Linguaggio di Programmazione

Linguaggio di Programmazione

E` una notazione per descrivere gli algoritmi.

Programma:
e` la rappresentazione di un algoritmo in un particolare linguaggio di programmazione.

In generale, ogni linguaggio di programmazione dispone di un insieme di "parole chiave" (*keywords*), attraverso le quali e` possibile esprimere il flusso di azioni descritto dall'algoritmo.

Ogni linguaggio e` caratterizzato da una sintassi e da una semantica:

- **sintassi**: e' l'insieme di regole formali per la composizione di programmi nel linguaggio scelto. Le regole sintattiche dettano le modalita` di combinazione tra le parole chiave del linguaggio, per costruire correttamente istruzioni (frasi).

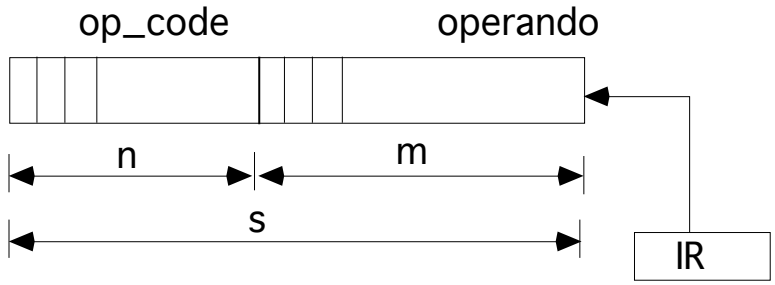
- **semantica**: e` l'insieme dei significati da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio scelto.

Il linguaggio macchina

Il **linguaggio macchina** e` direttamente eseguibile dall'elaboratore, senza nessuna traduzione.

- **Istruzioni**:
  Si dividono in due parti: un **codice operativo** ed, eventualmente, uno o piu' **operandi**:
  - Il **codice operativo** specifica l'operazione da compiere
  - gli **operandi** individuano le celle di memoria a cui si riferiscono le operazioni.

- Se consideriamo istruzioni ad un solo operando:



Istruzioni Macchina

s, lunghezza di un'istruzione (in bit)

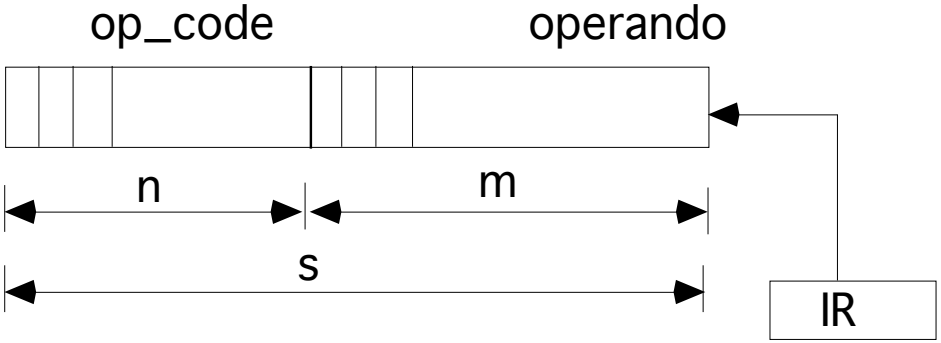
$$s = n + m$$

n, numero di bit dedicati al codice operativo

m, numero di bit dedicati all'indirizzamento degli operandi.

Insieme di istruzioni del linguaggio: al piu` 2^n istruzioni diverse (ciascuna ha un diverso *op_code*)

Memoria indirizzabile: al piu` 2^m celle di memoria diverse.



Set di Istruzioni di un elaboratore:

E` l'insieme delle istruzioni che la macchina e` in grado di eseguire direttamente.

- Ad esempio: 14 istruzioni (VAX della Digital 304!)

