COMP 1010- Summer 2015 (A01)

Jim (James) Young

young@cs.umanitoba.ca

jimyoung.ca

Hello!

James (Jim) Young
young@cs.umanitoba.ca
jimyoung.ca

office hours T / Th: 17:00 – 18:00
EITC-E2-582
(or by appointment, arrange by email)

boolean logic nesting

boolean result = ! (c || !(a<b)); // this is legal

Assuming that this runs, what do you know about the data types of the variables a, b, and c?

a and b must be numerical because we are doing a less-than operator.

c must be boolean because it is an operand to the boolean OR operator.

what is the result if a=3, b=1, and c=true; ?

boolean logic nesting

boolean result = ! (c || !(a<b)); // this is legal

what is the result if a=3, b=1, and c=true; ?

! (c || !(a<b))

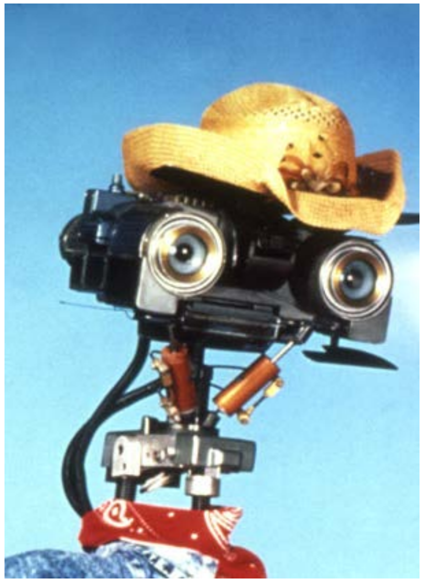
! (T || !(3<1))

! (T || !(F))

! (T || T)

! (T)

F



If you know that c is true, does it matter what a and b are?

No! T || anything is TRUE

Your computer may save work here

If statements and code blocks

alternate if syntax

We learned if statements with blocks, but blocks are not necessary

note: the curly braces {} are not necessary for **if-then-else**, or loops, but if you don’t use them, you are limited to ONE line of code each

```
if (boolean test)
{
…
} else {
…
}
```

```
if (boolean test)
    oneLineCommand;
else
    oneLineCommand:
```

easy to make problems…

```
if (shouldMugJim)
    putOnBlackMask();
    practiceToughVoice();
    mugJim();
continueProgram();
```

Only the first command (putOnBlackMask()) is related to the if statement!! everything else will run anyway. **Indenting is aesthetics only**

Get into the habit of always using {}, its safer.

using {} avoids the risk

```
if (shouldMugJim)
{
    putOnBlackMask();
    practiceToughVoice();
    mugJim();
}
continueProgram();
```

you can mix blocks with no blocks..

```
if (booleanStatement) {
        commandA1;
        commandA2;
} else if (booleanStatement)
        command B;
else
        command C
```

NOTE: always ensure to match opening { with closing } to end the block properly.. otherwise, doesn’t know where your block ends!!

Advice: just use code blocks everywhere

broken if-else chain example

```
if (temperature < -50 || temperature > 50) {  // invalid temperature
    severity = -1;

  } else { // temperature is valid

    if (temperature < -20)
      severity = 4;
    } else if (temperature < -10) {
      severity = 3;
    } else if (temperature < 0) {
      severity = 2;
    } else
      severity = 1;

  }
```

despite indentation, there is a mismatch of { and } such that Processing gets the blocks confused.

How to avoid?

