

Week 3

Coding Algorithms & Complexity

EXPLORING
Java



This Week

**Lecture: Coding algorithms,
Control constructs &
Complexity**

Java Genesis:

–Ch4: Control constructs

Quick Quiz for Ch3

Code Creation

1. **Specify** the problem
2. **Develop** a scenario to solve the problem
3. **Refine** the scenario into Java code
4. **Review** the code and modify/improve
5. **Test** the code for errors
6. **Verify** that the code works



Problem

finding the maximum

Problem Statement

Numbers are supplied one after the other. Find the maximum of this collection of numbers.

Specification

Find a number which

- is one of the numbers in the given collection of supplied numbers
- has the property that no other number in the collection is bigger

A Scenario

1. To begin, let the maximum-so-far be the first number supplied.
2. When a new number is supplied, let the maximum-so-far become the larger of the new number supplied and the previous value of the maximum-so-far.

```

import genesis.*;
public class Maximum {

    public static void main (String [ ] args) {
        int maxSoFar =
            DialogBox.requestInt("First number:");
        Transcript.print(maxSoFar+" ");
        int nextNum;
        for (int i=2; i<=6; i++) {
            nextNum =
                DialogBox.requestInt("Next number:");
            Transcript.print(nextNum+" ");
            maxSoFar = Math.max(maxSoFar, nextNum);
        }
        Transcript.println( );
        Transcript.println("maximum is "+maxSoFar);
    }
}

```

Verification

How can we verify that the algorithm works?

An Invariant

The value of the variable *maxSoFar* is always the maximum of the integers so far supplied.

Reason by structural induction:

- it is true initially (when the 1st number is supplied)
- it remains true after the for-loop block is executed each time

Observation

When trying to understand how a system changes, it is often helpful to concentrate on those aspects of the system that do not change

i.e. the **invariants!**

e.g. conservation of mass, energy, momentum



Problem

calculating the gcd

Specification

Given two positive integers, find that number which is

- a common factor of both the given integers
- larger than any other common factor

A Scenario

1. Find the factors of the first integer.
2. Find the factors of the second integer.
3. List all the numbers that are factors of both of the given integers.
4. From this list select the largest integer.

Example

- factors of **36** are 1, 2, 3, 4, 6, 9, 12, 18, 36
- factors of **24** are 1, 2, 3, 4, 6, 12, 24
- common factors are 1, 2, 3, 4, 6, 12
- **12** is the largest integer in this list;
hence the $\text{gcd}(\mathbf{36}, \mathbf{24}) = \mathbf{12}$

Euclid's Algorithm

1. Replace the larger of the two integers by the difference between the larger and the smaller.
2. Continually repeat this process until the two numbers are the same.
3. This common number is the gcd.

Example

- start with **36** and **24**
- replace **36** by **36-24** to give **12** and **24**
- replace **24** by **24-12** to give **12** and **12**
- hence the $\text{gcd}(\mathbf{36}, \mathbf{24}) = \mathbf{12}$

```

import genesis.*;
public class Euclid {

    public static void main (String [ ] args) {
        int num1 =
            DialogBox.requestInt("first integer:");
        int num2 =
            DialogBox.requestInt("second integer:");
        Transcript.print
            ("gcd of "+num1+" and "+num2+" is ");
        while (num1 != num2) {
            if (num1 > num2) num1 = num1 - num2;
            else num2 = num2 - num1;
        }
        Transcript.println(num1);
    }
}

```

Verification

Can we prove that Euclid's algorithm works?

An Observation

If we have a pair of positive integers and replace the larger by the difference between it and the smaller, the pair of integers that result have the same common factors!

An Invariant

The gcd of *num1* and *num2* is unchanged and is the value we want to compute.

Reason by structural induction:

- It is true initially (when *num1* and *num2* are the integers supplied).
- It remains true after replacing the larger by the difference between it and the smaller.

Termination

Can we be sure Euclid's algorithm terminates?

An Observation

- The numbers *num1* and *num2* are always positive integers.
- At the completion of each execution of the while-loop, the maximum of *num1* and *num2* has strictly decreased.



Problem

Create a program to test if a given +ve integer is a prime number.

More generally, create a program that lists all the prime factors of a given +ve integer.

Specification

Given a +ve integer, list every +ve integer which

- is a prime number;
- divides exactly into the given integer.

Example

Consider the +ve integer **144**:

factors of **144** are

1, 2, 3, 4, 6, 9, 12, 16, 18, 24, 36, 48, 72, 144

of which only **2** and **3** are prime.

Hence the prime factors of **144** are **2** and **3**.

A simpler problem?

Create a program that lists **all** the factors of a given +ve integer.

A scenario to extract all factors

Suppose n is the given +ve integer

1. let $k = 1$
2. if k divides exactly into n then k is a factor; add k to the list of factors of n
3. increase k by 1 and repeat steps 2 and 3, stopping when k exceeds n

```

import genesis.*;
public class AllFactors {

    public static void main (String [ ] args) {
        int num = DialogBox.requestInt
            ("supply an integer:");
        Transcript.println
            ("The factors of "+num+" are");
        int taf = 1;
        while (taf <= num/2) {
            if (num % taf == 0) {
                Transcript.print(taf + " ");
            }
            taf++;
        }
        Transcript.println(num);
    }
}

```

How can we extract just the **prime** factors rather than all the factors?

An observation:

the smallest factor (other than 1) of a number is always prime.

A scenario to extract prime factors

Suppose n is the given +ve integer

1. let $k = 2$
2. if k divides n then k is a prime factor;
add k to the list of prime factors of n
3. replace n by n/k ;
if k still divides n repeat this step
4. increase k by 1 and repeat steps 2, 3 and 4,
stopping when n equals 1

Example

Find the prime factors of **15925**:

2, 3 and 4 are not factors of 15925

5 is a factor of 15925: $15925 / 5 = 3185$

5 is a factor of 3185: $3185 / 5 = 637$

5 and 6 are not factors of 637

7 is a factor of 637: $637 / 7 = 91$

7 is a factor of 91: $91 / 7 = 13$

7, 8, 9, 10, 11, and 12 are not factors of 13

13 is a factor of 13: $13 / 13 = 1$

Another observation:

if a number is not prime, its smallest prime factor is less than or equal to the square root of the number.

```

import genesis.*;
public class PrimeFactors {

    public static void main (String [ ] args) {
        int num = DialogBox.requestInt("supply an integer:");
        Transcript.println("Prime factors of "+num+" are");
        int taf = 2;
        while (num != 1) {
            if (num % taf == 0) {
                Transcript.print(taf + " ");
                while (num % taf == 0) num = num / taf;
            }
            taf++;
            if (taf > Math.sqrt(num)) taf = num;
        }
        Transcript.println( );
        Transcript.println("Done");
    }
}

```

How long will it take to show that a 100 digit number is prime?

Suppose we can check 10^6 possible factors in a second. In 1 year we can check about

$$10^6 \cdot 3 \cdot 10^7 < 10^{14}$$

possible factors.

Total time to check the 100 digit number is therefore $10^{50}/10^{14} = 10^{50-14} = 10^{36}$ years!

PRIMES is in P

Agrawal, Kayal and Saxena

(Indian Institute of Technology, Kanpur)

August, 2002

<http://www.cse.iitk.ac.in/primality.pdf>