COMPILATORI

INTRODUZIONE

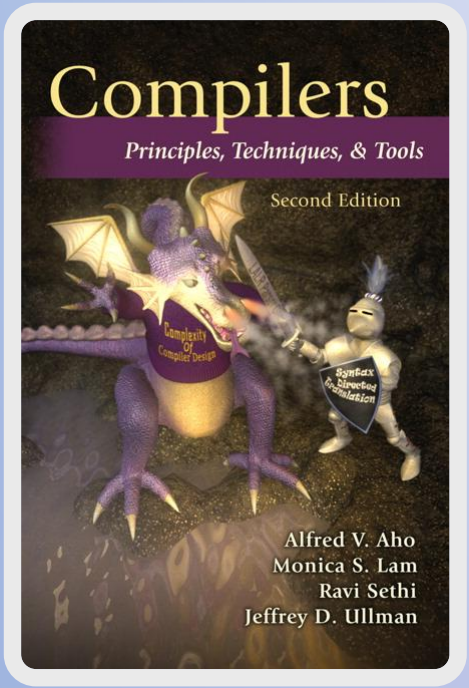
INFORMAZIONI GENERALI

- CFU: **6** ORE: **48**
- DOCENTE: **PROF. SABRINA MANTACI**
- ORARIO DELLE LEZIONI

  LUNEDI 11:00 – 13:30

  GIOVEDI 8.30:00 – 11:00
- ORARIO DI RICEVIMENTO:
  - GIOVEDI 15.00-18.00 CON PRENOTAZIONE TRAMITE PORTALE OPPURE
  - SU APPUNTAMENTO TRAMITE EMAIL sabrina.mantaci@unipa.it

LIBRI DI TESTO



- *A. Aho, M. Lam, R. Sethi, J. Ullman.* COMPILERS, Principles, techniques & tools *Addison Wesley*
- *A. Aho, M. Lam, R. Sethi, J. Ullman.* COMPILATORI. Principi, tecniche e strumenti *Addison* Wesley

LIBRI DI TESTO

- *S. Crespi Reghizzi, L. Breveglieri, A. Morzenti.* Linguaggi formali e compilazione *Società Editrice Esculapio, 2015*
- *S. Crespi Reghizzi, L. Breveglieri, A. Morzenti.* Formal languages and compilation *Springer 2013*
- Manuali di Flex & Bison (https://web.iitd.ac.in/~sumeet/flex_bison.pdf)
- Materiale didattico del docente

PROPEDEUTICITÀ

Per sostenere gli esami di Compilatori è necessario avere superato gli esami di tutte le materie propedeutiche al momento dell'esame.

Materie propedeutiche

PROGRAMMAZIONE E LABORATORIO C.I.

ALGORITMI E STRUTTURE DATI

INFORMATICA TEORICA

LINGUAGGI DI PROGRAMMAZIONE

SISTEMI OPERATIVI

ARCHITETTURE DEGLI ELABORATORI

MATEMATICA DISCRETA

Quelle sottolineate sono delle propedeuticità culturali oltre che formali

MODALITÀ D' ESAMI

Gli esami consistono di una prova pratica e una prova orale.

- Una settimana prima della data dell'esame, su richiesta degli studenti interessati verrà assegnato un progetto individuale o a coppie di studenti, che dovranno svilupparlo utilizzando flex e bison durante la settimana prima dell'esame.
- Il progetto va consegnato entro le ore 12:00 del giorno prima dell'esame orale.
- Durante l'esame orale gli studenti dovranno discutere il progetto e rispondere a domande inerenti alla parte teorica del corso.
- Il calendario degli esami sarà disponibile a breve
- E' prevista una prova in itinere (scritta) durante la pausa didattica di Aprile

OBIETTIVI DEL CORSO

1. Illustrare la struttura, le tecniche di costruzione e il funzionamento di un moderno compilatore
2. Usare le tecniche per la realizzazione di compilatori mediante l'uso di strumenti automatici:
   - Flex (fast lexical analyzer generator)
   - Bison (parser generator)
3. Applicare principi e tecniche per la progettazione dei compilatori in vari contesti legati all'analisi di testi.

UNA PANORAMICA SUI COMPILATORI

CONTESTO: LINGUAGGI DI PROGRAMMAZIONE

I linguaggi di programmazione sono gli strumenti di base dei programmatori. Tutti i software che girano sui calcolatori sono scritti in un qualche linguaggio di programmazione.

Prima che un programma possa essere eseguito deve essere prima trasformato (tradotto) in sequenze di istruzioni elementari espresse nel linguaggio che è comprensibile a un computer(linguaggio macchina).

Gli strumenti di traduzione si usano spesso, a volte "inconsapevolmente", nei processori di linguaggi di programmazione.

PROCESSORE DI UN LINGUAGGIO DI PROGRAMMAZIONE

E' un qualsiasi sistema che manipola programmi espressi in un particolare linguaggio. Consente di scrivere i programmi e prepararli per l'esecuzione. Può incorporare una varietà di sistemi tra cui:

- **Editors.** Consente la digitazione, modifica e salvataggio in un file.
- **Traduttori e compilatori:** Un traduttore traduce un testo scritto in un linguaggio in un altro. In particolare, un compilatore traduce programmi da linguaggio ad alto livello a linguaggio a basso livello, quindi prepara per l'esecuzione su una macchina. Prima di effettuare la traduzione un compilatore controlla la presenza di errori sintattici e contestuali.
- **Interpreti.** Un interprete prende un programma espresso in un particolare linguaggio e lo esegue immediatamente. Questa modalità di esecuzione, che omette la fase di compilazione, è preferibile in ambienti interattivi (ad es. molti linguaggi di interrogazione dei database sono interpretati)

PROCESSORI SEPARATI O INTEGRATI

Processori come software separati – utilizzati per esempio nel S.O. Unix

Processori o ambienti di sviluppo integrati (ide)

Per esempio per Java: eclipse, netbeans, borland jbuilder, per C: visual studio, devc++

In tali sistemi utente può aprire, editare, compilare ed eseguire il programma.

