アルゴリズム及び演習 第 3 回演習解答

小野 孝男*

2007 年 5 月 7 日

1. リストを用いたスタックの実装例を図 1 に示す. スタックでは, プッシュとポップは同じ端からデータのやりとりを行う. そのことに気をつけながら, “スタックでの操作” を “リストでの操作” に変換してゆけばよい. 但し, テキストのリストの実装では “リストの先頭からのデータのやりとり” は $O(1)$ 時間でできるが, “リストの末尾からのデータのやりとり” には $O(k)$ 時間 (k はリスト中のデータの個数) 掛かることに注意が必要である.
2. 定義 a, b から, 全ての閉括弧は “直前にある, 閉じ括弧とまだ対応していない開き括弧” に対応することがわかる. つまり, 形としてはスタックを使うのがよさそうだということがわかる.

   次に定義の c を考えると, 再帰を使うよりも繰り返しを使った方がプログラムが簡単になると考えられる. そこで, まずスタックを使った例を示す (図 2).

   図 2 のプログラムでは, スタックを使って “閉じ括弧と対応していない開き括弧” を管理している. “開き括弧” が来たときには (これは当然 “閉じ括弧と対応していない開き括弧” であるので) スタックにプッシュし, “閉じ括弧” が来たときには (スタックの先頭にある “開き括弧” が対応するので) スタックからポップする. そして, “スタックに開き括弧がない状態で閉じ括弧が来た” (その閉じ括弧に対応する開き括弧が存在しない) 場合及び “データを最後まで読んでもスタックに開き括弧が残っている” (スタックに残った開き括弧に対応する閉じ括弧が存在しない) 場合には “括弧が整合していない”, そうでなければ “括弧が整合している” と判断することができる.

   なお, スタックを用いれば複数種類の括弧が混ざっている場合でも正しく動作するが, 今回の問題のように括弧が 1 種類しかない場合には “スタックに存在するデータの個数” を数えれば十分である. このように考えると, スタックを使わないで括弧の整合性を判定するプログラムを作ることもできる (図 3).
3. リスト `old` を反転して新しいリスト `new` を作ることを考える. これは, “`old` の先頭からデータを 1 ずつ順に取り出し `new` の先頭に追加してゆく” ことで実現可能である (3 個のデータの場合の動作を図 4 に示す). これは, 実質的には “`delete` → `insert`” を繰り返せばよく, 図 5 のプログラムで実現することができる.

* ono@is.nagoya-u.ac.jp

```
struct stack {
    struct element *data;
    int num;
};

void createStack(struct stack *stack)
{
    stack->data = create();
    stack->num = 0;

    return stack;
}

void push(struct stack *stack, char item)
{
    insert(stack->data, 1, item);
    stack->num++;
}

void pop(struct stack *stack)
{
    struct element *top = stack->data->next;
    delete(stack->data);
    free(top);
    stack->num--;
}

char top(struct stack *stack)
{
    return access(stack->data, 1);
}

int empty(struct stack *stack)
{
    return stack->num == 0;
}
```

図 1　リストを用いたスタックの実装

```
int parensAreMatched(char data[], int n)
{
    int i;
    struct stack s;
    create(&s);

    for (int i = 0; i < n; ++i) {
        if (data[i] == '(') {
            push(&s, data[i]);
        } else {
            if (empty(&s)) {
                return 0;
            }
            pop(&s);
        }
    }
    return empty(&s);
}
```

図 2　スタックを使って括弧の整合を調べるプログラム

```
int parensAreMatched(char data[], int n)
{
    int i;
    int parens = 0;

    for (int i = 0; i < n; ++i) {
        if (data[i] == '(') {
            parens++;
        } else {
            if (parens == 0) {
                return 0;
            }
            parens--;
        }
    }
    return parens == 0;
}
```

図 3　スタックを使わずに括弧が整合しているかどうかを判定するプログラム

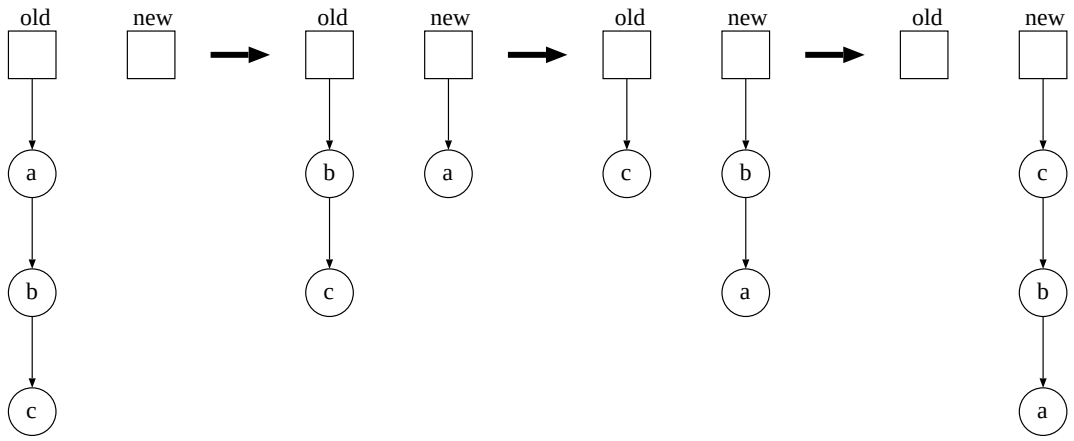

図 4　リスト反転の概略

```
struct element *nreverse(struct element *old)
{
    struct element *new = create();

    while (old->next != NULL) {
        struct element *item = old->next;
        delete(old, 1);
        item->next = new->next;
        new->next = item;
    }

    return new;
}
```

図 5　リストを (破壊的に) 反転するプログラム