2

AMBIENTI DI PROGRAMMAZIONE

È l'insieme dei programmi che consentono la scrittura, la verifica e l'esecuzione di nuovi programmi (***fasi di sviluppo***).

Sviluppo di un programma:

- Affinché un programma scritto in un qualsiasi linguaggio di programmazione sia comprensibile (e quindi eseguibile) da un calcolatore, *occorre* ***tradurlo*** dal linguaggio originario al linguaggio della macchina.
- Questa operazione viene normalmente svolta da speciali programmi, detti ***traduttori***.

2

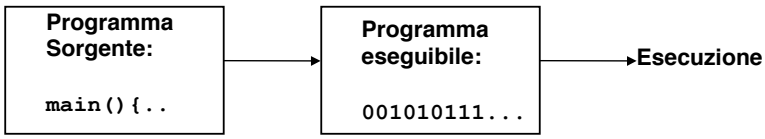
TRADUZIONE DI UN PROGRAMMA

PROGRAMMA	TRADUZIONE
`main()`	
`{ int A;`	`00100101`
`...`	
`A=A+1;`	`11001..`
`if....`	`1011100..`

Il traduttore converte

- ***il testo*** di un programma scritto in un particolare linguaggio di programmazione (***sorgenti***)
- nella corrispondente ***rappresentazione in linguaggio macchina*** (programma ***eseguibile***).

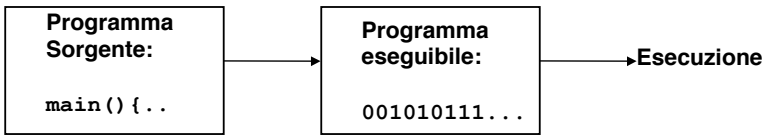
SVILUPPO DI PROGRAMMI



Due categorie di traduttori:

- i ***Compilatori*** traducono l'intero programma (senza eseguirlo!) e producono in uscita il programma convertito in linguaggio macchina
- gli ***Interpreti*** traducono ed eseguono immediatamente **ogni singola istruzione** del *programma sorgente.*

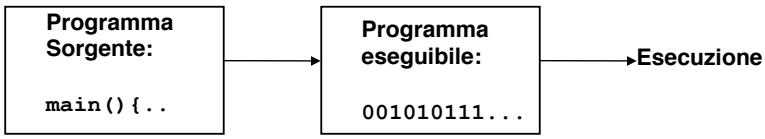
SVILUPPO DI PROGRAMMI



Quindi:

- **nel caso del compilatore**, lo schema precedente viene percorso ***una volta sola*** prima dell'esecuzione
- **nel caso dell'interprete**, lo schema viene invece attraversato ***tante volte quante sono le istruzioni*** che compongono il programma.

SVILUPPO DI PROGRAMMI



Generalmente, l'**esecuzione** di un programma ***compilato*** è **più veloce** dell'esecuzione di un programma ***interpretato***

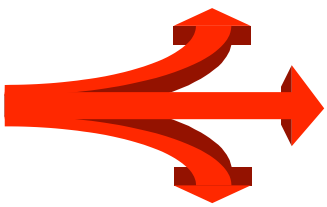
Però c'è da eseguire un passo di compilazione, che nel caso dell'interpretazione manca

AMBIENTI DI PROGRAMMAZIONE

COMPONENTI

- **Editor:** serve per creare file che contengono **testi** (cioè sequenze di caratteri).
  In particolare, l'editor **consente di scrivere il *programma sorgente.***

E poi….



2

AMBIENTI DI PROGRAMMAZIONE

I° CASO: COMPILAZIONE

- **Compiler:** opera la **traduzione di un programma *sorgente*** (scritto in un linguaggio ad alto livello) **in un *programma oggetto*** direttamente eseguibile dal calcolatore.

PRIMA si traduce ***tutto il programma***
POI si esegue **la versione tradotta**.

2

AMBIENTI DI PROGRAMMAZIONE

I° CASO: COMPILAZIONE (segue)

- **Linker:** nel caso in cui la costruzione del programma oggetto richieda l'unione di ***più moduli*** (compilati separatamente), il linker provvede a **collegarli** formando un unico *programma eseguibile.*
- **Debugger:** consente di **eseguire passo-passo** un programma, **controllando via via quel che succede**, al fine di ***scoprire ed eliminare errori*** non rilevati in fase di compilazione.

2

AMBIENTI DI PROGRAMMAZIONE

II° CASO: INTERPRETAZIONE

- **Interpreter: *traduce ed esegue*** direttamente ***ciascuna istruzione*** del *programma sorgente*, **istruzione per istruzione**.
  È alternativo al compilatore (raramente sono presenti entrambi).

