Concepte fundamentale ale limbajelor de programare

Implementarea limbajelor de programare

Curs 04

conf. dr. ing. Ciprian-Bogdan Chirila

Universitatea Politehnica Timisoara
Departamentul de Calculatoare si Tehnologia Informatiei

March 12, 2023

Implementarea limbajelor de programare

- Toate calculatoarele executa programe de nivel scăzut scrise in limbaj mașina
- Pentru a executa programe de nivel înalt sunt utilizate două metode:
  - interpretarea
  - translatarea

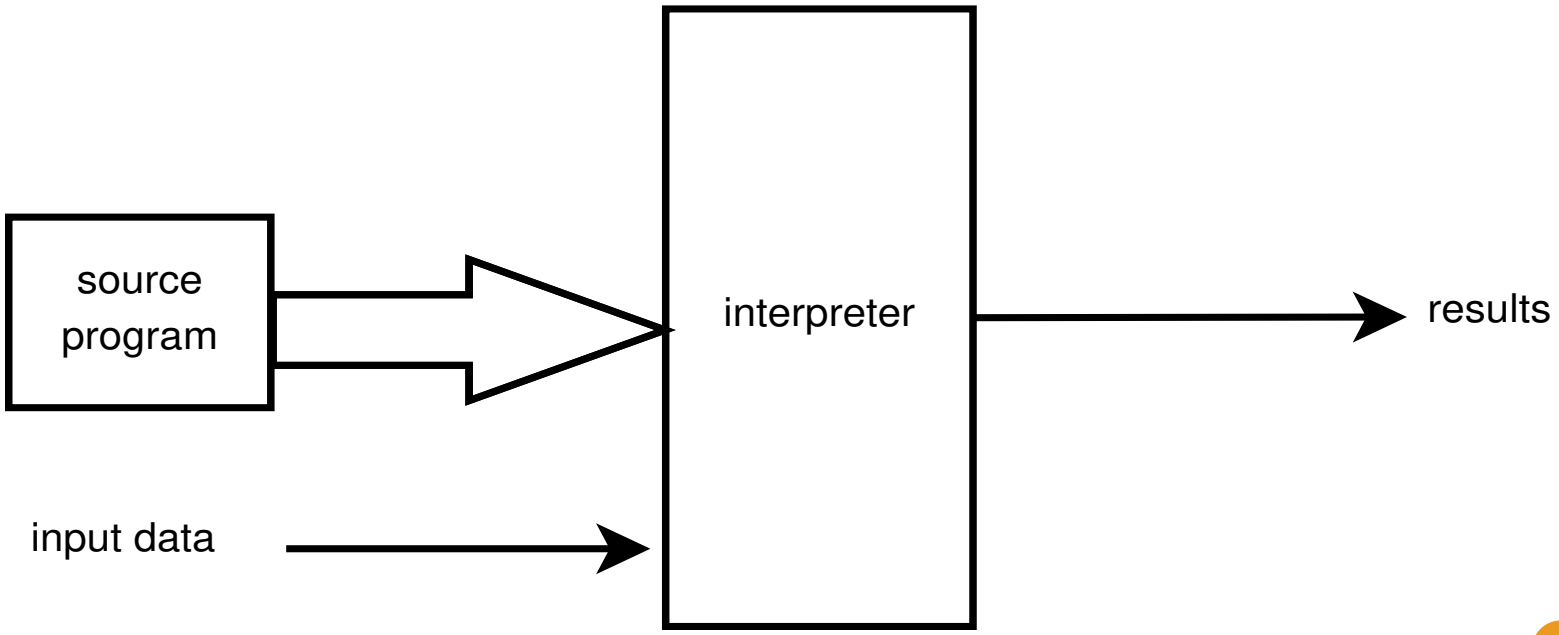
Cuprins

Interpretarea

- Înseamnă executarea directă a instrucțiunilor de nivel înalt
- Fiecare instructiune de nivel înalt este formată dintr-o secvență de instrucțiuni în limbaj mașina
- Execuția programului este realizată de un interpretor
  - Citește instrucțiunile de nivel înalt
  - Decodează aceste instrucțiuni
  - Execută secvența de instrucțiuni mașina

Ciclul de lucru al interpretorului

- Citește instrucțiunea următoare
- Decodează instrucțiunea
- Execută instrucțiunile mașină corespunzătoare



