【関東/関西/九州同時開催】
女性エンジニア大集合!新春LT×座談会

スクリプト・インタプリタを作ってみた

自己紹介

- 名前　　robo（兼高理恵）
- お仕事　　Ｊａｖａ 技術者
設計から実装まで
- 好きなもの　　モバイル端末

大阪生活??年の関西Java女子部所属なのですが、
昨年途中から東京での作業になりました。

ちょっと脱線

なぜスクリプト･インタプリタを作ってみたのか
というと

正月休み前

【関東/関西/九州同時開催】
女性エンジニア大集合! 新春LT×座談会とな?

よっしゃ、正月休み中になんか作ったろ!

『関西勢の底力を見せつけたるわ～!』

こ○けさん
ごめんなさい、ネタパクりました。

正月休み終わり

と、いうことで、
スクリプト・インタプリタを作り始めましたが、

あかんわ...
構文解析やコマンド解析までたどり着いたけど、
組込関数の実装まで間に合わへん

実行できな、ただのモックやんか
!il!il orz !il!il

臨場感をだすために
心情を関西弁に翻訳してみました。

LT大会から一週間後

LT大会には、間に合いませんでしたが、
インタプリタ(コンパイラ)がやっと完成しました。

Java女子部の皆さん、
一週間遅れてごめんなさい。

サンプル実装とスライドについて

LTスライド：made_the_script_interpreter.pdf
netbeans プロジェクト：TinyBasic.zip
スクリプト実行JAR　　：TinyBasic.jar
サンプル・スクリプト　：sample_script.txt

サンプル・スクリプト内容

```
String name = "John";
String greet = "Hello";
print(greet + " " + name + "!");
Integer num = +1+(2+3)*4+5-6;
print(num);
```

サンプル実行例

```
$ java -jar TinyBasic.jar ./sample_script.txt
Hello John!
20
```

スクリプトは、四則演算とprint機能しか実装されていません。

サンプル実装を参照しながら確認くださいね

インタプリタ⇒言語実装の基礎についての基本概念の覚書をまとめました。

(間違っていましたら御指摘ください)

参考書籍

Java言語で、
インタプリタを
作っていく本です。

初学者でもわかり
やすいと思います。

言語実装の基礎知識が
がっちり学べます。

言語実装パターン
コンパイラ技術によるテキスト処理から言語実装まで
著者：Terence Parr
株式会社オライリー・ジャパン
ISBN978-4-87311-532-0

いまどきの
プログラム言語の作り方
著者：randy
株式会社毎日コミュニケーションズ
ISBN4-8399-1923-2

プログラム言語実装の基本

- ソース　　⇒ 言語構文に従ったTEXT作成
- 字句解析 ⇒ ソースTEXTからトークン列への変換
- 構文解析 ⇒ トークン列から構文木への(入れ子)変換
- 命令解析 ⇒ 構文木から命令コマンド列への変換
- 命令実行 ⇒ コマンド列の子から親への逐次評価

上記は、プログラムを実行させるために
最低限必要な処置(実装)についての説明です。

【プログラム言語構文の策定】
ソーステキスト作成

あるプログラム言語のソーステキストを作るには、
あるプログラム言語の構文策定が前提です。

つまり、インタプリタの実装に入る前に、
言語構文について策定しておく必要があります。

(既存プログラム言語の構文を真似れば悩みも少ないと思います)

【プログラム言語構文の策定】
BNF(バッカス･ナウア)記法

バッカス記法 (BNF: Backus Nauer Form) の構文について (Microsoft)

バッカス･ナウア記法 (ウィキペディアより)

