プログラミング基礎演習
第6回
C言語編
【ファイル入出力】

長谷川禎彦

前回課題2

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void eig(double[2][2], double*, double*, double*, double*);
int main() {
  double val1;
  double val2;
  double vec1[2];
  double vec2[2];
  double mat[2][2] = {{1,5},{6,2}};
  eig(mat, &val1, &val2, vec1, vec2);
  printf("[%f, %f]\n", mat[0][0], mat[0][1]);
  printf("[%f, %f]\n", mat[1][0], mat[1][1]);
  printf("eigenvalue = %f, eigenvector = (%f, %f)\n", val1, vec1[0], vec1[1]);
  printf("eigenvalue = %f, eigenvector = (%f, %f)\n", val2, vec2[0], vec2[1]);

  return 0;
}
```

前回課題2

```
void eig(double mat[2][2], double *val1, double *val2, double *vec1, double *vec2) {
  double a = mat[0][0];
  double b = mat[0][1];
  double c = mat[1][0];
  double d = mat[1][1];

  double v = a*a - 2*a*d + 4*b*c + d*d;
  if (v >= 0) {
      *val1 = 0.5 * (-sqrt(v) + a + d);
      *val2 = 0.5 * (sqrt(v) + a + d);
      vec1[0] = -(sqrt(v) - a + d) / (2 * c);
      vec1[1] = 1;
      vec2[0] = -(-sqrt(v) - a + d) / (2 * c);
      vec2[1] = 1;
  }
}
```

前回課題3

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void reverse(char*);
int main() {
  char s[] = "hello world";
  reverse(s);
  printf("%s\n", s);
  return 0;
}
void reverse(char *s) {
  int slen = strlen(s);
  char *t = (char*) malloc(sizeof(char) * (slen + 1));
  strcpy(t, s);
  int i;
  for (i = 0;i < slen;i++) {
    s[i] = t[slen - i - 1];
  }
  free(t);
}
```

strlenは'\0'分を含まないので，コピーするときは+1が必要

前回課題4

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void uppercase(char*);
int main() {
  char s[] = "hello world";
  uppercase(s);
  printf("%s\n", s);
  return 0;
}
void uppercase(char *s) {
  int slen = strlen(s);
  char *t = (char*) malloc(sizeof(char) * (slen + 1));
  strcpy(t, s);
  int i;
  for (i = 0;i < slen;i++) {
    if('a' <= t[i] && t[i] <= 'z') {
       s[i] = t[i] + ('A' - 'a');
    }
  }
  free(t);
}
```

前回課題5

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

int char2val(char);
int roman2num(char*);
void replace2abc(char*);

int main() {
  printf("XLIII = %d\n",roman2num("XLIII"));
  printf("CDXCV = %d\n",roman2num("CDXCV"));
  printf("MDCCCLXXXVIII = %d\n",roman2num("MDCCCLXXXVIII"));
  printf("MCMXLV = %d\n",roman2num("MCMXLV"));
  printf("MMMCMXCIX = %d\n",roman2num("MMMCMXCIX"));

  return 0;
}
```

```
/*   a, b, c,...に従って値を返す. */
int char2val(char v) {
    int t = v - 'a';
    if (t % 2 == 0) {
       return (int) pow(10, t / 2);
    } else {
       return 5 * (int) pow(10, (t - 1) / 2);
    }
}
/*  I, V, X,...をa, b, c,...に置換する */
void replace2abc(char *roman) {
  int slen = strlen(roman);
  int i;
  for(i = 0;i < slen;i++) {
      switch(roman[i]) {
      case 'I': roman[i] = 'a'; break;
      case 'V': roman[i] = 'b'; break;
      case 'X': roman[i] = 'c'; break;
      case 'L': roman[i] = 'd'; break;
      case 'C': roman[i] = 'e'; break;
      case 'D': roman[i] = 'f'; break;
      case 'M': roman[i] = 'g'; break;
      }
  }
}
```

```c
int roman2num(char *roman_str) {
  int slen = strlen(roman_str);
  char *roman = (char*) malloc(sizeof(char) * (slen + 1)); // '\0'のためにslenより一つ大きいサイズが必要
  int i;
  strcpy(roman, roman_str);
   // I, V, X,...をa, b, c,...に置換する
  // こうすることで，値の上下関係が分かりやすい
  replace2abc(roman);
  int *coef = (int*) malloc(sizeof(int) * slen); // 記号が足すか引くかを表す値を格納する
  int *val = (int*) malloc(sizeof(int) * slen); // a, b, cに対応する値を格納する
  for (i = 0; i < slen; i++) {
       if (i == 0) {
       coef[i] = 1;
       val[i] = char2val(roman[i]);
       } else if (roman[i] <= roman[i - 1]) {
       coef[i] = 1;
       val[i] = char2val(roman[i]);
       } else {
       coef[i] = 1;
       coef[i - 1] = -1;
       val[i] = char2val(roman[i]);
       }
  }
  int r = 0;
  for (i = 0; i < slen; i++) {
       r += coef[i] * val[i];
  }
  free(coef);
  free(val);
  return r;
}
```

