提出日 2017/12/23
No. 201721691

問題記述と形式化レポート 担当(中井先生)　　福山 怜史

はじめに

本レポートでは，以下の課題に対する説明を行う．

1. JSON を入力として受け付ける構文解析器の作成
2. 構文解析木を作らずに構文解析木を表示するための dot 形式のファイルを出力する構文解析器の作成
3. 構文解析木を作成し，各ノードクラスにてアクションを定義し，構文解析木を表現する dot 形式を出力するプログラムの作成
4. 構文解析木を作成し，Visitor を用いることで構文解析木を表現する dot 形式を出力するプログラムの作成

ただし，字句解析器は事前に配布された「lexer.rb」を用いる．

課題 1 JSON を入力として受け付ける構文解析器の作成

課題 1 に対応するプログラムはディレクトリ“1”に格納されている．

- 設計

アルゴリズムは再帰的下向き構文解析である．また構文解析部はクラスによって構造化した．

- コーディングの工夫

標準入力ではなくプログラム内でファイルを読み込み，解析を行うようにした．非終端記号を示すメソッドは private にしておき，外部からの参照できないようにした．

- 苦労した点

非終端記号の数が多いため，比例してメソッドの数も多くなり，完成に時間がかかった．

課題 2 構文解析木を作らずに構文解析木を表示するための dot 形式のファイルを出力する構文解析器の作成

課題 2 に対応するプログラムはディレクトリ“2”に格納されている．

- 設計

構文解析木を使えないため，アルゴリズムは課題 1 と同様に再帰的下向き構文解析である．graphviz に対応した dot 形式を出力するために，課題 1 に以下の処理を追加した．

1. 各ノードの解析を行う直前に，ラベル表示の定義(marklabel メソッド)とラベルとラベルの有向エッジを張る処理(arrow)を行う．
2. クラス変数としてノードの識別番号(@@nodeno)を設ける．1. の処理を行ったら@nodeno をインクリメントする．

- コーディングの工夫

String, Int, Float のノードでは値の出力を行うため，ノードの値が格納されている変数(@lexime)にアクセスする処理を追加した．また，String ではダブルクォーテーションが文字列の先頭と末尾にあるためこれを取り除くためにスライシングの処理を加えた．

- 苦労した点

arrow においてノードが左辺と右辺にある場合で，@@nodeno の整合性が取れる処理を行わなければ，正しい出力が得られない．この処理の実装とデバッグに多くの時間を取られた．

課題 3 構文解析木を作成し，各ノードクラスにてアクションを定義し，構文解析木を表現する dot 形式を出力するプログラムの作成

課題 3 に対応するプログラムはディレクトリ“3”に格納されている．

設計
構文解析木を実装する．解析木自体は，12/4 の配布資料 P5 の Node クラスによって構築する．「JsonPerser.rb」に対して，親ノードのメソッド内で子ノードが出現する度に，親ノードの children に子ノードを追加する．以上により構文解析木のデータ構造を取得する．このノードクラスには marklabel 及び arrow の処理を行うアクションメソッド（show メソッド）が用意されているため，獲得した解析木に対して show メソッドにアクセスすることによって dot 用のテキストが出力される．
ファイルへの書き込みは課題 2 と同様である．

- コーディングの工夫

オブジェクト指向による実装を行い，各ノードのクラスを作成した．また，クラス変数であるノードの識別番号(@@nodeno)と arrow および marklabel メソッドを持つ親クラス(Node)を作成し，実際のノードクラスはこの Node クラスを継承する実装を行なった．また各ノードクラスにおいて，非終端記号では小ノードを示すインスタンス変数(@children)とノードのラベルを示すインスタンス変数(@label)を設け，終端記号では@label のみを設ける．そして，各ノードにおけるアクションの処理は，show メソッドにて個別の処理をオーバーライドしている．

- 苦労した点

Node クラスで個別のコンストラクタ及びアクションのメソッドの実装を行うことが大変であった．また当初は，親クラスである Node クラスにて@children, @label を同時に設けていたが，終端記号に子ノードが存在しないことから，存在しない変数を設けることは冗長であるとみなし，現在の親クラスがインスタンス変数を持たないように改良を加えた．

課題 4 構文解析木を作成し，Visitor を用いることで構文解析木を表現する dot 形式を出力するプログラムの作成

課題 4 に対応するプログラムはディレクトリ“4”に格納されている．

- 設計

構文解析木を Visitor のモデルで実装する．解析木自体は，12/4 の配布資料 P9 のソースコードを参考にした．ノードに対して個別のクラスを作成し，Json2dotVisitor クラスにて，それぞれのノードに対する個別のアクションを実装した．アクションの動作は，課題 3 と同様である．

- コーディングの工夫

課題 3 と同様に Node クラスを実際の各ノードクラスが継承する実装を行なっている．また Visitor の各ノードのアクションが格納された辞書(@node)も課題 3 の show メソッドの処理と同様である．

- 苦労した点

Visitor のモデルを理解することが大変であったが，一度理解すると実装は困難ではなかった．

感想

コンパイラの理論的な構成とととともに，データ工学あるいはソフトウェア工学的な視点で実装を行うことができたので，自分自身のコーディング力が向上した．特に，オブジェクト指向や Visitor モデルによる実装の演習がとても勉強になった．また今回の講義で初めて ruby という言語に触れ，その独特なイディオムに苦戦した．今後も積極的に ruby を活用して，実装力を高めていきたい．