-> sono sufficienti 4 bit (16 >14).

op_code	*istruzione*
0000	LOADA
0001	LOADB
0010	STOREA
0011	STOREB
0100	READ
0101	WRITE
0110	ADD
0111	DIF
1000	MUL
1001	DIV
1010	JUMP
1011	JUMPZ
1100	NOP
1101	HALT

Linguaggio Macchina:Esempio

```
0   READ     8
1   READ     9
2   LOADA    8
3   LOADB    9
4   MUL
5   STOREA   8
6   WRITE    8
7   HALT
8   DATO INTERO
9   DATO INTERO
```

Rappresentazione reale (binaria):

0	0100	0000	0000	1000
1	0100	0000	0000	1001
2	0000	0000	0000	1000
3	0001	0000	0000	1001
4	1000	0000	0000	0000
5	0010	0000	0000	1000
6	0101	0000	0000	1000
7	1101	0000	0000	0000
8	0000	0000	0000	0000
9	0000	0000	0000	0000

Il linguaggio ASSEMBLER

E' difficile leggere e capire un programma scritto in forma binaria:

Linguaggi assemblatori (Assembler):

- Le istruzioni corrispondono univocamente a quelle macchina, ma vengono espresse tramite nomi simbolici (parole chiave).
- I riferimenti alle celle di memoria sono fatti mediante nomi simbolici (identificatori).
- Identificatori che rappresentano dati (costanti o variabili) oppure istruzioni (etichette).

- Il programma prima di essere eseguito deve essere tradotto in linguaggio macchina (**assemblatore**).

Il linguaggio ASSEMBLER: esempio

```
READ    X
READ    Y
LOADA   X
LOADB   Y
MUL
STOREA  X
WRITE   X
HALT
X   INT
Y   INT
```

Linguaggi di programmazione

Linguaggio Macchina:

- necessita` di conoscere dettagliatamente le caratteristiche della macchina (registri, dimensioni dati, set di istruzioni)
- conoscenza dei metodi di rappresentazione delle informazioni utilizzati.
- conoscenza della collocazione in memoria di istruzioni e dati

Linguaggio Assembler:

- corrispondenza uno-a-uno tra istruzioni macchina e istruzioni assembler
- possibilita` di esprimere in modo simbolico istruzioni e dati

Limite:

- i programmi dipendono strettamente dalle caratteristiche architetturali del microprocessore: NON C'E` PORTABILITA` !!

Linguaggi di programmazione di alto livello

Con un **linguaggio di alto livello** il programmatore puo` astrarre dai dettagli legati all'architettura ed esprimere i propri algoritmi in modo simbolico.

Linguaggi di alto livello: Sono indipendenti dalla macchina:

- Capacita` di astrazione nel progetto di programmi;
- Portabilita` dei programmi.

Esecuzione di programmi scritti in linguaggi di alto livello:

- E` necessaria la ***traduzione*** di ogni programma nella corrispondente sequenza di istruzioni macchina (direttamente eseguibili dal processore), attraverso:
- ***interpretazione*** (ad es. BASIC)
- ***compilazione*** (ad es. C, FORTRAN, Pascal)

Definizione di un linguaggio

- Un linguaggio di programmazione viene definito mediante:
  - un **alfabeto** (o vocabolario): stabilisce tutti i simboli che possono essere utilizzati nella scrittura di programmi
  - **sintassi**: specifica le regole grammaticali per la costruzione di frasi corrette (mediante la composizione di simboli)
  - **semantica**: associa un significato (ad esempio, in terminni di azioni da eseguire) ad ogni frase sintatticamente corretta.

Sintassi di un linguaggio di programmazione

La sintassi di un linguaggio puo` essere descritta in modo informale (ad esempio, a parole) oppure in modo formale (mediante una grammatica formale).

Grammatica formale:

e` una notazione matematica che consente di esprimere in modo rigoroso la sintassi dei linguaggi di programmazione.

Ad esempio: Grammatica BNF (Backus-Naur Form)

Una grammatica BNF e` un insieme di 4 elementi:

1. un **alfabeto terminale** V: e` l'insieme di tutti i simboli consentiti nella composizione di frasi sintatticamente corrette;
2. un **alfabeto non terminale** N (simboli indicati tra parentesi angolari < >)
3. un insieme finito di **regole** (produzioni) P del tipo:

   $X ::= A$

   dove $X \in N$, ed A e` una sequenza di simboli α (stringhe): $\alpha \in (N \cup V)$.
4. un **assioma** (o simbolo iniziale) $S \in N$

Grammatica BNF

Esempio: grammatica BNF relativa al "linguaggio" per esprimere i naturali

V={0,1,2,3,4,5,6,7,8,9}

N={<naturale>, <cifra-non-nulla>, <cifra>}

P:
<naturale> ::= 0| <cifra-non-nulla> {<cifra>}
<cifra-non-nulla> ::= 1|2|3|4|5|6|7|9
<cifra> ::= 0|<cifra-non-nulla>

