
  int X=0;  /* X è inizializzato a 0 */
  char C,K; /* equiv. a char C; char K; */
  ```
  ```

Operatori aritmetici

- Gli operatori aritmetici in C:

operatore	tipo	significato
-	unario	cambio segno
+	binario	addizione
-	binario	sottrazione
*	binario	moltiplicazione
/	binario	divisione tra interi o reali
%	binario	modulo (tra interi)

- / è un simbolo unico per la divisione reale e intera
  - se `x`, `y` sono entrambi interi, `x/y` è la divisione tra interi
  - altrimenti è la divisione reale
- Operatore modulo: `A%B` = `A - (A/B)*B`

Assegnamento

- Semplificando, l'assegnamento permette di calcolare il valore di una espressione e attribuire tale valore a una variabile
- L'assegnamento causa la scrittura del valore nell'area di memoria associata alla variabile, distruggendo il precedente valore
- Sintassi semplificata

$$\langle \text{assegnamento} \rangle \longrightarrow \langle \text{nome-var} \rangle = \langle \text{espr} \rangle$$

- ```
  int X,Y;
  X = 0;
  Y = X + 1;
  Y = Y + 1;
  ```
  ```

Assegnamento

- Piú generalmente, l'assegnamento permette di specificare
  - a destra dell'uguale un'espressione, detta rvalue, da valutare,
  - a sinistra dell'uguale una espressione, detta lvalue, utilizzata per indirizzare l'area di memoria in cui scrivere il valore risultato della valutazione della espressione sulla destra
- Sintassi

$$\langle \text{assegnamento} \rangle \longrightarrow \langle \text{lvalue} \rangle = \langle \text{rvalue} \rangle$$

- Una variabile può essere sia $\langle \text{lvalue} \rangle$ che $\langle \text{rvalue} \rangle$
- Una costante può essere solo $\langle \text{rvalue} \rangle$

Assegnamento

- L'assegnamento è una particolare espressione!
  - Il valore dell'assegnamento è il valore ottenuto dalla valutazione della parte destra
  - L'effetto dell'assegnamento è la scrittura del valore nell'area di memoria denotata dalla parte sinistra
- Pertanto è lecito scrivere

  ```
  4 - (X = 1)
  ```

  espressione che vale 3, in quanto l'assegnamento `X = 1` ha valore 1

Assegnamento

- Il linguaggio C fornisce come operatori comode abbreviazioni per alcuni assegnamenti notevoli
- `++` operatore di incremento, `--` operatore di decremento:
  - forma prefissa: `++X`, `--X`
  - forma postfissa: `X++`, `X--`
  - `++X`, `X++` hanno su `X` lo stesso effetto di `X = X + 1`
  - `--X`, `X--` hanno su `X` lo stesso effetto effetto di `X = X - 1`
  - Se compaiono in una espressione, vengono valutati diversamente
    - la forma prefissa modifica `X` prima dell'utilizzo del valore di `X` nel calcolo dell'espressione
    - la forma postfissa modifica `X` dopo l'utilizzo del valore di `X` nel calcolo dell'espressione

Assegnamento

- Esempi

```
X = 5;
Y = X++;     /* X vale 6 , Y vale 5 */
Y = ++X;     /* X vale 7, Y vale 7 */

Z = 5;
Y = 10+ ++Z /* Z vale 6, Y vale 16 */
```

- Operatori +=, -=, *=, /=, %=

```
Y += Z;      /* Y vale 21 */
```

Espressioni

- Espressione $\Longleftrightarrow$ regola per il calcolo di un valore
- Si compone di
  - Operandi
    - valori costanti
    - valori correnti di variabili
    - risultati di funzioni
    - ...
  - Operatori
- Un'espressione si valuta secondo le regole di precedenza degli operatori
- A parità di precedenza, la valutazione procede da sinistra a destra
- Le parentesi alterano la precedenza come consueto

Espressioni

- Precedenza e associatività per alcuni operatori

Operatori	Categoria	Associatività	Precedenza
`* / %`	prodotto e divisione	$\leftarrow$	alta
`+ -`	somma e sottrazione	$\leftarrow$	
`= +=` $\cdots$	assegnamento	$\rightarrow$	bassa

- Tutti gli operandi hanno un tipo e gli operatori richiedono specifici tipi e restituiscono tipi determinati $\rightarrow$ il compilatore è sempre in grado di stabilire il tipo di una espressione
- Si noti la associatività dell'assegnamento a destra: sono cosí possibili *assegnamenti multipli* quali

