入出力データ構造に基づく プログラミングシステムの開発

三 河 佳 紀*

Development of a Programming System Based on Input-Output Data Structure

Yoshinori MIKAWA

要 旨

入力・出力データ構造に基づいたプログラミングシステムを開発した。本システムでは入力データ構造のみを用いることによりプログラミングが可能である。すなわち入力文(read文)は不要であり，JSPのようなデータ構造設計の手法を用いて入力（出力）データ構造を設計することにより直接プログラミングを行う。今回開発した本システムのプログラムは，C言語の表記を基に，ダイクストラの護衛付きコマンドに似た文で記述され，前処理としてCの原始プログラムに変換された後，実行される方式をとっている。このシステムの利用者はまず第一に入力データ構造を分析することが大切であり，さらにこのシステムで記述するIOSPプログラムを書くために計算手順を理解しなければならない。本報ではこのシステムにおけるプログラミングの表現方法と本システムの内部処理について報告している。

Abstract

The author has developed a programming system based on input-output data structure. In this programming system a program can only be made by using input data structures. Read statements are unnecessary. This means that a program can be made after constructing input (and output) stream structures using a method such as JSP(Jackson Structured Programming). Program expressions for this system are based on C language, and include a statement similar to a Guarded Command (E. W. Dijkstra). These programs are then translated into C language and subsequently compiled and executed. A programmer should primarily analyze the input data structure and then come up with a computational procedure for writing an IOSP program. In this paper some program expression methods for this system, and the inner processes within the system, are reported.

1. はじめに

ある業務について一つのシステム化を考えた場合，それはさまざまな要求に対してのプログラムを複数作成し，それを有機的に結合したものになる。一つの要求からそれに対する設計仕様がなされ，ある原理に基づいたプログラムが作成される。

ある業務に関するプログラムの仕様書においては，一度作成した後は仕様書の変更が一切無いという，いわゆる仕様書の固定化[1]が理想であろう。仕様書が固定化された場合，プログラムのメンテナンスなどは一切不要になるかもしれない。しかしながらシステム要求の変更ということは必ず起こり得ることであり，また業務を進める上ではそれが起こらなければ不自然である。仕様書の変更が可能だということは仕様書の更新，あるいはその仕様書に基づいて作成されたプログラムのメンテナンスにかなりの労力が必要とされることを意味する。しかし，かなりの労力が必要になるからと言って仕様書を変更不能にするわけにはいかない。そうであるなら，仕様書の更新に伴って

* 助 手 情報工学科

円滑にプログラムのメンテナンスも行えるような工夫をあらかじめ用意してゆく方が賢明である。その工夫の一つとしてシステム要求の中に存在する，ある原理を見い出すことが上げられる。たとえば扱うデータに着目した場合，それらはそれぞれデータの構造に基づいた原理をもっている。入力データがどのようなデータ構造であるか，また出力データがあるならば，出力データがどのようなデータ構造に基づいているかということを念頭にプログラムを作成することが大切である。

筆者はこのような入出力のデータ構造に着目し，ジャクソン法のようなデータ構造設計によって入力（出力）データの構造を直接プログラミング可能なシステムIOSP（Input-Output Structured Programming)の開発を行った。このシステムでは入力文を記述することなく入力データ構造にのみ基づいてプログラミングが可能である。また，プログラムはC言語の表記を基に，ダイクストラの護衛付きコマンド的な文を記述し，前処理によりCの原始プログラムに変換した後，実行に移される形態をとっている。本報ではこのシステムにおけるプログラムの表現方法などについて報告する。

