

实验9

练习1

题目

很久很久以前，有一个魔王。他说起话来非常抽象，没人能直接听懂。但他的话还是能逐步解释成为人的话的，其解释规则有以下两种形式（“ \rightarrow ”表示解释）：

$$(1) \alpha \rightarrow \beta_1 \beta_2 \cdots \beta_m$$

$$(2) (\theta \delta_1 \delta_2 \cdots \delta_n) \rightarrow \theta \delta_n \theta \delta_{n-1} \theta \cdots \theta \delta_1 \theta$$

设大写英语字母表示魔王的词汇，小写英语字母表示人的词汇，希腊字母表示魔王或人的词汇。魔王可用人的词汇。第一种形式的解释规则具体有以下两条：

$$\textcircled{1} B \rightarrow tAdA$$

$$\textcircled{2} A \rightarrow sye$$

编写程序，将魔王的话解释成为人的话。

例如，输入B(ehnxgz)B，输出tsyedsyeezegexenehetsyedsye。如果“t,d,s,y,e,z,g,x,n,h”分别与“天、地、上、一、鹅、追、赶、下、蛋、恨”对应，那么人的话是“天上一鹅地上一鹅鹅追鹅赶鹅下鹅蛋鹅恨鹅天上一鹅地上一鹅”。

解析

使用一个类型为HashMap的变量rules保存规则，使用链式队列Queue翻译第一种形式的规则，使用链栈Stack翻译第二种形式的规则。

这里定义了新的数据结构链式队列Queue，支持enqueue()、dequeue()、getHead()、getTail()、reverse()、reversed()、toArray()、toArrayInt()和迭代，并重写了toString方法。（这里定义的Stack将在之后的实验代码中多次使用）

代码

Queue.java

```
import java.util.Iterator;
import java.util.NoSuchElementException;

// 使用Java泛型实现的链式队列，支持迭代和toString
class Queue<T> implements Iterable<T> {

    private Node front;
    private Node rear;
    private int length;

    private class Node {

        private T data;
```

```
        private Node next;

        private Node(T data, Node next) {
            this.data = data;
            this.next = next;
        }
    }

    public boolean isEmpty() {
        return this.front == null;
    }

    public int length() {
        return this.length;
    }

    public T getHead() {
        if (this.front != null) {
            return this.front.data;
        } else {
            return null;
        }
    }

    public T getTail() {
        if (this.rear != null) {
            return this.rear.data;
        } else {
            return null;
        }
    }

    public void enqueue(T data) {
        if (this.front == null) {
            this.front = new Node(data, null);
            this.rear = this.front;
        } else {
            this.rear.next = new Node(data, null);
            this.rear = this.rear.next;
        }
        this.length++;
    }

    public T dequeue() {
        if (this.front == null) {
            return null;
        } else {
            T data = this.front.data;
            this.front = this.front.next;
        }
    }
}
```

```

        if (this.front == null) {
            this.rear = null;
        }
        this.length--;
        return data;
    }
}

public void reverse() {
    if (this.front == null) {
        return;
    }
    Node newFront = new Node(this.front.data, null);
    Node p = this.front.next;
    while (p != null) {
        newFront = new Node(p.data, newFront);
        p = p.next;
    }
    this.front = newFront;
}

public Queue<T> reversed() {
    if (this.front == null) {
        return new Queue<T>();
    }
    Queue<T> result = new Queue<T>();
    result.front = new Node(this.front.data, null);
    Node p = this.front.next;
    while (p != null) {
        result.front = new Node(p.data, result.front);
        p = p.next;
    }
    return result;
}

public T[] toArray() {
    T[] result = (T[]) new Object[this.length];
    Node p = this.front;
    for (int i = 0; i < this.length; i++) {
        result[i] = p.data;
        p = p.next;
    }
    return result;
}

public int[] toArrayInt() {
    int[] result = new int[this.length];
    Node p = this.front;
    for (int i = 0; i < this.length; i++) {
        result[i] = (int) p.data;
    }
}

```

```

        p = p.next;
    }
    return result;
}

public String toString() {
    Node p = this.front;
    StringBuilder builder = new StringBuilder();
    builder.append("[");
    while (p != null) {
        builder.append(p.data + ", ");
        p = p.next;
    }
    builder.delete(builder.length()-2, builder.length());
    builder.append("]");
    return builder.toString();
}

public Iterator<T> iterator() {
    return new QueueIterator();
}

private class QueueIterator implements Iterator<T> {

    private Node current = front;

    public boolean hasNext() {
        return current != null;
    }

    @Override
    public void remove() {
        // not supported
    }

    public T next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        T data = current.data;
        current = current.next;
        return data;
    }

}

}

