第12章 整列の応用

文字列配列のソート／構造体配列のソート

数値データについて整列を学習しましたが，それを応用して文字列データや構造体となっているデータの整列についても考えます．より実用的なテーマです．またここではC言語の標準ライブラリとして用意されている`qsort( )`関数を利用します．自分で作成したソート関数も味がありますが，既成のライブラリを利用することも，作業効率のうえから有効な手立てです．

12.1 文字列配列のソート

これまでの例題は，キーとなるデータは数値でした．整列の際の比較はその大小関係を見ることにより行うことができました．しかし文字列の整列順を変えたいという場合もあります．今までの単純な比較方法は利用できません．この節では，文字列を辞書順に整列する方法を考えます．

複数の同類の文字列を扱う場合には，文字列処理やコマンド・ラインの引数の節でも解説しましたが，文字列へのポインタ配列による方法が便利です．ここではその便利さの一端を紹介します．

例題12.1.1 与えられた複数の文字列を，それに対するポインタを整列することにより，整列を行う．

図12.1.1の左側のような文字列へのポインタ配列と，実際の文字列を想定します．このような文字列が初期化で与えられた，あるいはファイルから読み込んで生成されたとします．実際の文字列はそれぞれ長さも異なります．そのものを入れ替えて配置換えすることは困難です．

そこで，文字列へのポインタの配列だけを並べ替えるという手法を用います．同図右側は配列を整列したイメージです．文字列そのものは動いていません．整列後のポインタ配列にしたがって順に読み出せば，文字列そのものが整列されたような感覚で読み出すことができます．

□ `qsort( )`関数を使う

C言語には`qsort( )`という関数がライブラリとして用意されています．ここではソートの練習のためにこの関数を使ってみます．この関数は処理系によって違いはあるかもしれませんが，一般にはクイック・ソートのアルゴリズムを使って実現されています．そして実際に比較を行う関数も外部に定義できるようになっていますので，数値のみならず，文字列の整列にも応用できます．

`qsort( )`関数のプロトタイプは多少複雑ですが，

```
void qsort(void *base,size_t n,size_t size, int fnc(const void *a,const void *b));
```

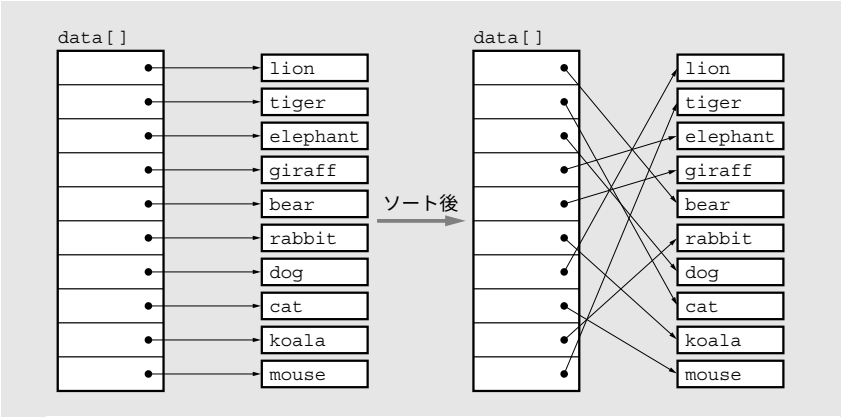

図12.1.1　文字列への配列のソート

です．ここでbaseは配列の先頭要素のポインタ，nは配列要素数です．そのオブジェクトの大きさがsizeです．そしてfncで指定される比較関数で実際の比較を行います．このように外部定義の比較関数を利用することから，比較対象が柔軟になり，数値そのものでなく，文字列などの比較によって並べ替えを行うことができるようになります．

fncの戻り値は第1実引数と第2実引数の大小関係により，a>bの場合には正の値，a=bの場合には0を，a<bの場合には負の値を返さなければなりません．strcmp()関数の戻り値と同じです．

