

IL LINGUAGGIO C

- Un elaboratore è un **manipolatore di simboli (segni)**
- L'architettura fisica di ogni elaboratore è **intrinsecamente capace** di trattare vari domini di dati, detti **tipi primitivi**
 - dominio dei **numeri naturali e interi**
 - dominio dei **numeri reali** (con qualche approssimazione)
 - dominio dei **caratteri**
 - dominio delle **stringhe di caratteri**

1

TIPI DI DATO

Il concetto di **tipo di dato** viene introdotto per raggiungere due obiettivi:

- esprimere in modo sintetico
 - la loro rappresentazione in memoria, e
 - un insieme di operazioni ammissibili
- permettere di *effettuare controlli statici* (al momento della compilazione) sulla *correttezza* del programma

2

TIPI DI DATO PRIMITIVI IN C

- **caratteri**

- **char** caratteri ASCII
- **unsigned char**

Dimensione di **int**
e **unsigned int**
non fissa. Dipende
dal compilatore

- **interi con segno**

- **short (int)** -32768 ... 32767 (16 bit)
- **int** ???????
- **long (int)** -2147483648 2147483647 (32 bit)

- **naturali (interi senza segno)**

- **unsigned short (int)** 0 ... 65535 (16 bit)
- **unsigned (int)** ???????
- **unsigned long (int)** 0 ... 4294967295 (32 bit)

3

TIPI DI DATO PRIMITIVI IN C

- **reali**

- **float** singola precisione (32 bit)
numeri rappresentabili da 10^{-38} a 10^{38} circa
- **double** doppia precisione (64 bit)
precisione 15 cifre decimali; numeri rappresentabili
da 10^{-308} a 10^{308} circa

- **boolean**

- *non esistono in C come tipo a sé stante*
- si usano gli interi:
 - **zero** indica **FALSO**
 - ogni altro valore indica **VERO**
- convenzione: suggerito utilizzare **uno** per **VERO**

4

COSTANTI DI TIPI PRIMITIVI

- **interi** (in varie basi di rappresentazione)

<i>base</i>	<i>2 byte</i>	<i>4 byte</i>
decimale	12	70000, 12L
ottale	014	0210560
esadecimale	0xFF	0x11170

- **reali**

– in doppia precisione

24.0 2.4E1 240.0E-1

– in singola precisione

24.0F 2.4E1F 240.0E-1F

5

COSTANTI DI TIPI PRIMITIVI

- **caratteri**

– singolo carattere racchiuso fra apici

'A' 'C' '6'

– caratteri speciali:

'\n' '\t' '\\'' '\\\'' '\\\"'

6

STRINGHE

- Una *stringa* è una *sequenza di caratteri* delimitata da virgolette

`"ciao"` `"Hello\n"`

- In C le stringhe sono semplici sequenze di caratteri di cui l'ultimo, ***sempre presente in modo implicito***, è `'\0'`

`"ciao" = {'c', 'i', 'a', 'o', '\0'}`

7

ESPRESSIONI

- Il C è un linguaggio basato su ***espressioni***
- Una ***espressione*** è una *notazione che denota un valore* mediante un processo di *valutazione*
- Una espressione può essere *semplice o composta* (tramite aggregazione di altre espressioni)

8

ESPRESSIONI SEMPLICI

Quali espressioni elementari?

- **costanti**

- 'A' 23.4 -3 "ciao"

- **simboli di variabile**

- x pippo pigreco

- **simboli di funzione**

- f(x)

- concat("alfa","beta")

- ...

9

OPERATORI ED ESPRESSIONI COMPOSTE

- Ogni linguaggio introduce un **insieme di operatori**
- che permettono di **aggregare altre espressioni (operandi)**
- per formare **espressioni composte**
- con riferimento a diversi **domini / tipi di dato** (numeri, testi, ecc.)

Esempi

2 + f(x)

4 * 8 - 3 % 2 + arcsin(0.5)

strlen(strcat(Buf,"alfa"))

a && (b || c)

...

10

CLASSIFICAZIONE DEGLI OPERATORI

Due criteri di classificazione:

- in base al *tipo* degli operandi
- in base al *numero* degli operandi

in base al <i>tipo degli operandi</i>	in base al <i>numero di operandi</i>
• aritmetici • relazionali • logici • condizionali • ...	• unari • binari • ternari • ...

