Cプログラミング演習1(再) ８

講義では、Cプログラミングの基本を学び
演習では、実践的なプログラミングを通して学ぶ

アルゴリズム

アルゴリズムとは

料理で言えばレシピ

要するに、処理の段取り

どのような処理をどのような順で行うか

アルゴリズムによって処理効率に大きな違いがあることも

文字列を対象としたソート

今まで取り上げた選択ソートもクイックソートも、並び替え対象は「数」であった。

今度は、文字列を対象としたソートを考えてみよう

文字列からなるファイル

lang.sist.chukyo-u.ac.jp/Classes/C/terms.txt
を対象とする。この中身は次のようなもの:

```
bus
cell
address
resource
random
access
map
selector
register
install
load
run
execute
fetch
decode
code
writeback
cache
device
driver
...
```

文字列のソートの設計

lang.sist.chukyo-u.ac.jp/Classes/C/terms.txt
を、「アルファベット順に並び替える(sort）」プログラムを作る
それには（前のソートのプログラム作成の経験から）次のようにすれば良さそう:

1. ファイルからデータを読み込み、配列に記憶する
2. 選択ソート、もしくはクイックソートの手法により配列の要素を並び替える
3. 並び替えた配列の要素をモニターに表示する、もしくはファイルに書き出す

どこにも問題なく、できそう？？？

文字列のソートで必要なもの

1. ファイルからデータを読み込み、配列に記憶する

(1) データの読み方、配列への格納

2. 選択ソート、もしくはクイックソートの手法により配列の要素を並び替える
3. 並び替えた配列の要素をモニターに表示する、もしくはファイルに書き出す

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,,j) と quickSort(array, j+1,*l*) を行って終了

(2) 文字列データの比較方法

文字列を配列に格納

まずは、文字列を配列に格納することを考えよう。

それには「文字列」を要素とする配列を用意すれば良い。

今まで扱っていた配列はほとんど、「数」を要素とする配列だったので、配列名を array とすると

```
int  array[100];
```

とすればよかった

文字列を配列に格納

「数」を要素とする配列は

```
int  array[100]
```

として用意出来たことを考えると、

「文字列」を要素とする配列は...

```
char array[100];
```

でできるだろうか？

文字列を配列に格納

「文字列」を要素とする配列は、実は前に検討していた

文字列の長さの最大を LEN

覚えるべき文字列の個数を LINES

(#define を使って、それぞれ 20 と 100 くらいにセットしておく）配列の名前を words とすると、

char words[LINES][LEN];

とすればできるのであった

質問： char words[LEN] [LINES]; とはどう違う？

ファイルからデータを読み込み、配列に記憶する

もはやこれはできるはず
ファイル名を terms.txt
配列の名前を words
文字列の最大長さを 20
ファイルの行数の最大を 100
として、「ファイルから読み込んで配列wordsにセットし、それを画面に表示させ」てみよう

qsortStringTemplate1.c

文字列を対象としたクイックソート

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,j) と quickSort(array, j+1,*l*) を行って終了

これは「数」を対象としたソート
どこをどのように直すか？

```
void quickSort(int f, int l) {
  int m;
  int i ,j, temp;
  if (f >= l)  return;
  m = array[(f+l)/2];  i=f; j=l;
  while (1) {
      while ((array[i] < m) && (i < l)) i++;
      while ((array[j] > m) && (j > f)) j--;
      if (i >= j)  return;
      temp = array[i]; array[i] = array[j]; array[j]=temp;
      i++; j--;

   }
  quickSort(ar, f, j);
  quickSort(ar, j+1, l);
}
```

(1)文字列の配列は大域変数arrayとする

(2)文字列、又はポインタ

(3)文字列の比較

文字列対象のクイックソート

変更箇所

```
void quickSort( int ar[], int f, int l) {
   int m;
   int i ,j, temp;
   if (f >= l)  return;
    m = ar(f+l)/2];
     i=f; j=l;
     while (1) {
         while ((ar[i] < m) && (i < l)) i++;
         while ((ar[j] > m) && (j > f)) j--;
         if (i >= j) return;
         temp = ar[i]; ar[i] = ar[j]; ar[j]=temp;
          i++; j--;

     }
   quickSort(ar, f, j);
   quickSort(ar, j+1, l);
}
```

大域変数
char words[LINES][LEN]

理由は後述

```
int  i, j;
char m[LEN], temp[LEN];
```

```
strcpy(m,words [(f+l)/2]);
```

```
strcmp(words[i], m) < 0
```

```
strcmp(words[j], m) > 0
```

```
strcpy(temp,words [i]);
strcpy(words[i],words[j]);
strcpy(words[j],temp);
```

