第A章 Oolong について

A.1 Oolong とは

Oolong は JVM (Java Virtual Machine) のコードをターゲットとするアセンブリー言語である。Oolong については、以下の書籍で紹介されている。

> Joshua Engel: “Programming for the Java Virtual Machine”
> ADDISON-WESLEY, ISBN 0-201-30972-6, 1999

JVM は、仮想 CPU（つまりソフトウェア上でエミュレートされる CPU）の一種である。JVM の命令セットは本質的な部分は Intel x86 や MIPS などの現実の CPU と似ている。しかし、レジスターベースではなく、スタックベースなので、レジスター割り付けの必要がなく、現実の CPU を対象とするよりはコード生成が容易である。

このため、本演習では、作成するコンパイラーのターゲットとして、Intel x86 などの現実の CPU の機械語ではなく、この Oolong（すなわち JVM のアセンブリー言語）を使用する。

JVM はもともと Java というプログラミング言語のコンパイラーのターゲットとして設計された仮想機械である。Java 言語自体は、C 言語によく似た制御構造を持つ“高水準”プログラミング言語であり、Java とアセンブリー言語である Oolong とは、まったくの別物である。（C 言語と Intel x86 のアセンブリー言語が全く異なるのと同じことである。） 実際、JVM をターゲットとする Java 言語以外のプログラミング言語のコンパイラーも、数多く存在する。

A.2 Oolong の実行方法

Oolong の処理系（アセンブラー）自体も JVM 上で実装されているので、Oolong を実行するためには、まず JVM（JRE という Java の実行環境）と oolong.jar というファイルを入手する必要がある。

Oolong ソースファイル の拡張子は `.j` である。*Filename*.j という名前の Oolong ファイルをアセンブルする時のコマンドは次のようになる。

```
java -jar oolong.jar Filename.j
```

oolong.jar を別のフォルダーに置いている場合は、上記の `oolong.jar` の部分は oolong.jar のフルパスを記述する。このコマンドで、 *Filename*.class というファイルが生成される。このファイルの中身は、仮想機械用のコードなので、直接実行することはできない。実行するには、次のように `java` コマンドを用いる。

```
java Filename
```

この時は、最後に拡張子の .class をつけないことに注意する。

A.3 Oolong のファイル構造

本演習で使用する Oolong ファイルは、すべて次のような雛型に従う。そして、*Statements* の部分を必要に応じて書き換える。Oolong の文法では、改行は意味を持っているので、この例のとおりに改行を入れる必要がある。

```
.super java/lang/Object
.method public static main([Ljava/lang/String;)V
Statements
.end method
```

なお、クラス名はソースファイル名の拡張子（.j）を除いた部分と同じ名前になる。

A.4 Oolong の命令文（*Statements*）

Statements は命令文（*Statement*）の並びである。Oolong では、必ず 1 行に 1 つの命令文を書く。本演習では、次のような命令文を使用する。浮動小数点数関係など、ここで紹介していないその他の Oolong（すなわち JVM）の命令文については、Oracle 社の Web ページ（http://docs.oracle.com/javase/specs/jvms/se10/html/index.html）や Jasmin (Jasmin は Oolong の基になった JVM アセンブラーである。 Jasmin と Oolong の文法はほぼ同一であるが、Jasmin で必要ないくつかの宣言を、Oolong では省略することが可能である。)の Web ページ（http://jasmin.sourceforge.net/guide.html）で知ることができる。

JVM はスタックベースの仮想機械である。つまり、Oolong のほとんどの命令文は（レジスターではなく）スタックに置かれているデータをパラメーターとして動作する。以下では、スタック操作関係、分岐関係、整数演算関係、変数操作関係、その他にわけて Oolong の命令文を紹介する。以下の説明中で、a はスタックの先頭の要素、b はスタックの 2 番目の要素を表す。「増減」はスタック中の要素数の変化を表す。

スタック操作関係

命令	増減	説明
ldc *Int*	1	整数定数をスタックにプッシュする。(**l**oa**d** **c**onstant)
ldc *String*	1	文字列定数をスタックにプッシュする。
dup	1	スタックの先頭の要素 a を複製する。(**dup**licate)
pop	-1	スタックの先頭の要素 a を取り除く。
swap	0	スタックの先頭の 2 要素 a, b を入れ換える。
nop	0	何もしない。(**n**o **op**eration)

分岐関係

JVM は goto などの無条件分岐命令と、条件付きの分岐命令を持っている。JVM のコード中では、オフセットを指定することによりジャンプするが、Oolong のソースコードでは *Label* で分岐先を指定することができる。 *Label* 名には、アルファベットからはじまり、空白文字を含まない任意の文字列を使用することができる。

