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（2）

三　河　佳　紀*・林　　雄　二**

Development of a Programming System Based on Input-Output Data Structure (2)

Yoshinori MIKAWA and Yuji HAYASHI

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IOSPの開発を行ってきた。このシステムのプログラム表現はC言語の表記を基にしており，IOSPプログラムで書かれたソースプログラムはC言語に変換された後，実行される。著者らはこのシステムにおけるプログラムの概念と内部処理の一部について改訂を行った。この改訂後のシステムでは、それぞれの連続的な入力データが先行するステップで認識でき（先読み），さらに読み込んで，次のステップをスタックすることも可能である。この認識によりプログラムはデータ構造と一致し，余分な入力文無しで記述することが可能である。本報ではこのシステムの有効性について典型的なコントロールブレークの例を通して述べる。

Abstract

We have developed a programming system based on input-output data structure. This system is called IOSP. Program expressions for this system are based on C language. The system translates a source program written in an IOSP program into the C language. These programs are subsequently compiled and executed.

We have revised some program notation for this system and the inner processes within the system. In the revised system, each consecutive input data is able to be recognized in the preceding step, and then read and stacked in the next step. With this recognition, the program coincides with its data structure can be written without inserting redundant previous input statements.

The advantages of this system are discussed and shown through a typical example, including control breaks.

1．はじめに

一つのプログラムを作成する場合，適切な仕様を得ることが必要であるが，それには問題の構造を正しく把握するとともに，与えられた要求仕様から適切な原理を見い出すことが大切である[1)2)]。

著者らはすでに扱う入力・出力のデータ構造に着目し，ジャクソン法に似たデータ構造設計によって得られるデータ構造図を直接プログラムとして記述・実行するプログラミングシステムIOSP（Input-Output Structured Programming）の開発を行ってきた[2)]。

このシステムにおけるプログラムの構成は，入力レコードを事前に認識し，後戻りなしに処理手順を決定するという概念に基づいている。これは入力レコードに対して瞬時に出力レコードを作成することに繋がるものであり，プログラムとデータ構造が一致していることを意味する。入力レコードを事前に認識（データの先読み）することにより，プログラムは従来の余分な入力文を用いなくても記述が可能になる。本システムではこれらの点について，プログラム表現方法及び内部処理の機能更新を行った。具体的には入力データの先読み機能をプログラムの表記に盛り込んだことが上げられる。本報ではこのシステムの有効性に

＊　助　手　情報工学科
＊＊　助教授　北海道情報大学
　　経営情報学部情報学科

ついて，典型的なコントロールブレークの例を用い，通常のプログラムとの比較検証を行ったので報告する。

2．IOSPの概要

プログラミングシステムIOSPはM.A.ジャクソン（Michael A.Jackson)が開発した構造化プログラミングの設計手法の1つであるジャクソン法[3)4)5)6)]（Jackson Structured Programming：JSP)に基づいて作成されている。ジャクソン法(以下JSPと記す）はプログラムの組み立てにおいては4段階（入出力データ構造設計・プログラムの構造設計・手続きの列挙・コード割り付け）の作業工程を経る[3)4)]。

JSPは入出力データ構造に基づいた構造を得ることによりプログラムを作成するのである。IOSPも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。IOSPは入力構造図と出力構造図を重ね合わせることにより新たな構造図を得，この構造図を基にプログラムを作成している。IOSPのプログラムは，構造図で得られた構造をいかにそのまま表現するかが大きなポイントとなる。IOSPのプログラムの記述（図1）は，C言語の表記を基にダイクストラ[7)]の護衛付きコマンド的な文を記述する。

