研究室希望配属先の申請のシステム化

02H077 中村健次

目次

1 はじめに 1

2 研究室振り分けシェルスクリプトの概要 2
2.1 研究室の振り分け方法 2
2.2 動作手順 2
2.3 使用するファイルの内容 3
2.4 実装 5
2.5 実行結果 7
2.6 実行結果の検証 9
2.7 インタフェース 10

3 まとめ 12
3.1 今後の課題 12

4 付録 13
4.1 研究室振り分けシェルスクリプトのソース 13
4.2 サンプルプログラム自動生成シェルスクリプトのソース 16

1　はじめに

近年、情報技術の進歩にともない世の中に存在する様々なものがシステム化されてきている。例えば、自動車教習所の教習予約システム、商品の在庫管理システムなどがそうである。これらのシステムに共通する事柄として、今まで人が手作業で同じ動作を繰り返し行ってきたもの自動化するということが挙げられる。繰り返し動作する部分をシステム化することで、人が行うよりも短時間で作業が終了し、ミスが無くなるというメリットがある。現在、大学の中にもシステム化した方が効率的になるものがある。その中で私は、卒業研究の研究室希望配属先の申請に注目した。

卒業研究の研究室配属先を決定するまでの過程は、まず生徒自身が希望配属先を専用の用紙に希望順を記入し申請する。その内容を元に卒業研究配属担当教員が各研究室に振り分けを行う。振り分けの方法として、各生徒の希望順を優先するが、研究室の定員を超えた場合、席次が上位の者を優先して振り分けられる。そして、定員から溢れたものは、再度、希望を受付け各研究室に振り分けられる。この繰り返しにより、卒業研究の研究室が決まっていく。

このように、卒業研究の研究室配属先を決定するまで、同じ動作の繰り返しである。このままでは、配属結果を出すまでに時間がかかり、苦労して出した結果であるが、その結果が正確であるかどうか確認するとなると、また余計に時間がかかるなどのデメリットがある。そこで、繰り返し行う部分をシステム化することができたなら、正確な結果を短時間で出すことが出来るだろうと考えた。

今回の研究では、各生徒の席次順位および研究室希望順位のデータと各研究室の定員数を元に、自動で各研究室に振り分け、その結果を表示するシェルスクリプトを作成した。

2　研究室振り分けシェルスクリプトの概要

2.1　研究室の振り分け方法

現在、実際に行われている研究室の振り分け方法は、全生徒の研究室配属希望順と研究室の定員数を元に振り分けられる。もし、希望者の数が研究室の定員数を越えてしまった場合、希望者の中で席次順位が上位のものを優先的に振り分けて行く。その方法の中で 2 つの疑問があった。それは、第二希望以降の研究室が定員だった場合の処理と、席次順位が同一である生徒同士が、研究室のボーダーを争っている場合の処理についてである。そこで、研究室振り分けシェルスクリプトが実際に行われている研究室振り分け方法と同じになるように、卒業研究配属担当教員である高根助教授に疑問についてお聞きすることにした。

高根助教授のお話によると、各研究室の定員数は厳密には決まっていないため、例えば研究室の定員数から 2 名程度溢れてしまった場合、卒業研究配属担当教員が、研究室の担当教員に、定員を増やしてもらえないかと相談をして、なるべく生徒の希望通りの配属を行うようにしているとのことだった。しかし、定員数を確定しなかったり、研究室の担当教員との相談といった要素をいれた振り分け方法をシステム化するのは、困難である。そのため、今回の研究で作成したシェルスクリプトでは、研究室の定員は確定されたものとし、各生徒の研究室希望順位および席次順位だけを用いて出た結果を表示するようにした。

次に、第 2 希望以降で、定員になっている研究室を希望している場合の処理として、その希望は無効にすることにした。その理由として、公平性を出す為である。実際に、行われている卒業研究の希望配属先の申請では、第 1 希望研究室の集中具合が確認することができ、自分の席次順位を元に、第 1 希望の研究室に入ることができるかをおおよそ推測することができる。そのため、自分の中では第 5 希望であるが安全に定員割をしている研究室を第 1 希望として申請する生徒が出てきたり、逆に、既に定員を越えてしまっている研究室を第 1 希望として申請し、定員から洩れて自分が希望していない研究室に配属されてしまう生徒もいる。

