情報科学科演習 D
課題 3

基礎工学部情報科学科ソフトウェア科学コース
学籍番号: 09B20001

イグレシウスエドゥアルド

2022 年 12 月 14 日

1　システムの仕様

1. 意味解析器は Checker.run(String) から実行される。
2. プログラムの入力としてはトークンごとに分割されたデータパス（ts ファイル）である。
3. 与えられたデータパスがない場合は”File not found”というエラーメッセージを出力して、プログラムを終了させる。
4. 与えられたデータパスを見つけた場合、
   （a）構文的にかつ意味的に正しい場合、標準出力に”OK”を出力する。
   （b）構文的なエラーがある場合、そのエラーは行の番号と共に、”Syntax error : line { LineNumber } ”というエラーメッセージを標準エラーに出力する。
   （c）意味的なエラーがある場合、そのエラーは行の番号と共に、”Semantic error : line { LineNumber } ”というエラーメッセージを標準エラーに出力する。
   （d）与えられたファイル内に複数の構文的か意味的なエラーがある場合、その最初のエラーのみは出力される。

2　プログラムの方針・設計

本課題のプログラムの設計は全体的に図 1 のようになる。

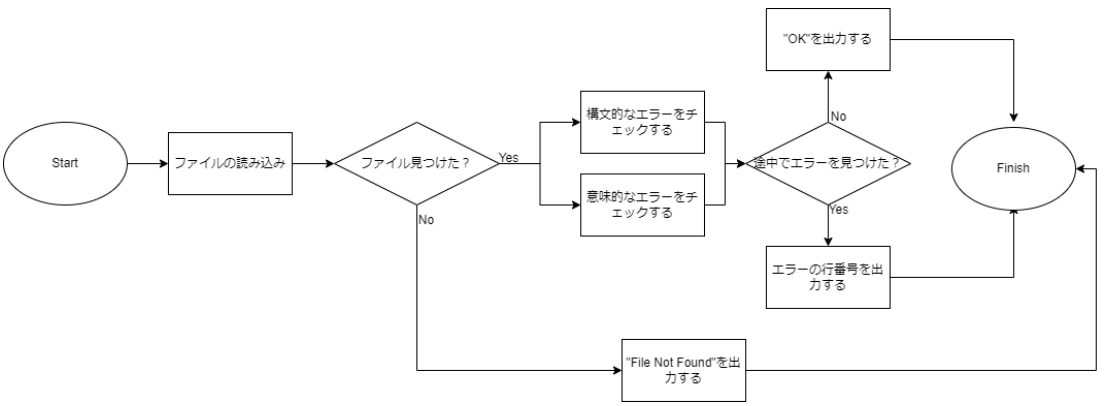

図 1　全体的なプログラムの流れ

本課題では前の課題の続き、つまり、課題 2 のプログラムを追加するので、本レポートではその追加の部分のみを説明する。

2.1　チェッカー

本課題のメイン処理となる。意味的なエラー検出（チェッカー）のアルゴリズムは図 2 のようになる。

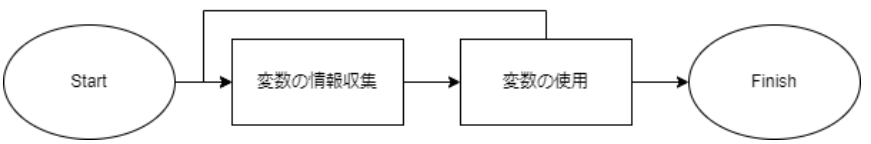

図 2　チェッカーの流れ

チェッカーは全体的に二つの大処理に分けられて、それぞれは以下に説明される。

1. 変数の情報収集、変数の情報収集は変数の定義部分から行われる処理である。すべての変数は表に格納して、作られた表は二つにわけて、それぞれ以下のようになる。
   - 変数表、変数表は変数の変数名と型と代入状態と参照状態（4 章参照）を格納する。変数表はスコープを考えて、スコープごとに（2.2 章参照）、それぞれのページに変数表を格納する。変数表は「変数宣言」と「仮パラ

メータ」のところで更新される。

- 関数表、関数表は関数の関数名と関数のパラメータを格納する。パラメータは変数と同じく見なす。関数表はスコープを考えない。関数表は「仮パラメータ」の構文のところでのみ更新される。

以上を実現するために、パーサーで作った関数を「オブジェクト化」して、定義部分にある返却値を持たない関数を対応される構文に応じて「型」を返す。実際には以下のように処理を変えた。