S = <naturale>

BNF: alcune estensioni

- **X::=[a]B** ⇨ a puo' comparire zero od una volta ⇨ equivale a:
  X::=B|aB

- **X:={a}^{n}B** ⇨ a puo' comparire da 0 ad un massimo di n volte
  Esempio: **X:={a}^{3}B** , equivale a:
  X::=B|aB|aaB|aaaB

- **X:={a} B** ⇨ se n e` omesso, a puo' comparire da 0 ad un massimo finito arbitrario
  ⇨ equivale a: X::=B|aX (*ricorsiva*)

- Per raggruppare categorie sintattiche si possono usare le parentesi tonde:

  X::= (a | b) D | c ⇨ equivale a **X::= a D | b D | c**

EBNF : esempio

Numeri interi di lunghezza qualsiasi con o senza segno (non si permettono numeri con piu` di una cifra se quella piu` a sinistra e` 0 es: 059)

V ={0,1,2,3,4,5,6,7,8,9}U{+,-}

N={<intero>, <intero-senza-segno>, <cifra>, <cifra-non-nulla>}

S = <intero>

P:

<intero>::=[+|-] <intero-senza-segno>

<intero-senza-segno>::=0|
<cifra-non-nulla> {<cifra>}

<cifra>::= <cifra-non-nulla>|0

<cifra-non-nulla>::=1|2|3|4|5|6|7|8|9

Semantica di un linguaggio di programmazione

Attribuisce un significato ad ogni frase del linguaggio sintatticamente corretta.

- Molto spesso e` definita **informalmente** (per esempio, a parole).

- Esistono **metodi formali** per definire la semantica:

- semantica ***operazionale***: azioni

- semantica ***denotazionale***: funzioni matematiche

- semantica ***assiomatica***: formule logiche

⇨ Benefici per il **programmatore** (comprensione dei costrutti, prove formali di correttezza), l'**implementatore** (costruzione del traduttore corretto), **progettista** di linguaggi (strumenti formali di progetto).

IL LINGUAGGIO DI PROGRAMMAZIONE
C

Il linguaggio C

Progettato nel **1972** da D. M. Ritchie presso i laboratori AT&T Bell, per poter riscrivere in un linguaggio di alto livello il codice del sistema operativo UNIX.

Definizione formale nel **1978** (B.W. Kernigham e D. M. Ritchie)

Nel **1983** e` stato definito uno standard (***ANSI C***) da parte dell'American National Standards Institute.

Caratteristiche principali:

- Elevato potere espressivo:
  - ***Tipi di dato*** primitivi e tipi di dato definibili dall'utente
  - ***Strutture di controllo*** (programmazione strutturata, funzioni e procedure)
- Caratteristiche di basso livello (gestione delle memoria, accesso alla rappresentazione)
- Sintassi definita formalmente

Elementi del testo di un programma C

Nel testo di un programma C possono comparire:

- ***parole chiave*** sono parole riservate che esprimono istruzioni, tipi di dato,e altri elementi predefiniti nel linguaggio
- ***identificatori:*** nomi che rappresentano oggetti usati nel programma (ad esempio: variabili, costanti, tipi, funzioni ecc.)
- ***costanti***: numeri (interi o reali), caratteri e stringhe
- ***operatori*** : sono simboli che consentono la combinazione di dati in espressioni
- ***commenti***

- **Parole chiave:** sono parole riservate (cioe` non possono essere utilizzate come identificatori) che esprimono istruzioni, tipi di dato,e altri elementi predefiniti nel linguaggio:

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

Identificatori

Un identificatore e` un nome che denota un oggetto usato nel programma (ad esempio: **variabili**, **costanti**, **tipi**, **funzioni**).

- Deve iniziare con una lettera (o con il carattere '_'), alla quale possono seguire lettere e cifre in numero qualunque:

<identificatore>::=(<lettera>|_){<lettera>|<cifra>}

- distinzione tra maiuscole e minuscole (il linguaggio e` ***case-sensitive***)

Ad Esempio:

Sono identificatori validi:
Alfa beta _a2
Gamma1 Gamma2

Non sono identificatori validi:
3X int

Regola Generale:
prima di essere *usato*, ogni identificatore deve essere gia` stato definito in una parte di testo precedente.