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

図1　IOSPプログラム

これらは幾つかの基本構文により構成されている。IOSPのプログラムの特徴として，入力(出力)データの動作を記述するのではなく，あくまで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ことが上げられる。さらに他のプログラミング言語と比較した場合，IOSPの大きな特徴として入力命令が不要なことが上げられる。すなわちread文のような命令が一切不要である。これは構造図で得られた構造をそのまま用いることによりプログラムを作成するためである。このことはプログラム設計をデータ構造に基づいて行うことが可能であることを意味している。本システムにおいては，IOSPで記述されたプログラムが最終的にC言語で処理されることになる。すなわちIOSPプログラムの作成後，Cの原始プログラムを一旦自動生成し，その後実行する形式を取っている。これらについても記述されたIOSPプログラムの検証あるいはドキュメントの充実[1)]などにも繋がり，本システムの特徴と言えよう。

3．機能更新

先に開発したIOSPプログラムの表現方法と内部処理について一部機能更新を行った。プログラムの表現方法については，従来の主な構文のうち〈反復文〉について一部改良を行った。

従来の反復文の構文では次のように表現していた。

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

このように従来の構文では，後判定反復文の継続条件式を独立させた形で表していたが，改良後はado～oda間に含有する形に変更した。

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

このことにより，従来想定していなかった多重反復も可能になり，プログラム記述においても操作性が向上した。

ここで最新のIOSPプログラムの主な構文について拡張ＢＮＦで記述する。

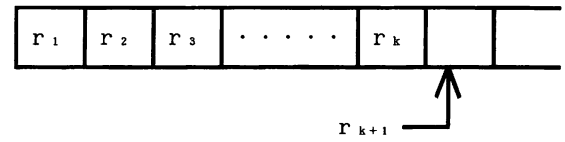
```
〈IOSP プログラム〉::=
  〈状態変数宣言〉*〈入力変数宣言〉〈文〉*

  〈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〉*
          〈条件式〉
      oda

  〈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〉
```

プログラムの構成においては入力レコードを先読みし，後戻りなしに処理手順を決めなければいけない。これは瞬時に入力レコードに対して出力レコードを作成することに繋がる。IOSP では一旦Cの原始プログラムへの置き換えを行い実行する形式を取るが，この過程では入力レコードの先読みの技法が重要になってくる。

IOSP プログラムにおける先読み処理については，必要により1つ先，2つ先，3つ先のようにデータを先読みするケースも考えられるため，これらについても改良を行った。IOSP プログラムにおける入力データの先読みに関しての記述は，独特の表現を用いることにした。例えば次のようにIOSP プログラム中で記述された場合，r は入力変数として認識される。

```
in
    r
ni
```

この入力変数 r を先読みした場合，どのように表現するかであるが、表記法として入力変数の前に $ を付与することにより対処した。すなわち $ r と記述した場合には，次に入力可能なデータを表し，付与する $ が増えるとその増えた分だけ先の次に入力可能なデータを示すことになる。これら $ の付与された入力変数はデータの入力動作を伴うものではなく，あくまでも $ r の後の r が本来の入力となる。r という変数と配列を用いてこれを IOSP の入力済みスタックと表せば，次のようになる。



この場合，$ r では入力済みスタックには加えられず，r により初めて r_{k+1} として入力済みスタックに加えられることになる。

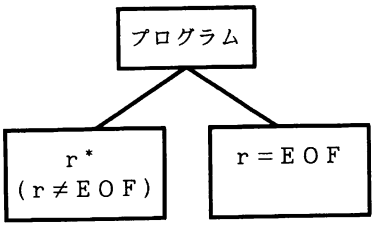
一般的に複数のデータ入力を扱うプログラムにおいては次のように記述する場合が多い。

```
scanf ("%d",&r);
 while (r!=EOF){
          処理
              scanf ("%d",&r);
 }
```

すなわち反復文の以前に，前処理としてデータを読み込む必要がある。しかし，入力データの先読みが可能であるならば，次のように表すことができる。(IOSP プログラムで記述)

```
ado
 $r!=EOF-> {r} {処理}
oda
 {r}
```

