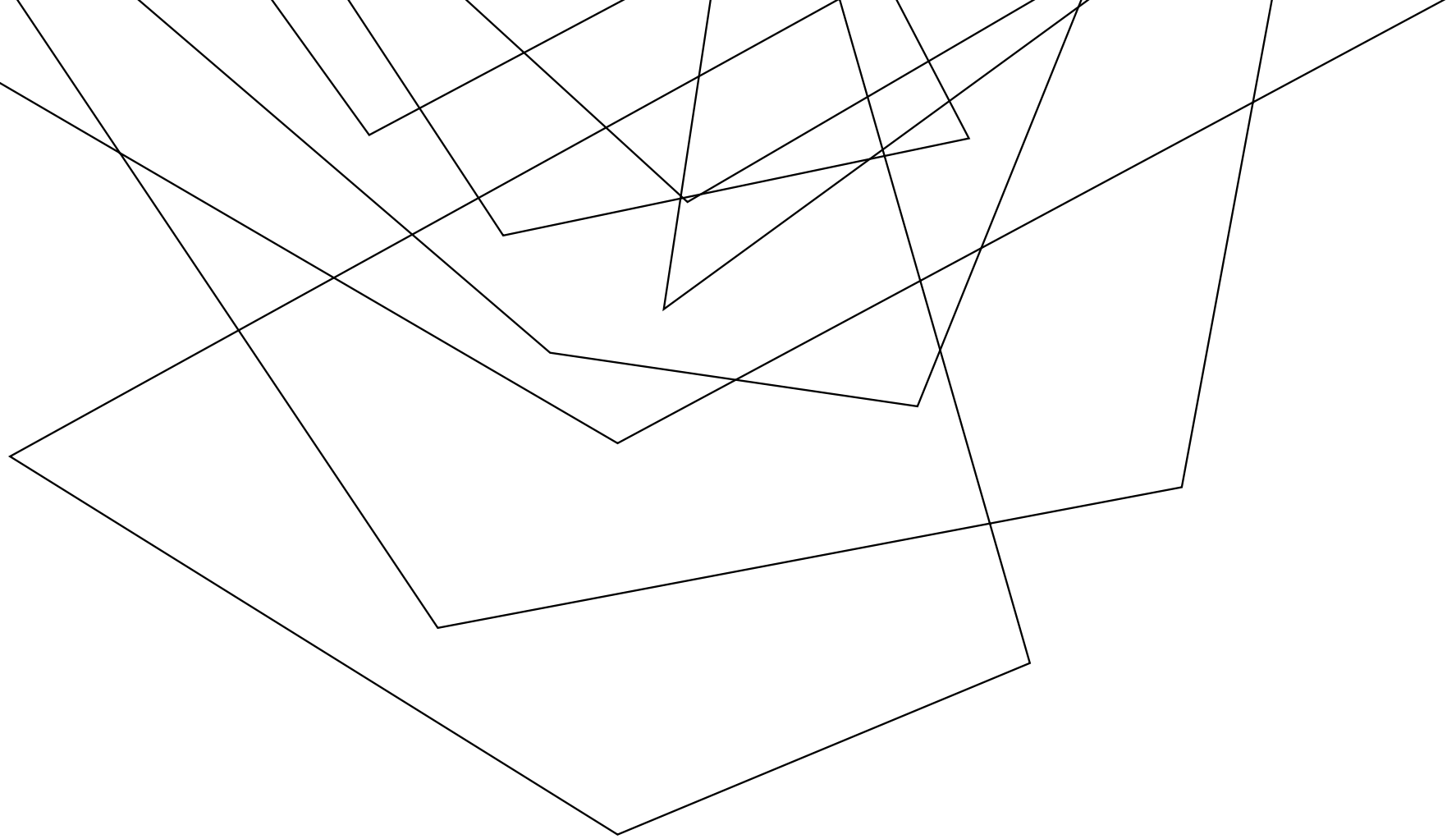
EXERCISE #2

OVERVIEW REVIEW

Write your name and answer the following on a piece of paper

- The companion to static analysis (analysis without running the target program) is *dynamic* analysis (analysis that includes running the target program). Give an example of a dynamic analysis.



