リスト

山本昌志*

2007 年 6 月 26 日

概 要

よく使われるデータ構造のひとつ，リストについて学ぶ．それを，配列との対比で説明する．

1　本日の学習内容

データ構造のひとつであるリストについて学習する．本日の学習のゴールは以下の通りである．

- リストと配列の違いが分かる．
- リストをつるく構造体の書き方が分かる．
- リストを使うための関数の書き方が分かる．

教科書 [1] の pp.170–184 が本日の範囲である．

2　データ構造とは

これから，3 回の授業ではデータ構造について学習する．リストとスタック，キュー，ツリー (木) である．いままでもいろいろなデータ構造を学習してきた．データ構造は，表 1 のようにまとめることができる．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

表 1: データ構造の種類

データ構造	基本データ構造	基本データ型	単純型	整数型
				実数型
				文字型
				論理型
				数え上げ型
			ポインタ型	
		構造型	配列型	
			レコード型	
		抽象データ型		
	問題向きデータ構造	線形リスト	単純リスト	
			双リスト	
			環状リスト	
		木 (ツリー)	二分木	完全二分木
				二分探索木
				バランス木
			多分木	
			バランス木	AVL 木
				B 木
		スタック		
		キュー		

データ構造とは，データのメモリーの格納の仕方のことを言う．

3 リストと配列

リストと配列の違いについて，比較を行う．

3.1 メモリーへの格納の仕方

順序づけられたデータのことをリストと言う．このリストは，データ構造のリストとは異なるので注意すること．一般には，各々のデータはカンマで区切り表現する．この順序づけられたデータの例としては，数列がある．

$$2, 63, 43, 12, 24, 77, 5, 23, 18, 37, 81, 57, 29, 62, 49, 30$$

数列に限らず，成績表等もリストの例である．

$$(\text{aoki}, 83), (\text{kato}, 73), (\text{sasaki}, 49), (\text{tanaka}, 55), (\text{noda}, 95), (\text{fukuda}, 35)$$

