I117（6）整列

知念

北陸先端科学技術大学院大学 情報科学研究科
School of Information Science,
Japan Advanced Institute of Science and Technology

おことわり

- 整列（ソート）各種技術はアルゴリズムの講義で
- ここでは軽く、ソートの利用方法を紹介する
- メインメモリ上のソート関数 qsort
  - 多く場合 qsort で事足りる
  - 例外
    - 高速性が要求される => 関数の形態を変更
    - 特殊なデータ内容 => アルゴリズムを変更
    - 配列では格納されていない

※ 外部メモリを使うソートは扱わない

qsort

- クイックソート（quick sort）の汎用的実装
- 多くの処理系で利用可能

```
void qsort(void *base, size_t nel,
    size_t width,
    int (*compar)(const void *,
                  const void *));
```

- 引数
  - 場所、要素数、要素サイズ、比較関数
  - 間違えやすいので注意

比較関数

```
int (*compar)(cons void*, const void*);
```

qsort は比較関数で整列順を決める
返り値に前提をおく

- $>$0 大きい
- $==$0 同じ
- $<$0 小さい

比較の詳細は比較関数の実装任せ

比較関数: int の比較

※ Solaris の man から引用

```
static int intcompare(const void *p1,
    const void *p2)
{
    int i = *((int *)p1);
    int j = *((int *)p2);
    if (i > j)  return (1);
    if (i < j)  return (-1);
    return (0);
}
```

比較関数: int の比較 *(cont.)*

実はこの程度で十分

```
static int intcompare(const void *p1,
    const void *p2)
{
    int i = *((int *)p1);
    int j = *((int *)p2);
    return i-j;
}
```

呼び出し例

```
int ar[] = { 41, 3, 92, 7, 13, 10};
int i;

qsort((void *)ar,
    sizeof(ar)/sizeof(int),
    sizeof(int), intcompare);
for (i = 0; i < sizeof(ar)/sizeof(int);
        i++) {
    printf("%d ", ar[i]);
}
printf("\n");
```

比較関数: int の比較 *(cont.)*

結果 — 昇順↗

```
% ./a.out
3 7 10 13 41 92
```

比較関数の引き算を j - i にすると降順↘

```
% ./a.out
92 41 13 10 7 3
```

比較関数: 文字列

文字列の大小比較に関数 strcmp を使う

```
int
cmp(const void *x, const void *y)
{
    char *xp, *yp;
    xp = *(char**)x;
    yp = *(char**)y;
    return strcmp(xp, yp);
}
```

呼び出し側

```
char *ar[] = { "41", "3", "92",
         "7", "13", "10"};
int i;
qsort((void *)ar,
    sizeof(ar)/sizeof(char*),
    sizeof(char*), cmp);
for (i = 0;
     i < sizeof(ar)/sizeof(char*); i++) {
    printf("%s ", ar[i]);
}
printf("\n");
```

比較関数: 文字列 *(cont.)*

結果

```
10 13 3 41 7 92
```

文字列なので、”13”＜”3” ”41”＜”7”

- strcmp の挙動

1) 先頭から文字の大小比較
2) 短い文字列が小さい返り値

比較関数: 文字列 *(cont.)*

アルファベットにすると

```
char *ar[] = { "openlog", "syslog",
        "closelog", "syslogmask"};
```

結果

```
closelog openlog syslog syslogmask
```

降順は -strcmp(xp,yp); か strcmp(yp,xp); に

```
syslogmask syslog openlog closelog
```

