CS 112 Introduction to Programming

Indefinite Loops: Motivation; while/do-while Statements

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Midterm
- Covers topics from Lec. 1-21 (today’s lecture)
- Support materials posted on the Schedule page
- Review on Wednesday
  - Please suggest topics you want to review

PS5 walk-through time?
Recap: Arrays

- **array**: an object that stores many values of the same type.

- Declare an array
  - `<type>[] <name> = new <type>[<length>];`

- Access array elements
  - `<name>[<index>]` // access
  - `<name>[<index>] = <value>;` // modify

- Access initializer
  - `<type>[] <name> = {<value>, <value>, ... <value>};`
<table>
<thead>
<tr>
<th>Operation</th>
<th>Code</th>
</tr>
</thead>
</table>
| Create an array with random values            | ```java
  double[] a = new double[N];
  for (int i = 0; i < N; i++)
    a[i] = Math.random();
``` |
| Print the array values, one per line          | ```java
  for (int i = 0; i < N; i++)
    System.out.println(a[i]);
``` |
| Find the maximum of the array values          | ```java
  double max = Double.NEGATIVE_INFINITY;
  for (int i = 0; i < N; i++)
    if (a[i] > max) max = a[i];
``` |
| Compute the average of the array values       | ```java
  double sum = 0.0;
  for (int i = 0; i < N; i++)
    sum += a[i];
  double average = sum / N;
``` |
| Copy to another array                         | ```java
  double[] b = new double[N];
  for (int i = 0; i < N; i++)
    b[i] = a[i];
``` |
| Reverse the elements within an array          | ```java
  for (int i = 0; i < N/2; i++)
  {
    double temp = b[i];
    b[i] = b[N-1-i];
    b[N-i-1] = temp;
  }``` |
Recap: Example Array Use Pattern: Using Array as Counters/Accumulators

- Create an array equal to the size of the number of categories

- Loop over each input
  - map input’s value to array index
  - increase the array element at index

- Display result
  - map each array index back to input to display
Example: Histogram

- Grading histogram with a bucket for every 5 points:
  - 00-04:
  - 05-09:
  - ..
  - 90-94:
  - 95-99:
  - 100:
Example Array Use Pattern: Array Elements as Counters/Accumulators

- Create an array equal to the size of the number of categories
  - Q: how many categories?
  ```java
  int[] counters = new int[100/5+1];
  ```

- Loop over each input
  - map input’s value to array index
    ```java
    grade -> index is grade/5
    ```
  - increase the array element at index

- Display result
  - map each array index back to input to display
    ```java
    [index * 5, index*5+4] before 100
    ```
Example: Grading Histogram

- Histogram with the following buckets:
  - 00-59:
  - 60-64:
  - 65-69:
  - 70-74:
  - ...
  - 90-94:
  - 95-99:
  - 100:
Example: Grading Histogram

- Create an array equal to the size of the number of categories
  - Q: how many categories?

```java
int[] counters = new int[1+(100-60)/5+1];
```

- Loop over each input
  - map input's value to array index

```java
if (grade < 60)
    index = 0;
else
    index = (grade-60) / 5 + 1;
```
  - increase the array element at index

- Display result
  - map each array index back to input to display
Letter Frequency Counting

Objective: Count the frequency of letters a to z in a text file.

The inventor of Morse code, Samuel Morse (1791-1872), counted letter frequencies to assign the simpler codes to the more frequently used letters. The counters he obtained:

- E: 12000
- T: 9000
- A: 8000
- ...
- X: 400
- Z: 200

Freq. counting is also the foundation of code breaking.
Using Array as Counters/Accumulators

- Create an array equal to the size of the number of categories
  - Q: how many categories?
    ```java
    int[] counters = new int[26];
    ```
  - Loop over each input
    - map input’s value to array index
      ```java
      ch -> array index is ch-'a'
      ```
    - increase the array element at index

- Display result
  - map each array index back to input to display
    ```java
    index -> (char)(‘a’+ index)
    ```
Array Elements as Counters

Count the number of characters in a line:

```java
int[] counts = new int[26];
String line = scan.nextLine();

for (int i = 0; i < line.length(); i++) {
    char ch = line.charAt(i);
    if ('a' <= ch && ch <= 'z') {
        counts[ch - 'a']++;
    }
}
```

Mapping data to index
public static void histogram(int[] counts) {
    for (int i = 0; i < counts.length; i++) {
        if (counts[i] > 0) {
            System.out.println((char)('a' + i) + " : ");
            int h = counts[i];
            for (int j = 0; j < h; j++)
                System.out.print("*");
        }
        System.out.println();
    }
} // end of histogram

Mapping index to data for better display
Exercise: Decode cipher.txt
CaesarFile reads a file with a fixed number of lines. What if the file can contain an arbitrary number of lines of text?
Motivation

- CaesarFile reads a file with a fixed number of lines. What if the file can contain an arbitrary number of lines of text?
  - If there are more lines than the constant used in the loop, the remaining lines are not processed.
  - If there are fewer lines than the constant, program will crash.
Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do-while)
Indefinite Loops

- Number of times of a loop is not a simple number, but rather a logical condition

- **Indefinite loops** are common in programming design, e.g.,
  - Read from user until input is a non-negative number.
  - Repeat until the user has typed "q" to quit.
  - Print random numbers until a prime number is printed.
  - Search for a solution in a set.

- A key to define an indefinite loop is to identify the loop’s test condition.
These methods of the `Scanner` do not consume input; they just give information about what the next input will be.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasNext()</td>
<td>returns <code>true</code> if there is a next token</td>
</tr>
<tr>
<td>hasNextInt()</td>
<td>returns <code>true</code> if there is a next token and it can be read as an <code>int</code></td>
</tr>
<tr>
<td>hasNextDouble()</td>
<td>returns <code>true</code> if there is a next token and it can be read as a <code>double</code></td>
</tr>
<tr>
<td>hasNextLine()</td>
<td>returns <code>true</code> if there is a next line.</td>
</tr>
</tbody>
</table>
CaesarFile using for Loop

