担当：　富井尚志　(tommy@ynu.ac.jp)

「アルゴリズムとデータ構造」講義日程

~~1. 基本的データ型~~
~~2. 基本的制御構造~~
~~3. 変数のスコープルール．関数~~
~~4. 配列を扱うアルゴリズムの基礎(1)．最大値，最小値~~
~~5. 配列を扱うアルゴリズムの基礎(2)．重複除去，集合演算，ポインタ~~
~~6. ファイルの扱い~~
~~7. 整列(1)．単純挿入整列・単純選択整列・単純交換整列~~
8. 整列(2)．マージ整列・クイック整列　← 本日の内容
9. 再帰的アルゴリズムの基礎．再帰におけるスコープ．ハノイの塔など．
10. バックトラックアルゴリズム．8 王妃問題など．
11. 線形リストを扱うアルゴリズム(1 回)
12. 木構造を扱うアルゴリズム(1) 基礎
13. 木構造を扱うアルゴリズム(2) 挿入，削除，バランスなど．
14. ハッシング
15. その他のアルゴリズム

第 7 回「整列（ソーティング）(2)」

☆ 整列（ソーティング）って？（復習）

- 与えられたものの集まりをある特定の「順番」に並べ直すこと．
- 形式的には，項目の集合　　a(1)，a(2)，a(3)，…, a(n)
  が与えられた時，ある順序　　a(k1), a(k2), a(k3), …, a(kn)
  に並べ変える．ただし，順序づけ関数 f に関して次の式を満足する．
  f(a(k1)) ≦ f(a(k2)) ≦ f(a(k3))≦ … ≦ f(a(kn))
- 通常，順序づけ関数の値は特定の計算規則によって値を求めるのではなく，項目内の特定の“成分”（C 言語の構造体に対応させるならメンバ）の値を直接用いる．この成分（の値）を「　　」という．

このマークの部分は講義中に書き入れて下さい

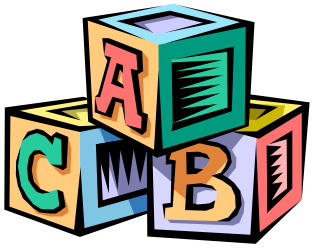
- 以降，以下の構造体を仮定．

```
struct item {
    int key;              /* メンバ key がキー．型は整数 */
    他のメンバの定義
};
```

※ キーが整数(int)型なので，**関係演算子**（<，＞等）がそのまま利用できる．

☆ 整列の種類（復習）

内部ソーティングと外部ソーティング



◎ 内部ソーティング

- 配列上にデータがあり，それを整列.
- 内部記憶上にランダムアクセス可能なデータがある場合.
- 一般に小規模なデータ向き
- 例 … 単純挿入／選択／交換整列，クイックソート

後述

◎ 外部ソーティング

- 順ファイル（sequential file）上にデータがあり，それを整列.
- 外部記憶（磁気ディスクなど）上にデータがある場合，順番に読むことができるが，任意の場所にアクセスすることは不可能／遅い.
- 一般に大規模なデータ向き
- 例 … マージソート

後述

任意のデータにアクセス可能

復習：【（順）ファイル(sequential file)】 ⟷ [　　]

直ちにアクセス可能なデータが特定の 1 個に限定されるデータの「列」．現在読みだし/書き込み可能なデータの位置は「現在位置(ファイル位置)」と呼ばれる．この構造の下，「列」への演算は，「空列の生成」「列の延長（末尾へのデータ書き込み）」「列の先頭へ移動」「次の項目への移動（データ読みだしならびに次の項目へ）」が定義される.

☆ 内部ソーティング（前回の続き）

超重要！

◎ クイックソート（交換に基づく）

- 考え方
  - 配列 a[ ]中のある項目 x（例えば（位置が）真中の項目）を選択し，そのキーを基準のキーとする.
  - この基準キーよりもキーが小さい項目のグループとキーが大きいグループの 2 つに「分割」する.
  - 上記の操作を分割されたそれぞれのグループに対して（再帰的に）適用する（分割した結果，要素が一つになるまで）.

→3	6	4	5	2	7	1←	左右から基準に向かう
3→	6	4	5	2	7	1←	大小関係異なる６，１
3	1→	4	5	2	7←	6	６と１を交換して次へ
3	1	4→	5	2←	7	6	５，２を発見
3	1	4	2	5	7	6	５，２を交換，５を確定
3	1	4	2	5	7	6	↓以下，分割後の組について再帰的に処理.
1	3	4	2	5	6	7	
1	3	4	2	5	6	7	
1	2	3	4	5	6	7	

- 「分割」

  列中のある位置を境にして，より添字の小さい側に，基準キーよりも小さいキーの項目たちが集まり，より添字の大きい側に基準キーよりも大きいキーをもつ項目たちが集まるように，配列の要素を交換していけばよい.