このように、実際に行われている研究室の振り分けでは、申請した生徒の自己責任という部分がある。そのため、今回作成した、シェルスクリプトでも生徒自身が慎重に希望順位を選ぶようにする振り分け方法にした。尚、席次順位が同一であった場合の処理であるが、今回、作成したシェルスクリプトでは対応していない。

2.2　動作手順

以上のことを踏まえて、完成した研究室振り分けシェルスクリプトの動作手順は以下の通りである。

1. 生徒のデータから、席次順位・学籍番号・第 N 希望の研究室を抽出。
2. 第 N 希望の研究室ごとに、生徒のデータを分類する。
3. 各研究室ごとに生徒のデータを分類した後、席次順に並び替える。
4. 研究室が定員か定員割かを判別する。定員割の場合は、各研究室の定員数に応じて確定者を決定し、定員から溢れたものを落選者とする。すでに定員だった場合、その研究室を希望していた者を全て落選者とする。
5. 生徒のデータから落選者のデータを取り出し、生徒のデータを更新する。
6. N を N+1 に変更し、希望順位を変える。
7. 落選者がいなくなるまで 1 ~ 6 を繰り返す。

2.3　使用するファイルの内容

研究室振り分けシェルスクリプトで使用するファイルは、次の 2 つである。

- 全生徒の学籍番号・席次順位・研究室希望順位が書かれた sample ファイル
- 全研究室の定員数が書かれた k_teiin ファイル

```
[kenji@localhost program]$ cat sample
021 88h001 H J G I A B C F E D
020 88h002 I D J H F G B A E C
029 88h003 C B G I H F J D E A
048 88h004 G E J H I F B A D C
041 88h005 A H D F B G E C J I
037 88h006 C D I H J B G E A F
045 88h007 E G D B C A H I J F
035 88h008 H C J I G D B F A E
034 88h009 H J A C G I E D B F
031 88h010 I H C D A G J E B F
014 88h011 F I D G E C B A H J
028 88h012 C E J G I B H D A F
042 88h013 A E B J C I H F G D
033 88h014 D C I B G J F A E H
007 88h015 B E I C F A H J D G
001 88h016 H D B F E C I J G A
015 88h017 G A H E J D I C F B
050 88h018 F G E D C H J A I B
036 88h019 G D E J H C F I B A
023 88h020 G J I E C A H D F B
032 88h021 J B H F I D E C G A
013 88h022 C F G D H E A J I B
003 88h023 A B G C J I F D H E
006 88h024 E G C B H J A I F D
047 88h025 C J H B I D F E G A
030 88h026 I F J H B C G D A E
024 88h027 H E F J A B D I C G
046 88h028 G D I B A C J F H E
019 88h029 A D F C B J E G I H
018 88h030 B I J D A C G H F E
026 88h031 G E C J F D B H A I
002 88h032 F J A D H C B E I G
038 88h033 B E C H D J A I F G
008 88h034 B H F C I J E G D A
025 88h035 H B G A C I J D F E
027 88h036 A F D G H E B J I C
049 88h037 D H C E J F A G B I
022 88h038 C J H D G I F B E A
004 88h039 G B C E H D J A F I
016 88h040 D H B C I A E F J G
```

図 1　sample の内容

sample ファイルに書かれた内容は、左から順番に、席次順位・学籍番号・研究室の希望順位となっている。研究室の希望順位は、左から順番に、希望順位が高い研究室であることを表している。

```
[kenji@localhost program]$ cat k_teiin
A  5
B  5
C  5
D  5
E  5
F  5
G  5
H  5
I  5
J  5
```

図 2　k_teiin の内容

k_teiin ファイルの内容は、左が研究室名、右が定員数を表している。

尚、k_teiin ファイルの内容は、決まった形式で書かれていれば、研究室の数、定員数を変更しても、研究室振り分けシェルスクリプトは問題無く動作する。

2.4 実装

システムの構築には GNU/Linux をインストールしたマシンを用いた。本システムはシェルスクリプトで構成しており、シェルスクリプトが動作する環境があれば、OS に依存することなく動作させることが出来る。