比較関数: 文字列 *(cont.)*

空白を含む長い文字列

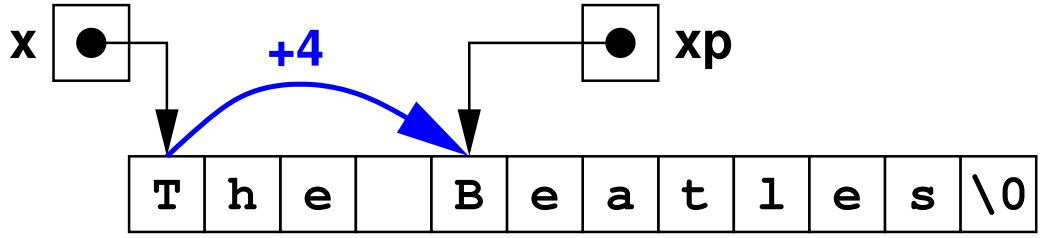
```
char *ar[] = {"The Beatles", "U2",
    "Norah Jones", "Johnny Cash",};
```

結果 — 1項目ずつ改行してある

```
Johnny Cash
Norah Jones
The Beatles
U2
```

データ内容に合わせた変更 — The を除いて比較

```
int cmp(const void *x, const void *y) {
    char *xp, *yp;
    xp = *(char**)x;
    if(strncasecmp(xp,"the ",4)==0) {
        xp += 4; }
    yp = *(char**)y;
    if(strncasecmp(yp,"the ",4)==0){
        yp += 4; }
    return strcasecmp(xp, yp);
}
```



比較関数: 文字列 *(cont.)*

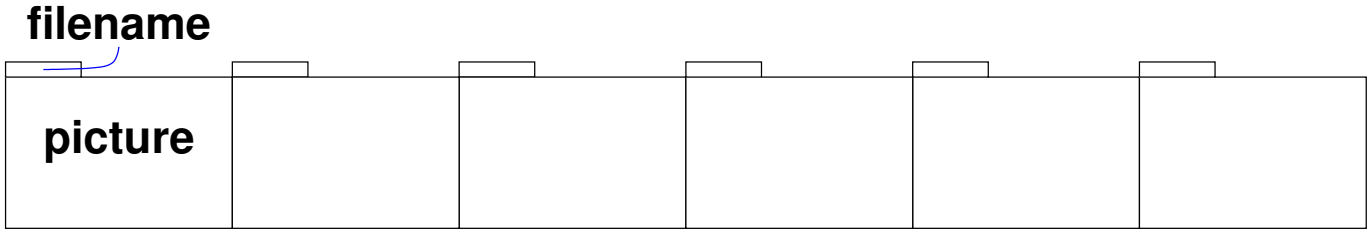
- The だけでなく空白も含めて比較
  - There, Them 等の単語を誤処理しないため
- 大文字小文字混在なので strcasecmp を使う

結果

```
The Beatles
Johnny Cash
Norah Jones
U2
```

参照配列によるソート

- 対象データ群が大きい
  - ソートに伴うコピーに時間がかかる
    数10MB/件 の画像を名前でソートしたり



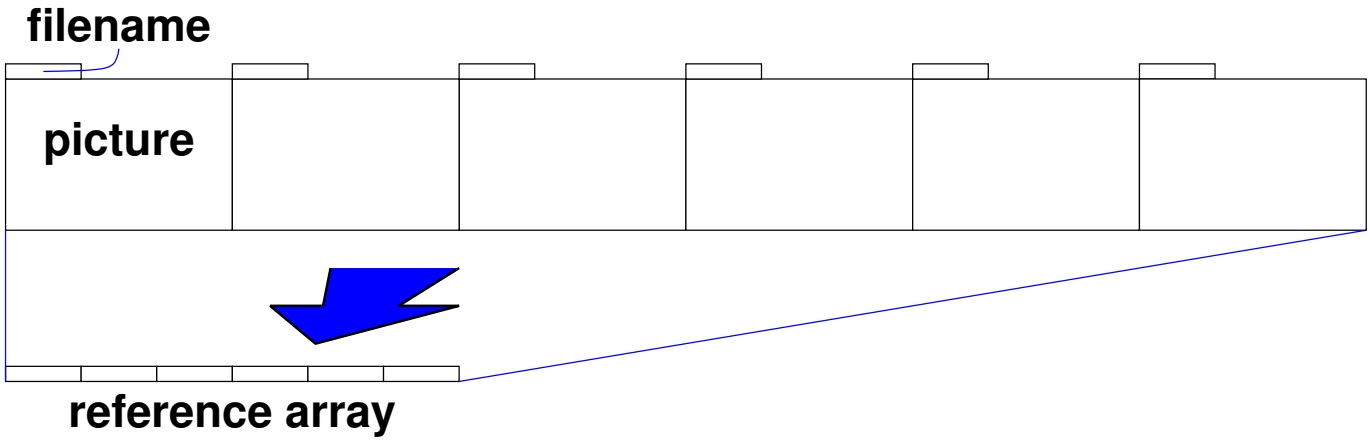
sizeof(filenames)<<sizeof(pictures)

