The Importance of Dry Running Programs

Paul Curzon
Queen Mary University of London

Linked resources available at
teachinglondoncomputing.org

Twitter @cs4fn

What do I mean by Dry Runs?

- Acting as a ‘computer’
  - in the original sense
  - of a human who is following an algorithm
- ‘Unplugged’ activities
  - away from a computer / compiler
- Working out in detail and step by step what a program does
  - on paper **drawing tables**
  - **acting out computation** / “compiling” on to humans

Why are these activities vital?

- Focus on semantics (meaning) not syntax (spelling)
  - understanding **what constructs do** not just rote learning how to write them
- Develops **logical thinking**
- Develops **attention to detail**
- Allows teacher to quickly **identify misconceptions** and faulty mental models
- Helps students over **barrier concepts**
- Transition step between explanations and programs

Queen Mary University of London

CS4FN

institute of CODING

EPSRC Engineering and Physical Sciences Research Council

Ada - the first programmer?

- The claim that Ada Lovelace was the first programmer arises not because of a program listing she wrote
- She created a dry run table for Babbage's machine
- She pointed out a mistake in his algorithm as a result

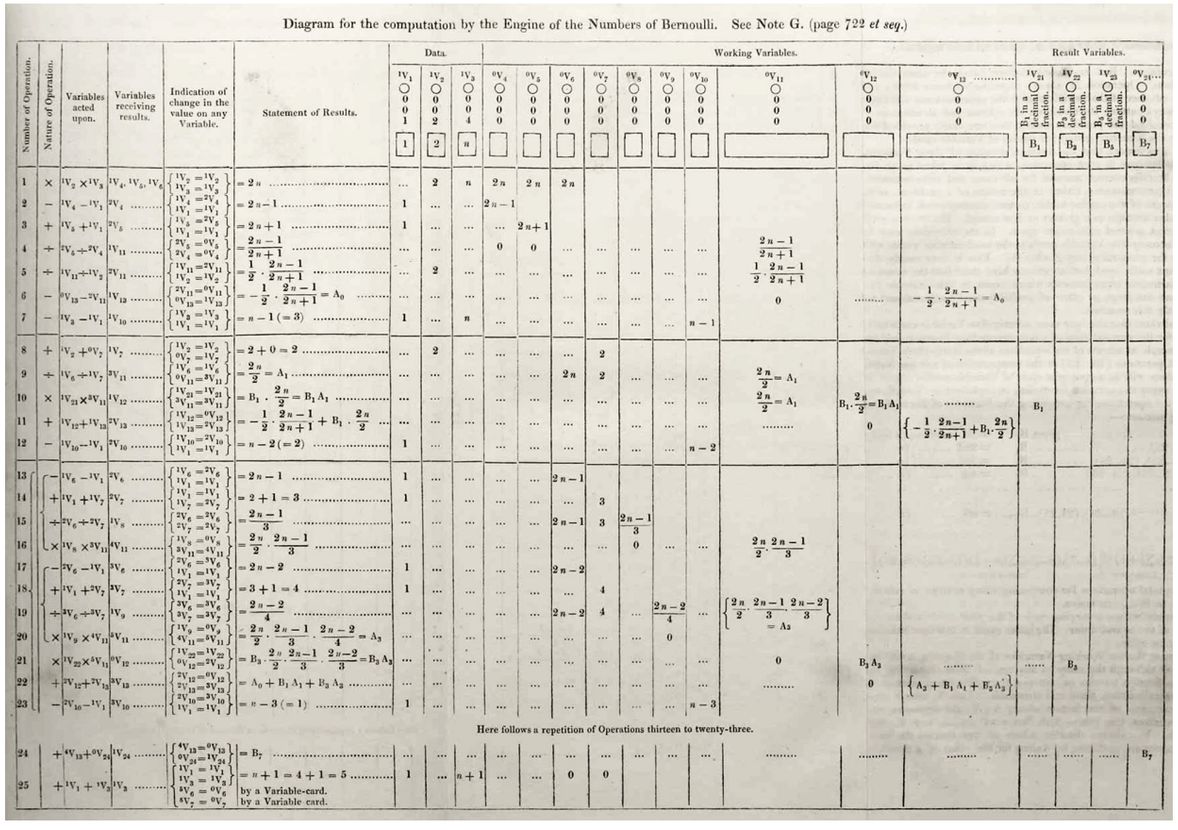


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.			Working Variables.								Result Variables.			
						$^1V_1$	$^1V_2$	$^1V_3$	$^0V_4$	$^0V_5$	$^0V_6$	$^0V_7$	$^0V_8$	$^0V_9$	$^0V_{10}$	$^0V_{11}$	$^0V_{12}$	$^0V_{13}$	$^1V_{21}$ B_1	$^1V_{23}$ B_2	$^1V_{25}$ B_3	$^0V_{27}$ ... B_7
						1	2	n														
1	×	$^1V_2 \times ^1V_3$	$^1V_4, ^1V_5, ^1V_6$	$\begin{cases} ^1V_2 = ^1V_2 \\ ^1V_3 = ^1V_3 \end{cases}$	$= 2n$	...	2	n	$2n$	$2n$	$2n$										
2	−	$^1V_4 - ^1V_1$	$^2V_4$	$\begin{cases} ^1V_4 = ^2V_4 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 2n-1$	1	...	...	$2n-1$												
3	+	$^1V_5 + ^1V_1$	$^2V_5$	$\begin{cases} ^1V_5 = ^2V_5 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 2n+1$	1	...	...	...	$2n+1$											
4	÷	$^2V_5 \div ^2V_4$	$^1V_{11}$	$\begin{cases} ^2V_5 = ^0V_5 \\ ^2V_4 = ^0V_4 \end{cases}$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\frac{2n-1}{2n+1}$					
5	÷	$^1V_{11} \div ^1V_2$	$^2V_{11}$	$\begin{cases} ^1V_{11} = ^2V_{11} \\ ^1V_2 = ^1V_2 \end{cases}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$					
6	−	$^0V_{13} - ^2V_{11}$	$^1V_{12}$	$\begin{cases} ^2V_{11} = ^0V_{11} \\ ^0V_{13} = ^1V_{13} \end{cases}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	.....	$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$			
7	−	$^1V_3 - ^1V_1$	$^1V_{10}$	$\begin{cases} ^1V_3 = ^1V_3 \\ ^1V_1 = ^1V_1 \end{cases}$	$= n-1\,(=3)$	1	...	n	...	...	...	...	...	...	$n-1$						
8	+	$^1V_2 + ^0V_7$	$^1V_7$	$\begin{cases} ^1V_2 = ^1V_2 \\ ^0V_7 = ^1V_7 \end{cases}$	$= 2+0 = 2$	...	2	...	...	...	...	2									
9	÷	$^1V_6 \div ^1V_7$	$^2V_{11}$	$\begin{cases} ^1V_6 = ^2V_6 \\ ^0V_{11} = ^2V_{11} \end{cases}$	$= \frac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\frac{2n}{2} = A_1$					
