## プログラミング演習 第 10 回

リスト構造は並び順に意味のあるデータを扱うときに有用である。今回は音楽プレイヤを例題に、部分演奏や曲順の編集にダミーヘッダ付リストを使ったプレイリストを実装する。

## 問 A [10A]：音楽プレイヤ

曲名データを 端末等 (標準入力) から 1 行ずつ読み取り、プレイリストとして保存し、読み取った順に (演奏する代わりに) 印字するプログラムを作れ。曲名の長さはとりあえず最大で 80 文字とする。データの保持には連結 (単方向、ダミーヘッダ付) リストを用いること。曲名データは 1 行に 1 曲であり、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。図 1 にある `list`、`position`、`elementtype` のデータ型を用いて、次のプロトタイプを持つリスト操作関数を作成して実装せよ (プログラム中で使用しないリスト操作関数であっても正確に実装すること)。また、リスト操作にはこれらの関数のみを使用し、これ以外の関数 (`main` 関数及び自分で作るリスト操作以外の関数) ではリストノードの要素 (`element` 及び `next`) を直接参照したり、操作してはならない[1]。

`list initlist(void)` — 空リスト (ヘッドノードのみ) を作成し返す

`position first(list l)` — リスト `l` の先頭の位置 (ヘッドノード) を返す。空リストの場合、`end(l)` が返す値と同じになる。

`position end(list l)` — リスト `l` の最後のノードの位置を返す。空リストの場合、`first(l)` が返す値と同じになる。

`position next(list l, position p)` — リスト `l` 内の位置 `p` の次の要素の位置を返す。

`void printlist(list l)` — リスト `l` 内の要素を順に表示する。

`void insert(list l, position p, elementtype e)` — リスト `l` 内の位置 `p` の次に要素の値が `e` である新たな要素を挿入する。以降はそれぞれずれる。

`elementtype retrieve(list l, position p)` — リスト `l` 内の位置 `p` にある要素の値 (この場合は、文字列の先頭アドレス) を返す。

```
#define MAXLENGTH 80            /* 曲名の最大長 */

typedef char* elementtype;
struct node {
  elementtype element;
  struct node *next;
};
typedef struct node *link;
typedef link list;
typedef link position;
```

図 1: 音楽プレイヤプログラムのデータ型 (連結リストの場合)

### ヒント

1. `elementtype` は `char` へのポインタであり、文字列の先頭を指すポインタである。この場合、`struct node` の領域を確保しただけでは、文字列を確保するための領域は確保されない。`insert` において、`struct node` の領域を確保した後、文字列を入れるための領域を別に確保する必要がある。
2. 型 `list` と型 `position` は、最終的には `struct node` へのポインタ型であるが、

[1]抽象データ型とみなし、「仕様と実装の分離」、「内部情報の隠蔽」、「モジュール化の実現」を目指すためである。

型 list は、ヘッドノードを指すことでリストを表す

型 position は、要素そのものを指す

という違いがあることに注意して、適切に使い分けること。

**入出力例 (問 A)**

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

## 問 B [10B] コマンドの実装

問 A で作成した音楽プレイヤに、コマンドで指示した動作を実行する機能を実装せよ。
コマンドはコマンド文字 'p'、'r'、 'd'、'm' または 'q' と範囲を示す数値の組からなり、次の形式とする。

**コマンド文字 数字 1 数字 2**

コマンドと機能は以下のとおりである。

- コマンド文字が p のときには、(数字 1) 番目から (数字 2) 番目まで順に表示する。存在している曲の範囲を超えている場合は，表示できる曲のみ表示する。(たとえば、(数字 1)>(数字 2) の場合は何も表示しない。)
- コマンド文字が r のときには、(数字 1) 番目から (数字 2) 曲だけ逆順に表示する。存在している曲の範囲を超えている場合は，表示できる曲のみ表示する。
- コマンド文字が d のときには、(数字 2) 番目の曲を削除し、リスト全体を表示する (数字 1 は使わない)。(数字 2) 番目の曲が存在しないときには、削除せずにそのままリスト全体を表示する。
- コマンド文字が m のときには、リストの (数字 1) 番目と (数字 2) 番目を入れ替え、リスト全体を表示する。該当する曲が存在しないときには、そのままリスト全体を表示する。
- コマンド文字が q のときには、プログラムを終了する (数字 1、数字 2 は使わない)。

