6章　中間言語

コンパイラが原始プログラムを目的プログラムに変換する際に作る中間的なプログラムまたはその言語を**中間コード**、**中間言語**といいます。中間言語には、解析木、構文木、逆ポーランド記法、三つ組、四つ組などがあります。

6.1　構文木

構文木は主に式の表現に用いられます。木の葉に被演算子を、木の節に演算子を置いた式の木構造表現です。

たとえば、 a = (b + c) * d/2.0 の構文木は、
（図）

のような表現となります。構文木は、構文解析の結果として出力される解析木から作られます。

6.2　後置記法（逆ポーランド記法）

後置記法は主に式の表現に用いられます。構文木を**後順走査**して得られる式の一次元表現です。
構文木の**後順走査**とは、子を順に左側から右側へたどり、最後に親をたどる走査方法です。
（図）

たとえば、a = (b + c) * d/2.0 の構文木の場合は、

a b c + d * 2.0 / =

となります。

※逆ポーランド記法（Polish　Notation）は、ポーランドの論理学者J. Lukasiewicz（ルカシィエビッツ）にちなんで付けられた名前です。

その他の表記法として、**前置記法**と**中置記法**があります。併せて押さえておきましょう。

・構文木を**前順走査**して得られるものを**前置記法**といいます。

構文木の**前順走査**とは、最初に親をたどり、次に子を順に左側から右側へたどる走査方法です。

たとえば、a = (b + c) * d/2.0 の構文木の例の前置記法は次になります

= a / * + b c d 2.0

・構文木を**(中)間順走査**して得られるものを**中置記法**といいます。

構文木の**(中)間順走査**とは、最初の子をたどり、次に親をたどり、その後で残りの子を順に左側からたどる走査方法です。

たとえば、a = (b + c) * d/2.0 の構文木の例の中置記法は次になります。

a = b + c * d / 2.0

前置記法と後置記法は、構文木をあいまいさ無く表現できます。

6.3 三つ組(二番地コード)

三つ組は、主に式の表現に用いられます。構文木を(演算子、被演算子1、被演算子2)で順に表す表現方法です。三つ組は、番号(演算子、被演算子1、被演算子2)の形をもちます。番号は対応する演算結果を参照するために用いられます。

たとえば、a = (b + c) * d/2.0 の構文木の場合は、以下のようになります。

1.(+, b, c)
2.(*, 1, d)
3.(/, 2, 2.0)
4.(=, a, 3)

6.4 四つ組(三番地コード)

三つ組に、演算の計算結果を代入する変数を明示する表現方法です。四つ組は、(演算子、被演算子1、被演算子2、結果の変数)の形をもちます。

たとえば、a = (b + c) * d/2.0 の構文木の場合は、以下のようになります。

(+, b, c, A)
(*, A, d, B)
(/, B, 2.0, a)

7章　コード生成

コード生成では、解析木や構文木、後置記法などの中間コードを機械語コードの列に置き換えます。記号表に登録された変数や定数などを主記憶上の番地に置き換えます。

7．1　スタック機械とコード生成

スタック機械とは、データの記憶領域がレジスタや主記憶（メモリ）ではなく、スタックにより構成される計算機モデルのことです。Java仮想マシン（VM）はスタック機械です。

スタックとは、後に入れたデータが先に出てくるLIFO（Last　In　First　Out）の特徴をもつデータ構造です。スタックとは逆の、先に入れたデータが先に出てくるFIFO（First　In　First　Out）の特徴をもつものに**キュー**があります。

