情報科学演習Ｄ

第3回レポート課題

提出年月日：平成30年10月30日
電子メール：u839996e@ecs.osaka-u.ac.jp
学籍番号　：09B16084
氏名　　：山野広大

1　課題 3 の目的

Pascal 風言語で記述されたプログラムが構文的に正しいかどうか判定すると共に，変数等の宣言や有効範囲の誤りや，演算子と被演算子の間や仮パラメータと実パラメータの間などでの型の不整合といった，意味的な誤りを発見するプログラムの作成。

2　プログラムの動作原理

2.1　プログラムで利用したデータの説明

ここでは本プログラムで使用したデータについての説明を行う。
課題 2 で作成したデータはそのまま使用したためここでは課題 3 で追加したもののみ記載する。

名前	型	用途
originalNameList	List<String>	入力されたソースコード中の名前を保存する
parameterList	List<Variable>	仮パラメータを保存する
globalVariableList	List<Variable>	グローバル変数を保存する
localVariableList	List<Variable>	ローカル変数を保存する
procedureNameList	List<String>	手続き名を保存する
errorFlag	int	発生したエラーが構文的か意味的かを保存する
programName	String	プログラム名を保存する
variable	Variable	変数の名前、型、配列型かどうかを保存する

次に Variable クラスの変数の説明を行う。

名前	型	用途
name	String	変数名を保存する
type	int	変数の型を保存する。int 型なのは型の ID を保存するからである (boolean:3,char:4,integer:11)
isArray	boolean	配列型かどうかを保存する

2.2　プログラムの流れ

1. 入力用のファイルを作成する
2. 入力ファイルを読み込みモードでオープンする　※ 1
3. 入力ファイルを 1 行読み込み、その行のトークンの ID、行数を各リストに追加する
4. 指導書に書かれている各構文要素について構文的な誤り、意味的な誤りがないかチェックする　※ 2
5. 標準出力に”OK” と出力する

※1　入力ファイルが見つからない場合は標準エラーに”File not found” と出力し、終了する
※2　構文的な誤りを発見した場合は”Syntax error: line”という文字列と共に最初に発見した誤りの行数番号を標準エラーに出力する。意味的な誤りを発見した場合は”Semantic error: line”という文字列と共に最初に発見した誤りの行数番号を標準エラーに出力する。

3　プログラムの実装方針

課題 2 で作成した構文的な誤りを発見するプログラムに意味的な誤りを発見するプログラムを追加する形で作成した。

3.1　スコープや変数をどう管理したか

3.1.1　変数の種類

変数の種類は 3 つに分類した。それぞれについて説明する。

- グローバル変数
  ブロック内の変数宣言で宣言された変数。以降の副プログラム、複合文のどこからでも参照できる。同一の名前を複数宣言してはいけない。ブロック内で変数宣言メソッドを呼び出す場合は引数として 1 を与える。

- ローカル変数
  副プログラム宣言内の変数宣言で宣言された変数。ローカル変数は、その副プログラムが駆動される毎に作り出され、駆動が終了する時点で破棄される。同一の名前を複数宣言すること、仮パラメータと同じ名前で宣言すること、宣言中のて手続き名と同一の名前で宣言することは禁止されている。副プログラム宣言内で変数宣言メソッドを呼び出す場合は引数として 2 を与える。

- 仮パラメータ
  副プログラム宣言内のパラメータで宣言された変数。ローカル変数同様その副プログラムが駆動される毎に作り出され、駆動が終了する時点で破棄される。同一の名前を複数宣言すること、ローカル変数と同じ名前で宣言すること、宣言中のて手続き名と同一の名前で宣言すること、配列型で宣言することは禁止されている。