Costanti

Valori interi: Rappresentano numeri relativi (quindi con segno):

	2 byte	4 byte
base decimale	**12**	**70000, 12L**
base ottale	**014**	**0210560**
base esadecimale	**0xFF**	**0x11170**

Valori reali:

24.0 2.4E1 240.0E-1

Suffissi: l, L, u, U (interi-long, unsigned)
f, F (reali - floating)

Prefissi:0 (ottale) 0x, OX(esadecimale)

Caratteri: Insieme dei caratteri disponibili (e` dipendente dalla realizzazione). In genere, ASCII esteso (256 caratteri). Si indicano tra singoli apici:

'a' 'A'
'1' ';'

Costanti

Caratteri speciali: sono caratteri ai quali non e` associato alcun simbolo grafico, ai quali e` associato un significato predefinito

newline	'\n'	se stampato, provoca l'avanzamento alla *linea* successiva
backspace	'\b'	se stampato, provoca l'arretramento al *carattere* precedente
form feed	'\f'	se stampato, provoca l'avanzamento alla *pagina* successiva
carriage return	'\r'	se stampato, provoca l'arretramento all'inizio della linea corrente

ecc.

Stringhe: Sono sequenze di caratteri tra doppi apici.

"abc" "a" ";&ç§" "" (stringa nulla)

Commenti:

Sono sequenze di caratteri ignorate dal compilatore.

- Vanno racchiuse tra /* e */

```
/*questo e`
   un commento
dell'autore  */
```

- I commenti vengono generalmente usati per introdurre ***note esplicative*** nel codice di un programma.

Struttura di un programma C

Nel caso piu` semplice, un programma C consiste in:

<programma>::=[<parte-dich-globale>] <main>[{<altre-funzioni>}]

Struttura del main:

<main> ::= `main()` { <parte-dichiarazioni> <parte-istruzioni> }

+ ⇨il <main> e` costituito da due parti:

- Una parte di **dichiarazioni** (variabili, tipi, costanti, etichette, etc.) in cui vengono descritti e definiti gli oggetti che vengono utilizzati dal main;

- Una parte **istruzioni** che descrive l'algoritmo risolutivo utilizzato, mediante istruzioni del linguaggio.

Formalmente:
il main e` una funzione che non restituisce alcun valore.

Esempio:

```
/*programma che, letti due numeri a terminale, ne
   stampa la somma*/

#include <stdio.h>
main()
{

int X,Y;  /* p. dichiarativa */

scanf("%d%d",&X,&Y);/*p.istruzioni*/
printf("%d",X+Y);

}
```

Variabili

Una **variabile** rappresenta un dato che puo` cambiare il proprio valore durante l'esecuzione del programma.

In generale: In ogni linguaggio di alto livello una variabile e` caratterizzata da un **nome** (***identificatore***) e 4 attributi base:

1. ***campo d'azione*** (*scope*), e` l'insieme di istruzioni del programma in cui la variabile e` nota e puo` essere manipolata;
   - C, Pascal, determinabile staticamente
   - LISP, dinamicamente
2. ***tempo di vita*** (o durata o estensione), e` l'intervallo di tempo in cui un'area di memoria e` legata alla variabile;
   - ForTRAN, allocazione statica
   - C, Pascal, allocazione dinamica
3. ***valore***, e` rappresentato (secondo la codifica adottata) nell'area di memoria legata alla variabile;
4. **tipo**, definisce l'insieme dei valori che la variabile puo` assumere e degli operatori applicabili.

Variabili in C

Ogni variabile, per poter essere utilizzata dalle istruzioni del programma, deve essere preventivamente definita.

Definizione di Variabili:

La definizione di variabile associa ad un identificatore (nome della variabile) un tipo.

<def-variabili>::=<identificatore-tipo> <identif-variabile> {,identif-variabile>};

Esempi:

```
int  A, B, SUM;/* Variabili A, B, SUM intere  */
float root, Root; /* Variab. root, Root reali */
char  C ;         /* Variabile C carattere */
```

Definizione di una Variabile

Effetto della definizione di variabile:

- La definizione di una variabile provoca come effetto l'***allocazione*** in memoria della variabile specificata (allocazione automatica).

- Ogni istruzione successiva alla definizione di una variabile A, potra` utilizzare A

Assegnamento

L' assegnamento e' l'operazione con cui si attribuisce un nuovo valore ad una variabile.

Il concetto di variabile nel linguaggio C rappresenta un'astrazione della cella di memoria.

L'assegnamento, quindi, e` l'astrazione dell'operazione di **scrittura** nella cella che la variabile rappresenta.

