

# データ構造とアルゴリズム実験レポート

## 課題 2 : <連結リスト, スタック, キュー>

<201811320><1 クラス><江頭輝>

締切日 : <2019 年 4 月 22 日>

提出日 : <2019 年 4 月 27 日>

### 1 必須課題 2-1

この課題では、セルに整数値を格納する連結リストを実現するクラス List を Java 言語で実装する。

#### 1.1 List.java の作成

##### 1.1.1 実装の方針

まず、List クラスを定義する。

##### 1. コンストラクタ

セルの直後 (ポインタは) 'p.next' のような形で表現し、セルの値は 'p.data' のように表記する。そして、リストの最初を表すリストを head として定義した。

##### 2. insert\_cell メソッド

リスト p の次に d を値として持つリストを挿入するので、まずデータが d のリストを作成する。その後、p の次にそのリストが来て、その次に p.next が来るような実装を行う。

##### 3. delete\_cell メソッド

引数として指定したリスト p のポインタを表すコピーを作成して、p のポインタは p のポインタのポインタになるようにリスト関係を実装する。

##### 4. display

まず head のコピーを作成する必要がある。

##### 5. 実装コードおよびコードの説明

##### 1.1.2 実装の方針

insert\_cell と insert\_cell\_top では即席変数である new\_cell を作成しそのデータに引数を渡すことで新たなセルを作成する。top の場合はポインタにこれまでの head をよび、前者では引数として渡したのポインタがポインタになるようにする。delete\_cell ではポインタに p のポインタのポインタを代入することでセルの削除を行うようにしている。delete\_cell\_top ではではなく textthead になった。display ではポインタが null になるまで値をループで出力する。

```

1 // List.java
2 public class List {
3     static List head;
4     List next;
5     int data;
6
7     // 新たなリストを生成 p => new_cell => p.next
8     static void insert_cell(List p, int d) {
9         List new_cell = new List();
10        new_cell.data = d;
11        new_cell.next = p.next;
12        p.next = new_cell;
13    }
14
15    // head = new_cell データは(d) => old_head
16    static void insert_cell_top(int d) {
17        List new_cell = new List();
18        new_cell.data = d;
19        new_cell.next = head;
20        head = new_cell;
21    }
22
23    // p.next を p.next.next で上書きしてp.を削除next
24    static void delete_cell(List p) {
25        List q = p.next;
26        p.next = q.next;
27    }
28
29    // head を head.next で上書き
30    static void delete_cell_top() {
31        List q = head;
32        head = q.next;
33    }
34
35    // head のコピーを.で終わりまでループしてそれぞれのデータを表示next
36    static void display() {
37        List tmp = head;
38        while (tmp != null) {
39            System.out.print(tmp.data);
40            tmp = tmp.next;
41        }
42        // 改行
43        System.out.println();
44    }
45
46    // List クラスの動作確認
47    public static void main(String[] args) {
48        insert_cell_top(1);
49        insert_cell(head, 3);
50        insert_cell(head, 2);
51        display();
52
53        delete_cell(head);
54        display();
55
56        delete_cell_top();
57        display();
58    }
59 }

```

図 1 List.java のソースコード

### 1.1.3 実行結果

まず, List.java を以下のコマンドでコンパイルする.

```

-----
$ javac List.java
-----

```

コンパイル後に以下のコマンドで実行を行った。

```
-----  
$ java List  
123  
13  
3  
-----
```

`main` メソッドでまず、1 を先頭に追加、先頭の直後に 3, 2 を順番に追加していった。なので、43 行目の `display` 関数で出力される値は 123 である。次に `head` セルの次のセルを削除したので 46 行目の `display` 関数では 13 が出力される。最後に `delete_cell_top` 関数を呼び出し、先頭のセルを削除したため、49 行目での `display` 関数では 3 のみが出力された。

#### 1.1.4 考察

このコードだと、`head` の先頭にセルを追加する場合新しいリストを作る必要があります。また、`head` のみ仕様が異なるので、余分に `insert_cell_top` メソッドと `delete_cell_top` メソッドを呼ぶ必要がありました。それを踏まえると、ダミーのデータ部を持つ `head` を作成するほうが実装が単純になると考えられます。