文脈自由文法を定義するのに用いられるメタ言語

左辺と交換できる右辺パターン(繰り返し可)を表現します

（表記例）<symbol> ::= <expression with symbols>

オレオレ言語作成にあたっての構文策定でしたら、
独自表現で構文パターンをまとめればよいと思います。
（例）整数型変数定義 ← int 変数名

字句解析

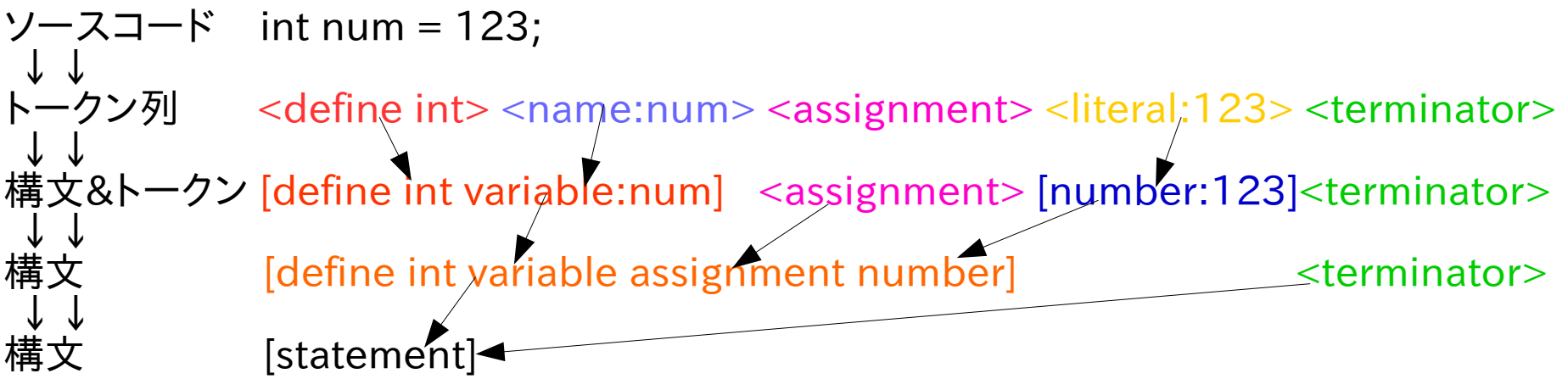
ソースコードの文字列を要素に分解するため、
特定文字パターンを表すトークン列に変換します。

(例)print ”hello!”; ⇒
<keyword:print> <literal:”hello!”> <terminator:;>

字句解析器を作る場合は、正規表現を利用して、
言語構文の意味ある特定文字パターンを規定し、
ソースコード中のマッチ部分をトークンに変換していき、
全文字列をトークン列に置換することになります。

構文解析

複数のトークンや他構文の
組み合わせパターン(繰り返し可)を持った構文により、
トークン列を構文の入れ子構造（構文木）に変換
していきます。



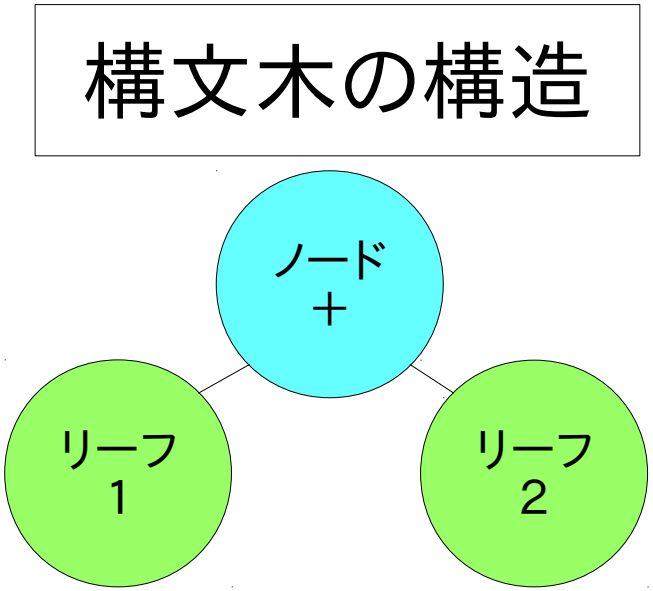
構文解析

木の頂点から処理を開始して、
末端ノードへ向かって下向きに構文解析するものを
再帰的下向き構文解析器と呼びます。(前ページ例は逆向き)

