C 言語 025 リスト構造

【総武線】

三鷹、吉祥寺、西荻窪、荻窪、阿佐ヶ谷、高円寺、中野、東中野、大久保、新宿

JR 東日本の総武線、三鷹から新宿までの駅データを「リスト構造」を使用して記録してみます。

＜train.h＞

```
#ifndef __TRAIN_H__
#define __TRAIN_H__

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

/* 駅情報 */
typedef struct station {
    char name[32];           /* 駅名 */
    struct station* prev;    /* 前の駅 */
    struct station* next;    /* 次の駅 */
}station;

/* 路線情報 */
typedef struct line {
    station* start;          /* 始発駅 */
    station* now;            /* 現在地 */
    station* end;            /* 終着駅 */
}line;

/* 関数プロトタイプ宣言 */
line* create_line(void);              /* 路線情報を作る関数 */
void del_line(line*);                 /* 路線情報、駅情報を削除する関数 */
int set_station(line*, int, char*);   /* 路線の order 番目に駅情報を登録する関数 */
station* get_station(line*, int);     /* 路線の order 番目の駅情報を取得する関数 */
int add_station(line*, char*);        /* 路線の最後に駅情報を追加する関数 */
void del_station(line*, int);         /* 路線の order 番目に駅情報を削除する関数 */

#endif
```

＜train.c＞

```
#include "train.h"

/* 路線情報を作る関数 */
line* create_line(void) {

    /* ヒープ領域の確保 */
    line* this = (line*)malloc(sizeof(line));
    if (this == NULL) { exit(-1); }

    /* 初期化 */
    this->start = NULL;
    this->now = NULL;
    this->end = NULL;

    return this;
}

/* 路線情報、駅情報を削除する関数 */
void del_line(line* line) {

    /* 変数宣言 */
    station* this = line->start;
    station* del = NULL;

    /* 順番に駅情報を解放していく */
    while (this != NULL) {
        del = this;
        this = this->next;
        free(del);
    }

    /* 路線情報を解放する */
    free(line);
}
```

（続く）

```
/* 路線の order 番目に駅情報を登録する関数 */
int set_station(line* line, int order, char* name) {

    /* 登録する前後の駅情報を取得 */
    station* prev = get_station(line, order - 1);    /* 前の駅 */
    station* next = get_station(line, order);        /* 次の駅 */

    /* ヒープ領域の確保 */
    station* this = (station*)malloc(sizeof(station));
    if (this == NULL) { return -1; }

    /* 駅情報登録 */
    strcpy(this->name, name);    /* 駅名 */
    this->prev = prev;           /* 前の駅 */
    this->next = next;           /* 次の駅 */

    /* 前の駅の次の駅はこの駅 */
    if (prev != NULL) {
        prev->next = this;
    }

    /* 次の駅の前の駅はこの駅 */
    if (next != NULL) {
        next->prev = this;
    }

    return 0;
}
```

（続く）

```
/* 路線の order 番目の駅情報を取得する関数 */
station* get_station(line* line, int order) {

    /* 変数宣言 */
    station* this = line->start;    /* 駅情報 */
    int count = 0;                  /* カウント用変数 */

    /* 順番に駅情報を見ていく */
    while  (this->next != NULL){    /* 次の駅が NULL は終着駅 */
        if (order == count) { break; }
        this = this->next;
        count++;
    }

    return this;                    /* 見つからない場合は NULL が返る */
}

/* 路線の最後に駅情報を追加する関数 */
int add_station(line* line, char* name) {

    /* ヒープ領域の確保 */
    station* this = (station*)malloc(sizeof(station));
    if (this == NULL) { return -1; }

    /* 駅情報登録 */
    strcpy(this->name, name);    /* 駅名 */
    this->prev = line->end;      /* 前の駅 */
    this->next = NULL;           /* 次の駅はまだ不明 */

    /* 始発駅か確認 */
    if (line->end == NULL) {
        line->start = this;      /* 路線の始発駅はこの駅 */
    }
```

（続く）

```
    else {
        this->prev->next = this;    /* 前の駅の次の駅はこの駅 */
    }

    /* 路線の終着駅はこの駅 */
    line->end = this;

    return 0;
}

/* 路線の order 番目の駅情報を削除する関数 */
void del_station(line* line, int order) {

    /* 削除する駅とその前後の駅の情報を取得 */
    station* prev = get_station(line, order - 1);    /* 前の駅 */
    station* this = get_station(line, order);        /* 削除する駅 */
    station* next = get_station(line, order + 1);    /* 次の駅 */

    /* 「前の駅の次の駅」と「次の駅の前の駅」を書き換える */
    prev->next = next;
    next->prev = prev;

    /* 駅情報を削除 */
    free(this);
}
```

＜main.c＞

```
#include "train.h"

/* メイン関数 */
int main(void) {

    /* 路線情報「総武線」を作る */
    line* soubu = create_line();

    /* 駅情報、三鷹～新宿を登録 */
    add_station(soubu, "三鷹");
    add_station(soubu, "吉祥寺");
    add_station(soubu, "西荻窪");
    add_station(soubu, "荻窪");
    add_station(soubu, "阿佐ヶ谷");
    add_station(soubu, "中野");        /* 阿佐ヶ谷の次は高円寺 */
    add_station(soubu, "東中野");
    add_station(soubu, "大久保");
    add_station(soubu, "新久保");      /* 新大久保は山手線 */
    add_station(soubu, "新宿");

    /* 5 番目に高円寺駅を追加 */
    set_station(soubu, 5, "高円寺");

    /* 9 番目の新大久保を削除 */
    del_station(soubu, 9);

    /* 総武線上り、始発駅から出発 */
    soubu->now = soubu->start;
    while (1) {
        printf("%s¥n", soubu->now->name);
        if (soubu->now->next == NULL) {    /* 終点であれば終了 */
            break;
        }
```