## 2 必須課題 2-2

### 2.1 QueueArray.java の作成

#### 2.1.1 実装の方針

`QueueArray` クラスにはエンキューを行うための `enqueue` メソッドとデキューを行うための `dequeue`。キュー全体を表示する `display` メソッドを作成した。そしてその動作を確認するための `main` メソッドを作成する。

##### 1. `QueueArray` クラスのコンストラクタ

キューの作成に必要なものとしてコンストラクタで整数の配列を表す `queue`、格納できるメモリ量を表す長さを表す `length`、キューの先頭を `front` 末尾を `rear` と定義した。

##### 2. `enqueue` メソッド

まずオーバーフローの処理を行う。もともとの配列 `Queue` を超えるような値の挿入が起こってしまう可能性があるからだ。

オーバーフローしないことがわかったら要素の末尾に引数で指定した値を代入して `rear` に適切な処理を行う。

##### 3. `dequeue` メソッド

まずアンダーフローの処理を行う。それはキューの要素が存在しない場合にデキューを行わないようにするためである。

アンダーフローの処理後に要素の先頭の処理をしてその値を返すようにする。

##### 4. `display` メソッド

要素の位置はコンストラクタで指定した `front` と `rear` をつかって表現することができるので、その間の値を返すように実装を行う。

```

1 public class QueueArray {
2     int length, front, rear;
3     int[] queue;
4
5     // 指定された長さの配列を生成するコンストラクタ
6     QueueArray(int len) {
7         queue = new int[len];
8         length = len;
9         front = 0;
10        rear = 0;
11    }
12
13    // queue の後ろに引数を代入する
14    void enqueue(int val) {
15        if (rear > length-1) {
16            System.err.println("QueueOverflow!!");
17            System.exit(1);
18        }
19        queue[rear] = val;
20        rear++;
21    }
22
23    // queue の先頭を返し、番号を一つずらす
24    int dequeue() {
25        if (front == rear) {
26            System.err.println("QueueUnderflow!!");
27            System.exit(1);
28        }
29        int x = queue[front];
30        front++;
31        return x;
32    }
33
34    // queue の先頭から末尾までを出力
35    void display() {
36        for (int i = front; i < rear; i++) {
37            System.out.print(queue[i]);
38        }
39        System.out.println();
40    }
41
42    // QueueArray クラスの動作確認
43    public static void main(String[] args) {
44        QueueArray queue = new QueueArray(10);
45
46        queue.enqueue(1);
47        queue.enqueue(2);
48        queue.display(); // 12
49
50        System.out.println(queue.dequeue()); // 1
51        System.out.println(queue.dequeue()); // 2
52    }
53 }

```

図 2 QueueArray.java のソースコード

### 2.1.2 実装コードおよびコードの説明

図 2 QueueArray.java のソースコードを示す。

4.1.1 節で述べた、enqueue メソッドと dequeue メソッドは、それぞれ 14~21 行目、24~32 行の部分に相当する。

enqueue は末尾が配列の長さより大きくなる場合はオーバーフローとなり終了する。15~18 行目でその実装をしている。

オーバーフローを起こす場合、System.exit(1) を書いて終了することができる。

オーバーフローを起こさない場合、queue[rear] が要素の末尾にあたるのでそこに引数として与えた int 型

の変数 `val` を代入する。

キューのサイズを変えるために `rear` の値を一つ上げる。そうすることで要素の末尾は右に一つずれたような挙動になる。

`dequeue` は要素の中身がない場合はアンダーフローとなり終了する。25~28 行目でその実装をしている。`front` と `rear` が等しいときに `dequeue` を行くと、アンダーフローになるので、その条件のときに `Queue Underflow!!` と出力して先程と同様に `System.exit(1)` によりプログラムを終了させる。デキューされるのは `queue[front]` であるのでその値を返す値である `x` に代入する。引数の値を代入した後に `front` を一つ動かし要素を拡大する。