この記述は次に示す入力データに関するジャクソン構造図と同様な構造である。



このようにプログラム記述の際には入力文(read 文) を用いない関係上，入力データを参照する場合には入力変数を用い表現することにし，先読みを行う場合にはプログラム上で定義された入力変数の先頭に＄を付与することにより表現するのである。実際の IOSP プログラムでの記述は図１のとおりである。

内部処理においてＣの原始プログラムに置き換えを行う際は，先読みしたレコードを環状あるいは鎖状のバッファなどに蓄えればよく，Ｃ言語への置き換えの過程では実際には線形リスト構造を利用した。この結果，従来の IOSP プログラムに比べた場合，プログラム記述上もかなりの自由度が増した。このように入力可能なデータを先読みすることを強化することは，Prolog システムに組み込まれているような，バックトラック(backtracking ：自動後戻り) に相当し，IOSP システムとしてもより有効な処理が可能になった。

4．IOSP プログラムの比較検証

IOSP プログラムの有効性について他のプログラムとの比較を行ってみる。

たとえば次のような業務プログラムを考える。

＜例題：ある鉄道会社のプログラム＞

全国規模の鉄道会社Ａにおいて，１年間の車両検修（検査及び修繕）のデータがある。このデータは次のようなレコードからなる。

１）車両工場のコード（例1000　札幌工場）
２）全般検査の車両台数（車検のようなもの）
３）要部検査の車両台数（６ヶ月点検　〃　）
４）臨時検査の車両台数（突発的な検査〃　）
５）２）～５）に携わった人員数

これらのレコードがデータとして複数入力（図２）される。同一の車両工場コードは複数存在する。データのストッパは車両工場コードが０である。このデータを用いて各工場毎の年間の各種検査毎の検修車両台数と検修に携わった延べ人員を集計する。

この問題はコントロールブレークの問題でもある。この場合コントロールグループはこれらのレコードであり，コントロールキーは車両工場コードである。すなわちコントロールキーの車両工場コードが変化するのを判断するには，次の車両工場コードあるいはストッパとして０が入力された時である。プログラムを作成する場合の一般的なコントロールブレークの検出方法としては大まかに次のような手順を踏む。

・作業領域の確保
・作業領域へのコントロールキーの移動と比較

この例題を概要ＰＡＤ図で表せば図３のようになる。また，通常のＣのプログラムでこの例題プログラムを記述した場合，例えば図４のように表され，その実行結果は図５に示すとおりである。

IOSP においてこの例題プログラムを組立ててみる。IOSP は JSP と同様にプログラムを作成する過程には４段階の手順を経る。第１段階では入出力データ構造設計について作業を行う。この例題の入力データ列において，工場コードのみに着目すると次のような構造になっている。

・・・・1700 1700 1800 1800 1900 1900 1900 0 EOF

同じ工場コードが複数存在するのが解る。また，

```
1000 18  8  2  252
1000 20  2  3  207
1000 13  2  1  302
1000 14  5  5  200
1000 10  9  6  189
1100 20  8 20  298
1100 50  0 33  350
1100 26  1 44  780
1200 38  1  5  569
1200 21  0 66  170
1200 36  2 25  520
1200 58 14 32 1102
1300 68 54  2 2102
1300 51  1  5  198
1300 12  3  7  158
1400 56 58  8 2500
1400 65 25  9 3600
1400 98 20 25 4021
1400 50 21 31  698
1500 98 56 42 6987
1500 25 65 22  321
1500 36 65 56  987
1500 25 32 56 1586
1600 32 25 25  458
1600 25 98  3  547
1600 36 98  4  251
1600 65 87 66  635
1700 58 98 99  778
1700 25 36 25 1900
1800 25  3 32  178
1800 23 25  2 2500
1900 35 56  9  189
1900 36 58 87  253
1900 58 56 25  320
1900 58 56 36 3200
0
```

