

# テキストエディタを作ろう (実装編：C)

Go Suzuki

## 目次

1. ファイルを読み書きしよう 楽	2
1.1. 作るモノ .....	2
1.2. 可変長配列 .....	3
1.2.1. 易 newString のヒント .....	3
1.2.2. 難 insertString のヒント .....	4
1.2.3. 易 removeString のヒント .....	4
1.3. 読み書きする関数を用意しよう .....	4
1.3.1. 易 openFile のヒント .....	4
1.3.2. 易 writeFile のヒント .....	4
1.4. 各処理を作っていこう .....	4
1.4.1. 易 printLines のヒント .....	5
1.4.2. 難 appendLines のヒント .....	5
1.4.3. 易 removeLines のヒント .....	5
1.5. コマンドを解析しよう .....	5
1.5.1. 難 ヒント .....	5
1.6. main を用意しよう .....	5

# ライセンス

この文書は CC-BY である。また、この文書により生じる一切の請求、損害、その他の義務について何らの責任も負わない。

## 1. ファイルを読み書きしよう 楽

では、ラインエディタと呼ばれるものを最初に作ってみよう。ラインエディタは ed が有名である。

### 1.1. 作るモノ

```
> p0,3
hoge
piyo
fuga
funya
> a1
puru
pura
.
> p0,5
hoge
piyo
puru
pura
fuga
funya
> r4,100000
> p0,1000
hoge
piyo
puru
> q (プログラム終了)
```

作るラインエディタは、引数に読み書きするファイル名を指定する。> を表示してコマンドを待機する。コマンドは次の通りである。

(1) q : プログラムを終了する

- (2) w : 変更を保存する (ファイルに書き出す)
- (3) p : 指定された範囲の行を表示する
- (4) r : 指定された範囲の行を削除する
- (5) a : 指定された行の後に入力された行を追加する ( . で入力終了. )

範囲指定は, `<start>,<end>` という書式で行う. `<start>` 以上 `<end>` 未満であることを表す. `<start>` とだけ渡された場合は, `<start>` 以上 `<start> + 1` 未満であると解釈する.

## 1.2. 可変長配列

テキストファイルのデータを保存するために, 可変長配列を用意しよう. そのため, 次の構造体を用意しよう.

```
struct buffer {
    char * buf; // buffer
    int len; // length
    int cap; // capacity
};
```

`buf` はバッファへのポインタ, `cap` は確保したバッファのサイズ, `len` は実際の文字列のサイズである.

そして, 次の関数を用意してみよう.

```
int newString(char * content, int size, struct buffer * result)
    // content の内容で size 分だけのバッファを確保し, result に格納する. 返回值は成功したか.
int insertString(struct buffer * me, char * content, int start, int size)
    // me に content (長さは size) を start の位置から挿入する. 返回值は成功したか.
void removeString(struct buffer * me, int start, int size)
    // me を start の位置から size 分だけ削除する,
```

実装する際は絵を描いて考えてみよう.

### 1.2.1. 易 newString のヒント

- `malloc` を用いてバッファを確保し, `memcpy` で内容をコピーしよう.

### 1.2.2. 難 insertString のヒント

- `me->cap < (me->len + size)` のときは、バッファを確保しなそう。
- まず、`start` の位置から最後まで文字を `size` だけ後に動かそう。
- 動かしたら、`start` の位置に `content` を書き込もう。

### 1.2.3. 易 removeString のヒント

- `start+size` から最後まで文字を `size` だけ前に動かさなきゃいけない。

## 1.3. 読み書きする関数を用意しよう

まずはファイルを読み書きする関数を用意してみよう。

```
int openFile(struct buffer * buf, char * path)
    // ファイル読み込みする。内容は buf に格納される。返り値は成功したか
int writeFile(struct buffer * buf, char * path)
    // ファイル書き込みする。返り値は成功したか
```

ファイル読み込みは `fopen` を `r` モードで開く。戻り値チェックなどは忘れずに行おう。  
(`openFile` 関数) そして、読み込んで、作った可変長配列に格納してみよう。

### 1.3.1. 易 openFile のヒント

- `newString` して可変長配列を作ろう。
- `gets` と `insertString` でどんどん読み込んでいこう。

### 1.3.2. 易 writeFile のヒント

- `fwrite` で一気にやろう。

## 1.4. 各処理を作っていこう

各コマンドに対応する処理を作っていこう。

```
// 全て start は始まりの行, end は終わりの行です！
void printLines(struct buffer * buf, int start, int end)
    // 内容を表示する。
int appendLines(struct buffer * buf, int start, int end)
    // 追加する。end は無視しよう。返り値は成功したか
void removeLines(struct buffer * buf, int start, int end)
```

```
// 削除する.
```

#### 1.4.1. 易 printLines のヒント

- `for` ループと `putc` で 1 行ずつ処理していこう.
- `\n` が来たら, 次の行に移った合図だ.

#### 1.4.2. 難 appendLines のヒント

- `printLines` と同様に行をカウントしていこう.
- 挿入は `insertString` を使おう.
- とりあえず 1 行あたり 1024 文字まで入力できる, と仮定しておこう. (可変長はコンソール API を使わなきゃいけない)

#### 1.4.3. 易 removeLines のヒント

- `printLines` と同様に行をカウントしていこう.

### 1.5. コマンドを解析しよう

コマンドの範囲選択のところを解析する関数を作ろう.

```
int parse(char * buf, int * start, int * end) // 返回值は成功したか
```

#### 1.5.1. 難 ヒント

- まず, どこからどこまでが数字かを判断して, しっかりとカンマがあるか, あるいは改行が来て終わるか判別しよう.
- 書式が正しいことを確認したら `sscanf` を用いて, 数を読み取ろう.
- 改行ならば終わり, カンマがあるならば, 次の数のところからまた `sscanf` を用いよう.

### 1.6. main を用意しよう

ここまでできた君ならば行けるはず!