11

OPERATORI ARITMETICI

<i>operazione</i>	<i>operatore</i>	<i>C</i>
inversione di segno	<i>unario</i>	-
somma	<i>binario</i>	+
differenza	<i>binario</i>	-
moltiplicazione	<i>binario</i>	*
divisione fra interi	<i>binario</i>	/
divisione fra reali	<i>binario</i>	/
modulo (fra interi)	<i>binario</i>	%

NB: la divisione a/b è fra interi se sia a sia b sono interi,
è fra reali in tutti gli altri casi

12

OPERATORI: OVERLOADING

In C (come in Pascal, Fortran e molti altri linguaggi) **operazioni primitive associate a tipi diversi possono essere denotate con lo stesso simbolo**. Ad esempio, le operazioni aritmetiche su reali o interi

In realtà **l'operazione è diversa e può produrre risultati diversi**

```
int X,Y;  
se X = 10 e Y = 4;  
X/Y vale 2
```

```
int X; float Y;  
se X = 10 e Y = 4.0;  
X/Y vale 2.5
```

```
float X,Y;  
se X = 10.0 e Y = 4.0;  
X/Y vale 2.5
```

13

CONVERSIONI DI TIPO

- In C è possibile combinare tra di loro operandi di tipo diverso:
 - espressioni **omogenee**: tutti gli operandi sono dello stesso tipo
 - espressioni **eterogenee**: gli operandi sono di tipi diversi
- **Regola adottata in C:**
 - sono eseguibili le espressioni eterogenee in cui tutti i tipi referenziati risultano **compatibili** (cioè, dopo l'applicazione della regola automatica di conversione implicita di tipo del C risultano omogenei)

14

COMPATIBILITÀ DI TIPO

- Consiste nella possibilità di usare, *entro certi limiti*, oggetti di un tipo al posto di oggetti di un altro tipo
- Un tipo T1 è compatibile con un tipo T2
se
il dominio D1 di T1 è contenuto
nel dominio D2 di T2
 - `int` è compatibile con `float` perché $\mathbb{Z} \subset \mathbb{R}$
 - ma `float` non è compatibile con `int`

15

COMPATIBILITÀ DI TIPO - NOTA

- `3 / 4.2`
è una divisione *fra reali*, in cui il primo operando è convertito automaticamente da `int` a `double`
- `3 % 4.2`
è una operazione *non ammissibile*, perché 4.2 non può essere convertito in `int`

16

CONVERSIONI DI TIPO

Data una espressione $x \text{ op } y$

- 1. Ogni variabile di tipo **char** o **short** viene convertita nel tipo **int**;
- 2. Se dopo l'esecuzione del passo 1 l'espressione è ancora eterogenea, rispetto alla seguente gerarchia

int < long < float < double < long double

si converte temporaneamente l'operando di tipo *inferiore* al tipo *superiore* (***promotion***)

- 3. A questo punto l'espressione è **omogenea** e viene eseguita l'operazione specificata. Il risultato è di tipo uguale a quello prodotto dall'operatore effettivamente eseguito (in caso di overloading, quello più alto gerarchicamente)

17

CONVERSIONI DI TIPO

```
int x;  
char y;  
double r;  
(x+y) / r
```



La valutazione dell'espressione procede da sinistra verso destra

- **Passo 1** **(x+y)**
 - **y** viene convertito nell'intero corrispondente
 - viene applicata la somma tra interi
 - **risultato intero tmp**
- **Passo 2**
 - **tmp / r** **tmp** viene convertito nel double corrispondente
 - viene applicata la divisione tra reali
 - **risultato reale**

18

COMPATIBILITÀ DI TIPO

In un **assegnamento**, *l'identificatore di variabile e l'espressione* devono essere dello **stesso tipo**

- Nel caso di tipi diversi, se possibile si effettua la conversione implicita, altrimenti l'assegnamento può generare perdita di informazione

```
int x;
```

```
char y;
```

```
double r;
```

```
x = y;    /* char -> int*/
```

```
x = y+x;
```

```
r = y;    /* char -> int -> double*/
```

```
x = r; /* troncamento*/
```

19

COMPATIBILITÀ IN ASSEGNAIMENTO

- In generale, sono automatiche le conversioni di tipo che non provocano perdita d'informazione
- Espressioni che *possono* provocare perdita di informazioni non sono però illegali

Esempio

```
int i=5; float f=2.71F;; double d=3.1415;
```

```
f = f+i;  /* int convertito in float */
```

```
i = d/f;  /* double convertito in int !*/
```

```
f = d;    /* arrotondamento o troncamento */
```