Esempi:

```
/*HP:  a, X e Y sono variabili */
int a;
float X,Y;
   ...
/*assegnamento ad a del   valore 100: */
   a=100;

/* assegnamento a Y del risultato di una espr. aritmetica: */

   Y = 2*3.14*X;
```

Tipo di dato

Il **tipo di dato** rappresenta una categoria di dati caratterizzati da proprieta` comuni.

In particolare:

un tipo di dato T e` definito

- dall'insieme di valori che le variabili di tipo T possono assumere;
- dall'insieme di operazioni che possono essere applicate ad operandi del tipo T.

Per esempio:

Consideriamo i numeri naturali

Tipo_naturali = [**N**, {+, -, *, /, =, >, <, … }]

- N e` il dominio
- {+, -, *, /, =, >, <, … } e` l'insieme delle operazioni effettuabili sui valori del dominio.

Il Tipo di dato

Un linguaggio di programmmazione e` ***tipato*** se prevede costrutti specifici per attribuire tipi ai dati utilizzati nei programmi.

Se un linguaggio e` tipato:

- Ogni dato (variabile o costante) del programma deve appartenere ad uno ed un solo tipo.
- Ogni operatore richiede operandi di tipo specifico e produce risultati di tipo specifico.

Vantaggi:

- **Astrazione**: l'utente esprime e manipola i dati ad un livello di astrazione piu` alto della loro organizzazione fisica (bit !). Maggior portabilita`.
- **Protezione**: Il linguaggio protegge l'utente da combinazioni errate di dati ed operatori (controllo statico sull'uso di variabili, etc. in fase di compilazione).
- **Portabilita`**: l'indipendenza dall'architettura rende possibile la compilazione dello stesso programma su macchine profondamente diverse.

Il Tipo di Dato in C

Il C e` un linguaggio tipato.

Classificazione dei tipi di dato in C: due criteri di classificazione "ortogonali"

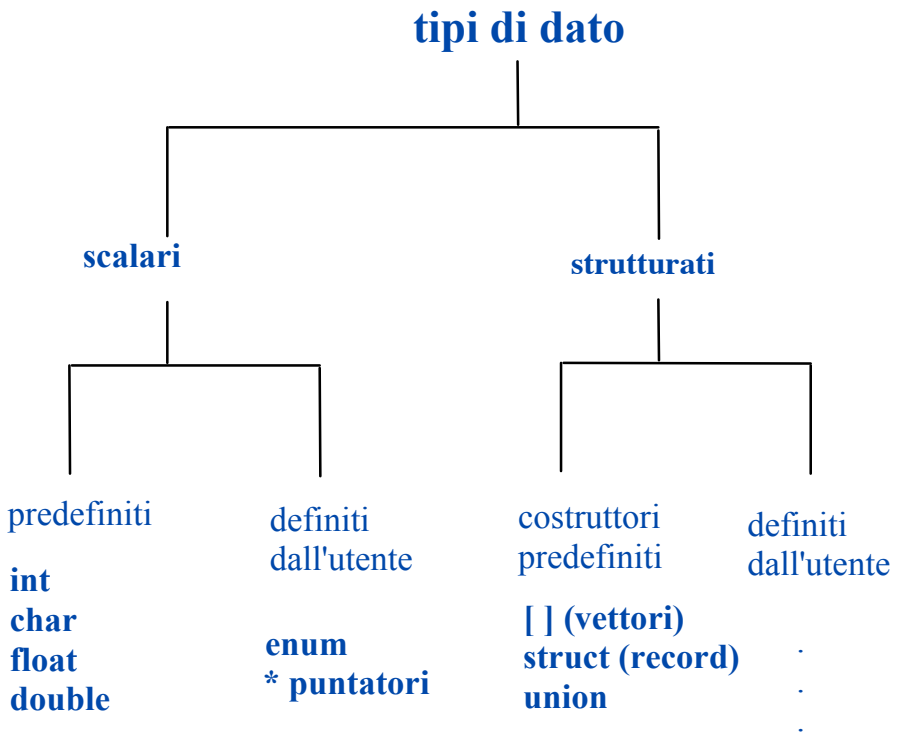
1. Si distingue tra:

- ***tipi primitivi***: sono tipi di dato previsti dal linguaggio (built-in) e quindi rappresentabili direttamente.
- ***tipi non primitivi***: sono tipi definibili dall'utente (mediante appositi costruttori di tipo, v. typedef).

2. Inoltre, si distingue tra:

- ***tipi scalari***, il cui dominio e` costituito da elementi atomici, cioe` logicamente non scomponibili.
- ***tipi strutturati***, il cui dominio e` costituito da elementi non atomici (e quindi scomponibili in altri componenti).