- 対象データが線形に格納されていない
  - qsort が扱えない

=> ソートのために配列を別に設ける

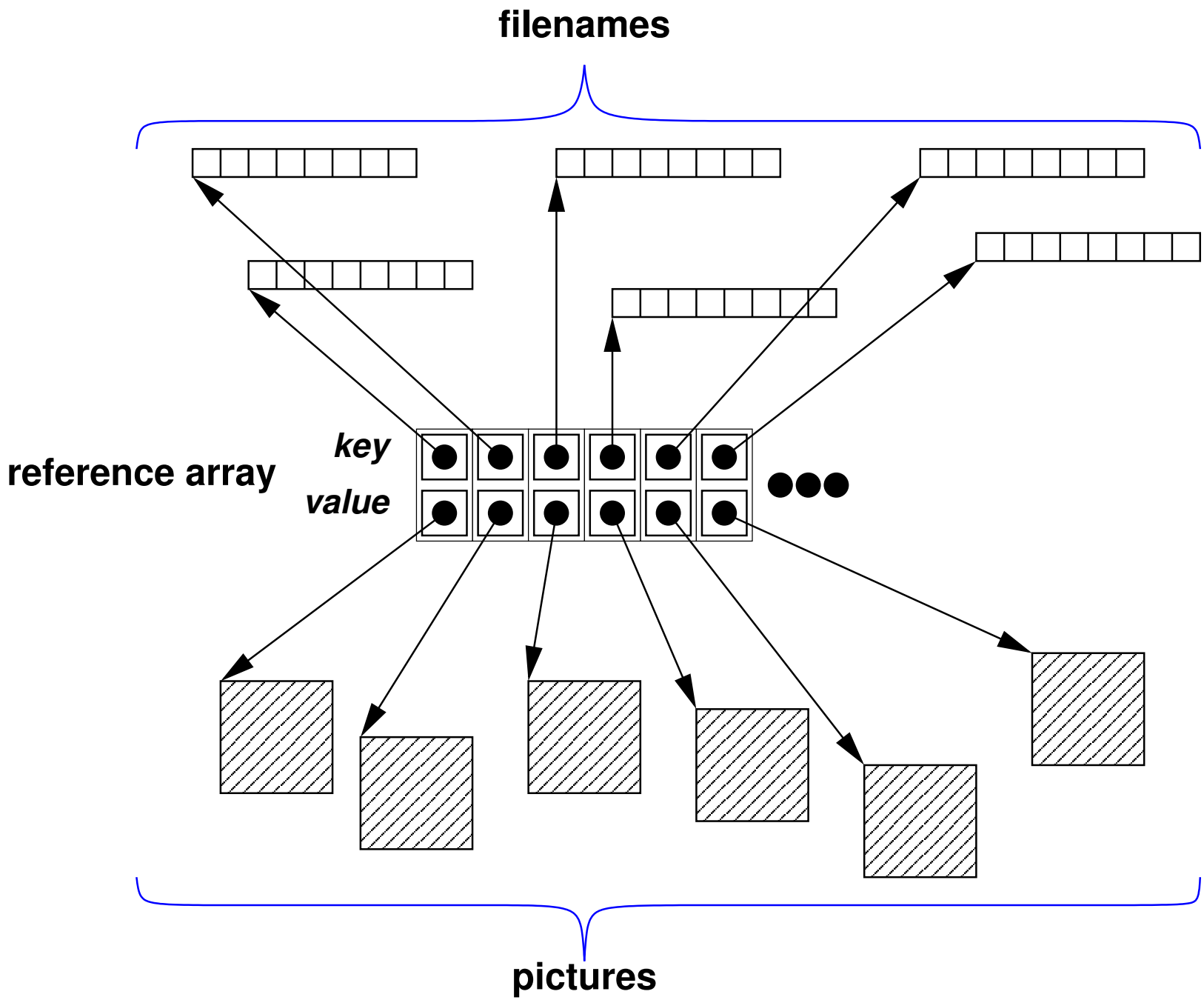
参照配列によるソート *(cont.)*

大略



ソート結果を使うために参照も設ける

```
typedef struct {
    char *key;
    void *value;
} cell;
```


filenames
reference array
key
value
pictures

参照配列によるソート *(cont.)*

こうすると...

- ソート処理中のコピーは参照配列だけ
  - ◇ 小さな領域（8バイト）に押し込められる
- 参照配列のソート後、参照配列をたぐるだけで対象データがソートされたことになる。

ちなみに、参照メンバ value の型を void* にしたのは任意のポインタを扱うため

画像は大変なので、ここでは文字列でサンプルを作る

```
char* pickstr2key(void *src) {
    char *dst, *p = src, *q;
    dst = (char*)malloc(strlen(src)+1);
    if(strncasecmp(src, "the ", 4)==0) {
        p += 4;
    }
    q = dst;
    while(*p) {
        *q++ = toupper(*p++);
    }
    *q = '\0';
    return dst;  }
```

conv は pickstr2key を使う

```
int str2cell(cell dst[], void *src[],
    int nlen, char *(conv)(void*))
{
    int i;
    for(i=0;i<nlen;i++) {
        dst[i].key = conv(src[i]);
        dst[i].value = src[i];
    }
    return 0;
}
```

参照配列によるソート *(cont.)*

呼び出し側

