

LINGUAGGI DI PROGRAMMAZIONE, SISTEMA DEI TIPI, SISTEMA DEGLI OPERATORI

Fondamenti di Programmazione 2022/2023

Francesco Tortorella




Linguaggi di programmazione

- Abbiamo visto come le informazioni si possono codificare come sequenze di bit di dimensione prefissata.
- Una simile rappresentazione può essere assunta anche per le **istruzioni**, cioè le singole azioni elementari che l'unità centrale può eseguire.



Linguaggi di programmazione

- Nello specificare un'istruzione, bisogna precisare l'**operazione** da compiere e i **dati** coinvolti nell'operazione.

- Esempio:


- Come rappresentare le operazioni ?
- L'insieme delle diverse operazioni che l'unità centrale è in grado di eseguire è **finito** e quindi è possibile codificarlo con un certo numero di bit (**codice operativo**).

somma	0000
sottrai	0001
moltiplica	0010
dividi	0011
...	...

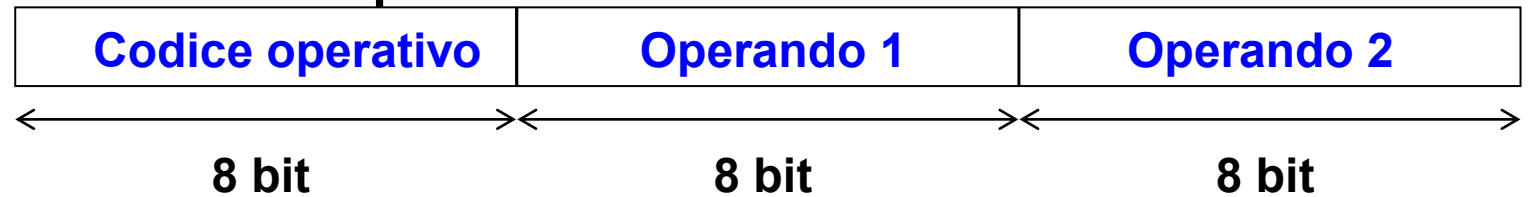


Linguaggi di programmazione

- Una istruzione sarà quindi rappresentabile da una sequenza di bit divisa in due parti:

- un **codice operativo**

- uno o più **operandi**



- In questo modo, un esecutore automatico può permettere la memorizzazione e l'esecuzione di un **programma**, cioè di una sequenza di istruzioni che realizzano un particolare algoritmo e che sono descritte nel linguaggio interpretabile dal calcolatore, ma ...



Linguaggio macchina

- ... ma quali sono le caratteristiche di tale linguaggio ?
 - è codificato tramite sequenze di bit
 - ogni istruzione può compiere solo azioni molto semplici. In particolare, non sono disponibili come istruzioni di base i costrutti visti precedentemente.
 - non gestisce direttamente i tipi di dati di interesse
 - è strettamente legato alla particolare macchina su cui è definito
- Non a caso viene definito **linguaggio macchina**



Scrivere un programma ...

- Se si volesse implementare un dato algoritmo scrivendo un programma in linguaggio macchina sarebbe quindi necessario:
 - conoscere dettagliatamente tutti i codici operativi e la loro codifica
 - decidere in quali porzioni di memoria vadano memorizzati i dati
 - definire un'opportuna tecnica di codifica per ogni tipo di dati considerato
 - determinare, per ogni singola operazione richiesta dall'algoritmo, la sequenza di istruzioni in linguaggio macchina che la realizzano
- limitarsi a utilizzare solo i calcolatori per cui esista una tale competenza, tenendo comunque presente che il programma scritto per un certo calcolatore non è eseguibile su altre macchine

Impresa difficile, ma non impossibile ...



Scrivere un programma ...

■ Esecutore umano

- linguaggio formale (es. flow chart), semplice e di uso generale
- istruzioni in grado di implementare direttamente i principali costrutti di programmazione
- gestione completa dei tipi

■ Sistema di elaborazione

- linguaggio rigido, complicato e specifico del particolare esecutore
- istruzioni estremamente semplici
- gestione dei tipi quasi nulla



Scrivere un programma ...

