Lezione 5–Programmazione in Linguaggio C

Corso — Programmazione
Programmazione imperativa – Linguaggio C

Marco Anisetti, Stefano Ferrari

e-mail: marco.anisetti@unimi.it
web: http://homes.di.unimi.it/anisetti/

Variabile

- Ha una esistenza limitata
- Inizia ad esistere dalla sua dichiarazione e cessa alla chiusura del blocco in cui è stata dichiarata

- In realtà si tratta più correttamente di **visibilità** non di esistenza
    - Ovvero il programma non riconosce più come definito l'identificatore della variabile

- Si parla di **scope** di una variabile per indicarne il suo ambito di visibilità

- **L'assegnamento** è l'operazione che cambia il valore di una variabile

Valore di una espressione

- **Nel caso di una costante:** il suo valore
- **Nel caso di una variabile:** il suo valore corrente, cioè l'ultimo valore assegnatole

- **Nel caso di una funzione:** il valore restituito, cioè il risultato della funzione

```
res=max(10,15);
```

- **Nel caso di un'espressione composta:** il valore ottenuto eseguendo l'operazione indicata dall'operatore sui valori degli operandi

Operatori aritmetici (1)

- I tipi degli argomenti hanno un ruolo nel decidere la semantica dell'operatore
- Per esempio l'operatore modulo % è definito solo tra operandi di tipo intero
- L'operatore divisione intera / denota la divisione intera se ha operandi interi, la divisione reale se almeno uno degli operandi è un float o double

```
int a=1;
float b=2;

a/b // è la divisione tra reali
```

Operatori aritmetici (1)

- I tipi degli argomenti hanno un ruolo nel decidere la semantica dell'operatore
- Per esempio l'operatore modulo % è definito solo tra operandi di tipo intero
- L'operatore divisione intera / denota la divisione intera se ha operandi interi, la divisione reale se almeno uno degli operandi è un float o double

```
int a=1;
int b=2;
float res;

res=a/b; //cosa torna?
```

Operatori aritmetici (1)

- I tipi degli argomenti hanno un ruolo nel decidere la semantica dell'operatore
- Per esempio l'operatore modulo % è definito solo tra operandi di tipo intero
- L'operatore divisione intera / denota la divisione intera se ha operandi interi, la divisione reale se almeno uno degli operandi è un float o double

```
int a=1;
int b=2;
float res;

res=a/b; //cosa torna? 0! Non 0.5
```

Operatori aritmetici (2)

- Significa che pur avendo la stessa definizione grammaticale,
    - es. <expr> “/” <expr> devono essere specificate due semantiche diverse

<expr>.int “/” <expr>.int (usa la divisione intera)
<expr>.float “/” <expr>.float (usa la divisione reale)

Operatori aritmetici (2)

- Significa che pur avendo la stessa definizione grammaticale,
    - es. <expr> “/” <expr> devono essere specificate due semantiche diverse

<expr>.int “/” <expr>.int (usa la divisione intera)
<expr>.float “/” <expr>.float (usa la divisione reale)

Notazioni semantiche

Operatori aritmetici (2)

- Significa che pur avendo la stessa definizione grammaticale,
    - es. <expr> “/” <expr> devono essere specificate due semantiche diverse

<expr>.int “/” <expr>.int (usa la divisione intera)
<expr>.float “/” <expr>.float (usa la divisione reale)

Verrà tradotta nell’assembler che scatena la divisione intera

Operatori aritmetici (2)

- Significa che pur avendo la stessa definizione grammaticale,
    - es. <expr> “/” <expr> devono essere specificate due semantiche diverse

<expr>.int “/” <expr>.int (usa la divisione intera)
<expr>.float “/” <expr>.float (usa la divisione reale)

Verrà tradotta nell’assembler che scatena la divisione tra reali

Operatori aritmetici (2)

- Significa che pur avendo la stessa definizione grammaticale,
    - es. <expr> "/" <expr> devono essere specificate due semantiche diverse

<expr>.int "/" <expr>.int (usa la divisione intera)
<expr>.float "/" <expr>.float (usa la divisione reale)

Nota: la semantica per "/" tra int e float non è specificata quindi che succede? Non si trova una traduzione corretta nella grammatica?

