情報科学演習Ｄ　課題3レポート

氏名　山久保孝亮
所属　大阪大学基礎工学部情報科学科ソフトウェア科学コース
メールアドレス　u327468b@ecs.osaka-u.ac.jp
学籍番号　09B22084

提出日　2024 年 12 月 12 日
担当教員　桝井晃基　松本真佑

1　システムの仕様

課題 3 の外部仕様は以下のようになる．

- 第一引数で指定された ts ファイルを読み込んで意味解析を行い，意味的に正しい場合は文字列”OK”を，正しくない場合は文字列”Semantic error: line”に続いて対応する行番号を返す．
- 複数の意味的誤りが含まれる場合では，最初に見つけた誤りのみを出力する．
- 意味解析よりも先に構文解析を適用し，構文エラーを見つけると文字列 ”Syntax error: line ”に続いて対応する行番号を返す．
- 入力ファイルが見つからない場合は文字列 ”File not Found ”を返す．

2　課題達成の方針と設計

課題 3 のテストケースをパスするために実装した機能は以下の通りである．

1. 変数が重複して定義されない．(二重定義)
2. 未定義の変数及び関数を参照しない．
3. 型における制約を守る．
4. 代入文の左辺に配列型の変数名を使用しない．

課題 3 では課題 2 の parser.java のプログラムの一部を変更して実装した．上記の機能を満足するために変更及び追加した方針は以下の通りである．

- 変数，関数を記憶するための表を作成する．今回作成した表は言語処理工学 A の授業スライドを参考にして作成した．[1] この表により変数の二重定義，未定義の変数及び手続きの参照を防止できる．それぞれの表の構成要素は以下の表 1 の通りである．

変数表	変数名	変数の型	変数のサイズ
関数表	関数名	引数の型	null

表 1: それぞれの表の構成要素

  null は要素が存在しないことを，つまり関数表には二つの要素しか存在しないことを表す．変数のサイズには配列の際は要素数を，配列でなければ 1 を格納する．

- 変数の宣言において，指導書よりプログラムの宣言と手続きの宣言で処理を分ける必要がある．[2] したがって，変数表はグローバル変数用のものとローカル変数用のものをそれぞれ用意した．これにより，グローバル変数を宣言する処理の際は global_variable_table を，ローカル変数を宣言する処理の際は local_variable_table を参照することによって要件を満足することができた．
- 式，単純式，項，因子，定数を判定するメソッドが型を返すようにした．課題 2 では構文定義を判定するメソッドはすべて void 型であったが，前述のメソッドが ”integer ”, ”char ”, ”boolean ”のいずれかを String 型で返すようにした．これにより，各被演算子の型を把握することができるので，型の整合性がとれているのか，特定の演算子に適した型が使用されているのかを判定できるようになった．

3　実装プログラム

今回実装したプログラムは以下の通りである．

3.1　表の作成

2 で記述したように，変数の情報を記録しておくために表を作成した．それぞれの表に対応する variable,function というクラスを作成し，それぞれグローバル変数のリストとしてどのメソッドからでも参照できるようにした．それぞれのクラス内の変数は表 1 に対応しており，それぞれのクラス内で以下のようなメソッドを作成した．

```
1  public Type get_A_B(){
2    return B;
3  }
```

ここでは A がどの表であるか，B が表のどの変数であるかに対応している．これにより，例えば A というリストの i 番目の要素に対して上記のメソッドを実行すれば i 番目の B の値を参照できるようになる．

3.2　表への格納

変数表，関数表への格納は以下の図 1 のフローチャートに従って実行される．

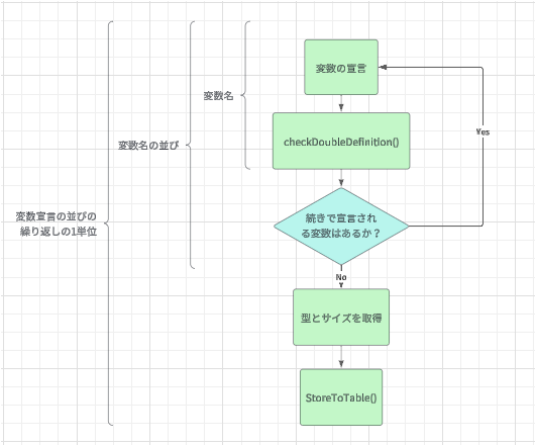

図 1: 変数宣言の並びの繰り返しの 1 単位

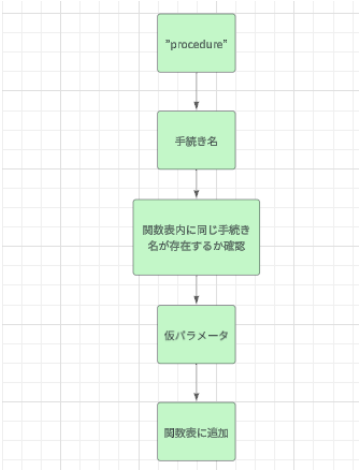

図 2: 副プログラム頭部

図 1 は 1 回以上繰り返される変数宣言の並びの繰り返しの 1 単位の処理を表している．checkDoubleDefinition() と StoreToTable() はそれぞれ二重定義の処理と変数表への格納の処理を行う．また，ここではグローバル変数として list_of_variablename という String 型のリストを使用し，checkDoubleDefinition() の結果二重定義でなければ add() を使用し追加している．ここでいきなり変数表に格納しない理由は，この段階では変数名がわかっただけでありその変数の型やサイズが不明であるためである．したがって，型とサイズが判明してから StoreToTable() を呼び出している．以下では checkDoubleDefinition() と StoreToTable() の詳細について述べる．