- Come passare in maniera semplice e rigorosa dalla descrizione dell'algoritmo comprensibile all'esecutore umano ad un programma eseguibile dal sistema di elaborazione?
- Possibile se fosse disponibile un linguaggio ad alto livello che
 - metta a disposizione del programmatore un supporto semplice ed efficace all'impiego dei costrutti di programmazione e dei principali tipi di dati
 - permetta una traduzione affidabile del programma espresso in tale linguaggio in un programma espresso in linguaggio macchina eseguibile sul sistema di elaborazione



Linguaggi ad alto livello (HLL)

- Linguaggi formali, con costrutti precisi per la definizione dei dati e delle operazioni.
- Permettono una gestione completa dei tipi fondamentali. Hanno la possibilità di definire tipi strutturati.
- Offrono un'implementazione immediata dei costrutti di programmazione visti precedentemente.



Vantaggi

- L'uso di linguaggi ad alto livello permette di :
 - realizzare un programma che implementa l'algoritmo in maniera precisa tramite costrutti più vicini al livello di astrazione con il quale un essere umano descrive un algoritmo
 - trascurare tutti i dettagli relativi alla rappresentazione dei dati all'interno del sistema di elaborazione
 - realizzare un programma che non dipende dal particolare sistema di elaborazione su cui è stato realizzato



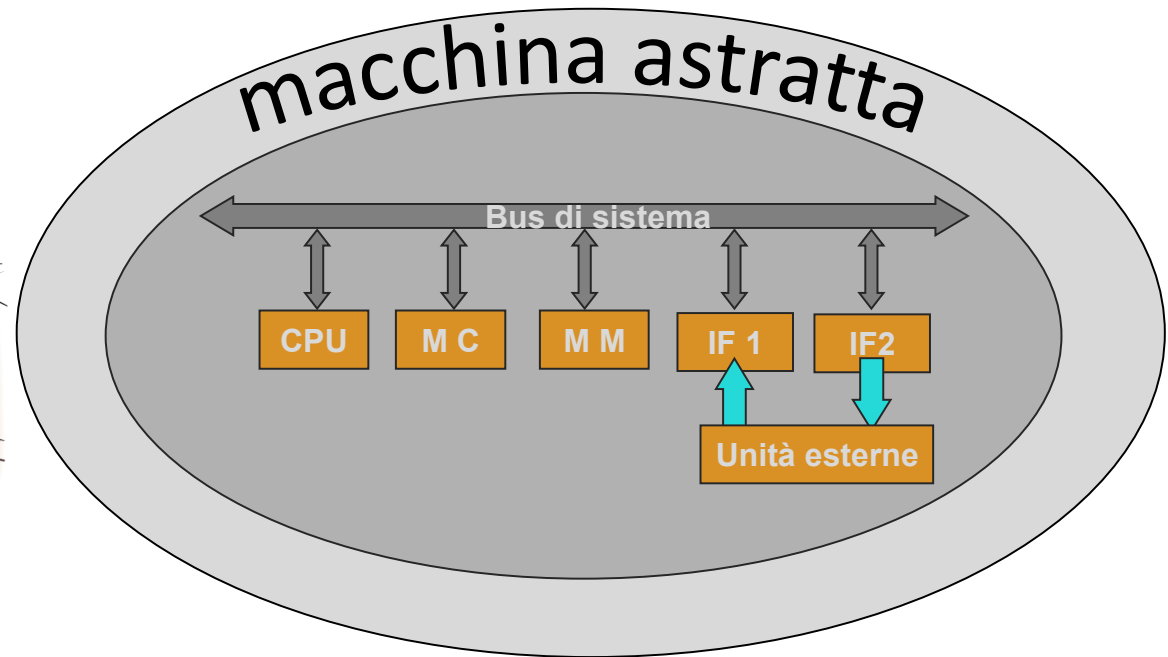
Vantaggi

- Tali caratteristiche agevolano significativamente il compito della programmazione, rendendola più semplice e veloce.
- Soprattutto consentono di scrivere programmi di qualità adeguata, facilmente **manutenibili** e **riusabili**.



Linguaggio=macchina virtuale

In effetti, il programmatore non deve interagire con la macchina reale e le sue limitazioni, ma “vede” una **macchina astratta** che nasconde le particolarità della macchina reale e con la quale è molto più agevole interagire



Dal linguaggio ad alto livello al linguaggio macchina

- I linguaggi di programmazione ad alto livello sono **linguaggi formali**, in cui la forma delle frasi, cioè la **sintassi**, e il loro significato, la **semantica**, sono definiti sulla base di regole rigide e precise.
- In tal modo viene eliminata l'ambiguità e le ridondanze tipiche del linguaggio naturale ed **è possibile realizzare in modo automatico l'analisi di un programma scritto in un linguaggio ad alto livello e la sua traduzione in linguaggio macchina.**
- I programmi che svolgono il compito di tradurre un programma in linguaggio ad alto livello in un programma in linguaggio macchina sono detti **traduttori** (*compilatori o interpreti*).