- L'editor consente di distinguere tra variabili, parole chiave, commenti;
- il compilatore è integrato con l'editor, nel caso di errori di compilazione rimanda alla frase contenente presumibilmente l'errore;
- se ci sono errori di esecuzione, si rimanda nell'editor alla frase del codice oggetto in cui è fallita l'esecuzione .

UN PO' DI STORIA…

IL LINGUAGGIO MACCHINA

Il linguaggio macchina è il linguaggio nativo del computer su cui un certo programma viene eseguito. Esso consiste di sequenze di bit che sono eseguite da un meccanismo interno al computer. Può essere espresso in sistema binario o in esadecimale.

Quando i primi computer elettronici Turing completi apparvero (nel tra il 1940 e il 1950, modello di John von Neumann), era possibile programmare solo in linguaggio macchina, con ovvie difficoltà nella realizzazione di programmi complessi.

PROGRAMMARE IN LINGUAGGIO MACCHINA

Le operazioni (estremamente semplici) realizzabili mediante linguaggio macchina sono dette di basso livello e sono strettamente legate al tipo di macchina. Esse sono del tipo:

- Spostare dati da una locazione all'altra;
- Confrontare due valori;
- Sommare il contenuto di 2 registri;

Esempio: sul processore Intel 80x86: il codice **C7 06 0000 0002** dice alla CPU di inserire 2 nella locazione 0000.

IL LINGUAGGIO ASSEMBLY

I **linguaggi assembly** sono nati negli anni '50. Inizialmente erano soltanto rappresentazioni mnemoniche di istruzioni in linguaggio macchina, ovvero abbreviazioni o parole che rendevano più facile ricordare le sequenze di bit. Vengono introdotte le variabili per rappresentare aree di memoria e i primi tipi di dato fondamentali come gli interi e i reali. Dopo, vennero aggiunte anche le istruzioni macro.

Esempio: il codice precedente potrebbe diventare **mov x,2.**

Un programma in linguaggio assembly è tradotto in linguaggio macchina da un programma chiamato **assembler** che traduce una alla volta le istruzioni scritte in assembly in codice macchina.

I LINGUAGGI AD ALTO LIVELLO

- I software per i primi computer erano scritti in linguaggio assembly.
- I linguaggi ad alto livello (meno dipendenti cioè dalla macchina) nacquero dall'esigenza di riutilizzare lo stesso software su diversi tipi di macchine.
- Il più grande vantaggio di questi linguaggi consiste nel fatto di rendere più semplice la scrittura dei programmi perché più vicini al ragionamento umano includendo addirittura termini del linguaggio naturale (if, while, for…). Essi danno la possibilità di utilizzare istruzioni che implementano lunghe sequenze di istruzioni in linguaggio macchina. In media un'istruzione in linguaggio ad alto livello corrisponde a 10 istruzioni in assembly.

ESEMPIO: In un linguaggio ad alto livello (per esempio il linguaggio C) l'istruzione precedente potrebbe diventare

x=2.

I LINGUAGGI AD ALTO LIVELLO

Con l'esigenza di introdurre dei linguaggi ad alto livello nasce l'esigenza di scrivere un software in grado di trasformare automaticamente i programmi scritti in tali linguaggi in programmi in linguaggio macchina.

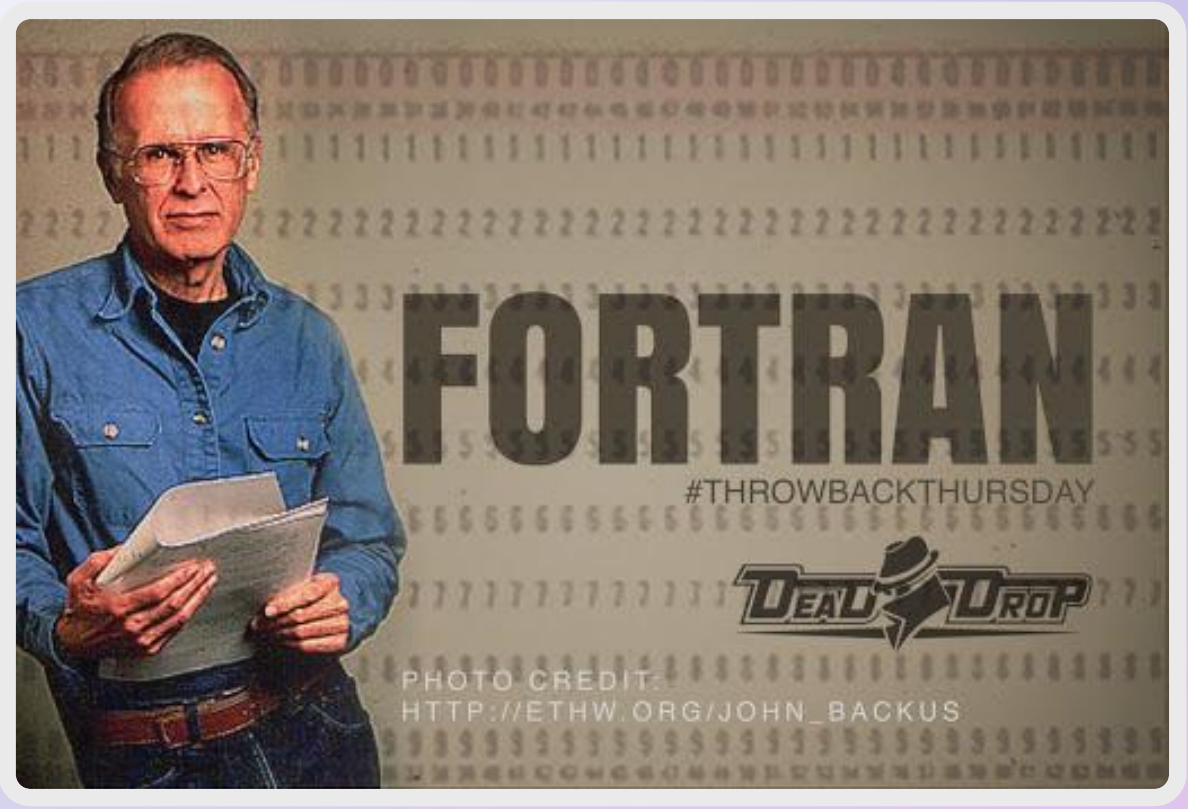
L'idea di un linguaggio automaticamente "traducibile" in linguaggio macchina, ma più vicino alla logica umana fu introdotta in informatica negli anni cinquanta. Nascono i primi compilatori.