主な動作は、以下の通りである。

1. 生徒の総数を数える
   sample ファイルから wc コマンドを用いて、生徒の総数を数える。
   そして、総数を TOTAL ファイルに保存する。

2. 研究室の数を数える。
   k_teiin ファイルから wc コマンドを用いて、研究室の総数を数える。
   そして、総数を SOUSU ファイルに保存する。

3. 研究室確定者の数を全て 0 にする。
   各研究室の.kazu ファイルに 0 を代入し保存する。

4. 全生徒のデータを、ファイルに保存する。
   全生徒のデータが書かれた sample ファイルの内容を、DATA ファイルに保存する。
   sample ファイルを DATA ファイルに保存するのは、sample ファイルを書換えられてしまうことを避けるためである。そのため、DATA ファイルを元に振り分けを行っていく。

5. 第 N 希望の研究室を抽出し、研究室ごとに生徒のデータを分類する。
   DATA ファイルから、awk コマンドを用いて、生徒の席次順位・学籍番号・第 N 希望の研究室を抽出する。
   そして、grep コマンドを用いて、研究室ごとに分類し、各研究室の.kibou ファイルに保存する。

6. 各研究室の.kibou ファイルを元に、席次順に並び替える。
   sort コマンドを用いて、各研究室の.kibou ファイルの内容を、席次順位に並び変える。

7. 研究室が、既に定員かどうかを判別し、確定者と落選者を決定する。
   研究室の定員数と研究室の確定者数との差が、1 以上ならば定員割、0 以下ならば定員であると判別する。
   このとき、研究室の定員数は、k_teiin ファイルから、sed コマンドと awk コマンドを用いて抽出し、研究室の確定者数は、.kazu ファイルから取り出して来る。
   定員割の場合、研究室希望者が書かれたファイルから、1 行目から研究室の定員数と研究室の確定者数の差だけ、sed コマンドで表示させ、.kakutei ファイルに保存する。逆に、1 行目から研究室の定員数と研究室の確定者数の差だけ sed コマンドで消去し残りを表示させたものを、落選者用の OTI ファイルに保存する。

8. 研究室の確定者数を数える。
   各研究室の.kakutei ファイルを wc コマンドを用いて、確定者の数を数える。

9. 第 N 希望に入れなかった人の総数を数える
   OTI ファイルから、wc コマンドを用いて総数を数えた結果を TOTAL ファイルに保存する。

10. 落選者のデータを sample ファイルから取り出す。
    OTI ファイルから、awk コマンドを用いて落選者の学籍番号を取り出し、grep コマンドと学籍番号を利用し、sample ファイルから落選者のデータを取り出し TMP2 ファイルに保存する。

11. 落選者のデータをファイルに書換える。
    落選者のデータが入っている TMP2 ファイルの内容を DATA ファイルに移す。

12. 第 N 希望から第 N+1 希望に変更
    希望順位を変更する為に、変数 kibou に格納された数に 1 を加える。

13. 落選者がいなくなるまで 5 ~ 12 を繰り返す。
    落選者の有無は、TOTAL ファイルに格納された数で判別する。この場合、格納された数が 0 になるまで繰り返す。

14. 振り分け結果を表示する。
    各研究室の.kakutei ファイルの内容を表示する。

2.5　実行結果

サンプルファイルを元に実行した結果が以下の通りである。

```
[kenji@localhost program]$ ./step1.sh
A
------------
003 88h023 A
011 88h044 A
019 88h029 A
027 88h036 A
041 88h005 A
B
------------
007 88h015 B
008 88h034 B
018 88h030 B
038 88h033 B
039 88h049 B
C
------------
009 88h043 C
013 88h022 C
022 88h038 C
028 88h012 C
029 88h003 C
D
------------
016 88h040 D
033 88h014 D
036 88h019 D
037 88h006 D
049 88h037 D
E
------------
006 88h024 E
026 88h031 E
042 88h013 E
045 88h007 E
048 88h004 E
```

