Linguaggi di programmazione

C

LINGUAGGI DI PROGRAMMAZIONE

Il “potere espressivo” di un linguaggio è

caratterizzato da:

- **quali tipi di dati** consente di rappresentare (direttamente o tramite definizione dell’utente)
- **quali istruzioni di controllo** mette a disposizione (quali operazioni e in quale ordine di esecuzione)

PROGRAMMA = DATI + CONTROLLO

Linguaggio C

- **UN PO' DI STORIA**
  - definito nel 1972 (AT&T Bell Labs) per sostituire l'Assembler
  - prima definizione precisa: Kernigham & Ritchie (1978)
  - prima definizione ufficiale: ANSI (1983)

Linguaggio C

- Caratteristiche:
  - linguaggio sequenziale, imperativo, strutturato a blocchi
  - usabile anche come linguaggio di sistema
  - adatto a software di base, sistemi operativi, compilatori, ecc.
  - portabile, efficiente, sintetico
  - ma a volte poco leggibile...

Linguaggio C

- Caratteristiche:
  - portabilita’
  - non e’ un linguaggio fortemente tipizzato
  - consente la gestione di bit, byte e indirizzi
  - utilizza solo 32 parole riservate (27 K&R, 5 Ansi)
  - linguaggio strutturato
  - linguaggio per programmatori

Linguaggio C

- **Basato su pochi concetti elementari**
  - dati (tipi primitivi, tipi di dato)
  - espressioni
  - dichiarazioni / definizioni
  - funzioni
  - istruzioni / blocchi

Linguaggio C

- 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.

Linguaggio C (Tipi di dato primitivi)

- **caratteri**
  - **char**
  - **unsigned char**
- **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)

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

Linguaggio C (Tipi di dato primitivi)

- **Reali**

  float singola precisione (32 bit)

  double doppia precisione (64 bit)

- **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**

Costanti

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

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**

- **caratteri**

  singolo carattere racchiuso fra apici

  ’A’ ’C’ ’6’

  caratteri speciali:

  ’\n’ ’\t’ ’\’’ ’\\’ ’\"’

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'}

Parole riservate

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

caratteri

maiscolo ≠ minuscolo

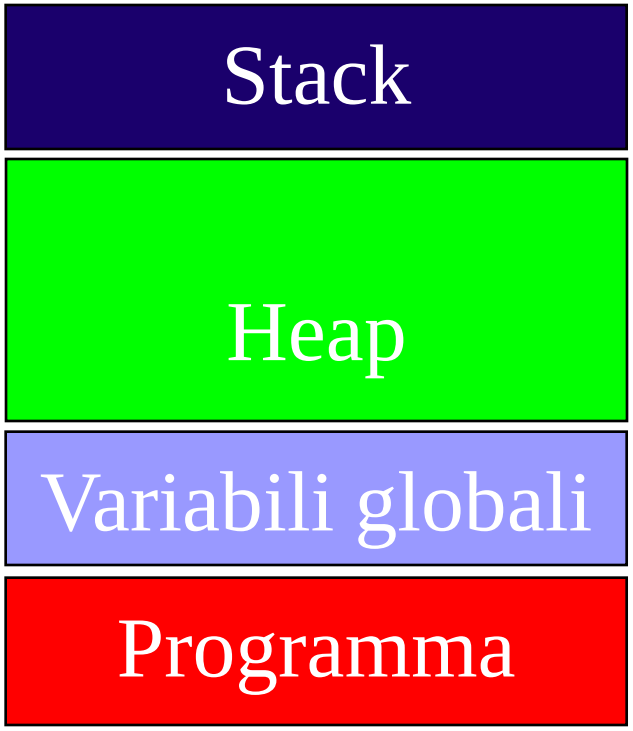
STRUTTURA DI UN PROGRAMMA C

<programma> ::=
<dichiarazioni globali>
<funzione> {<funzione>}
<funzione> ::=
<tipo_ritorno> <nome_funz> (<elenco_argomenti>)
{
 <sequenza_istruzioni>
}