NASCITA DEI COMPILATORI

Il termine **compilatore** fu coniato da **Grace Murray Hopper** nel 1950. E' stata una matematica, informatica e militare statunitense che è stata creatrice del primo compilatore, quindi il primo linguaggio di programmazione ad alto livello, l'A-o System (1951), e inoltre ebbe un ruolo fondamentale nello sviluppo del COBOL, introdotto nel 1961, il primo compilatore per la gestione degli archivi commerciali, che poteva essere usato su diverse architetture.

IL FORTRAN

A **John Backus** (IBM), si deve invece il primo linguaggio ad alto livello ad avere avuto ampia diffusione, il Fortran (1957), adatto al calcolo scientifico per cui riceve **premio Turing nel 1977**.



ALTRI LINGUAGGI

Da questi linguaggi ne derivarono via via altri, come il LISP per il calcolo simbolico (1959), l'algol (1960), il BASIC (1964). Tali linguaggi(di programmazione strutturata) prevedono

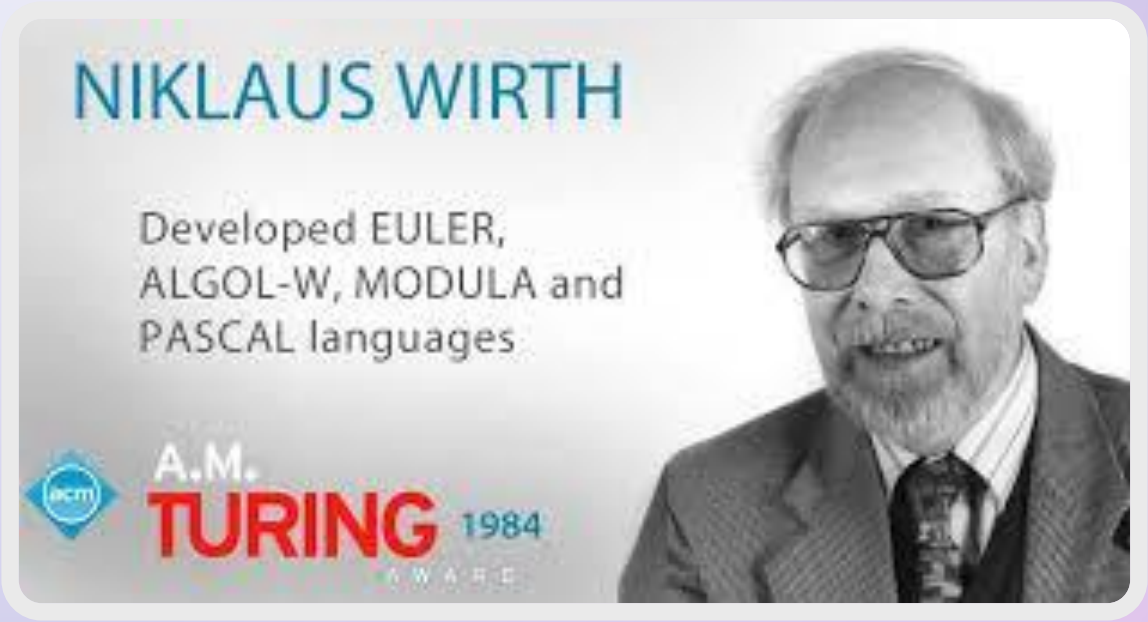
- Istruzione IF (Realizza la SELEZIONE)
- Cicli WHILE e FOR (realizzano l'ITERAZIONE)
- Istruzioni CASE e SWITCH (altri tipi di SELEZIONE)

Nel 1967 nacque il simula, che introdusse per primo il concetto (allora appena abbozzato) di oggetto software.

IL PARADIGMA IMPERATIVO E LA PROGRAMMAZIONE STRUTTURATA

Nel 1970 **Niklaus Wirth** pubblica il Pascal, il primo linguaggio strutturato, a scopo didattico.

Anche lui vincitore nel 1984 del Turing Award in quanto progettista di diversi linguaggi di programmazione. Oltre al Pascal, partecipa alla progettazione dei linguaggi Euler, Algol-W, Modula, Modula-2, Oberon, Oberon-2, and Oberon-07