図 3　振り分け結果の表示 1

```
F
------------
002 88h032 F
014 88h011 F
035 88h008 F
044 88h050 F
050 88h018 F
G
------------
004 88h039 G
005 88h046 G
012 88h042 G
015 88h017 G
017 88h047 G
H
------------
001 88h016 H
021 88h001 H
024 88h027 H
025 88h035 H
034 88h009 H
I
------------
020 88h002 I
030 88h026 I
031 88h010 I
040 88h045 I
046 88h028 I
J
------------
010 88h041 J
023 88h020 J
032 88h021 J
043 88h048 J
047 88h025 J
```

図 4　振り分け結果の表示 2

2.6　実行結果の検証

今回、作成した研究室振り分けシェルスクリプトの実行結果が、手作業で振り分けを行った時と同じ結果になるかどうかを検証した。検証するにあたって、生徒の人数は 50 名、研究室の数は 10 室、各研究室の定員数は 5 名づつという条件で行った。サンプルファイルは各生徒の研究室希望順位に偏りがでないように、シェルスクリプトで自動生成したものを使用した。

検証の結果、研究室振り分けシェルスクリプトの実行結果と、手作業で出した振り分け結果と一致した。この点から研究室振り分けシェルスクリプトの正確性が実証されたと言える。

2.7　インタフェース

本研究では、研究室の振り分けを自動で行うシェルスクリプトを作成した。そのシェルスクリプトを実際にシステムとして使用する時は、WEB ブラウザを用いて、CGI として稼働させる予定である。