Possibile Warning: *conversion may lose significant digits*

20

CAST

In qualunque espressione è possibile **forzare una particolare conversione** utilizzando l'*operatore di cast*

(<tipo>) <espressione>

Esempi

```
int i=5; long double x=7.77; double y=7.1;
i = (int) sqrt(384);
x = (long double) y*y;    //non necessario
i = (int) x % (int)y;
```

21

ESEMPIO

```
main()
{
    /* parte dichiarazioni variabili */
    int X,Y;
    unsigned int Z;
    float SUM;
    /* segue parte istruzioni */
    X=27;
    Y=343;
    Z = X + Y -300;
    X = Z / 10 + 23;
    Y = (X + Z) / 10 * 10;
    /* qui X=30, Y=100, Z=70 */
    X = X + 70;
    Y = Y % 10;
    Z = Z + X -70;
    SUM = Z * 10;
    /* qui X=100, Y=0, Z=100 , SUM =1000.0*/
}
```

22

OPERATORI RELAZIONALI

Sono tutti operatori *binari*:

<i>relazione</i>	<i>C</i>
uguaglianza	<code>==</code>
diversità	<code>!=</code>
maggiore di	<code>></code>
minore di	<code><</code>
maggiore o uguale a	<code>>=</code>
minore o uguale a	<code><=</code>

23

OPERATORI RELAZIONALI

Attenzione:

non esistendo il tipo *boolean*, in C le espressioni relazionali ***denotano un valore intero***

- **0** denota *falso*
(condizione non verificata)
- **1** denota *vero*
(condizione verificata)

24

OPERATORI LOGICI

<i>connettivo logico</i>	<i>operatore</i>	<i>C</i>
not (negazione)	<i>unario</i>	!
and	<i>binario</i>	&&
or	<i>binario</i>	

- Anche **le espressioni logiche denotano un valore intero**
- da interpretare come vero (1) o falso (0)

25

OPERATORI LOGICI

- Anche qui sono possibili espressioni miste, utili in casi specifici
5 && 7 0 || 33 !5
- Valutazione in **corto-circuito**
 - la valutazione dell'espressione cessa *appena si è in grado di determinare il risultato*
 - il secondo operando è valutato *solo se necessario*

26

VALUTAZIONE IN CORTO CIRCUITO

– **22 || x**

già vera in partenza perché 22 è vero

– **0 && x**

già falsa in partenza perché 0 è falso

– **a && b && c**

se **a&&b** è falso, il secondo **&&** non viene neanche valutato

– **a || b || c**

se **a||b** è vero, il secondo **||** non viene neanche valutato

27

ESPRESSIONI CONDIZIONALI

Una espressione condizionale è introdotta dall'operatore ternario

condiz ? espr1 : espr2

L'espressione denota:

- o il valore denotato da *espr1*
- o quello denotato da *espr2*
- in base al valore della espressione *condiz*

- se ***condiz*** è vera, l'espressione nel suo complesso denota il valore denotato da *espr1*
- se ***condiz*** è falsa, l'espressione nel suo complesso denota il valore denotato da *espr2*

28

ESPRESSIONI CONDIZIONALI: ESEMPI

– **3 ? 10 : 20**

denota sempre 10 (3 è sempre vera)

– **x ? 10 : 20**

denota 10 se x è vera (diversa da 0),
oppure 20 se x è falsa (uguale a 0)

– **(x>y) ? x : y**

denota il maggiore fra x e y

29

ESPRESSIONI CONCATENATE

Una espressione concatenata è introdotta
dall'**operatore di concatenazione** (la virgola)

espr1, espr2, ..., esprN

- tutte le espressioni vengono valutate (da sinistra a destra)
- l'espressione esprime il valore denotato da ***esprN***
- Supponiamo che
 - i valga 5
 - k valga 7
- Allora l'espressione: ***i + 1, k - 4***
denota il valore denotato da ***k-4***, cioè 3

30

OPERATORI INFISSI, PREFISSI E POSTFISSI

- Le espressioni composte sono **strutture** formate da **operatori** applicati a uno o più **operandi**
- Ma... *dove* posizionare l'operatore rispetto ai suoi operandi?