10	×	$^1V_{21} \times ^2V_{11}$	$^1V_{12}$	$\begin{cases} ^1V_{21} = ^1V_{21} \\ ^2V_{11} = ^0V_{11} \end{cases}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$	.....	B_1		
11	+	$^1V_{12} + ^1V_{13}$	$^2V_{13}$	$\begin{cases} ^1V_{12} = ^0V_{12} \\ ^1V_{13} = ^2V_{13} \end{cases}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$			
12	−	$^1V_{10} - ^1V_1$	$^2V_{10}$	$\begin{cases} ^1V_{10} = ^2V_{10} \\ ^1V_1 = ^1V_1 \end{cases}$	$= n-2\,(=2)$	1	...	...	...	...	...	...	...	...	$n-2$						
13	−	$^1V_6 - ^1V_1$	$^2V_6$	$\begin{cases} ^1V_6 = ^2V_6 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 2n-1$	1	...	...	...	...	$2n-1$										
14	+	$^1V_1 + ^1V_2$	$^2V_7$	$\begin{cases} ^1V_1 = ^1V_1 \\ ^1V_2 = ^2V_7 \end{cases}$	$= 2+1 = 3$	1	...	...	...	...	...	3									
15	÷	$^2V_6 \div ^2V_7$	$^1V_8$	$\begin{cases} ^2V_6 = ^2V_6 \\ ^2V_7 = ^2V_7 \end{cases}$	$= \frac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\frac{2n-1}{3}$								
16	×	$^1V_8 \times ^3V_{11}$	$^4V_{11}$	$\begin{cases} ^1V_8 = ^0V_8 \\ ^3V_{11} = ^4V_{11} \end{cases}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\frac{2n}{2} \cdot \frac{2n-1}{3}$					
17	−	$^2V_6 - ^1V_1$	$^3V_6$	$\begin{cases} ^2V_6 = ^3V_6 \\ ^1V_1 = ^1V_1 \end{cases}$	$= 2n-2$	1	...	...	...	...	$2n-2$										
18	+	$^1V_1 + ^2V_7$	$^3V_7$	$\begin{cases} ^1V_1 = ^1V_1 \\ ^2V_7 = ^3V_7 \end{cases}$	$= 3+1 = 4$	1	...	...	...	...	...	4									
19	÷	$^3V_6 \div ^3V_7$	$^1V_9$	$\begin{cases} ^3V_6 = ^3V_6 \\ ^3V_7 = ^3V_7 \end{cases}$	$= \frac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\frac{2n-2}{4}$	...						
20	×	$^1V_9 \times ^4V_{11}$	$^5V_{11}$	$\begin{cases} ^1V_9 = ^0V_9 \\ ^4V_{11} = ^5V_{11} \end{cases}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_2$	...	...	...	...	...	...	...	...	0		$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3}\right\} = A_2$					
21	×	$^1V_{22} \times ^5V_{11}$	$^6V_{12}$	$\begin{cases} ^1V_{22} = ^1V_{22} \\ ^5V_{12} = ^2V_{12} \end{cases}$	$= B_2 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_2 A_2$	...	...	...	...	...	...	...	...	...	...	0	$B_2 A_2$	.....	.....	B_2	
22	+	$^2V_{12} + ^2V_{13}$	$^3V_{13}$	$\begin{cases} ^2V_{12} = ^0V_{12} \\ ^2V_{13} = ^3V_{13} \end{cases}$	$= A_0 + B_1 A_1 + B_2 A_2$	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{A_2 + B_1 A_1 + B_2 A_2\right\}$			
23	−	$^2V_{10} - ^1V_1$	$^3V_{10}$	$\begin{cases} ^2V_{10} = ^3V_{10} \\ ^1V_1 = ^1V_1 \end{cases}$	$= n-3\,(=1)$	1	...	...	...	...	...	...	...	...	$n-3$						
Here follows a repetition of Operations thirteen to twenty-three.																					
24	+	$^4V_{13} + ^0V_{24}$	$^1V_{24}$	$\begin{cases} ^4V_{13} = ^0V_{13} \\ ^0V_{24} = ^1V_{24} \end{cases}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	.....	.....	.....	.....	.....	.....	B_7
25	+	$^1V_1 + ^1V_3$	$^1V_3$	$\begin{cases} ^1V_1 = ^1V_1 \\ ^1V_3 = ^1V_3 \\ ^1V_6 = ^0V_6 \\ ^1V_7 = ^0V_7 \end{cases}$	$= n+1 = 4+1 = 5$ ..... by a Variable-card. by a Variable card.	1	...	$n+1$	...	...	0	0										