```java
public static void encode(Scanner scan, int key) {
    for (; scan.hasNextLine(); )
        String text = scan.nextLine();
        String result = Caesar.encode(text, key);
        System.out.println(result);
}
```
Java has three kinds of loop statements:
- the *for* loop
- the *while* loop,
- the *do* loop, and

They have the same expressive power

Which loop statements to use will depend on the context: pick the one that is more intuitive to read
The **while** loop

- **while loop**: Repeatedly executes its body as long as a logical test is true.

```java
while (<test>) {
    <statement(s>);
}
```

- **while is a keyword**

**Example:**

```java
int num = 1; // initialization
while (num <= 10) { // test
    System.out.print(num + " ");
    num = num * 2; // update
}
// output: 1 2 4 8
```
The **do/while** loop

- **do/while loop**: Similar to while, except move test at the end:

  ```java
do {
    <statement(s)>;
  } while (<test>);
```

- **Example**:

  ```java
  int num = 1; // initialization
  do {
    System.out.print(num + " ");
    num = num * 2; // update
  } while (num <= 10); // test
  // output: 1 2 4 8
  ```
public static void encode(Scanner scan, int key) {

    while ( scan.hasNextLine() ) {

        String text = scan.nextLine();

        String result = Caesar.encode(text, key);
        System.out.println( result );
    }
}
public static void encode(Scanner scan, int key) {
    for (; scan.hasNextLine(); )
        String text = scan.nextLine();
        String result = Caesar.encode(text, key);
        System.out.println(result);
}

public static void encode(Scanner scan, int key) {
    while (scan.hasNextLine()) {
        String text = scan.nextLine();
        String result = Caesar.encode(text, key);
        System.out.println(result);
    }
}
Summary: Flow Control Statements

- Loop statements
  - for
  - while
  - do/while

- Choosing the loop control structure depends on the specific situation and personal taste

```c
for ( initialization ;
     condition ;
     increment ) {
  statement list;
}

while ( condition ) {
  statement list;
}

do {
  statement list;
} while ( condition );
```
Summary: for loop

- typically used in sequential iteration

- for loop is easy to read, since it provides visual aid to recognize the four components

  - If complex or empty initialization, increment, or condition, a for loop may not read as well as a while or do/while loop

```plaintext
for ( initialization ;
    condition ;
    increment ) {
    statement list;
}
```
Summary: while, do/while loop

- The choice between do/while and while depends on the logic of the program:
  - first check condition or first execute statement

```plaintext
while ( condition ) {
    statement list;
}
```

```plaintext
do {
    statement list;
} while ( condition );
```
(Offline Exercise): Coupon Collector

- Write a program to simulate the number of coupons to collect before collecting at least one of each N distinct coupons
Roadmap: Program Flow Control

- We have covered all syntax of program flow control
- Mastering program control structure is among the most important in terms of fully mastering computer programming
  - Program control structure may involve complex logical analysis
  - Program control structure often has substantial flexibility and hence depends on personal style
    - multiple types of conditional statements
      - if/else; switch; conditional operator
    - multiple types of loop statements
      - for, while, do/while
    - break/return in the middle of a loop
How to Grasp Program Flow Control

- Learn common loop patterns
- Conduct program analysis
- Read and practice
  - Try out the offline exercises
  - Read sample programs
Roadmap

- **Today**
  - An example loop pattern: search loops

- **Monday after break**
  - Program analysis of flow control
  - Loop pattern: robust input processing

- **Wednesday after break**
  - Loop pattern: arrays sorting loops

- **Friday after break**
  - Loop pattern: two-dimensional Page-rank loop
Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do/while)
  - Common indefinite loop patterns
    - Search loops
Example

- Design a method `isInt`: returns `true` if all chars of a string are numbers, e.g.,
  - `isInt("4822116")` returns `true`
  - `isInt("2016Mar3")` returns `false`

- Design questions:
  - for, while, or do while loop?
  - At each loop iteration (over a char), what conclusion can we have?
    - if the char is not a number
      - false; An evidence that is enough to draw conclusion; need to `break` (return)
    - if the char is a number
      - not enough evidence, need to `continue`
public static boolean isInt(String word) {
    for (int i = 0; i < word.length(); i++) {
        if (!isDigit(word.charAt(i))) {
            return false;
        }
    }
    return true;
}

public static boolean isDigit(char ch) {
    return '0' <= ch && ch <= '9';
}
Some programmers do not like a method having multiple exit (return) points. Objective: design a single exit point.
public static boolean isInt(String word) {
    for (int i = 0; i < word.length(); i++) {
        if (!isDigit(word.charAt(i))) {
            return false;
        }
    } // end of for
    return true;
}

⚠️ Question: logical conditions that the for loop should exit?

i >= word.length  || !isDigit(i)

Condition to continue the loop:

!(i >= word.length  || !isDigit(i))
De Morgan's Law

- **De Morgan's Law**: Rules used to negate boolean tests.
  - Useful when you want the opposite of an existing test.

<table>
<thead>
<tr>
<th>Original Expression</th>
<th>Negate</th>
<th>Equivalent Negate</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &amp;&amp; b</td>
<td>!(a &amp;&amp; b)</td>
<td>!a</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td>b</td>
</tr>
</tbody>
</table>