順序づけられたデータ—リスト—をコンピュータープログラムで取り扱う場合，配列あるいはリストと呼ばれるデータ構造が使われる[1]．例えば，数列

$$63,\ 27,\ 82,\ 79,\ 12$$

をこれらのデータ構造で表現する．配列のモデルとメモリーへの格納の様子は，図 1 のようになる．その特徴は，次の通りである．

- 配列の添字を使って，順序を表している．
- メモリーの連続した領域に，データの順序通りに格納される．

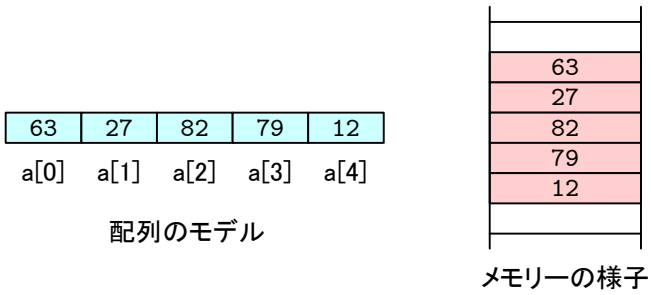

図 1: 配列のモデルとメモリーの様子．

それに対して，リストは図 2 のように表現することができる．その特徴は，次の通りである．

- ポインターを使って，順序を表している．
- メモリーの連続した領域にデータを格納する必要がない．また，データの順序通りに値を格納する必要もない．

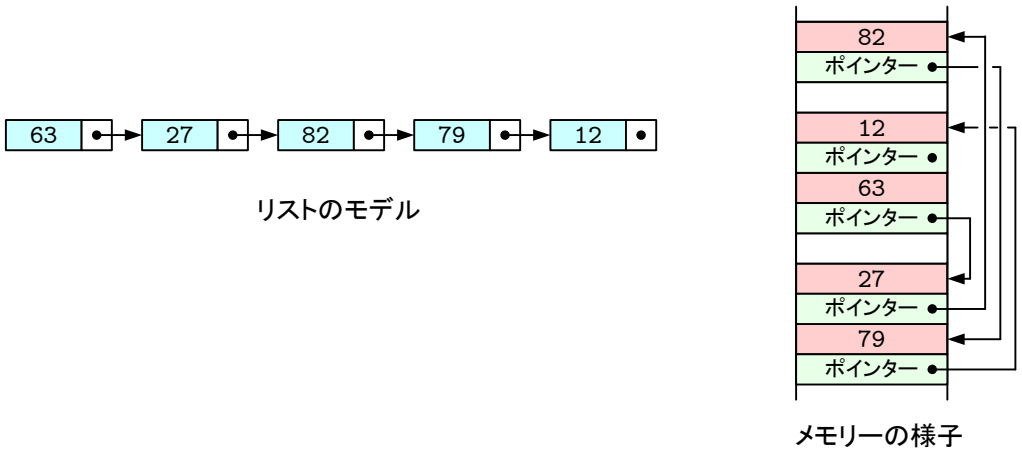

図 2: リストのモデルとメモリーの様子．

[1] これらの他のデータ構造でも取り扱うこともできる

3.2　データの挿入

配列とリストにデータを挿入する場合を考える．先ほど示した数列の 63 の次に 99 を挿入することを考える．

配列の場合，図 3 のようにしてデータを挿入する．後ろの方からひとつずつ，右側に値をコピーする．そして，データを挿入する場所にあった値をのコピーが終われば，そこに挿入する値をコピーする．挿入する場所にもよるが，データ数に比例してコピーの手間は増える．データ数を N とするとコピーの回数—コンピューターの処理回数—は，$N/2$ 程度である．もちろん，コピー回数は挿入する場所にも依存するが，平均して $N/2$ となる．これを $O(N)$ と表現する.「オーダー N」と読む．N に比例して処理の回数が増える—ということを表している．

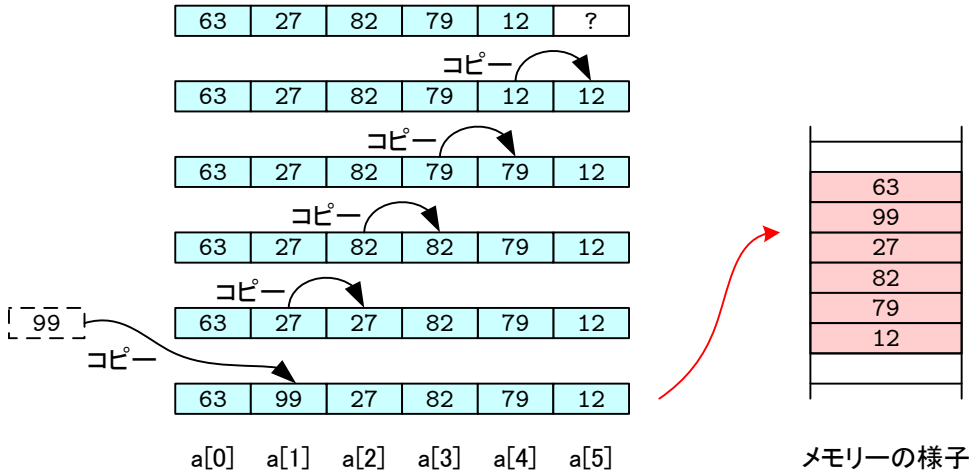

図 3: 配列のデータの挿入の様子．

それに対して，リストの場合は図 4 のようにしてデータを挿入する．データ 63 のポインターを 99 に接続し，データ 99 のポインターを 27 に接続する．この方法だと，データがいくらあっても処理の回数は同じである．$O(1)$ である．配列に比べて，処理の回数は少なく，コンピューターで高速の処理ができる．データ数が多くなるとその差は顕著になる．

配列とリストではデータを挿入する場合，処理の回数が決定的に異なる．リストの方が処理の回数が少ない．処理の回数が異なるのは，データのメモリーへの格納の仕方による．配列は頑固に，データと同じ並びで連続した領域に，値を格納する．一方，リストはメモリーに適当に格納して，順序を表すポインターを使う．