ALWAYS USE CODE BLOCKS!

aside: more globals

width

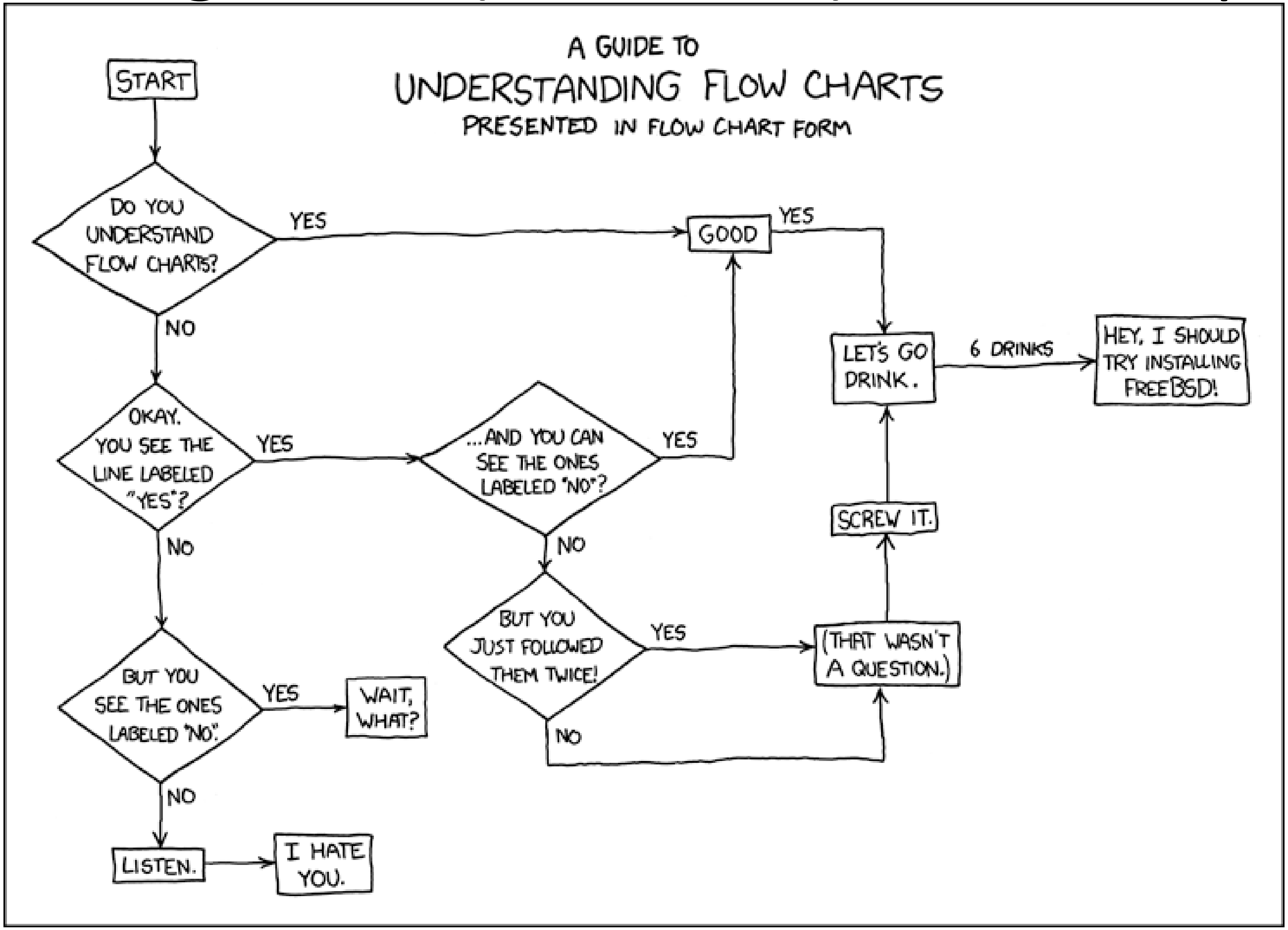
height

Carful! Only use those AFTER you set the canvas size.

LOOPS!!!!

sometimes, in a program you want to repeat an operation a bunch of times.