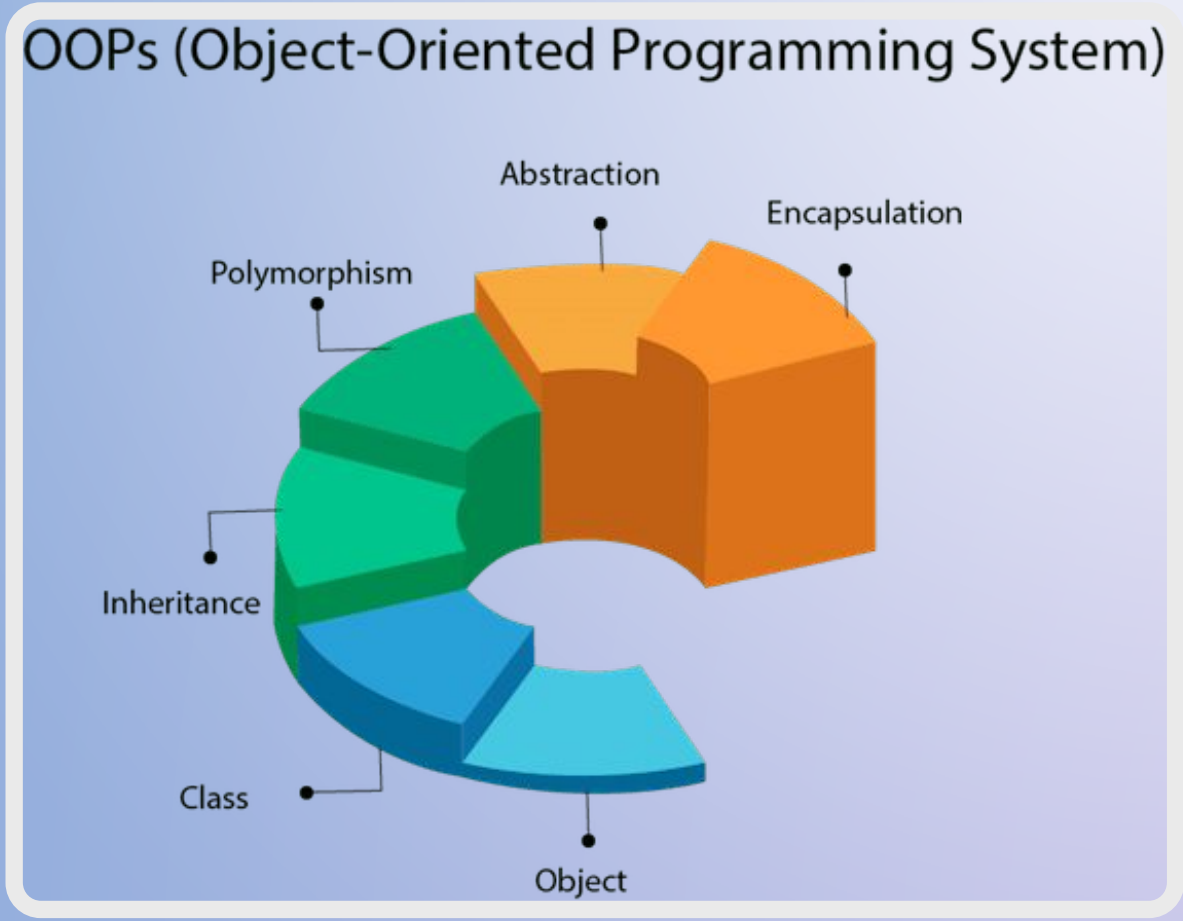


IL LINGUAGGIO C

Nel 1972 dal linguaggio BCPL nascono prima il linguaggio **B** (rapidamente dimenticato) da cui **Dennis Ritchie** (che nel **1983** riceve il **Premio Turing** per aver implementato il sistema operativo UNIX) sviluppò il **C**, che invece fu fin dall'inizio un grande successo.

Insieme a FORTRAN, ALGOL, COBOL e Pascal costituiscono esempi del **paradigma di programmazione imperativo**.

PARADIGMA OBJECT ORIENTED



Nel 1983 vede la luce **smalltalk**, il primo linguaggio realmente e completamente **ad oggetti**, che si ispira al simula e al lisp. Esempi di linguaggi object-oriented odierni sono **Eiffel**, **C++** nel 1986, e successivamente **Delphi**, **JAVA** nel 1995.

nota: contestualmente alla nascita dei primi compilatori **Noam Chomsky** cominciò il suo studio dei linguaggi naturali e alla classificazione dei linguaggi in base alla potenza di calcolo delle grammatiche.

LINGUAGGI SPECIALIZZATI

Sono i linguaggi progettati per applicazioni specifiche, come per esempio lo sviluppo di applicazioni commerciali.

Hanno come obiettivo quello di ridurre al minimo gli sforzi per la programmazione, il tempo per lo sviluppo di un software e il relativo costo.

Ad esempio:

- sql, per query in database
- postscript, per generare report
- mathematica, matlab, per la manipolazione e l'analisi di dati
- ...

PARADIGMA LOGICO

Si risolve un problema usando i vincoli dati al problema stesso piuttosto che usare un l'algoritmo scritto da un programmatore. «cosa» piuttosto che «come»

Usati principalmente in intelligenza artificiale, istruiscono il computer per trovare la soluzione.

Ad esempio:

- linguaggi basati su linguaggi logici come prolog, ops5
- mercury
- …

LINGUAGGI E COMPILATORI

L'evoluzione dei linguaggi di programmazione è profondamente in relazione con quella dei compilatori.

Un compilatore deve tradurre in modo corretto un insieme potenzialmente infinito di programmi scritti in linguaggio sorgente. Il codice prodotto deve essere il più possibile efficiente, sia da un punto di vista del tempo di esecuzione che della memoria utilizzata.

La maggior parte degli utenti vedono un compilatore come una "black box". I compilatori consentono ai programmatori di ignorare i dettagli della programmazione dipendenti dalla macchina.

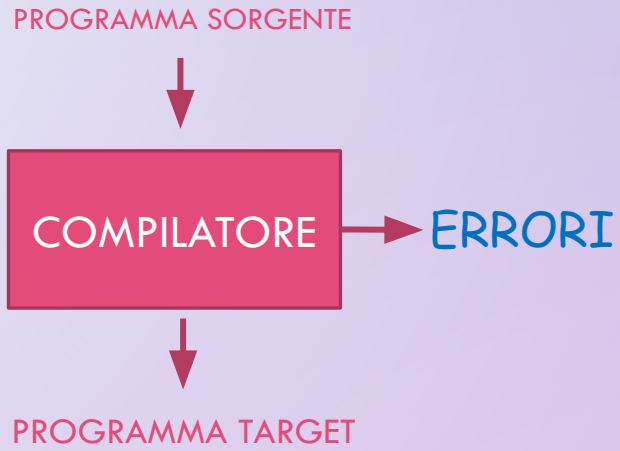
COMPILATORI E INTERPRETI

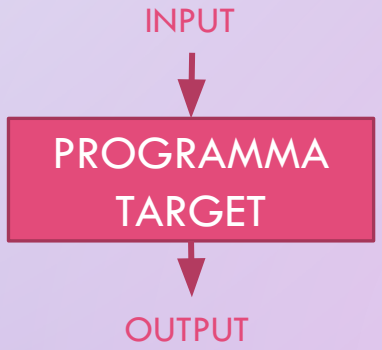
COMPILATORI

Un compilatore è un programma capace di leggere un programma in un linguaggio (linguaggio sorgente) e tradurlo in un programma equivalente in un altro linguaggio (linguaggio destinazione o target). Di fatto esso costituisce un'interfaccia tra architettura e applicazioni

Un ruolo collaterale importante del compilatore consiste nel riportare gli eventuali errori nel programma sorgente incontrati durante il processo di traduzione.

