リスト

山本昌志*

2005 年 11 月 21 日

1　前回の復習と本日の学習内容

1.1　前回の復習

先週は，サーチについて学習した．学習したサーチの方法は，以下の 2 通りの方法である．

- リニアサーチ

  マッチするデータを端から，順に捜していく方法である．データがランダムに並んでいる場合，この方法しかない．計算のオーダーは，$O(N)$ である．

- バイナリーサーチ

  まずは，データを規則に従って並び替える (ソート)．マッチするデータは，その真ん中と比較して，候補を半分に絞る．これを繰り返すことにより，高速に候補の数を減らし，サーチを行う．計算のオーダーは，$\log_2 N$ である．

1.2　本日の学習内容

本日は，データ構造である．リストについて学習する．特に，配列との比較を行い，その違いを理解しなくてはならない．

後期の最初の講義で述べたように，データ構造とはデータのメモリー上の表現方法のことである．このことを忘れないで，講義を聴くように．

2　プログラムで解くべき問題

リストと配列の違いを説明するために，教科書では次のような処理をするプログラムが書かれている．

- 整数をキーボードから読み込む
- 読み込まれた整数の合計を表示する．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

- ただし，ゼロが読み込まれると，読み込む整数の終わりとする．

教科書の List 3-1〜List 3-3 が配列を使った例である．List 3-4 がリストを使った例である．

いたって，単純な問題である．ただし，ここでは，読み込む毎に加算する方法は取らないこととする．読み込んだデータは，一旦，メモリーの中に記憶するものとする．さあ，どうするか?

3 リストを使わない方法

3.1 固定長さの配列を使う方法

3.1.1 原理

もっとも単純な方法は，配列に入力データを格納する方法である．教科書の List 2-1(p.70) の方法である．このプログラムをプリントのリスト 1 にフローチャートを図 1 に示す．

このプログラムの大きな問題点は，

- データを 10 個以上，入力すると，プログラムがクラッシュする．

ことである．コンパイラーによって，予約した分以上のメモリーの領域にデータを書き込むと，プログラムは異常な動作を行う[1]．

それで，最初から配列を大きくする方法もある．ただ，データ数が分からないと，それも限界がある．とてつもなく大きな配列を用意することも考えられるが，そうすると次の問題が生じる．

- メモリー領域の大部分に値が格納されないことが生じ，メモリーの無駄遣いになる．

このようにデータ数が分からない場合，配列を使うのははなはだ無駄が生じやすい．

3.1.2 フローチャート

教科書の List 3-1(p.70-71) あるいはこのプリントのリスト 1 のプログラムのフローチャートを図 1 に示す．

このプログラムは，おもに，

- キーボードの整数を配列に格納する．
- 格納された配列を合計する．

から構成される．このプログラムのもっともまずい点は，配列の終わりを確認しないで，データを格納しているところである[2]．普通のプログラマーであれば，確実に配列の大きさを確認しながら，データを格納する．