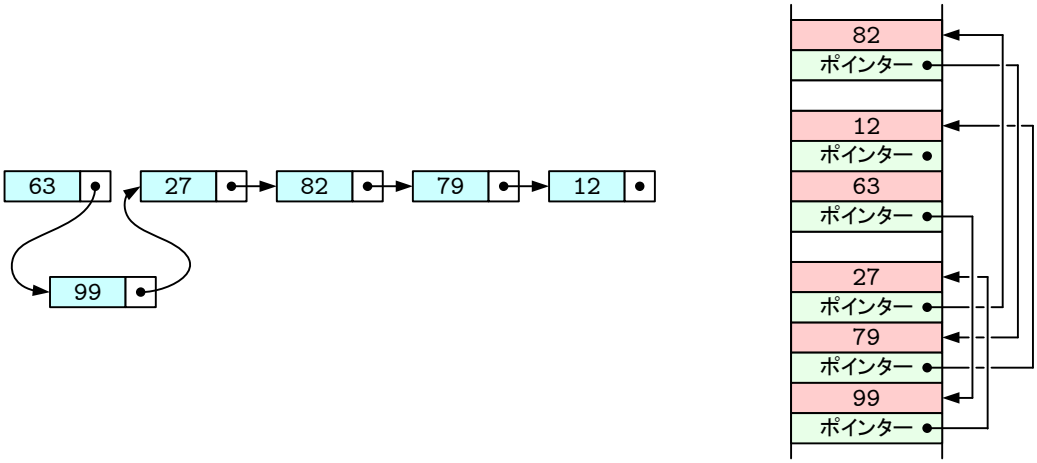

図 4: リストのデータの挿入の様子.

3.3 データの削除

次に，先ほど 99 を挿入した数列

$$63,\ 99,\ 27,\ 82,\ 79,\ 12$$

から，99 を削除することを考える．

配列の場合，図 5 に示した方法で，データを削除する．挿入とは逆に，前の方から順番にコピーを繰り返す．最後のデータ—図 5 の最後の 12—は意味のないデータとなる．このような方法では，データの数 N とすると平均して $N/2$ 回の処理が必要である．したがって，$O(N)$ となる．

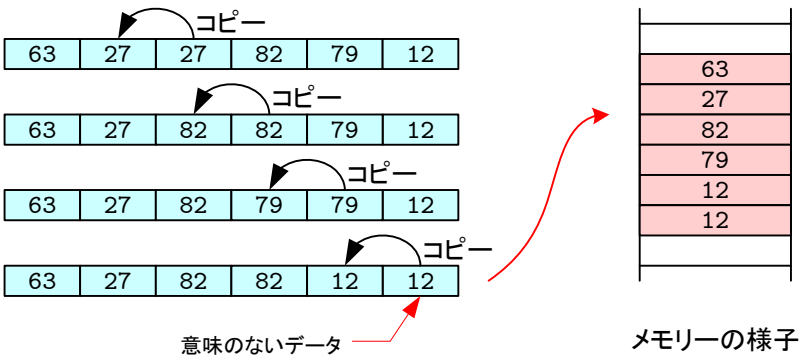

図 5: 配列のデータの削除の様子.

リストの場合，図 6 に示した方法で，データを削除する．データ 63 のポインターをデータ 27 に接続して，データ 99 を削除する．データ数とは関係なく処理の回数は決まっており，オーダーは $O(1)$ となる．配列よりも高速な処理ができ，データ数が多くなればその差は顕著になる．

