

アルゴリズムとデータ構造 授業中練習問題8

次のプログラムは「ポインタによる線形リストの実現例」(教科書のList9-3 とほぼ同じ)である。このプログラムに関して、以下の問いに答えなさい。さらに、このプログラムを入力し、自分のパソコンでコンパイル、実行できることを確認してください。なお、プログラムの日本語部分は、英語、ローマ字に変更してかまいません。

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

#define MEMBER_NO    1 /* 番号を表す定数値 */
#define MEMBER_NAME  2 /* 氏名を表す定数値 */

/*--- 会員データ ---*/
typedef struct {
    int no;          /* 番号 */
    char name[20];   /* 氏名 */
} Member;

/*--- ノード ---*/
typedef struct __node {
    Member data; /* データ */
    struct __node *next; /* 後続ポインタ */
} Node;

/*--- 線形リスト ---*/
typedef struct {
    Node *head; /* 先頭ノードへのポインタ */
    Node *crnt; /* 着目ノードへのポインタ */
} List;

/*--- 会員の番号の比較関数 ---*/
int MemberNoCmp(const Member *x, const Member *y){
    return x->no < y->no ? -1 : x->no > y->no ? 1 : 0;
}

/*--- 会員の氏名の比較関数 ---*/
int MemberNameCmp(const Member *x, const Member *y){
    return strcmp(x->name, y->name);
}

/*--- 会員データ (番号と氏名) の表示 (改行なし) ---*/
void PrintMember(const Member *x){
    printf("%d %s", x->no, x->name);
}

/*--- 会員データ (番号と氏名) の表示 (改行あり) ---*/
void PrintLnMember(const Member *x){
    printf("%d %s\n", x->no, x->name);
}

/*--- 会員データ (番号と氏名) の読み込み ---*/
Member ScanMember(const char *message, int sw){
    Member temp;

    printf("%s するデータを入力してください。 %n", message);

    if (sw & MEMBER_NO) { printf("番号:"); scanf("%d", &temp.no); }
    if (sw & MEMBER_NAME) { printf("氏名:"); scanf("%s", temp.name); }
```

```

    return temp;
}
/*--- 一つのノードを動的に生成 ---*/
static Node *AllocNode(void) {
    return calloc(1, sizeof(Node));
}
/*--- n の指すノードの各メンバーに値を設定 ----*/
static void SetNode(Node *n, const Member *x, const Node *next) {
    n->data = *x; /* データ */
    n->next = next; /* 後続ポインタ */
}
/*--- 線形リストを初期化 ---*/
void Initialize(List *list) {
    list->head = NULL; /* 先頭ノード */
    list->crnt = NULL; /* 着目ノード */
}
/*--- 関数 compare によって x と一致すると判定されるノードを探索 ---*/
Node *Search(List *list, const Member *x,
              int compare(const Member *x, const Member *y)) {
    Node *ptr = list->head;

    while (ptr != NULL) {
        if (compare(&ptr->data, x) == 0) { /* キー値が一致 */
            list->crnt = ptr;
            return ptr; /* 探索成功 */
        }
        ptr = ptr->next; /* 後続ノードに着目 */
    }
    return NULL; /* 探索失敗 */
}
/*--- 先頭にノードを挿入 ---*/
void InsertFront(List *list, const Member *x) {
    Node *ptr = list->head;
    list->head = list->crnt = AllocNode();
    SetNode(list->head, x, ptr);
}
/*--- 末尾にノードを挿入 ---*/
void InsertRear(List *list, const Member *x) {
    if (list->head == NULL) /* 空であれば */
        InsertFront(list, x); /* 先頭に挿入 */
    else {
        Node *ptr = list->head;
        while (ptr->next != NULL)
            ptr = ptr->next;
        ptr->next = list->crnt = AllocNode();
        SetNode(ptr->next, x, NULL);
    }
}
/*--- 先頭ノードを削除 ---*/
void RemoveFront(List *list) {
    if (list->head != NULL) {
        Node *ptr = list->head->next; /* 2 番目のノードへのポインタ */

```

```

    free(list->head);          /* 先頭ノードを解放 */
    list->head = list->crnt = ptr; /* 新しい先頭ノード */
}
}
/*--- 末尾ノードを削除 ---*/
void RemoveRear(List *list) {
    if (list->head != NULL) {
        if ((list->head)->next == NULL) /* ノードが一つだけであれば */
            RemoveFront(list);          /* 先頭ノードを削除 */
        else {
            Node *ptr = list->head;
            Node *pre;

            while (ptr->next != NULL) {
                pre = ptr;
                ptr = ptr->next;
            }
            pre->next = NULL;          /* pre は末尾から 2 番目 */
            free(ptr);                /* ptr は末尾 */
            list->crnt = pre;
        }
    }
}
/*--- 着目ノードを削除 ---*/
void RemoveCurrent(List *list) {
    if (list->head != NULL) {
        if (list->crnt == list->head) /* 先頭ノードに着目していれば */
            RemoveFront(list);        /* 先頭ノードを削除 */
        else {
            Node *ptr = list->head;

            while (ptr->next != list->crnt)
                ptr = ptr->next;
            ptr->next = list->crnt->next;
            free(list->crnt);
            list->crnt = ptr;
        }
    }
}
/*--- 全ノードを削除 ---*/
void Clear(List *list) {
    while (list->head != NULL) /* 空になるまで */
        RemoveFront(list);    /* 先頭ノードを削除 */
    list->crnt = NULL;
}
/*--- 着目ノードのデータを表示 ---*/
void PrintCurrent(const List *list) {
    if (list->crnt == NULL)
        printf("着目ノードはありません。");
    else
        PrintMember(&list->crnt->data);
}

```