31

OPERATORI INFISSI, PREFISSI E POSTFISSI

Tre possibili scelte:

- **prima** → notazione *prefissa*
Esempio: **+ 3 4**
- **dopo** → notazione *postfissa*
Esempio: **3 4 +**
- **in mezzo** → notazione *infissa*
Esempio: **3 + 4**



È quella a cui siamo abituati,
perciò è adottata *anche in C*

32

OPERATORI INFISSI, PREFISSI E POSTFISSI

- Le notazioni **prefissa** e **postfissa** non hanno problemi di **priorità** e **associatività** degli operatori
 - non c'è mai dubbio su **quale** operatore vada applicato a **quali** operandi
- La notazione **infissa** richiede **regole di priorità** e **associatività**
 - per identificare univocamente **quale** operatore sia applicato a **quali** operandi

33

OPERATORI INFISSI, PREFISSI E POSTFISSI

- Notazione prefissa:
*** + 4 5 6**
 - si legge come $(4 + 5) * 6$
 - denota quindi 54
- Notazione postfissa:
4 5 6 + *
 - si legge come $4 * (5 + 6)$
 - denota quindi 44

34

PRIORITÀ DEGLI OPERATORI

- **PRIORITÀ:** specifica l'ordine di valutazione degli operatori quando in una espressione compaiono *operatori (infissi) diversi*

Esempio: $3 + 10 * 20$

- si legge come $3 + (10 * 20)$ perché l'operatore $*$ è più prioritario di $+$

- NB: operatori diversi possono comunque avere *uguale priorità*

35

ASSOCIATIVITÀ DEGLI OPERATORI

- **ASSOCIATIVITÀ:** specifica *l'ordine di valutazione* degli operatori quando in una espressione compaiono *operatori (infissi) di uguale priorità*
- Un operatore può quindi essere *associativo a sinistra* o *associativo a destra*

Esempio: $3 - 10 + 8$

- si legge come $(3 - 10) + 8$ perché gli operatori $-$ e $+$ sono equiprioritari e **associativi a sinistra**

36

PRIORITÀ E ASSOCIATIVITÀ

Priorità e associatività predefinite possono essere *alterate mediante l'uso di parentesi*

Esempio: **(3 + 10) * 20**

– denota 260 (anziché 203)

Esempio: **30 - (10 + 8)**

– denota 12 (anziché 28)

37

INCREMENTO E DECREMENTO

Gli operatori di incremento e decremento sono *usabili in due modi*

- come **pre-operatori**: **++v**
prima incremento e poi uso nell'espressione
- come **post-operatori**: **v++**
prima uso nell'espressione poi incremento

Formule equivalenti:

v = v + 1;

v +=1;

++v;

v++;

38

CHE COSA STAMPA?

main()	Soluzione:
{ int c;	
c=5;	
printf("%d\n", c);	5
printf("%d\n", c++);	5
printf("%d\n\n", c);	6
c=5;	
printf("%d\n", c);	5
printf("%d\n", ++c);	6
printf("%d\n", c); }	6

39

ESEMPI

- int i, k = 5;
i = ++k /* i vale 6, k vale 6 */
- int i, k = 5;
i = k++ /* i vale 5, k vale 6 */
- int i=4, j, k = 5;
j = i + k++; /* j vale 9, k vale 6 */
- int j, k = 5;
j = ++k - k++; /* DA NON USARE */
/* j vale 0, k vale 7 */

40

RIASSUNTO OPERATORI DEL C (1)

Priorità	Operatore	Simbolo	Associatività
1 (max)	chiamate a funzione selezioni	() [] -> .	a sinistra
2	operatori unari: op. negazione op. aritmetici unari op. incr. / decr. op. indir. e deref. op. sizeof	! + - ++ -- & * sizeof	a destra
3	op. moltiplicativi	* / %	a sinistra
4	op. additivi	+ -	a sinistra

41

RIASSUNTO OPERATORI DEL C (2)

Priorità	Operatore	Simbolo	Associatività
5	op. di shift	>> <<	a sinistra
6	op. relazionali	< <= > >=	a sinistra
7	op. uguaglianza	== !=	a sinistra
8	op. di AND bit a bit	&	a sinistra
9	op. di XOR bit a bit	^	a sinistra
10	op. di OR bit a bit		a sinistra
11	op. di AND logico	&&	a sinistra
12	op. di OR logico		a sinistra
13	op. condizionale	? . . . :	a destra
14	op. assegnamento e sue varianti	= += -= *= /= %= &= ^= = <<= >>=	a destra
15 (min)	op. concatenazione	,	a sinistra

42