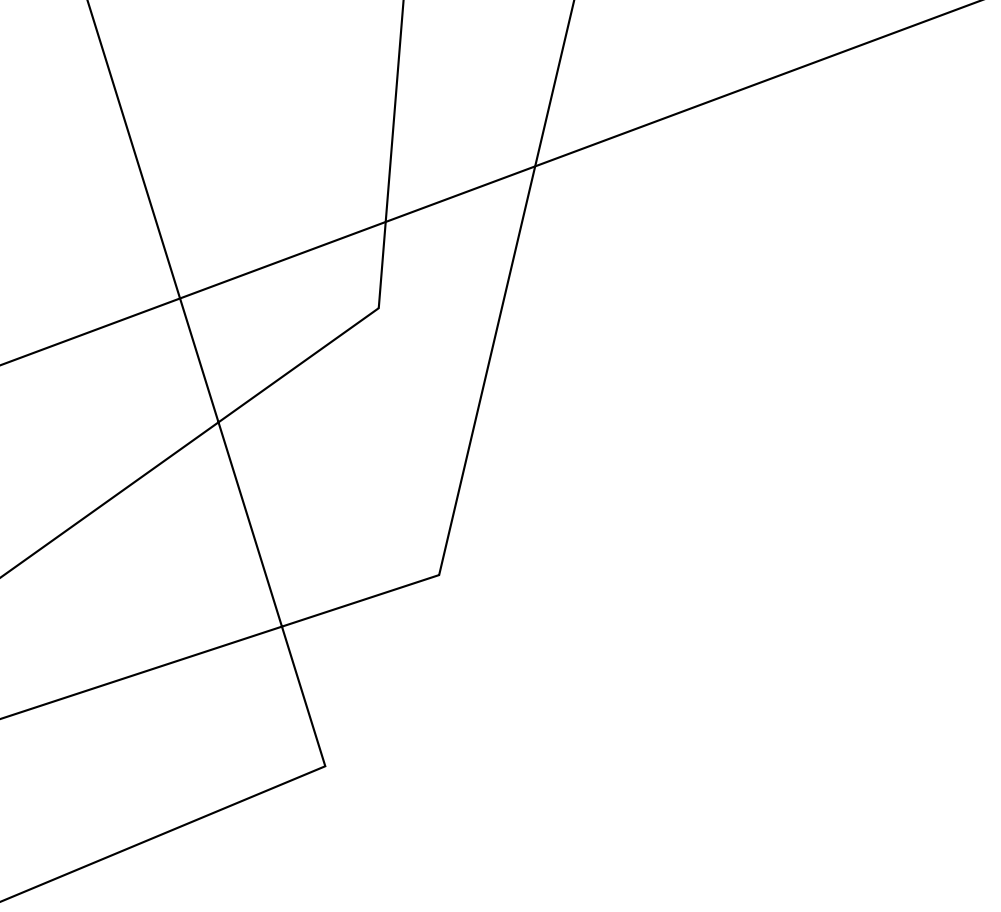
ABSTRACTING CODE

EECS 677: Software Security Evaluation

Drew Davidson

ADMINISTRIVIA AND ANNOUNCEMENTS

- **Miss a class? It's not too late to get points for the check-in assignment!**
- **The Entry Survey: results and thoughts**