そして文字列の配列を対象にした整列方法ですが，文字列を扱う場合には，複数の文字列へのポインタの配列を用意し，そのポインタのみを整列するという操作を行うのが一般的です．ここが数値そのものをキーとした整列と，文字列の扱いかたの大きな違いです．

□ 文字列の比較

C言語においての文字列処理は，8.4節などの解説でも述べたように，<string.h>にある文字列処理の関数群を利用するのが簡便です．ここでも文字列の比較には，strcmp()関数を利用します．

ただし注意なければならない点があります．前述qsort()関数の比較関数の引数は整列の対象へのポインタです．いま整列の対象となっているのは文字列へのポインタですから，qsort()から渡されるのは実際にはcharへのポインタになります．ところが比較関数の引数には汎用のポインタconst void * 型をとっています．そこでこれを実際の型(char **)にキャストして，間接参照して渡さなければなりません．そこで，与えられた文字列へのポインタstrに対して，

```
*(char **)str
```

などとします．(char **)は文字列へのポインタの配列であることを示し，しかもその中身を参照することを先頭の*で示しています．この関係を**図12.1.2**に示します．多重ポインタの詳細については「ポインタ編」を参照してください．

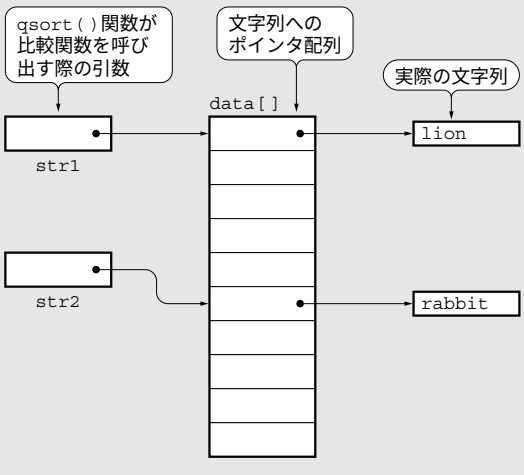

図12.1.2　比較関数の引数

■ プログラム･リスト

プログラムの例を**リスト12.1.1**に示します．qsort()関数を利用するためには，<stdlib.h>をインクルードしておきます．また前述の理由で，比較関数compare()を別途用意しています．中身はstrcmp()関数ですが，引数のキャスト変換のみをしています．

リスト12.1.1　文字列へのポインタ整列

```
/*  文字列のソート              */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define DMAX  10                                /* 最大数          */

char  *data[] = {"lion", "tiger", "elephant", "giraff", "bear",
      "rabbit", "dog", "cat", "koala", "mouse" };

int compare ( const void *, const void * );    /* 比較関数        */

int main ( void )
{
  int n;

  printf ("登録されている単語は¥n");
  for ( n=0; n<DMAX; n++ ){
    printf ("%p : %s¥n", data[n], data[n] );
  }
  printf ("¥n");

  qsort ( data,DMAX,sizeof(char *),compare );  /* 文字列配列比較  */

  printf ("¥n整列結果は¥n");
  for ( n=0; n<DMAX; n++ ){
    printf ("%p : %s¥n", data[n], data[n] );
  }
  printf ("¥n");
  return 0;
}

/*  比較関数                    */
int compare ( const void *str1, const void *str2 )
{

#ifdef DEBUG
  printf ( "compare¥t%s : %s¥n", *(char **)str1, *(char **)str2 );
#endif
  return ( strcmp( *(char **)str1, *(char **)str2 ));
}
```

```
登録されている単語は
0x401090 : lion
0x401095 : tiger
0x40109b : elephant
0x4010a4 : giraff
0x4010ab : bear
0x4010b0 : rabbit
0x4010b7 : dog
0x4010bb : cat
0x4010bf : koala
0x4010c5 : mouse

compare  lion : rabbit
compare  rabbit : mouse
compare  lion : mouse
compare  tiger : mouse
compare  lion : mouse
compare  elephant : mouse
compare  giraff : mouse
compare  bear : mouse
compare  rabbit : mouse
compare  koala : mouse
compare  dog : mouse
compare  cat : mouse
compare  lion : giraff
compare  dog : giraff
compare  elephant : giraff
compare  cat : giraff
compare  bear : giraff
compare  koala : giraff
compare  koala : giraff
compare  bear : dog
compare  dog : elephant
compare  elephant : cat
compare  dog : cat
compare  bear : cat
compare  koala : lion
compare  rabbit : tiger

整列結果は
0x4010ab : bear
0x4010bb : cat
0x4010b7 : dog
0x40109b : elephant
0x4010a4 : giraff
0x4010bf : koala
0x401090 : lion
0x4010c5 : mouse
0x4010b0 : rabbit
0x401095 : tiger
```