### 2.1.3 実行結果

1.1.3 節で述べたコンパイル方法と同様に、`QueueArray.java` を `javac` コマンドでコンパイルし、`QueueArray.class` を生成する。その後、以下のコマンドを入力することによって、`QueueArray.java` のプログラムを実行する。

```
-----  
$ java QueueArray  
12  
1  
2  
-----
```

配列の長さが 10 の QUEUE を作成し、そこに順番に 1, 2 をエンキューする。`display` すると 12 が出力される。

デキューをすると 1 次に 2 が出力される。この状態でデキューを行うと `underflow!` が出力されて終了することも確認できた。

### 2.1.4 考察

`enqueue` メソッドの計算量は  $O(1)$  であり、`dequeue` メソッドの計算量も  $O(1)$  である。これはあらかじめ先頭と末尾の位置を変数で格納していることにより実現できた。

## 3 発展課題 2-3

### 3.1 双方向循環リストの作成

#### 3.1.1 実装の方針

`ListDL` クラスには次のポインタを表す `next` と前のポインタを表す `prev` 値を表す `val` を定義する。それらに対して以下の処理を行い実装をする。

##### 1. コンストラクタ

はじめは循環するように `next` と `prev` の関係を作る。そのために、`this` の次も前も指すポインタ部は `this` になるように書く。引数が与えられたときはその値を格納する。

##### 2. `insertNext`, `insertPrev` メソッド

引数となるリストを cell とおくと前者は `this->cell->this.next`  
後者は `this.prev->cell->this`  
という位置関係になっている。この2つの関係は3つのリストの位置関係というコードで共通するため、`_insertNext` で共通化してコードを実装する。

### 3. delete メソッド

`this.prev->this.next` の結びつけをおこない、その後 `this` を初期化する。それぞれ `_Delete` メソッドと `initLinks` メソッドを作成する。

### 4. search メソッド

コピーを作成し、ループさせる。データが引数と同じ場合にそのリストを返すように実装します。またリストのループはポインタを移動していき、リストを一周して `this` になった場合に終了する。終了したら `null` を返すように実装する。

### 5. display メソッド

4と同様のループ機構を書き、データ部をコマンドラインへ出力するようにする。

## 3.1.2 実装コードおよびコードの説明

図 7ListDL.java のソースコードを示す。4.1.1 節で述べた、

コンストラクタは 6~13 行目の部分に相当する。head セルはデータ値を持たないようにするため、引数がない場合にはデータ値である `val` は定義しない。そして、引数がある場合のコンストラクタにはデータ値を定義する。またどちらにも循環するようにリンクの初期化メソッドを使っている。

`insertNext`, `insertPrev` メソッドは 36~43 行目の部分に相当する。セルの前後にそれぞれ双方向にリストを結びつけるための4つコードがそれぞれ書かれている。

`delete` メソッドは 46~49 行目の部分に相当する。セルの前後をリンク付ける。その後削除されたセルから前後のセルが呼び出せられるのを防ぐため、リンクの初期化を行う。

`search` メソッドは 52~61 行目の部分に相当する。コピーを一周ループさせ、データの評価を行う。このときにコピーをポインタ部のセルにさせようにして終了条件をそのコピーが `this` になったときにする。そうすることでループの開始では `while` の条件に含まれないようにしている。データ値と引数が等しいときにそのリストを返す。このループ中でこの条件を満たすものがない場合、`null` を返す。

`display` メソッドは 63~70 行目の部分に相当する。ループ機構は同じである。`while` ループの中でデータ値を出力している。一周ループした後の 69 行目で改行をした。

このコードの確認のため、このクラス内に `main` メソッドを実装した。head をダミーセルとして `1->2->5` と表示できるようにリストを挿入する。

`elem` は `search` メソッドを使うことで `2->5->1` というリストになる。3 を挿入して `1->2->3->5` のリストに、最後に 5 を削除して `1->2->3` というリストになることを想定している。

## 3.1.3 実行結果

1.1.3 節で述べたコンパイル方法と同様に、ListDL.java を `javac` コマンドでコンパイルし、ListDL.class を生成する。その後、以下のコマンドを入力することによって、ListDL.java のプログラムを実行する。