- 例

「２　５　３　４　１」の整列

↓基準キー（＝３）

２　５　３　４　１　　両端から見ていき，基準キーとの位置に関して逆転しているところを見つける．まず，５と１がみつかる．

２　１　３　４　５　　１と５を交換．さらに進めてキーの位置に関して逆転しているところを見つける．両者とも３に到達して入れ換えを必要とする項目がないことがわかる．以下，部分列「２　１」と部分列「４　５」を整列する．

「２　１」　の整列

↓基準キー

２　１　　同様に，交換すべき２と１をみつける．

１　２　　２と１を交換．もう交換すべき項目はない．

引続き，部分列「１」と部分列「２」を整列する．

「１」の整列

１項目だけなので何もしない．

「１」の整列 ==> 「１」

階層が深くなる

「２」の整列

１項目だけなので何もしない．

「２」の整列 ==> 「２」

「２　１」　の整列 ==> 「１　２」

「４　５」の整列

（同様なので省略）

「４　５」の整列 ==> 「４　５」

「２　５　３　４　１」の整列 ==> 「１　２　３　４　５」

- 効率

キーの比較回数 C(平均)＝2×ln 2×n×log n(＝ O(n×log n))

重要！必ず覚えること

項目間の移動(置換)回数 M(平均)＝(ln 2×n×log n)／3+g+1(＝ O(n×log n))

参考：単純挿入整列　C(平均)＝$O(n^2)$，M(平均)＝$O(n^2)$

単純選択整列　C＝$O(n^2)$，　M(近似平均)＝O(n×ln n)

単純交換整列　C＝$O(n^2)$，　M(近似平均)＝$O(n^2)$

※クイックソートの効率が良いことがわかる（∵　n が大きいとき　$n\times\log n < n^2$）

☆ 外部ソーティング

◎ 順ファイルとマージ操作

- **順ファイル**…一時に一つの項目だけがアクセス可能
- **マージ操作**

  複数の列があり，それぞれが整列しているとする．これらを何らかの方法で統合し一つの整列した列にする操作を「マージ（merge，併合）」という．

  → 各々の列については一時に一つの項目だけアクセスできれば良い．

例) 列 A「１　３　４」と列 B「２　５」のマージ（それぞれ整列されていることに注意）
両者の先頭要素をみて，小さい方を別の配列に書き込むという動作を繰り返す．

```
１　３　４　　２　５
↑　　　　　↑　　　　１の方が小さい．「１」
　↑　　　　↑　　　　２の方が小さい．「１　２」
　↑　　　　　↑　　　３の方が小さい．「１　２　３」
　　↑　　　　↑　　　４の方が小さい．「１　２　３　４」
　　　↑　　　↑　　　列 A 終り．列 B の残りをつける．「１　２　３　４　５」
```

先頭から順に一つずつしか項目へのアクセスがない → 順ファイルに都合が良い．

◎ 単純マージソート

- **考え方**

  すでに整列されている長さ m の部分列 2 つをマージすることにより長さが 2 m の整列部分列ができる．これを，m＝1 から n まで（m は 2 の冪乗になる）繰り返す．ただし，n は整列すべき列の長さ．（テストプログラム参照）

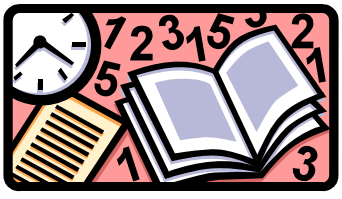
- マージ動作を行なうためには 3 つの順ファイル（入力用 2 つ，出力用 1 つ．もちろん配列でも可）があれば良いが，次のマージ動作の際には，完成した部分列の列を再度二つに配分する必要がある．そこで最初から出力する際に 2 つの順ファイルに振り分ける．（テストプログラム参照）

- **効　率**

  記憶領域計算量は内部ソートよりも当然劣る．

  マージソートの場合は外部記憶装置などのように読み書きが遅い装置を使うので，移動操作に関心がある．

  項目の移動(置換)回数 M ＝ n×log n (＝ O(n×log n))