Relazione tra tipi e promozioni

- I tipi di dato possono essere in relazione gerarchica tra di loro
- Per esempio in C sappiamo che esiste una gerarchia tra *char* che è contenuto in *int* e sappiamo che *int* è contenuto in *float* the è contenuto in *double*
    - *ci sono molti altri tipi che vedremo più avanti*
- È possibile effettuare promozioni di tipo che rispettino questa gerarchia
- In alcuni casi tali promozioni sono implicite e il programmatore non se ne accorge
- **Nell'esempio precedente int viene promosso implicitamente a float per poter scegliere la regola con la semantica corretta (divisione reale)**

Conversioni di tipo

- **Regole** di conversione **implicite**
- **Operatori espliciti** di conversione

- La conversione **implicita** avviene quando
  - un'**espressione composta** ha operandi di tipo diverso
  - un **assegnamento** ha tipo r-value e l-value diversi
  - una chiamata a **funzione** ha argomenti di tipo diverso dai parametri corrispondenti
  - un'istruzione **return** e seguita da un'espressione di tipo diverso dal tipo del risultato della funzione

- La regola di conversione è che si converte al tipo più ampio

Tipi di dato: approfondimenti(1)

- Variabili (e costanti) hanno associato un tipo

- Sappiamo che un tipo indica:
  - l'insieme dei valori che può assumere
  - il formato binario del valore (es IEEE 754)
  - lo spazio occupato in memoria dal valore
  - le operazioni che si possono compiere

- La dichiarazione dei tipi delle variabili serve per
  - riservare preventivamente in memoria lo spazio giusto alla variabile
  - dirimere la grammatica tramite informazioni addizionali

Tipi di dato: approfondimenti(2)

- **Tipi elementari:** informazioni composte da un solo valore
- **Tipi strutturati:** informazioni composte da più valori concettualmente legati
- Un linguaggio offre tipi predefiniti e tipi definiti dall'utente

Tipi predefiniti del C(1)

- Alcuni li abbiamo già introdotti allo scopo di permetterci di fare i primi esercizi

- **Numeri interi:** valori da -2^{n-1} a $2^{n-1}-1$
  - short, **int** e long
- **Numeri naturali:** valori tra 0 e $2^{n}-1$
  - unsigned short, unsigned int, unsigned long
- **Numeri reali:** in realtà sono numeri razionali
  - **float**, **double**, long double

Tipi predefiniti del C(2)

- **Caratteri:** dipende dagli standard ASCII Latin-1 ecc. Sono un insieme ordinato
    - **char**, unsigned char e signed char

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	¡	=	¿	?	@	A	B	C	D	E

Costanti

- Le abbiamo già viste, e sappiamo che il tipo è implicito nella forma della costante

Es 1.4 è double, ‘A’ è un carattere

- Visto che ci sono molti tipi che condividono la stessa forma es 1.4 può essere sia double che float, si può aggiungere una lettera per facilitare la comprensione del tipo

Es 1.4f è float, 1L è long mentre 1 da solo sarebbe int

Conversione implicite negli assegnamenti

<variabile> “=” *<espressione>*

- Si converte espressione nel tipo della variabile
- Se la variabile ha tipo più ampio, nessun problema
- Se la variabile ha tipo più limitato:
  - **Valore dell’espressione intero e variabile intera di tipo inferiore o carattere**: se il valore rispetta l'intervallo, **si converte**; altrimenti, diventa indefinito
  - **Valore dell’espressione reale e variabile reale di tipo inferiore**: se il valore rispetta l'intervallo, **si approssima**; altrimenti, diventa indefinito
  - **Valore dell’espressione reale e variabile intera**: si tronca il valore alla parte intera

Conversioni Esplicite

- Si parla di casting in caso di conversioni esplicite

“(” <tipo> “)” <espressione>

- Serve per superare le regole implicite e esprimere conversioni non banali

- **E’ un operatore unario ad alta priorità**

Esempi:

```
int a, b; double r;
r = a/b;
r = (double) a/b;
r = a/(double)b;
r = (double) a/(double) b;
```

Conversioni Esplicite

- Si parla di casting in caso di conversioni esplicite

"(" <tipo> ")" <espressione>

- Serve per superare le regole implicite e esprimere conversioni non banali

- **E' un operatore unario ad alta priorità**

Esempi:

int a, b; double r;
r = a/b; *prima calcola divisione intera poi converte in double*
r = (double) a/b; *converte prima (a e poi b) e calcola divisione esatta*
r = a/(double) b; *come sopra partendo da b*
r = (double) a/(double) b; *converte prima a, poi b …*

Esempi

Dati: long i; short j;

- i = j * j;
    1. calcola il prodotto short in aritmetica modulare
    2. lo converte in long e lo assegna ad i

- i = (long) j * j;
    1. converte il primo j in long (effetto del casting)
    2. converte il secondo j in long implicitamente
    3. calcola il prodotto long e lo assegna ad i

- i = (long) (j * j);
    1. calcola il prodotto short in aritmetica modulare
    2. lo converte in long (effetto del casting)
    3. lo assegna ad i

Operatori Aritmetici

- Con più operatori, servono regole di precedenza per definire l'ordine di esecuzione

