# 1L－2　ノービスプログラマにおける繰返し処理のアルゴリズム

島田 利博　小山 裕徳
東京電機大学

## 1．はじめに

プログラミングの学習過程において，繰返し処理を含むアルゴリズムの習得は初心者にとって重要なステップの1つである．本稿では、プログラミング言語に繰返しアルゴリズムが比較的自由に表現できる制御構造を与えた場合に，ノービスプログラマが繰返し処理のアルゴリズムをどう構築するかを実験的に検討した結果について報告する．

## 2．繰返し処理とPascalによる解

ノービスプログラマが最初に遭遇する繰返し処理の例として，図1に示すようなデータ入力とその処理を繰り返す問題を考えた．これに対するPascalによる解を図2に示す．

キーボードから複数の正整数データを入力して、それらの平均を求め、画面に表示するプログラムをつくる。ただし、1つ以上の正整数を入力したあとでデータの終わりの合図として負の整数を入力するものとする．

図1　繰返し処理の例

```
i:= 0; sum := 0;
readln(data);
while data >= 0 do
begin
   i := i + 1;
   sum := sum + data;
   readln(data)
end;
average := sum / i;
```

```
i := 0; sum := 0;
repeat
   readln(data);
   if data >= 0 then
   begin
      i := i + 1;
      sum := sum + data
   end;
until data < 0;
average := sum / i;
```

(a)while文を使った例　(b)repeat文を使った例

図2　Pascalによる解

繰返し処理として，基本的にデータの入力［以下，readと記す］と処理（iの数え上げとsumへのdataの足し込み）［以下，数え上げをcount，足し込みをadd，合わせた処理をprocessと記す］が必要になる．図2(a)では，繰返しの継続を入口で判定するというwhile文の制約から，前処理としてデータ入力が必要であり，繰返し処理はprocess/readの順になる．一方，図2(b)では、繰返しはread/processであり，processには条件判定が必要になる．これはrepeat文が出口で継続を判定する制約からきている．いずれの場合も，言語で提供されている繰返しの制御構造の制約が，とくにノービスプログラマにとっては，アルゴリズムの構築に大きな影響を与える．

## 3．loop文とbreak文

図3に構文を示すloop文は、loopとagainで囲まれた〈文〉の並びを無限回繰り返すための制御構造である．一方，図4のbreak文はloop文の中だけで使用でき、実行されると繰返しが中断されagainの次の文へ実行が移る．break文をif文と併せて使用すれば，一定の条件下で繰り返しを中断する処理［以下，leaveと記す］の記述が可能になる．このloop文とbreak文を使えば，繰返しアルゴリズムを自由に記述することができる．図5に，図1に対する解の例を示す．

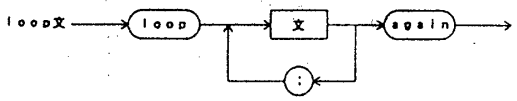


図3　loop文の構文図

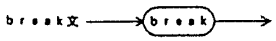


図4　break文の構文図

```
i := 0; sum := 0;
loop
   readln(data);          {read}
   if data < 0 then       {leave}
      break;
   i := i + 1;            {count}
   sum := sum + data      {add}
again;
average := sum / i;
```

図5　loop文，break文を使った解の例

## 4．実験と結果

実験は、コンピュータを専門としない理工系学部1年生で，Pascalによるプログラミングの基礎教育を約6ヶ月間学んだ学生127名を対象に行った。学生に対して図1を課題としてPascalと，Pascalからwhile文とrepeat文を除きloop文とbreak文を加えた言語［以下，PascalLと記す］の両方で記述させた。図6は，正解のパターンと誤答の人数を示したものである．なお，変数の初期設定や，繰返し後の計算式でケアレスミスと見なされるものは正解に含めた．