[1] 配列を越えて書き込みが行われないようにするのはプログラマーの仕事である．

[2] この本の筆者の弁護をしておこう．これほどの本を書く人であるから，当然分かっている．読者に分かりやすくするために，本論から外れる部分は省いたと考えられる．

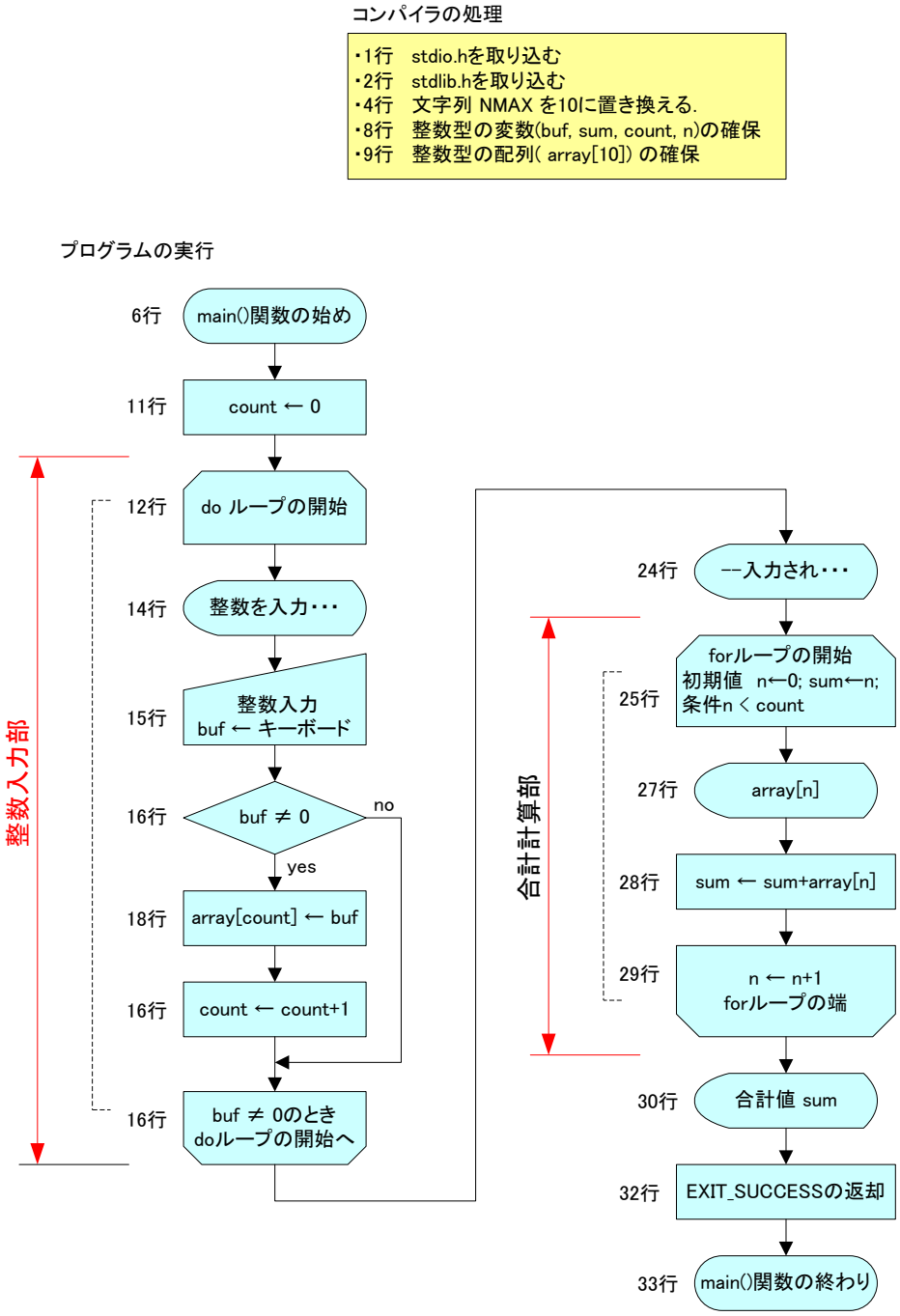

図 1: 固定長の配列を使うリスト 1 のフローチャート.

3.1.3 プログラム

固定長の配列を使うプログラムのリスト 1 に示す．これは，教科書の List 3-1(p.70-71) と同じである．ここで使われているプログラムのテクニックは，すでに学習済みであるが，分かりにくいものだけ述べておく．

- 2 行目 `#include <stdlib.h>`

  これは，コンパイルの作業を行う前に，ソースファイルに，`stdlib.h` というヘッダーファイルを取り込む命令 (プリプロセッサー) である．このプログラムでは，`main` 関数の戻り値を `EXIT_SUCCESS` としている．`EXIT_SUCCESS` の定義が `stdlib.h` に書かれている．私のコンパイラーでは，

  ```
  #define EXIT_SUCCESS 0
  ```

  となっていた．戻り値を示すリスト 1 の 32 行目は，

  ```
  return 0;
  ```

  と同じである．

- 4 行目 `#define NMAX 10`

  これもプリプロセッサーで，ソースプログラムをコンパイルする前に，文字列 `NMAX` を 10 に置き換える．文字列を置き換えるので，これを文字列置換と言う．いっぺんに多くの文字列や値を置き換えたいときに便利である．置き換える元の文字をすべて大文字で書くのが習慣となっている．