【補足】アドレスを求める&演算子

- &演算子を使うと、変数のアドレスを求めることが可能

```
int a = 1000;
printf("%d\n", a);
printf("%p\n", &a);
```

aが格納されているメモリのアドレスを表示する

【補足】アドレスを求める演算子&

```
// & を使う
int main() {
  double mean, variance;
  stats(N, data, &mean, &variance);
}
//ポインタ変数を使う
int main() {
  double *mean, *variance;
  mean = (double*) malloc(sizeof(double));
  variance = (double*) malloc(sizeof(double));
  stats(N, data, mean, variance);
}
void stats(int N, double *data, double *mean, double *variance)
{  …. }
```

【補足】
ポインタと一次元配列の違い

	malloc	配列
メモリ領域	ヒープ	スタック
メモリ管理	freeで解放	関数を抜けると解放

- mallocで確保した場合と，配列で確保した場合はメモリの領域が違う
  - スタック領域のサイズは固定（物理的なメモリ容量の限界値より遙かに小さい）
  - ヒープ領域のサイズは可変（物理的なメモリ容量が限界値）

【補足】
ポインタと一次元配列の違い

```
int main() {
  int a[1000000];
  return 0;
}
```

Crash

スタックサイズは限られているので，
大きな配列を確保すると落ちる！
これは高々4MBで落ちている

配列版

```
int main() {
  int* a;
  a = (int*)malloc(sizeof(int)*1000000);
  return 0;
}
```

ポインタ（malloc）版

ヒープサイズは可変なので，大きな
サイズをとっても大丈夫！

【補足】
ポインタと一次元配列の違い

```c
int main() {
  int* a;
  a = f();
  printf("%d",*a);
}
int* f() {
  int* a;
  a=(int*)malloc(sizeof(int));
  a[0]=1;
  return a;
}
```

freeしない限り，mallocした変数領域は開放されないので，aにアクセス可能

ポインタ(malloc)版

```c
int main() {
  int* a;
  a = f();
  printf("%d",*a);
}
int* f() {
  int a[1];
  a[0]=1;
  return a;
}
```

関数f()を出ると配列は解放されるため，ここでaにアクセスすることは出来る保証はない

配列版

【補足】文字列

```c
int main() {
  char a[]="hello";
  a[0]='H';
  printf("%s\n",a);
  return 0;
}
```

こちらは正しく動く！

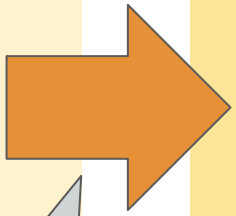
```c
int main() {
  char *a="hello";
  a[0]='H';
  printf("%s\n",a);
  return 0;
}
```

Crash

文字列をこう書くと，書き換え禁止領域に確保される．aにはその禁止領域のアドレスが格納されている．

【補足】文字列

```c
int main() {
  char a[]="hello";
  a[0]='H';
  printf("%s\n",a);
  return 0;
}
```



```c
int main() {
    char a[6];
    a[0]='h';
    a[1]='e';
    a[2]='l';
    a[3]='l';
    a[4]='o';
    a[5]='\0';
    a[0]='H';
    printf("%s\n",a);
    return 0;
}
```