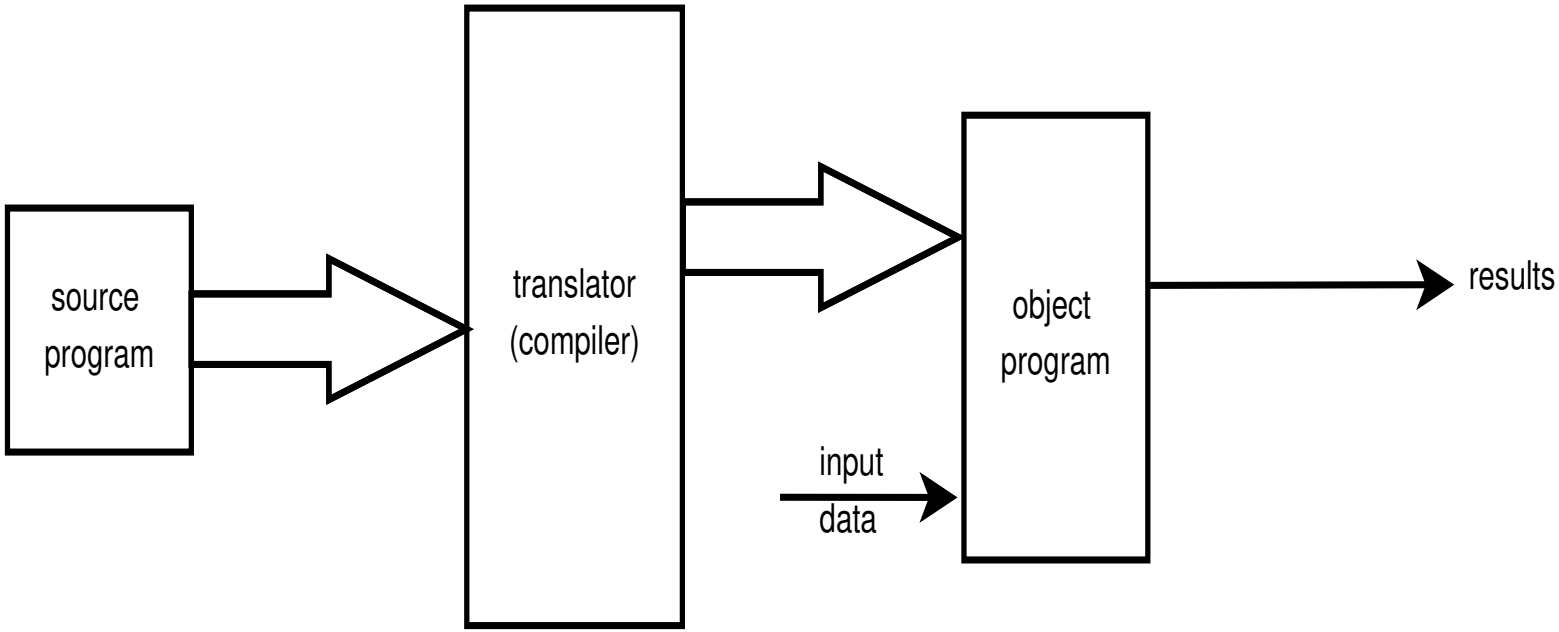
Interpretarea

- Urmăreste ciclul normal al lui von Neuman
- Este
  - o simulare pe un calculator obișnuit
  - a unui calculator imaginar cu limbaj de nivel înalt

Translatarea

- Înainte de execuție programul este translatat
  - Din limbaj de nivel înalt
  - În cod masină
- Procesul este foarte complex
- Este executat in mai mulți pași
- Este realizat de programe specializate
  - Preprocesoare
  - Compilatoare
  - Assembloare
  - Editoare de legături
- Rezultatul este programul obiect ce contine cod masină ce poate fi executat

Implementarea prin translatie



- Compilator -> operatia de compilare

Translatarea cu interpretare

- Se translateaza sursele în program obiect
  - Dar care nu este cod masină
  - Este un cod intermediar pentru o masină abstractă
- Apoi se interpretează codul intermediar
- Metoda este prezentă in Java si in C#

