

WES-CS GROUP MEETING #5

Exercise 1: Boolean conditions

Play a game using the decks of yellow and green cards. Each green card has an English sentence that is either true or false. Each yellow card has one of three symbols: !, &&, or || (meaning *not*, *and*, *or*). The game is played as follows:

- Each player starts with 6 cards, 3 yellow and 3 green.
- When it's your turn, if you can make a logical formula that evaluates to true using at least 3 of your cards, put down the formula and draw new cards of the same colors as the ones you used. Otherwise, trade in two of your cards for two new cards of the same color.
- The first person to put down at least 13 cards wins.

The precedences of the logical operators are as follows:

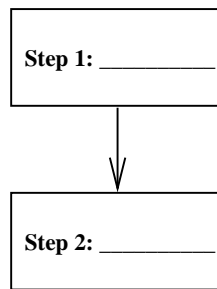
!	highest precedence
&&	next highest precedence
	lowest precedence

Exercise 2: Logical Thinking and Coding

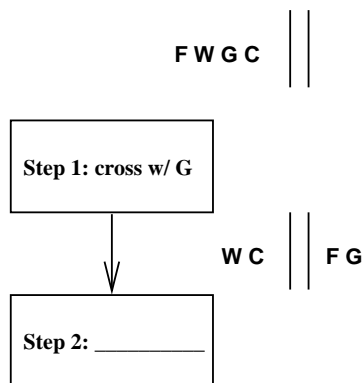
A farmer is taking a wolf, a goat, and a head of cabbage to the market. To do this, the farmer must cross a river. He has a boat, but it is only big enough to carry him and at most one of the wolf, goat, or cabbage (the farmer can also choose to cross the river alone). If the farmer leaves the wolf and the goat alone together, the wolf will eat the goat. If the farmer leaves the goat and the cabbage alone together, the goat will eat the cabbage.

Part (a).

Work with a partner to find an algorithm that gets the farmer, the wolf, the goat, and the cabbage across the river without anything getting eaten. Write the algorithm graphically like this:

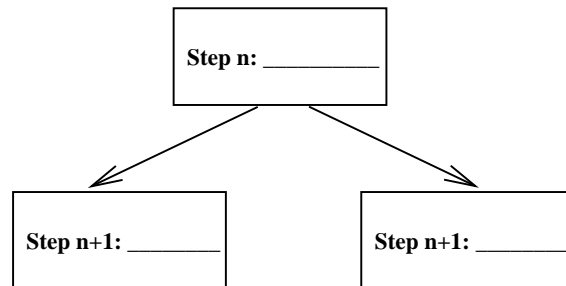


where each step says who crosses the river. You may find it helpful to keep track of who is on which side of the river by drawing *state diagrams* before and after each step of the algorithm as shown below.



Part (b).

There are two solutions to this problem. See if two groups found the two solutions; if not, everyone try to find the second solution. Once you have both solutions, change your algorithm diagram to show that there is a point where it is possible to make two different choices, both of which lead to successful results. In other words, at some point in your new diagram, you'll have a split like this:



Part (c).

Now turn the diagram you drew for Part (b) into Java code. Assume that classes `Farmer` and `Animal` have been defined, and that you have a `Farmer` variable called `john`, plus three `Animal` variables called `wolf`, `goat`, and `cabbage` (yes, this cabbage is considered an animal!). Also assume that the `Farmer` class has the methods

```
public void crossAlone()  
public void crossWithAnimal( Animal a )
```

Write a sequence of java statements (calls to these methods) to represent your diagram. To write the code for the choice in the diagram, use the method `Math.random()`, which returns a random double value greater than or equal to 0.0, and less than 1.0. This means that after some calls to the `Farmer` methods, you will have code like this:

```
double randNum = Math.random();  
if (randNum < 0.5) {  
    //Code for one solution here  
} else {  
    //Code for other solution here  
}
```

Part (d).

Look at the code you wrote for Part (c). Do you see a sequence of statements that gets repeated twice in different places? Find a way to change your code so that those statements only occur once, then draw a diagram like the one you drew for part (b) that represents your new, non-repetitive code.