コマンド行の読み取りには scanf を用いてよい。数字 1、数字 2 は正の整数としてよい。

実行後に問 A と同様に曲名を入力した後に，コマンドを入力する。コマンドの処理を行った後は、現在保持しているリストに対して再びコマンドを受け付ける。

この問題では、次のプロトタイプを持つ関数を作成し、実装せよ (プログラム中で使用しなくても正確に実装すること)。また、リスト操作には問題 A と以下で示す関数のみを使用し、これ以外の関数 (main 関数及び自分で作るリスト操作関数以外の関数) ではリストノードの要素 (element 及び next) を直接参照したり、操作してはならない。

`position lseek(list l, int num)` — リスト `l` の先頭から `num` 番目の位置を返す (`num=1` のときに 1 番目のリストの位置を返す)。`num` 番目がないときには (`position` 型の)0 を返す。

`position previous(list l, position p)` — リスト `l` 内の位置 `p` の直前の要素の位置を返す。

`void delete(list l, position p)` — リスト `l` 内の位置 `p` の要素を削除する。当該要素 (と値) を保持するために使用していたメモリ領域を関数 `free` を使って解放すること。

入出力例 (問 B)

```
入力
  a song from japan
  boy meets girl
  cool hip hop
  dear my friend
  .
  p 2 4
出力
  boy meets girl
  cool hip hop
  dear my friend
入力
  r 3 2
出力
  cool hip hop
  boy meets girl
入力
  m 2 4
出力
  a song from japan
  dear my friend
  cool hip hop
  boy meets girl
入力
  d 0 2
出力
  a song from japan
  cool hip hop
  boy meets girl
入力
  q 0 0
(プログラム終了)
```

## 問 C [10C] 抽象データとしてのリスト -両方向リストによる実装-

問 B で作成した音楽プレイヤを両方向リストを用いて実装せよ。ただし、問 A 及び 問 B で示したプロトタイプの関数群は同じ仕様 (同じ機能、名前、戻り値、引数を持つ) とする。それら以外の関数 (`main` 関数及び自分で作るリスト操作関数以外の関数) は変更しないこと。

入出力例は、問 B と同じである。

ヒント

struct node の定義の中身は、両方向リストを用いるように変更する必要があるが、list、position、elementtype の定義自体は変更する必要がないことに注意。

struct node の定義変更にともない、問 A 及び 問 B で示したプロトタイプの関数群の中身も適切に変更する。

## 問 D [10D] 抽象データとしてのリスト -配列による実装-

問 B で作成した音楽プレイヤをリストを用いず、配列によって実現せよ。このとき、問 A 及び 問 B で示したプロトタイプの関数群は同じ仕様 (同じ機能、名前、戻り値、引数を持つ) とする。ただし、配列を用いる場合は position の型はアドレスでなく配列内での位置としてもよい。それら以外の関数 (main 関数及び自分で作るリスト操作関数以外の関数) は変更しないこと。

入出力例は、問 B と同じである。

ヒント

struct node、list、position の定義自体を変更する必要があることに注意。これらの変更にともない、問 A 及び 問 B で示したプロトタイプの関数群の中身も適切に変更する。

たとえば、曲名を保存する配列は elementtype のポインタ、すなわち、文字列 (char *) へのポインタのポインタである。この配列の大きさ、使用中の要素数、配列自体を一体として管理するために、次の構造体へのポインタを list とみなし、position はこの配列の添字とする。

```
typedef char* elementtype;
struct arraylist {
  int arraysize;      /* 'array' の最大要素数 */
  int arrayused;      /* 実際に使用中の個数 (つまり、有効な添字- 1 ) */
  elementtype *array;
};
typedef arraylist *list;
typedef int position; /* array[n] へのポインタとしてもよい */
```

list l が指す領域を確保する (initlist における操作例) には、たとえば、

```
list l = (list)malloc(sizeof(struct arraylist));
l->arraysize = [曲数];  /* とりあえずの曲数 */
l->arrayused = 0;
l->array = (elementtype*)malloc(sizeof(elementtype) * [曲数]);
```

とすればよい。なお、insert で要素を挿入する際に arraysize では足りなくなった場合には、realloc で array の領域を増やせばよい。

配列を用いた場合、insert で要素を挿入、delete で要素を削除するときに、当該要素以降にある各要素をひとつづつずらす必要があることに注意すること。

l->array[0]、 l->array[1] はそれぞれ 1 曲目、2 曲目の文字列の先頭アドレスとなり、各曲名用の文字列は

```
l->array[0] = (elementtype)malloc( [1 曲名の文字数] + 1 );
l->array[1] = (elementtype)malloc( [2 曲名の文字数] + 1 );
```

のように確保する。C における配列の (0 から始まる) 添字式と (1 から始まる) 曲の番号との違いに注意すること。