命令	増減	説明
Label：	–	goto 文など分岐命令の分岐先ラベルを設定する。
goto *Label*	0	*Label* に無条件に分岐する。
if_icmpeq *Label*	-2	a, b をスタックから取り除き（以下同様）、 b == a ならば、 *Label* に分岐する。 (**if i**nteger **comp**are **equ**al)
if_icmpne *Label*	-2	b != a ならば、 *Label* に分岐する。 (**if i**nteger **comp**are **not e**qual)
if_icmpge *Label*	-2	b >= a ならば、 *Label* に分岐する。 (**if i**nteger **comp**are **g**reater than or **e**qual)
if_icmpgt *Label*	-2	b > a ならば、 *Label* に分岐する。 (**if i**nteger **comp**are **g**reater **t**han)
if_icmple *Label*	-2	b <= a ならば、 *Label* に分岐する。 (**if i**nteger **comp**are **l**ess than or **e**qual)
if_icmplt *Label*	-2	b < a ならば、 *Label* に分岐する。 (**if i**nteger **comp**are **l**ess **t**han)
ifeq *Label*	-1	a == 0 ならば、 *Label* に分岐する。
ifne *Label*	-1	a != 0 ならば、 *Label* に分岐する。
return	–	メソッドから値を返さずに return する。 **注:** 1.3 節で紹介した雛型でも、メソッドの最後で必ず return する必要がある。

整数演算関係

ここでは整数に関する算術演算の命令のみを紹介する。

命令	増減	説明
iadd	-1	b + a、加算（スタックから a, b を取り除き、b + a をスタックに積む。以下同様。）（**i**nteger **add**）
isub	-1	b – a、減算（**i**nteger **sub**tract）
imul	-1	b * a、乗算（**i**nteger **mul**tiply）
idiv	-1	b / a、（整数としての）除算（**i**nteger **div**ide）
irem	-1	b % a、（整数としての）剰余（**i**nteger **rem**ain）

変数操作関係

JVM では、変数は番号で参照され、利用できる番号は 0 ～ 65535 までである。ちなみに、3 節の雛型の場合は、このうち 0 番の変数はメソッドの引数として使用済みである。

命令	増減	説明
iload *Int*	1	*Int* 番目の変数の値をスタックにプッシュする。
istore *Int*	-1	スタックから a をポップし、 その値を*Int* 番目の変数に格納する。

その他

整数や文字列を画面に出力するために必要な命令を紹介する。スタックの先頭（a）に出力したいデータ、スタックの 2 番目（b）に出力ストリームが入っている状態で、出力のための命令を呼び出す。

命令	増減	説明
getstatic java/lang/System/out Ljava/io/PrintStream;		
	1	標準出力ストリームをスタックにプッシュする
invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V		
	-2	出力ストリーム b に a（文字列）を出力する。
invokevirtual java/io/PrintStream/println(I)V		
	-2	出力ストリーム b に a（整数）を出力する。
invokevirtual java/io/PrintStream/println(C)V		
	-2	出力ストリーム b に a（文字）を出力する。

A.5 Oolong のプログラム例

まず、“Hello World!” と出力する Oolong のコードを紹介する。

ファイル名: Hello.j

```
1  .super java/lang/Object
2  .method public static main([Ljava/lang/String;)V
3    getstatic java/lang/System/out Ljava/io/PrintStream;
4    ldc "Hello World!"
5    invokevirtual
   java/io/PrintStream/println(Ljava/lang/String;)V
6    return                        ; 最後にreturnが必要
7  .end method
```

なお、Oolong はセミコロン「;」から行末までが**コメント**になる。

次は、繰返しを用いて “Hello World!” を 10 回出力するプログラムである。

ファイル名: ManyHello.j

```
1  .super java/lang/Object
2  .method public static main([Ljava/lang/String;)V
3    ldc 0
4    istore 1                      ; 変数1に 0を代入する
5
6    loop:                         ; ここからループ
7      iload 1
8      ldc 10
9      if_icmpge exit              ; 変数1が 10以上なら exit
   へ
10     getstatic java/lang/System/out
   Ljava/io/PrintStream;
11     ldc "Hello World!"
12     invokevirtual
   java/io/PrintStream/println(Ljava/lang/String;)V
13     iload 1
14     ldc 1
15     iadd
16     istore 1                    ; 変数1に 1を足す
17   goto loop                     ; loopへジャンプ
18
19   exit:
20     return                      ; 最後にreturnが必要
```

```
.end method
```

ちなみに、これらは、それぞれ次のような Java のプログラムのコンパイル結果に相当する。この中で `System.out.println` は C 言語の `puts` に相当する出力メソッドである。

ファイル名: Hello.java

```java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello World");
    return;
  }
}
```

ファイル名: ManyHello.java

```java
public class ManyHello {
  public static void main(String[] args) {
    int i = 0
    while (true) {
      if (i >= 10) break;
      System.out.println("Hello World");
      i = i + 1;
    }
    return;
  }
}
```