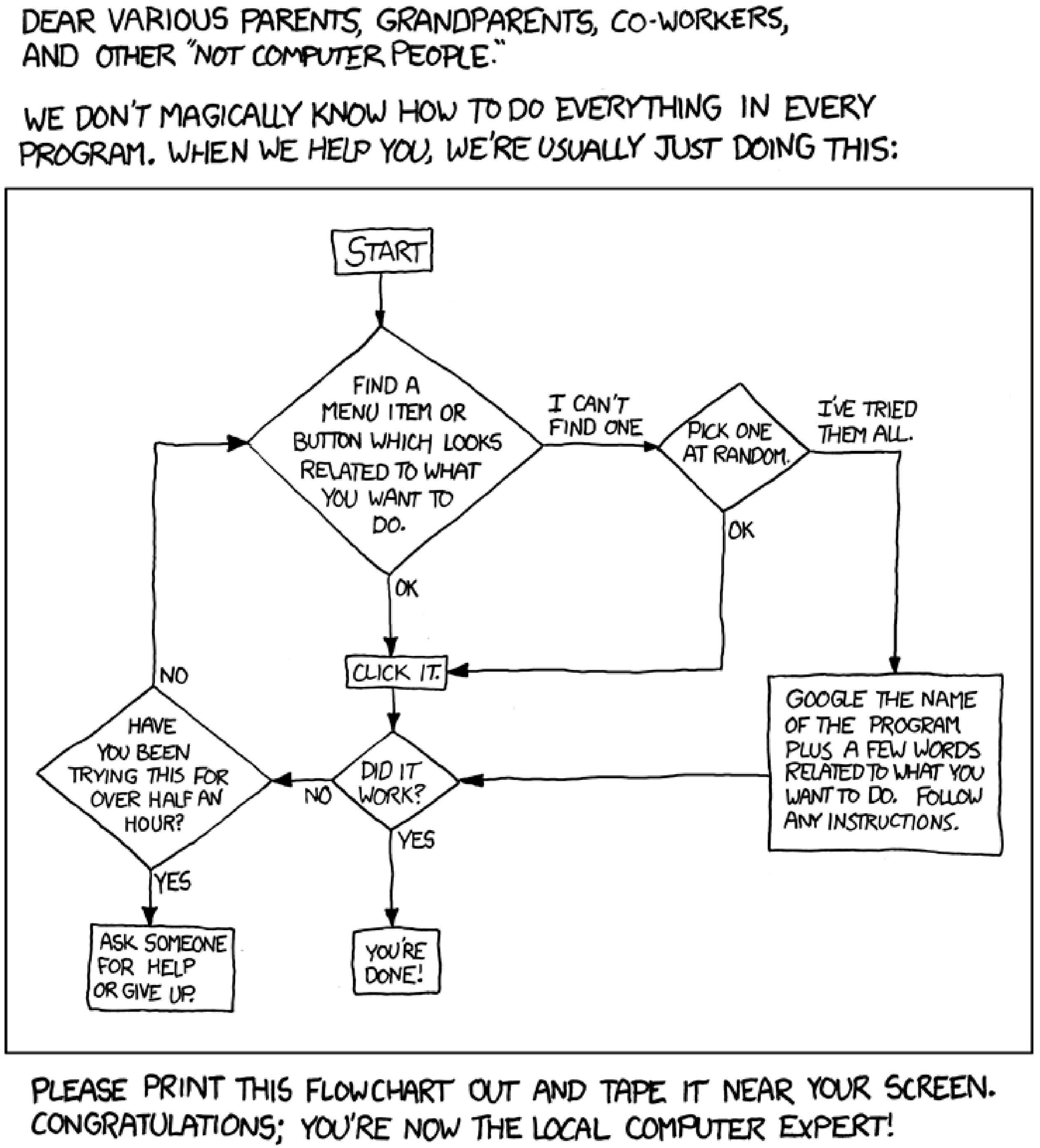
this algorithm (flow chart) has no loops


A GUIDE TO
UNDERSTANDING FLOW CHARTS
PRESENTED IN FLOW CHART FORM
START
DO YOU UNDERSTAND FLOW CHARTS?
YES
GOOD
YES
NO
OKAY. YOU SEE THE LINE LABELED "YES"?
YES
...AND YOU CAN SEE THE ONES LABELED "NO"?
YES
LET'S GO DRINK.
6 DRINKS
HEY, I SHOULD TRY INSTALLING FREEBSD!
NO
NO
BUT YOU SEE THE ONES LABELED "NO".
YES
WAIT, WHAT?
NO
LISTEN.
I HATE YOU.
BUT YOU JUST FOLLOWED THEM TWICE!
YES
(THAT WASN'T A QUESTION.)
NO
SCREW IT.