2．入出力データ構造

入出力データ構造に基づいたプログラムの作成では入力あるいは出力ファイルの間に存在する原理を見い出すことが大切である。たとえば入力ファイルから出力ファイルを作成するプログラムを考えた場合それら二つのファイル間には，ある一定の順序や構造の対応付けが可能となる。これら入出力データを構造図で表し，さらにその中より入力データ出力データの関係を見い出し，その状態からプログラムを作成すると良い。この入力ファイルから出力ファイルを得るプログラムについては視点を変えると，入力データが列挙されたファイルをソースプログラムAと考え，出力ファイルはソースプログラムAより新たに作成されるソースプログラムBと考えることが出来る。これは高レベル言語で書かれたプログラムを機械語あるいはアセンブラ言語などの，より低レベルな言語に翻訳するプログラムすなわちコンパイラとして考えることが出来る。このような見地に基づいた構造化プログラミングの設計方法としてはM. A. ジャクソン（Michael A. Jackson)が開発したジャクソン法（Jackson Structured Programming:JSP)[2)3)4)5)]が上げられる。このジャクソン法（以下JSPと記す）は事務計算のように入出力データ間に構造の対応付けがある場合のプログラム等に於いて非常に有効な設計方法とされている。この手法は，入出力データ構造に基づいた構造を持つプログラムを作成しようというものである。このJSPではプログラムの組み立てに関してジャクソンの第1原理と呼ばれる次の4つの手順を用意している。[2)3)]

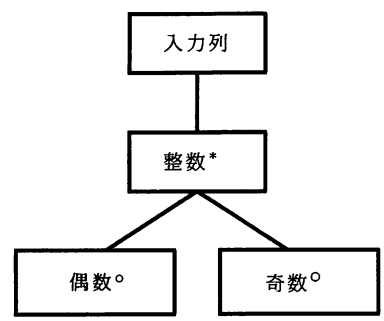
- 入出力ファイルを木構造図によって定義する（入出力データ構造設計）
- 入力から出力への対応が正しく成立するかを確認する（プログラムの構造設計）
- プログラムに必要な手続きの列挙
- プログラムの組み立て

筆者の開発したプログラミングシステムIOSPもこのJSPに基づいている。次にこのIOSPのプログラムの組み立てを例を上げて解説する。

例1：データとして整数値を入力し，その値が偶数か奇数かを判定しそれぞれの個数をカウントし出力する。

この例における手順の第1段階としての入力データ構造については図－1のように，入力列として整数の繰り返しであり，それはそれぞれ単位記録の列である偶数あるいは奇数の選択として表すことが出来る。さらに出力データ構造については図－2のように出力列として判定の繰り返しと個数の出力であり，判定の繰り返しについては偶数の表示あるいは奇数の表示の選択として表すことができる。

さらに手順の第2段階として入出力データ構造の設計から入力データ項目が出力データのどの項目に変換されるのかを調査する。JSPではプログラムの構成については入力レコードを1つ先読みし，後へ戻ることなしに処理手順を決定しなけれ



図－1　入力データ構造(例1)

ばならない。すなわち瞬時に入力レコードに対して出力レコードを作成するのである。第2段階ではこれらのことが可能か（入力と出力の対応関係があるか）を調査することになり，最終的には入力データ構造の木から出力データ構造の木へたどり着けるかを確認することになる。この場合，入力・出力構造図に示される箱の間の対応において次のことが満足しているかが重要になる。[2)]

・2つの箱が同じ個数のデータを表す
・2つの箱のデータが同じ順序で出現
・2つの箱のデータを一対として扱う処理の存在

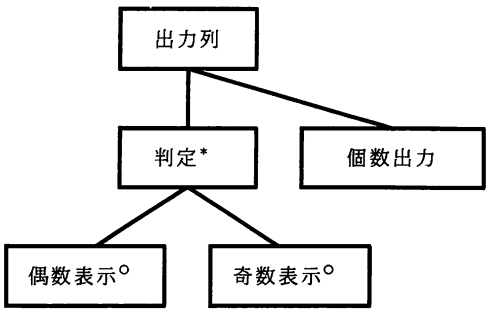
例1については図－1，図－2を重ね合わせることにより図－3のように表すことが出来る。ここでは得られた図が，入力構造図，出力構造図の両方を含んでおり，上述の条件が満足し入力データ構造の木から出力データ構造の木への変換が行われたことが確認できる。この様にプログラム作成においては入力データ構造・出力データ構造に基づく入力データと出力データのプロセスの対応関係が重要になる。