```

Exercise1.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

class Exercise1 {

    private static Queue<Character> translate(char character, Map<Character,
String> rules) {
        Queue<Character> result = new Queue<>();
        String value = rules.get(character);
        for (int i = 0; i < value.length(); i++) {
            char c = value.charAt(i);
            if (c >= 65 && c <= 90) {
                Queue<Character> queue = translate(c, rules);
                while (queue.getHead() != null) {
                    result.enqueue(queue.dequeue());
                }
            } else if (c >= 97 && c <= 122) {
                result.enqueue(c);
            } else if (c == '(') {
                int j = i + 1;
                while (value.charAt(j) != ')') {
                    j++;
                }
                Stack<Character> stack = translateBracketContents(value.substring(i
+ 1, j));

                while (stack.peek() != null) {
                    c = (char) stack.pop();
                    if (c >= 65 && c <= 90) {
                        Queue<Character> queue = translate(c, rules);
                        while (queue.getHead() != null) {
                            result.enqueue(queue.dequeue());
                        }
                    } else if (c >= 97 && c <= 122) {
                        result.enqueue(c);
                    }
                }
                i = j;
            }
        }
        return result;
    }

    private static Stack<Character> translateBracketContents(String expr) {
        Stack<Character> result = new Stack<>();
        for (int i = 1; i < expr.length(); i++) {
            char c = expr.charAt(i);
```

```

        result.push(expr.charAt(0));
        result.push(c);
    }
    return result;
}

public static void main(String[] args) {
    // input
    Scanner in = new Scanner(System.in);
    System.out.println("Please enter the num of rules:");
    int n = Integer.parseInt(in.nextLine());
    System.out.println("Please enter rules by line (For example, \"B
tAdA\\"):");
    HashMap<Character, String> rules = new HashMap<Character, String>();
    for (int i = 0; i < n; i++) {
        String line = in.nextLine();
        char key = line.split(" ")[0].charAt(0);
        String value = line.split(" ")[1];
        rules.put(key, value);
    }
    System.out.println("Please enter the expression:");
    String expr = in.nextLine();
    in.close();
    // process
    Queue<Character> result = new Queue<Character>();
    for (int i = 0; i < expr.length(); i++) {
        char c = expr.charAt(i);
        if (c >= 97 && c <= 122) {
            result.enqueue(c);
        } else if (c >= 65 && c <= 90) {
            Queue<Character> queue = translate(c, rules);
            while (queue.getHead() != null) {
                result.enqueue(queue.dequeue());
            }
        } else if (c == '(') {
            int j = i + 1;
            while (expr.charAt(j) != ')') {
                j++;
            }
            Stack<Character> stack = translateBracketContents(expr.substring(i
+ 1, j + 1));
            while (stack.peek() != null) {
                c = (char) stack.pop();
                if (c >= 65 && c <= 90) {
                    Queue<Character> queue = translate(c, rules);
                    while (queue.getHead() != null) {
                        result.enqueue(queue.dequeue());
                    }
                } else if (c >= 97 && c <= 122) {
                    result.enqueue(c);
                }
            }
        }
    }
}

```

```

        }
    }
    i = j;
}
}
// print
while (result.getHead() != null) {
    System.out.print(result.dequeue());
}
}
}

```

需要注意的是，括号不支持嵌套，例如(eh(nxg)z)是不支持的。
如果一定需要实现括号嵌套，请像下面这样输入：

```

Please enter the num of rules:
1
Please enter rules by line (For example, "B tAdA"):
C nxg
Please enter the expression:
(ehCz)

```

这样的操作是支持的。

输入

```

Please enter the num of rules:
2
Please enter rules by line (For example, "B tAdA"):
B tAdA
A sye
Please enter the expression:
B(ehnxgz)B

```

输出

```

tsyedsyeezegexenehetsyedsye

```

附Stack.java

这里使用的Stack即在实验8中定义的链栈Stack，尽管其代码与实验8中完全相同，但为严谨起见，还是附在实验报告中。

```
import java.util.Iterator;
import java.util.NoSuchElementException;

// 使用Java泛型实现的链栈，支持迭代和toString
class Stack<T> implements Iterable<T> {

    private Node top;
    private int length;

    private class Node {

        private T data;
        private Node next;

        private Node(T data, Node next) {
            this.data = data;
            this.next = next;
        }

    }

    public boolean isEmpty() {
        return this.top == null;
    }

    public int length() {
        return this.length;
    }

    public T peek() {
        if (this.top == null) {
            return null;
        } else {
            return this.top.data;
        }
    }

    public void push(T data) {
        this.top = new Node(data, this.top);
        this.length++;
    }

    public T pop() {
        if (this.top == null) {
```



```

        return null;
    } else {
        T data = this.top.data;
        this.top = this.top.next;
        this.length--;
        return data;
    }
}

public String toString() {
    Node p = this.top;
    StringBuilder builder = new StringBuilder();
    builder.append("[");
    while (p != null) {
        builder.append(p.data + ", ");
        p = p.next;
    }
    builder.delete(builder.length() - 2, builder.length());
    builder.append("]");
    return builder.toString();
}

public Iterator<T> iterator() {
    return new StackIterator();
}

private class StackIterator implements Iterator<T> {

    private Node current = top;

    public boolean hasNext() {
        return current != null;
    }

    @Override
    public void remove() {
        // not supported
    }

    public T next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        T data = current.data;
        current = current.next;
        return data;
    }
}

```

```
}
```

心得体会

1. 使用队列和栈可以使很多问题的解法变得简单。（但实际上在本题中，引入队列和栈使得问题变得更加复杂）
2. 链式队列应当加入头结点和尾结点，使入队和出队操作的时间复杂度降为 $O(1)$ 。
3. 练习1的代码应当还有优化空间，例如引入更多函数，优化递归。