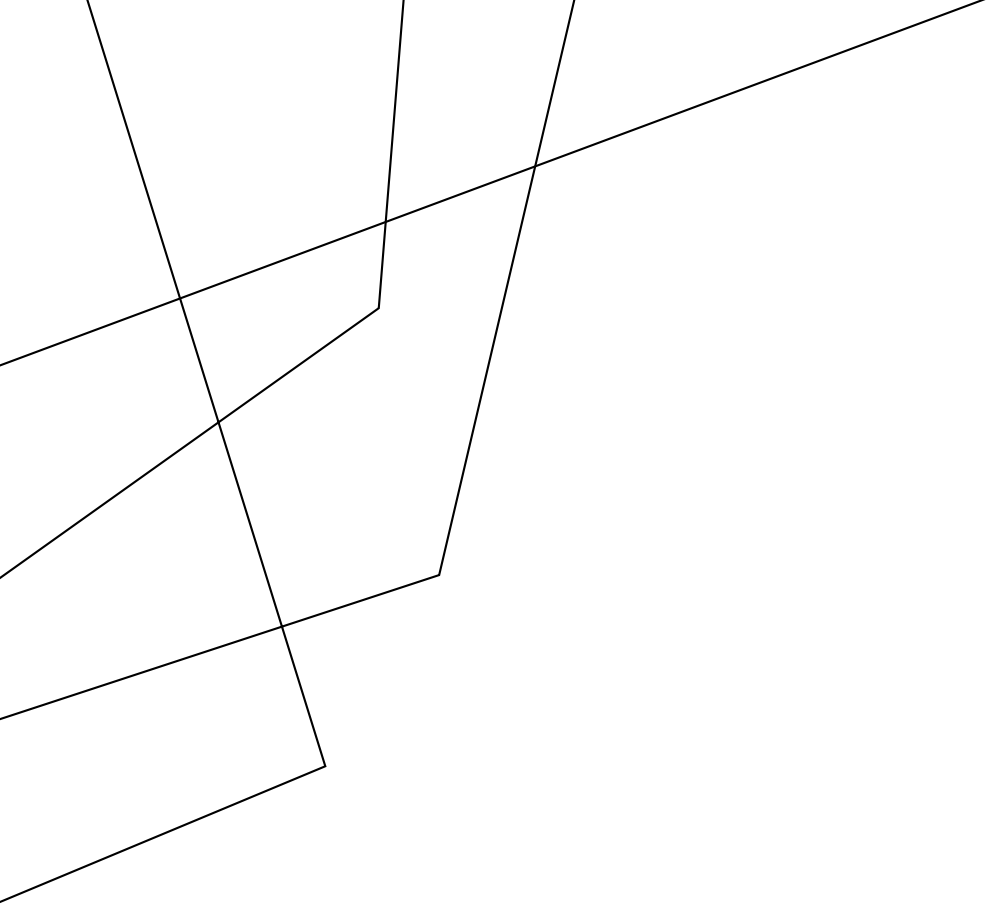


ADMINISTRIVIA AND ANNOUNCEMENTS

“Please record lectures”

ADMINISTRIVIA AND ANNOUNCEMENTS

“My given name is <X> but I go by <Y>”



ADMINISTRIVIA AND ANNOUNCEMENTS

Lots of interest in learning about vulnerabilities

ADMINISTRIVIA AND ANNOUNCEMENTS

Some concern about workload

ADMINISTRIVIA AND ANNOUNCEMENTS

How long are the quizzes?

LAST TIME: OVERVIEW

REVIEW: OVERVIEW

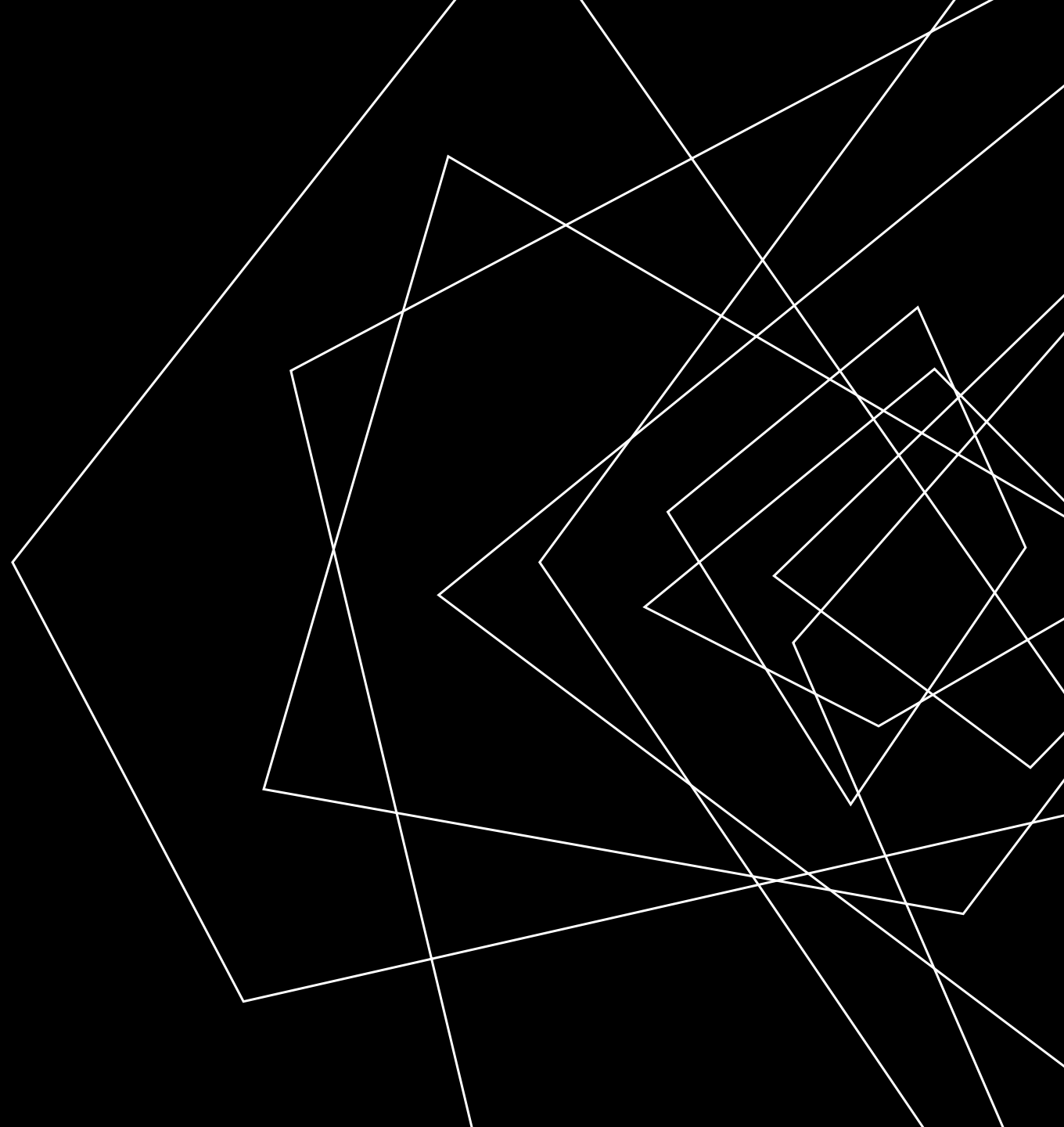
Discussed the insufficiency of manual source code analysis for security evaluation.