3.1.2　変数宣言メソッドの説明

初めにグローバル変数、ローカル変数を宣言するメソッドの説明を行う。
まず引数として与えられた数値 (1 または 2) によってその宣言がグローバル変数なのかローカル変数なのかを判定し、分岐する。
次に for 文を使って宣言しようとしている変数の名前が既に使用されていないか調べる。同一名で宣言されていた場合は意味的なエラーとしてエラー処理を行う。
同一の名前が宣言されていない場合、Variable クラスのオブジェクトを作成し、name に名前をセットする。変数宣言は” 変数名の並び”、” 型” の順で行われるため変数名を読んだ時点では型がわからないため type には-1 をセットしておく。

その後作成したオブジェクトをグローバル変数リストまたはローカル変数リストに追加する。
変数名の並びの後、型を読むときに type の値が-1 のオブジェクトを、読み込んだ型で置き換えることにより変数オブジェクトの作成を行った。また、読んだ型が配列型だった場合は対象のオブジェクトの isArray に true をセットする。
次に仮パラメータの宣言メソッドの説明を行う。指導書内の EBNF では変数宣言と分けてあったため、メソッドとしても分けて作成したがメソッドの内容は型宣言で配列型だった場合にエラー処理を行うこと以外は変数宣言メソッドと同じである。

3.1.3　手続き宣言の説明

副プログラム宣言のメソッドが呼び出されると宣言された手続き名が手続き名リストに追加される。手続き呼び出し文のメソッドではこのリストを参照するため、宣言されていない手続きを呼び出すことはできなくなっている。

3.1.4　スコープ管理の説明

スコープ管理は副プログラム宣言群のメソッドが呼び出されるごとにローカル変数リストと仮パラメータリストを初期化することで実装している。それによって 1 つの副プログラムが終了し、新しい副プログラムが宣言されるごとにローカル変数と仮パラメータが作り出され、駆動が終了する時点で破棄される。

3.2　式の型をどのようにチェックしたか

式の型は式、式から呼び出される単純式、単純式から呼び出される項、項から呼び出される因子の各メソッドを int 型にして、返り値として各メソッドの型を返すことで実装した。式、単純式、項は演算子が現れる可能性がある。演算子が現れたら演算子の前後の返り値を比較し、型が異なった場合は意味的誤りとしてエラー処理を行う。また、式の演算子は関係演算子なので演算子が出現した式は boolean 型を表す 3 を返す。

3.3　エラー処理の説明

エラーは構文的な誤りを発見した場合は syntaxError、意味的な誤りを発見した場合は semanticError のメソッドを呼び出す。syntaxError では errorFlag に 2、semanticError では errorFlag に 1 を代入した後、RuntimeException を作成する。そして Run 内の catch 文で RuntimeException を受け取ったら errorFlag の値に応じてエラーメッセージを出力し、プログラムを終了する。

4　指導書にない意味的な誤りの判定

私は指導書にない意味的な誤りとして、配列型の変数が宣言されたときの添字の判定を行った。具体的には添字の最小値、最大値がどちらも負の数でないか、最大値が最小値以下でないかである(最大値が最小値と一致する場合は配列型を使用する意味がないためエラーとして扱った)。

5　まとめ

課題 3 では Pascal 風言語で記述されたプログラムが構文的に正しいかどうか判定すると共に，変数等の宣言や有効範囲の誤りや，演算子と被演算子の間や仮パラメータと実パラメータの間などでの型の不整合といった，意味的な誤りを発見するプログラムの作成を行った。この課題はスコープや変数の管理、式の型のチェックという 2 種類の課題に分けられる。スコープや変数の管理については変数リストや手続き名リストの作成、利用により実装し、式の型のチェックについては式、単純式、項、因子のメソッドを int 型にして、各メソッドの型を返り値として返すことでチェック可能にした。

6　感想

私ははじめ変数型のリストを作成するのではなく、変数名リスト、変数の型リストなど、変数の持つ性質を複数のリストに分けて管理していた。しかしこのやり方だと可読性が低くなり、デバッグの時に少々苦労した。そのため変数型のクラスを作成し、そのリストを作成することで 1 つの変数の性質を 1 つのリストで管理することが可能になったため可読性が上がり、非常に効果的であった。