Traduzione ed esecuzione sono *intercalate*, e avvengono *istruzione per istruzione.*

1

LINGUAGGI DI PROGRAMMAZIONE

- Un linguaggio di programmazione viene definito mediante:
  - **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 termini di azioni da eseguire) ad ogni frase sintatticamente corretta.

SEMANTICA

Attribuisce un significato ad ogni frase del linguaggio sintatticamente corretta.

- Molto spesso è definita **informalmente** (per esempio, a parole).
- Esistono **metodi formali** per definire la semantica di un linguaggio di programmazione in termini di…
  - azioni (semantica ***operazionale***)
  - funzioni matematiche (semantica ***denotazionale***)
  - formule logiche (semantica ***assiomatica***)

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

SINTASSI DI UN LINGUAGGIO DI PROGRAMMAZIONE

La sintassi di un linguaggio può essere descritta:
- in modo informale (ad esempio, *a parole*), oppure
- in modo formale (mediante una *grammatica formale*).

Grammatica formale:

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

Un formalismo molto usato:

BNF (Backus-Naur Form)

GRAMMATICHE BNF

Una grammatica BNF è un insieme di 4 elementi:

1. un **alfabeto terminale** V: è 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∈ N, ed A è una sequenza di simboli α ("stringhe") tali che $\alpha \in (N \cup V)$.
4. un **assioma** (o simbolo iniziale, o scopo) S ∈ N

ESEMPIO DI GRAMMATICA BNF

Esempio: il "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>

DERIVAZIONE

Una BNF definisce un **linguaggio** sull'alfabeto terminale, mediante un meccanismo di ***derivazione*** (o riscrittura):

Definizione:

Data una grammatica G e due stringhe β, γ elementi di $(N \cup V)^*$, si dice che

- **γ deriva direttamente da β**
- **($\beta \rightarrow \gamma$)**
- se le stringhe si possono decomporre in:

$\beta = \eta\ A\ \delta$ $\qquad$ $\gamma = \eta\ \alpha\ \delta$

ed esiste la produzione $A ::= \alpha$.

Si dice che **γ deriva da β** se: $\beta = \beta 0 \rightarrow \beta 1 \rightarrow \beta 2 \rightarrow \ldots \rightarrow \beta n = \gamma$

IL TOPO MANGIA IL GATTO

V : { `il` , `gatto` , `topo` , `Tom` , `Jerry` , `mangia` }

N : { <frase> , <soggetto> , <verbo> , <compl-oggetto> , <articolo> , <nome> }

S : <frase>

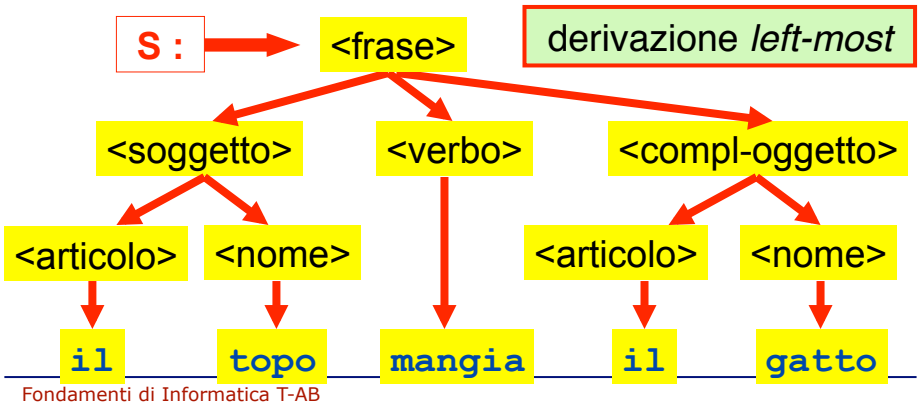
P *(**p**roduzioni)* :

<frase> ::= <soggetto> <verbo> <compl-oggetto>
<soggetto> ::= <articolo> <nome> | <nome-proprio>
<compl-oggetto> ::= <articolo> <nome> | <nome-proprio>
<articolo> ::= `il`
<nome> ::= `gatto` | `topo`
<nome-proprio> ::= `Tom` | `Jerry`
<verbo> ::= `mangia`

ALBERO SINTATTICO

Albero sintattico: esprime il processo di derivazione di una frase mediante una grammatica. Serve per analizzare la correttezza sintattica di una stringa di simboli terminali.

"`il topo mangia il gatto`"



LINGUAGGIO

Definizioni:

Data una grammatica G, si dice **linguaggio generato da G**, LG, l'insieme delle sequenze di simboli di V (frasi) derivabili, applicando le produzioni, a partire dal simbolo iniziale S.