Compilatori ed interpreti

- La traduzione eseguita tramite compilatore (**compilazione**) si applica all'intero programma scritto in linguaggio ad alto livello (**linguaggio sorgente**) e produce un file oggetto, contenente la traduzione in linguaggio macchina del codice sorgente, ma non ancora eseguibile.
- Una successiva operazione di **collegamento** (**linking**) completa il codice oggetto con codice messo a disposizione dal compilatore (**libreria** del compilatore) e produce il programma eseguibile finale.



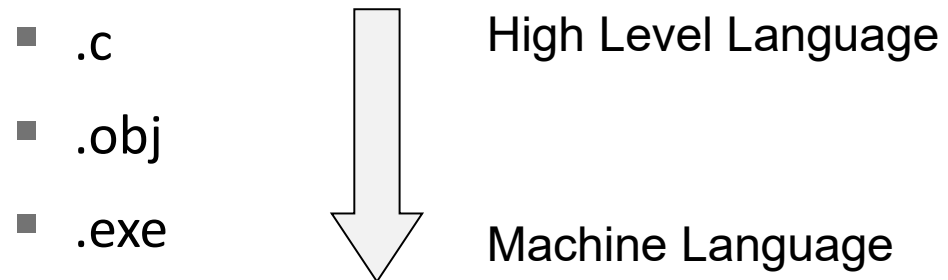
Compilatori ed interpreti

- Un interprete effettua la traduzione istruzione per istruzione
 - iterativamente viene letta un'istruzione del programma in linguaggio sorgente, viene tradotta nel corrispondente insieme di istruzioni in linguaggio macchina e queste ultime eseguite
- Una soluzione ibrida tra compilazione ed interpretazione prevede la compilazione del codice sorgente in un linguaggio intermedio, detto **bytecode**. Il bytecode viene in fase di esecuzione interpretato da una *virtual machine* (VM) ossia un interprete di basso livello che emula una CPU.