図　2　車両検修データ

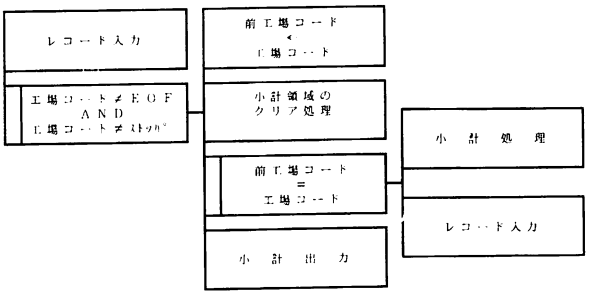


図３　コントロールブレークの検出

入力データ全体を通してみると，先頭レコードの後にその先頭レコードと同じレコードが複数続く形式であるのも解る。したがって，入力列は工場コードの繰り返しであり，検修レコードの繰り返しである。また，検修レコードの繰り返しは単位記録の列として全検データ，要検データ，臨検データ、人員データとして表すことが出来る。この入力データ構造及び出力データ構造をジャクソン構造図で表すとそれぞれ図6・図7のように表すことができる。

さらに第2段階のプログラム構造設計では第1段階で得られた図6と図7の構造図を重ね合わせることによりプログラム構造図が得られる。ここでは入力データ構造と出力データ構造のプロセスの対応関係が取れたことが確認出来る。第3段階と第4段階では，それぞれ手続きの列挙と，得られたプログラム構造図へのコード割り付けを行う。その結果得られたコード割付後のプログラム構造図を図8に示す。コード割り付けとしては次のようになる。

(1) レコード入力
(2) 小計クリア
(3) 小計加算
(4) 小計出力

このプログラム構造図を基にコーディングしたIOSPプログラムと実行結果は，それぞれ図9，図10に示す。

作成されたIOSPプログラム（図9）を解説する。〈単変数部〉の {int zcnt <-0,･･･} については作業変数の確保と初期化を行っている。〈入力変数宣言〉の in ～ ni 間には入力データについて記述している。ado ～ oda の〈反復文〉においては，〈護衛付文〉の構文を用いている。また反復文においては機能追加した多重反復を使用している。$code!=EOF の記述では，入力変数 code の先頭に$が1つ付与されているので，入力データを1つ先読みしていることを示している。その後，護衛付文の式を判定し，wcode をはじめとする作業用変数へ入力データの格納と初期化を行っている。この際，{wcode <-$code} の記述に見られるように，入力変数 code には$が付与されているため，code は入力動作を伴わないことを示している。ここでは1つ先読みした code の値を作業用変数に入れている。次に多重反復の内側の反復処理へと進む。ここでの護衛付文の式は wcode==code となっており，入力動作を伴う code と作業用変数 wcode に待避させた値の比較を行っている。式の判定の結果、偽であればコントロールブレーク発生時の処理を行う。真であれば小計の加算処理を行う。この際，{zcnt += zdata} のように zdata には$が付与されていないので入力動作を伴うことを示している。内側の反復におけるストッパ判定は1つ先読みした code と待避している wcode の値が異なる時である。内側の反復を終了した後は，コントロールブレーク発生時の処理である小計出力を行い，外側の反復に戻る。外側の反復のストッパ判定は1つ先読みした code が0の時である。

このIOSPプログラムを実行した結果は通常のプログラムと同様な結果（図10）を得ることになった。

このようにIOSPで記述したプログラムは，図8の構造図と比較してみると構造的に一致していることが確認できる。すなわち入力（出力）データ構造に基づいてプログラミングが可能であり、このことはIOSPの大きな特徴である。

次にIOSPで記述したプログラムと，先程のコントロールブレーク検出のPAD図（図3）について比較し，コントロールブレーク処理がIOSPにおいて有効なのか否かを検討してみる。例題のプログラムにおけるコントロールブレーク検出については，一般的に図3のような流れになる。このPAD図において繰り返し部分に着目すると，IOSPプログラムでも同様な動作をしているのが解る。しかし，レコード入力については多少異なる。IOSPではレコードを先読みし、その後判定する形態を取っているが、一般的には図3のように入力文などを用いてデータの入力を行った後に判定する形態を取る。しかしIOSPでは先読みの機能をプログラムの記述に盛り込んだため、余分な入力文の記述が不要である。この点は本システムの大きな特徴と言える。このようにIOSPプログラムの記述において入力データの順番を先読みという手法で捉えることが可能であることから，IOSPプログラムの記述は，コントロールブレークの問題においても有効な事が確認された。

