第13章 探索

線形探索／2分探索／ハッシュ法／文字列探索と置換

データの集まり・集団の中から必要なデータを探し出すことを探索(search)といいます．C言語ではさまざまなデータ構造を扱うことができ，対象の形態によってどのような方法で探索するかも変わります．この章では，すでに格納されているデータ群の中から，目的のデータを探索する手法について解説します．線形な探索方法，2分探索，ハッシュ法による探索などです．2分探索木の探索については第9章で解説しています．

13.1 線形探索

線形探索は，データ群が単純な配列になっている場合に適用できます．なにもかまわず，先頭から順に見つかるまで探す方法です．逐次探索などとも呼びます．ウォーミング・アップを兼ねて，簡単な問題から始めましょう．

13.1.1 順に探す ― その1

すでに格納されている整数の1次元配列の中から，指定のデータを探すという問題から始めます．線形(linear)の意味を理解してください．

例題13.1.1 与えられた整数型配列から指定のデータを探索するプログラムを作成する．

■ アルゴリズム

一番単純な方法で，整列されていないデータに対しても適用できます．その動作を**図13.1.1**に示します．

いま探したいデータを7とします．配列の先頭から順に，そのデータと比較して，等しくなければ次に移動します．この動作を配列の末尾まで継続します．比較動作の回数は最大では配列の要素数となります．これは目的のデータが配列の末尾にあったという場合です．また逆に最少では，先頭にあった場合で，1回の探索で見つかることになります．平均では，n個要素とすれば，$n/2$ということになります．

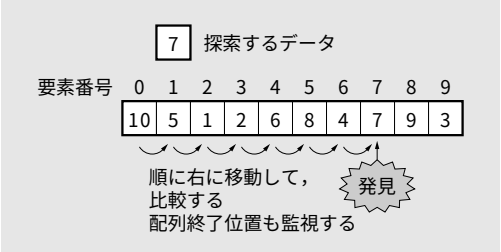

図13.1.1　線形探索の動き

■ 流れ図

線形探索の流れ図を**図13.1.2**に示します．探索位置を配列の先頭とするため，そのポインタ`i`を0に初期化してから，探索のループに入ります．ループの継続条件は，目的のデ

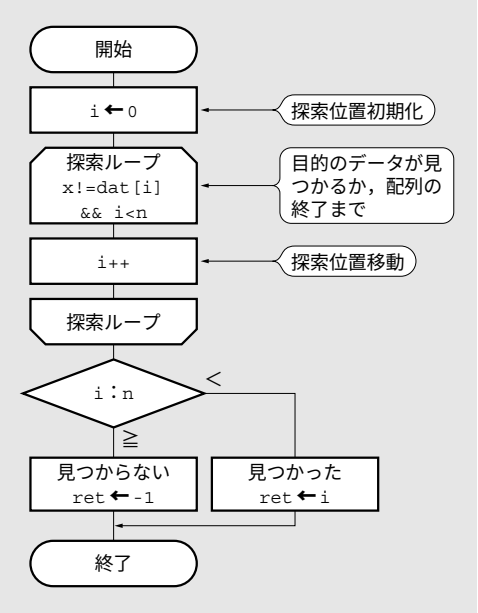

図13.1.2　線形探索の流れ図

リスト13.1.1　線形探索

```
/*  線形探索 -1                */
#include <stdio.h>

#define N 10                          /* データ数          */

int Lsearch1 ( int, int *, int );   /* 関数の原型宣言     */
int Lsearch11 ( int, int *, int );
int Lsearch12 ( int, int *, int );

int dat[] = {2,3,4,5,1,6,8,10,7,9}; /* データ            */

int main ( void )
{
  int a, i;

#ifdef DEBUG
  for ( i=0; i<N; i++ ){
    printf ("%d  ", dat[i] );
  }
  printf ("¥n");
#endif
  printf ("探すデータ > " );
  scanf ("%d", &a );                                    /*
探索データ入力  */
  i = Lsearch1 ( a, dat, N );
  if ( i>=0 ){
    printf ("そのデータは配列の %d 番要素にあります．¥n", i );
  } else {
    printf ("そのデータはありません．¥n");
  }
  return 0;
}

/*  線形探索関数 while利用     */
int Lsearch1 ( int x, int buf[], int n )
/*       x:     探したいデータ     */
/*       buf:   データ格納配列     */
/*       n:     配列の大きさ       */
{
  int i = 0;

  while ( x != buf[i] && i < n ){   /* データ不一致、最終 */
    i++;
  }
  if ( i>=n ){                      /* 最終に達していれば */
    return ( -1 );                  /* データなし         */
  } else {                          /* 最終未満ならば     */
    return ( i );                   /* 発見               */
  }
```