- 16 行目 `if(buf)`

  これには，比較の演算子がない．変数 `buf` の値の真/偽を判断することになる．最初，`if` 文を学習したときのことを思い出してほしい．C 言語では，

  - 値が 0 の時は，偽である．
  - 値が 0 以外の時は，真である．

  となっていた．

- 25 行目 `sum=n=0`

  代入演算子 (`=`) は右辺から評価していく．このことから，この文は次のように解釈する．

  - `n=0` が実行される．すなわち，`n` に `0` が代入される．
  - `n=0` という文の値は `0` となる．この `0` が `sum` に代入される．

リスト 1: 固定長の配列を使い合計を計算するプログラム

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define NMAX    10
5
6 int main(void)
7 {
```

```
 8      int buf,sum,count,n;
 9      int array[NMAX];
10
11      count=0;
12      do
13      {
14          printf("整数を入力してください(0を入力すると終了):");
15          scanf("%d",&buf);
16          if(buf)
17          {
18              array[count]=buf;
19              ++count;
20          }
21      } while(buf!=0);
22
23      /* 合計値を算出 */
24      printf("--入力されたのは以下の数です--\n");
25      for(sum=n=0;n<count;++n)
26      {
27          printf("%d \t",array[n]);
28          sum+=array[n];
29      }
30      printf("\n----\n以上の数の合計値は%dです。\n",sum);
31
32      return EXIT_SUCCESS;
33  }
```

3.2 配列のサイズを実行の最初に決める

3.2.1 原理

ユーザーに合計を計算するデータの数を最初に聞いて，その分，配列を用意する方法である．通常であれば，これは良い方法である．一般的に，よく使われる方法である．

ただし，最初に決めた配列をこえてのデータ入力ができない．ユーザーの気が変わって，データ数を増加させるときに問題が発生する．さらに，ユーザー自身，データの個数が分からないときには，対応ができない．

3.2.2 フローチャート

教科書の List 3-2(p.72) あるいはこのプリントのリスト 2 のプログラムのフローチャートを図 2 に示す．このプログラムは，おもに，

- ユーザーが指定したデータ数分の配列領域を確保する．
- キーボードの整数を配列に格納する．
- 格納された配列を合計する．

から構成される．このプログラムも，配列のサイズを確認しないで，配列にデータを格納している．これは問題である．

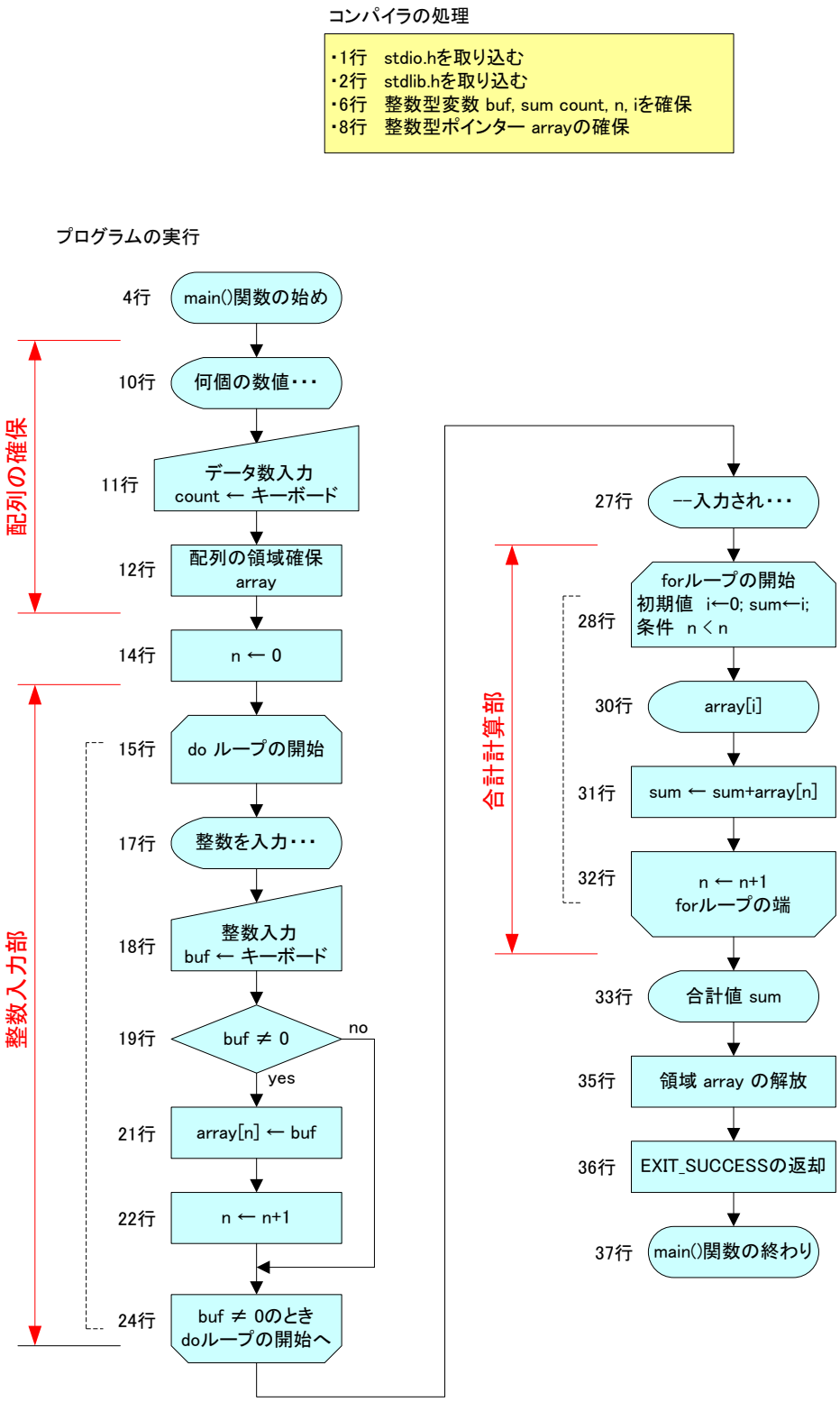

図 2: 最初に配列のサイズを決めるプログラムのリスト 1 のフローチャート.

3.2.3 プログラム

最初に配列のサイズを決めてそれに値を代入するプログラムをリスト 2 に示す．これは，教科書の List 3-2(p.72) と同じである．ここで使われているプログラムのテクニックは，次の通りである．