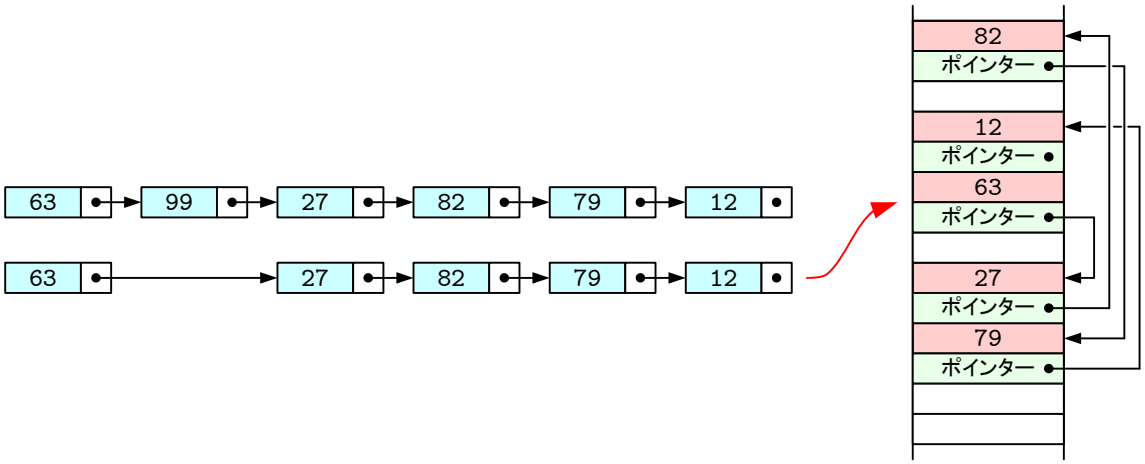

図 6: リストのデータの削除の様子.

3.4 データへのアクセス

次に，データのアクセスを考える．図 1 や 2 のように，配列やリストを用いて，数列

$$63,\ 27,\ 82,\ 79,\ 12$$

がメモリーに格納されているとする．これの 79 というデータにアクセスするにはどうするか?

配列の場合，a[3] とすれば 79 のデータにアクセスすることができる．それぞれのデータは配列名と添字により，アクセスできる．これは，個々のデータに名前が付いているようなものである．コンピューターはなんの迷いも無く目的のメモリーアドレスが分かる．これはデータの順序通りにメモリーにすき間無く値が格納されているから，可能なのである．データへアクセスする場合の処理のオーダーは $O(1)$ となる．次の述べるリストよりも格段に高速である．

リストの場合，目的のデータへのアクセスは手間がかかる．データのある場所が分かるのは先頭のデータのみである．これは後で述べる特別な方法で先頭のデータのみ分かるようにしている．先頭のデータ 63 の次のデータ →27 の次のデータ →82 の次のデータ →79—というように先頭からたぐりよせなくてはならない．このように先頭からたぐり寄せなくてはならないのは，それぞれのデータに名前が無いからである．したがって，N 個のデータがある場合，処理の平均的な回数は $N/2$ となる．$O(N)$ である．配列よりも処理の回数が増える．

3.5 リストと配列の違い

配列もリストも，複数のデータの値と順序を記憶するデータ構造である．しかし，その使い方はかなり異なり，その性質をよく理解しなくてはならない．

配列は目的のデータにランダムアクセスが可能で，目的のデータの値を得たり，データの値の変更が高速にできる．添え字を指定するだけで，それらが可能である．しかし，データの削除と挿入には計算回数が多

くなる．たとえば，10 番目のデータを削除するとなると，11 番目を 10 番目に移動，12 番目を 11 番目に移動，13 番目を 12 番目に移動 · · · とデータを順次移動させる必要がある．

一方，リストは目的のデータにアクセスするためには，シーケンシャルアクセス[2]を行う．そのため，データへのアクセス回数が多くなり配列より低速になる．しかし，データの削除と挿入は簡単で，その前後へのノードのポインターの値を変更するだけである．したがって，挿入と削除は高速になる．

また，データの個数の柔軟性はリストの方が高い．配列の場合，要素の数はコンパイル時に予め指定する必要がある．連続したメモリー領域を確保するためである．配列の場合，データ数が予め分からない場合は，十分大きなメモリー領域を確保することになる．そのため，メモリーを無駄遣いすることがある．それに対して，リストはデータの個数は後で追加/削除できる．

これらの違いをまとめると，表 2 のようになる．一般的には，データの追加/削除が多い場合にはリスト，データのアクセスが多い場合には配列を使う．

表 2: 配列とリストとの違い

	配列	リスト
データへのアクセス	添え字によるランダムアクセス可能	リストを順にたどる
アクセスのための計算量	$O(1)$	$O(N)$
データの挿入/削除	計算コスト大 ($O(N)$)	計算コスト小 ($O(1)$)
メモリーのコスト	小	配列より大
データ数	コンパイル時に決定	追加/削除可能