-----

```
$ java ListDL
```

12

1

2

-----

### 3.1.4 考察

`search` メソッドの計算量は単純な線形探索を採用しているので  $O(n)$  である。`insert`, `delete` メソッドの計算量は  $O(1)$  である。

## 4 必須課題 2-4

### 4.1 双方向循環リストの改良

#### 4.1.1 実装の方針

`ListDLInt` はデータを `int` 型にする。`ListDL` のデータの型は `Object` 型にした。これはどのような型に対しても代入可能にするためである。その後に配列や外部ファイルからのリスト作成用メソッドを実装する。

##### 1. `readFromArray` メソッド

引数として受けとった配列を順に `insert` する。

##### 2. `writeToArray` メソッド

ループするごとにそのインデックスに応じた配列に格納する。このときリスト数分の配列をさくせいするためにリストの長さを返すメソッド `length` も作成する。

##### 3. `readFromFil`, `writeToFile`, メソッド

ファイルから一行ずつ読み込むコードを実装する。

#### 4.1.2 実装コードおよびコードの説明

図 7ListDL.java のソースコードを示す。追加コードは 75~143 行目である。リストをループするコードを少々改良した。for 文で実装することでコード行数を減らしている。`readFromFile`, `writeToFile` メソッドは `BufferedWriter` でファイルを開き、その後終了した後にメモリ管理のためにファイルを閉じる必要がある。そのときに `try_finally` 文を使った。これにより、エラーなどにも対応できる。

#### 4.1.3 実行結果

動作を確認するコード `ListDLmain.java` を作成しました。コードはいかに記す。また `writeToArray` メソッドの動作確認用のテキストファイル `array.txt` を作成した。これらはこれまで同様 `ListDLmain.java` をコンパイルした後にはかのコマンドを実行する。

-----

```
$ java ListDLmain
```

```
5 -> 1 -> 2 ->
```

```
5 -> 3 -> 1 -> 2 ->
```

```
3 -> 1 -> 2 ->
```

```
1 -> 2 -> 3 -> 4 -> 5 ->
12345
1357 -> 25 -> 262 -> 7224 -> 43 -> 63
-> 7747 -> 3622 -> 43 -> 4 ->
```

```
-----
out.txt-----
3
1
2
-----
```

#### 1. readFromArray メソッド

1, 2, 3, 4, 5 の配列を作成して、それを引数として、readFromArray メソッドを呼び出したので、readedList には 1 -i 2 -i 3 -i 4 -i 5 が入っている。

#### 2. writeToArray メソッド

foreach 文で readedList を配列として取り出したところ、12345 が出力された。

#### 3. readFromFile メソッド

先程作成した array.txt からのリストを表示したところ、

1357 -i 25 -i 262 -i 7224 -i 43 -i 63 -i 7747 -i 3622 -i 43 -i 4 -i  
と表示される。

#### 4. writeToFile メソッド

out.txt という引数で作成したところ想定通りの出力が行われた。

### 4.1.4 考察

この問題で2つの課題に直面しました。

1つ目は型の改良です。一つ前の課題での変数 val をどの型にしようか迷いました。調べたところ、ジェネリクスを用いた型の指定が多く出てきました。今回はこれまでで学んだ構文のみで継承を行おうと思ったので、Object 型を使うことにしました。これは型の決まっていないものなのでできるだけ利用を避けるべきです。なので外部のファイルではできるだけ使わないようなコードを心がけました。