XKCD

LOOPS!

repeat until some condition is met, like doing for more than 30 minutes or you solved the problem


DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS, AND OTHER "NOT COMPUTER PEOPLE."
WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:
START
FIND A MENU ITEM OR BUTTON WHICH LOOKS RELATED TO WHAT YOU WANT TO DO.
I CAN'T FIND ONE
PICK ONE AT RANDOM.
I'VE TRIED THEM ALL.
OK
OK
CLICK IT.
GOOGLE THE NAME OF THE PROGRAM PLUS A FEW WORDS RELATED TO WHAT YOU WANT TO DO. FOLLOW ANY INSTRUCTIONS.
NO
HAVE YOU BEEN TRYING THIS FOR OVER HALF AN HOUR?
NO
DID IT WORK?
YES
YES
ASK SOMEONE FOR HELP OR GIVE UP.
YOU'RE DONE!
PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN. CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

for loops!

this is tough

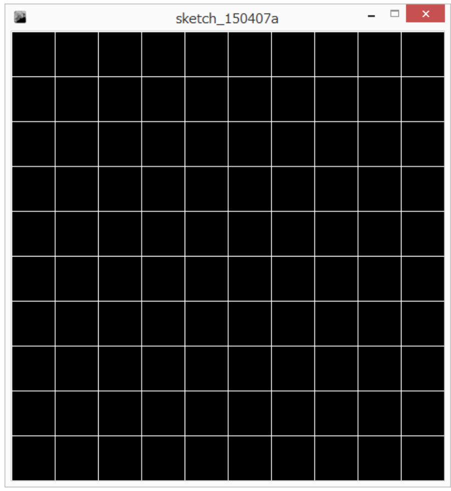
Let’s make a grid on the screen

Setup variables: grid size, cell size

Start making your horizontal lines

A lot of work!!!

What if we wanted a grid of 100?



for loop

wouldn’t it be nice if we could tell processing…

for each value of the variable **i** in the range from **small** to **large** do the following…

e.g., for each value of an integer i from 0 to 10, do our line.

(this is **pseudo-code**, fake code!)

```
for   0 <= i < 10:
        draw our line at grid position i
```

for loop – general line formula

```
line(0, 0, width-1, 0); // top line
line(0, cellSizeY*1, width-1, cellSizeY*1);
line(0, cellSizeY*2, width-1, cellSizeY*2);
```

(this is **pseudo-code**, fake code!)

```
for   0 <= i < 10:
        line(0, cellSizeY*i, width-1, cellSizeY*i);
```

(not correct) kind of do it with an if

```
for   0 <= i < 10:
        line(0, cellSizeY*i, width-1, cellSizeY*i);

int i=0;  // declare our counter, start the counter at 0
if (i < 10)
{
        line(0, cellSizeY*i, width-1, cellSizeY*i);
        i++;    // increment i
} < go back to the if and do it again >
```

With this, the loop runs 10 times, from 0..9

Key pieces

```
int i=0;                                        initialization priming (start)
if (i < 10)                                                 condition test (end)
{
        line(0, cellSizeY*i, width-1, cellSizeY*i);                      (body)
        i++;    // increment i                                  upkeep (step)
        // go back to the if and do it again
}
```

These pieces, the **initialization, condition, body,** and **upkeep**, generally exist in loops that iterate over a series of values.

the processing for loop syntax!

for (**initialization; condition; upkeep**)