Described the need to deal with abstractions of software. These abstractions can do two things:

- Emphasize some under-appreciated aspect of the target program
- Simplify or ease a form of reasoning about the target program

LECTURE OUTLINE

- Instruction Flowcharts
- Control Flow Graphs

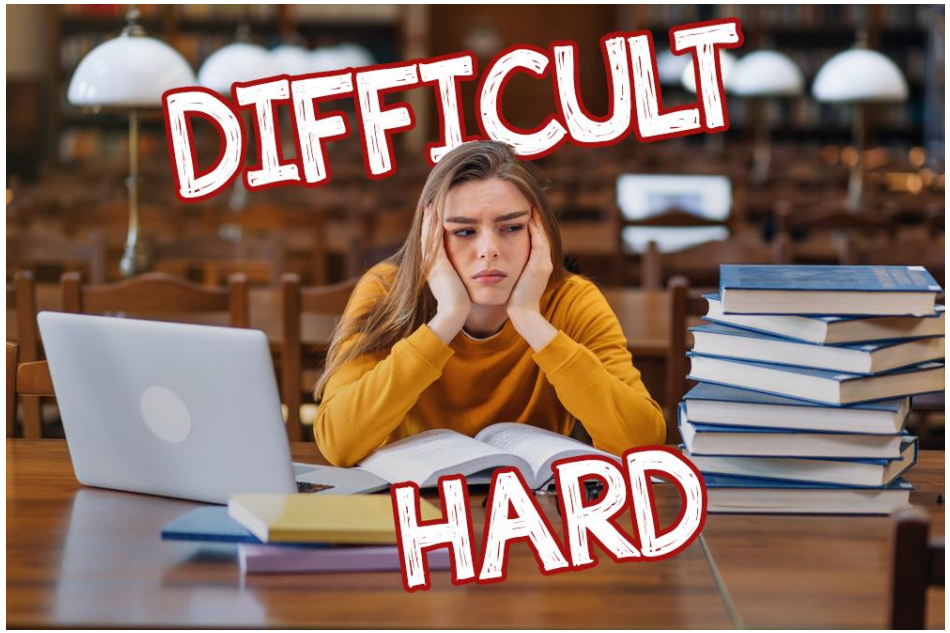


VISUALIZING PROGRAMS

INSTRUCTION FLOWCHARTS

Reading code is hard!

- It's really important to determine how code *flows* from one instruction to the next



CODE GRAPHS
INSTRUCTION FLOWCHARTS

Program analysis relies heavily on two questions

- (How) can we get to a particular program point?
- What is the program configuration at a given point?