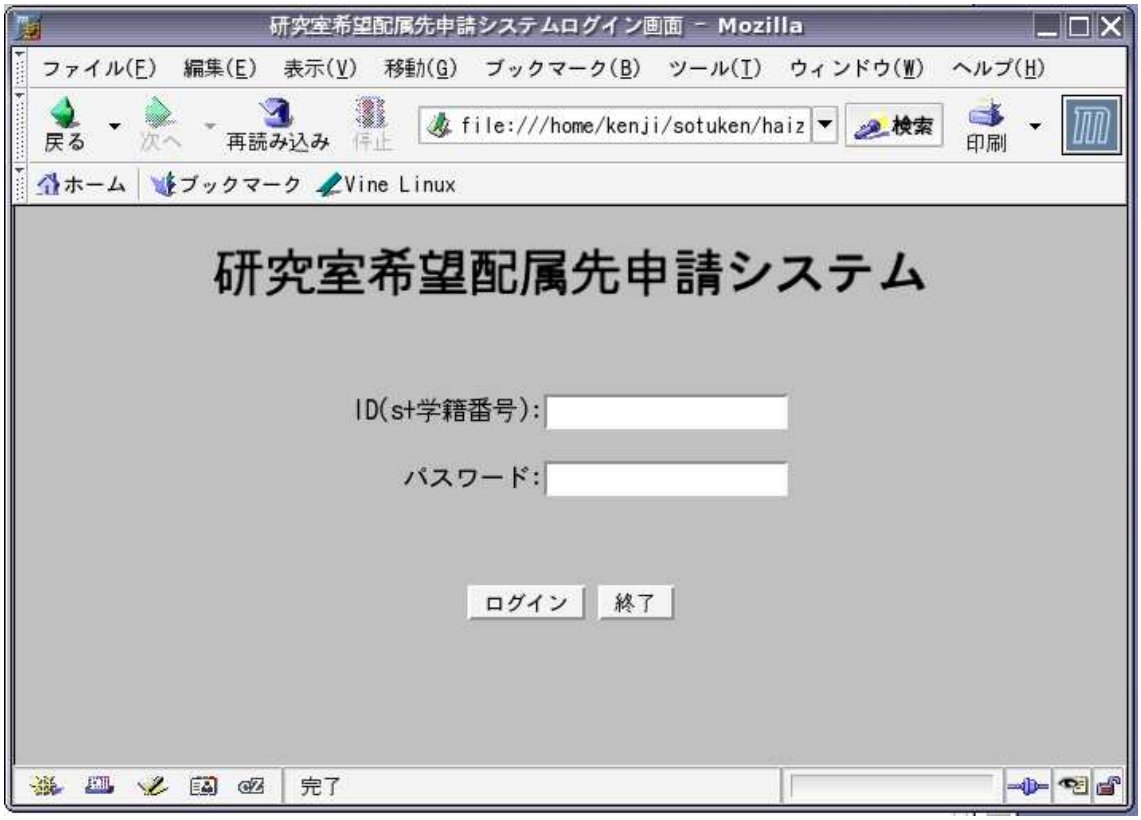

図 5　ログイン画面

ログイン画面では、ID とパスワードをユーザーに入力してもらう。尚、ID は、s+ 学籍番号、パスワードは情報科学センター発行のパスワードを使用する。ID、パスワード入力後、ログインボタンをクリックすることで、ログインできる。終了ボタンをクリックすると、画面を閉じることが出来る。