Queen Mary University of London

CS4FN

institute of CODING

EPSRC Engineering and Physical Sciences Research Council

Elf Computers

- Compile programs on to people
- Act out the computation step by step
- Why?
  - GOOD WAY TO EXPLAIN SEMANTICS
  - see work on ‘semantic waves’

Queen Mary University of London
CS4FN
institute of CODING
EPSRC Engineering and Physical Sciences Research Council

Let’s explore an example unplugged role play dry run activity:

Box Variables

Queen Mary University of London

CS4FN

institute of CODING

EPSRC
Engineering and Physical Sciences Research Council

We are going to explore what a variable is and how they work

You will be able to:

- explain what a variable is
- work out what simple programs that manipulate variables do, step by step

Queen Mary University of London
CS4FN
institute of CODING
EPSRC Engineering and Physical Sciences Research Council

What is a variable?

- Variables are like storage boxes
- Special ones that can
  - Store
  - Create
  - Destroy

Queen Mary University of London
CS4FN
institute of CODING
EPSRC Engineering and Physical Sciences Research Council

What does this (python) code do?

```
colour1 = “red”
colour2 = “green”

temp = colour1
colour1 = colour2
colour2 = temp
```

Let’s act it out!

Queen Mary University of London

CS4FN

institute of CODING

EPSRC
Engineering and Physical Sciences Research Council

Exercise

Write down as many facts about variables that we have just come across as you can

Main points

- A Variable is a storage space that holds data for later use in a program
- Variables have names
  - so the computer knows which one is meant.
- Variables hold values
  - the actual data that is being stored.
- Do not confuse variable names with their values!