Helpful to structure program instructions as a graph

- Visualize transfer of control
- Avail ourselves of graph analyses (e.g. reachabilty)

FLOWCHARTS

ABSTRACTING CODE: INSTRUCTION FLOWCHARTS

NOTATION

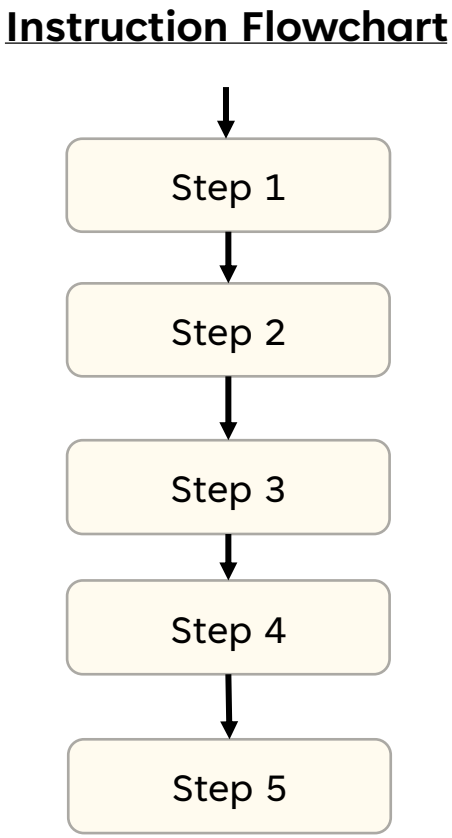
NODES ARE INSTRUCTIONS

EDGES GO TO SUCCESSOR NODE

OPERATION

EXECUTE CURRENT INSTRUCTION

PROCEED TO SUCCESSOR NODE



FLOWCHART EXAMPLE – HOW TO FLOSS

ABSTRACTING CODE: INSTRUCTION FLOWCHARTS

NOTATION

NODES ARE INSTRUCTIONS

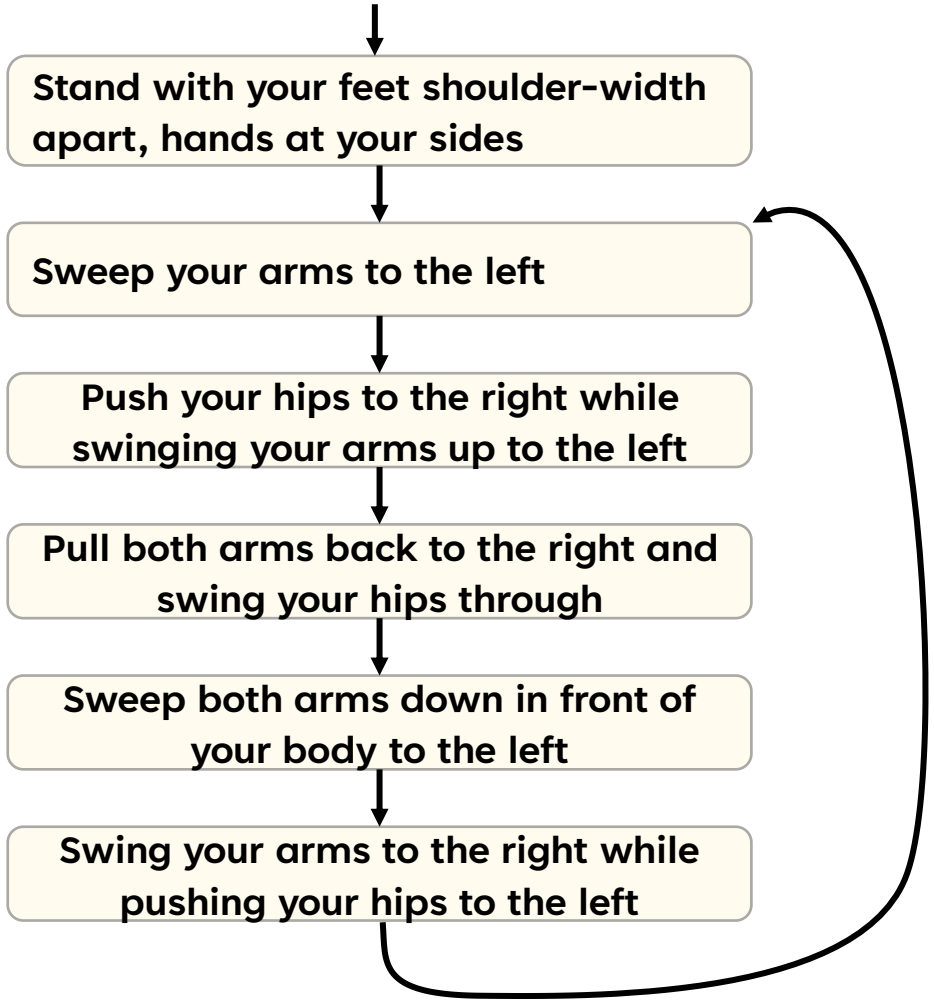
EDGES GO TO SUCCESSOR NODE

OPERATION

EXECUTE CURRENT INSTRUCTION

PROCEED TO SUCCESSOR NODE

Repetition!