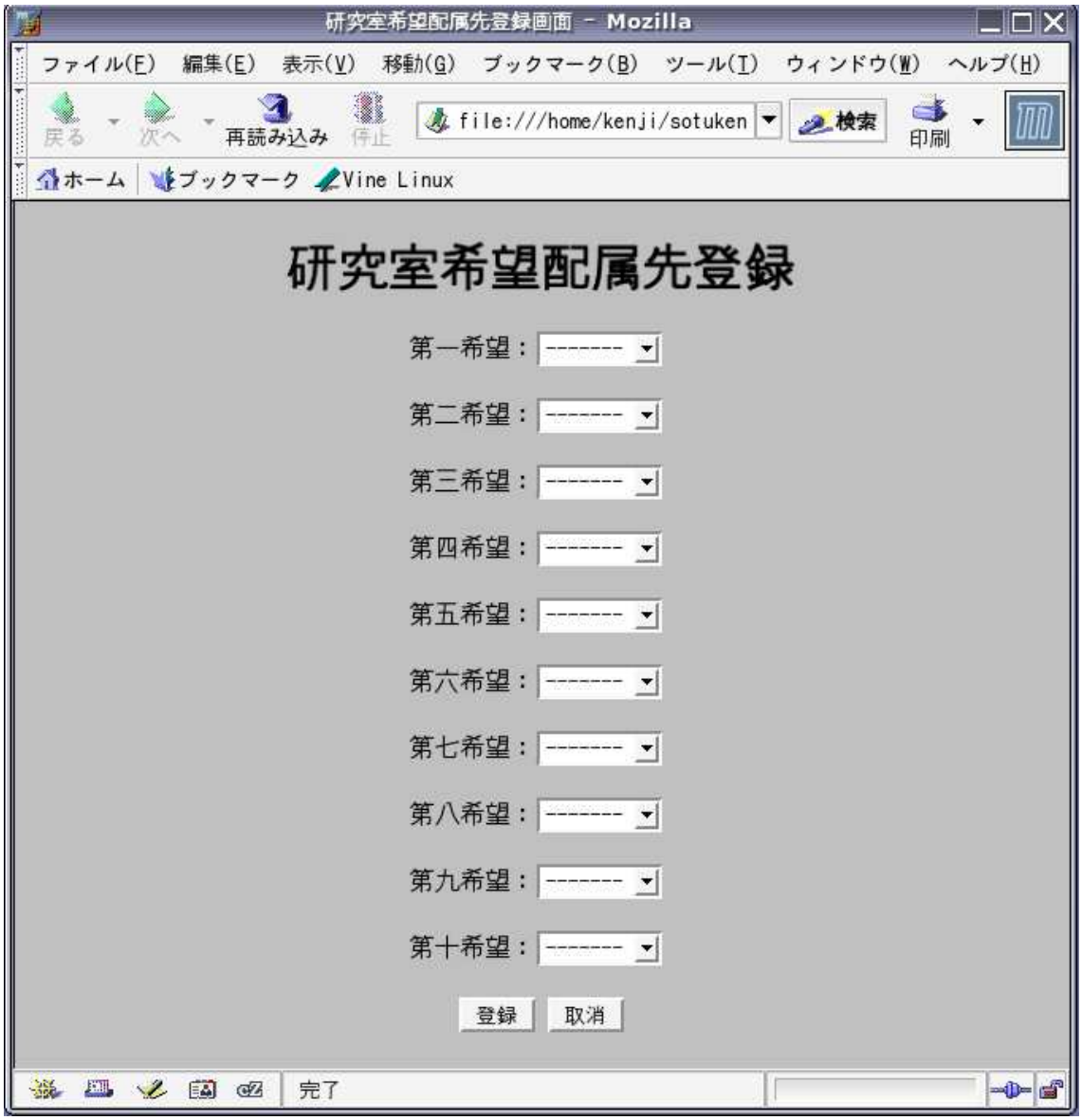

図 6　研究室希望配属先申請の画面

研究室希望配属先の申請は、ドロップダウンメニューを用いて行う。図では 10 個の研究室を第 1 希望から第 10 希望までを選択し申請するようになっている。選択が終わり、申請する場合は申請ボタンを、申請をやりなおす場合は取消ボタンを押す。

3　まとめ

今回の研究では、シェルスクリプトを用いて、あらかじめ用意しておいた各生徒の席次順位および研究室の希望順位が書かれたサンプルファイルを自動で各研究室に振り分けて結果を表示するというところまでは完成した。

3.1　今後の課題

研究室振り分けシェルスクリプトを実用させるにあたって、以下のことを実装させるのが課題である。

- 生徒が実際に入力したデータをファイルに書き込み、そのファイルを元にシステムが稼働するようにしなければならない。
- サンプルファイルでは、席次順位が全て順番どおりになっているが、実際の場合、席次順位が同一順位の生徒も出てくる可能性がある。しかし、今回作成したシェルスクリプトでは、同一順位に対する処理ができていない。今後は、サンプルファイルに同一席次順位を混ぜて、現状のままでは、どういう処理を行うかを検証し、問題がある場合はその対処法についても考えなければならない。

以上の 2 点が課題である。

謝辞

本研究を進めていく上で、御指導および御協力を頂いた大垣斉講師・藤井信夫教授・高根慎也助教授ならびに fken.a4w ML メンバーの方々、そして情報システム工学専攻の大学院生である宮崎純氏に、深く感謝いたします。