次に手順の第3段階においては，具体的な手続きを列挙する。ここでは実際のプログラムの命令文の割り付けなどを行うことになる。この例1における具体的な手続きとして以下のものが上げられる。なおIOSPではC言語の表記法を基にプログラムを記述するので〈 〉内にはC言語の具体的な命令を記述した。

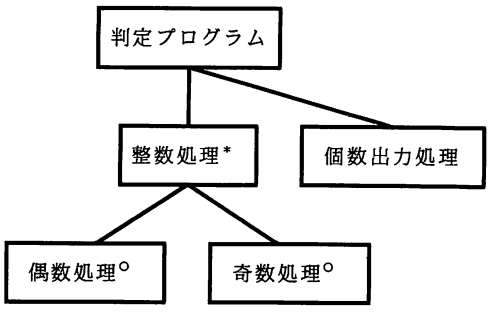
(1)偶数の個数を出力
　〈 printf("Even Number＝%d¥n",cnt)； 〉
(2)奇数の個数を出力
　〈 printf("Odd Number＝%d¥n",cnt2)； 〉
(3)偶数の個数をカウント
　〈 cnt＋＋； 〉
(4)奇数の個数をカウント
　〈 cnt2＋＋； 〉
(5)偶数データの識別表示
　〈 printf("an Even Number.¥n")； 〉
(6)奇数データの識別表示
　〈 printf("an Odd Number.¥n")； 〉
(7)偶数の個数をカウントする変数 ←0
　〈 cnt＝0； 〉
(8)奇数の個数をカウントする変数 ←0
　〈 cnt2＝0； 〉

手順の第4段階はプログラムの組み立てである。ここでは入力データ構造の木に葉の補足追加をし，プログラムの木を構成する。さらにこのプログラムの木に第3段階で列挙した具体的な手続きを埋め込む。実際に埋め込んだ状態を図－4に示す。

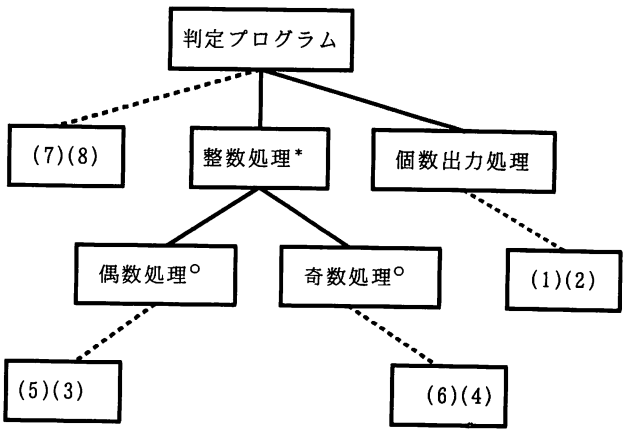
通常はこの段階で入力に関する命令が補足追加されるが，IOSPではread命令のような入力命令は一切追加しなくても良い。これはIOSPは入力命令を意識せず，あくまでも構造図で得られた構造を用いることによりプログラムを作成するためである。このことはIOSPのプログラム記述における大きな特徴の1つである。



図－2　出力データ構造(例1)



図－3　得られたプログラム構造図(例1)



図－4　コード割り付け(例1)

3．プログラムの記述と実行

JSPの手法では段階的に手順を踏みながらプログラムを作成する。筆者もJSPの考え方を踏まえ,「入力・出力のデータ構造を定めることによりプログラムが作成出来る」という概念に基づきIOSPの開発を行った。IOSPのプログラムに於いては構造図で得られた構造をいかにそのまま表現するかが問題になる。さらに記述方法としては記述が扱い易いこと，視覚的に見てもプログラム的であることなどを念頭に置かなければいけない。それらを考慮して次の表現方法を試みた。この方法は入出力データ構造に基づいて作成されたプログラムの木に基づいてダイクストラ的表現[6]でプログラムを記述するものである。ダイクストラ構造のプログラムでは「護衛付き命令」といわれるものがあり，これは次のように表すことができる。