- checkDoubleDefinition() では変数名が二重定義されていないかを確認している．引数には宣言された変数名が渡されており，それが各表の名前と一致していないかを確認する．一致している場合即ち二重定義されている場合，文字列”NO”を返し，一致していない場合文字列”YES”を返す．まず関数表内

の関数名が引数の名前と一致していないかを確認する．繰り返し文と 3.1 で紹介したメソッドを用いて表に格納された関数名を取り出して比較する．次に，list_of_variable_name と一致していないかを確認する．そして，プログラムの宣言の際はグローバル変数用の表と，手続きの宣言の際はローカル変数用の表と一致していないかを確認する．確認方法は関数表と同様である．処理の分岐は global_flag を利用して実装する．

- StoreToTable() では表への格納を行っている．引数の list は二つの要素があり，一つ目は格納する変数の型，二つ目は格納する変数のサイズである．global_flag によってグローバル変数用の表に格納するか，ローカル変数用の表に格納するかを分岐させている．このフラグはプログラムの宣言の際に立ち上げ，手続きの宣言の際には下げられている．また，list_of_variablename はこのメソッドが呼び出されるたびに初期化される．これは，新たな変数宣言の繰り返し単位が存在した場合に再度表に追加されてしまうことを防ぐためである．

関数表への格納は図 2 のように副プログラム頭部のメソッドに処理を追加して実装した．関数名を関数表へ追加する前に同じ手続き名が存在しないかを確認する．確認は変数の際と同様にループを用いて実装した．仮パラメータの処理については変数の表への追加と基本的に同じだが，型をリストにして格納している．つまり一つの関数名に対して一つの仮パラメータのリストが存在するということである．これは考察で述べる仮パラメータと実引数との整合性を保つために実装している．

3.3 未定義の変数の参照

未定義の変数の参照を防ぐために checkVariable() というメソッドを定義した．これは変数の定義の後に呼び出される．引数として識別子が渡され，その文字列が変数表内で定義済みかどうかを判定する．定義されていない場合は文字列”NO”を，定義されている場合はその変数の標準型を文字列として返す．ここでは変数のサイズが 1 かどうかも判定され，1 でなければグローバル変数である array_flag が true となる．これを用いた具体的な処理は後述する．

3.4 型の制約

ここでは，各演算子において被演算子の型と結果の型の整合性を確認する処理について記述する．

- まず，被演算子の型の整合性について記述する．加法演算子と乗法演算子の場合”and”と”or”の場合は boolean 型を，それ以外の場合は integer 型を，被演算子の型として処理するようにした．また，被演算子同士の型が一致している必要があるので，最初の被演算子の型を格納する変数 first_type と，もう一つの被演算子の型を格納する変数 second_type が一致しているかどうかを判定する．式や単純式，項のように，0 回以上の繰り返しがある場合は first_type は変えずに，新たな被演算子が現れるたびに second_type を更新し一致しているかを判定することで全ての被演算子の型が一致していることを確かめた．

- 次に，結果の型の整合性について記述する．加法演算子と乗法演算子の場合”and”と”or”の場合は boolean 型を，それ以外の場合は integer 型を結果の型として処理するようにした．また，関係演算子の結果は boolean 型として処理するようにした．

3.5 左辺の型の分類

代入文の構文定義において，左辺が純変数か添え字付き変数かを判定する処理が行われる．このとき，純変数として配列型の変数名が使用されないようにするために 3.2 で記述した array_flag を使用する．構文的

に純変数であると判明した後にこの array_flag が true となっているかを判定する．これにより，上記の 4 番目のテストケースに対応することができるようになる．なお，array_flag は checkVariable() が呼ばれるたびに false に初期化されるため，各変数の参照に対して更新される．

4　考察

今回のテストケースは最低限満たす必要のあるものであったが，実際にはほかにも意味エラーが発生することが考えられる．以下は私が追加で考えられる意味エラーのパターンである．

- 配列の定義の際に，添え字の最小値と最大値の大小関係が逆転している．
- 仮パラメータと実パラメータの整合性がとれていない．

具体的な実装方法は以下のとおりである．

- 添え字の最小値と最大値をそれぞれ保持しておき，大小比較をして最大値＞最小値でなければ意味エラーとすることで実装した．
- 3.2 で記述したように関数表における引数の型をリストとして定義し保持しておく．手続き呼び出し文において式の並びを実装したメソッドの返り値を型のリストに変更し，返された型のリストと，関数表に保持しているリストの要素が同じであるかをループを使って確認することで実装した．

以上のパターンの意味エラーの処理を作成して，今後の拡張内容として以下のような点が挙げられると考えた．

1. 意味エラーの出力だけでなく，どのようなエラーなのか (未定義の変数を参照しているのか，二重定義をしているのか等) を出力する．これにより，プログラマにどのような意味エラーが起きているのかを伝えられるためより親切な設計にすることができる．また，出力する文字列を変更するだけでよいので実装の難易度は低いと考えられる．
2. 変数表に格納されている識別子名の内，参照されていない変数は取り除くもしくは表から削除する．具体的には参照されたかどうかを表すフラグを表に追加し，参照された場合はそのフラグを 1 にそれ以外は 0 にすることで，checker を実行した後にフラグが 0 になっている変数は使用されていないということになるので消去しても問題がないと考えらえる．

5　感想

課題 3 を通して学んだ感想としては，テストファーストな開発がとても効率的であったということである．課題 2 においてウォータフォール型の開発をしたことの反省からテストファーストな開発を試してみて，かなり効率的に実装を進めることができたと感じた．課題 4 でも同様にテストファーストな開発で進めていきたいと思う．

参考文献

[1] 言語処理工学 A　講義スライド

[2] 情報科学演習 D　指導書