Abordarea Java

- Programul sursă este compilat in bytecode
- Bytecode-ul este o secventă de instrucțiuni pentru masina virtuală Java (Java Virtual Machine - JVM)
- Bytecode-ul poate fi transferat pe orice masină ce are un interpretor
- Advantaje
  - Portabilitate crescută
  - Independentă de platformă
- Dezavantaje
  - Timp crescut la interpretare
- Soluția de compromis
  - compilator JIT (just in time) – din bytecode in cod masină

Abordarea C#

- Rezultatul compilării este un pseudocod numit Microsoft Intermediate Language (MSIL)
- MSIL
  - Este un limbaj de asamblare portabil
  - Are nevoie de Comon Language Runtime (CLR) pentru a fi convertiți in limbaj masină
  - CLR functionează ca un compilator JIT ce convertește codul MSIL in cod masină dupa necesitați
- rapid
- portabil

Cuprins

Compararea timpului de executie

- Programele compilate sunt mai rapide decat cele interpretate
- Interpretarea instrucțiunilor înseamna translatarea instrucțiunilor
- Programul obiect
  - Este compilat o singura data
  - Ruleaza de mai multe ori fara a fi translatat
- Interpretarea devine critică când aplicatia este rulata de mai multe ori

Compararea spațiului de memorie

- Interpretarea ia mai putină memorie
- Compilarea implică inlocuirea fiecărei instrucțiuni de nivel înalt cu o secventa de instrucțiuni masină

Cuprins

Procesul de compilare

- Functionalitățile de baza ale compilatorului
  - Analiza codului sursa
  - Sinteza programului destinație
- Părțile unui compilator
  - Partea de analiză
    - Descompune programul în componente elementare
    - Crează o reprezentare intermediară
- Partea de sinteză
  - Construiește programul destinație pe baza reprezentării intermediare