〈護衛付き命令〉::＝〈護衛〉－〉〈文の並び〉

ここでの護衛は論理式などである。

たとえば＊n％2＝＝0－〉{printf (”an even number.”);} のように＊n%2＝＝0に対して文の並びがあるというものである。例1から導き出された図－4の状態をIOSPのプログラムとして記述したものを図－5に示す。

図－5における例1のIOSPプログラムに関して解説する。in～niの間には入力データにおける変数名を記述（入力変数宣言）する。ado～odaについては後判定の反復文を意味する。これはC言語のdo～while文にあたり,adoif～fiodaについては後判定反復文の継続条件式であり，正常なデータとしてあり得ない値や文字などのストッパ[2]を記述している。＊n%2＝＝0－〉{･･･}は護衛付文である。(護衛に対して文の並びがある)

```
{int cnt<-0;}
{int cnt2<-0;}
in
   n
ni
ado
    *n%2==0->{printf("an even number.¥n");}{cnt++;}
   |*n%2!=0->{printf("an odd number.¥n");}{cnt2++;}
oda
adoif
   *++n!='¥0'
fioda
{printf("even number =%d¥n",cnt);}
{printf("odd number  =%d¥n",cnt2);}
```

図－5　IOSPでの記述(例1)

{･･･} には文の並びを記述する。

ここでIOSPプログラムの主な構文について拡張BNFで記述する。

〈IOSPプログラム〉::＝
　〈状態変数宣言〉* 〈入力変数宣言〉〈文〉*

〈状態変数宣言〉::＝ {〈型〉〈単変数並び〉;}

〈単変数部〉::＝〈変数〉|〈変数〉〈- 〈初期値〉

〈単変数並び〉::＝
　〈単変数部〉 |〈単変数部〉,〈単変数並び〉

〈入力変数宣言〉::＝in　〈変数並び〉　ni

〈変数並び〉::＝〈変数〉|〈変数〉,〈変数並び〉

〈単純文〉::＝〈代入文〉;|〈出力文〉;

〈複合文〉::＝ {〈単純文〉*}

〈反復文〉::＝
　ado
　　〈文〉*
　oda
　adoif　〈条件式〉　fioda

〈護衛付文or〉::＝〈護衛付文〉 |
　　〈護衛付文〉'|'〈護衛付文の並び〉 |

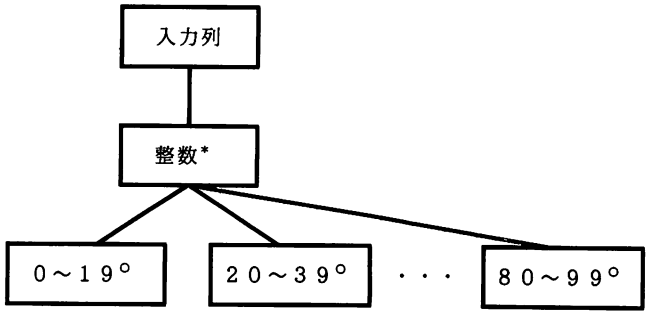
〈護衛付文〉::＝〈条件式〉－〉〈文〉*

〈文〉::＝〈単純文〉|〈複合文〉|〈反復文〉
　　|〈護衛付文or〉

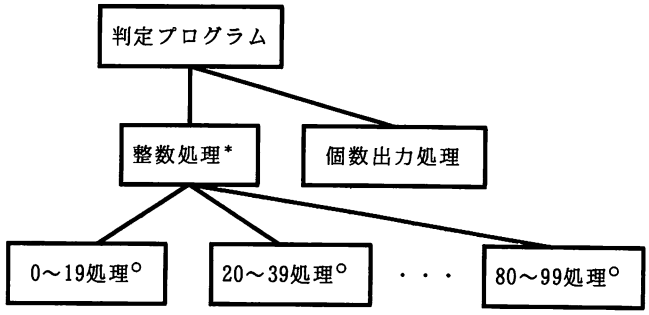
なお，ここでの〈代入文〉はC言語で許される変数への代入を表す文であり，〈変数〉＝〈式〉の表現や〈変数〉＋＋，〈変数〉--などである。また〈出力文〉はC言語で許される出力文でありprintf(〈式の並び〉) などである。