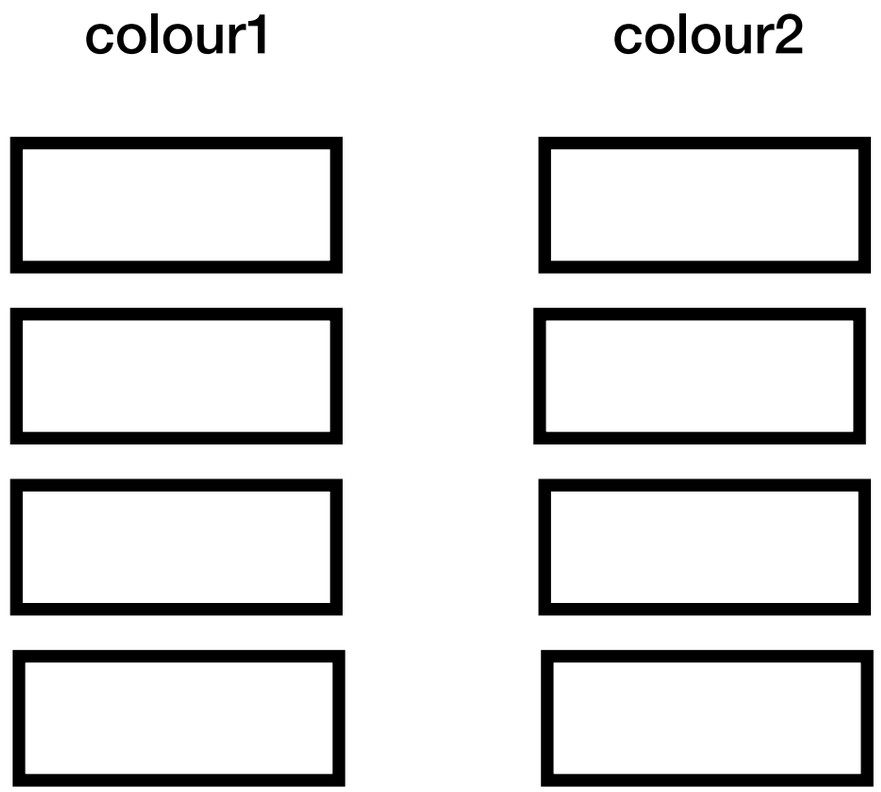
- A variable can only store one value at a time.
- When accessing a variable's value you make a copy
  - that variable's value is untouched.
- When you store a value in a variable you destroy anything that was previously there.

Towards Dry Run Tables
What does this program do?

Draw a series of pictures of boxes to find out…

colour1 = “red”
colour2 = “green”

colour1 = colour2
colour2 = colour1



Queen Mary University of London
CS4FN
institute of CODING
EPSRC
Engineering and Physical Sciences Research Council

Dry Run Tables: Example

- Let's draw a table to see what this does:

```
a = 1
b = 2

b = a
a = b
```

Dry Run Tables: Example

a = 1

b = 2

b = a

a = b

a
1

Dry Run Tables: Example

a = 1

b = 2

b = a

a = b

a	b
1	2

Dry Run Tables: Example

a = 1

b = 2

b = a

a = b

a	b
1	2
	1

Dry Run Tables: Example

a = 1

b = 2

b = a

a = b

a	b
~~1~~	~~2~~
	1
1	

Your turn

- What does this code do?
- Fill in the this dry run table…

colour1 = “red”
colour2 = “green”

colour1 = colour2
colour2 = colour1

colour1	colour2

Dry Run tables

- This leads directly to the idea of dry run tables
- Combine the boxes into a table

colour1 = “red”
colour2 = “green”

colour1 = colour2
colour2 = colour1

colour1	colour2
~~“red”~~	
	~~“green”~~
“green”	
	“green”

Diagnostic Tests

- A powerful way to detect (and correct) misconceptions early
- Set weekly dry run tests of (eg 5-12) short, critical dry runs
- Require table solutions
- Mark immediately
- Fix the mental models of those making mistakes

Show all your working in the following. Justify your answers!