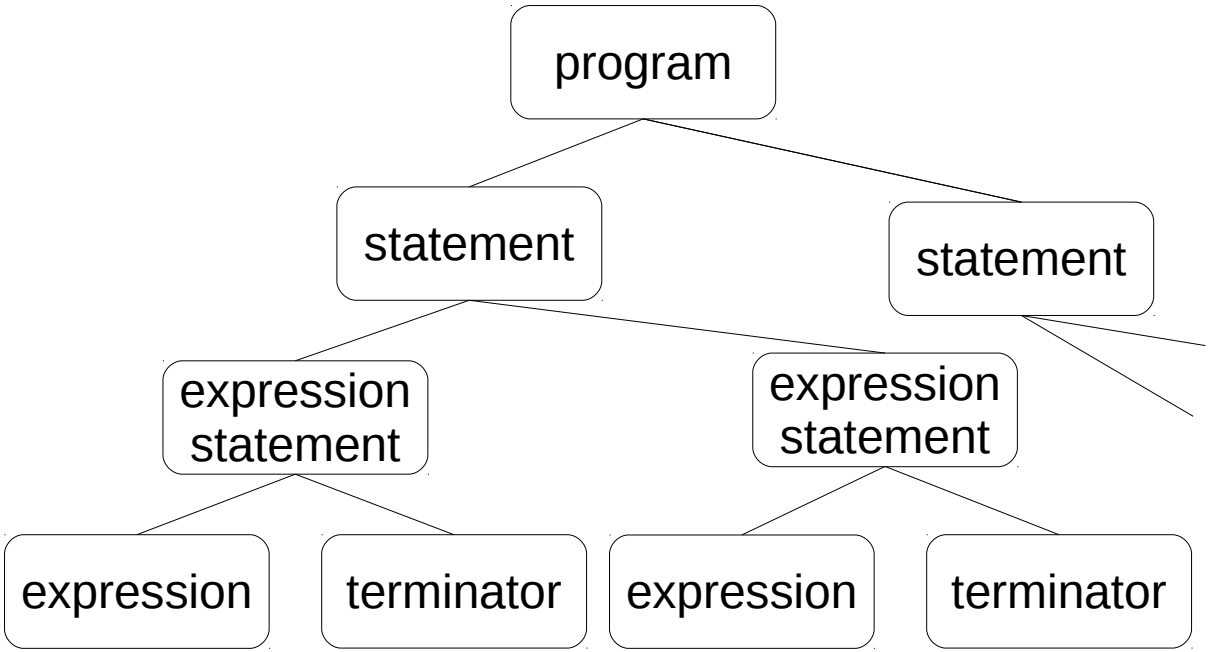
前ページの例では、
入れ子化で最終的に1つの構文木に収束すれば、
構文解析に成功したことになります。

四則演算の + - と * / () で優先順位がありますように、
構文の優先順位や依存関係を注意して策定しないと、
正しい構文解析が行えません。

構文解析



1+2を表す構文木



サンプルでは、リーフが3つまでで、
program をルートに、statement、expression...と
構文概念が細分(詳細)化していく構造になっています。

