コンパイラ演習参考資料　第１回　　担当：佐々木晃

http://cis.k.hosei.ac.jp/~asasaki/lect/compiler に資料を置く予定。(現在は昨年度の資料がある。)

〇算術式の処理と逆ポーランド記法（第一回スライド２９ページ)

（1）実数値(double の値)を格納するスタックを実装せよ。(配列やリストを使うとよい)

（2）逆ポーランド記法によって実数値の算術演算を行う計算機のプログラムを作成せよ。演算子や被演算子の各要素同士は空白で区切られるものとする。

（a）四則演算のみ

なお、実行結果は、次の例と自分の作成した例で示せ。

(1) 4.0 3.0 -

(2) 5.0 4.0 3.0 -　*

(3) 5.0 4.0 3.0 * -

(4) 4.0 3.0 - 5.0 6.0 + *

（b）三角関数なども使えるようにしてみよ。（余力のある人）

"2.0 pi sin"は、sin(2.0*π) (答えは 0 (実際は丸め誤差が生ずる)),

→pi sin はそれぞれ単項のオペレータ（スタックトップに対して演算を行う演算子）

→2.0 pi は 2.0 をスタックに積む。スタックトップの 2.0 とπを掛けた結果を

スタックのトップとする。(2.0 2.0 * pi sin は、[2.0, 2.0] → [4.0] →[4.0π] →[0]

→java.lang.Math ライブラリなどを利用すればよい。

（3）第１回スライドｐ２９にある方法（この資料の最後でも説明している）を用いて、中置記法による算術式（四則演算を行う）を逆ポーランド記法へ変換するプログラムを作成せよ。ただしオペランド（被演算子数）は任意の文字列とし、演算子、被演算子、括弧は空白で区切られるものとする。（問題１と同様）なお、実行結果は、次の例と自分の作成した例で示せ。

(0) aa + bb

(1) aa + bb * cc

(2) (aa + bb) * cc

(3) d - e * (f + g) - h

(4) n - (o + p * q) * r

後置記法に変換するアルゴリズム(１回目講義スライド p29 をまとめたもの)。

初期設定:スタックに'$'を積む。
以下の I,II を繰り返す：
I. 次の入力記号（トークン）ai を読む。読み込むトークンがない場合は ai='$'とする。(1)
II. ai で場合分けの処理を行う。
(a) ai がオペランドの場合、そのまま表示する。(2)
(b) ai が'('の場合→それをスタックに積む(3)
(c) ai が')'の場合→最初の'('が現れるまでスタックの内容を順番に降ろす。(4)
その際、降ろした内容がオペレータ（演算子）であった場合はそれを表示する。

(d) ai が'$'の場合→'$'が現れるまでスタックの内容を順番に降ろし処理を終了。
その際、降ろした内容がオペレータ（演算子）であった場合はそれを表示する。(5)
(e) それ以外（つまり、ai がオペレータ(演算子)の場合（※）の処理を行う。(6)

（※）以下、まず(a)を行い、次に(b)を行う。
(a) b がオペレータで、かつ、prec(b) >= prec(ai)である間以下を繰り返す。
1. スタックトップ（つまり b）を降ろす。
2. b を表示する。
(b) b がオペレータでない場合、あるいは、prec(b)< prec(ai)となった場合（すなわち、(a)が成り立たなくなった場合））ai をスタックに乗せる。

ここで、b はスタックトップの内容、prec(x)を演算子順位の値を返す関数であるとする。
prec('+') = 1, prec('-') = 1,　prec('*') = 2,　prec('/') = 2

プログラム作成までのステップ:
(0) アルゴリズムを理解する。例えば以下の式をアルゴリズムに沿って紙に書いて処理を追ってみる。
a + b * c
→　オペランド（被演算子）やオペレータ（演算子）がどのようにスタックに積まれるか、演算子の優先順位はどのように処理されるか？
(a + b) * c
→括弧がある場合はどのように処理されるか？

(1) 擬似コードにする。

```
allTokens = allTokens + "$";
push(“$”);
while(true){
  ai = 次のトークンを読む();
```

```
    if(ai がオペランド) ai を表示;
      else if (ai == '(')   /* (2)の処理 */
        push(ai);
      else if (ai == ')')   /* (3)の処理 */
        do {
          c = pop();
          if (c がオペランド)
              c を表示;
        } while(c が'(');
      else if ....
  }
```