- 12 行目 `array=(int *)malloc(sizeof(int)*count);`
  少々複雑に見えるが，処理される順に見ていけば簡単である．
  1. 最初に `sizeof(int)` が評価される．関数 `sizeof()` は，型のバイト数を返す．ここでは，整数型 `int` のバイト数である 4 が返される．
  2. 次に，この 4 とデータ数である `count` の乗算が行われる．この結果は，データを格納するために必要なバイト数の計算になっている．
  3. `malloc()` 関数は，*Memory ALLOCate*(記憶割付) の略で，引数で渡されたバイト数のメモリーを確保して，その先頭のポインター (アドレス) を返す．
  4. `(int *)` はキャストと呼ばれる強制型変換である．`malloc` で返されるポインターは強制的に整数型としている．従って，ポインターを+1 加算すると，アドレスは 4 つ進むことになる．
  5. 整数型のポインター (アドレス) を整数型のポインター `array` に代入している．
- 7 行目 `int *array` と 21 行目 `array[n]`

  ポインターで宣言しているものを配列として使っている．これは，配列がどのように処理されているか，考えればよく分かる．配列 `array[n]` は，`*(array+n)` と評価される．すなわち，配列 `array[n]` は，ポインター `array` に `n` 加算したポインター (アドレス) が示す値と言うことである．このようなことから，配列で宣言していなくても，配列のように使うことができるのである．
- 35 行目 `free(array);`

  関数 `free()` は，そのプログラムで使用しているメモリー領域を開放するためにある．ここでは，12 行目の `malloc` で確保された領域を開放している．

リスト 2: 最初に配列サイズを決めてから，値を代入するプログラム．

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      int buf,sum,count,n,i;
7      int *array;
8
9      /* 入力するデータの個数を最初に聞いて，必要なメモリを確保 */
10     printf("何個の数値を入力しますか：");
11     scanf("%d",&count);
12     array=(int*)malloc(sizeof(int)*count);
13
14     n=0;
15     do
```

```
16      {
17          printf("整数を入力してください（0を入力すると終了）:");
18          scanf("%d",&buf);
19          if(buf)
20          {
21              array[n]=buf;
22              ++n;
23          }
24      } while(buf!=0);
25
26      /* 合計値を算出 */
27      printf("--入力されたのは以下の数です--\n");
28      for(sum=i=0;i<n;++i)
29      {
30          printf("%d\t",array[i]);
31          sum += array[i];
32      }
33      printf("\n----\n以上の数の合計値は %d です。\n",sum);
34
35      free(array);
36      return EXIT_SUCCESS;
37  }
```

