CSCI 136: Data Structures and Advanced Programming
Lecture 21
More iterators

Instructor: Kelly Shaw
Williams

Topics

• Iterators
• Integer representation

Your to-dos

1. Read before Wed: Bailey, Ch 14-14.1, 14.3
2. Quiz due Sat by 6pm

Announcements

• CS Colloquium this Friday, Nov 4 @ 2:35pm in Wege Auditorium (TCL 123)

Rachit Nigam (Cornell University)
Programming Support for Hardware Accelerators

Rachit Nigam is a visiting researcher in the PLSE group at University of Washington and a PhD candidate studying computer science at Cornell University. He is a part of the CAPRA and PS-Cornell research groups and is advised by Adrian Sampson. His research (Dahlia, Calypso) is focused on building high-level programming models for designing hardware accelerators.
Nonetheless, we can signal our intent with an interface.

How would we write an OrderedStructure interface?

Do its elements need to have any special property? (i.e., how would we compare them?)

Let's think about how we might implement this.

(code)

OrderedVector

Let's think about implementing an OrderedVector.

(code)

The bits of an integer

An integer is represented in computer memory as a sequence of bits, each having a value of either 0 or 1. This representation is called binary.

Binary is number system where each digit can take one of two values; i.e., the base of the system is 2.

You are probably more familiar with the base 10 number system, aka decimal.

Any integer can be represented in either system.
**Java int**

The int data type in Java has 32 bits.

```
00000000 00000000 00000000 00010111
```

is the number 23.

\[
\begin{align*}
(000000000000000000000000000000001111)_2 \\
= (0 \times 2^0) + (0 \times 2^1) + (0 \times 2^2) + (0 \times 2^3) \\
+ (0 \times 2^4) + (0 \times 2^5) + (0 \times 2^6) + (0 \times 2^7) \\
+ (0 \times 2^8) + (0 \times 2^9) + (0 \times 2^{10}) + (0 \times 2^{11}) \\
+ (0 \times 2^{12}) + (0 \times 2^{13}) + (0 \times 2^{14}) + (0 \times 2^{15}) \\
+ (0 \times 2^{16}) + (0 \times 2^{17}) + (0 \times 2^{18}) + (0 \times 2^{19}) \\
+ (0 \times 2^{20}) + (0 \times 2^{21}) + (0 \times 2^{22}) + (0 \times 2^{23}) \\
+ (0 \times 2^{24}) + (1 \times 2^{25}) + (1 \times 2^{26}) + (1 \times 2^{27}) \\
+ (0 \times 2^{28}) + (0 \times 2^{29}) + (0 \times 2^{30}) + (0 \times 2^{31}) \\
= (23)_{10}
\end{align*}
\]

---

**Bitwise Operations**

We can use bitwise operations to manipulate the 1s and 0s in the binary representation

- Bitwise 'and': &
- Bitwise 'or': |

Also useful: bit shifts
- Bit shift left: <<
- Bit shift right: >>

---

**& and |**

Given two integers \(a\) and \(b\), the bitwise or expression \(a \mid b\) returns an integer s.t.
- At each bit position, the result has a 1 if that bit position had a 1 in EITHER \(a\) OR \(b\)
- \(3 \mid 6 = ?\)
  - 011 | 110 = 111

Given two integers \(a\) and \(b\), the bitwise and expression \(a \& b\) returns an integer s.t.
- At each bit position, the result has a 1 if that bit position had a 1 in BOTH \(a\) AND \(b\)
- \(3 \& 6 = ?\)
  - 011 & 110 = 010

---

**>> and <<**

Given two integers \(a\) and \(i\), the expression \((a << i)\) returns \((a \times 2^i)\)
- Why? It shifts all bits left by \(i\) positions
  - \(1 \ll 4 = ?\)
    - 00001 << 4 = 10000

Given two integers \(a\) and \(i\), the expression \((a >> i)\) returns \((a / 2^i)\)
- Why? It shifts all bits right by \(i\) positions
  - \(1 \gg 4 = ?\)
    - 00001 >> 4 = 00000
  - \(97 \gg 3 = ?\)
    - 1100001 >> 3 = 1100
Recall: Iteration

Iteration is the repetition of a process in order to generate a (possibly unbounded) sequence of outcomes. Each repetition of the process is a single iteration, and the outcome of each iteration is then the starting point of the next iteration.

Example: Iteration with an Iterator

```java
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
100 101 102 Ø
sum 0
d 0
```

```
Iterables<Double>
head
current
sum 0
d 0
```
List<Double> ls = new SinglyLinkedList<>();
// _ initialize ls _
double sum = 0.0;
for (double d : ls) {
    sum += d;
}

Example: Iteration with an Iterator
Example: Iteration with an Iterator

```java
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

A bit iterator

Suppose we want to do the following:

On each iteration, get the next most significant bit, starting initially with the least significant bit.

BIterator to the rescue.

Note that we’re showing you this in the hope that it will serve as inspiration for Lab 7—however, the iterator you need to write for Lab 7 will be different.

(code)

What’s an `Iterator<T>`?

```java
public interface Iterator<E> {
    boolean hasNext();
    E next();
    ...
}
```

It’s a stateful object that lets you iterate through a data structure.

Example: Iteration with an Iterator

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

Iteration is terminated!
<table>
<thead>
<tr>
<th>Recap &amp; Next Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Today:</strong></td>
</tr>
<tr>
<td>- Iterators</td>
</tr>
<tr>
<td>- Number representations</td>
</tr>
<tr>
<td><strong>Next class:</strong></td>
</tr>
<tr>
<td>- Tree ADT</td>
</tr>
</tbody>
</table>