プログラミング入門 II

ファイル入出力（2）

ストリームという考え方

- ストリーム(stream) = データの列
  - キーボードから打つ文字列
  - 画面に出力される文字列
  - ファイル

ファイルのストリーム

- ファイルのオープン
  - 操作を始める前に、ストリームとファイルを結びつける
- ファイルの操作
  - 読み／書き
- ファイルのクローズ
  - 操作が終わった後に、ストリームを切り離す

ファイルのオープン

- ストリームは、ファイルポインタ（FILE *)であらわされている。
- fopen関数でファイルをオープンし、ストリームを返す

```
FILE *fp;
      …
fp = fopen(“test.txt”,”w”);
```

fopen(ファイル名,オープンモード)

- ファイル名：オープンしたいファイル名
- オープンモード：読みこみ”r”, 書き込み”w”

fopenのモード

- fopen のモード（下の表）
- バイナリモード（"wb"、"rb" のように指定）

モード	動作	ファイルがあるとき	ファイルがないとき
"r"	読み出し専用	正常	エラー（NULL返却）
"w"	書き込み専用	サイズを 0 にする（上書き）	新規作成
"a"	追加書き込み専用	最後に追加する	新規作成
"r+"	読み込みと書き込み	正常	エラー（NULL返却）
"w+"	書き込みと読み込み	サイズを 0 にする（上書き）	新規作成
"a+"	読み込みと追加書き込み	最後に追加する	新規作成

ファイルのクローズ

- ストリームをファイルから切り離す

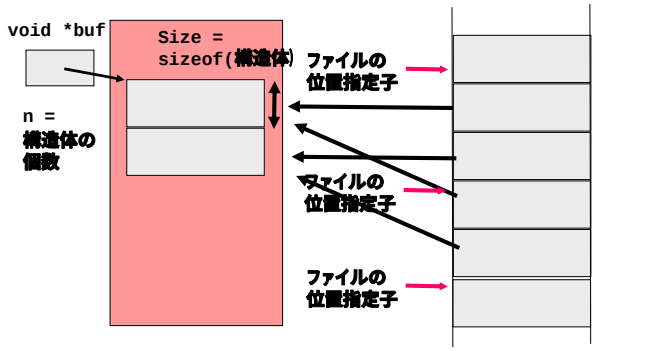
fclose(ストリーム)

- ストリームは、fopenで得たファイルポインタ
- 成功した場合は１、失敗した場合は0を返す
- 書き込んだ内容は、fcloseしないと全部かきこまれないので注意。

fread関数

- ファイルfpからsizeバイトのデータをn個読み込み、bufに格納します。
- 返り値は、読み込んだデータの個数、EOFの時は0
  - したがって、EOFかエラーかは、feofとferrorで判定する。
- ファイル位置指示子を読み込んだデータバイト分進めます。
  - エラーが発生した場合にはファイル位置指示子の値は不定です。

```
#include <stdio.h>
 size_t fread(void *buf, size_t size,
  size_t n, FILE *fp);
```

関数fwrite

- bufからファイルfpへsizeバイトのデータをn個書き込みます。
- エラーはfreadと同じ
- ファイル位置指示子を書き込んだデータバイト分進めます。
  - エラーが発生した場合にはファイル位置指示子の値は不定です。

```
#include <stdio.h>
  size_t fwrite(const void *buf, size_t
  size, size_t n, FILE *fp);
```


feek、ftell

- *stream* によって指定されたストリームにおいて、ファイル位置表示子 *(file position indicator)* をセットする
- *whence* 引数が SEEK_SET, SEEK_END, SEEK_CUR

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
long ftell(FILE *stream);
```

- ftell
  - *stream* によって指定されたストリームにおける、ファイル位置表示子の現時点での値を与える

fflush

- ユーザー空間でバッファリングされているすべてのデータを与えられた出力に書き出す (フラッシュする)
- あるいはストリーム *stream* の下位にある書き込み関数を用いてこのストリームを更新する。ストリームは開いた状態のままであり、この関数によって何の影響も受けない。

```
#include <stdio.h>
int fflush(FILE *stream);
```


プログラム例

- プログラム１：データファイル（テキスト、以前の例と同じ）を読んで、バイナリデータファイルを作る
- プログラム２：バイナリデータファイルから、データを読んで検索する

プログラム1

テキストファイルからバイナリファイルを作る

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

struct record {
  char name[10];
  int point;
};

int main(int argc, char *argv[])
{
  FILE *fp;
  FILE *fq;
  int x,r;
  char name[10], buf[256];
  struct record d;

  if (argc != 2) {
    printf("missing file argument¥n");
    return 1;
```

プログラム1

テキストファイルからバイナリファイルを作る

```
  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    printf("can't open %s¥n", argv[1]);
    return 1;
  }
  fq = fopen("data-file","w");
  if (fq == NULL) {
    printf("can't open data-file¥n");
    return 1;
  }
  while (fgets(buf, sizeof(buf), fp) != NULL){
    sscanf(buf, "%s %d", name, &x);
    d.point = x;
    strcpy(d.name,name);
    r = fwrite(&d,sizeof(d),1,fq);
    if(r != 1){
      printf("write error¥n");
      exit(1);
    }
    printf("name=%s, point=%d stored¥n",d.name,d.point);
  }
```

プログラム1

テキストファイルからバイナリファイルを作る

```
while (fgets(buf, sizeof(buf), fp) != N
    sscanf(buf, "%s %d", name, &x);
    d.point = x;
    strcpy(d.name,name);
    r = fwrite(&d,sizeof(d),1,fq);
    if(r != 1){
      printf("write error¥n");
      exit(1);
    }
    printf("name=%s, point=%d stored¥n",d.name,d.point);
  }
  fclose(fp);
  fclose(fq);

  return 0;
}
```

プログラム2

バイナリファイルから検索

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

struct record {
  char name[10];
  int point;
};

int main(int argc, char *argv[])
{
  FILE *fp;
  int r,x;
  char *name;
  struct record d;

  if (argc != 2) {
    printf("missing file argument¥n");
    return 1;
  }
  name = argv[1];
```

プログラム2

バイナリファイルから検索

```
  fp = fopen("data-file","r");
  if (fp == NULL) {
     printf("can't open data-file¥n");
     return 1;
  }

  x = -1;
  while(fread(&d,sizeof(d),1,fp) == 1){
    if(strcmp(d.name,name) == 0){
      x = d.point;
      break;
    }
  }
  if(x >= 0)
    printf("name '%s', point = %d¥n",name,d.point);
  else
    printf("name '%s', data is not found¥n",name);

  fclose(fp);

  return 0;
}
```


考えてみましょう

- このプログラムでは、１個1個データを読み込んでいますが、数十個ずずデータを読み込んだほうが効率がいいです。どうすればいいでしょうか？