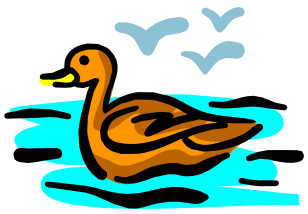


```
/*************************************************************
     アルゴリズムとデータ構造
     サンプルプログラム
     <<クイックソート(高速整列)>> qsort.c
     copyright (c) T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/
#include <stdio.h>
#define  CHARLEN  20
/* １データ分を表す構造体 */
struct item {
  int    key;
  char   name[CHARLEN];
};
void printitem(struct item a[], int n);
void quicksort(struct item a[], int left, int right);
```

main 関数

```
int main(void)
{
#define N 8
  struct item table[N] = {
    { 65, "国語"},{ 90, "数学"},{ 85, "理科"},{ 70, "社会"},
    { 86, "英語"},{ 92, "体育"},{ 63, "音楽"},{ 85, "美術"}
  };
  int m = N;

  printf("table[]: ");
  printitem( table, m );
  printf("quicksort(table,0,m-1)¥n");
  quicksort(table,0,m-1);
  printf("table[]: ");
  printitem(table,m);
  return 0;
}
```



```
/* 配列内のデータを印刷する */
void printitem(struct item a[], int n)
{
  int i;

  for(i=0; i<n; i++)
    printf("<%d,%s> ", a[i].key, a[i].name);
  printf("¥n");
}

/* クイックソート */
void quicksort( struct item a[], int left, int right )
{
  int i, j;
  struct item x, w;
```

```
50
51      /* 整列すべき配列部分の項目数が 2 以上なら整列作業を行なう */
52      /* データが 1 個以下なら何もしない．(ただし一番外側の if はな */
53      /* くても正常に動作する) */
54      if ( left < right ) {
55        i = left; j = right;
56        /* 整列すべき範囲(添字 left～right)のちょうど真中のデータ(x)の */
57        /* キーを基準にして，それより大きいキーを持つグループと，*/
58        /* 小さいキーを持つグループに分割する．  */
59        x = a[ (left + right) / 2 ];
60        do {
61
62          while ( a[i].key < x.key )
63              /* x 以上のキーを持つ a[i] を先頭から順に探す */
64              i ++;
65
66          while ( x.key < a[j].key )
67              /* x 以下のキーを持つ a[j] を末尾から順に探す */
68              j --;
69
70          if ( i <= j ) {
71            if ( i < j ) {
72              /* 交換すべき状態なら a[i]と a[j]の内容を入れ換える */
73              w = a[i];  a[i] = a[j];  a[j] = w;
74            }
75            i ++;  j --;    /* 次の要素を調べるために添字を変化させる */
76          }
77        } while (i <= j);      /* 調べる要素が残っている時には続ける */
78        /* この時点で分割終了．a[]の内容は以下のようになっている．
79         *    a[left]～a[j]     : x のキー以下のキーを持つ項目が入っている
80         *    a[j+1]～a[i-1]   : x のキーと同じキーを持つ項目が入っている
81         *    a[i]～a[right]    : x のキー以上のキーを持つ項目が入っている
82         */
83        if ( left < j )
84          quicksort(a, left, j); /* x 以下の項目が入っている範囲で再帰実行 */
85        if ( i < right )
86          quicksort(a, i, right);
87          /* x 以上の項目が入っている範囲で再帰実行 */
88      }
89    }
90
91
92    【実行結果】
93    table[]: <65,国語> <90,数学> <85,理科> <70,社会> <86,英語> <92,体育>
94    <63,音楽> <85,美術>
95    quicksort(table,0,m-1)
96    table[]: <63,音楽> <65,国語> <70,社会> <85,美術> <85,理科> <86,英語>
97    <90,数学> <92,体育>
```

i を左端を表す変数，j を右端を表す変数としている．

left + right が奇数の場合は切り捨て．
例）left=0, right=3 → x=1

入れ替え作業

処理の流れの補足

最初の状態　　基準 x

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
65	90	85	70	86	92	63	85

行番号 46 で,　　quicksort(a[],0,7)として quicksort の関数が呼ばれた.
行番号 59 で,　　基準 x = a[(0+7)/2] = a[3] = 70 となる.

１回目の検査

行番号 62～64 で,　a[1](=90) > x だから，i = 1 となる.
行番号 66～68 で,　a[6](=63) < x だから，j = 6 となる.
行番号 73 の入れ替えにより a[i](=a[1])と a[j](=a[6])が交換され次のようになる.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
65	63	85	70	86	92	90	85

行番号 75 により,　i = 2，j = 5 となる.
行番号 77 の while(i <= j)が満たされているので，行番号 61 に戻る.

