

APS105S Winter 1999 Lecture 9: Algorithms

- up to now... only shown you the mechanics of Java
 - syntax
 - semantics
- and you've seen a few programs I've written...
- but how do you create your own??

designing good programs is an *art*

- there are many ways to write a program
 - most of you will have different solutions to your labs
 - creative process is unique to each person
 - but you must also *understand* the work of others (and vice-versa)
 - you must be clear in your work!!!!

In this lecture I want to teach you 2 things...

1. how to design a program
2. how to express it → readable by others

Algorithms (Textbook 1.11)

- sequence of instructions to solve a problem
 - unambiguous, precise
 - executable (practical to carry out)
 - terminating (doesn't take infinity...)
- when designing a program to solve a problem, you are creating an algorithm!!!
 - but algorithms are language-independent
 -

Can you write a program to do this?

First, understand the input and output requirements.

INPUT: sequence of positive integers, ending in neg. number

OUTPUT: the largest number

Next, think of the variables you may need

- one to hold the user input number
- one to hold the largest seen

Write an English outline (pseudocode):

- initialize
 - start with "largest seen" to some small value (how small?)
- compute loop
 - get a number
 - is < 0 ?
 - yes, we're done... quit loop
 - else
 - is $\text{number} > \text{largest seen}$?
 - yes, remember number
 - go back and get another number
- output
 - print "largest seen"

Write the program...

...include English comments (sometimes as pseudocode) in your program to make it readable. It must explain what your program does.

```
// this is a comment
// it is ignored by the compiler
// from the '//' to the end of the line
```

Pseudocode

- algorithms must be unambiguous
 - Java is accurate, unambiguous,
 - but specifying in Java is tedious, long, detailed
 - English is concise, abstract
 - but specifying in English can be ambiguous:
 - "Fruit flies like bananas"
- how can we specify concisely, yet accurately?
- use *pseudocode*!
 - combine English *and* Java (or any programming language)
 - use basic programming constructs: if, while, methods
 - don't worry about messy syntax rules
 - readable by programmers
 - easily (?) translated into programming language
 - easier for you to think / work with **BEFORE** writing your program on the computer

Break up the problem

Usually programs can be divided into 3 phases:

- initialize and get input
- compute
 - break this up further as necessary
- print output

Example

Write a program to read a sequence of positive integers from the user, ending with a negative number. Print the largest number entered.

eg: 1 5 26 6 3 105 98 -1

outputs: 105

```
// This program gets a sequence of
// positive numbers from the user,
// ending with a negative one.
// The largest number entered is
// then printed.
class GetLargest
{
    public static void main(String[] args)
    {
        int number;
        int largestSeen = -1;

        do {
            number = Stdin.getInt();

            if( number < 0 )
                break; // quit the 'do' loop

            if( number > largestSeen )
                largestSeen = number;

        } while( true );

        // output answer only if i have one
        if( largestSeen >= 0 )
            System.out.println( largestSeen );
        else
            System.out.println( "You didn't " +
                "enter any numbers" );
    }
}
```

Another Example

Ask the user for n . Compute the Fibonacci number F_n .
Recall $F_n = F_{n-1} + F_{n-2}$, $F_2 = 1$, $F_1 = 1$.

First, understand F_n :

$$\begin{aligned} F_1 &= 1 \\ F_2 &= 1 \\ F_3 &= F_2 + F_1 = 1 + 1 = 2 \\ F_4 &= F_3 + F_2 = 2 + 1 = 3 \\ F_5 &= F_4 + F_3 = 3 + 2 = 5 \\ F_6 &= 5 + 3 = 8 \\ F_7 &= 8 + 5 = 13 \\ F_8 &= 13 + 8 = 21 \\ &\text{etc...} \end{aligned}$$

Notice at each step we just remember the previous two Fibonacci numbers to compute the next one.

- input
 - get n
- compute
 - compute F_1
 - compute F_2
 - compute F_3
 - ...
 - compute F_n
- output
 - print F_n

But

- F_1, F_2 are easy
- $F_3..F_n$ in a loop

```

1 class Fibonacci
2 {
3     public static void main(String[] args)
4     {
5         int fibN1 = 1;
6         int fibN2 = 1;
7         int fibN = 1;
8         int n = Stdin.getInt();
9
10        int i;
11        for( i = 3; i <= n; i++ )
12        {
13            fibN = fibN1 + fibN2;
14            fibN2 = fibN1;
15            fibN1 = fibN;
16        }
17        System.out.println( fibN );
18    }
19 }
    
```

Tracing

How can you tell what a program is doing?
Is it doing what you expect?

By hand... trace the program execution...

- examine it step-by-step
- write the values of the variables at each step
- use a table format

Another way to trace... put

```
System.out.println(somethingUseful);
```

after each important step.

Src Code	n	i	i <= n	fibN	fibN1	fibN2
beginning	undefined					
5					1	
6						1
7				1		
8	5					
10		0				
11		3				
			true			
13				2		
14						1
15					2	
11		4				
			true			
13				3		
14						2
15					3	
11		5				
			true			
13				5		
14						3
15					5	

Src Code	n	i	i <= n	fibN	fibN1	fibN2
11		6				
			false			
17						

Reading

Chapter 1

- 1.11

Chapter 5 of textbook

- 5.10, 5.11, 5.12