4 プログラム例

リストを使ったプログラム例をリスト 1 に示す．

データのイメージは図 7 のとおりである．リストの先頭は head を用いて表している．最後のノードは次のノードが無いため，NULL ポインターをいれている．

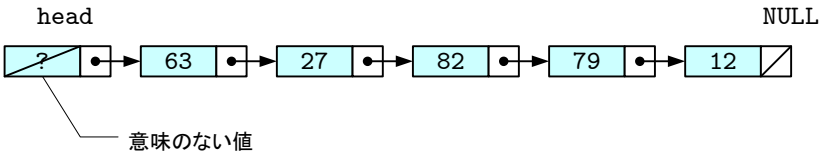

図 7: 先頭を最後を表すノードを追加したリストの構造．

リスト 1: 数列をリストで管理するプログラム例．

```
1 #include <stdio.h>
2 #include <stdlib.h>
```

[2]データを先頭から順番に読み込み、あるいは書き込みを行なう方法。

```
3
4  typedef struct seqn_tag{       // ノードを表す構造体
5    int data;
6    struct seqn_tag *next;
7  }seqn;
8
9  void insert(seqn *head, int n, int new_num);
10 void delete(seqn *head, int n);
11 void prt_node(seqn *head);
12
13 //============= メイン関数 ====================
14 int main(void)
15 {
16   seqn head;
17
18   head.next=NULL;
19
20   insert(&head, 0, 63);
21   insert(&head, 1, 27);
22   insert(&head, 2, 82);
23   insert(&head, 3, 79);
24   insert(&head, 4, 12);
25   prt_node(&head);
26
27   insert(&head, 1, 99);
28   prt_node(&head);
29
30   delete(&head, 1);
31   prt_node(&head);
32
33   return 0;
34 }
35
36 //============= 挿入 ====================
37 void insert(seqn *head, int n, int new_num)
38 {
39   int i=0;
40   seqn *p, *new_node;
41
42   new_node = (seqn *)malloc(sizeof(seqn));
43
44   new_node->data = new_num;
45
46   p=head;
47
48   while(i<n){
49     i++;
50     p=p->next;
51     if(p->next == NULL)break;
52   };
53
54   new_node->next = p->next;
55   p->next = new_node;
56 }
57
58 //============= 削除 ====================
59 void delete(seqn *head, int n)
60 {
61   int i=0;
62   seqn *p, *del_node;
63
64   p=head;
```

```
65    if(p->next == NULL)return;
66
67    while(i<n){
68      i++;
69      p=p->next;
70      if(p->next == NULL)break;
71    };
72
73    del_node=p->next;
74    p->next = del_node->next;
75    free(del_node);
76  }
77
78  //============= 表示  ====================
79  void prt_node(seqn *head)
80  {
81    seqn *p;
82
83    p=head;
84
85    if(p->next == NULL)return;
86
87    do{
88      p=p->next;
89      printf("%d,",p->data);
90    }while(p->next != NULL);
91
92    printf("\n");
93  }
```

実行結果

```
63,27,82,79,12,
63,99,27,82,79,12,
63,27,82,79,12,
```

5 プログラム作成の練習

[**練習 1**] リスト 1 のプログラムを理解せよ.

6 課題

以下の課題を実施し，レポートとして提出すること.

[**問 1**] (復予) 教科書 [1]pp.170–199 を 3 回読め．レポートには「3 回読んだ」と書け.

[**問 2**] (復) 本日配布したプリントを 2 回読め．レポートには「2 回読んだ」と書け.

[**問 3**] (復) リスト 1 のプログラムでのリストの実現方法をまとめよ．そして，各関数の動作をまとめよ.

[**問 4**] (復) ここでの学習内容でわからないところがあれば，具体的に記述せよ.

提出要領はいつものとおり．ただし，期限は 7 月 3 日 (火)AM 8:45，課題名は「課題　リスト」とすること．

参考文献

[1] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　応用編　第 2 版. (株) オーム社, 2006.