課題 2b:リストの基本操作

1 整列挿入

それまでに探索された整数すべてを小さい順に並べた整数列を配列に格納しておき，ある整数値 key を引数として呼び出されたら，key が探索されたことがなければ 0(No の意味) を返すとともに引数を小さい順に並ぶように指数列配列に格納し，key が既に探索したことがあれば 1(yes の意味) を返すだけのという，関数 (int insert_sorted_list()) を作れ.

以下に list 構造を用いて作成した inset_sorted_list 関数とそれを構成する関数群を示す.

```
/**
 * @brief 指定したポインタの後ろに key の値を挿入
 * @param key:配列に格納したい変数
 * @param pt:挿入したい位置の前のノードのポインタ
 * @return 新しく挿入したノードのアドレス
 *
 */
struct node *insert_after(int key, struct node *pt)
{
    //挿入されるキーを格納するノードへのポインタ
    struct node *new_node;
    //ノードの確保
    new_node = (struct node *)malloc(sizeof *new_node);
    if (new_node == NULL)
    { //確保できなかった場合は
        printf("Not enough memory\n");
        exit(1); //終了
    }
    new_node->key = key;        //新しいノードのキーに値を格納
    new_node->next = pt->next; //new_node の next は pt の next
    pt->next = new_node;        //pt の next は new_node
    return new_node;            //new_node の先頭アドレスを返す
}
/**
 * @brief 配列の中に key と同値が存在するか探索し存在しなければ昇順の適切な位置に挿入する.
 * @param key:配列に格納したい変数
 * @return 配列に既に key が存在すれば 1(Yes の意味)
 *          存在しなければ 0(No の意味)
 */
int insert_sorted_list(int key)
{
```

```
    struct node *p;
    struct node *insert_pt = head;
    for (p = head->next; p != NULL && p->key <= key; insert_pt = p, p = p->next)
    {
        if (key == p->key)
        {
            return 1;
        }
    }
    insert_after(key, insert_pt);
    return 0;
}
```

メールに添付したソースコードを実行すると以下のような出力を得られる.

```
insert:8
No
  8,
insert:3
No
  3,  8,
insert:12
No
  3,  8, 12,
insert:1
No
  1,  3,  8, 12,
insert:8
Yes
  1,  3,  8, 12,
insert:10
No
  1,  3,  8, 10, 12,
insert:5
No
  1,  3,  5,  8, 10, 12,
insert:0
No
  0,  1,  3,  5,  8, 10, 12,
insert:3
Yes
```

```
0, 1, 3, 5, 8, 10, 12,
```

この inset_sorted_list 関数は, 変数 p が NULL になるか，p の key の値が引数 key より大きくなるまで for 分を回している．もし，list の中に引数 key と同値の数があれば 1 を return する．変数 insert_pt は関数 insert_after を用いるための変数であり，for 文を抜けたときの p のアドレスの一個前の node のアドレスが格納されている．

2 全削除

全削除関数 (void delete_all(void)) をつくれ
以下に今回作成した delete_all 関数を示す.

```
/**
 * @brief list の全ノードを削除する
 * @return なし
 */
void delete_all(void)
{
    struct node *p;
    struct node *temp;
    //p の値が NULL になるまで繰り返す
    for (p = head->next; p != NULL; p = p->next)
    {
        //head の次の次のノードアドレスを一時変数に記憶しておく
        temp = p->next;
        //head の次のノードのメモリを開放する
        free(head->next);
        //head の次のノードを，temp に記憶しておいた次の次のノードのアドレスにつなぎ直す.
        head->next = temp;
    }
}
```

現在は特定のリストしか delete できないので，実用上はリストの先頭のアドレスを渡すほうがいいと思った.

3 指定されたキーのあるノードの削除

key の値により指定されたノードの削除関数 (int delete(int key)) を作れ．返り値は，削除したなら 1, 削除できなかったなら 0 とする．ただし，リストは整列されていなく，key の重複はないものとする．
以下に作成した delete 関数を示す.

```
/**
 * @brief 指定された key を持つノードを削除する
 * @param key:削除したい値
 * @return 0:削除できなかった
 *         1:削除
 */
int delete (int key)
{
    struct node *p;
    struct node *temp;
    //リストの最後まで for 文を回す
    for (p = head->next; p != NULL; p = p->next)
    {
        if (p->next != NULL && key == p->next->key)
        {//p の次のノードが key と同じ値であった場合
            //削除したいノードの次のノードのアドレスを一時変数に記憶させておく
            temp = p->next->next;
            //key と同じ値を持つノードのメモリを開放
            free(p->next);
            //p の next に削除したノードの次のノードのアドレスを格納
            p->next = temp;
            //削除したので 1 を返す
            return 1;
        }
    }
    //key と同じ値がなければ 0 を返す
    return 0;
}
```

実行結果を以下に示す.
　指定されたノードがリストから削除されていることがわかる

4 リストの分割

　与えられたリストについて，key の値が奇数であるリストと偶数であるリストの 2 つに分割する関数
void oddeven(struct node *head, struct node *oddhead, struct node *evenhead)
を作れ.
　以下に作成した oddeven 関数を示す.

```
/**
 * @brief 与えられたリストについて key が偶数なら偶数リストへ奇数なら奇数リストに分割する
```

```
 * @param *head:分割したいリストの先頭のアドレス
 *        *oddhead:奇数リストの先頭のアドレス
 *        *evenhead:偶数リストの先頭のアドレス
 */
void oddeven(struct node *head, struct node *oddhead, struct node *evenhead)
{
    //操作用の変数を作成
    struct node *oddlist = oddhead, *evenlist = evenhead, *p;
    //head の最後まで for 文を回す
    for (p = head->next; p != NULL; p = p->next)
    {
        //遇奇判定
        if ((p->key) % 2 != 0)
        {
            //奇数の場合奇数リストに p のアドレスを追加する
            oddlist->next = p;
            //oddlist が指す場所を更新
            oddlist = oddlist->next;
        }
        else
        {
            //動作は奇数と同様
            evenlist->next = p;
            evenlist = evenlist->next;
        }
    }
    //リストの最後に NULL を追加して終了
    oddlist->next = NULL;
    evenlist->next = NULL;
}
```

実行結果を以下に示す.

考察感想

リスト構造を使った場合，配列よりも自由度が高い実装ができるので便利だと感じた，一方で今回は単方向リストだったので先頭からしか探索できないなどの制約があったので双方向リストも試して見たいと思った．また，動作の説明が文章だけだとわかりにくいので最初から出力結果の例みたいなのをもっと示してほしいと思った.

協力者

19257504:太田那菜