プログラミング演習 第 13 回

疑似音楽プレイヤに曲順の整列機能をつけよう。部分演奏や曲順の編集に 配列 を使ったプレイリストを実装する。各曲データは配列の各要素として持ち、それを管理するため、図 1 に示す音楽プレイヤ全体を表す構造体 ListPlayer を用いるとよい。(なお、本プログラムは配列のコピーが頻繁に行われるため、効率はあまり良くないことを注意しておく。)

問 A [13A]: 整列音楽プレイヤ

曲名データを 端末 (標準入力) から 1 行ずつ受け取り、このとき曲名の辞書順に挿入ソートを行いながらプレイリストとして保存し、コマンドによって整列した順に印字するプログラムを作れ。

表示コマンドは、範囲を示す数値の組と『p』一文字とからなり、次の形式とする。

開始行番号,終了行番号 p

曲名データは 1 曲 1 行とし、データの終了は先頭が '.' (ピリオド) で始まる行によって示される。参考として図 1 以降に、プレイヤデータの初期化、追加を行う関数のプロトタイプ宣言と、入力部などのプログラムの断片を示す。

入出力例 (問 **A**)

```
入力
  dear my friend
  a song from japan
  cool hip hop
  boy meets girl
  fantastic sounds
  easy to dance
  .
  1,6p
  .
出力
  a song from japan
  boy meets girl
  cool hip hop
  dear my friend
  easy to dance
  fantastic sounds
```

入出力例 (問 **B**)

```
入力
  can you hear me?
  boy meets girl
  cool hip hop
  best  my friend
  .
  /cool
  .
出力
  cool hip hop

入力
  a song from japan
  boy meets girl
  cool hip hop
  best  my friend
  .
  /b
  .
出力
  best my friend          整列されている
  boy meets girl          ことに注意
```

問 B [13B]: 検索機能 / コマンドの実装

問 A で配列上に作成した音楽プレイリストのなかから、コマンドで示した文字列と一致する部分を出力 (演奏) する検索表示関数 `LP_search()` を作れ。関数 LP_search の雛形 (図 3) の中にある printf は動作確認用なので、プログラム提出時にはコメントアウトすること。

検索コマンドは『/』(スラッシュ) 一文字に検索パターン (曲名の先頭からの文字列) が続く、以下の形式とする。

/文字列

ヒント

それぞれの関数のプロトタイプとヒントは図 1、2、3 に含まれている。曲のプレイリストは整列しているので二分探索を用いると良く、部分文字列の比較には strncmp() 関数を用いてもよい。

問 C [13C]: ソート済みデータの読み込みとマージソート

問 B までで作成した音楽プレイヤに変更を加え、一つ目の'.' ピリオドまでの入力は挿入ソートを行い、その次の行から二つ目の'.' ピリオドまでで与えられる整列済みのデータを追加読み込みしながら マージソート をし、全体の整列結果を表示するプログラムを作れ。

入出力例 (問 C)

```
入力
  cool hip hop
  a song from japan
  fantastic sounds
  .
  boy meets girl        ここの三行は
  dear my friend        正しくソート
  easy to dance         済みとする
  .
  1,6p
  .
出力
  a song from japan
  boy meets girl
  cool hip hop
  dear my friend
  easy to dance
  fantastic sounds
```

入出力例 (問 D)

```
入力
  a song from japan
  boy meets girl
  cool hip hop
  .
  2,3p
  2,p
  s
  .
出力
  boy meets girl        2,3p の結果
  cool hip hop

  boy meets girl        2,p の結果

  2 boy meets girl      s コマンドの結果
  1 cool hip hop
  0 a song from japan
```

問 D [13D]: 再生回数での整列

問 B まで で作成したプログラムに、つぎの機能を追加する。

1. p コマンドによる再生において、再生回数の記憶機能 を加え、
2. s コマンドで再生回数により降順に整列して表示する機能を実装せよ。なお、再生回数が同じ場合は曲名の辞書順とすること。表示形式は「再生回数 曲名」とする。