Es. 5+4*3

- Ogni operatore ha una priorità e quindi si considerano prima gli operatori a priorità superiore e poi gli operatori a priorità inferiore

- Tuttavia la priorità non ordina completamente gli operatori

- Se un'espressione ha più operatori della stessa priorità, l'associatività determina se eseguire le operazioni associate

Gli operatori aritmetici precedono quelli relazionali i quali precedono quelli logici

Operatori logici

- AND (&&) ovvero al congiunzione
- a&&b vale 1 (vero) se la valutazione di a e di b da vero, vale 0 (falso) altrimenti

- OR (||) ovvero la disgiunzione
- a||b vale 1 se la valutazione di almeno uno tra a e b da vero, vale 0 (falso) altrimenti

- Espressioni con congiunzioni e disgiunzioni vengono valutate da sinistra a destra

- La valutazione avviene in ***short-circuit*** ovvero si interrompe appena posso dire se si tratta di una espressione vera o valsa
    - Es. (1 ||(a+b)) a+b non viene mai valutata
    - Es. (a>0)||(f(a)) può voler dire che f(a) a volte non viene eseguita

Operatori logici

- Altri esempi:

int b1=1; (ovvero vero)
int b2=0; (ovvero falso)

- ((b1||b2)<b1)+7 quanto vale?

Operatori logici

- Altri esempi:

int b1=1; (ovvero vero)
int b2=0; (ovvero falso)

- ((b1||b2)<b1)+7 quanto vale? 7
- ((b1&&b2)<b1)+7 quanto vale?

- Occhio ad alcune espressioni

int a=4;

(10<a<6) quanto vale?

Operatori logici

- Altri esempi:

int b1=1; (ovvero vero)
int b2=0; (ovvero falso)

- ((b1||b2)<b1)+7 quanto vale? 7
- ((b1&&b2)<b1)+7 quanto vale?

- Occhio ad alcune espressioni

int a=4;

(10<a<6) quanto vale? 1

Espressioni e macro

- Macro e una costante simbolica definita con una direttiva del preprocessore #define

- Per questo motivo viene risolta dal preprocessore tramite una meccanica sostituzione nel sorgente della definizione relativa

```
#define LARGHEZZA 20
```

Espressioni e macro

- Il valore di una macro può essere un'espressione composta

```
#define SOMMA 3+2

a=5*SOMMA;
a=5*3+2=17
```

```
#define SOMMA (3+2)

a=5*SOMMA;
a=5*(3+2)=25
```

Dichiarazioni di variabili

- Sappiamo che un programma C è costituito al minimo da una funzione, il main
- Per ora abbiamo sempre dichiarato variabili all'interno di blocchi all'interno della funzione main
- Le variabili dichiarate all'interno di una funzione in generale si chiamano **variabili locali** o **variabili automatiche** in terminologia gergale del C
- Una variabile va dichiarata prima di usarla ed è buona norma dichiarare subito tutto all'interno della funzione/blocco che userà quelle variabili
- Quando dichiaro una variabile dichiaro il tipo e il suo nome che si chiama anche **identificatore**

Identificatori

- Un identificatore è un elemento lessicale della grammatica del C che può essere definito a piacere dal programmatore ma deve rispettare delle regole
- Oltre che come nomi delle variabili un identificatore può essere usato come nome di funzioni
- Tra le regole grammaticali per definire un identificatore c'è che non può iniziare con una cifra e non può contenere spazi e nemmeno certi caratteri speciali
- In aggiunta sono case sensitive ovvero *Ciccia* e *ciccia* sono due identificatori diversi
- **Esercizio** pensare alla grammatica che descrive la dichiarazione di una variabile in C

Semantica assegnamenti

<variabile> “=” *<espressione>*

- Si valuta l’espressione e il suo risultato viene assegnato alla variabile
- Se quest’ultima contiene già un valore (e lo contiene sempre) questo viene sovrascritto

Esercizio:
Scrivere un programma C che scambia il valore di due variabili

Assegnamento: approfondimento

- L'assegnamento è un operatore e quindi ha un valore che è il valore dell'espressione a destra di =
- L'assegnamento è associativo destro

a=b=5; diventa a=(b=5)

- Oltre al valore ha anche un **effetto secondario (side effect)** ovvero modifica il valore dell'operando a sinistra
- Il **side effect** si materializza dopo la valutazione del valore di =

- Da non confondere con uguaglianza (==)

```
i=1;
b=(i==4)
```

```
b=(i=4)
```

Assegnamento: approfondimento

- L'assegnamento è un operatore e quindi ha un valore che è il valore dell'espressione a destra di =
- L'assegnamento è associativo destro

a=b=5; diventa a=(b=5)