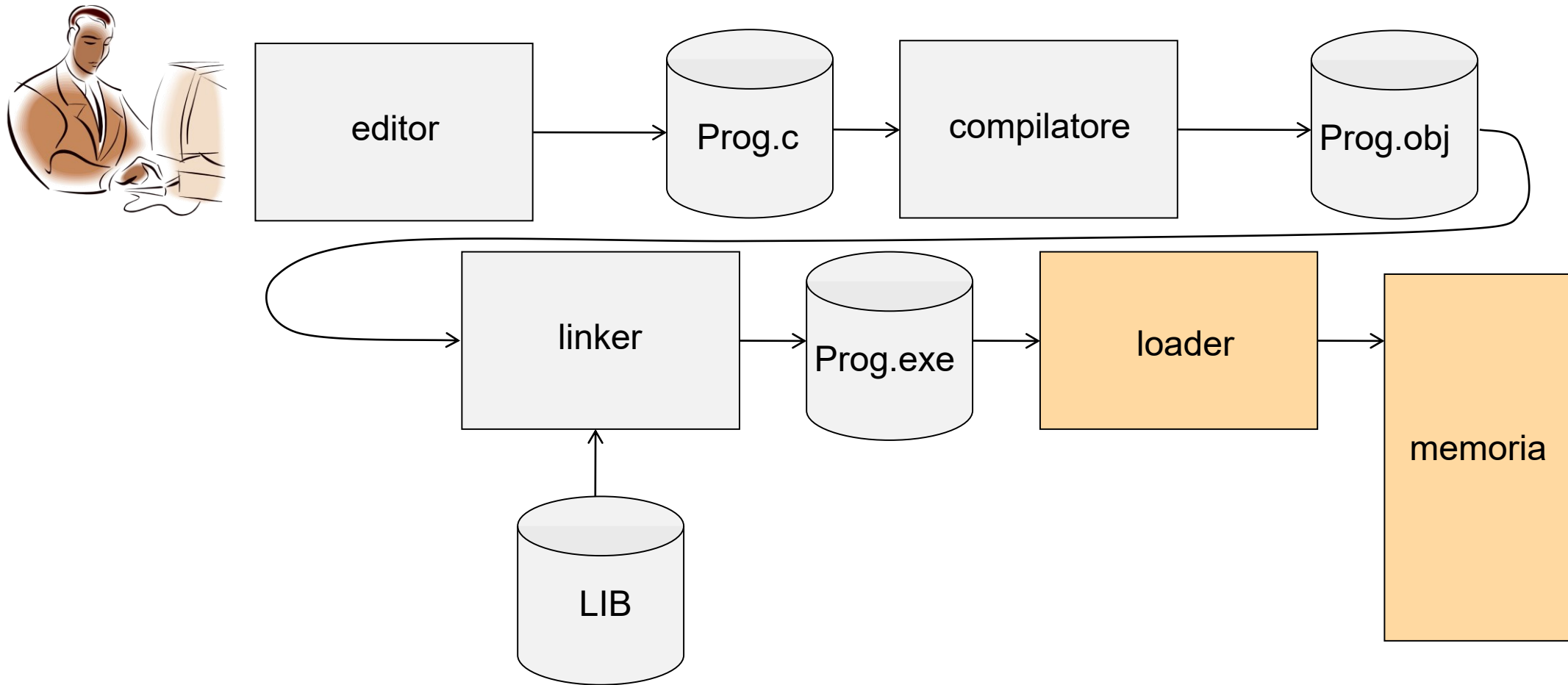


Dal produttore all'...esecutore

- Prima di essere eseguito, un programma attraversa le seguenti fasi:
 - Compilazione (traduzione)
 - Collegamento
 - Caricamento in memoria
- I prodotti delle varie fasi sono ospitati in files



Dal produttore all'...esecutore



Gli strumenti di sviluppo

- Per ognuna delle fasi evidenziate sono disponibili strumenti software appositi:
 - Compilatore (compiler)
 - Collegatore (linker)
 - Caricatore (loader)
- Un ambiente agevole per la progettazione e la realizzazione di programmi è fornito dagli **Integrated Development Environment (IDE)**
- Un IDE è una collezione di strumenti software integrati, con interfaccia visuale, che aiuta il programmatore nello sviluppo di un programma in tutte le fasi della realizzazione.



Gli strumenti di sviluppo

- Tipicamente un IDE è composto da:
 - un **editor** di codice sorgente guidato da sintassi (syntax highlighting); è in grado di riconoscere le principali strutture lessicali e sintattiche del linguaggio e agevolare la fase di redazione del programma, evidenziando le parole chiave, le strutture di controllo e operando semplici controlli nel codice sorgente per diminuire la probabilità di errori;
 - un **compilatore** e/o un interprete che possono essere avviati all'interno dell'IDE stesso, in maniera friendly;
 - un tool di building automatico, che collega tutte le unità che costituiscono il programma e produce il file eseguibile;
 - un **debugger** che offre la possibilità di eseguire il programma controllandone l'esecuzione mediante la visualizzazione dei valori che assumono le sue variabili o prevedendone l'avanzamento istruzione per istruzione.



Il sistema dei tipi

- Abbiamo visto che, per ogni informazione presente in un programma, va specificato il tipo entro cui essa appartiene.
- A questo scopo, ogni linguaggio di programmazione mette a disposizione del programmatore un insieme di tipi predefiniti detto **sistema dei tipi**
- I tipi considerati sono di fatto piuttosto comuni



Il sistema dei tipi

- **Tipi numerici:**
 - intero
 - reale
 - reale doppia precisione
- **Tipi non numerici:**
 - carattere
 - logico
 - stringa



Il tipo `intero`

- È costituito da un sottoinsieme limitato dei numeri interi
- Caratteristiche:
 - Rappresentazione in complementi alla base su un numero prefissato m di bit
 - Esempio: per $m=32$
 - Valore minimo: -2147483648
 - Valore massimo: +2147483647
- Operazioni ammesse:
 - Assegnazione =
 - Somma +
 - Sottrazione -
 - Moltiplicazione *
 - Divisione /
 - Modulo %
 - Confronto > < >= <= == !=



Il tipo `real`

- È costituito da un sottoinsieme limitato e discreto dei numeri reali
- Caratteristiche:
 - Rappresentazione su un numero prefissato m di bit (*floating point*)
 - Esempio per $m=32$:
 - Valore minimo (abs): $3.4E-38$
 - Valore massimo (abs): $3.4E+38$
- Operazioni ammesse:
 - Assegnazione =
 - Somma +
 - Sottrazione -
 - Moltiplicazione *
 - Divisione /
 - Confronto > < >= <= == !=