1. What are the new values of a and b after the following code fragment has executed?

```
int  a = 10;
int  b = 20;
a = b;
```

a is now _____ **b is now _____**

2. What are the new values of a and b after the following code fragment has executed?

```
int  a = 10;
int  b = 20;
b = a;
```

a is now _____ **b is now _____**

3. What are the new values of big and small after the following code fragment has executed?

```
int  big  =  10;
int  small = 20;
big = small;
```

big is now _____ **small is now _____**

Queen Mary University of London

CS4FN

institute of CODING

EPSRC Engineering and Physical Sciences Research Council

Dry Run tables are important

- Drawing tables matters (educationally)
  - Do NOT let students do it in their head.
  - Once you can program doing this is rare
- BUT **pedagogically** it is a critical activity
- Programmers reason at a higher level of abstraction most of the time
  - writing tables gives a foundation for that

If statements, loops etc

- The ideas extend to loops and if statements
- When dry running:
  - **add an extra column for any test**
- Compile programs on to ‘Imp Computers’
  - people represent instructions,
  - linked together by rope to represent the control structure

If statements, loops etc

```
a = 2
if a==3:
    b = 5
else
    b = 7
```

a	b	a==3?
2		
		2==3? false
	7	

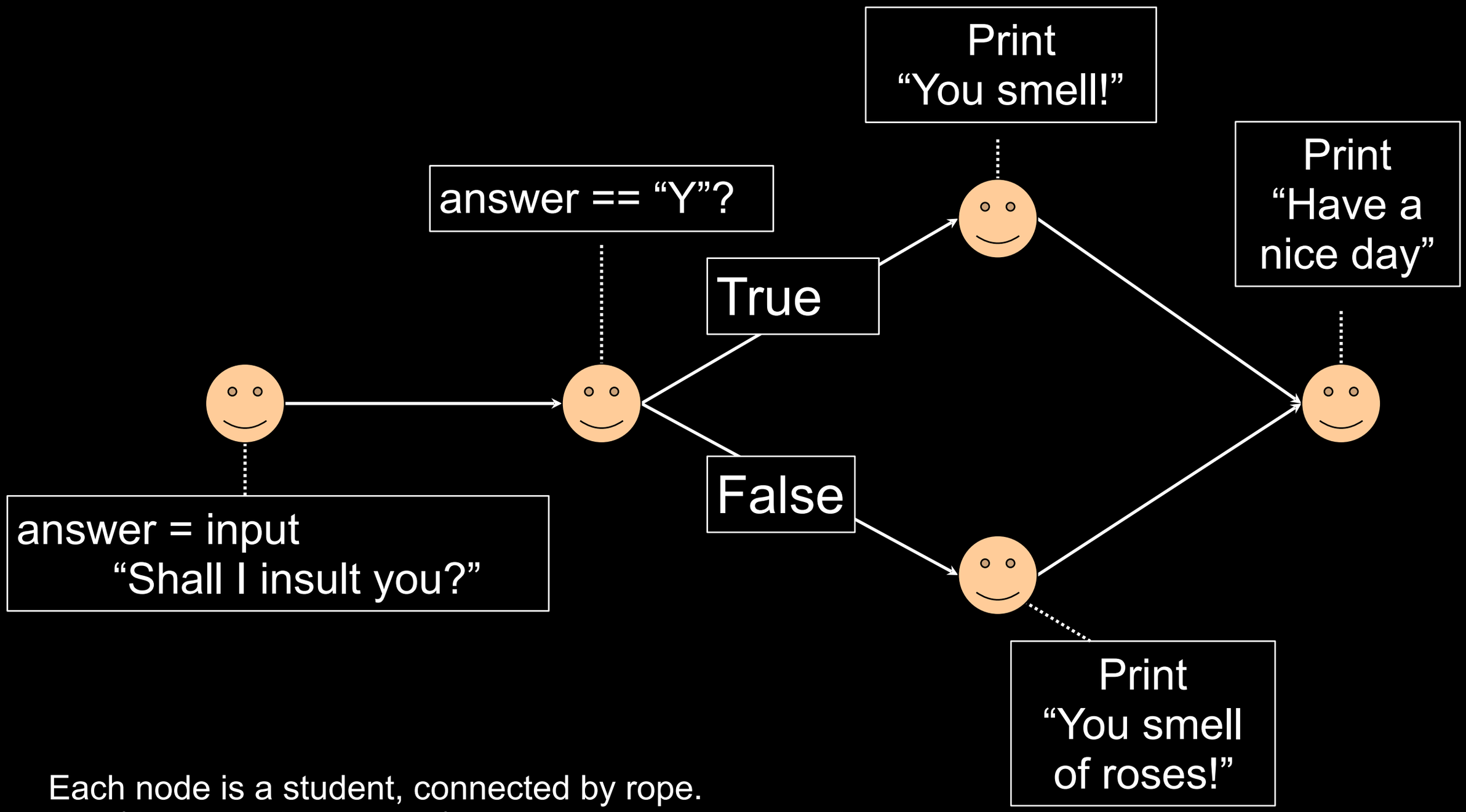
An Insulting Program

```
answer = input (“Shall I insult you?”)

if answer == “Y”:
        print (“You smell!”)
else:
        print (“You smell of roses!”)

print (“Have a nice day”)
```