Structura unui compilator

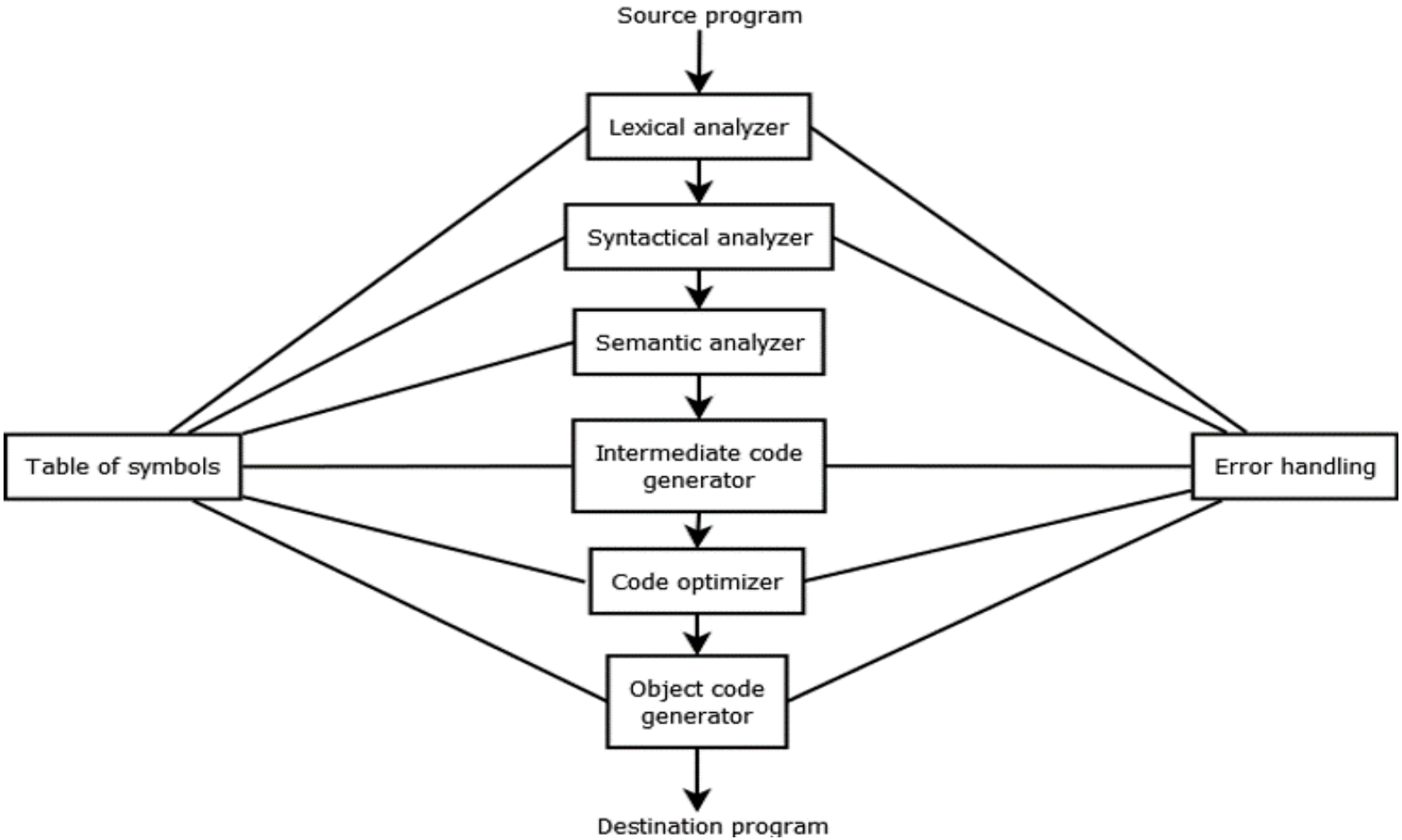


Tabela de simboluri

- Sarcina de baza a compilatorului
  - Să gestioneze identificatorii
  - Să colectioneze informații despre atributele lor
    - Constante
    - Variabile (domeniu, tip)
    - Funcții (număr, tip, ordine, tip de transmisie pentru parametri)

Tabela de simboluri

- Este o structură de date ce menține câte o înregistrare pentru fiecare identificator
- Trebuie sa permită o căutare rapidă a identificatorilor
- Înregistrările se adaugă în timpul analizei lexicale si sintactice
- Și alte informații auxiliare sunt adăugate în tabelă în timpul analizei
- Informațiile sunt utilizate pentru
  - a verifica acțiunile semantice
  - a genera codul obiect corect

Gestionarea erorilor

- Erorile pot fi descoperite încă din prima fază a compilarii
- Se poate reacționa în diferite moduri
  - Se oprește compilarea la prima eroare, se așteaptă corecții și apoi se recompilează de la inceput
  - Se pot gestiona erorile încat să se continue compilarea, să se detecteze și celelalte erori ca apoi să poată fi corectate global
    - Tehnici de revenire din eroare
- Majoritatea erorilor sunt detectate în fazele de analiză sintactică și semantică

Fazele de analiză

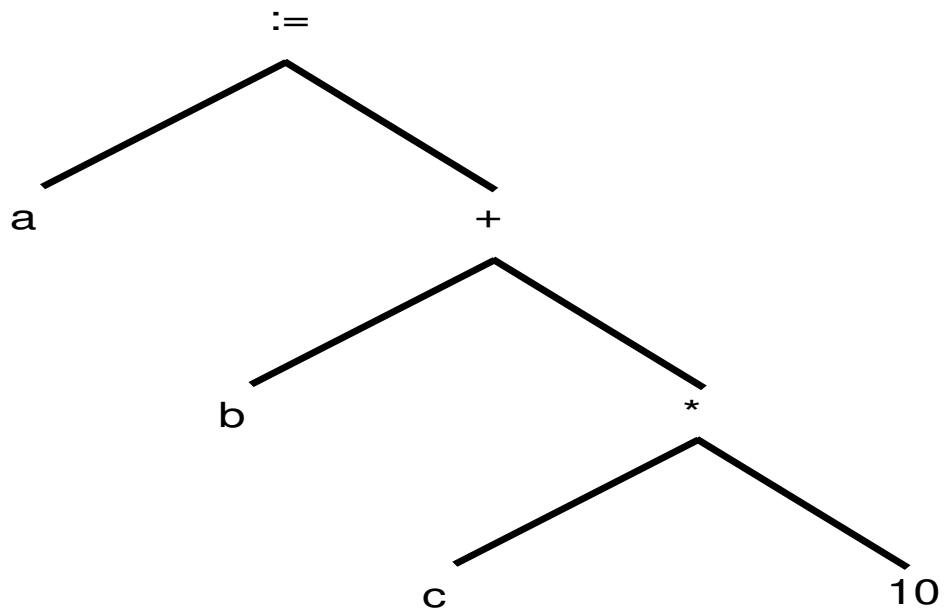
- Analiza lexicală sau liniară
- Analiza sintactică sau ierarhică
- Analiza semantică sau contextuală

