2D-6

プログラムの分解・再合成をするモジュール方式

印南 順一 ・ 間野 浩太郎
青山学院大学 理工学部 経営工学科

1. はじめに

プログラムリストを読み易くしプログラミングを容易にするために、プログラミング言語に、関数やサブルーチンという手続きの概念、ブロックIF構造・WHILEブロック構造等に代表される論理制御構造の概念、一群のデータとその処理手続きを一体化した抽象データ型のパッケージの概念等、各種の抽象化概念が導入されてきた。さらには、オブジェクト指向プログラミングでは、インスタンスやクラスといった概念が導入されてきた。これらは、プログラムのある形態の部分を抽象化するものである。つまり、プログラミングにおける抽象化とは、複雑な操作を一つの簡潔な用語で表現し、その中の仮引数を実引数の表現で置き換えることと考えることができる。

本稿では、プログラムの上部構造の抽象化という新しい概念をこれまでの抽象化概念に追加し導入することにより、複雑なプログラムの構造を単純なアルゴリズムのパターンの組合わせに帰着するための方法論とその実現方法について述べる。

2. 基本思想

データ抽象化のパッケージを用いて、アルゴリズムを表す部分を部品化してプログラミングの効率を高めたり、プログラムをわかりやすいものにしたりすることがなされている。しかし、これらすべてのパッケージを共通ライブラリに登録すると、似たような部品が増加してしまう。そこで、この部品の数を減らすことを考える。

一般に、パッケージ化されたプログラムの段階になると、共通に使えるようになるものが減るか、あるいは無くなるので、これらは機械語レベルでの個々のプログラムに対応したライブラリ（個人ライブラリ）に登録する。共通ライブラリに登録するものは、データ構造を取り扱う基本的なアルゴリズムを満足するもの、つまり手続きそのものを合成することができるように抽象化された小さな部分にする必要がある。そして、この共通ライブラリに登録された部品をソースコードレベルで記述する。これらを組合わせたり、または、処理対象の属性を記述する欄を追加したりして、多くのプログラムをつくることができる。

3. 上部構造の抽象化

抽象化されたものが目的に適合していると言えるためには、この概念を説明する一般的な用語を具体的なもので置き換えた時に、実際の場合に正しく適合していることが必要である。プログラミングにおいては、仮引数と実引数の関係に相当する。より局所的な下部の処理を抽象化するのであれば、通常のマクロや手続きを用いればよい。

抽象化する上部の制御構造の記述の中には、仮引数のように実際に使われる場合には必ず具体的なもので置き換えられるものが必要である。つまり、下部の処理で決定する仮引数が必要になる。一方、下部の処理が上部の処理概念の一部となっている時、処理のタイミングが正しく行なわれるためには、上部の構造の処理の中の適切な場所で下部の処理が呼び出される必要がある。この機構を実現するには、下部のものが上部の構造中のある仮引数を決定すると同時に下部の処理の本体が実引数となって一つの仮引数を置き換えることが必要である。

本研究では、この引数の概念を変数や配列の宣言にまで拡張する。

4. マクロとモジュール

我々は、通常のコンパイラに対するソースコードレベルでの変換・置換できることを前提にしている。ただし、マクロが引用される時に同時に仮引数が実引数で置き換えられるのに対して、我々は、引用された時にその中の仮引数をそのままにしておき、後で会話型エディタで指定した実引数で置き換えるという方式を採用している。つまり、仮引数に対する実引数の置換は、パターンの引用の後に切り放して行なう。この引用されるパターンをモジュールと呼ぶことにする。そして、このパターンを仮引数を交えた記述（@で括る）のままのソースコードで共通ライブラリに登録する。この一例を図1に示す。

Modular programming in order to analyze and synthesize a program

Junichi INNAMI , Koutarou MANO
Aoyama Gakuin Univ.

5. 処理系

処理系の概要を図2に示す。

プログラムの適所で、登録されているモジュールをその名前で引用し、エディタを用いて仮引数を実引数の文字列に置換する。この際、多段階で仮引数を実引数に置換していく。つまり、すべての仮引数が引用時に必ずしも同時に置換されるとは限らないので、具体化の各段階の経過をファイルとして、ライブラリに格納する。このファイルをヒストリファイルと呼ぶ。仮引数の実引数化が完了すると、完全なMDL-FORTRANのコードが生成される。ここで必要があれば、個人ライブラリに登録する。これをプリプロセッサでFORTRANのソースコードに変換する。