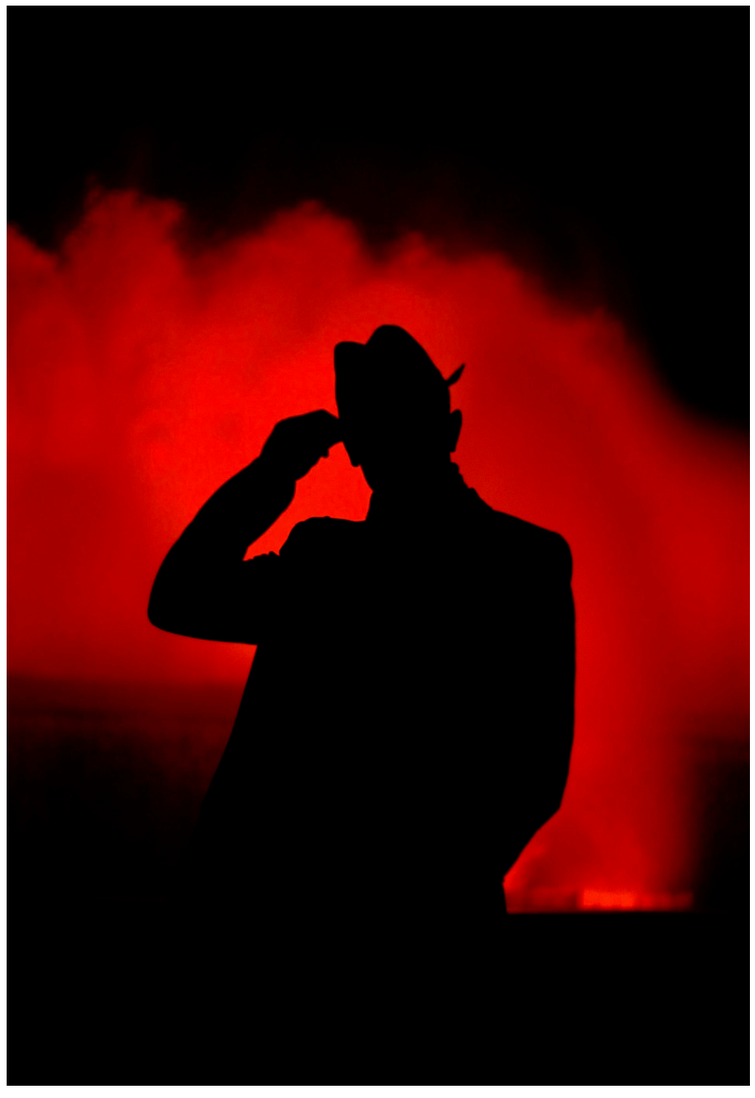
What it compiles to …



Each node is a student, connected by rope.
You (the operating system) pass in a tube at one end.
When it is handed back something should be printed on the screen (written on the board)

Explaining programming concepts

- We've made an invisible program tangible
- Can now explain all sorts of concepts
  - If statements
  - Control structures
  - The program counter
  - Run-time versus compile time
- Similar approach for loops



Queen Mary University of London

CS4FN

institute of CODING

EPSRC Engineering and Physical Sciences Research Council

More advanced concepts

- Can also explore more advanced ideas
  - How program changes change structure
  - Optimizing compilers
  - Bugs