コンパイルするときにこのように解釈されているので，配列を用いた例では文字列は書き換え禁止領域ではない！

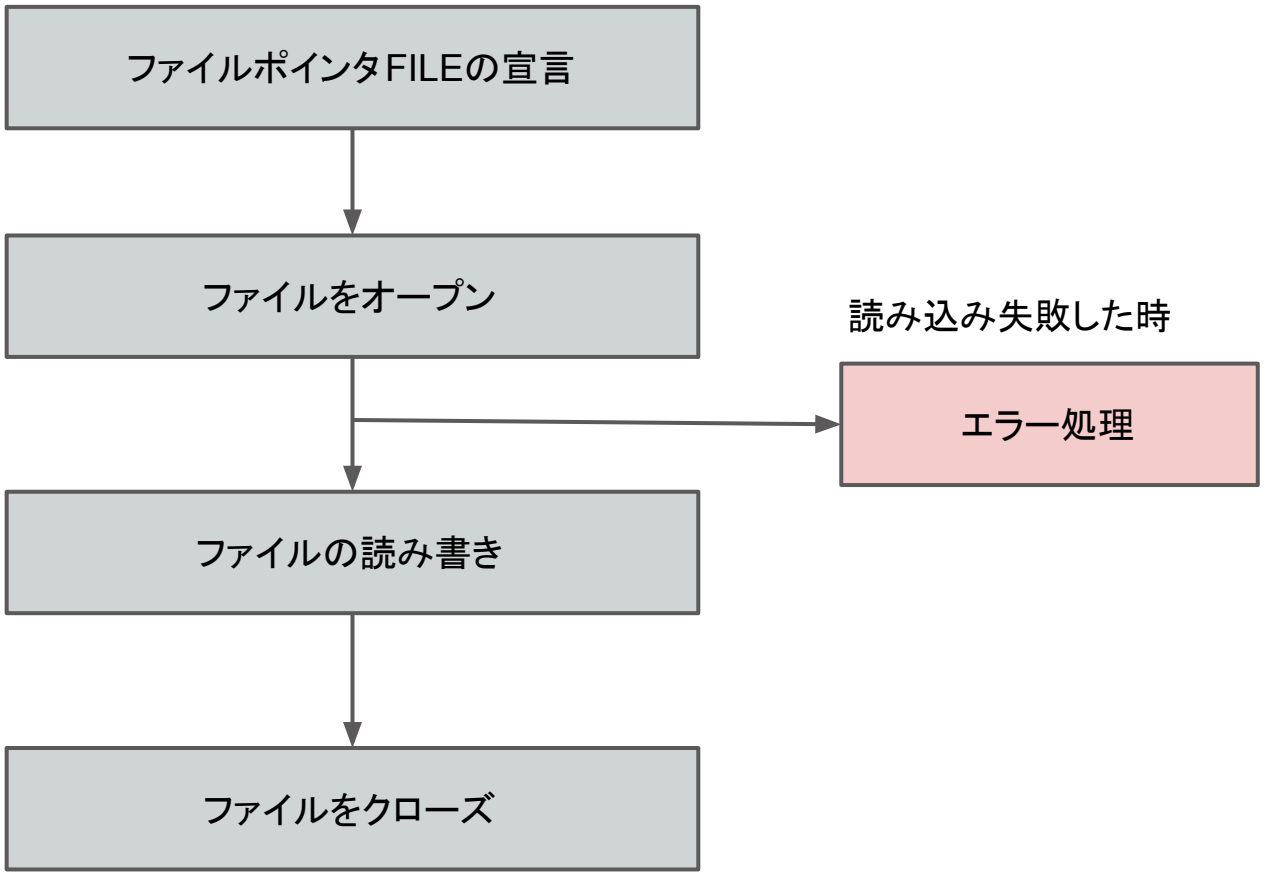
今日の内容

- ファイル入出力
  - 外部のテキストデータなどを読み込める
  - 計算結果を外部のテキストファイルに保存できる

ファイルへの読み書き

- ファイルを開く
  - `fopen`
- ファイルへ書く
  - `fputs, fprintf, putc`
- ファイルから読み込む
  - `fgets, getc, fscanf`
- ファイルを閉める
  - `fclose`

ファイルへの読み書き


ファイルポインタFILEの宣言
ファイルをオープン
読み込み失敗した時
エラー処理
ファイルの読み書き
ファイルをクローズ

ファイルを開く

FILE * fopen(char * filename, char *mode)

例：**FILE * fp = fopen(“data.txt”, “r”)**

- modeの種類
  - “r” = read 読み込むために開く
  - “w” = write 書き込むために開く
  - “a” = append 追記するために開く
- 既に存在するファイルを”w”で開くとその中身が削除される. 既存の内容に追加したければ”a”で開く.
- パスが存在しない，ファイル権限がない，ディスクが一杯などでファイルを開く操作が失敗したら fpがNULLとなる

ファイルから入出力

- 文字単位での入出力
  - `int getc(FILE *fp)`
  - `int putc(int c, FILE *fp)`
- 行(文字列)単位での入出力
  - `int fscanf(FILE *fp, char * format)`
  - `int fprintf(FILE *fp, char *format)`
  - `int fgets(char buf*, int n, FILE *fp)`

補足:fgets()

```
char *fgets(char *s, int n, FILE
*fp);
```

- fpで指定したファイルから，最大n文字分1行読み込み文字列sに格納する.
  - 最大文字数には '\0' も含まれる．実際に読み込める文字数は n-1
- 入力が終了したら NULL を返す

補足:fputs()

```
int fputs(const char *s, FILE *fp);
```

- ファイルポインタ(fp)で指定したファイルへ文字列(s)を書き込む
- 返り値
  - 書き込み成功時: 正の値
  - 書き込み失敗時: EOF

補足:getc()

```
int getc(FILE *fp);
```

- ファイルポインタ(fp)で指定したファイルから1文字読み込む
  - getchar()に似ている
- 入力が終了したら EOF を返す.