同じ上部構造のパターンが一つのコンパイリング単位のプログラム内で引用される時に生じる重要な問題点がある。一つのプログラム単位（一つの分割コンパイルの対象）で、一つのモジュールが複数回引用されたり、または、複数のモジュールが引用され組み合わされたりする場合に、モジュール内の仮引数でない識別名が絶対に同一の識別名に展開されないように、次に示す命名規則を採用する。

モジュール内のニーモニック	_	そのモジュール名	_	そのコンパイル単位でのそのモジュールの引用番号

6. おわりに

本研究では、モジュールとして、レコード（あるいはセル）間の関係を明示的に記述するようなものを対象にしている。完成されたシステムは主に学校教育の手助けとなることを目的としている。

FORTRAN言語を目的言語としているので、静的な環境下で行なわれる。よって、実行効率を低下させないで、ソースコードレベルでの様々な操作で実現できる。逆に、実行時に文脈により処理の型を決定する機構を組み入れることができない。

現在、μ-VAX II のVMS上で開発を進めている。

```
$SPEC FORTRAN
$PROGRAM Array_Queue
$BSUBPGM AQ
$SHARE-PART
C
C   配列型の待ち行列   :   array_queue
C
C       待ち行列の長さ
C       箱の型名と箱名
C       仮引数の型名と仮引数名
C
$SHARE-PART
      integer       size
      parameter   (size = $CONST:待ち行列の長さ$ )
      ( $DECL:箱の型名$ $箱名$ (size) )
      ( $DECL:仮引数の型名$ $仮引数名$ )
      integer       front
      integer       rear
$END-SHARE
$MAIN-SCTN
      *ENTRY init_array_queu
        front=0
        rear=0
      RETURN
$END-MNSCTN
$MAIN-SCTN
      *ENTRY put_into_array_queue ( ( $仮引数名$ ) )
        *IF(rear.eq.size)
            rear=1
        *ELSE
            rear=rear+1
        *END-IF
        *IF(rear.eq.front)
            call errsgn('待ち行列が満杯')
        *END-IF
        ( $箱名$ (front) = $仮引数名$ )
        *IF(front.eq.0)
            front=1
        *END-IF
      RETURN
$END-MNSCTN
$MAIN-SCTN
      *ENTRY get_from_array_queue ( ( $仮引数名$ ) )
        *IF(front.eq.0)
            call errsgn('要素が無い')
        *END-IF
        ( $仮引数名$ = $箱名$ (rear) )
        *IF(front.eq.0)
            call errsgn('待ち行列が不正')
        *END-IF
        *IF(front.eq.rear)
            front=0
            rear=0
        *END-IF
        *IF(front.eq.size)
            front=1
        *ELSE
            front=front+1
        *END-IF
      RETURN
$END-MNSCTN
$END-$SUB
$END-PGM
```

図1：モジュールの例（配列型の待ち行列）

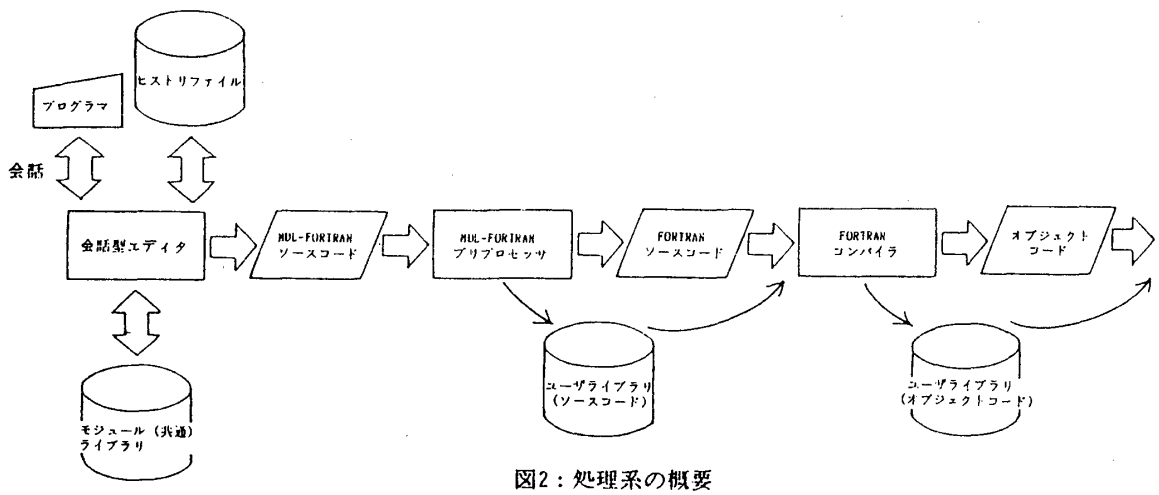

図2：処理系の概要