Classificazione dei tipi di dato in C



Tipi scalari primitivi

Il C prevede quattro tipi scalari primitivi:

- `char` (caratteri)
- `int` (interi)
- `float` (reali)
- `double` (reali in doppia precisione)

Il tipo `int`

Dominio:

Il dominio associato al tipo `int` rappresenta l'insieme dei numeri interi con segno: ogni variabile di tipo `int` e` quindi l'astrazione di un intero.

Esempio: definizione di una variabile intera.

(E` la frase mediante la quale si *introduce* una nuova variabile nel programma.)

```
int A; /* A e` una variabile intera */
```

⇨ Poiche` si ha sempre a disposizione un numero finito di bit per la rappresentazione dei numeri interi, il dominio rappresentabile e` di estensione finita.

Hp:

se il numero n di bit a disposizione per la rappresentazione di un intero e` 16, allora il dominio rappresentabile e` composto di:

$(2^n) = 2^{16} = 65.536$ *valori*

Quantificatori e qualificatori

A dati di tipo `int` e` possibile applicare i quantificatori *short* e *long*: influiscono sullo spazio in memoria richiesto per l'allocazione del dato.

- **`short`: la rappresentazione della variabile in memoria puo` utilizzare un numero di bit ridotto rispetto alla norma.**
- **`long`: la rappresentazione della variabile in memoria puo` utilizzare un numero di bit aumentato rispetto alla norma.**

Esempio:

```
int X;      /* se X e` su 16 bit..*/
long int Y;/*..Y e` su 32 bit */
```

I quantificatori possono influire sul dominio delle variabili:

- Il dominio di un `long int` puo` essere piu` esteso del dominio di un int.
- Il dominio di uno `short int` puo` essere piu` limitato del dominio di un int.

Gli effetti di short e long sui dati dipendono strettamente dalla realizzazione del linguaggio.

In generale:

spazio(short int) <= spazio(int) <= spazio(long int)

Quantificatori e qualificatori

A dati di tipo `int` e` possibile applicare i qualificatori *signed* e *unsigned*:

- **`signed`**: viene usato un bit per rappresentare il segno.Quindi l'intervallo rappresentabile e`:

$$[-2^{n-1}-1, +2^{n-1}-1]$$

- **`unsigned`**: vengono rappresentati valori a priori positivi. Intervallo rappresentabile:

$$[0, (2^{n}-1)]$$

I qualificatori condizionano il dominio dei dati.

Il tipo int

Operatori:

Al tipo int (e tipi ottenuti da questo mediante qualificazione/quantificazione) sono applicabili gli operatori ***aritmetici***, ***relazionali*** e ***logici***.

Operatori aritmetici:

forniscono risultato intero:

+ , -, *, /	somma, sottrazione, prodotto, divisione intera.
%	operatore modulo: calcola il resto della divisione intera. 10%3 →1
++, --	incremento e decremento: richiedono un solo operando (una variabile) e possono essere postfissi (a++) o prefissi (++a) (v. espressioni)

Operatori relazionali

Si applicano ad operandi interi e producono risultati *logici* (o "*booleani*").

Booleani: sono grandezze il cui dominio e` di due soli valori (valori logici): {*vero*, *falso*}

Operatori relazionali:

==, != uguaglianza (==), disuguaglianza(!=):

10==3 → *falso*

10!=3 → *vero*

<, >, <=, >= minore, maggiore, minore o uguale, maggiore o uguale

10>=3 → *vero*

Booleani

Sono dati il cui dominio e` di due soli valori (valori logici):

{*vero*, *falso*}

➔ in *C* **non esiste un tipo primitivo per rappresentare dati booleani** !

Come vengono rappresentati i risultati di espressioni relazionali ?

Il *C* prevede che i valori logici restituiti da espressioni relazionali vengano rappresentati attraverso gli interi {0,1} secondo la convenzione:

- *0* equivale a *falso*
- *1* equivale a *vero*

Ad esempio:

l'espressione A == B restituisce:

→ 0, se la relazione non e` vera
→ 1 , se la relazione e` vera

Operatori logici

Si applicano ad operandi di tipo logico (in C: `int`) e producono risultati *booleani* (cioe` interi appartenenti all'insieme {0,1}).