２回目の検査

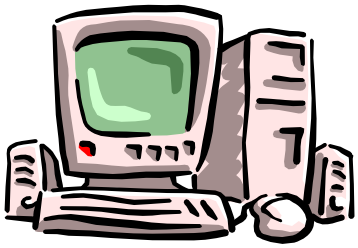
行番号 62～64 で，a[2](=85) > x であり，i = 2 となる（i の値は変化しなかった）.
行番号 66～68 で，該当なしのため，j = 3(=k)となる.
行番号 73 の入れ替えにより a[i](=a[2])と a[j](=a[3])が交換され次のようになる.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
65	63	70	85	86	92	90	85

行番号 75 により,　i = 3，j = 2 となる.
行番号 77 の while(i <= j)が満たされなくなるので行番号 61～78 の do～while ループの処理が終了する.
行番号 83 の if 文の条件式(left(=0)<j(=2))が真なので quicksort(a,0,2);となる.
行番号 85 の if 文の条件式(i(=3)<right(=7))が真なので quicksort(a,3,7);となる.

a[2]と a[3]の間で，それ以下の要素，それ以上の要素になったことに注目！

```
/*************************************************************
     アルゴリズムとデータ構造
     サンプルプログラム
     <<併合(マージ)整列>> merge.c
     copyright (c) T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/
#include <stdio.h>
#define  CHARLEN  20
#define  MAX  30
/* 1 データ分を表す構造体 */
struct item {
  int    key;
  char   name[CHARLEN];
};
void printitem(struct item a[], int n);
void mergesort(struct item a[], int n);
```



main 関数

```
int main(void)
{
  #define N 8
  static struct item table[N] = {
    { 65, "国語"},{ 90, "数学"},{ 85, "理科"},{ 70, "社会"},
    { 86, "英語"},{ 92, "体育"},{ 63, "音楽"},{ 85, "美術"}
  };
  int m = N;

  printf("table[]: ");
  printitem(table,m);
  mergesort(table,m);
  printf("mergesort(table,m)¥n");
  printf("table[]: ");
  printitem(table,m);
  return 0;
}

/* マージソート */
void mergesort(struct item a[], int n)
{
  struct item t[2][2][MAX];
  int l[2][2];
  int s,d,dt;
  /* s,d は併合の入力および出力に相当する配列のインデクスを表す.
  t[s][][] は 入力の配列(の組. 2 つある. )
  t[d][][] は 出力の配列(の組. 2 つある. )を表す.
  つまり, 3 次元配列 t は 2 つの入力用順ファイルと 2 つの出力用順
  ファイルを模している.
  (外部記憶をつかうときには t へのアクセスを書き直せばよい)
```

```
50      dt は，出力の配列の組のうち，どちらに併合した結果を書き出すかを
51      表すインデクスである．t[d][dt][] は 出力の配列の組のうち現在出力に
52      なっている配列を表す．
53
54       l[][] は各配列中にあるデータの数を表している．
55       添字は，t[][][] の最初の二つの次元に対応する．
56       */
57      int i,j,k,p,q,r;
58
59      s = 0; d = !s;
60
61      /* 初期データを二つの入力列に振り分けておく */
62      for( i=0; i<n/2; i++ )
63        t[s][0][i] = a[i];
64      for( j=0; i<n; i++, j++ )
65        t[s][1][j] = a[i];
66      l[s][0] = n/2;   l[s][1] = n-n/2;
67      l[d][0] = 0;     l[d][1] = 0;
68
69      /* t[s][0][], t[s][1][] 内のデータをマージして，
70       * t[d][0][], t[d][1][] に振り分けて書き込む
71       * l[d][0], l[d][1] は t[d][0][],t[d][1][] にどこまで書き込んだかを
72       * 示すインデクスでもある
73       */
74
75      p = 1;    /* p は一度にマージする項目数．最初は 1 つづつ */
76      do {
77        /* i, j はそれぞれ入力列 t[s][0][], t[s][1][]において
78        次に考慮すべき項目を表す添字 */
79        i = 0;   j = 0;   dt = 0;
80        do {
81          /* q は t[s][0][] からマージする項目数．通常，p づつであるが，
82          最後に端数がでる．  */
83          if ( l[s][0] >= p )  q = p;  else  q = l[s][0];
84          l[s][0] -= q;
85          /* r は t[s][1][] からマージする項目数．通常，p づつであるが，
86          最後に端数がでる．  */
87          if ( l[s][1] >= p )  r = p;  else  r = l[s][1];
88          l[s][1] -= r;
89          k = l[d][dt];
90          /* k は出力列において書き込むべき位置を表す添字(==l[d][dt]) */
91          /* t[s][0][] から q 項目，t[s][1][] から r 項目マージし，
92          t[d][dt][k] から順に書いていく */
93          while ( q != 0  &&  r != 0) {
94            if ( t[s][0][i].key  <  t[s][1][j].key ) {
95              t[d][dt][k] = t[s][0][i];
96              i++;  q--;
97            } else {
```

