Laboratorul 5.
Analiză Semantică I. Rezolvarea simbolurilor

1 Tabela de simboluri

În acest laborator ne vom ocupa de partea de rezolvare a simbolurilor pentru limbajul CPLang, alături de verificarea anumitor erori identificate la nivelul de analiză semantică. Vom folosi AST-ul implementat în laboratorul trecut, și, adăugând noi clase pentru simboluri și scope-uri vom implementa noi visitori care vor adnota AST-ul cu aceste simboluri și scope-uri.

Pentru a realiza acest lucru, în funcție de natura limbajului, se pot folosi una sau mai multe parcurgeri. Motivul pentru care o singură parcurgere nu este suficientă este existența de forward references, adică folosirea unei funcții înainte că ea să fie declarată. Astfel, în prima parcurgere, în timp ce găsim un apel de funcție, nu putem garanta la acel moment dacă funcția aceea chiar există sau va fi definită mai târziu.

Clasele de bază implementate în schelet sunt Symbol și Scope:

1.1 Symbol

Clasa Symbol conține detalii despre un simbol. Acesta poate fi ori variabilă, ori funcție. Simbolurile sunt adăugate în scope-uri și ne ajută să ne dăm seama dacă într-un scope avem o variabilă cu același nume definită de mai multe ori.

1.2 Scope

Clasa Scope conține o lista de simboluri și un Scope părinte. La vizitarea nodului Program vom inițializa acest scope cu un DefaultScope, care denotă scope-ul global. Alte exemple de scope-uri sunt funcțiile si clasele, noi tratând doar funcțiile în cadrul acestui laborator.

2 Schelet

Scheletul conține clasele AST-ului din laboratorul trecut, cu o mică modificare: clasa Id conține și un Symbol și un Scope. Astfel, setând (adnotând) în visitor simbolurile aferente, în treceri ulterioare putem obține scope-ul și simbolurile foarte ușor.

3 Cerințe

3.1 Scope-uri pentru funcții

Să se definească simboluri pentru funcții (similar cu IdSymbol, deja implementat), ținând cont de faptul că funcțiile introduc scope-uri pentru parametrii formali. Porniți de la clasa FunctionSymbol care extinde clasa IdSymbol și implementează interfață Scope. Urmăriți TODO 1 în fișierul FunctionSymbol.java.

3.2 Definirea și rezolvarea simbolurilor pentru variabile și funcții

Implementati, în două treceri, definirea și rezolvarea de simboluri pentru variabile globale, parametri formali și funcții.

În prima trecere veți defini toate simbolurile și rezolva referirile la variabile globale și parametri formali. În a două trecere, se vor rezolva referirile la funcții.

Urmăriți comentariile și TODO 2-urile din fișierele DefinitionPassVisitor și ResolutionPassVisitor.

Verificarea acestui exercițiu este dată de faptul că nu obținem NullPointerException la rularea pe fișierul de input manual.txt. Testarea o vom face la exercițiul următor.

Atenție! Limbajul CPLang nu permite forward references la variabile, ci doar la funcții!

3.3 Verificarea erorilor

Din moment ce nu putem testa exercițiul anterior unde totul merge bine, ne propunem să găsim niște erori uzuale de analiză semantică. Acestea verifică de asemenea că AST-ul nostru a fost adnotat corect. Astfel de erori includ:

- variabilă sau funcție nedefinită
- variabilă sau funcție redefinită
- încercarea de apelare pe o variabilă (e.g. Int x; x(1, 2)), sau invers, atribuirea unei funcții (e.g. Int x() ; x = 5)
- existența mai multor parametri formali cu același nume.

Notă: Spațiul de nume global e același pentru variabile și funcții, deci nu pot există variabile globale și funcții cu același nume

Urmăriți comentariile și TODO 3-urile din fișierele Test.java, DefinitionPassVisitor și ResolutionPassVisitor.

După ce ați reușit să obțineți toate erorile pe fișierul cu erori semantice, rulați din nou pe manual.txt și rezolvați erorile nou apărute (ar trebui să apară ca print_float, print_bool și print_int nu sunt definite.). Le puteți defini în scope-ul global, acestea fiind predefinite de limbaj.