In particolare l'insieme degli operatori logici e`:

&& operatore AND logico
|| operatore OR logico
! operatore di negazione (NOT)

Definizione degli operatori logici:

a	b	a&&b	a||b	!a
falso	*falso*	*falso*	*falso*	*vero*
falso	*vero*	*falso*	*vero*	*vero*
vero	*falso*	*falso*	*vero*	*falso*
vero	*vero*	*vero*	*vero*	*falso*

Operatori Logici in C

In C, gli operandi di operatori logici sono di tipo int:

- se il valore di un operando e` **diverso da zero**, viene interpretato come ***vero***.
- se il valore di un operando e` **uguale a zero**, viene interpretato come ***falso***.

Definizione degli operatori logici in C:

a	b	a&&b	a||b	!a
0	0	0	0	1
0	$\neq 0$	0	1	1
$\neq 0$	0	0	1	0
$\neq 0$	$\neq 0$	1	1	0

Valutazione a *Corto circuito*:

Gli operatori di and e or vengono eseguiti valutando gli operandi da sinistra a destra:

- A&&B: viene valutato A; se A è 0 il risultato è certamente 0 -> B non viene valutato
- A||B: viene valutato A; se A è $\neq 0$ il risultato è certamente 1 -> B non viene valutato

Operatori tra interi: esempi

37 / 3	→	12
37 % 3	→	1
7 < 3	→	0
7 >=3	→	1
5 >=5	→	1
0 || 1	→	1
0 || -123	→	1
15||0	→	1 ("corto circuito")
12 && 2	→	1
0 && 17	→	0 ("corto circuito")
! 2	→	0

I tipi reali: `float` e `double`

Dominio:
Concettualmente, e` l'insieme dei numeri reali R.

In realta` , e` un sottoinsieme di R :
- limitatezza del dominio (limitato numero di bit per la rappresentazione del valore).
- precisione limitata: numero di bit finito per la rappresentazione della parte frazionaria (*mantissa*)

Lo spazio allocato per ogni numero reale (e quindi l'insieme dei valori rappresentabili) dipende dalla realizzazione del linguaggio (e, in particolare, dal metodo di rappresentazione adottato).

Differenza tra float/double:

`float`	singola precisione
`double`	doppia precisione (maggiore numero di bit per la *mantissa*)

➔ Alle variabili `double` e` possibile applicare il quantificatore `long`, per aumentare ulteriormente la precisione: spazio(float) <= spazio(double) <= spazio(long double)

Esempio: definizione di variabili reali

```
float x;/* x e` una variabile reale "a singola precisione"*/
double A, B; /* A e B sono reali "a doppia precisione"*/
```

I tipi `float` e `double`

Operatori: per dati di tipo reale sono disponibili operatori aritmetici e relazionali.

Operatori aritmetici:

+, -, *, / si applicano a operandi reali e producono risultati reali

Operatori relazionali: hanno lo stesso significato visto nel caso degli interi:

==, != uguale, diverso

<, >, <=, >= minore, maggiore etc.

Operazioni su reali: esempi

5.0 / 2 → 2.5
2.1 / 2 → 1.05
7.1 < 4.55 → 0
17== 121 → 0

➔ A causa della rappresentazione finita, ci possono essere errori di conversione. Ad esempio, i test di uguaglianza tra valori reali (in teoria uguali) potrebbero non essere verificati.

(x / y) * y == x

Meglio utilizzare "un margine accettabile di errore":
(X == Y) → `(X<=Y+epsilon) && (X>=Y-epsilon)`

dove, ad esempio: `float epsilon=0.000001;`

Il tipo `char`

Dominio:

E` l'insieme dei caratteri disponibili sul sistema di elaborazione (set di caratteri).

Comprende:

- le lettere dell'alfabeto
- le cifre decimali
- i simboli di punteggiatura
- altri simboli di vario tipo (@, #, $ etc.)
- caratteri speciali (backspace, carriage return, ecc.)
- ...

Tabella dei Codici

Il dominio coincide con l'insieme rappresentato da una ***tabella dei codici***, *dove*, ad ogni carattere viene associato un intero che lo identifica univocamente (il ***codice***).

- Il dominio associato al tipo char e` **ordinato**: l'ordine dipende dal codice associato ai vari caratteri.