命令(コマンド)解析

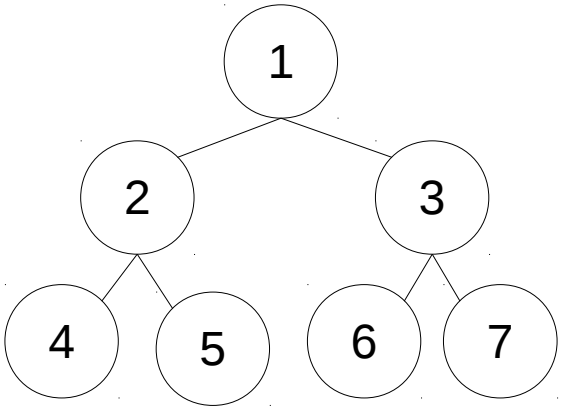
サンプルでは、
構文木構造をあらかじめ解析して、
各構文に対応する命令(コマンド)を選別し、
あらたに命令(コマンド)列を生成しておいてから、
命令(コマンド)を評価していく手法を採用しました。
(下記のスキャナ実装を巨大化させたくなかったため)

プログラムの実行は、スキャナを使って、
構文木の内容を直接評価させていく方法でも可能です。
この場合は、命令(コマンド)列を作りません。

命令(コマンド)解析

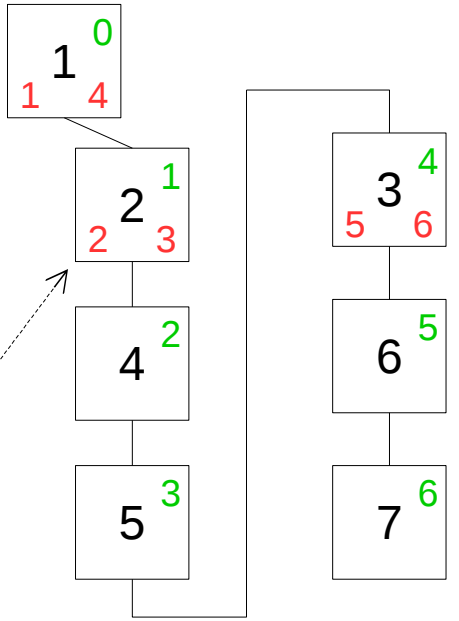
サンプルでのコマンド列作成概略図

構文木



構文に対応するコマンドを作り、左リーフ優先で再帰的に木構造を展開

コマンド列



コマンドは、木構造でないため、コマンド内部に、
子リーフの位置情報(コマンド列インデックス)を
保持させて、必要時にアクセス可能にしています。

例：コマンド2は、内部に4[2]と5[3]のインデックスを持つ

命令(コマンド)実行

ノードは、リーフの評価を行い、
全リーフの評価結果を取得してから、
ノード自身の評価を行い結果を親に返します。

これにより全てのノードの評価が
期待する訪問順序(評価順序)で実行される
ことになります。

評価は、ルート構文木(コマンド)から開始しますが、
ルート構文木(コマンド)のノードが評価されるのは、
一番最後になります。

命令(コマンド)実行

【補足】
プログラムを実行できるようにするには、
コマンドの他に、実行環境や組み込み機能も
必要です。

例えば、変数名と変数値を扱うには、
それらを管理する実行環境が必要になります。

また print など入出力を伴う関数は、
プログラム実行元のシステム機能を利用するため
組み込み関数(機能)も別途必要になります。

構文解析器生成系 (既存ツール)

サンプル実装では、勉強のためフルスクラッチで独自に字句解析器や構文解析器を作りましたが、これらは、すでに定番の生成ツールがあります。

ANTLR

http://www.antlr.org/

(Java製の構文解析器)

ANTLR Island

http://www.limy.org/program/java/antlr/

(ANTLRの使い方説明)

プログラミング言語を作る/yaccとlex

http://kmaebashi.com/programmer/devlang/yacclex.html

(lexが字句解析器で、yaccが構文解析器)

サンプル実装を作ってみて

プログラム言語作成は、
計算機科学的に基本が確立されているので、
手順と考え方や概念は、難しくはないけれど、
アルゴリズムを考えるのがややこしいと思いました。

プログラム要素の各意味を表し、整合性を持つ
構文の策定が一番重要で時間がかかるものと
思い ます。

ご清聴、ありがとうございました。