IOSPシステムの内部処理においては，IOSPプログラムを一旦C言語の原始プログラムに変換し，その後実行する形態を取る。その際プログラム上の入力変数宣言では入力データ構造に基づいて変数が宣言される。これらの変数に対応する入力データは単一データの繰り返しあるいはコントロールブレーク処理を行う時のような複数データの繰り返しである。そのためどの様なデータの繰

り返しにも対応可能なようにC言語の変換の際には内部処理において自己参照構造体を用いて表すようにしている。また入力データについては先読みの手法を生かすため，前処理として全てのデータをリスト構造である自己参照構造体へ格納する。結果としてプログラム中で自由にデータを参照することが可能となっている。入出力データ構造図を基にして得られたプログラム構造図を，そのままプログラミングするIOSPプログラムでは，利点としてデータの入力におけるread文などを意識せず，あくまでデータの順番を記述していることや，ドキュメントの充実に繋がり，IOSPプログラムの正当性の検証にも繋がるC言語原始プログラムへの自動生成（図11）が上げられる。さらには記述するプログラムのステップ数が通常のプログラムに比して少なくて済むという大きな利点もある。例題のコントロールブレークの問題を通常のプログラム（図4）とIOSPプログラム（図9）で比べた場合，通常のプログラムでいうところのmain（）関数に到達するまでの宣言やユーザ関数定義関係の記述に関してはIOSPではわずか4ステップであり，main（）関数内の一連の処理に関してはわずかに11ステップであった。全体で23ステップも記述が短いという結果が得られた。

```
#include <stdio.h>
void main(void);
void read1(void);

FILE *fp;
int code,z_overhaul,y_overhaul;
int r_overhaul,worker;
int work_code=0,z_count=0;
int y_count=0,r_count=0,w_count=0;

void read1(void)
{
        fscanf(fp,"%d",&code);
        if(code!=0){
          fscanf(fp,"%d",&z_overhaul);
          fscanf(fp,"%d",&y_overhaul);
          fscanf(fp,"%d",&r_overhaul);
          fscanf(fp,"%d",&worker);
        }
}

void main(void)
{
        fp=fopen("tpdata5","r");
        printf(" code   zen   you   rin ninzu¥n");
        read1();
        while(code!=EOF && code!=0){
          work_code=code;
          z_count=y_count=r_count=w_count=0;
          while(work_code==code){
                  work_code=code;
                  z_count+=z_overhaul;
                  y_count+=y_overhaul;
                  r_count+=r_overhaul;
                  w_count+=worker;
                  read1();
          }
          printf("%5d %5d %5d ",work_code,z_count,y_count);
          printf("%5d %5d¥n",r_count,w_count);
        }
        fclose(fp);
}
```

図4　Cプログラムでの記述

code	zen	you	rin	ninzu
1000	75	26	17	1150
1100	96	9	97	1428
1200	153	17	128	2361
1300	131	58	14	2458
1400	269	124	73	10819
1500	184	218	176	9881
1600	158	308	98	1891
1700	83	134	124	2678
1800	48	28	34	2678
1900	187	226	157	3962