ヒント

たとえば、

```
typedef struct{
  char * name;
  int playnum;
} elementtype;
```

のように elementtype を曲名と再生回数をあわせて保持する構造体にするとよい。
ただし、こうするとプログラム中では、

```
  lp.music[3].name = music;
  lp.music[3].playnum = 0;
```
のようにメンバが一段深くなることに注意すること。

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

#define LISTMAX 1000

typedef char * elementtype;

typedef struct {
  elementtype music[LISTMAX];
  int n;
} ListPlayer;

ListPlayer LP_init();
ListPlayer LP_addmusic(ListPlayer lp, elementtype music);
void LP_play(ListPlayer lp, int b, int e);
void LP_search(ListPlayer lp, char * buf);
void chomp(char *buf, int n);

main(){
  ListPlayer myplayer;
  char * music;
  int b, e, d, bi, ci;
  char buf[1024], cmdbuf[16], cmd;

  myplayer = LP_init();

  while(1){
    fgets(buf, sizeof(buf), stdin);
    chomp(buf, sizeof(buf));
    if(buf[0] == '.') break;
    music = strdup(buf);        /* keep a text line on memory */
    myplayer = LP_addmusic(myplayer, music);
  }
/* その２につづく */
```

図 1: 整列音楽プレイヤプログラムのヒント その 1

```
  /* command parser */
  while(1){
    b = e = d = 0;
    fgets(buf, sizeof(buf), stdin);
    bi=0;
    ci=0;
    while(isdigit(buf[bi])){
      cmdbuf[ci] = buf[bi];
      bi++; ci++;
    }
    cmdbuf[ci] = '\0';
    b = atol(cmdbuf);
    if(buf[bi] == ','){
      bi++;
      ci=0;
      while(isdigit(buf[bi])){
        cmdbuf[ci] = buf[bi];
        bi++; ci++;
      }
      cmdbuf[ci] = '\0';
      e = atol(cmdbuf);
    }
    cmd = buf[bi];

    if(cmd != '/'){
      bi++;
      ci=0;
      while (isdigit(buf[bi])){
        cmdbuf[ci] = buf[bi];
        bi++; ci++;
      }
      cmdbuf[ci] = '\0';
      d = atol(cmdbuf);
    }

    /* command execution */
    if (cmd == 'p'){
      LP_play(myplayer, b, e);
    }else if (cmd == '/'){
      chomp(&buf[bi+1], sizeof(buf)-bi-1);
      LP_search(myplayer, &buf[bi+1]);
    }else if (cmd == '.'){
      break;
    }
  }
}
```

図 2: 整列音楽プレイヤプログラムのヒント その 2 (1 の続き)

```
void LP_search(ListPlayer lp, char * buf){
  printf("LP_search : %s size %d\n", buf, (int)strlen(buf));
  /* 問 B の / 検索と表示をここで行う */
}

ListPlayer LP_addmusic(ListPlayer lp, elementtype music){
  int j;

  if(lp.n >= LISTMAX) return lp;
  /* 問 A の 挿入ソートとなるプログラムをここに書く */
  return lp;
}

void LP_play(ListPlayer lp, int b, int e){
  if(e<b) e = b;
  /* ListPlayer 内の「配列」から範囲を出力するプログラムをここに書く */
}

ListPlayer LP_init(){
  ListPlayer lp;
  lp.n = /* 適切な値を設定すること */
  lp.music[0] = strdup(""); /* 文字列の番兵 (最小の文字列) */
  return lp;
}

void chomp(char *buf, int n){
  int i;
  for(i=0; i<n; i++){
    if(buf[i] == '\n'){
      buf[i] = '\0';
      break;
    }
  }
}
```

図 3: 整列音楽プレイヤプログラムのヒント その 3