Analiza lexicală sau liniară

- Programul sursa
  - Este un șir de caractere
  - Se citește de la stanga la dreapta
  - Se grupeaza în simboluri lexicale – secvențe de caractere cu semantică specifică
- Exemplu: `a:=b+c*10`
  - Identificatori: `a, b, c`
  - Operatori: `:=, +, *`
  - Întregi: `10`
- Spațiile albe sunt ignorate

Analiza sintactică sau ierarhică

- Simbolurile sunt grupate în colecții mai mari
  - Expresii, declarații, instrucțiuni
- Arborele sintactic
  - Fiecare nod reprezintă o operație
  - Nodurile fiu reprezintă argumente



Analiza sintactică sau ierarhică

- Structura ierarhică este exprimată prin reguli recursive
  - Reguli recursive pentru definirea
    - expresiilor
    - instrucțiunilor
- Împarțirea analizelor în lexicală si sintactică este arbitrară
  - Se simplifică procesul de analiză în ansamblu
  - Numerele, șirurile de caractere, identificatorii, punctuația sunt simboluri lexicale
  - Expresiile, instrucțiunile, declarațiile sunt construcții sintactice

Analiza semantică sau de context

- Se fac verificări în legatură cu înțelesul programului
- Se verifică restricții contextuale ale programelor
- Se preiau informații de tip pentru generarea de cod
- Identifică operatorii, operanzii și instrucțiunile utilizând structura ierarhică
- Verificarea de tipuri – verifică daca fiecare operator are operanzii corect setați
  - De exemplu, un număr real nu poate fi utilizat pentru a indexa o tabelă
- Analiza de domeniu – verifică faptul că fiecare indentificator este utilizat în domeniul său de vizibilitate

Fazele de sinteză

- Generarea codului intermediar
- Optimizarea de cod
- Generarea codului obiect

Generarea codului intermediar

- Generarea reprezentării intermediare a codului sursă este realizată după etapele de analiză lexicală și sintactică
- Poate fi vazut ca un program pentru un calculator abstract
- Sunt mai multe forme de reprezentare a codului intermediar
- Codul cu 3 adrese:
  - Arată ca un limbaj de asamblare pentru un calculator
  - Fiecare locatie de memorie joaca rolul unui registru

Codul cu 3 adrese

- Este o secventă de intrucțiuni
- Are cel mult 3 operanzi
- Fiecare instrucțiunie are cel mult un operator împreuna cu atribuirea
- Compilatorul trebuie să decidă ordinea operațiilor bazată pe prioritatea operatorilor
- Pentru a păstra valorile calculate de fiecare instructiune, compilatorul trebuie să genereze variabile temporare
  - Acestea nu au nicio relație cu codul sursa
- Pot fi utilizate instrucțiuni cu mai puțin de 3 operanzi

Optimizarea de cod

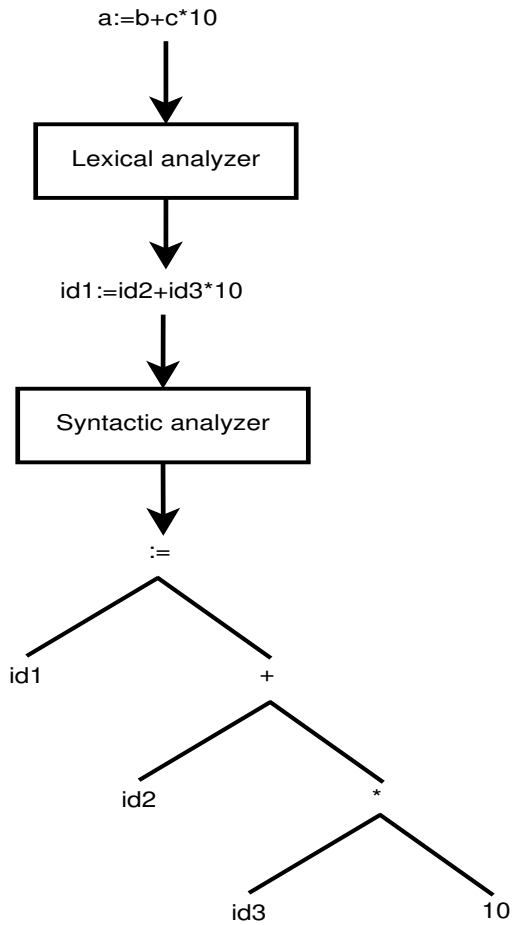
- Scopul este de a optimiza codul intermediar pentru a genera cod masina rapid
- Sunt eliminate
  - Redundanțele
  - Calculele inutile, variabilele
- Compilatoare cu optimizare
  - Timpul consumat cu optimzarea poate fi mare
- Optimizările simple
  - Genereaza cod bun și eficient
  - Nu incetinesc prea mult compilarea