図 5　Cによる実行結果

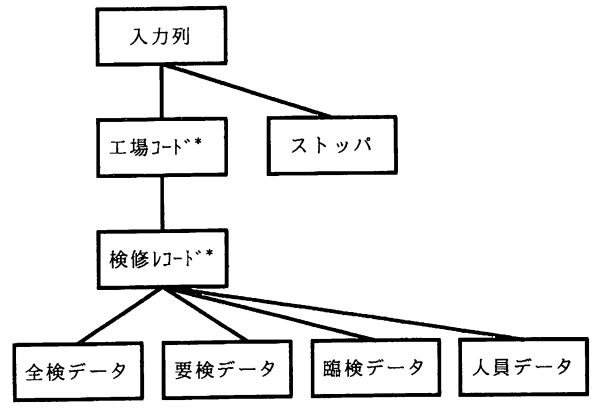


図 6　入力データ構造

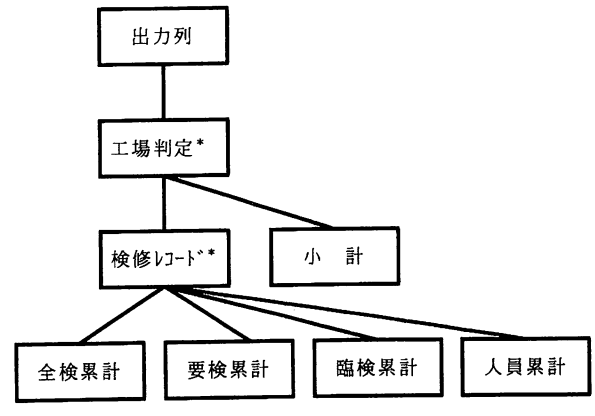


図 7　出力データ構造

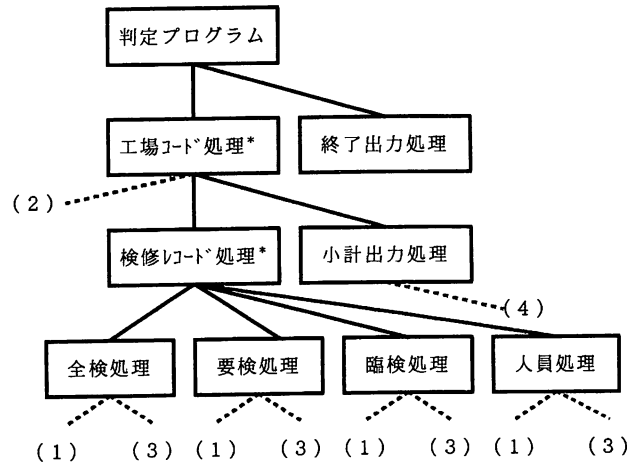


図 8　得られたプログラム構造図

```
{int zcnt<-0,ycnt<-0,rcnt<-0,wcnt<-0,wcode<-0}
in
 code zdata ydata rdata worker
ni
ado
  $code!=EOF->{wcode<-$code}{zcnt<-0}{ycnt<-0}{rcnt<-0}
              {wcnt<-0}
       ado
         wcode==code->{zcnt+=zdata}{ycnt+=ydata}{rcnt+=rdata}
                      {wcnt+=worker}
         (wcode==$code)
       oda
  {printf("%5d %5d %5d %5d %5d¥n",wcode,zcnt,ycnt,rcnt,wcnt)}
  ($code!=0)
oda
```

図9　IOSP でのプログラム

```
1000    75   26    17  1150
1100    96    9    97  1428
1200   153   17   128  2361
1300   131   58    14  2458
1400   269  124    73 10819
1500   184  218   176  9881
1600   158  308    98  1891
1700    83  134   124  2678
1800    48   28    34  2678
1900   187  226   157  3962
```

図10　IOSP による実行結果

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
void main(void);
void read1(void);
void read2(void);
void cpr(void);
struct cell{
        int code;
        int zdata;
        int ydata;
        int rdata;
        int worker;
        struct cell *next;
};
typedef struct cell CELL;
FILE *fp;
CELL *head,*cp,*sp,*ep;
int ar,br,cr,dr,er;

void read1(void)
{
        fscanf(fp,"%d",&ar);
        if(ar!=0){
                fscanf(fp,"%d",&br);
                fscanf(fp,"%d",&cr);
                fscanf(fp,"%d",&dr);
                fscanf(fp,"%d",&er);
        }
}