（図）

全体のコード生成は、

(1) 制御構造のコード生成

(2) 関数呼び出しのコード生成

(3) 算術式のコード生成

(4) 論理式のコード生成

などいくつかに分類して考えることができます。ここでは、スタック機械上で実行されるコードを想定し、(1) (3) (4)について説明します。

スタック機械の主な命令には、次のものがあります。

命令 (オペコード)	パラメータ (オペランド)	機能	意味
PUSH	para	プッシュ	paraが変数であればその中の値をスタックへ積み、paraが数値であればその値をスタックへ積む。
POP		ポップ	スタックのトップの値を取り出す。
ASSIGN	var	アサイン	スタックのトップをポップし、その値を変数varへ書き込む。
JUMP	label	分岐	ラベルlabelへ飛ぶ。
FJUMP	label	分岐	スタックのトップをポップし、0であればラベルlabelへ飛ぶ。
TJUMP	label	分岐	スタックのトップをポップし、0でなければラベルlabelへ飛ぶ。
INV		符号反転	スタックのトップをポップし、その値の符号を反転する。結果をスタックのトップへプッシュする。
ADD		加算	スタックのトップと2番目をポップし、それらを加算する。結果をスタックのトップへプッシュする。
SUB		減算	スタックのトップと2番目をポップし、2番目からトップを減ずる。結果をスタックのトップへプッシュする。
MULT		乗算	スタックのトップと2番目をポップし、それらを乗算する。結果をスタックのトップへプッシュする。
DIV		除算	スタックのトップと2番目をポップし、2番目をトップで割る。結果をスタックのトップへプッシュする。
MOD		剰余	スタックのトップと2番目をポップし、2番目をトップで割ったときの余りを計算する。結果をスタックのトップへプッシュする。
GTOP		>	スタックのトップと2番目をポップし、2番目がトップより大きければ1、そうでなければ0をスタックのトップへプッシュする。
GEOP		>=	スタックのトップと2番目をポップし、2番目がトップ以上であれば1、そうでなければ0をスタックのトップへプッシュする。
LTOP		<	スタックのトップと2番目をポップし、2番目がトップより小さければ1、そうでなければ0をスタックのトップへプッシュする。
LEOP		<=	スタックのトップと2番目をポップし、2番目がトップ以下であれば1、そうでなければ0をスタックのトップへプッシュする。
EQOP		==	スタックのトップと2番目をポップし、2番目とトップが等しいならば1、そうでなければ0をスタックのトップへプッシュする。
NEOP		!=	スタックのトップと2番目をポップし、2番目とトップが等しくないならば1、そうでなければ0をスタックのトップへプッシュする。
ANDOP		&&	スタックのトップと2番目をポップし、2番目とトップで一方または両方が0ならば0、そうでなければ1をスタックのトップへプッシュする。
OROP		||	スタックのトップと2番目をポップし、2番目とトップで両方が0ならば0、そうでなければ1をスタックのトップへプッシュする。

※論理の“真”は 1 以上の整数 、“偽”は 0 で表現されます。

9.2 制御構造のコード生成

if文、while文、switch文などの制御構造は、次のようにスタック機械の分岐命令に置き換えていけばよいです。

〔while文の場合〕
【構文】 while(式)文
(図)

【機械語コード】
L1: (式のコード)
　　FJUMP L2
　　(文のコード)
　　JUMP L1
L2:

〔if文の場合〕
【構文】 if(式) 文1 else 文2
(図)

【機械語コード】
　　(式のコード)
　　FJUMP L1
　　(文1のコード)
　　JUMP L2
L1: (文2のコード)
L2:

9.3 算術式のコード生成

四則演算などの算術式は、その構文木の後置記法から次のようにスタック機械の機械語コードを容易に生成できます。

【算術式】 $w = x * y - (10 - y)/(-^{t}y + z)$
ここで$-^{t}$は単項演算子マイナスです

【構文木】
(図)

【後置記法】ｗ ｘ ｙ ＊ 10 ｙ － ｙ $-^t$ ｚ ＋ / － ＝
ここで$-^t$は単項演算子マイナスです

【機械語コード】

```
PUSH  x
PUSH  y
MULT
PUSH  10
PUSH  y
SUB
PUSH  y
INV
PUSH  z
ADD
DIV
SUB
ASSIGN  w
```

後置記法の
w○○＝
の部分は、変数wへの代入です。構文木の
w＝○○
を取りのぞいた部分の後置記法を生成して、
ｘ ｙ ＊ 10 ｙ － ｙ $-^t$ ｚ ＋ / －
代入は最後のASSIGN命令で別に行っています。

9.4 論理式のコード生成

論理式のコード生成は、算術式と同じ方法で容易に生成できます。

【論理式】ｐ ＞ 10 && ｑ ＜ 0

【構文木】
（図）

【後置記法】p 10 ＞ q 0 ＜ &&

【機械語コード】

```
PUSH p
PUSH 10
GTOP
PUSH q
PUSH 0
LTOP
ANDOP
```

論理式の評価では、部分の評価で全体の評価が決まってしまうことがあります。たとえば、上の例では、p＝5の場合、qの値が何であろうと全体の評価は偽となります。

これは、次のように分岐命令を用いてコード化することにより実現できます。

（論理積の場合）
【構文】 式1 ＆＆ 式2

【論理】
〔式1が偽であれば → 全体は偽〕

【機械語コード】

```
      （式1のコード）
      FJUMP L1
      （式2のコード）
      FJUMP L1
      PUSH 1
      JUMP L2
L1:   PUSH 0
L2:
```

（論理和の場合）
【構文】 式1 ｜｜ 式2

【論理】
〔式1が真であれば → 全体は真〕

【機械語コード】

```
      （式1のコード）
      TJUMP L1
      （式2のコード）
      TJUMP L1
      PUSH 0
      JUMP L2
L1:   PUSH 1
L2:
```

たとえば、 p ＞ 10 && q ＜ 0 のときは次のような機械語コードとなります。網掛けは条件を評価するコードです。

【機械語コード】

```
        PUSH  p
        PUSH  10
        GTOP
        FJUMP  L1
        PUSH  q
        PUSH  0
        LTOP
        FJUMP  L1
        PUSH  1
        JUMP  L2
L1：  PUSH  0
L2：
```