プログラムの完成とテスト

右のアルゴリズムをプログラムにしてみよう

そして terms.txt を対象として、ソートした結果を表示させてみよう

1. ファイルからデータを読み込み、配列に記憶する

2. クイックソートの手法により配列の要素を並び替える
3. 並び替えた配列の要素をモニターに表示する、もしくはファイルに書き出す

文字列を対象とするクイックソート解説

(1) void quickSort(int f, int l)

　関数に配列を引数として渡すか、大域変数とする（引数としない）か、という選択があった

ここで「関数の呼び出し」について解説：

　関数の引数として渡されるデータに対し「関数呼び出しスタック」上に記憶領域が作られる。普通の変数なら（一次元配列も含む）一個の場所ですんでいた。

しかし、words(仮引数array)は２次元配列なので、多くの場所を確保しなければならない。そのため配列全体の大きさが必要であった

⇒　この面倒を避けるには、２次元配列を関数の引数として渡すのではなく、大域(global)変数とするという方法をとった

文字列を対象とするクイックソート解説

strcpy(m,words[(f+l)/2]);

strcpy(x,y) : 文字列yの内容を文字列xにコピー

ここでは、words[(f+l)/2]の値（文字列）を文字列mに記憶

strcpy(temp,words[i]);

strcpy(words[i],words[j]);

strcpy(words[j],temp);

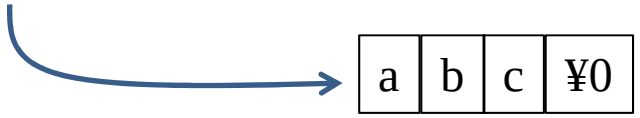
words[i]とwords[j]の内容を取り替える(swap)するために、

これらが必要

文字配列について

1 次元文字配列の場合： 例 char str[ ] = “abc”;

str に割り当てられた場所に次のデータが記憶される

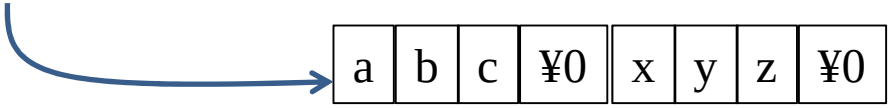


変数strの場所から文字列データが書き込まれる
だから *str = ‘a’, *(str+1)=‘b’, ... となる

参考: *変数 とは
変数が指す場所の値
を取り出す

2次元文字配列の場合： 例 char str[ ] [4]= {“abc”, “xyz”, ... }

str に割り当てられた場所に次のデータが記憶される



だから *str = “abc”, *(str+1)=“xyz”, ... となる

文字配列について

検証プログラム

(1) charTest.c

```
#include <stdio.h>
int main(void) {
  char str[ ] = "abc";
  printf("%x %s¥n", str,str);
  printf("%c-%c-%c-%d¥n",
    *str,*(str+1),*(str+2),*(str+3));
  return 0;
}
```

実行結果
18ff50 abc
a-b-c-0

(2) stringTest.c

```
#include <stdio.h>
int main(void) {
  char str[ ][4] = {"abc","xyz","uv"};
  printf("%x %x %x¥n", str[0],str[1],str[2]);
  printf("%s-%s-%s¥n",
        *str,*(str+1),*(str+2));
  return 0;
}
```

実行結果
18ff48 18ff4c 18ff50
abc-xyz-uv

文字列を対象とするクイックソート

```
void quickSort( int ar[], int f, int l) {
  int m;
  int i ,j, temp;
  if (f >= l)  return;
   m = ar(f+l)/2];
    i=f; j=l;
    while (1) {
       while ((ar[i] < m) && (i < l)) i++;
       while ((ar[j] > m) && (j > f)) j--;
       if (i >= j) return;
       temp = ar[i]; ar[i] = ar[j]; ar[j]=temp;
        i++; j--;

   }
  quickSort(ar, f, j);
  quickSort(ar, j+1, l);
}
```

以上の説明を元に

整数配列 ar に対して作った
クイックソート関数を元にして
文字列の配列を対象とした
クイックソート関数を作ってみよう。

```
int i, j;
char m[LEN], temp[LEN];
strcpy(m,words[(f+l)/2]);
strcmp(words[i], m) < 0
```

ただし、整数配列ar の代わりに、
大域変数の words が使われること
に
注意

```
strcpy(temp,words[i]);
strcpy(words[i],array[j]);
strcpy(words[j],temp);
```