{

        body;

}

read this as:

        for the start condition **initialization**, while **condition** is true, loop the **body** and do the **upkeep** at the end

the Java for loop syntax!

```
for (initialization; condition; upkeep)
{
        body;
}
```

1,2,3,4,

2,3,4

2,3,4

```
         1          2        4
for (int i = 0; i < 10; i++)
{
            3
          line(0, cellSizeY*i, width-1, cellSizeY*i);
}
```

read this as:

for the start condition **i=0**, while **i<10**, loop the **body** and use the upkeep i++

for loop

```
for (initialization; condition; upkeep) {
        body;
}
```

initialization:

done only ONCE!!!

condition:

checked **BEFORE** each loop through

upkeep:

done at the END of each time through the loop

body:

executed each time through the loop

super common for loop usage:

declare the iterator variable in the initializer:

```
for (int i = 0; i < 999; i++) {
        do some stuff;
}
println(i);
```

.... but there’s a problem here. what is it?

for loop and scope

note: just as with any **code block**, variables created within a for loop code block can only be accessed within that block. Once the block is finished, the variable is destroyed and no longer accessible

```
for (int counter=0; counter < 200; counter++) {
        // some code!
}
println(counter); // error!!
```

the counter variable only exists within the scope of the for loop.

it is not visible outside the scope of the for loop.

for loop and scope

```
for (int i=0; i < 200; i++) {
        // some code!
}
for (int i=0; i < 300; i++) {
        // some code!
}
```

note: this is valid because the counter variable exists only within the for loop scope. When the second for loop is created, it cannot see the previous counter variable, so there is no problem. It creates a new one.

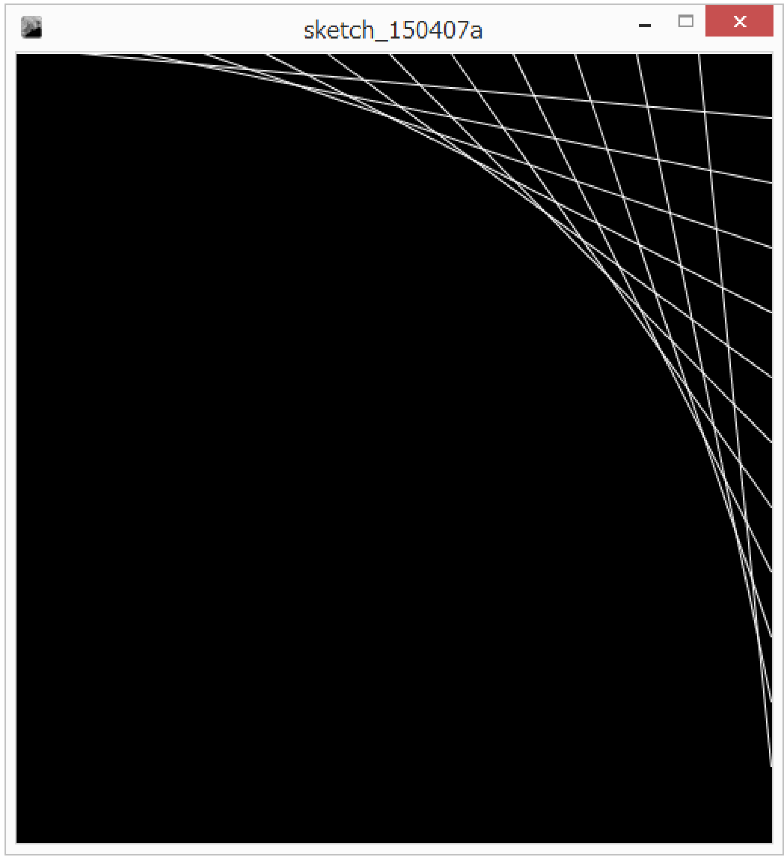
Example: line art

Define how many points and calculate spacing

Hard-code 2-3 lines

Make general case

Put inside for loop


sketch_150407a