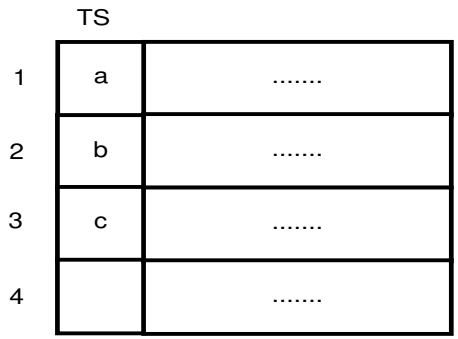
Generarea codului obiect

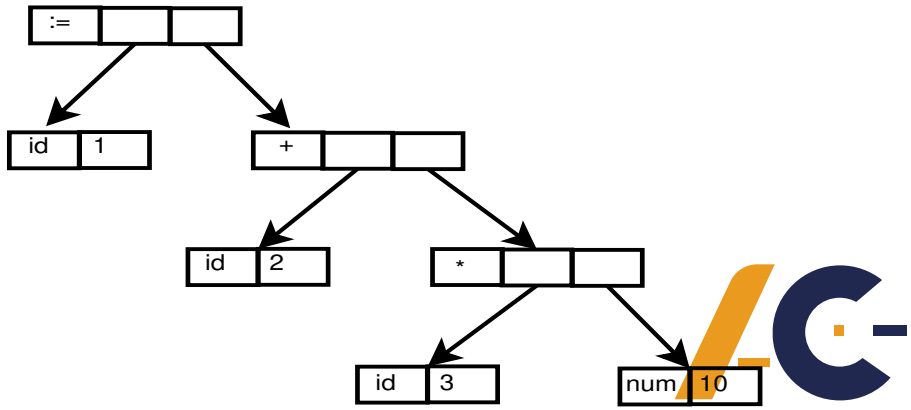
- Reprezintă etapa finală a unui compilator
- Codul obiect generat poate fi
  - Cod masină relocatabil
  - Cod virtual
- Se translatează codul intermediar în cod masină
- Sunt selectate și alocate celule de memorie pentru variabilele de program
- Sunt alese și implementate cele mai bune accese la variabile utilizând facilitați de adresare hardware: indexarea, indirectarea etc.
- Sunt alocați regiștri pentru calcularea și stocarea temporară a rezultatelor intermediare