補足:putc()

```
int putc(int c, FILE *fp);
```

- ファイルポインタ(fp)で指定したファイルへ1文字(c)書き込む
  - putchar()に似ている
- 返り値
  - 書き込み成功時: 書き込んだ文字
  - 書き込み失敗時: EOF

補足:fprintf()

```
int fprintf(FILE *fp, const char *format, ...);
```

- ファイルポインタ(fp)で指定したファイルに書式つき(printfと同じ)で書き込む
  - printf()に似ている
- 返り値
  - 成功した場合: 書き出された文字数
  - 失敗した場合: 負の値

補足:fscanf()

```
int fscanf(FILE *fp, const char *format, ...);
```

- ファイルポインタ(fp)で指定したファイルから書式つきで読み込む
  - scanfに似ている
- 返り値
  - 成功した場合: 代入された入力項目の個数
  - 失敗した場合: EOF

ファイル読み込みの例:fgets

```c
#include <stdio.h>
#define MAX_LEN 100
int main() {
  char buffer[MAX_LEN];
  FILE *fp;
  fp = fopen("test.txt", "r");
  //check whether we opened the file correctly.
  if (fp == NULL){
      printf("Failed to open the file\n");
      return 1;
  }
  //read one line at a time and display it.
  while(fgets(buffer, MAX_LEN, fp) != NULL){
      printf("%s", buffer);
  }
  //close the file.
  fclose(fp);
  return 0;
}
```

ファイル読み込みの例:getc

```c
#include <stdio.h>
int main(){
  FILE *fp;
  fp = fopen("test.txt", "r");
  if (fp == NULL){
      printf("Failed to open the file\n");
      return 1;
  }
  int tmp;
  char c;
  //read one character at a time and display it.
  while((tmp = getc(fp)) != EOF){
      c = (char) tmp;
      printf("%c", c);
  }
  //close the file.
  fclose(fp);
  return 0;
}
```

ファイル出力の例：fprintf

```c
#include <stdio.h>
int main(){
  char *name = "Hello world";
  int age = 25;
  FILE *fp = fopen("users.txt", "a");
  if (fp == NULL){
     printf("File open failed\n");
     return 1;
  }
  // write the data.
  if (0 > fprintf(fp, "name = %s, age = %d\n", name, age)){
     printf("Failed writing to file\n");
  }
  fclose(fp);
  return 0;
}
```

課題1(必須課題)

namelist.txtの名前は乱数文字なので，文字化けではない

コマンドライン引数から読み込むファイル（namelist.txt）を指定して読み込み，各項目の平均，最大値，最小値，標準偏差を計算し，表示させよ．データは

名前，年齢，身長，体重

の順に並んでいる．

出力例

```
Age : (max,min,mean,stdev) = (XXX, XXX, XXX, XXX)
Height : (max,min,mean,stdev) = (XXX, XXX, XXX, XXX)
Weight : (max,min,mean,stdev) = (XXX, XXX, XXX, XXX)
```

課題1(必須課題)

fscanfを使う場合

```
fscanf(fp,"%s,%d,%lf,%lf\n",.....)
```

ではカンマを含めて文字列として読み込んでしまう.

```
fscanf(fp,"%[^,],%d,%lf,%lf\n",.....);
```

とする必要がある.

getcやfgetsを使うのであれば，カンマを分離させるようなコードを書く.

課題2（自由課題）

sentence.txtにおいて，文頭を大文字にし，一人称の「i」も大文字にするプログラムを書け．さらに，変換した結果を他のテキストファイルに書きだせ．なお，読み込むファイル名，書き出すファイル名はコマンドライン引数から指定するようにせよ．

課題3(自由課題)

(実質レポートの一部なので来週解説なし)

- タブと改行で区切られた数値データを読み込むプログラムを書け(レポートの一部になる). コマンドライン引数からファイル名を指定せよ.
- 読み込むファイルの行数と列数はあらかじめ分かっていないとする.
- 数値の型はint型とせよ. 実際にホームページにある int_matrix_file.txt を読んでみよ.

課題3(自由課題)

```
> ./kadai03 int_matrix_file.txt
54 73 61 37 12 13 0 65…
….
```

のように表示させてみよ.

参考情報

- 行列のファイルサイズが分からない場合，一度目の読み込みで行数を知りたい場合がある．この場合，データを読むに当たりFILEポインタを再び最初に戻す必要がある．
- **`fseek(fp,0L,SEEK_SET)`**
  - FILEポインタ**`fp`**を最初に戻す

課題提出方法

- 課題xの答えをkadai0x.cファイルに書く.
  - xは1,2,3,...
- 締め切りは日曜日の23:59
- ファイルをzipファイルにまとめる
  - ファイル名：学籍番号.zip
  - 例：学籍番号が641234の場合，641234.zipとする.
  - 圧縮方法はホームページに記載してある
- 学籍番号.zipファイルのみを提出する
  - 提出は https://goo.gl/QJ3LMP