ファイル terms.txt　のソート

qsortStringTemplate2.c

qsortStringTemplate1.c の内容にquickSort関数を修正して付け加える

別解（少し難度が高い...）

文字列の置き換えで、いちいち文字列をコピーするのは無駄である。次のようにするともっと効率が良いはず（要解説）

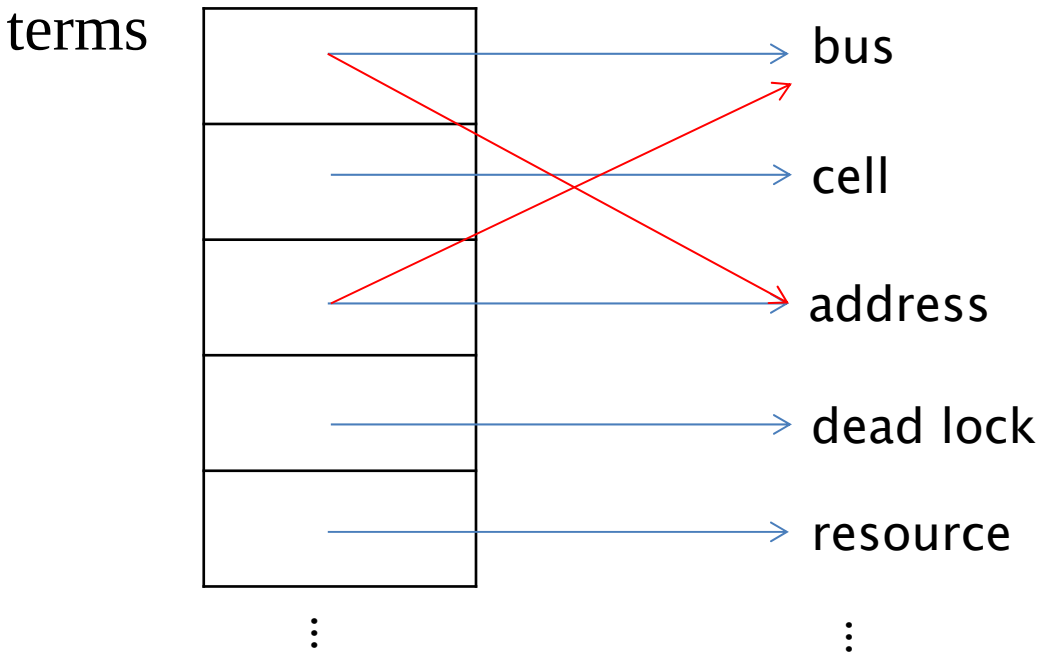
(1) 読み込んだ文字列に対し、それぞれ記憶領域を確保し
その記憶領域へのポインタからなる配列(terms)を作る

```
while (fscanf(fp,“%s”, line) != EOF) {  // 一行読み込み
        len = strlen(line);             // char *p;  として宣言したもの
        p = malloc(len+1);              // サイズ len+1の記憶領域を作る
        strcpy(p,line);                 //  文字列(line)をそこにコピー
        terms[n++] = p;                 // termsにはそのポインタ(pの値)を記憶
}
```

(2)ソートにおいては、その文字列のポインタを用いて比較、置き換えを行う (temp は　char* 型の変数）

```
temp = array [i]; array [i] = array [j]; array [j]=temp;
```

文字列ポインタの配列



配列 terms の要素は
それぞれの文字列への
ポインタ（記憶領域の
アドレス）

ポインタと指し示すものの
関係を→で表す

ここで bus へのポインタとaddressへのポインタをswapすると

別解プログラム(主要部)

```
void quickSort(char *array [], int f, int l) {
  char *m, *temp;
  int i = f, j = l;
  if (f < l) {
    m = array [(f+l)/2];
    while (1) {
      while ((strcmp(array [i], m) < 0) && (i < l)) i++;
      while ((strcmp(array [j], m) > 0) && (j > f)) j--;
      if (i >= j) return;
      temp = array[i]; array[i] = array[j]; array[j]=temp;
      i++; j--;
    }
    quickSort(array, f, j);
    quickSort(array, j+1, l);
  }
}
```

```
int main(void) {
  char *p,*terms[NUM];
  char line[LEN];
  int i, len, n=0;
  FILE *fp;
  fp = fopen(InFile,"r");

  while (fscanf(fp,"%s", line) != EOF) {
        len = strlen(line);
        p = malloc(len+1);
        strcpy(p,line);
        terms[n++] = p;
  }

  printf("%d lines are read¥n",n);

  quickSort(terms, 0, n-1);
  for(i=0; i < n; i++)
        printf("%s¥n",terms[i]);

  return 0;
}
```

実際に動かしてみましょう

qsortString3.c