参考文献

[1] Linux コマンド入門 http://cyberam.dip.jp/category/linux_command.html

4　付録

4.1　研究室振り分けシェルスクリプトのソース

```
#! /bin/sh

TMP=/tmp/k-$$-TMP
TMP2=/tmp/k-$$-TMP2
NULL=/tmp/k-$$-NULL
DATA=/tmp/k-$$-DATA
OTI=/tmp/k-$$-OTI
TOTAL=/tmp/k-$$-TOTAL
SOUSU=/tmp/k-$$-SOUSU
TEIIN=/tmp/k-$$-TEIIN
KEN=/tmp/k-$$-KEN

trap "rm -f /tmp/k-$$*; exit 0" 0 1 2 3 15

#中身が空の NULL ファイルを作成
touch $NULL

#生徒の総数
wc -l sample | awk '{print$1}' > $TOTAL
total=`cat $TOTAL`

#研究室の総数
wc -l k_teiin | awk '{print$1}' > $SOUSU
sousu=`cat $SOUSU`

#研究室確定者数を全てゼロにする。
count=1
while [ $count -le $sousu ];
do
sed -n ''$count'p' k_teiin | awk '{print$1}' > $KEN
kenkyu=`cat $KEN`
echo 0  > "$kenkyu.kazu"
count=`expr $count + 1`
done

#サンプルデータを data ファイルに入れる
cat sample > $DATA
```

```
#サンプルファイルで、研究室の希望が 3 列目からなので合わせる
kibou=3

while [ $total -ne 0 ]; #残留者がいなくなるまで繰り返す
do #while 文の do

#第 N 希望の研究室の抽出し研究室ごとに分類
count=1
while [ $count -le $sousu ];
do
sed -n ''$count'p' k_teiin | awk '{print$1}' > $KEN
kenkyu=`cat $KEN`
cat $DATA | awk '{print$1,$2,$'$kibou'}' | grep $kenkyu > "$kenkyu.kibou"
count=`expr $count + 1`
done
            #各研究室ごとに定員数を取り出し、確定か否かを判別
            count=1 #研究室カウント
            for i in *.kibou
            do #for 文の do
            sort $i > $TMP #研究室内の席次順に並び替え
            sed -n ''$count'p' k_teiin | awk '{print$1}' > $KEN
            sed -n ''$count'p' k_teiin | awk '{print$2}' > $TEIIN

            teiin=`cat $TEIIN`
            kenkyu=`cat $KEN`
            ninzu=`cat "$kenkyu.kazu"`
            kakutei=`expr $teiin - $ninzu`

            #定員割しているか?
            if [ $kakutei -gt 0 ];
            #定員割している場合
            then
            sed -n -e '1,'$kakutei'p' $TMP >> "`cat $KEN`.kakutei" #確定者
            sed -e '1,'$kakutei'd' $TMP >> $OTI #落選者
            #既に定員だった場合
            else
            cat $i >> $OTI
            fi

            #確定者数を数える
            wc -l "`cat $KEN`.kakutei" | awk '{print$1}' > "$kenkyu.kazu"

            #研究室カウンタが最後まで来たか?
```

```
            if [ $count -ne $sousu ];
            #来ていない場合
            then
            count=‘expr $count + 1‘
            #来ている場合
            else
            count=1 #最初にもどす
            fi

            done #for 文の done

                        #第 N 希望に入れなかった人の総数
                        wc -l $OTI | awk ’{print$1}’ > $TOTAL
                        total=‘cat $TOTAL‘

                        #残留者のデータをサンプルファイルから取り出す。
                        count=1
                        while [ $count -le $total ];
                        do
                        sed -n ’’$count’p’ $OTI | awk ’{print$2}’ > $TMP
                        gakuseki=‘cat $TMP‘
                        grep $gakuseki sample >> $TMP2
                        count=‘expr $count + 1‘
                        done

cat $TMP2 > $DATA    #data ファイルを更新
cat $NULL > $TMP2    #tmp2 ファイルを初期化
cat $NULL > $OTI     #okkoti ファイルを初期化

kibou=‘expr $kibou + 1‘ #第 N 希望から第 N+1 希望に変更

done #while 文の done

                                #結果の表示
                                for i in *.kakutei
                                do
                                sort $i > $TMP
                                echo ${i%.kakutei} $’\n’------------
                                cat $TMP
                                done

#後始末
rm *.kibou *.kakutei *.kazu
```

4.2 サンプルプログラム自動生成シェルスクリプトのソース

```
#! /bin/sh

# 学生数
MEM=50
# 研究室
RWS=10

TMP1=/tmp/.mk-$$-1
TMP2=/tmp/.mk-$$-2
TMP3=/tmp/.mk-$$-3

trap "rm -f $TMP1 $TMP2 $TMP3; exit 0" 0 1 2 3 15

yes | head -${MEM} | awk '{printf("88h%03d\n" ,NR)}' > $TMP1
touch $TMP2
for i in `cat $TMP1`
do
$0.rm.2 $i >> $TMP2
done

rev $TMP2 | sort | rev | awk '{printf("%s-%d\t\n",$1,NR) }' > $TMP3

/bin/rm -f $TMP1
touch      $TMP1
for i in `cat $TMP3`
do
echo -n $i | sed -e 's/-/ /g' >> $TMP1
echo -n " " >> $TMP1
$0.rm $RWS >> $TMP1
done

cat $TMP1 | sort
```

```
#! /bin/sh

TMP4=/tmp/.mk-$$-4

trap "rm -f $TMP4 ; exit 0" 0 1 2 3 15

export RWS=$1

echo Hoge | awk '{
for ( i = 0 ; i < '$RWS' ; i++ ) {
printf("'$0'.2 %c\n",64+1+i);
}
}' > $TMP4

sh $TMP4 | rev | sort | rev | awk '{printf("%c ",$1)}END{printf("\n")}'

exit
```

```
#! /bin/sh

RMS=‘date +%s‘
sleep 1

SUM=‘echo $1 ${RMS} | md5sum | awk ’{print $1}’‘
echo $1 $SUM

exit 0
```