```

/*--- 着目ノードのデータを表示（改行付き） ---*/
void PrintLnCurrent(const List *list){
    PrintCurrent(list);
    putchar('\n');
}
/*--- 全ノードのデータをリスト順に表示 ---*/
void Print(const List *list){
    if (list->head == NULL)
        puts("ノードがありません。");
    else {
        Node *ptr = list->head;

        puts("【一覧表】");
        while (ptr != NULL) {
            PrintLnMember(&ptr->data);
            ptr = ptr->next; /* 後続ノードに着目 */
        }
    }
}
/*--- 線形リストの後始末 ---*/
void Terminate(List *list){
    Clear(list); /* 全ノードを削除 */
}
/*--- メニュー ---*/
typedef enum {
    TERMINATE, INS_FRONT, INS_REAR, RMV_FRONT, RMV_REAR, PRINT_CRNT,
    RMV_CRNT, SRCH_NO, SRCH_NAME, PRINT_ALL, CLEAR
} Menu;
/*--- メニュー選択 ---*/
Menu SelectMenu(void){
    int i, ch;
    char *mstring[] = {
        "先頭にノードを挿入", "末尾にノードを挿入", "先頭のノードを削除",
        "末尾のノードを削除", "着目ノードを表示", "着目ノードを削除",
        "番号で探索", "氏名で探索", "全ノードを表示", "全ノードを削除"
    };

    do {
        for (i = TERMINATE; i < CLEAR; i++) {
            printf("(%2d) %-18.18s ", i + 1, mstring[i]);
            if ((i % 3) == 2)
                putchar('\n');
        }
        printf("( 0) 終了 :");
        scanf("%d", &ch);
    } while (ch < TERMINATE || ch > CLEAR);

    return (Menu)ch;
}
/*--- メイン ---*/
int main(void){
    Menu menu;
    List list;

```

```

Member x;

Initialize(&list);    /* 線形リストの初期化 */

do {
    switch (menu = SelectMenu()) {
        case INS_FRONT : /* 先頭にノードを挿入 */
            x = ScanMember("先頭に挿入", MEMBER_NO | MEMBER_NAME);
            InsertFront(&list, &x);
            break;
        case INS_REAR : /* 末尾にノードを挿入 */
            x = ScanMember("末尾に挿入", MEMBER_NO | MEMBER_NAME);
            InsertRear(&list, &x);
            break;
        case RMV_FRONT : /* 先頭ノードを削除 */
            RemoveFront(&list);
            break;
        case RMV_REAR : /* 末尾ノードを削除 */
            RemoveRear(&list);
            break;
        case PRINT_CRNT : /* 着目ノードのデータを表示 */
            PrintLnCurrent(&list);
            break;
        case RMV_CRNT : /* 着目ノードを削除 */
            RemoveCurrent(&list);
            break;
        case SRCH_NO : /* 番号による探索 */
            x = ScanMember("探索", MEMBER_NO);
            if (Search(&list, &x, MemberNoCmp) != NULL)
                PrintLnCurrent(&list);
            else
                puts("その番号のデータはありません。");
            break;
        case SRCH_NAME : /* 氏名による探索 */
            x = ScanMember("探索", MEMBER_NAME);
            if (Search(&list, &x, MemberNameCmp) != NULL)
                PrintLnCurrent(&list);
            else
                puts("その名前のデータはありません。");
            break;
        case PRINT_ALL : /* 全ノードのデータを表示 */
            Print(&list);
            break;
        case CLEAR : /* 全ノードを削除 */
            Clear(&list);
            break;
    }
} while (menu != TERMINATE);

Terminate(&list); /* 線形リストの後始末 */
return 0;
}

```

- 1) このプログラムの動作直後に、「先頭にノードを挿入」を指示しました。このとき、`switch` 文中の列挙型の変数 `menu` の値と `switch` 文から呼びだされる関数名を答えなさい。
- 2) このプログラムの動作直後に、「先頭にノードを挿入」を指示し、データ (番号: 1018999, 氏名: funfun) を入力しました。このとき、次の間に答えなさい。
 - (ア) `main` 関数中の `list.head->data.no` の内容を示しなさい。
 - (イ) `main` 関数中の `list.crnt->data.name[3]` の内容を示しなさい。
 - (ウ) さらに、「末尾にノードを挿入」を指示し、データ (番号: 1018777, 氏名: fun2) を入力しました。このとき、`main` 関数中の `list.head->data.name[3]` の内容を示しなさい。
- 3) このプログラムの動作直後に、「末尾にノードを挿入」を連続し 4 回指示し、4 つのデータを `[2, funfun]`, `[7, fun2]`, `[2, saito]`, `[6, ueda]` の順番で入力しました。続けて、3 回連続して「先頭にノードを挿入」を連続して指示し、3 つのデータを `[1, taka]`, `[5, ueda]`, `[2, konno]` の順番で入力しました。このとき、次の間に答えなさい。
 - (ア) この状態での線形リストの各ノードの「番号」と「名前」、先頭から末尾に向かって順に答えてください。
 - (イ) この状態で、さらに「氏名で探索」で `[ueda]` を探索し、その後「着目ノードを削除」を指示しました。削除後の線形リストの各ノードの「番号」と「名前」、先頭から末尾に向かって順に答えてください。
 - (ウ) この状態で、さらに「番号で探索」で `[2]` を探索し、その後「着目ノードを削除」を指示しました。削除後の線形リストの各ノードの「番号」と「名前」、先頭から末尾に向かって順に答えてください。