void cpr(void)
{
        if((cp=(CELL *)malloc(sizeof(CELL)))==NULL){
                printf("not memory");
                exit(1);
        }
}

void read2(void)
{
        while((fscanf(fp,"%d",&ar))!=EOF){
                fscanf(fp,"%d",&br);
                fscanf(fp,"%d",&cr);
                fscanf(fp,"%d",&dr);
                fscanf(fp,"%d",&er);
                cpr();
                ep->next=cp;
                ep=ep->next;
                cp->code=ar;
                cp->zdata=br;
                cp->ydata=cr;
                cp->rdata=dr;
                cp->worker=er;
                cp->next=NULL;
        }
}

void main(void)
{
        int zcnt=0,ycnt=0,rcnt=0,wcnt=0,wcode=0;
        fp=fopen("tpdata5","r");
        read1();
        cpr();
        head=cp=ep;
        cp->code=ar;
        cp->zdata=br;
        cp->ydata=cr;
        cp->rdata=dr;
        cp->worker=er;
        cp->next=NULL;
        read2();
        sp=head;
        do {
             if(sp->code!=EOF){
                wcode=sp->code;
                zcnt=0;
                ycnt=0;
                rcnt=0;
                wcnt=0;
             }
             do {
                  if(wcode==sp->code){
                        zcnt+=sp->zdata;
                        ycnt+=sp->ydata;
                        rcnt+=sp->rdata;
                        wcnt+=sp->worker;
                  }
                  sp=sp->next;
             }while(wcode==sp->code);
              printf("%5d %5d %5d ",wcode,zcnt,ycnt);
              printf("%5d %5d¥n",rcnt,wcnt);
        }while(sp->code!=0);
        fclose(fp);
}
```

図11　IOSP が自動生成した例題プログラム

5. おわりに

プログラムの設計におけるさまざまな方法論の中，著者らはデータ構造設計法に基づくIOSPシステムを作成し試行している。特に入力（出力）データ構造に基づく本システムでは得られた構造図をそのままプログラムとして記述し実行するため，プログラム修正時にドキュメント関係のメンテナンスも同時に行えるという利点もある。本システムでの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るところが多いが，入力文を意識しなくても良いという特徴がある。今回，IOSPプログラムの表現方法と内部処理についてさらに機能充実させるため，データの先読み技法や反復処理などについて改良を行った。その結果，業務プログラムに多く見られるような，コントロールブレークを含む問題について，IOSPプログラムと通常のプログラムを比較してみたが，入力（出力）データ構造図から得られたプログラム構造図が定まれば難なくIOSPプログラムを記述することが出来，実行結果についても同様の結果が得られることが確認された。さらには記述するプログラムのステップ数も通常のプログラムに比して非常に少ないという大きな利点も確認出来た。

IOSPプログラムは独自の構文を有し記述される。このことはプログラムの記述において，限られた構文や語句しか使えないといった欠点にも繋がる。しかしC言語を基に記述されるIOSPプログラムは実行と同時に自動的にC言語のソースプログラムを生成するため，既にC言語に慣れ親しんでいる人や，これからC言語を学習しようとする人には利点の方が優先するかもしれない。

プログラムの記述においては，Cの表現を一部採用しているが，今後は更に簡略化のため改良を進めなければいけない。また，入力（出力）データ構造に着目し作成された本システムの大きな目標はプログラムの部品化[8)]，プログラムの再利用[8)]であり，これらの実現に向け更なる改良を重ねていくことが今後の課題である。

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）三河佳紀：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，苫小牧工業高等専門学校紀要，第31号，pp53-60，1996
3）林雄二：プログラム設計の基礎，森北出版，1993
4）米口肇：ジャクソン法，bit，vol.12，No11，pp.81-88，1980
5）米口肇：ジャクソン法，bit，vol.12，No12，pp.61-72，1980
6）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
7）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
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
9）久野靖，言語プロセッサ，丸善株式会社，1993

（平成8年11月29日受理）