Le frasi di un linguaggio di programmazione vengono dette **programmi** di tale linguaggio.

EBNF: Extended BNF

- **X ::= [a]B**
  a puo' comparire **al più una** volta: equivale a **X ::= B | aB**
- **X ::= {a}^{n} B**
  a puo' comparire da **0** a **n** volte
  Es.: **X ::= {a}^{3} B** , equivale a **X ::= B | aB | aaB | aaaB**
- **X ::= {a} B**
  equivale a **X ::= B | aX** (*ricorsiva*)
  (a puo' comparire da 0 ad un massimo **finito arbitrario** di volte)
- **()** per raggruppare categorie sintattiche:
  Es.: **X ::= (a | b) D | c** equivale a **X ::= a D | b D | c**

ESEMPIO

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

V : {0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 } ∪ { + , – }
N : { <intero> , <int-senza-segno> , <cifra> , <cifra-non-nulla> }
S : <intero>
P :

<intero>	::=	[+ | –] <intero-senza-segno>
<int-senza-segno>	::=	0 | <cifra-non-nulla> { <cifra> }
<cifra>	::=	<cifra-non-nulla> | 0
<cifra-non-nulla>	::=	1 | 2 | 3 | 4 | 5 | 6 | 7 | 9

1

Albero sintattico:

Albero sintattico per la generazione del numero ***-3547*** usando la grammatica EBNF:

3

Il linguaggio C
Struttura e sintassi di un programma

```
main()
{
/*definizioni variabili: */
char y='a'; /*codice(a)=97*/
int       x,X,Y;
unsigned int Z;
float SUM;
double r;

/* 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;        /* X=100, Y=0, Z=100 , SUM=1000.0*/
x=y;          /* char -> int: x=97*/
x=y+x; /*x=194*/
r=y+1.33; /* char -> int -> double*/
x=r; /* coercizione -> troncamento: x=98*/
}
```

Breve storia e caratteristiche

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** è stato definito uno standard (***ANSI C***) da parte dell'American National Standards Institute.

Alcune caratteristiche:

- 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 della memoria, accesso alla rappresentazione

Esempio di programma in C

```
#include <stdio.h>
main()  {
 // dichiarazione variabili
 int A, B;
 // input dei dati
 printf( "Immettere due numeri: " );
 scanf( "%d %d", &A, &B );
 // eseguo semisomma e mostro risult.
 printf( "Semisomma: %d\n", (A+B)/2 );
}
```

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, etc.)
- ***costanti:*** numeri (interi o reali), caratteri e stringhe
- ***operatori:*** sono simboli che consentono la combinazione di dati in espressioni
- ***commenti***

Elementi del programma in C

```
/* programma per la semisomma */

#include <stdio.h
main()  {
 // dichiarazione variabili
 int A, B;
 // input dei dati
 printf( "Immettere due numeri: " );
 scanf( "%d %d", &A, &B );
 // eseguo semisomma e mostro risult.
 printf( "Semisomma: %d\n", (A+B)/2 );
}
```

commenti

Parole chiave

Esprimono istruzioni, tipi di dato,e altri elementi predefiniti nel linguaggio

Sono parole riservate (cioè non possono essere utilizzate come identificatori)

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 è un nome che denota un oggetto usato nel programma (es.: **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 (***case-sensitive***)

Es.: **Alfa , beta , Gamma1 , X3** sono identificatori validi
3X , int *non* sono identificatori validi

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`, `0X` (esadecimale)

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

`'a'` `'A'` `' '`
`'1'` `';'` `'\''`

Costanti

Caratteri speciali: sono caratteri ai quali non è associato alcun simbolo grafico, ai quali è 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.

`"a"` `"aaa"` `""` (stringa nulla)

Commenti:

Sono sequenze di caratteri ignorate dal compilatore:
- vanno racchiuse tra `/*` e `*/` …
- …oppure precedute da `//`

```
/*       questo codice non deve essere eseguito:
int X, Y;
*/
int A, B; // ho cambiato i nomi alle variabili
```

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

Struttura di un programma C

Nel caso più semplice, un programma C consiste in:

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

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

dichiarazioni: oggetti che verranno utilizzati dal main (variabili, tipi, costanti, etichette, etc.);

istruzioni: implementano l'algoritmo risolutivo utilizzato, mediante istruzioni del linguaggio.

Formalmente, **il main è una funzione**

Esempio:

```
#include <stdio.h>
main()  {
 // dichiarazione variabili
 int A, B;
 // input dei dati
 printf( "Immettere due numeri: " );
 scanf( "%d %d", &A, &B );
 // eseguo semisomma e mostro risult.
 printf( "Semisomma: %d\n", (A+B)/2 );
}
```

dichiarazioni

istruzioni