データ構造とアルゴリズム実験レポート
課題２：<Java によるプログラミングの復讐>

<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。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。

1.1.2　実装コードおよびコードの説明

1.1.3　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 の場合はのポインタにこれまでの textttthead をよび、前者では引数として渡したのポインタがのポインタになるようにする。delete_cell ではのポインタに 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 textttthead になった。display ではのポインタが null になるまで値をループで出力する。

1.1.4　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
```

```java
// List.java
public class List {
    static List head;
    List next;
    int data;

    // 新たなリストを生成 p => new_cell => p.next
    static void insert_cell(List p, int d) {
        List new_cell = new List();
        new_cell.data = d;
        new_cell.next = p.next;
        p.next = new_cell;
    }

    // head = new_cellデータは(d) => old_head
    static void insert_cell_top(int d) {
        List new_cell = new List();
        new_cell.data = d;
        new_cell.next = head;
        head = new_cell;
    }

    // p.next を p.next.next で上書きしてp.を削除next
    static void delete_cell(List p) {
        List q = p.next;
        p.next = q.next;
    }

    // head を head.next で上書き
    static void delete_cell_top() {
        List q = head;
        head = q.next;
    }

    // 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
    static void display() {
        List tmp = head;
        while (tmp != null) {
            System.out.print(tmp.data);
            tmp = tmp.next;
        }
        // 改行
        System.out.println();
    }

    // List クラスの動作確認
    public static void main(String[] args) {
        insert_cell_top(1);
        insert_cell(head, 3);
        insert_cell(head, 2);
        display();

        delete_cell(head);
        display();

        delete_cell_top();
        display();
    }
}
```

図 1　List.java のソースコード

```
123
13
3
---------------------------------------
```

main メゾッドでまず、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 関数で出力っされる値は 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。

1.1.5 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 メゾッドと delete_cell_top メゾ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 メゾ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 メゾッドを作成した。そしてその動作を確認するための main メゾッドを作成する。

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 メゾッド

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 メゾッド

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 メゾッド

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。

2.1.2 実装コードおよびコードの説明

図 2QueueArray.java のソースコードを示す．2.1.1 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

```java
public class QueueArray {
    int length, front, rear;
    int[] queue;

    // 指定された長さの配列を生成するコンストラクタ
    QueueArray(int len) {
        queue = new int[len];
        length = len;
        front = 0;
        rear = 0;
    }

    // queue の後ろに引数を代入する
    void enqueue(int val) {
        if (rear > length-1) {
            System.err.println("Queue␣Overflow␣!!");
            System.exit(1);
        }
        queue[rear] = val;
        rear++;
    }

    // queue の先頭を返し、番号を一つずらす
    int dequeue() {
        if (front == rear) {
            System.err.println("Queue␣Underflow␣!!");
            System.exit(1);
        }
        int x = queue[front];
        front++;
        return x;
    }

    // queue の先頭から末尾までを出力
    void display() {
        for (int i = front; i < rear; i++) {
            System.out.print(queue[i]);
        }
        System.out.println();
    }

    // QueueArray クラスの動作確認
    public static void main(String[] args) {
          QueueArray queue = new QueueArray(10);

          queue.enqueue(1);
          queue.enqueue(2);
          queue.display();                          // 12

          System.out.println(queue.dequeue());     // 1
          System.out.println(queue.dequeue());     // 2
    }
}
```

図 2　QueueArray.java のソースコード

る。front と rear が等しいときに `dequeue` を行うと、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

1.1.4 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 メゾッドの計算量は O(1) であり、dequeue メゾッドの計算量も 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。