```
Y = X = 3;
```

il cui effetto è di porre sia `X` che `Y` a 3

Espressioni

- Sintassi delle espressioni (parziale)

$$\langle \text{espr} \rangle \longrightarrow \langle \text{costante} \rangle \mid \langle \text{nome-var} \rangle \mid \texttt{(}\ \langle \text{espr} \rangle\ \texttt{)} \mid$$
$$\langle \text{espr-assegn} \rangle \mid \langle \text{espr-incr-decr} \rangle$$
$$\langle \text{espr-assegn} \rangle \longrightarrow \langle \text{nome-var} \rangle\ \langle \text{oper-assegn} \rangle\ \langle \text{espr} \rangle$$
$$\langle \text{oper-assegn} \rangle \longrightarrow \texttt{=} \mid \texttt{+=} \mid \texttt{-=} \mid \texttt{*=} \mid \texttt{/=} \mid \texttt{\%=}$$
$$\langle \text{espr-incr-decr} \rangle \longrightarrow \texttt{++}\ \langle \text{nome-var} \rangle \mid \texttt{--}\ \langle \text{nome-var} \rangle \mid$$
$$\langle \text{nome-var} \rangle\ \texttt{++} \mid \langle \text{nome-var} \rangle\ \texttt{--}$$

Input e Output

- C non ha istruzioni predefininte per input e output
- Esiste tuttavia una Libreria Standard di funzioni, definite in `stdio.h`
- Un programma che svolge input e output deve includere la *direttiva*

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

- Funzioni per input/output di caratteri, linee, formattato
- Principali funzioni
  - `printf`
  - `scanf`

Output

- Semplice programma che visualizza dati in output

```
#include <stdio.h> /*include la libreria standard */
main () {
  int base, altezza, area;

  base = 3;
  altezza = 4;
  area = base*altezza;

  printf("L'area è: %d",area);
}
```

- la funzione `printf` ha argomenti (anche detti *parametri*) di due tipi
  - il primo è una costante stringa, la stringa di formato
  - i successivi, se presenti, sono espressioni il cui valore viene visualizzato

Output

- La stringa di formato contiene
  - Testo da visualizzare (`L'area è:  `)
  - numero e tipo dei dati da visualizzare (`%d` $\rightarrow$ decimale)
- Esempi
  - `printf("%d %d %d", base, altezza, area)`

    `3 4 12`
  - `printf("%4d%4d%6d", base, altezza, area)`

    `   3   4    12`
  - `printf("%-5d%-5d%-10d", base, altezza, area)`

    `3    4    12`
  - con sequenze di escape

    `printf("%d\n%d\n%d", base, altezza, area)`

    `3`

    `4`

    `12`

Output

- `printf`: *converte*, *formatta*, *stampa* i suoi argomenti sotto il controllo delle indicazioni della stringa di formato
- La stringa di formato contiene due tipi di caratteri
  - ordinari: copiati sull'output
  - specifiche di conversione: ognuna provoca la conversione e stampa del successivo argomento

carattere	tipo valore	stampato come
`d`	`int`	numero dec.
`c`	`int`	carattere singolo
`s`	`char *`	stampa tutta la stringa
`f`	`double`	$x \ldots x.dddddd$
`e`,`E`	`double`	$x \ldots x.ddddddezz$, $x \ldots x.ddddddEzz$

Output

- Tra il % e il carattere di conversione:
  - – $\mapsto$ allineamento a sinistra
  - numero $\mapsto$ ampiezza minima del campo di output
  - punto $\mapsto$ separatore tra ampiezza e precisione
  - numero $\mapsto$ precisione

Input

- Funzione `scanf`: *legge* caratteri da input, li *interpreta* secondo quanto indicato dalla stringa di formato, *memorizza il risultato* nelle variabili riportate come argomenti
- `scanf("%d", &base);`

  acquisisce un intero decimale e memorizza nella variabile `base`
- primo argomento: stringa di formato, contiene solo specifiche di conversione
- successivi argomenti: nomi di variabili a cui assegnare i valori acquisiti
- I nomi delle variabili devono essere preceduti dal simbolo `&`, estrae l'indirizzo della variabile consecutiva.

Input

- Stringa di formato
  - Spazi e tabulazioni sono ignorati
  - i caratteri normali (non %) devono corrispondere al successivo carattere non bianco in input
  - Le specifiche di conversione riflettono quelle della funzione `printf`
  - Per le variabili numeriche
    - gli spazi bianchi precedenti cifre sono ignorati
    - le cifre sono accettate fino al carattere di terminazione: `spazio` o `invio`