図12.1.3　文字列ソートの実行結果

プログラムには内容確認のために，整列前と整列後の，文字列へのポインタと文字列を表示しました．またデバッグと経過確認用として，比較対象の文字列を表示できるようにしています．

■ 実行結果

実行結果を**図12.1.3**に示します．初期の文字列の登録状態をみると，ポインタが順序良く並んでいます．整列後は，ポインタの並びは昇順にはなっていません．しかしポインタと文字列の対照は変わっていません．ポインタの配列の入れ替えが行われたことを物語っています．

また，整列されていく様子を観察すると，半分，半分としたクイック・ソートが行われていることも分かります．処理系によっても経過は異なります．

12.2　構造体配列のソート

実際のアプリケーションでは，扱うデータはレコード型，つまり構造体になっていることも多くあります．ここでは複数のメンバを持つ構造体の整列について解説します．メンバ個別にソート・キーになれること，前述qsort()関数の便利な使い方の一端を知ってください．

ここで例題とする構造体は，名簿管理のようなものを想定し，名前，年齢，身長そして体重の構造体を考えます．

```
typedef struct member {
  char  name[10];    /* 名前       */
  int    age;        /* 年齢       */
  int    hight;      /* 身長       */
  int    weight;     /* 体重       */
} MEMB;
```

とします．この構造体型を何度も利用するので，MEMBという型を定義しました．以降，この構造体の型はMEMBを利用して定義できます．その年齢，身長，体重ごとに整列を考えてみます．

12.2.1　構造体の内容移動による整列

前述のような構造体のデータを内容移動により整列する問題を考えてみます．

例題12.2.1 与えられた構造体にあるデータの，年齢をキーとして昇順に整列する．

リスト12.2.1　構造体移動によるソート(1)

```
/*  構造体の整列  -1          */
#include <stdio.h>

typedef struct member {                  /* 構造体型の定義      */
  char  name[10];
  int   age;
  int   hight;
  int   weight;
} MEMB;

MEMB memb[] = { { "赤井太郎", 18, 170, 60 },
                { "青田次郎", 19, 172, 62 },
                { "白岩三郎", 17, 173, 59 },
                { "緑川四郎", 20, 169, 61 } };

int main ( void )
{
  int i, j;
  MEMB  tmp;

#ifdef DEBUG
  printf ("元の名簿¥n");
  for ( i=0; i<4; i++ ){
    printf("%d : %s¥t%2d %3d %2d¥n", i,memb[i].name, memb[i].age,
      memb[i].hight, memb[i].weight );
  }
#endif

  for ( i=0; i<4; i++ ){                 /* 構造体自体の整列    */
    for ( j=3; j>i; j-- ){
      if ( memb[i].age > memb[j].age ){
        tmp     = memb[i];
        memb[i] = memb[j];
        memb[j] = tmp;
      }
    }
  }
#ifdef DEBUG
  printf ("整列後の名簿¥n");
  for ( i=0; i<4; i++ ){
    printf("%d : %s¥t%2d %3d %2d¥n", i,memb[i].name, memb[i].age,
      memb[i].hight, memb[i].weight );
  }
#endif
  return 0;
}
```