Se il programma target è programma in un linguaggio-macchina eseguibile, allora esso può essere chiamato dall'utente per processare un input e produrre un output.





INTERPRETI

Gli **interpreti** traducono una singola istruzione del programma sorgente nell'istruzione in linguaggio macchina equivalente, la quale viene subito eseguita, per poi passare al processamento dell'istruzione successiva. Invece di produrre il programma target come traduzione, un **interprete** esegue direttamente le singole operazioni specificate nel programma sorgente su input forniti dall'utente.

Il **programma target** prodotto da un compilatore è solitamente 10 volte più veloce di quello prodotto da un interprete.

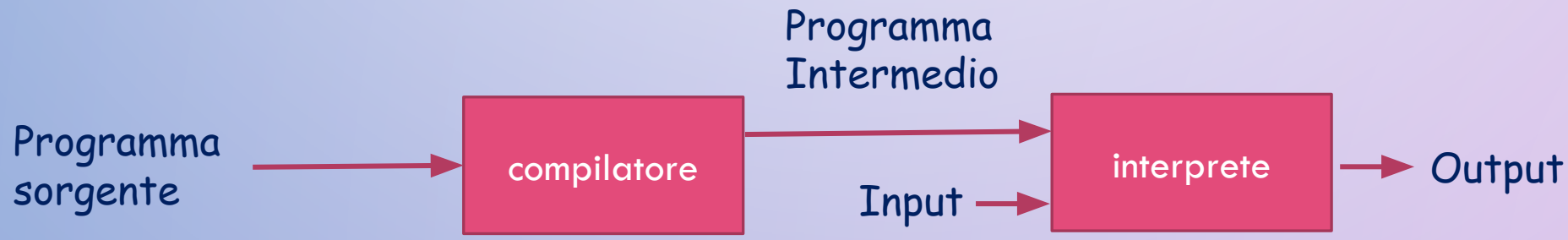
Un interprete comunque può meglio eseguire una diagnostica degli errori poiché esegue il programma sorgente istruzione dopo istruzione.

Esempi di programmi interpretati: javascript, perl, PHP, python, …

COMPILATORI IBRIDI

Esistono anche compilatori ibridi che combinano compilazione e interpretazione. Ad esempio, un programma sorgente in JAVA viene prima compilato e trasformato in linguaggio di bytecode. I bytecode vengono poi interpretati o compilati da una virtual machine.

Alcuni compilatori java, detti compilatori just in time, traducono il bytecode in linguaggio macchina immediatamente prima di doverlo eseguire.



VIRTUAL MACHINE

E' un software che crea un ambiente virtuale in cui l'utente può eseguire alcune applicazioni senza tener conto del sistema operativo sottostante.

Dal momento che esistono macchine virtuali scritte per diverse piattaforme, il programma compilato può "girare" su qualsiasi piattaforma su cui "gira" la macchina virtuale ("write once, run everywhere").

Progenitore delle odierne VM è la "macchina P", cioè il calcolatore astratto per cui vengono compilati i programmi in Pascal nelle prime fasi della compilazione (producendo il cosiddetto P-code)

ESEMPI DI CODICE PER MACCHINA VIRTUALE

- **p-code** prodotto dal Pascal p-compiler per una **virtual stack machine**
- **bytecode** prodotto dai compilatori java per una **virtual java machine** progettata da sun microsystems.
- **Common intermediate language (cil)** della piattaforma .net eseguito dal **common language runtime (clr):** la macchina virtuale .net progettata da microsoft per funzionare con qualsiasi sistema operativo. (esempi di linguaggi .NET: C#, F#, Visual Basic.NET e il Managed C++).

COME LAVORA LA VIRTUAL MACHINE?

Si tratta di un programma che emula un calcolatore. I programmi applicativi vengono scritti in un linguaggio che viene compilato per questo calcolatore immaginario (cioè tradotti nelle sue istruzioni native e non quelle del calcolatore ospite) e, una volta compilati, vengono eseguiti sulla macchina virtuale software, che può agire o come interprete o come compilatore "al volo" (compilazione just in time (jit): traducono il codice intermedio in linguaggio macchina immediatamente prima che venga processato l'input).

Le più recenti implementazioni di virtual machine incorporano un jit compiler.