（※）の処理部分の擬似コード

```
      while(true){
        b = peek();  // スタックトップの内容を（おろさずに）調べる関数
        if(b がオペレータかつ prec(b) >= prec(ai))
            pop();
            b を表示;
        else
            break;
      }
      push(ai);
```

(2) プログラムを書く。

試すときは、いきなり長い式を処理しようとしない。単純な式から、それが動くことを確認して、だんだん複雑な式を試す。

a → a+b → a+b+c → a+b*c → a*b+c → (a+b) → (a+b)*c

まずは、問題を簡単にして括弧のない式だけが読めるプログラムを目標にして作っても良い。その動きを確認したら、括弧を使えるものに拡張する。

うまく動かない場合は、まず、必要となる小さな部品が想定どおりに動作するかを確かめる。今回の場合は、大きく分けて以下の部品があるので、それらがきちんと動くかを確かめる。

- スタックの処理(push, pop, peek)の部分の理解。
- 単語の読み込み(トークンの切り出し)の部分。
- メインルーチンやサブルーチン。

□練習問題 スタックの実装。（＝問題(1)）

(ヒント。配列を利用する場合は、下記のようにする。isEmpty(), pop(), peek(), push(double d)の中身を埋めよ。次の言葉が分からない人は、java の復習をすること。

- 配列
- コンストラクタ
- メソッド、引数、返り値

)

```
public class MyStack {
    static final int NUM_CONTENTS = 100; // 最大値は１００
    double [] contents;
    private int index = -1;

    public MyStack() {
    //コンストラクタ
        contents = new double[NUM_CONTENTS];
    }

    boolean isEmpty(){
        // スタックの中身が空かどうかを返すメソッド
    }

    double pop(){
    //スタックの先頭（トップ）から要素をおろす。返り値はおろした値
    }

    double peek(){
    //スタックの先頭（トップ）の内容を返す。
    }

    void push(double d){
    //引数 d で与えられた double 値をスタックに積む。
    }

    public static void main(String[] args) {
        // テストプログラム
        MyStack st = new MyStack();
        st.push(12.34);
        st.push(45.68);
```

```
            System.out.println("st.pop() = " + st.pop());
            System.out.println("st.peek() = " + st.peek());
            st.pop();
            System.out.println("st.isEmpty() = " + st.isEmpty());
            st.pop();     // エラー

    }
}
```

□練習問題（＝問題(2)、(3)への準備）

java.util.Stack クラスを利用して、スタックの動きを確かめよ。下記は例。

```
import java.util.Stack;
public class StackTest {
    public static void main(String[] args) {
            System.out.println("String Stack test ----");
            Stack<String> st = new Stack<String>();
            st.push("abc");
            System.out.println("st = " + st);
            st.push("def");
            System.out.println("st = " + st);
            st.pop();
            System.out.println("st = " + st);
            System.out.println("st.peek() = " + st.peek());
            System.out.println("st = " + st);

            System.out.println("Double Stack test ----");
              Stack<Double> dst = new Stack<Double>();
            dst.push(123.0);
            dst.push(987.456);
            dst.push(Double.parseDouble("123.4567"));
            System.out.println("dst = " + dst);
            System.out.println("dst.pop() = " + dst.pop());
    }
}
```

□練習問題（＝問題(2)）

逆ポーランドの計算機を作成せよ。

(1) まず、入力した式を空白に区切る実験。

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class CalcPostfixPre {
	public static void main(String[] args) {
		BufferedReader r= new BufferedReader(new InputStreamReader(System.in));
		String line = "";
		try {
			line = r.readLine();
		} catch (IOException e) {
			e.printStackTrace();
			System.exit(0);
		}
		String [] allTokens = line.split(" "); // 空白で区切る
		 for(String token : allTokens){		// for each
			System.out.println("token = " + token);
		}
	}
}
```

(2)スタックを使って処理を行う。((1)への修正。他の演算子の部分を作る。)

```
		String [] allTokens = line.split(" ");
		Stack<Double> st = new Stack<Double>(); // 先の MyStack を使ってもよい。
		for(String token : allTokens){
			System.out.println("token = " + token);
			if ("+".equals(token)) {
				double y = st.pop();
				double x = st.pop();
				st.push(x + y);
			} else if("-".equals(token)){
			   ....
			} else if ..../// 他の演算子の場合を書く。

			else {// オペランドの場合
				st.push(Double.parseDouble(token));
			}
		}
```