Exercise 3: Algorithms

Part (a).

For this exercise, we'll say that a nickname is formed by taking any non-empty prefix of a name (the first one or more letters) followed by "ie". For example, this rule gets us the following:

Name	Nickname	Not a Nickname
Susan	Susie	Suzy
Martin	Martie	Smartie
Ann	Annie	nnie

It also gets us some nonsense nicknames:

Name	Nickname
Susan	Sie
Martin	Martinie
Ann	Anie

but we won't worry about that.

Your job is to write an algorithm (in English or in Java) to determine whether one string (called *nick*) is a nickname for another string (called *name*). For example, if *nick* is "Susie" and *name* is "Susan" then your algorithm should say that *nick* **is** a nickname for *name* (or your Java code should return *true*). If *nick* is "Susie" and *name* is "George" then your algorithm should say that *nick* is **not** a nickname for *name*.

Whether you write your algorithm in English or in Java, you should make use of the *String* methods to find out how long *nick* and *name* are, and to extract and compare substrings. Think about which pieces of the two strings you need to look at, and what the values of those pieces need to be.

Some useful *String* methods are given on the next page.

Some String Methods

char	<code>charAt(int index)</code> Returns the char value at the specified index.
boolean	<code>equals(Object anObject)</code> Compares this string to the specified object.
int	<code>indexOf(int ch)</code> Returns the index within this string of the first occurrence of the specified character.
int	<code>indexOf(int ch, int fromIndex)</code> Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	<code>indexOf(String str)</code> Returns the index within this string of the first occurrence of the specified substring.
int	<code>indexOf(String str, int fromIndex)</code> Returns the index within this string of the first occurrence of the specified substring, starting the search at the specified index.
int	<code>length()</code> Returns the length of this string.
String	<code>substring(int beginIndex)</code> Returns a new string that is the substring of this string that starts at position <i>beginIndex</i> .
String	<code>substring(int beginIndex, int pastEnd)</code> Returns a new string that is a substring of this string that starts at position <i>beginIndex</i> and ends at position <i>pastEnd-1</i> .

Part (b).

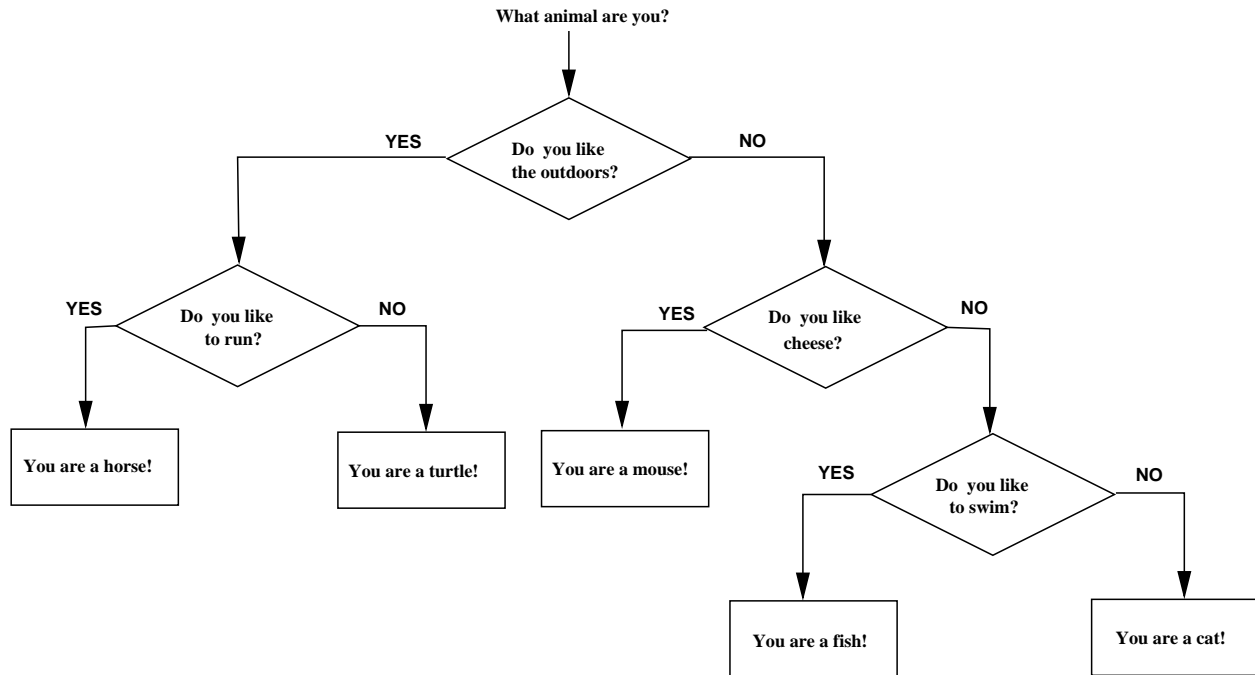
Now we'll write an algorithm that figures out whether the string in *word1* is a **subsequence** of the string in *word2*. String *word1* is a subsequence of *word2* if all of the letters in *word1* occur in *word2* in the same order, but perhaps with some other letters in between. For example