STRUTTURA DI UN COMPILATORE

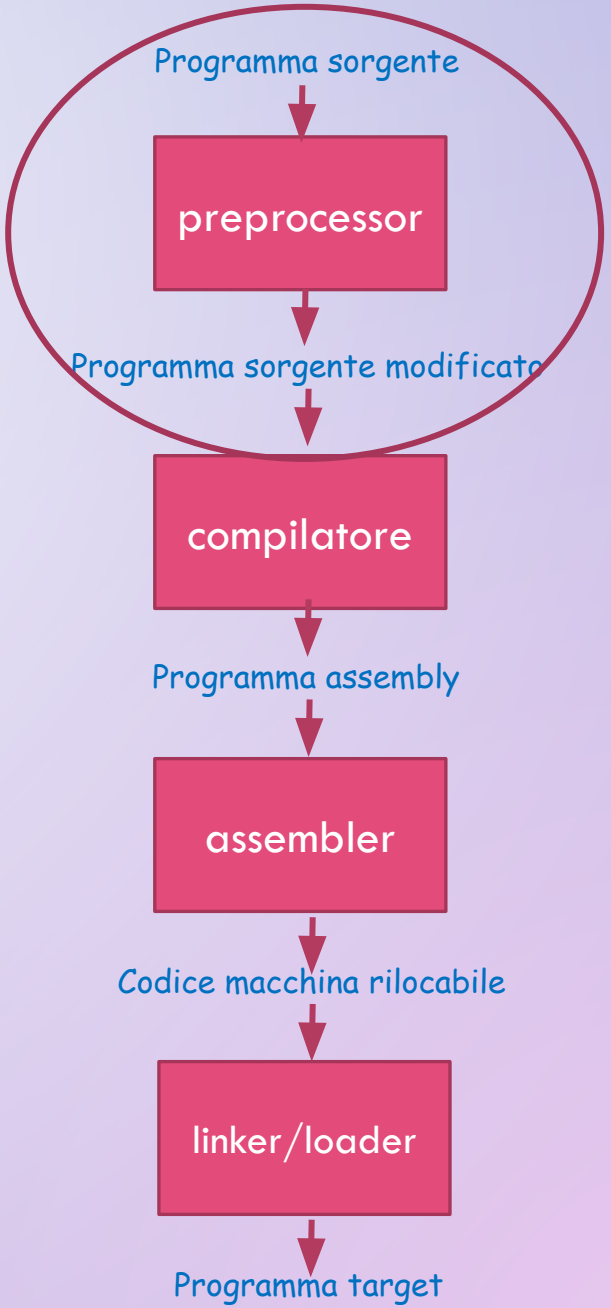
LE FASI DELLA COMPILAZIONE

LINGUAGGI COMPILATI: DAL SORGENTE ALL'ESEGUIBILE

Il programma scritto dal programmatore deve attraversare diverse fasi prima di essere reso eseguibile

Anzitutto un programma sorgente potrebbe essere diviso in moduli memorizzati in file separati. Il compito di mettere assieme il codice sorgente è a volte affidato al **preprocessor** o **precompilatore**. Esso può anche espandere le macro in statement del linguaggio sorgente.

Il programma sorgente modificato è il vero e proprio input del **compilatore**.



PREPROCESSORE

In alcuni casi il codice sorgente subisce una trasformazione (precompilazione) prima di essere compilato. La precompilazione può essere definita come la trasformazione del codice sorgente in un altro.

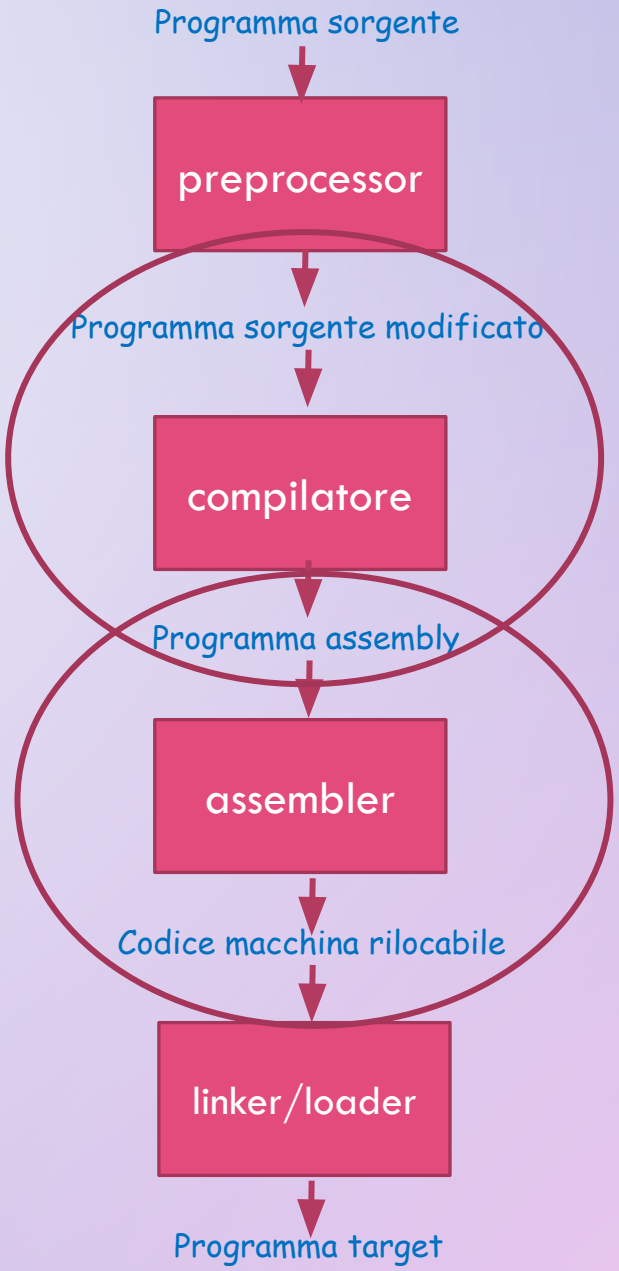
Esempi:

- Traduzione del lisp in C
- Prime versioni del C++ (C con classi)
- Utilizzo del SQL embedded
- Adattare il programma in funzione dell'installazione (compilazione condizionale)
- Espansione delle macro

LINGUAGGI COMPILATI: DAL SORGENTE ALL'ESEGUIBILE

Il **compilatore** ha lo scopo di tradurre il programma sorgente modificato nel corrispondente codice in linguaggio assembly, che da un lato è più facile da produrre rispetto al linguaggio macchina e dall'altro rende più facile l'esecuzione del debug.

Il programma in Assembly ottenuto dal processo di compilazione viene processato dall'**Assembler** per essere tradotto in codice macchina. Esso però non è in grado di determinare in maniera assoluta gli indirizzi di memoria poiché non conosce quella che sarà la situazione della memoria all'atto dell'esecuzione del programma. Il codice generato dall'assemblatore risulta essere quindi un codice intermedio, detto codice macchina rilocabile, in cui i riferimenti agli indirizzi di memoria sono specificati in maniera relativa e non assoluta.