- 配列ar の長さn は sizeof(ar)/sizeof(ar[0]) で算出
- 参照配列 refar の長さは sizeof(cell)*n

```
char *ar[] = {"The Beatles", "U2",
    "Norah Jones", "Johnny Cash",};
cell *refar;
int n = sizeof(ar)/sizeof(ar[0]);
refar = (cell*)malloc(sizeof(cell)*n);
str2cell(refar, (void*)ar, n,
        pickstr2key);
```

参照配列によるソート *(cont.)*

あとは qsort を適用して、表示する

```
qsort(refar, n, sizeof(refar[0]),
          cellkeycmp);
for(i=0;i<n;i++) {
    printf("%-16s %s\n",
          refar[i].value, refar[i].key);
}
```

- cellkeycmp は単純な strcmp で十分
- refar のソート結果から ar の内容を表示

参照配列によるソート *(cont.)*

pickstr2key で加工(”the ”除去、大文字化)しているので単純な比較で良い

```
int cellkeycmp(void const *xp,
        void const *yp)
{
    char *x; char *y;
    x = ((cell*)xp)->key;
    y = ((cell*)yp)->key;
    return strcmp(x,y);
}
```

参照配列によるソート *(cont.)*

結果

```
The Beatles             BEATLES
Johnny Cash             JOHNNY CASH
Norah Jones             NORAH JONES
U2                      U2
```

- 元データ操作なし、ソート中のコピー処理が軽い
- ソート前に加工済、キー比較の処理が軽い
- 参照配列の分だけメモリを消費（デメリット）