これらIOSPの構文を用いて次の例2についてプログラムを作成してみる。

例2：0～99までの整数値について度数分布表(きざみ20) を作成する。



図－6　入力データ構造(例2)



図－7　得られたプログラム構造図(例2)

この度数分布表を作成する例題では入力データ構造として図－6のように表すことができる。すなわち入力列として整数の繰り返しであり，その整数は0から始まるきざみ幅20毎の単位記録の列の選択である。出力データ構造図は省略しているが，この2つの構造図を重ね合わせて得られるプログラム構造図は図－7のようになる。

得られたプログラム構造図からIOSPのプログラムは図－8のように記述することができる。IOSPの構文として〈入力変数宣言〉,〈反復文〉,〈護衛付文or〉,〈単純文〉などを使って記述しているのが読みとれる。また，ここでも入力文の記述は全く意識せず，コーディングすることが出来た。IOSPのプログラムでは入力（出力）データの動作を記述したものではなく，あくまでもデータ構造図に基づいた入力（出力）データの順番を記述しているので一般的な入力命令（read文）等は一切不要なのである。

このシステムにおいては最終的にC言語で処理することを前提にしているためIOSPのプログラム作成後は一旦，自動的にCの原始プログラムに置き換えられその後実行する形態を取っている。

入力の木と出力の木の対応付けを考えると，レコードを先読みし，次のレコード読んだ時に出力の木に対応するレコードを作り出すことが必要になる。このことからIOSPプログラムからCの原始

```
{int cnt1<-0,cnt2<-0,cnt3<-0,cnt4<-0,cnt5<-0;}
in
    n
ni
ado
    *n/20==0->{cnt1++;}
    |*n/20==1->{cnt2++;}
    |*n/20==2->{cnt3++;}
    |*n/20==3->{cnt4++;}
    |*n/20==4->{cnt5++;}
oda
adoif
    *++n!='¥0'
fioda
{printf(" 0-19 = %d¥n",cnt1);}
{printf("20-39 = %d¥n",cnt2);}
{printf("40-59 = %d¥n",cnt3);}
{printf("60-79 = %d¥n",cnt4);}
{printf("80-99 = %d¥n",cnt5);}
```

図－8　IOSPでの記述(例2)

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

FILE *fp;
int *p,*head,*n;
int c1[500];
int c;

void main(void)
{
        int cnt=0;
        int cnt2=0;
        fp=fopen("idata1","r");
        p=head=n=c1;
        while((fscanf(fp,"%d",&c)) !=EOF){
                *n=c;
                n++;
        }
        *n='¥0';
        n=head;
        do {
            if(*n%2==0){
                printf("an even number.¥n");
                cnt++;
            } else if(*n%2!=0){
                        printf("an odd number.¥n");
                        cnt2++;
                    }
        }while(*++n!='¥0');
        printf("even number =%d¥n",cnt);
        printf("odd number  =%d¥n",cnt2);
        fclose(fp);
}
```

図－9　IOSPによるC言語への変換