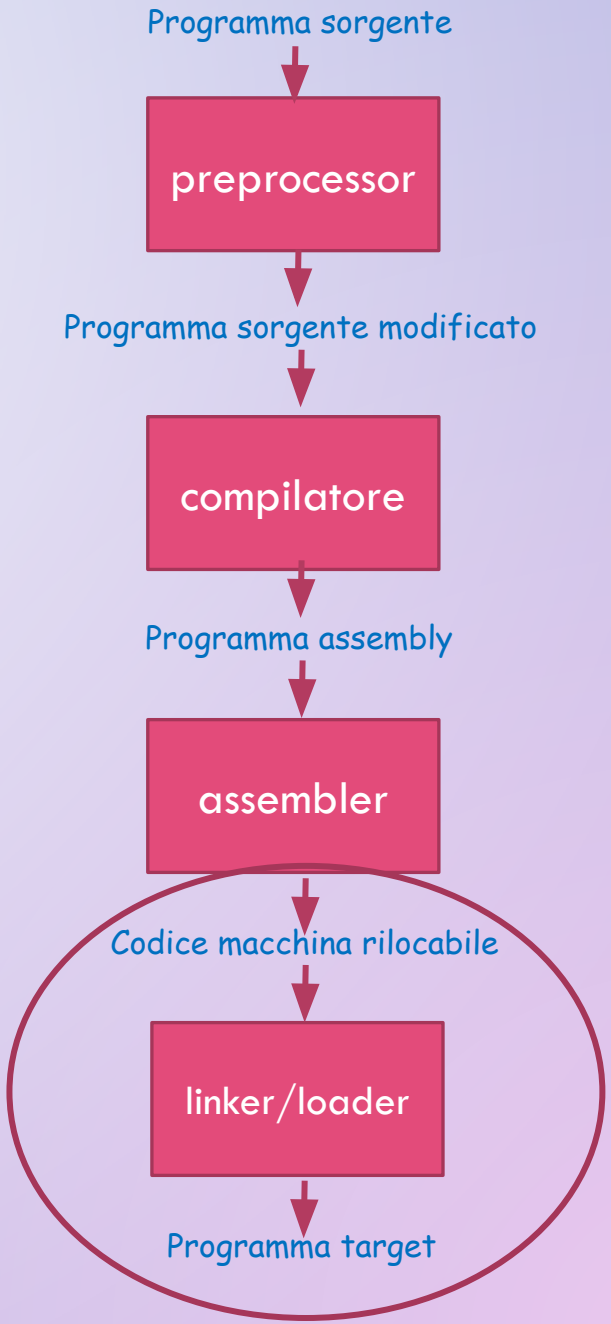


LINGUAGGI COMPILATI: DAL SORGENTE ALL'ESEGUIBILE

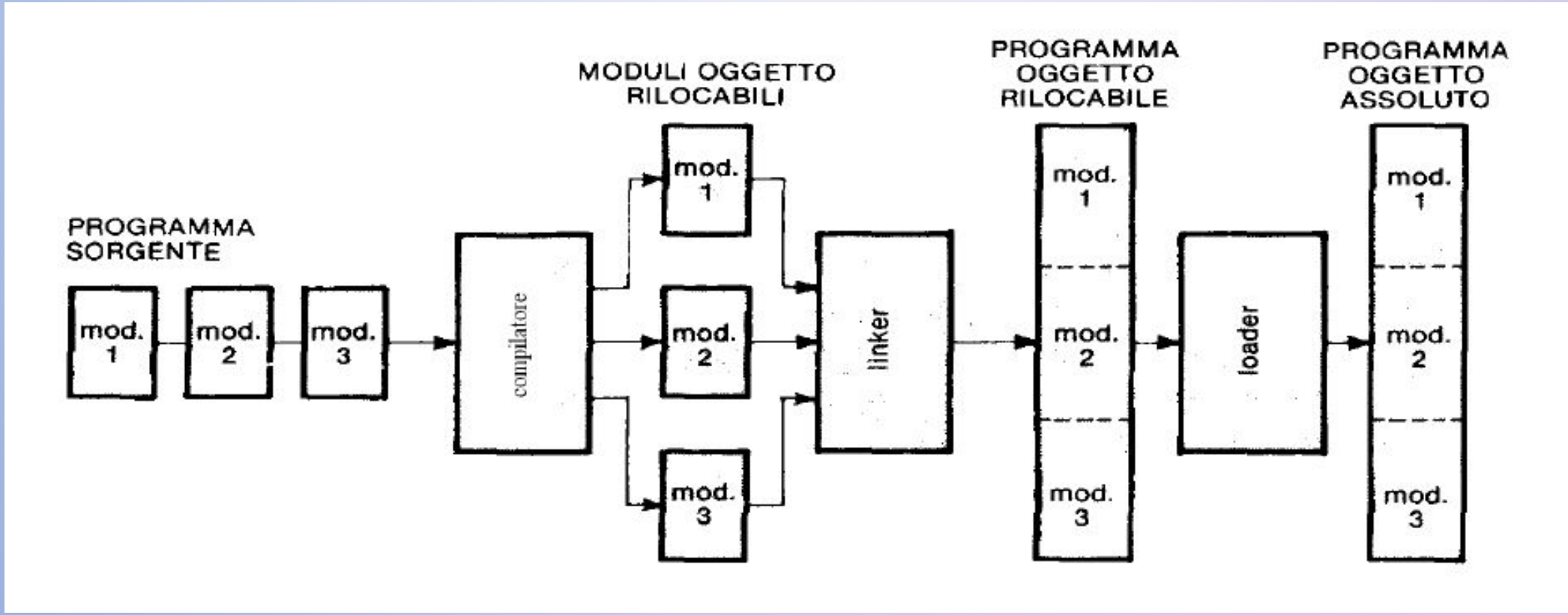
Sarà compito di un altro programma, il loader, quello di "risolvere gli indirizzi" una volta che il programma viene caricato in memoria per essere eseguito. Inoltre programmi lunghi sono spesso compilati a pezzi, così il codice macchina rilocabile deve essere linkato con altri file in codice rilocabile e file di libreria al fine di produrre un codice che giri sulla macchina.