La tabella dei codici ascii

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
0	(null)	NUL	32	(space)	64	@	96	
1	☺	SOH	33	!	65	A	97	a
2	☻	STX	34	"	66	B	98	b
3	♥	ETX	35	#	67	C	99	c
4	♦	EOT	36	$	68	D	100	d
5	♣	ENQ	37	%	69	E	101	e
6	♠	ACK	38	&	70	F	102	f
7	(beep)	BEL	39	'	71	G	103	g
8	◘	BS	40	(	72	H	104	h
9	(tab)	HT	41	)	73	I	105	i
10	(line feed)	LF	42	*	74	J	106	j
11	(home)	VT	43	+	75	K	107	k
12	(form feed)	FF	44	´	76	L	108	l
13	(carriage return)	CR	45	-	77	M	109	m
14	♫	SO	46	.	78	N	110	n
15	☼	SI	47	/	79	O	111	o
16	►	DLE	48	0	80	P	112	p
17	◄	DC1	49	1	81	Q	113	q
18	↕	DC2	50	2	82	R	114	r
19	!!	DC3	51	3	83	S	115	s
20	π	DC4	52	4	84	T	116	t
21	§	NAK	53	5	85	U	117	u
22	▬	SYN	54	6	86	V	118	v
23	↨	ETB	55	7	87	W	119	w
24	↑	CAN	56	8	88	X	120	x
25	↓	EM	57	9	89	Y	121	y
26	→	SUB	58	:	90	Z	122	z
27	←	ESC	59	;	91	[	123	{
28	(cursor right)	FS	60	<	92	\	124	|
29	(cursor left)	GS	61	=	93	]	125	}
30	(cursor up)	RS	62	>	94	^	126	~
31	(cursor down)	US	63	?	95	_	127	⌂

Il tipo char

Definizione di variabili di tipo char: esempio

```
char C1, C2;
```

Costanti di tipo char:

Ogni valore di tipo char viene specificato tra singoli apici.

```
'a'  'b'    'A'    '0'    '2'
```

Rappresentazione dei caratteri in C:

Il linguaggio C rappresenta i dati di tipo char come degli interi su 8 bit:

- ogni valore di tipo char viene rappresentato dal suo **codice** (cioe`, dall'intero che lo indica nella tabella ASCII)

➔ Il dominio associato al tipo `char` e` **ordinato**: l'ordine dipende dal codice associato ai vari caratteri nella tabella di riferimento.

Il tipo char: Operatori

I char sono rappresentati da interi (su 8 bit):

➔ sui char e` possibile eseguire tutte le operazioni previste per gli interi. Ogni operazione, infatti, e` applicata ai codici associati agli operandi.

Operatori relazionali:

==,!=,<,<=,>=,> per i quali valgono le stesse regole viste per gli interi

Ad esempio:

```
char x,y;
```

`x < y` se e solo se codice(x) < codice(y)

`'a'>'b'` *falso* perche` codice('a') < codice('b')

Operatori aritmetici:

sono gli stessi visti per gli interi.

Operatori logici:

sono gli stessi visti per gli interi.

Operazioni sui char: esempi

Esempi:

'A' < 'C'	→	1	(infatti 65 < 67 e` vero)
' " ' + ' ! '	→	'C'	(codice(")+codice(!)=67)
! 'A'	→	0	(codice(A) e` diverso da 0)
'A' && 'a'	→	1	

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. Per esempio:

```
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
```

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).

Conversione implicita di tipo

REGOLA:

Data una espressione *x 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.** L'operazione specificata puo` essere eseguita se il tipo degli operandi e` compatibile con il tipo dell'operatore. Il risultato è di tipo uguale a quello prodotto dall'operatore effettivamente eseguito.

Conversione implicita di tipo: esempio

```
int x;
char y;
double r;
```

`(x+y) / r`

Hp: 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 (*double*)**

Conversione esplicita di tipo

In C si puo` forzare la conversione di un dato in un tipo specificato, mediante l'operatore di ***casting*** :

(<nuovo tipo>) <dato>

➔ il <dato> viene convertito esplicitamente nel <nuovo tipo>.

Esempio:

```
int A, B;
float C;
C=A/(float)B;
```

➔ viene eseguita la divisione tra reali.

Definizione /inizializzazione di variabili di tipo semplice

Definizione di variabili

Tutti gli identificatori di tipo primitivo descritti fin qui possono essere utilizzati per definire variabili.

Ad esempio:

```
char lettera;
int, x, y;
unsigned int P;
float media;
```

Inizializzazione di variabili

E` possibile specificare un valore iniziale di una variabile in fase di definizione.

Ad esempio:

```
int x =10;
char y = 'a';
double r = 3.14*2;
int z=x+5;
```