- 変数宣言、
  - 標準型 → そのトークンをそのまま返す。
  - 配列型 → 配列型の最後の構文の「標準型」を確認したときに、その型をそのまま返す。ただし、標準型と区別できるように表に登録する前に、文字列の先頭に”Array of”を追加する。
  - 型 → 標準型と配列型があるが、配列型か標準型かという二つの選択肢になるが、両方とも同じ文字列型で、区別されたので、返却値をそのまま返す。
  - 変数名 → 構文的にエラーがない場合、そのトークンをそのまま返す。
  - 変数の並び → 毎回変数名を確認した後、その変数名をリストに格納する。構文の終わりでそのリストを返す。
  - 変数宣言の並び →「変数名の並び」からの変数名リストと「型」からの一つの型に対して、リストにある変数名にすべて同じ「型」を与えて、すべての変数を現スコープの変数表に登録する。
  - 変数宣言 → 変数宣言がある場合、その現スコープの変数表が得られた後、一番前のページにその変数表を置いておく。
- 仮パラメータ、
  - 仮パラメータ名 →「変数名」の構文と同じく仮パラメータ名の構文を確認した後、そのトークンをそのまま返す。
  - 仮パラメータ名の並び →「変数名の並び」の構文と同じく毎回仮パラメータ名を確認した後、そのパラメータ名をリストに登録して、最後に、そのリストを返す。
  - 仮パラメータの並び →「変数宣言の並び」の構文と同じく毎回パラメータのリストとその型の情報が得られた後、すべてのパラメータに同じ「型」を与えて、現スコープの変数表を更新する。また、同時に関数表のために、そのパラメータのリストを返す。
  - 仮パラメータ → 得られた仮パラメータのリストをそのまま返す。
  - 手続き名 → 構文的にチェックした後、そのまま手続き名を返す。
  - 副プログラム先頭 → 手続き名とそれぞれのパラメータの情報が得られた後、関数表のページにそのデータを格納する（関数表の更新）。
  - 副プログラム宣言 → 副プログラム先頭には副プログラム先頭と変数宣言つまり、現スコープには変数宣言だけではなく、手続きのパラメータもあるので、その二つを同じスコープページにして、「変数宣言」の構文を確認する前に、現スコープのリストを初期化せずに、確認した後、最前ページにその変数表を登録する。
  - 副プログラム宣言群 → 一つの副プログラム宣言を確認した後、最新の変数表（その副プログラムの変数表）を解除する 。

また、どちらの表にせよ、表に登録する前に、重複宣言された変数をチェックする。すでに表に存在する場合は意味的なエラーを出力するが、ない場合は変数表に登録する。

2. 変数の使用、変数の使用は変数の情報収集から得られた関数表と変数表を使う。また、変数の使用部分になる構文は以下に述べる。
   - 変数表は「式」という構文に使う。
   - 関数表は「手続き呼出し文」という構文に使う。

   実際には以下のように構文に新たな処理を加える。
   - 変数表
     - 純変数 → 得られた変数名を変数表にチェックして、その変数の型を返す。
     - 添え字付き変数 → 得られた変数名から変数表にチェックして、その配列の型のみを返す。つまり、”Array of ”を消して、型の部分のみを返す。
     - 変数 → 純変数か添え字付き変数から返された型をそのまま返す。
     - 定数 → 以下のようになる。
       * 符号なし整数の場合、”integer”を返す。

＊ 文字列の場合、
・ 長さ 3（2 個の’ を含み）の場合、”char”を変えす。
・ それ以外の場合は”Array of char”を返す。
＊ ”false”又は”true”の場合、”boolean”を返す。
– 因子 → 変数や定数の構文からの型をそのまま返して、「”not” 因子」という構文の場合は因子を再帰的に型を求めて、型が”boolean”ではないと意味的なエラーを出力する。そうでなければならば、”boolean”を返す。また、”(”式”)”の場合は再帰的に「式」から返された型をそのまま返す。
– 項と単純式と式 → 最初の被演算子の型を登録して、演算子と二つ目の被演算子がある場合、その演算子によって（指導書の 7.2.10 の表に沿って)、両方の被演算子の型をチェックして結果の型を返す。
– 式の並び → 各々の式の型をリストに入れていき、そのリストを返す。
– if 文と while 文 →”if”や”while”トークの後の「式」の型を確認する。”boolean”ではない場合、意味的なエラーを出力して、”boolean”である場合、エラー検出を続く。
– 左辺 → チェックされた変数の型をそのまま返す。
– 代入文 → 左辺と右辺の式の型をチェックして、一致する場合は処理を続くが、一致しない場合は処理を意味的なエラーを出力する。

- 関数表
  – 手続き名 → 手続き名をそのまま返す。
  – 手続き呼出し文 →「手続き名」の構文からの手続き名を使い、関数表の関数名を検索する。その関数を関数表から見つけた場合、「式の並び」の構文からの型のリストを使って、順番に関数表に登録された関数のパラメータがユーザー側から与えられたパラメータの型と一致するかどうかを確認する。最後まで一致するかつリストのサイズが一致するならば、処理を続くが、途中で一致しない場合は意味的なエラーを出力する。