Il **linker** assegna e risolve gli indirizzi di memoria esterna, poiché un codice in un file può riferirsi ad una locazione in un altro file. Produce il Codice Rilocabile unificato.

Il **loader** mette insieme tutti i file oggetto eseguibili in memoria per l'esecuzione, calcolandone gli indirizzi fisici



LINKER E LOADER



LINGUAGGI SIMBOLICI

Ogni compilatore utilizza linguaggi simbolici definiti da:

- alfabeto, insieme dei simboli usati
- lessico, insieme delle parole che formano il linguaggio (vocabolario)
- sintassi, insieme delle regole per la formazione delle frasi (istruzioni)
- semantica, significato da associare ad ogni parola ed istruzione

FASI DELLA COMPILAZIONE

- Analisi lessicale
- Analisi sintattica
- Analisi semantica
- Generazione del codice intermedio
- Ottimizzazione
- Generazione del codice target o codice oggetto

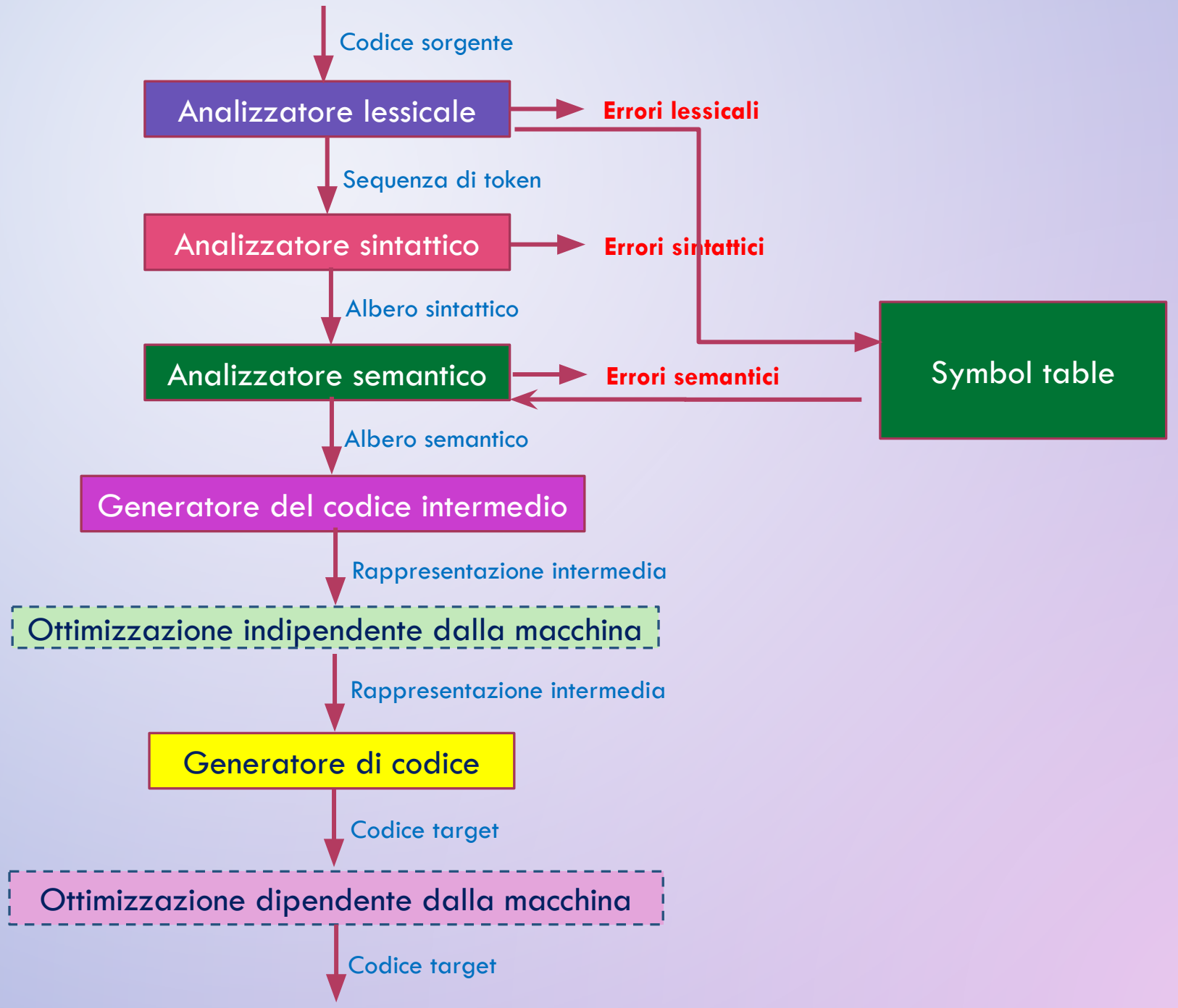
Analisi

Sintesi

ESEMPIO Errore lessicale:
simbolo non riconosciuto

ESEMPIO Errore sintattico:
`If x:=3 then…`

ESEMPIO Errore semantico:
`x:=y+s`
Con `x,y int` e
`s string`



ANALISI E SINTESI

- La prima parte della compilazione, detta **analisi** (Analisi lessicale, Analisi sintattica, Analisi semantica, Generazione del codice intermedio) dipende solo dal tipo di linguaggio che deve essere tradotto viene anche detta **"front end"** del compilatore.
- La seconda parte, detta **sintesi** (che consiste delle fasi di ottimizzazione e creazione del codice target) insieme alle ottimizzazioni dipendenti dalla macchina viene detta **"back end"** del compilatore.