---

On Algorithms of a Looping Process in Novice Programmers
Toshihiro Shimada, Hironori Koyama
Tokyo Denki University

| PascalL言語 | 人 |
|---|---|
| read/process | 87 |
| process/read | 6 |
| 構文不足 or 誤答 | 34 |
| 合　　計 | 127 |

| Pascal言語 | 人 |
|---|---|
| repeat文 | 78 |
| while文 | 13 |
| 構文不足 or 誤答 | 36 |
| 合　　計 | 127 |

図6　PascalLとPascalの解答

| PascalL言語 | Pascal言語 | 人 |
|---|---|---|
| read/process | repeat文 | 73 |
| | while文 | 13 |
| | 構文不足 or 誤答 | 1 |
| process/read | repeat文 | 4 |
| | while文 | 0 |
| | 構文不足 or 誤答 | 2 |
| 構文不足 or 誤答 | repeat文 | 1 |
| | while文 | 0 |
| | 構文不足 or 誤答 | 33 |

図7　PascalLとPascalの解答分類

PascalLの正解の68.5%はread/processであり，Elliot Solowayら[1]によってすでに指摘されていることが確認された．このことはPascalでrepeat文による正解の比率と対応し，ノービスプログラマが概念的にread/processとしてアルゴリズムを構築する傾向にあることを示している．図7はそれぞれの解答を対応させた表である．PascalLでread/processで正解したものの多数は，Pascalで同じパターンのrepeat文で正解している．

## 5．前後処理とアルゴリズムの選択

図8は，PascalLの正解のうち，read/processアルゴリズムを繰返し部分の処理順序に着目してさらに分類したものである．(パターン1，2は、addとcountが逆順のものも含めて考える。)

| read/processのパターン | 人 |
|---|---|
| パターン1 | 36 |
| パターン2 | 30 |
| パターン3 | 19 |
| パターン4 | 2 |

パターン1（図5）

```
loop
   readln(data);          {read}
   if data < 0 then       {leave}
      break;
   i := i + 1;            {count}
   sum := sum + data      {add}
again;
```

パターン2（repeat文の書換え）

```
loop
   readln(data);          {read}
   i := i + 1;            {count}
   sum := sum + data;     {add}
   if data < 0 then       {leave}
      break
again;
```

パターン3

```
loop
   readln(data);          {read}
   i := i + 1;            {count}
   if data < 0 then       {leave}
      break;
   sum := sum + data      {add}
again;
```

パターン4

```
loop
   readln(data);          {read}
   sum := sum + data;     {add}
   if num < 0 then        {leave}
      break;
   i := i + 1             {count}
again;
```

図8　read/processの分類

それぞれのパターンは繰返し部分だけを示したものであり，正しいアルゴリズムになるためには，繰返し前の初期設定などの前処理はもちろんのこと，繰返し後の平均の計算において，パターンによっては余分に行った数え上げや足し込みをキャンセルする処理や記述が必要になる．パターン1（図5）は後処理が最も簡単に記述できるアルゴリズムであり，図8は繰返し後の処理がより簡単になるアルゴリズムが選択される傾向にあることを示している．

このことは，ノービスプログラマにとって，繰返し処理自体とその前後処理を結びつけて全体として正しいアルゴリズムを構築することの難しさを意味している．前処理にreadや作為的な初期設定が必要になるprocess/readよりもread/processが選択されることも同様に考えられる．さらに，誤答のほとんどが繰返し前後の処理が適切に記述できないことによるものであったことからも同じことが推測される．

## 6．おわりに

本稿では、ノービスプログラマが繰返しを含むアルゴリズムを構築するときに，繰返しの前後処理の複雑が，正しいアルゴリズムを導くための重要な要素となることを示した．

今後は、プログラミングの経験とともに繰返しアルゴリズムの構築がどのように変化するかを調査するとともに，アルゴリズムの構築においてノービスプログラマが陥りやすい，その他の問題についても検討していきたい．

＜参考文献＞

[1] Elliot Soloway, James C. Spohrer : Studying The Novice Programer LEA, Inc. (1989)