Compilarea unei instrucțiuni de atribuire

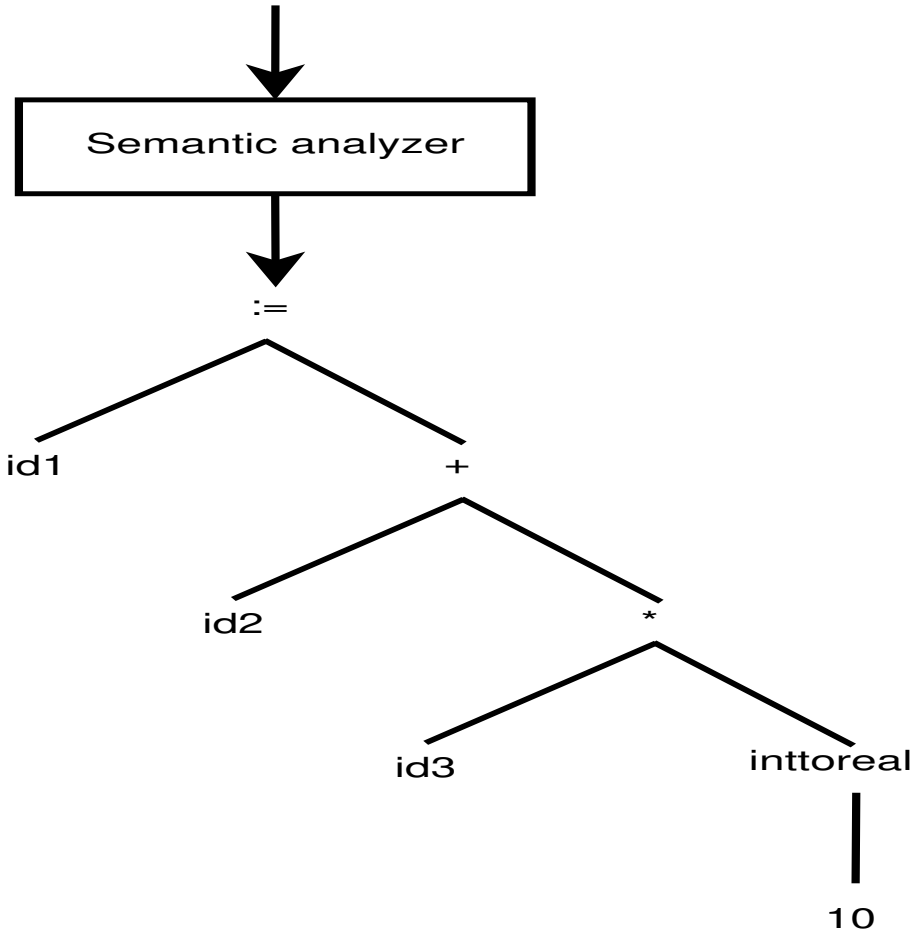
- `a:=b+c*10`
- `a, b, c` – variabile de tip real



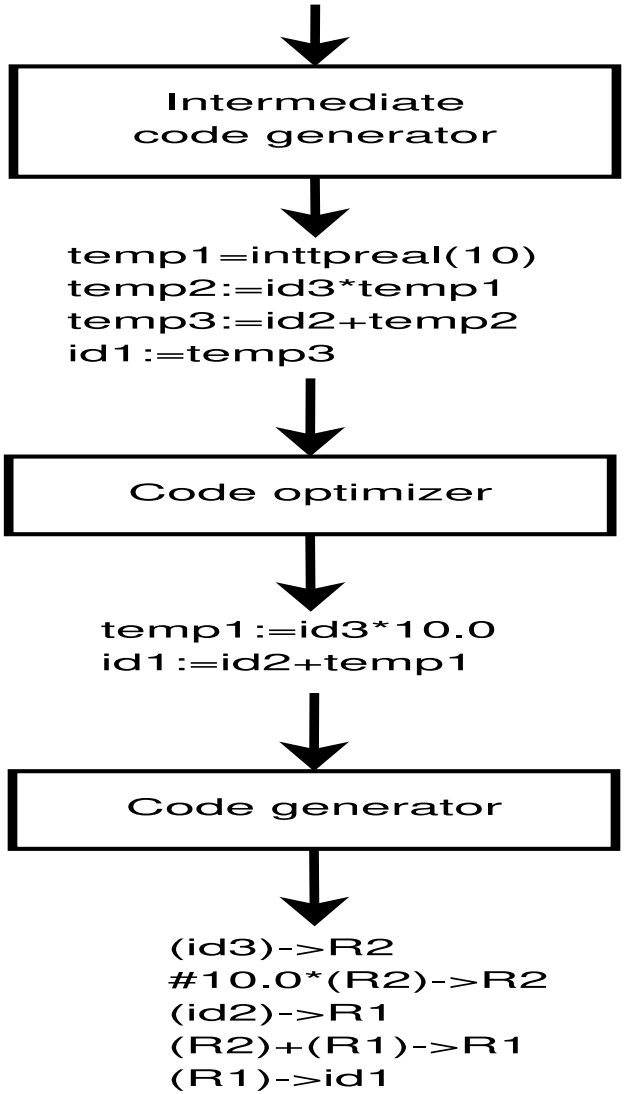




Compilarea unei instrucțiuni de atribuire



Compilarea unei instrucțiuni de atribuire



Bibliography

1. Brian Kernighan, Dennis Ritchie, C Programming Language, second edition, Prentice Hall, 1978.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Horia Ciocarlie – Universul limbajelor de programare, editia 2-a, editura Orizonturi Universitare, Timisoara, 2013.