Librerie e compilazione separata

- Compilazione di un programma C
  - creazione del programma
  - compilazione del programma
  - collegamento del programma con le funzioni di libreria richieste

Suddivisone concettuale della memoria per un programma C



STRUTTURA DI UN PROGRAMMA C

- **La parte <main> è l'unica *obbligatoria*, ed è definita come segue:**

```
main> ::=
<tipo> main()
{
[<dichiarazioni-e-definizioni>]
[<sequenza-istruzioni>]
}
```

- *Intuitivamente il main e' definito dalla parola chiave **main()** è racchiuso tra parentesi graffe al cui interno troviamo*
  - *le dichiarazioni e definizioni*
  - *una sequenza di istruzioni*

STRUTTURA DI UN PROGRAMMA C

<dichiarazioni-e-definizioni>

- *introducono i nomi* di costanti, variabili, tipi definiti dall'utente

<sequenza-istruzioni>

- sequenza di frasi del linguaggio ognuna delle quali è un'istruzione

- **Il main() è *una particolare unità di traduzione*** (una *funzione*).

STRUTTURA DI UN PROGRAMMA C

set di caratteri ammessi in un programma

- dipende dall'implementazione; solitamente ASCII + estensioni

identificatori

- sequenze di caratteri tali che

<Identificatore> ::= <Lettera> { <Lettera> | <Cifra> }

- *Intuitivamente un identificatore e' una sequenza (di lunghezza maggiore o uguale a 1) di lettere e cifre che inizia obbligatoriamente con una lettera.*

STRUTTURA DI UN PROGRAMMA C

commenti

- sequenze di caratteri racchiuse fra i delimitatori**/*** e ***/**

<Commento> ::= /* <frase> */

<frase> ::= { <parola> }

<parola> ::= { <carattere> }

commenti non possono essere innestati.

VARIABILI

Una *variabile* è un'astrazione della *cella di memoria.*

- **Formalmente, è un simbolo *associato a un indirizzo fisico (L-value)...***
- Perciò, l' **L-value** di x è 1328 **(fisso e immutabile!)**.

SIMBOLO	INDIRIZZO
x	

… che *denota un valore (R-value).*

4

..e l' **R-value** di x è ***attualmente*** 4 (può cambiare).

DICHIARAZIONE DI VARIABILE

- **Una variabile utilizzata in un programmadeve essere definita.**
- **La definizione è composta da**

  il *tipo* dei valori (R-value) che possono essere denotati alla variabile

  il nome della variabile (identificatore)

<tipo> <identificatore>;

Esempi

```
int x;      /* x deve denotare un valore intero */
float y;    /* y deve denotare un valore reale */
char ch;    /* ch deve denotare un carattere */
```

Definizione/Inizializzazione di una variabile

- **Contestualmente alla dichiarazione è possibile *specificare un valore iniziale* per una variabile**
- **Inizializzazione di una variabile:**

  <tipo> <identificatore> = <espr> ;

- **Esempio**

```
int x = 32;
double speed = 124.6;
```

Linguaggio C

Codifica in linguaggio C dell'algoritmo che converte gradi Celsius in Fahrenheit

```
main(){
   float c, f; /* Celsius e Fahrenheit */
   printf("Inserisci la temperatura da convertire");
   scanf("%f", &c);
   f = 32 + c * 9/5;
   printf("Temperatura Fahrenheit %f", f);
}
```

Caratteristiche di una variabile

- **campo d'azione** (**scope**): è la parte di programma in cui la variabile è nota e può essere manipolata

  in C è determinabile *staticamente*

- **tipo**: specifica la *classe di valori* che la variabile può assumere (e quindi gli operatori applicabili)
- **tempo di vita:** è l'intervallo di tempo in cui rimane valida l'associazione simbolo/indirizzo fisico (L-VALUE)
- **valore**: è rappresentato (secondo la codifica adottata) nell'area di memoria associata alla variabile