3.3 実行時に配列のサイズを拡張する

3.3.1 原理

データの数が分からないので，最初に小さな配列を用意する．データ数が多く不足した場合，さらに大きな配列を用意する方法である．

この方法も良さそうですが，以下のような問題があります．

- 配列のコピー回数が多く，コストがかかりすぎる．
- コピー回数を減らすために，一度に多くの配列の領域を確保すると，大きな使われないメモリー領域ができてしまう可能性がある．

3.3.2 フローチャート

教科書の List 3-3(p.74) あるいはこのプリントのリスト 3 のプログラムのフローチャートを図 3 に示す．このプログラムは，おもに，

- キーボードの整数を配列に格納する．
  - 配列の範囲を超える場合，より多くのメモリーを確保する．
  - 確保された多くのメモリーに元のデータをコピーする．
  - 不要になったメモリー領域は，解放する．
- 格納された配列を合計する．

から構成される．このプログラムも，配列のサイズを確認しないで，配列にデータを格納している．これは問題である．

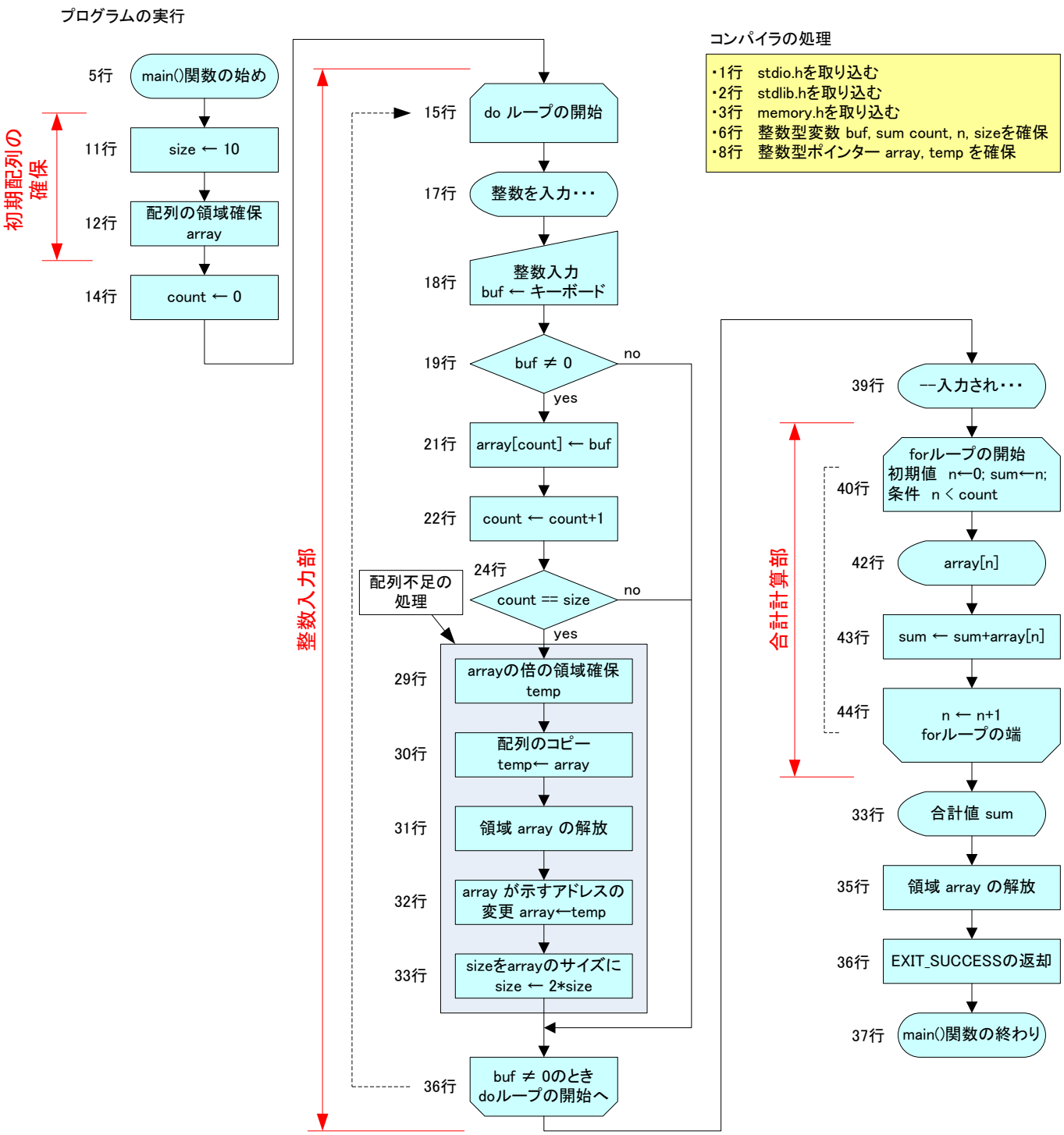

図 3: 番兵を使うリニアサーチのフローチャート.

3.3.3 プログラム

リスト 3 に示す．これは，教科書の List 3-3(p.74) と同じである．
このプログラムで使われている主なテクニックは，以下の通りである．

- 3 行目 `#include <memory.h>`
  これは，30 行目の `memcpy()` 関数を使うために必要で，その関数の宣言などが書かれている．

- 30 行目 `memcpy(temp, array, sizeof(int)*size)`
  `memcpy()` 関数は，*MEMory CoPY*(メモリー複写) の略で，メモリーの内容をコピーする．
  ここでは，ポインター `array` が示す場所から `4*size` バイト分のデータを，先頭をポインター `temp` が示す場所としてコピーする．

リスト 3: 実行時に配列のサイズを拡張する方法.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <memory.h>
4
5  int main(void)
6  {
7      int buf,sum,count,n,size;
8      int *array,*temp;
9
10     /* 最初の配列サイズは10 */
11     size=10;
12     array=(int*)malloc(sizeof(int)*size);
13
14     count=0;
15     do
16     {
17         printf("整数を入力してください（0を入力すると終了）:");
18         scanf("%d",&buf);
19         if(buf)
20         {
21             array[count]=buf;
22             ++count;
23             /* 配列が満杯になったら，倍の大きさに拡張する */
24             if(count==size)
25             {
26                 /* 新しい大きなメモリブロックを確保して，
27                    元の内容をコピー。
28                    realloc()を使っても同様の作業が行われる */
29                 temp=(int*)malloc(sizeof(int)*size*2);
30                 memcpy(temp,array,sizeof(int)*size);
31                 free(array);
32                 array=temp;
33                 size*=2;
34             }
35         }
36     } while(buf!=0);
37
38     /* 合計値を算出 */
39     printf("--入力されたのは以下の数です--\n");
40     for(sum=n=0;n<count;++n)
41     {
42         printf("%d\t",array[n]);
```