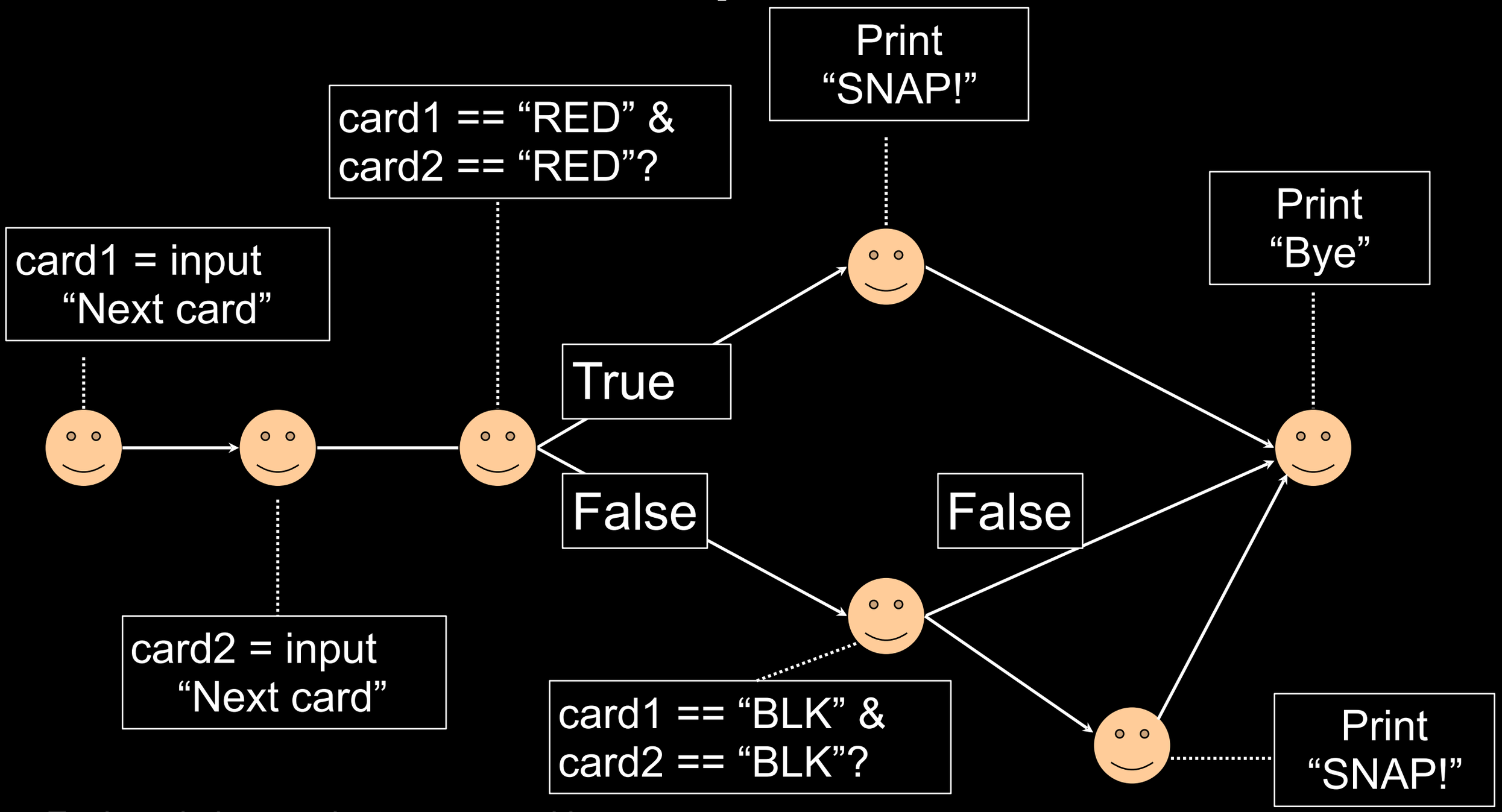
A Snap Program

```
card1 = input (“Next card”)
card2 = input (“Next card”)

if card1 == “RED” and card2 == “RED”:
        print (“SNAP!”)
else:
        if card1 == “BLACK” and card2 == “BLACK”:
                print (“SNAP!”)

print (“Bye”)
```