- Oltre al valore ha anche un **effetto secondario (side effect)** ovvero modifica il valore dell'operando a sinistra
- Il **side effect** si materializza dopo la valutazione del valore di =

- Da non confondere con uguaglianza (==)

i=1;
b=(i==4) (varrà 0 ovvero falso)

b=(i=4) (varrà vero)

Assegnamento: approfondimento

- Cosa succede in questi casi?

```
i=1;
If (i=0) printf("vero")
else printf("falso")
printf("%d",i)
```

```
i=1;
b=1;
If ((b)||(i=0)) printf("vero ");
else printf("falso ");
printf("%d",i);
```

- Posso usare chiamate a funzione nella condizione

```
#include <ctype.h>
…
If (isdigit(c)) {printf("Hai digitato una cifra")}
```

- isdigit è una funzione che controlla se c è una cifra (ctype.h)

Assegnamento composto (i)

L'assegnamento si può comporre con gli altri operatori aritmetici

+= -= *= /=

Es

variabile += espressione

equivale a

variabile = variabile + (espressione)

a+=i*j equivale a a=a+(i*j)

Assegnamento composto (ii)

Occhio alla semantica però, perché = e += son diversi

i=i+2;

- valuta prima la parte destra (valuta i valuta 2 e poi li somma)
- valuta dopo la parte sinistra (valuta i ancora) e poi assegna il valore destro al sinistro

i+=2;

- valuta prima la parte destra (valuta 2)
- valuta dopo la parte sinistra (valuta i) e poi assegna la somma del destro + sinistro

- Cosa succede? i viene valutata 2 volte in un caso e una volta nell'altro
- Potremmo avere effetti collaterali dati da questa doppia valutazione (non in questo caso semplice)

Incremento e decremento

- L'operatore ++ (incremento) incrementa di 1 una variabile intera

- L'operatore -- (decremento) decrementa di 1 una variabile intera

- Si possono usare prefissi (++i e --i) o postfissi (i++ e i--)
    - se prefissi, eseguono l'operazione subito
    - se postfissi, la eseguono per ultima prima dell'istruzione seguente

a=++b; (incrementa b e assegna il valore ad a)

a=b++; (assegna b ad a e poi incrementa b)

Priorità

		Associatività
Priorità alta	! ++ -- - (unario)	dx -> sx
	* / %	sx -> dx
	+ - (binario)	
	> >= < <=	
	== !=	
	&&	
	||	
Priorità bassa	=	

Esempi(1)

- Supponiamo di avere dichiarato e inizializzato alcune variabili intere

```
int a, b, c, d, e, f;
a = 1; b = 2; c = 3; d = 4; e = 5;
```

Valutiamo alcune espressioni composte

```
d = a * b / c;
```

```
d = ++a * b - c;
```

```
e = 5 + c * d / e;
```

```
e = 30 / e++ + 29 % c;
```

Esempi(1)

- Supponiamo di avere dichiarato e inizializzato alcune variabili intere

```
int a, b, c, d, e, f;
a = 1; b = 2; c = 3; d = 4; e = 5;
```

Valutiamo alcune espressioni composte

```
d = a * b / c;
d = (1 * 2) / 3;

d = ++a * b - c;
d = 2 * 2 - 3;

e = 5 + c * d / e;
e = 5 + (3 * 4) / 5;

e = 30 / e++ + 29 % c;
```

Esempi(2)

- Supponiamo di avere dichiarato e inizializzato alcune variabili intere

int a, b, c, d, e, f;
a = b = c = d = e = 2; f = 1;

- Valutiamo l'espressione

a = b += c++ - d + --e / -f;
a = b += 2 - 2 + 1 / -1;
a = 1

Esempi(3)

- L'ordine di valutazione degli operandi può influire sul valore e sull'effetto collaterale dell'espressione

int a=0; int b=0;

(b = a + 2) - (a = 1)

(a = 1) - (b = a + 2)

- Valutazioni short-circuit

c=(a=1)||(b=2);

Istruzioni

- In C una espressione che termina con ; è una istruzione.

```
i+1;
```

- Viene valutato e il risultato non assegnato a nulla.
- **Motivo:** Per fare in modo che un blocco abbia una struttura regolare (esprimibile con una grammatica)
- È ragionevole che ogni istruzione abbia un side-effect, e uno solo (l'esempio sopra non ne ha)

Istruzioni

- Le direttive non sono istruzioni (non terminano con ;)
    - Trattate dal preprocessore
- Dichiarazioni sono istruzioni
- Assolvono a diversi compiti primo fra tutti dire al processore quanta memoria allocare, creare una tabella di simboli da usare per le traduzioni del compilatore ecc.