```
10 14 25 47 56 49 87 68 99 100 70


an even number.
an even number.
an odd number.
an odd number.
an even number.
an odd number.
an odd number.
an even number.
an odd number.
an even number.
an even number.
    even number =6
    odd number  =5
```

図－10　実行結果（例１）

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

FILE *fp;
int *p,*head,*n;
int c1[500];
int c;

void main(void)
{
        int cnt1=0,cnt2=0,cnt3=0,cnt4=0,cnt5=0;
        fp=fopen("idata2","r");
        p=head=n=c1;
        while((fscanf(fp,"%d",&c)) !=EOF){
                *n=c;
                n++;
        }
        *n='¥0';
        n=head;
        do {
          if(*n/20==0){
             cnt1++;
          } else if(*n/20==1){
                    cnt2++;
                  } else if(*n/20==2){
                            cnt3++;
                          } else if(*n/20==3){
                                    cnt4++;
                                  } else if(*n/20==4){
                                            cnt5++;
                                          }
        }while(*++n!='¥0');
        printf(" 0-19 = %d¥n",cnt1);
        printf("20-39 = %d¥n",cnt2);
        printf("40-59 = %d¥n",cnt3);
        printf("60-79 = %d¥n",cnt4);
        printf("80-99 = %d¥n",cnt5);
        fclose(fp);
}
```

図－11　IOSPによるC言語への変換

```
1 25 36 66 98 58 45 54 55 65 28
36 42 47 90 87 85 84 14 2 3 4


 0-19 = 5
20-39 = 4
40-59 = 6
60-79 = 2
80-99 = 5
```

図－12　実行結果（例２）

プログラムへの置き換えの過程では入力レコードの先読みの技法が大切になる。これは事務計算に向いているといわれるCOBOL言語におけるread文のようにファイルのオープン直後にはじめのレコードを読むことと同様である。これらを表現する場合，先読みした入力レコードは環状あるいは鎖状のバッファなどに蓄えれば円滑に動作するため，置き換えの過程ではこの点も考慮し作成している。

IOSPの実行は次のようにコマンドラインより行う。

IOSP ソースリスト名　〈入力データ〉

ソースリスト名はデータ構造図に基づいて作成されたIOSPのプログラム名である。入力データについてはあらかじめファイルが用意されていればそのファイル名を指定し，標準入力からのデータの入力であればstdinと指定する。このコマンドの入力を行うと一旦C言語の原始プログラムに自動変換され，その後実行される。IOSPでの手順は以下のとおりである。

- データ構造図に基づくIOSPでのプログラムの記述
- C言語の原始プログラムへの変換
- コンパイル・リンク
- プログラムの実行

例１における変換されたC言語のリストと入力データ及び出力結果をそれぞれ図－９，図－10に示す。また例２の変換されたC言語のリストと入力されたデータ及び実行結果を，それぞれ図－11，図－12に示す。

4．システムの有効性

ここでは今回開発したシステムの有効性について評価する。

プログラムの設計についてはさまざまな方法論があり，たとえば機能分解を重ねることで設計を行う機能的分解法，比較的に直線的なプログラム構造に到達するデータフロー設計法，データ構造図や図式的なロジックの両方をドキュメンテーションの永久的要素として保存可能なデータ構造設計法などが上げられる。本システムはJSPに代表されるデータ構造設計法に基づいて，入力及び出力のデータ構造に着目し，それらを明確に定めることによりプログラムを作成し実行する。これには入力データと出力データのプロセスの対応関係が重要なポイントとなっている。入力をある単位記録の列と考えた場合，それぞれの単位記録は複数のパターンをもつ。出力はこれら単位記録に対応したものにならなければいけない。この考えでは例2においては当然0～99までの入力データに対応した出力が得られなければならない。入力がこの範囲の数値入力だけであれば図－8に示した記述だけで問題はない。しかしながら，まれにこの範囲外のデータが入力されることも想定しなければいけない。たとえば100の入力を考える。この場合，現行のままではこのデータに対応していないのは図－8の記述より明らかである。

データ構造設計法ではプログラム修正＝構造図の更新でもあるので，本システムでは修正方法として構造図にその入力データの判定をつけ加えることにする。すなわち入力ストリームから取り込まれたデータが判定要素に満足しなければそれを入力ストリームに戻し，次の新しいデータを取り込めば良いことになる。このように行うことにより木構造を破壊することなく修正追加が可能になる。　ここで例2においてこの部分を追加記述したリストを図－13に示す。この追加記述は〈護衛付文or〉の構文を使うことにより行っている。すなわち，ここでは前からの判定の流れに，入力エラーに対する新たな判定を付け加えている。