What it compiles to …

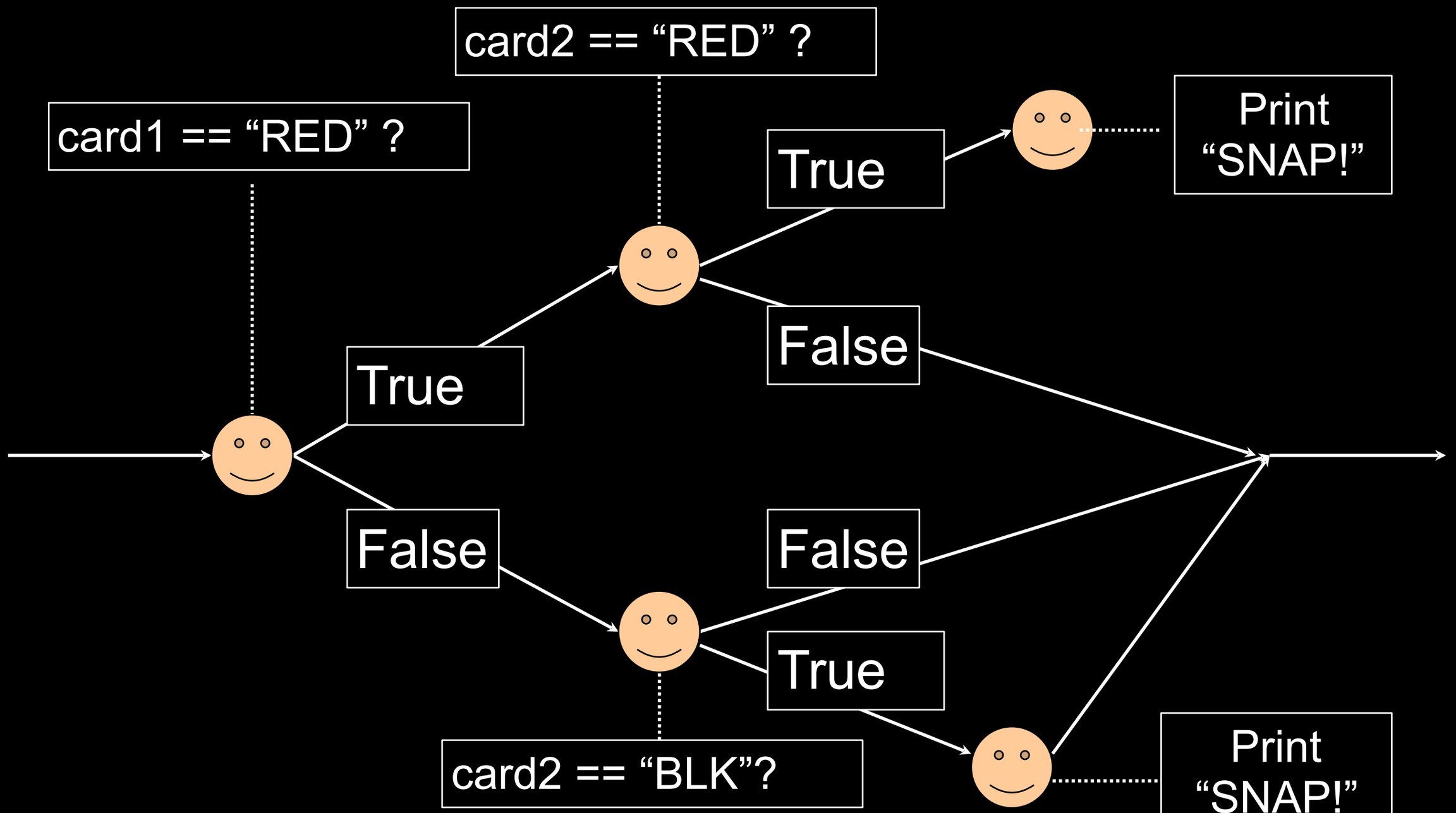


Each node is a student, connected by rope.
You (the operating system) pass in a tube at one end.
When it is handed back something should be printed on the screen (written on the board)

Exploring Different Programs

```
if card1 == “RED”:
    if card2 == “RED”:
        print (“SNAP!”)
else:
    if card2 == BLACK:
        print (“SNAP!”)
```

What it compiles to …



Each node is a student, connected by rope.
You (the operating system) pass in a tube at one end.
When it is handed back something should be printed on the screen (written on the board)

Summary

- It is understanding semantics that matters when learning to program
- Unplugged Role play (‘Imp computers’) is a powerful first step
- Making students draw dry run tables is the next vital learning activity

Queen Mary University of London
CS4FN
institute of CODING
EPSRC Engineering and Physical Sciences Research Council

Why are these activities vital?

- Help in understanding **what constructs do** not just rote learning how to write them
- Develops **logical thinking**
- Develops **attention to detail**
- Allows teacher to quickly **identify misconceptions** and faulty mental models
- Helps students over **barrier concepts**
- **Transition** step between explanations and programs

Queen Mary University of London
CS4FN
institute of CODING
EPSRC Engineering and Physical Sciences Research Council

Thank You

Resources at:

teachinglondoncomputing.org
Twitter: @cs4fn

Queen Mary University of London

CS4FN

institute of CODING

EPSRC
Engineering and Physical Sciences Research Council