

# 情報ネットワーク応用 課題 1 レポート

1280391 情報学群 2 年  
細川 夏風

2026 年 1 月 26 日

## 1 実行例

本プログラムの実行例をいかに示す。

```
1   % cat test.txt
2   abcdefg
3   % ./a.out test.txt
4   count:0 index:0 | 00 00 00 00 00 |
5   count:5 index:1 | 61 62 63 64 65 |
6   c:61
7   count:5 index:2 | 61 62 63 64 65 |
8   c:62
9   count:5 index:3 | 61 62 63 64 65 |
10  c:63
11  count:5 index:4 | 61 62 63 64 65 |
12  c:64
13  count:5 index:5 | 61 62 63 64 65 |
14  c:65
15  count:3 index:1 | 66 67 0a 64 65 |
16  c:66
17  count:3 index:2 | 66 67 0a 64 65 |
18  c:67
19  count:3 index:3 | 66 67 0a 64 65 |
20  c:0a
21  count:0 index:0 | 66 67 0a 64 65 |
```

## 2 プログラムの説明

本プログラムの主要部分の説明をいかに示す。

### 2.1 構造体の定義

```
1 // ファイル構造体FILEに相当する構造体
2 // 本物のライブラリーで用意されている FILE を使ってはいけない。
3 struct my_file {
4     int fd;                      /* ファイルディスクリプター(何個ファイル
5                                をオープンしたか, 更にそれらを識別するためのもの) */
6     int count;                   /* ファイルから読んだ文字数 */
7     int index;                   /* 次に使う位置 */
8     char buffer[MyBufferSize];   /* バッファー */
9 }
```

以上のように構造体を定義している。構造体 `my_file` はファイルディスクリプターと呼ばれる。`open` したファイルを区別するための変数とファイルから読んだ文字数、次の位置の記憶のために使われる `index`、読み込んだ文字を格納する配列 `buffer` を持つ。この時、`MyBufferSize` はすでに定義されている定数である。

### 2.2 オープンファイルを表す構造体

```
1 struct my_file *my_fopen (char *filename) {
2     int fd;
3     fd = open (filename, O_RDONLY);
4     if (fd != -1) { // openの失敗は-1
5         struct my_file *fp;
6         fp = (struct my_file *) malloc (sizeof (struct my_file));
7         // 構造体内の変数の初期化
8         fp->fd = fd;
9         fp->count = 0;
10        fp->index = 0;
11        return fp;
12    } else {
13        return NULL;           /* オープンできなかった場合 */
14    }
15 }
```

---

オープンしたファイルから、前述の構造体の初期化を行う。fpはその開いたファイルの情報を持つ構造体のポインターを表す。fdはそのファイル自体のファイルディスクリプターとなっている。初期化のため構造体変数に対して初期値の0を代入している。

### 2.3 ファイルのクローズ処理

```
1 int my_fclose (struct my_file *fp) {
2     int r;
3     // OSに対してクローズ処理を要求
4     // ファイル構造体用のメモリー領域を解放
5     // システムコールを実行(rにclose()の値を入れることによって成功の成
6     // 否を判断する材料にする)
7     r = close(fp->fd);
8     // メモリーの解放 (free(fp)するだけで値にNULLを入れなくてもヒープ領
9     // 域に戻ってくれる)
10    free(fp);
11    return r;
12 }
```

ディスクリプタからオープンしているファイルを選んで閉じる。その後、確保されていたメモリ領域を解放する。freeを実行するだけで構造体変数にNULLを代入したくても自動的にヒープ領域に戻る。また、rにclose(fp->fd);を代入することによってclose()の結果をrに格納できる。

### 2.4 fgetc関数の再現

```
1 int my_fgetc(struct my_file *fp) {
2     int c, size;
3     // バッファーが空ならOSから read する
4     // バッファーから1文字取って返す
5     if (fp->index == fp->count) {
6         size = read(fp->fd, fp->buffer, MyBufferSize);
7         if (size <= 0) {
8             return EOF;
9         }
10        fp->count = size;
11        fp->index = 0;
12    }
```

```
13     c = (unsigned char)fp->buffer[fp->index++];
14     return c;
15 }
```

`read()` によって、指定したファイルディスクリプターから定数指定分の値を読み込む。読み込むファイルがなくなった場合、ファイルの終端である `EOF` を返す。次に `count` には読み込んだサイズを入れ、`index` を初期化する。最後にバッファーの一文字を返す。

### 3 考察

どのようにしてファイルを読む際に次に読むべき場所を把握しているかを理解していなかったため、少し調べてみた。これにはファイルオフセットが使われている。これはしおりのような働きがあり、ファイルを読んだ際次の読み込む場所に目印をつけておく。これにより次に読むべき場所を把握している。

### 4 感想

私は以前、興味がありシステムコールの関数自体を AI の協力のもと作成したことがある。内部の殆どがアセンブリ言語で作成されており、意味不明な定数を指定するなどしてパケットを作成した。これにより現在の C 言語がいかに楽にコーディング可能かを思い知った。しかし、C 言語の境界値を判断しないという仕様は多くの開発者を苦しめている。この解決のために他の多くの言語が代替に名乗りを上げている状態だが C 言語で書かれたプロダクトはすでに多くのプログラムコードが高く積まれている。そのため、それを他の言語で置き換えるのにも多大なるコストを要する。ただ、この C 言語の仕様が長年開発者を保守という時間に勾留した呪いのようなものなのも確かなのである。私は早くこの事態が解決することを一刻も早く望んでいる。

### 参考文献

- [1] ファイルオフセットのための参考文献、ここではファイルオフセットではなくファイルポインタで説明している。 [https://qiita.com/Jim\\_Jin/items/c252cd2c8a0ba570620d](https://qiita.com/Jim_Jin/items/c252cd2c8a0ba570620d)