2.2 ページ（変数表リスト）の管理

ページのイメージは関数表のリストのイメージと同じく、図 3 のようになる。

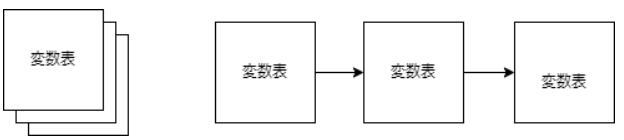

図 3　変数表のページのイメージ

また、新たなページを挿入したいときに、一番後ろではなく、一番前に挿入する。解除も同じく、一番後ろからではなく、一番前から解除する。ページの挿入と解除は図 4 と図 5 に示される。

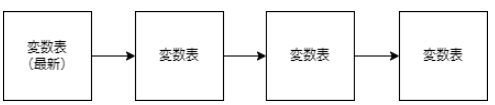

図 4　ページ列の挿入イメージ

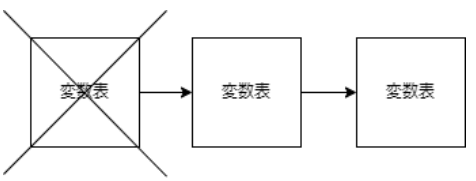

図 5　ページ列の解除イメージ

3 実装プログラム

3.1 意味的なエラーを見つけた対策

意味的なエラーを見つけた場合、system.exit() を使って、プログラムを強制終了させるのは危険なものであるので、本プログラムでは、それを使わずに新しい例外クラスを定義する。例外クラスは図 6 に示される。

意味的なエラーは構文的なエラーと同様、エラーを見つけたところで、現トークンの行番号を引数としてそのオブジェ

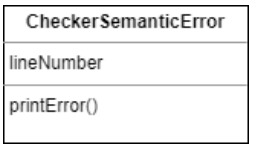

図 6　意味的なエラーのオブジェクト

クトに渡して、構文検出の処理から脱出した後、その行番号をエラーメッセージと共に出力する。

3.2　チェッカー

本課題では構文解析器のプログラムの続きとなり、構文解析器は構文ごとに関数に分割するので、2.1 章に示された構文の型や計算結果の型は関数の返却値としてみなす。ただし、変数型や関数型のオブジェクトは Java にはないので、そのオブジェクトを新しく作成した。変数型は図 7 に示される。関数型は図 8 に示される。

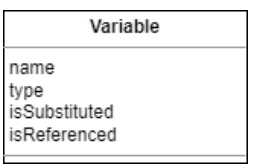

図 7　Variable のオブジェクト

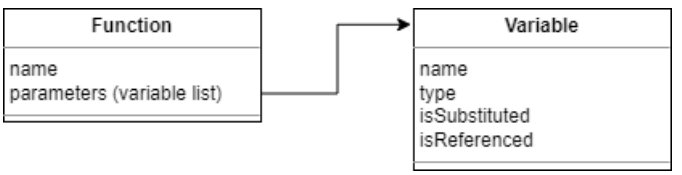

図 8　関数のオブジェクト

その後は、処理に沿ってかつ必要に応じて、新しい関数を定義する。

4　工夫点・拡張

本プログラムの拡張として定義された変数に対して参照状態と代入状態を加えて、未代入や未参照変数を警告として出力する。基本的には変数の定義部分で変数の参照と代入を”FALSE”にして、変数の使用のところでその参照と代入を場合によって（以下にに説明される）”TRUE”にする。最後に、あるスコープやプログラム全体からぬける前に現在のスコープの変数の代入と参照状態を確認して、未代入や未参照の変数は警告として出力する。

- 変数の参照、変数の参照について「左辺」の構文で変数名を検出するときにだけその変数の参照状態を”TRUE”にする。
- 変数の代入、変数の代入について、パラメータの変数は直接に代入状態を”TRUE”にするが、その他の変数は「因子」と「変数の並び」の「変数」構文を検出するときにだけ”TRUE”にする。

ただし、単体テストがそういう警告の場合を考慮しないので、以上の拡張はテキストファイルに保存する。（パス ： $enshud\backslash src\backslash main\backslash java\backslash enshud\backslash s3\backslash checker\backslash CheckPoint(kakucho).txt$）

また、本課題は多い行数のプログラムからプログラムを訂正したり、追加したりするので、プログラムを変える途中でエラーやバッグが発生されたときに、元にもどすために、複数のチェックポイントのテキストファイルを用意して、プログラムを一回訂正した後、そのファイルに保存する。それでデバッグもしやすくなる。

5　感想

本課題は前の課題の続きとなるので、そのプログラムの枠はすでにある。意味的なエラー検出の関数だけを構文解析器のどこかの関数を追加する。本課題もプログラムに書く前に、プログラムの方針を先に作ってみたので、時間がかからなくなった。今後もそのようなやり方でプログラムを作成してみたいと思う。最後の課題も今までの課題より難しそうなので、方針があれば、プログラムがより簡単に作られると思う。