! は否定演算子. s=0 なら d=1, s=1 なら d=0 となる.
いまは s=0 なので d=1 となる.

a[4]～a[7]を t[0][1][]に代入.

l[0][0]=4, l[0][1]=4,
l[1][0]=0, l[1][1]=0

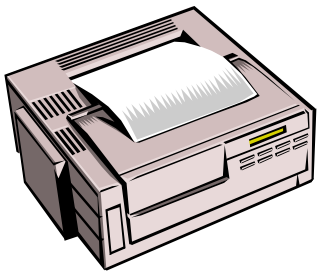
```
            t[d][dt][k] = t[s][1][j];
            j++;  r--;
          }
        k++;
        }
        while ( q != 0 ) {
          /* t[s][0][] にデータが残った場合，それをコピーする */
          t[d][dt][k] = t[s][0][i];
          i++;  q--;  k++;
        }
        while ( r != 0 ) {
          /* t[s][1][] にデータが残った場合，それをコピーする */
          t[d][dt][k] = t[s][1][j];
          j++;  r--;  k++;
        }
        l[d][dt] = k;
        /* 1 回のマージ操作終了 */
        dt = !dt;
        /* 出力先の配列を切替えてマージデータを振り分ける */
      } while ( l[s][0] > 0  ||  l[s][1] > 0 );

      d = s;  s = !s;
      p = p*2;
        /* 入力配列と出力配列を入れ換え，一度にマージする長さを倍
        にして，次のステップに進む */
    } while (p < n);

    for ( i=0; i<n; i++ ) {
      a[i] = t[!d][!dt][i];
      /* 最後に配列のインデクスが 1,0 反転するので,
       * t[!d][!dt][] に答が得られる */
    }
}

/* 配列内のデータを印刷する */
void printitem(struct item a[], int n)
{
  int i;

  for( i=0; i<n; i++ )
    printf("<%d,%s> ", a[i].key, a[i].name);
  printf("¥n");
}
```



【実行結果】

```
table[]: <65,国語> <90,数学> <85,理科> <70,社会> <86,英語> <92,体育>
<63,音楽> <85,美術>
mergesort(table,m)
table[]: <63,音楽> <65,国語> <70,社会> <85,美術> <85,理科> <86,英語>
<90,数学> <92,体育>
```

マージソートのプログラムの補足

最初の状態：

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
65	90	85	70	86	92	63	85

もとのデータを２つに分割：

（入力用配列１）t[0][0][ ]：65, 90, 85, 70　　長さ l[0][0] = 4
（入力用配列２）t[0][1][ ]：86, 92, 63, 85　　長さ l[0][1] = 4
（出力用配列１）t[1][0][ ]：　　長さ l[1][0] = 0
（出力用配列２）t[1][1][ ]：　　長さ l[1][1] = 0

■ p = 1

各入力用配列から 1 つずつマージし，最初の出力先である t[1][0][ ]に書く．

t[1][0][ ]：65, 86　　長さ l[1][0] = 2

次に出力先を t[1][1][ ]に変えて書く．

t[1][1][ ]：90, 92　　長さ l[1][1] = 2

次に出力先を t[1][0][ ]に変えて引き続き書く．

t[1][0][ ]：65, 86, 63, 85　　長さ l[1][0] = 4

次に出力先を t[1][1][ ]に変えて引き続き書く．

t[1][1][ ]：90, 92, 70, 85　　長さ l[1][1] = 4

■ p = 1×2 = 2

t[1][0][ ]，t[1][1][ ]を入力側，t[0][0][ ]を出力側に切り替えて，2 つずつマージして同様な処理を実行する．

t[0][0][ ]：65, 86, 90, 92　　長さ l[0][0] = 4

次に出力先を t[0][1][ ]に変えて書く．

t[0][1][ ]：63, 70, 85, 85　　長さ l[0][1] = 4

■ p = 2×2 = 4

t[0][0][ ],t[0][1][ ]を入力側，t[1][0][ ]を出力側に切り替えて 4 つずつマージする．

t[1][0][ ]：63, 65, 70, 85, 85, 86, 90, 92　　長さ l[1][0] = 8

出力先を t[1][1][ ]に切り替える必要はもはやない．

t[1][1][ ]：　　長さ l[1][1] = 0

t[1][0][]が最終的な結果となる．

非常に複雑に見えますが，動作を理解すればさほどでもありません．マージソートの原理は単純ですから，しっかり頭にいれておいて下さい．

MEMO