```
43              sum+=array[n];
44          }
45          printf("\n-----\n以上の数の合計値は%dです。\n",sum);
46
47          free(array);
48          return EXIT_SUCCESS;
49      }
```

4 リストを使う方法

4.1 配列とリストとの違い

リストと配列はともにデータの列をメモリーに確保するという点では似ている．ここで話を簡単にするために，データは整数とする．データは整数の数列である．配列の場合，この数列はメモリーの連続した領域に格納される．そして，配列の添え字を使って，目的のデータにアクセスする．

それに対して，リストは前後のデータのある位置をメモリーに入れておく．それを元に数列を構成する．

ここのところの図は，本日間に合わなかったので，来週渡す．

4.2 リストを使ったプログラム

4.2.1 原理

データを入力する毎に，新たにノードを作成して，そのノードにデータを格納している．前後のデータの関係は，リストを用いて表現している．

4.2.2 フローチャート

教科書の List 2-1(p.37) あるいはこのプリントのリスト 1 のプログラムのフローチャートを図 1 に示す．

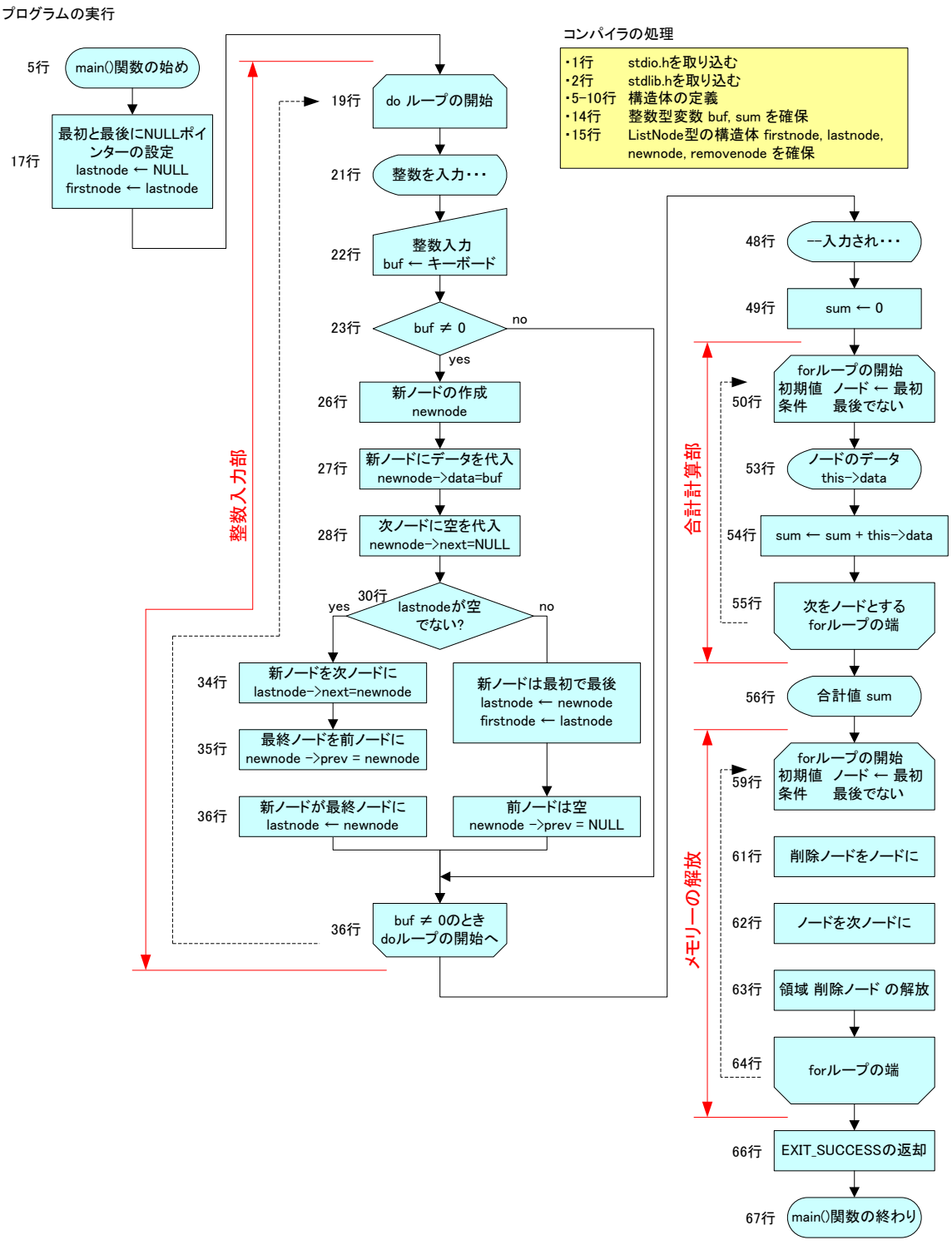

図 4: リストを使ったプログラムのリスト 4 のフローチャート.

4.2.3 プログラム

固定長の配列を使うプログラムのリスト 4 に示す．これは，教科書の List 3-4(p.75-77) と同じである．このプログラムで使われている主なテクニックは，以下の通りである．

- 5 行目 `typedef`
  これは，*TYPE DEFine*(型定義) と呼ばれるもので，型に別名をつける役割がある．ここでは，`struct tagListNode` と言う型を，`ListNode` と呼ぶことにしている．15 行目にそれが使われている．

- 17 行目 `lastnode=NULL`
  これは，ヌルポインター，あるいは空ポインターと呼ばれるものである．これは，`lastnode` が何も指し示していないことを保証するために，使っている．

- 27 行目 `newnode->data=buf;`
  `newnode` はポインターである．ポインターが指し示す構造体のメンバーにアクセスする場合，アロー演算子 (`->`) を使う．ここでは，ポインター `newnode` が示す構造体のメンバー `data` に `buf` の値を代入している．ポインターではなく，普通の変数となっている構造体の場合，そのメンバーにアクセスするにはドット演算子 (`.`) を使う．

リスト 4: リストを使ったプログラム

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* リストの要素（ノード）を表す構造体 */
5  typedef struct tagListNode
6  {
7      struct tagListNode *prev;     /* 前の要素へのポインタ */
8      struct tagListNode *next;     /* 次の要素へのポインタ */
9      int data;    /* この要素がもっているデータ */
10 } ListNode;
11
12 int main(void)
13 {
14     int buf,sum;
15     ListNode *firstnode,*lastnode,*newnode,*thisnode,*removenode;
16