ータが見つかるまで，そして，配列の終了までです．配列内に見つかった場合には，iは配列の大きさnより小さいはずです．見つからなければ，ループを終了した時点でnになっているはずです．簡単ですからこれ以上の説明は不要と思います．

目的のデータが見つかったときは，その配列の位置を，見つからなかったときは，－1を返すものとしています．

■ プログラム･リスト

プログラムの例を**リスト13.1.1**に示します．データは初期値として与えてあります．線形探索の関数は，引数として探索する数値，探索する配列へのポインタ，配列の大きさが与えられるものとしました．また戻り値は，前述の通り，指定の数値が見つかった場合には，その要素番号を，見つからなかった場合には－1を返すものとしました．

13.1.2　順に探す ― その2

練習のために他の方法でも試しておきます．

例題13.1.2 例題13.1.1の探索関数をdo～while文またはfor文で作成する．

■ プログラム･リスト

リスト13.1.2にdo-whileによる方法(Lsearch11)と，for文による方法(Lsearch12)を示しました．関数部分のみを示しましたが，引数や戻り値は前の例題と同じです．

do-whileによる方法では，iの初期値を－1として，判定前に，その内容を＋1していることに注意してください．

リスト13.1.2　線形探索関数の別方法

```
/*  線形探索  do-while 利用   */
int Lsearch11 ( int x, int buf[], int n )
/*      x:    探したいデータ    */
/*      buf:  データ格納配列    */
/*      n:    配列の大きさ      */
{
  int i = -1;

  do {                                  /* データ不一致        */
    ;                                   /* かつ最終未満ならば  */
  } while ( x != buf[++i] && i < n );   /* 歩進する            */
  if ( i>=n ){                          /* 最終に達していれば  */
    return ( -1 );                      /* データなし          */
  } else {                              /* 最終未満ならば      */
    return ( i );                       /* 発見                */
  }
}

/*  線形探索  for 利用        */
int Lsearch12 ( int x, int buf[], int n )
{
  int i ;

  for ( i=0; i<n; i++ ){                /* 先頭から末尾まで    */
    if ( x == buf[i] ){                 /* データが一致したら  */
      break;                            /* 終了                */
    }
  }
  if ( i>=n ){                          /* 最終に達していれば  */
    return ( -1 );                      /* データなし          */
  } else {                              /* 最終未満ならば      */
    return ( i );                       /* 発見                */
  }
}
```

for文には，初期設定機能もありますので，とくに初期値は与えていません．等しい値が見つかった場合には，`break`でループを脱出しています．forの継続条件に`x!=buf[i]`を追加併記する方法も考えられます．

13.1.3　番兵を置く方法

前述の線形探索では，ループの継続条件はいずれの場合でも二つあります．それは，目的のデータが見つかったことと，探索位置が配列のサイズを越えていないかということです．このうち後者は目的のデータが配列内に見つからなかったことを想定しているからです．もし配列内に必ず目的のデータが存在するという仮定を立てれば不必要になります．プログラムも単純にできることになります．

そこで，配列の末尾に目的のデータを強制的に付加しておけば，目的のデータは必ず見つかるわけで，この条件を省略することができます．末尾に強制的に目的のデータを置くということは，終端を見張る番兵(sentinel)を置くようなものなので，番兵を置く方法などと呼ばれています．ただし，番兵を置くために配列に余裕がない場合には利用できません．