2つ目は writeToArray メソッドです。ここではうまく表現できずに Object 型で返すようにしました。ここは引数の型を判別して、それに対応した型を返すべきだと思いました。またリストに連番を振っていないため for 文に追加で Count という変数を用意して、リストに番号をつけました。for 文で複数の引数を持ったり、for 文自体のインデックスを利用する方法が必要だと感じました。



```

1 // ListDL.java
2 public class ListDL {
3     private ListDL prev, next;
4     int val;
5
6     ListDL() {
7         initLinks();
8     }
9
10    ListDL(int val) {
11        this.val = val;
12        initLinks();
13    }
14
15    // prev <=> cell <=> next
16    private static void __Insert(ListDL cell, ListDL prev, ListDL next) {
17        prev.next = cell;
18        next.prev = cell;
19        cell.next = next;
20        cell.prev = prev;
21    }
22
23    // prev と next の結びつけ
24    private static void __Delete(ListDL prev, ListDL next) {
25        // 条件を満たすときはもでないnextnull
26        prev.next = next;
27        next.prev = prev;
28    }
29
30    // 結びつきをなくす
31    private void initLinks() {
32        this.prev = this.next = this;
33    }
34
35    // this => cell => this.next
36    void insertNext(ListDL cell) {
37        __Insert(cell, this, this.next);
38    }
39
40    // this.prev => cell => this
41    void insertPrev(ListDL cell) {
42        __Insert(cell, this.prev, this);
43    }
44
45    // this.prev => this.next (no this)
46    void delete() {
47        __Delete(this.prev, this.next);
48        initLinks();
49    }
50
51    // リストのコピーを走査して、データ値が引数と同じものを返す
52    ListDL search(int i) {
53        ListDL tmp = this.next;
54        while (tmp != this) {
55            if (tmp.val == i) {
56                return tmp;
57            }
58            tmp = tmp.next;
59        }
60        return null;
61    }
62
63    void display() {
64        ListDL tmp = this.next;
65        while (tmp != this) {
66            System.out.print(tmp.val);
67            tmp = tmp.next;
68        }
69        System.out.println();
70    }

```

図3 ListDL.java のソースコード 1

```

1
2 // リストの実装をテストするためのクラス
3 public static void main(String[] args) {
4     ListDL head = new ListDL(); // ダミーセルの生成
5     ListDL elem;
6
7     head.insertNext(new ListDL(2)); // セルの先頭への追加
8     head.insertNext(new ListDL(1));
9     head.insertPrev(new ListDL(5)); // セルの末尾への追加
10    head.display(); // リストの表示
11
12    elem = head.search(2); // セルを探す elem.val == 5
13    elem.insertNext(new ListDL(3)); // 探したセルの直後にセルを追加
14    head.display();
15
16    elem = head.search(5); // セルを探す elem.val == 5
17    elem.delete(); // 探したセルを削除
18    head.display();
19 }
20 }

```

図 4 ListDL.java のソースコード 2

```

1  import java.io.*;
2
3  // ListDLInt.java
4  public class ListDL {
5      private ListDL next, prev;
6      Object val;
7
8      ListDL() {
9          // 循環する
10         initLinks();
11     }
12
13     ListDL(Object val) {
14         initLinks();
15         this.val = val;
16     }
17
18     // prev <=> cell <=> next
19     private static void __Insert(ListDL cell, ListDL prev, ListDL next) {
20         prev.next = cell;      // prev => cell
21         next.prev = cell;      // next => cell
22         cell.next = next;      // cell => next
23         cell.prev = prev;      // cell => prev
24     }
25
26     // prev と next の結びつけ
27     private static void __Delete(ListDL prev, ListDL next) {
28         prev.next = next;
29         next.prev = prev;
30     }
31
32     // 初期化
33     private void initLinks() {
34         this.next = this;
35         this.prev = this;
36     }
37
38     // this => cell => this.next
39     void insertNext(ListDL cell) {
40         __Insert(cell, this, this.next);
41     }
42
43     // this.prev => cell => this
44     void insertPrev(ListDL cell) {
45         __Insert(cell, this.prev, this);
46     }
47
48     // this.prev => this.next (no this)
49     void delete() {
50         __Delete(this.prev, this.next);
51         initLinks();
52     }
53
54     // リストのコピーを走査して、データ値が引数と同じものを返す
55     ListDL search(Object i) {
56         for (ListDL j = this.next; j != this; j = j.next) {
57             if (i.equals(j.val)) {
58                 return j;
59             }
60         }
61         return null;
62     }
63
64     void display() {
65         for (ListDL j = this.next; j != this; j = j.next) {
66             if (j.val != null) {
67                 System.out.print(j.val + "→");
68             }
69         }
70         System.out.println();
71     }
72
73     // 配列からリストの要素を読み込むメソッド readFromArray()
74     ListDL readFromArray(Object[] Array) {
75         for (Object val : Array) {
76             this.insertPrev(new ListDL(val));
77         }
78         return this;
79     }