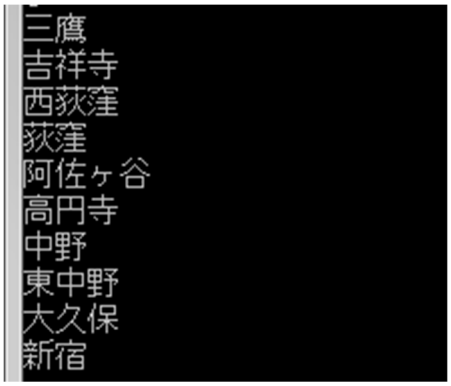
（続く）

```
        else {
            soubu->now = soubu->now->next;    /* 次の駅情報を取得 */
        }
    }

    /* 総武線と全駅をメモリから解放する */
    del_line(soubu);

    return 0;
}
```

＜実行結果＞



【解説】

レコードを構造体でまとめて、その構造体を「malloc 関数」で確保したヒープ領域に記録するのは、よくある手法です。

malloc 関数は多用される関数ですが不便な点として、ヒープ領域を確保する際にデータサイズを指定する必要があることが挙げられます。

例えば、ファイルから取得したレコードをヒープ領域に記録する場合、1 件 10 バイトのデータが 10 件あった場合は 100 バイト確保することになります。

しかし、ファイル内のデータ件数は固定でないため、正確なバイト数を確保することができません。

では、最初にまとめてヒープ領域を確保するのではなく、必要に応じて少しずつヒープ領域を確保していく方法はどうでしょうか？

【malloc 関数について】

＜コード 1＞

```
int* heap = (int*)malloc(20);
```

上記のコードの場合、int 型を 4 バイト、先頭アドレスを 100 番地とすると、ヒープ領域には下記の領域が確保され、変数 heap には先頭アドレスの 100 番地が代入されます。
この領域は「添字」や「ポインタのインクリメント」とループ文を使用して、代入、参照等、処理をまとめることができます。

アドレス	100 番地	104 番地	108 番地	112 番地	116 番地
添字	0	1	2	3	4
値					

← 連番である

＜コード 2＞

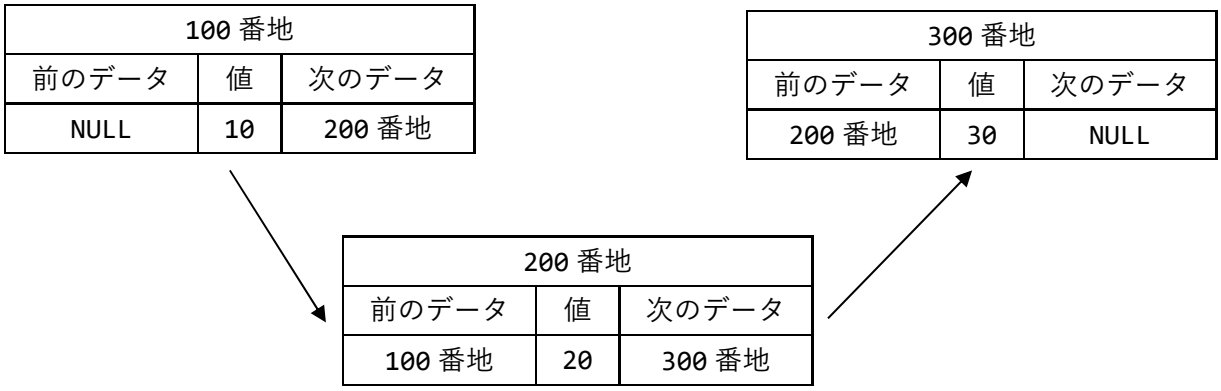
```
int* heap0 = (int*)malloc(4);
int* heap1 = (int*)malloc(4);
int* heap2 = (int*)malloc(4);
int* heap3 = (int*)malloc(4);
int* heap4 = (int*)malloc(4);
```

malloc 関数の戻り値は確保したヒープ領域の先頭アドレスですが、アドレスはメモリの空き状況によって変動します。
上記のコードの場合、malloc 関数を使用して 4 バイトずつヒープ領域を確保していますが、アドレスは連番になるとは限りません。
よって、コード 1 と同様に 20 バイト分確保していますが、添字やポインタのインクリメントが使用できず、処理をまとめることができません。

【リスト構造】

「アドレスが連番でないから不便である」ならば「連番になるような工夫」を行います。

下記の図のように、データの中に「前のデータのアドレス」と「次のデータのアドレス」を記録しておけば、データのアドレスはバラバラであっても「連結」させることができます。これを「リスト構造」と称します。



リスト構造を使用すれば、動的にヒープ領域を確保することができ、各ユーザ定義関数を使用すれば、自由に駅データを追加、削除することができます。

今回の「総武線」では、リスト構造を使用して、三鷹～新宿までをヒープ領域に記録してみました。
イメージ図は下記の通りです。