<i>word1</i>	<i>word2</i>	<i>word1</i> is a subsequence of <i>word2</i> ?
win	wisconsin	yes
wise	wisconsin	no
snow	wisconsin	no
sin	wisconsin	yes

Again, you can write your algorithm in English or in Java, but in either case you should use the String methods.

Exercise 4: If statements

Take the following quiz:



This diagram, or flowchart, is like a series of *if* statements in a program. The diamond shapes represent the *if* conditions (the decisions you need to make in order to move onward) and the square blocks represent the code in the *then* and *else* branches of the *if*.

Part (a).

Working in two groups, arrange the blue (or pink) cards in the right order so you create Java code that implements the diagram. Assume that the user will enter 1 for yes, and 0 for no. To read input, use the Java *Scanner* class. Here are two useful methods of the *Scanner* class:

int	nextInt() Finds and returns the next token of the input as an int
String	next() Finds and returns the next complete token from the input.

Hint: Start by finding the code just for the lower-right part of the diagram (the part that asks the question “*Do you like to swim?*”).

Once you have that, add the code for “*Do you like cheese?*”

Then finish the code.

Part (b).

What would you have to change in the code if the user types in Yes or No instead of 1 or 0?

Part (c).

Draw a similar diagram for a "What animal (or food, or plant, or whatever) are you?" quiz. Don't use more than four diamonds. Swap diagrams with a partner, and each write code for the other person's quiz.

blank

Blue or Pink Cards

```
Scanner in = new Scanner(System.in);

int input;

System.out.println("Do you like the outdoors?");

input = in.nextInt();

if ( input == 1) {

System.out.println("Do you like to run?");

input = in.nextInt();

if (input == 1) System.out.println("You are a horse!");

else System.out.println("You are a turtle!");

} else {

System.out.println("Do you like cheese?");
```

Blue or Pink Cards

```
input = in.nextInt();

if (input == 1) System.out.println("You are a mouse!");

else {

System.out.println("Do you like to swim?");

input = in.nextInt();

if (input == 1 ) System.out.println("You are a fish!");

else System.out.println("You are a cat!");

}

}
```

Green cards

The Earth is flat

Madison is the capital of Wisconsin

In Java, * has higher precedence than +

In Java, + has higher precedence than -

The moon is made of green cheese

17 is a prime number

There is not a DDR machine in Memorial Union

The dorms get Cartoon Network

Red and blue make purple

Green cards

A chestnut tree makes acorns

A giraffe has a black tongue

Ducks can't fly

We are learning C++ in CS302

To compare two Java strings, use `.equals` not `==`

The sun moves around the Earth

The UW-Madison was founded in 1492

There are 10 WES-CS groups

This room is CS 1240

Green cards

Miss Muffet is afraid of spiders

The time now is 3:33pm

The sun sets in the west

Java programs start executing in a class named Main

The following code will compile: `boolean b = "true";`

An if statement will repeat until the condition is false

Yellow cards

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

Yellow cards

||

||

||

||

||

||

||

||

||

||

||

||

!

!

!

Yellow cards

!

!

!

!

!

!

!

!

!