```java
!( i >= word.length || ! isDigit(i) )

(i < word.length) && isDigit(i)
```
Alternative Version

```java
public static boolean isInt(String word) {
    for (int i = 0; i < word.length(); i++) {
        if (!isDigit(word.charAt(i))) {
            return false;
        }
    }
    return true;
}
```

```java
public static boolean isInt(String word) {
    for (int i = 0; i < word.length() && isDigit(word.charAt(i));
        i++) {
    // ??? return;
}
```

Alternative Version (any issues?)

```java
public static boolean isInt(String word) {
    for (int i = 0;
        i < word.length() && isDigit(word.charAt(i));
        i++) { }
    // ??? return;
}
```

```java
public static boolean isInt(String word) {
    for (int i = 0;
        i < word.length() && isDigit(word.charAt(i));
        i++) { }
    if (i < word.length())
        return false;
    else return true;
}
```
public static boolean isInt(String word) {
    int i = 0;
    for (;
        i < word.length() && isDigit(word.charAt(i));
        i++) {}
    return i >= word.length();
}
public static boolean isInt(String word) {
    int i = 0;
    for (;
        i < word.length() && isDigit(word.charAt(i));
        i++ ) { }
    return i >= word.length();
}

public static boolean isInt(String word) {
    int i = 0;
    while (i < word.length() && isDigit(word.charAt(i)) {
        i++;
    }
    return i >= word.length();
}
(Offline) Exercise

- **Design a method** `hasAnOddDigit` : **returns** true if **any** digit of an integer n is odd, e.g.,
  - `hasAnOddDigit(4822116)` **returns** true
  - `hasAnOddDigit(2448)` **returns** false

- **Design questions:**
  - How do we loop over each digit from n
    ```java
digit = n % 10;
n = n / 10;
```
  - How do we stop
    - If the digit we just saw is odd, can we draw any conclusion?
      - Yes. We found an evidence, return true
    - If the digit is even, can we draw any conclusion?
      - No, unless we have seen all digits.
    - When do we know we have seen all digits?
public static boolean hasAnOddDigit(int n) {
    while (n != 0) {
        int digit = n % 10;
        if (digit % 2 != 0) {  // find an example,  
                               // enough to draw conclusion
            return true;
        }
        n = n / 10;
    }
    return false;  // if we reach here, no  
                   // evidence of odd, return false
}
Designs method `allDigitsOdd`: returns `true` if every digit of an integer is odd.

- `allDigitsOdd(135319)` returns `true`
- `allDigitsOdd(9174529)` returns `false`
public static boolean allDigitsOdd(int n) {
    while (n != 0) {
        if (n % 2 == 0) { // find a counter example,
            // enough to draw conclusion
            return false;
        }
        n = n / 10;
    }
    return true; // if we reach here, no
    // evidence of counter example,
    // return true
Backup Slides
Example: MatchDB

- Design a program to query a match-making database
  - database (file) format:
    - line 1: number of candidates
    - each line represents a candidate: name age and then a sequence of words (tags) each describing a character
  - user commands
    - list: display each candidate
    - match <a list of tag words> // print all matching
    - match1 <a list of tag words> // print first matching
Problem: Loop does not stop after matching first.
Alternatives

```java
public static void match1(String[] mTags) {
    for (int i = 0; i < N; i++) {
        if (match(mTags, i)) {
            System.out.println("Find " + name[i]);
            return;
        }
    }
}
```

```java
public static void match1(String[] mTags) {
    int i;
    for (i = 0;
         i < N && !match(mTags, i);
         i++ ) {
    } // end of for
    if (i < N) System.out.println("Find " + name[i]);
}
```

```java
public static void match1(String[] mTags) {
    int i = 0;
    while (i < N && !match(mTags, i)) {
        i++;
    }
    if (i < N) System.out.println("Find " + name[i]);
}
```
Implement match command that limits at most `<maxNumberMatches>` results:

```
matchN <maxNumberMatches> <tags>
```
(Offline) Practice

- Implement `matchOr` command that matches if any one of the search tags matches:
  `matchOr <searchtags>`