17     firstnode=lastnode=NULL;
18
19     do
20     {
21         printf("整数を入力してください（0を入力すると終了）:");
22         scanf("%d",&buf);
23         if(buf)      /* 新たな入力があったら */
24         {
25             /* 新しいノードを作成 */
26             newnode=(ListNode*)malloc(sizeof(ListNode));
27             newnode->data=buf;
28             newnode->next=NULL;
29
30             if(lastnode!=NULL)
31             {
32                 /* すでにあるリストの末尾に
```

```
33                      新しいノードをつなげる */
34                  lastnode->next=newnode;
35                  newnode->prev=lastnode;
36                  lastnode=newnode;
37              }
38              else
39              {
40                  /* これが最初の要素だった場合 */
41                  firstnode=lastnode=newnode;
42                  newnode->prev=NULL;
43              }
44          }
45      } while(buf != 0);
46
47      /* 合計値を算出 */
48      printf("--入力されたのは以下の数です--\n");
49      sum=0;
50      for(thisnode=firstnode;thisnode!=NULL;
51                                  thisnode=thisnode->next)
52      {
53          printf("%d\t",thisnode->data);
54          sum+=thisnode->data;
55      }
56      printf("\n----\n以上の数の合計値は%dです。\n",sum);
57
58      /* リストの全ノードを削除 */
59      for(thisnode=firstnode; thisnode!=NULL;)
60      {
61          removenode=thisnode;
62          thisnode=thisnode->next;
63          free(removenode);
64      }
65
66      return EXIT_SUCCESS;
67  }
```

4.3 リストと配列の違い

表 1: 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

5 練習問題

[**問 1**] クラスメイトの名前を読み込んで，読み込んだ逆の順にディスプレイに表示するプログラムを作成せよ．ただし，プログラムは以下の通りとする．

– 名前はローマ字で，30 文字以内とする．
– 名前のデータはリストで管理するものとする．
– 教科書の p.75 List 3-3(プリントのリスト 4) を参考にすること．