例題13.1.3　番兵を置いて線形探索するプログラムを作成する．

■ アルゴリズム

図13.1.3にその動きを示します．ここでは目的のデータ配列要素10個の末尾，11個目に番兵の部屋を設置しました．そしてそこに，目的のデータを設定しておきます．

同図(**a**)は，目的のデータが配列内に見つかった場合ですが，これは前述の線形探索の場合と同じです．配列内に目的のデータが存在しないと，探索は配列の末尾にまでいたります．そこには強制的に設定された番兵がいますから，見つかったということになります．しかし，配列要素を示しているポインタは，当初の範囲を外れていますか

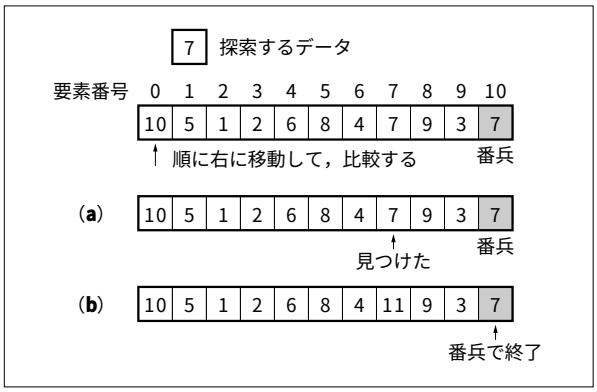

図13.1.3　番兵を置いた線形探索の動き

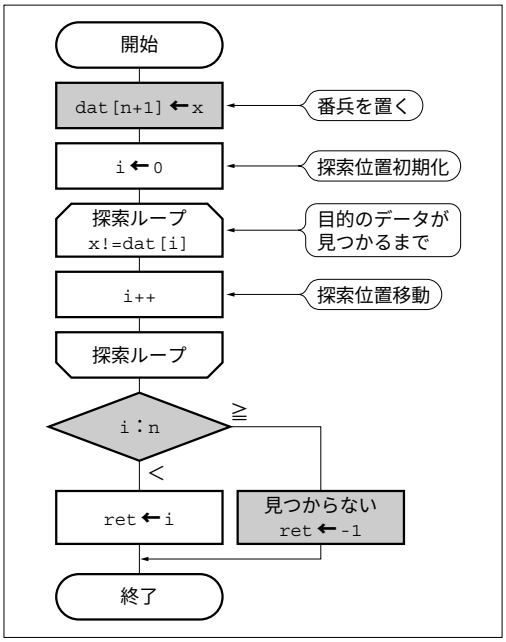

図13.1.4　番兵を置いた線形探索の流れ図

リスト13.1.3　番兵をおいた線形探索

```
/*  線形探索 -2 番兵を置く方法  */
#include <stdio.h>

#define N 10                              /* データ数            */

int Lsearch2 ( int, int * );              /* 関数の原型宣言      */

int dat[N+1]={2,3,4,5,1,6,8,10,7,9,0};  /* データ              */

int main ( void )
{
  int a, i;

#ifdef DEBUG                              /* 原データ表示        */
  for ( i=0; i<N; i++ ){
    printf ("%d  ", dat[i] );
  }
  printf ("¥n");
#endif
  printf ("探すデータ > " );
  scanf ("%d", &a );                                          /* 探索データ入力
*/
  dat[N] = a;                             /* 番兵設定            */
  i = Lsearch2 ( a, dat );                /* 探索                */
  if ( i>=0 && i<N){                                          /* データ数未満のとき
*/
    printf ("そのデータは配列の %d 番要素にあります．¥n", i );
  } else {                                /* データ数に等しいとき */
    printf ("そのデータはありません．¥n");
  }
  return 0;
}

/*  線形探索関数  2 */
int Lsearch2 ( int x, int buf[] )
{
  int i = 0;

  while ( x != buf[i] ){                  /* データが一致するまで */
    i++;                                  /* 歩進                */
  }
```