```

図5 ListDL.java<sub>1</sub>のソースコード 1

```

1
2 // リストの要素数を返す
3 int length() {
4     int Count = 0;
5     for (ListDL i = this.next; i != this; i = i.next) {
6         Count++;
7     }
8     return Count;
9 }
10
11 // リストの要素を書き出した配列を返すメソッド writeToArray()
12 int[] writeToArray() {
13     // リストと同じ長さの配列を作成してリストの値を代入する。
14     int[] res = new int[this.length()];
15     int Count = 0;
16     for (ListDL i = this.next; i != this; i = i.next) {
17         res[Count] = (int) i.val;
18         Count++;
19     }
20     return res;
21 }
22
23 // ファイルからリストの要素を読み込むメソッド readFromFile()
24 void readFromFile(String readfile) {
25     try {
26         // 引数のパスにあるファイルを読み込んで一行ずつリストに追加
27         File file = new File(readfile);
28         BufferedReader br = new BufferedReader(new FileReader(file));
29         try {
30             String str;
31             while ((str = br.readLine()) != null) {
32                 insertPrev(new ListDL(str));
33             }
34         } finally {
35             // ファイルを閉じる
36             br.close();
37         }
38     } catch (FileNotFoundException e) {
39         // ファイルが存在しない
40         System.out.println("FileNotFound");
41     } catch (IOException e) {
42         e.printStackTrace();
43     }
44 }
45 // ファイルにリストの要素を書き出すメソッド writeToFile() を追加しなさい。
46 void writeToFile(String fileName) {
47     try {
48         // リストをループするごとに 引数のパスにあるファイルにデータを書き込む
49         File file = new File(fileName);
50         PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(file)));
51         try {
52             for (ListDL i = this.next; i != this; i = i.next) {
53                 pw.println(i.val);
54             }
55         } finally {
56             // ファイルを閉じる
57             pw.close();
58         }
59     } catch (IOException e) {
60         e.printStackTrace();
61     }
62 }
63 }

```

図 6 ListDL.java のソースコード 2

```

1  /**
2   * ListDLInt
3   */
4  public class ListDLInt extends ListDL {
5      int val;
6      ListDLInt() {
7          // super();
8      }
9      ListDLInt(int val) {
10         // super();
11         this.val = val;
12     }
13 }

```

図 7 ListDLInt.java のソースコード

```

1  // ListDLmain.java
2  // リストの実装をテストするためのクラス
3  public class ListDLmain {
4      public static void main(String[] args) {
5          ListDLInt head = new ListDLInt(); // セルの生成
6          ListDL elem;
7
8          head.insertNext(new ListDL(1)); // セルの先頭への追加
9          head.insertNext(new ListDL(5));
10         head.insertPrev(new ListDL(2)); // セルの末尾への追加
11         head.display(); // 5 -> 1 -> 2
12
13         elem = head.search(5); // セルを探す
14         elem.insertNext(new ListDL(3)); // 探したセルの直後にセルを追加
15         head.display(); // 5 -> 3 -> 1 -> 2
16
17         elem = head.search(5); // セルを探す
18         elem.delete(); // 探したセルを削除
19         head.display(); // 3 -> 1 -> 2
20
21         // readFromArray
22         ListDL readedList = new ListDLInt();
23         int[] array = {1, 2, 3, 4, 5};
24         readedList = readedList.readFromArray(array);
25         readedList.display();
26
27         // writeToArray
28         for (int i : readedList.writeToArray()) {
29             System.out.print(i);
30         }
31         System.out.println();
32
33         // readFromFile
34         ListDLInt fileList = new ListDLInt();
35         fileList.readFromFile("ex2/array.txt");
36         fileList.display();
37
38         // writeToFile
39         head.writeToFile("ex2/out.txt");
40     }
41 }

```

図 8 ListDLmain.java のソースコード