このような修正追加を行った場合，データフロー設計法では変更が生じた場合は木構造破壊を起こすことも考えられ，機能的分解法では入力データについての機能分解にまで至らないこともあり，これは問題構造とプログラムの構造の対応付けが取れないことにもつながる。それに対し，データ構造設計法に基づく本システムでは構造図をそのまま記述し実行するので，プログラム修正とドキュメントのメンテナンスも同時に行うことも可能であり，作業的な負担もかなり軽減される。

```
{int cnt1<-0,cnt2<-0,cnt3<-0,cnt4<-0,cnt5<-0;}
{int cnt6<-0;}
in
   n
ni
ado
    *n/20==0->{cnt1++;}
    |*n/20==1->{cnt2++;}
    |*n/20==2->{cnt3++;}
    |*n/20==3->{cnt4++;}
    |*n/20==4->{cnt5++;}
    |*n!='¥0'->{cnt6++;}
oda
adoif
    *++n!='¥0'
fioda
{printf(" 0-19 = %d¥n",cnt1);}
{printf("20-39 = %d¥n",cnt2);}
{printf("40-59 = %d¥n",cnt3);}
{printf("60-79 = %d¥n",cnt4);}
{printf("80-99 = %d¥n",cnt5);}
{printf("input error = %d¥n",cnt6);}
```

図－13　IOSPでの記述（例2改良版）

本システムでのIOSPプログラムの記述は一般のプログラミング言語とは異なり，あくまでも入出力データ構造に基づき「データがこのような状態であればこのように処理する」ということに主眼を置いている為，特にデータをreadする（入力文）ということについては意識しなくても良い。このことはプログラムの設計をデータ構造に基づいて行うことが可能であるのを意味し，設計後は必要な命令を順に組み込んで行けば良いことになる。これらは通常のプログラムと大いに異なる点であろう。また記述したプログラムからC言語の原始プログラムを自動生成することはIOSPで記述されたプログラムの検証さらにはドキュメントの充実などにつながり，これも大きな特徴として上げられる。

5. おわりに

ただ闇雲に仕様書通りのプログラムを大切な原理を無視してつぎはぎだらけで作成した場合，仕様の変更が生じてからのプログラムのメンテナンスは非常に困難なものになる。そこでこれらを回避する為に筆者は入力（出力）データ構造に着目し，それに基づいて得られるデータ構造図をそのままプログラムとして記述し実行するシステムIOSPの開発を試みた。1つの大きなプロセスを構造化分析することにより，さらに細分化する。その細分化された中の1番小さなプロセスに着目す

ると，その一つ一つの構造は入力と出力の関係になっている。このことを考えると本システムの発想を発展させることはプログラムの再利用，プログラムの部品化[7][8]，さらには類似アプリケーションへの再利用[8]へとつながるはずである。本システムは再利用，部品化への確立されたプログラミングシステムと成るべく更に機能追加などの改良を重ねることが今後の課題である。

謝　辞

本研究を進めるに当たり有益な御助言を与えて下さった北海道情報大学経営情報学部情報学科
林　雄二助教授に深く感謝致します。

参考文献

1）トム・デマルコ：構造化分析とシステム仕様，日経BP出版センター，1994
2）林雄二：プログラム設計の基礎，森北出版，1993
3）米口肇：ジャクソン法，bit, vol. 12, No11, pp. 81-88, 1980
4）米口肇：ジャクソン法，bit, vol. 12, No12, pp. 61-72, 1980
5）G. D. Bergland：構造化設計法，bit増刊，ソフトウェア・ツール，pp.15-39, 1982
6）E. W. ダイクストラ・W. H. J. フェイエン，ダイクストラ　プログラミングの方法，サイエンス社，1991
7）林雄二：データフローネットワークを基礎にしたプログラミング言語の開発，信学技法，SS95-10, pp.1-8, 1995
8）中谷多哉子・永田守男：例題によるオブジェクト指向のモデル化作法，情報処理，vol. 35, No. 5, pp.402-411, 1994

（平成7年11月28日受理）