アルゴリズムとデータ構造　第 10 回演習　　年　　組　　番 氏名

1. 配列は**ランダムアクセス**が可能なデータ構造であり，整数の添字（インデックス）で要素を参照することができる。この特徴を用いると，整数からデータへの変換表を作ることができる。以下は，配列を用いて 1 桁の整数を英単語に“翻訳”するプログラムである。空欄 1 ヶ所を埋めて動作を確認せよ。

```
import java.util.Scanner;

public class Program {
   // final は変更されないことを示す
   final static String[] ewords = {
      "zero", "one", "two", "three",
      "four", "five", "six", "seven",
      "eight", "nine" };
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      Sysmtem.out.print("number? ");
      int n = sc.nextInt();
      if (0 <= n && n <= 9)
         System.out.println(ewords[    ]);
   }
}
```

2. **ハッシュ関数**は**キー**と呼ばれるデータを入力とし，それからなるべくバラバラな整数値（**ハッシュ値**）を算出する関数である。以下は文字列のハッシュ関数の例であり，入力された文字列に対してなるべく重複しない整数を生成する。このハッシュ関数によって，次の文字列はどのような数値に変換されたか。

(a) zero _____　(b) one _____　(c) two _____　(d) three _____　(e) four _____

(f) five _____　(g) six _____　(h) seven _____　(i) eight _____　(j) nine _____

```
import java.util.Scanner;

public class Program {
   // 素数がよい
   final static int HASHSIZE = 31;

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      for (int i = 0; i < 10; i++) {
         System.out.print("string? ");
         String str = sc.next();
         // ハッシュ関数によって整数に変換
         int h = hash(str);
         System.out.println("code: " + h);
      }
   }

   // 任意の文字列を 0〜30 の整数に変換
   public static int hash(String key) {

      int len = key.length();
      int h = 0;
      for (int i = 0; i < len; i++) {
         // i 番目の文字の文字コード
         int code = key.charAt(i);
         // System.out.println(code);

         // これも別の素数がよい
         h = 37 * h + code;
      }
      // Java では%の結果は負になる可能性
      return Math.abs(h % HASHSIZE);
   }
}
```

3. **ハッシュテーブル**は，データをそのハッシュ値が示す配列の要素（**バケット**）に格納することによって，探索を高速化する仕組みである。下記はその原理を理解するための単純なプログラムである。空欄に 2.で求めた整数を埋めて動作を確認せよ。また，登録データ数 n に対する計算量（探索処理のみ）を考察せよ。

```
import java.util.Scanner;

public class Program {
   // 2.と同じ数にする
   final static int HASHSIZE = 31;

   // ハッシュテーブル（要素は文字列）
   static String[] hashtable;

   public static int hash(String key) {
      // 2.と全く同じ内容
   }

   public static void main(String[] args) {
      hashtable = new String[HASHSIZE];

      // 各バケットに初期データを登録する
      hashtable[     ] = "zero";
      hashtable[     ] = "one";
      hashtable[     ] = "two";
      hashtable[     ] = "three";
      hashtable[     ] = "four";
      hashtable[     ] = "five";
      hashtable[     ] = "six";
      hashtable[     ] = "seven";
      hashtable[     ] = "eight";
      hashtable[     ] = "nine";
```

```
    Scanner sc = new Scanner(System.in);
    System.out.print("word? ");
    String str = sc.next();

    // ハッシュ関数によるハッシュ値の算出
    int h = hash(str);
    if (hashtable[h] == null)
       System.out.println("not found.");
    else
       System.out.println(hashtable[h]);
  }
}
```

4. 3.のようなハッシュテーブルでは，ハッシュ値が互いに等しい（**衝突**する）データは複数登録することができない。**チェーン法**では各要素を連結リストにすることによって，複数のデータの登録を可能とする。下記は，単語の出現回数を数えるプログラムである。空欄を埋めて完成させ，動作を理解せよ。

```
/* Node.java */
public class Node {
   public String word; // 単語（キー）
   public int count;  //出現回数
   Node next;  // 次のノード

   Node(String word) {
      this.word = word;
      this.count = 0;
      this.next = null;
   }
}

/* HashTable.java */
public class HashTable {
   private final int HASHSIZE;

   // 配列の各要素は連結リスト
   private Node[] hashtable;

   public HashTable(int size) {
      HASHSIZE = size;
      hashtable = new Node[size];
   }

   public int hash(String key) {
      // 2.と全く同じ内容
   }

   // 全要素を表示する
   public void printAll() {
      for (int i = 0; i < HASHSIZE; i++) {
         for (Node n = hashtable[i];
               n != null; n = n.next) {
            System.out.printf("%2d %s%n",
               n.count, n.word);
         }
      }
   }

   // 単語を検索し、なければ登録する
   public Node lookUp(String word) {
      // ハッシュ値でハッシュテーブルを探索
      int h =

      Node n =

      while (n != null) {
         if (word.equals(n.word))
            return

         // 連結リストをたどる
         n =
      }

      // 未登録ならば連結リストの先頭に挿入
      n = new Node(word);
      n.next =

      hashtable[    ] = n;
      return n;
   }
}

/* Program.java */
import java.util.Scanner;

public class Program {
   public static void main(String[] args) {
      HashTable ht = new HashTable(59);
      Scanner sc = new Scanner(System.in);
      while (true) {
         String word = sc.next();
         If (word.equals("QUIT")) break;
         Node n = ht.lookUp(word);
         n.count++;
      }
      ht.printAll();
   }
}
```

5. 4.のプログラムについて，「59」の部分はそのままでハッシュ関数を下記のものなどに置き換えて表示を比較し，なるべく高速な探索のためにはどのようなハッシュ関数がよいか考察せよ。

```
public static int hash(String key) { return key.length() % HASHSIZE; }
public static int hash(String key) { return key.charAt(0) % 13; }
public static int hash(String key) { return key.charAt(0) % HASHSIZE; }
```