Il tipo **reale** doppia precisione

- È costituito da un sottoinsieme limitato e discreto dei numeri reali, ma con range e precisione maggiore rispetto al tipo **reale**
- Caratteristiche:
 - Rappresentazione su un numero prefissato m di bit (*floating point*) maggiore di quanto fissato per il tipo **reale**
 - Esempio per $m=64$:
 - Valore minimo (abs): $1.7E-308$
 - Valore massimo (abs): $1.7E+308$
- Operazioni ammesse:
 - Assegnazione =
 - Somma +
 - Sottrazione -
 - Moltiplicazione *
 - Divisione /
 - Confronto > < >= <= == !=



Il tipo carattere

- Consiste in un insieme di caratteri, alcuni stampabili (caratteri alfabetici, cifre, caratteri di punteggiatura, ecc.) ed altri non stampabili tramite i quali si gestisce il formato dell'input/output (**caratteri di controllo**).
- I sottoinsiemi delle lettere e delle cifre sono ordinati e coerenti.
- Per la rappresentazione interna, viene tipicamente usato il codice ASCII, che mette in corrispondenza ogni carattere con un numero intero compreso tra 0 e 255.



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]



Il tipo logico

- È un tipo costituito dai due soli valori **vero** e **falso**. Il tipo rappresenta le informazioni di tipo logico (es. il risultato di un confronto, il verificarsi di una situazione).
- Operazioni ammesse
 - assegnazione =
 - disgiunzione ||
 - congiunzione &&
 - negazione !



Il tipo `stringa`

- Consente di trattare un'informazione costituita da una sequenza di caratteri (es. una parola)



Tipi enumerativi

- Un tipo enumerativo è un **tipo di dati definito dal programmatore** costituito da un insieme di valori denominati **elementi** o **membri**.
- I nomi degli elementi sono generalmente identificatori che si comportano come costanti nel linguaggio.
- Ad una variabile appartenente ad un tipo enumerativo può essere assegnato uno qualsiasi degli elementi come valore.
- Gli elementi di un tipi enumerativo sono confrontabili e ordinati.
- La rappresentazione interna è tipicamente gestita direttamente dal compilatore.



Tipi enumerativi - esempio

- Si assuma che **Semi** sia un tipo enumerativo formato dagli elementi **{picche, fiori, quadri, cuori}**
- Siano **x, y, z** variabili di tipo **Semi**; sono lecite le seguenti operazioni:

x = quadri

y = fiori

z = y

- Assumendo che l'ordine degli elementi nella definizione del tipo definisca anche il loro ordinamento (dal minimo al massimo), qual è il valore delle seguenti espressioni?

x >= y

z == picche



Il sistema degli operatori

- Un **operatore** è un simbolo che specifica un'operazione da eseguire su uno o due **operandi** definendo un'**espressione**.
- All'interno di un linguaggio il **sistema degli operatori** è un insieme di regole e convenzioni sintattiche che consentono al programmatore di definire e valutare in maniera corretta espressioni anche complesse che coinvolgono più operatori ed operandi.



Il sistema degli operatori

- Il sistema degli operatori specifica i seguenti aspetti:
 - operatori disponibili e loro tipologia
 - precedenza degli operatori
 - associatività degli operatori



Tipologia degli operatori

- Gli operatori possono essere classificati in base al numero di operandi che accettano, ovvero in base al numero di dati su cui lavorano:
 - gli **operatori unari** lavorano su un singolo operando
 - gli **operatori binari** lavorano su due operandi
 - gli **operatori ternari** lavorano su tre operandi



Tipologia degli operatori

- Un'altra possibile classificazione si basa sulla tipologia di operazione realizzata:
- Operatori aritmetici
- Operatori relazionali
- Operatori logici



Operatori aritmetici

- Comprendono le quattro operazioni fondamentali e, in aggiunta, altre operazioni possibili (es. elevazione a potenza, modulo, ...)
 - Esempio: $+$ $-$ $/$ $*$ $\%$ $^$
- Possono essere
 - **unari**
 - Esempio: $+3$ $-x$
 - **binari**
 - Esempio: $a+b$ $x*5$ $q\%2$



Operatori relazionali (o di confronto)

- Hanno lo scopo di confrontare i valori degli operandi cui si applicano, e forniscono un risultato logico.
- Esempio: l'operatore `>` applicato a due operandi consente di verificare se il valore dell'operando a sinistra è maggiore del valore dell'operando a destra (`x > 3`). Quando l'espressione viene valutata, il valore restituito è **vero** se la relazione è verificata, **falso** altrimenti.
- Operatori tipici: `>` `<` `≥` `≤` `==` `≠`



Operatori logici

- Sono operatori che si applicano a valori di tipo **logico**:
 - **and** congiunzione
 - **or** disgiunzione
 - **not** negazione
- Gli operatori **and** e **or** sono **binari**, mentre l'operatore **not** è **unario**.