Espressioni

Il C e' un linguaggio basato su **espressioni**

Un’espressione e' una notazione che denota un valore mediante un processo di valutazione

Una espressione puo' essere semplice o composta tramite aggregazione di altre espressioni

- **Una variabile**
  - **può comparire in una espressione**
  - **può assumere un valore dato dalla valutazione di un'espressione**
- **double speed = 124.6;**
- **double time = 71.6;**
- **double km = speed * time;**

Operatori infissi, prefissi e postfissi

Le espressioni composte sono strutture formate da operatori applicati a uno o più operandi.

Dove posizionare l'operatore?

Tre possibili scelte:

prima $\Rightarrow$ notazione prefissa Esempio + 3 8

dopo $\Rightarrow$ notazione postfissa Esempio 3 8 +

in mezzo $\Rightarrow$ notazione infissa Esempio 3 + 8

Operatori infissi, prefissi e postfissi

Le notazioni prefissa e postfissa non hanno problemi di priorità e associatività degli operatori

non c’e' 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

Esempi * + 3 4 6

3 4 6 * +

3 * 4 + 6

Priorità degli operatori

La priorità degli operatori specifica l'ordine di valutazione degli operatori quando in una espressione compaiono operatori infissi diversi

Esempio

3 + 4 * 10

si legge 3 + (4 * 10)

N.b. operatori diversi possono avere uguale priorità

Associatività degli operatori

L’associatività degli operatori specifica l’ordine di valutazione degli operatori quando nell’espressione compaiono operatori infissi di ugual priorità

Un operatore puo' essere associativo a destra o a sinistra

3 - 10 + 8

+ e - sono equiprioritari e associativi a sinistra

Priorità e associatività predefinite possono essere alterate mediante l’uso delle parentesi

Operatori ed espressioni semplici

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** (char, int)

Espressioni composte

2 + f(x)

4 * 8 + pippo

Classificazione degli operatori

Due criteri per classificare gli operatori:

- in base al tipo degli operandi
- in base al numero di operatori

In base al tipo degli operandi	In base al numero degli operandi
Aritmetici	Unari
relazionali	binari
logici	ternari

Operatori aritmetici

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

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

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

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

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

Operatori di relazione

uguaglianza	**==**
diversità	**!=**
maggiore di	**>**
minore di	**<**
maggiore o uguale a	**>=**
minore o uguale a	**<=**

Espressioni e operatori di relazione

In C non esiste il tipo **boolean**

In C le espressioni **relazionali** denotano un tipo intero

- 0 denota il valore falso (condizione non verificata)
- 1 denota il valore vero (condizione verificata)

Quindi sono possibili espressioni miste come

(n != 0) == n

da evitare

Operatori logici

connettivo logico	operatore	C
not (negazione)	unario	!
and	binario	&&
or	binario	||

Espressioni e operatori logici

In C le espressioni **logiche** denotano un tipo intero da interpretare come vero (1) o falso (0)

Anche qui sono possibili espressioni miste, utili in alcuni casi,

5 && 7 !5 0 || 33

Valutazione in **corto circuito**

la valutazione dell'espressione cessa appena si e' in grado di determinare il risultato

il secondo operando e' valutato solo se necessario

Valutazione in corto circuito

22 || x

e' vera in partenza perché 22 e' vero

0 && x

e' falsa in partenza perché 0 e' falso

a && b && c

il secondo && viene valutato solo se a && b e' vero

a || b || c

il secondo || viene valutato solo se a || b e' falso

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

S**e 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**

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

Espressioni concatenate

Una espressione concatenata introdotta dall'operatore di concatenazione

espr1, espr2, … esprN

tutte le espressioni vengono valutate da sinistra a destra

l'espressione esprime il valore denotato da esprN