Operatori logici

- Come accade per gli operatori aritmetici
 - gli operatori binari si scrivono tra i due operandi
 - l'operatore unario si antepone all'unico operando
- Assumiamo che A e B siano due variabili di tipo **logico**, cioè due variabili che possono assumere uno tra i due valori **vero** e **falso**. Si scrive:
 - A **and** B
 - A **or** B
 - **not** A



Operazioni logiche

- Per capire quale sia il risultato delle operazioni che possiamo effettuare con gli operatori visti, consideriamo quali siano le diverse combinazioni che possono assumere i loro operandi

A
F(false)
T(true)

A	B
F	F
F	T
T	F
T	T

Operazioni logiche

- L'operazione **A and B** restituisce **true** se e solo se sia A che B sono **true**, altrimenti restituisce **false**
- L'operazione **A or B** restituisce **false** se e solo se sia A che B sono **false**, altrimenti restituisce **true**
- L'operazione **not A** restituisce **true** se A è **false**, restituisce **false** se A è **true**



Operazioni logiche

A	not A
F	T
T	F

A	B	A or B
F	F	F
F	T	T
T	F	T
T	T	T

A	B	A and B
F	F	F
F	T	F
T	F	F
T	T	T



Operazioni logiche

- Esistono diverse notazioni alternative per gli operatori logici
 - **and**: \wedge , `&&`
 - **or**: \vee , `||`
 - **not**: \neg , `!`
- Esempio:
 - espressione vera se x è compreso tra 18 e 30: **`(x >= 18) && (x <= 30)`**



Operazioni interne ed esterne

- Definiamo **interne** le operazioni il cui risultato è dello stesso tipo degli operandi coinvolti nell'operazione
 - Esempio (x e y siano variabili intere):
 - $-x$ $7*y$
- Definiamo **esterne** le operazioni in cui il tipo del risultato è diverso dal tipo degli operandi coinvolti nell'operazione
 - Esempio (x e y siano variabili intere):
 - $x < 0$ $x \geq 9$



Precedenza e associatività

- Quando si considerano espressioni complesse, contenenti diversi operatori, è necessario avere delle regole che stabiliscano come valutarle senza ambiguità
- Esempi: $-2*3+18$ $7-4+2$
- In particolare è necessario stabilire per ogni operatore quale sia
 - la sua precedenza
 - la sua associatività



Precedenza

- Un insieme di regole che stabiliscono le convenzioni relative a quali operazioni eseguire per prime per valutare una data espressione.
- In particolare, un operatore viene applicato se tutti gli eventuali altri operatori più prioritari sono già stati applicati.



Precedenza – regole comuni

- Gli operatori unari **+** e **-** e quello logico **not** hanno precedenza maggiore degli altri operatori aritmetici, relazionali e logici.
- Gli operatori aritmetici hanno priorità maggiore rispetto agli operatori logici e di quelli relazionali.
- Tra gli operatori aritmetici, ***** e **/** hanno la medesima priorità, maggiore della priorità di **+** e **-**, che hanno la stessa priorità.
- Gli operatori relazionali hanno priorità inferiore agli operatori aritmetici.
- Gli operatori logici **and** e **or** hanno una priorità inferiore a quella degli operatori relazionali; l'operatore **and** ha priorità maggiore di quella di **or**.
- Esempi: $-2*3+18$? $a+b > c-d$?



Associatività

- L'associatività di un operatore è una proprietà che determina come vengono raggruppati gli operatori della stessa precedenza in assenza di parentesi
- Un operatore è associativo a sinistra se, a parità di priorità, viene applicato da sinistra verso destra.



Associatività – regole comuni

- Per riflettere l'uso normale, gli operatori di addizione, sottrazione, moltiplicazione e divisione sono solitamente associativi a sinistra.
- Per lo stesso motivo, gli operatori unari $+$, $-$ e **not** sono associativi a destra.
- Esempi: $7-4+2$? $8^{*}-+2$?