FLOWCHARTS – CONDITIONALS

ABSTRACTING CODE: INSTRUCTION FLOWCHARTS

NOTATION

NODES ARE INSTRUCTIONS

EDGES GO TO SUCCESSOR NODES

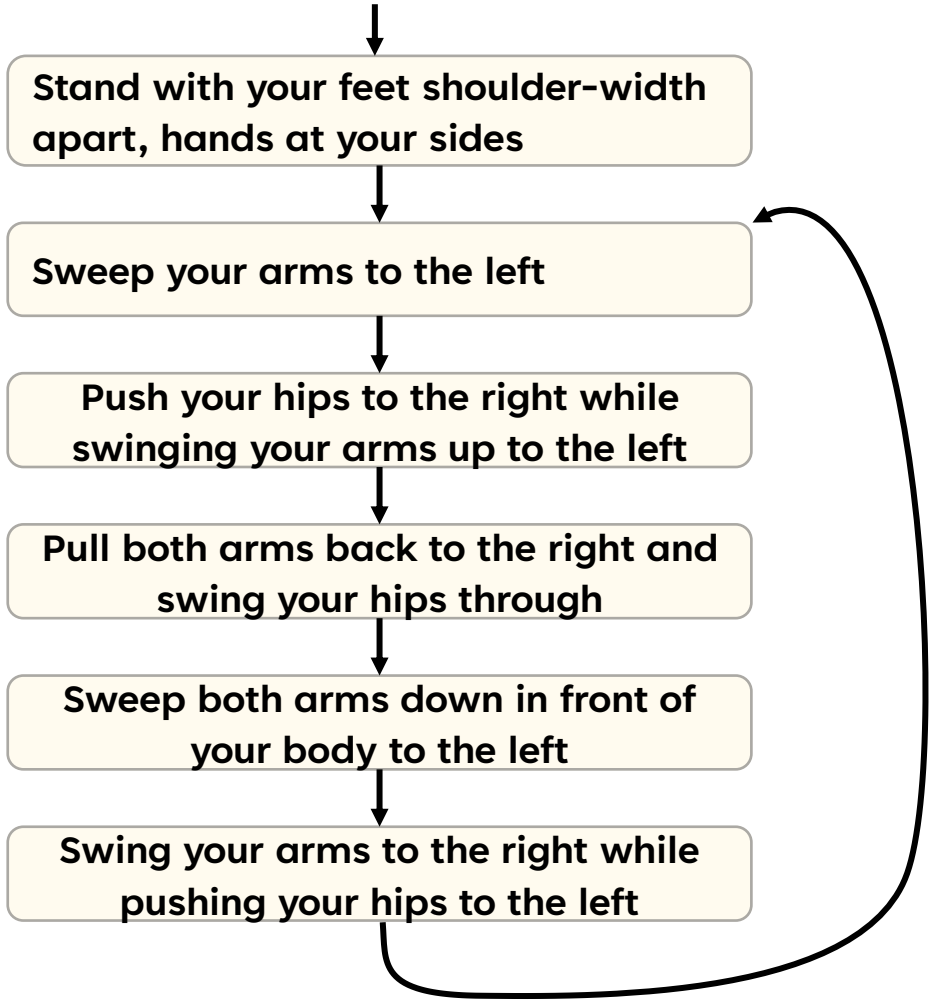
(DISAMBIGUATED WITH CONDITIONS)

OPERATION

EXECUTE CURRENT INSTRUCTION

PROCEED TO SUCCESSOR NODE

(ACCORDING TO CONDITION)



FLOWCHARTS – CONDITIONALS

ABSTRACTING CODE: INSTRUCTION FLOWCHARTS

NOTATION

NODES ARE INSTRUCTIONS

EDGES GO TO SUCCESSOR NODES

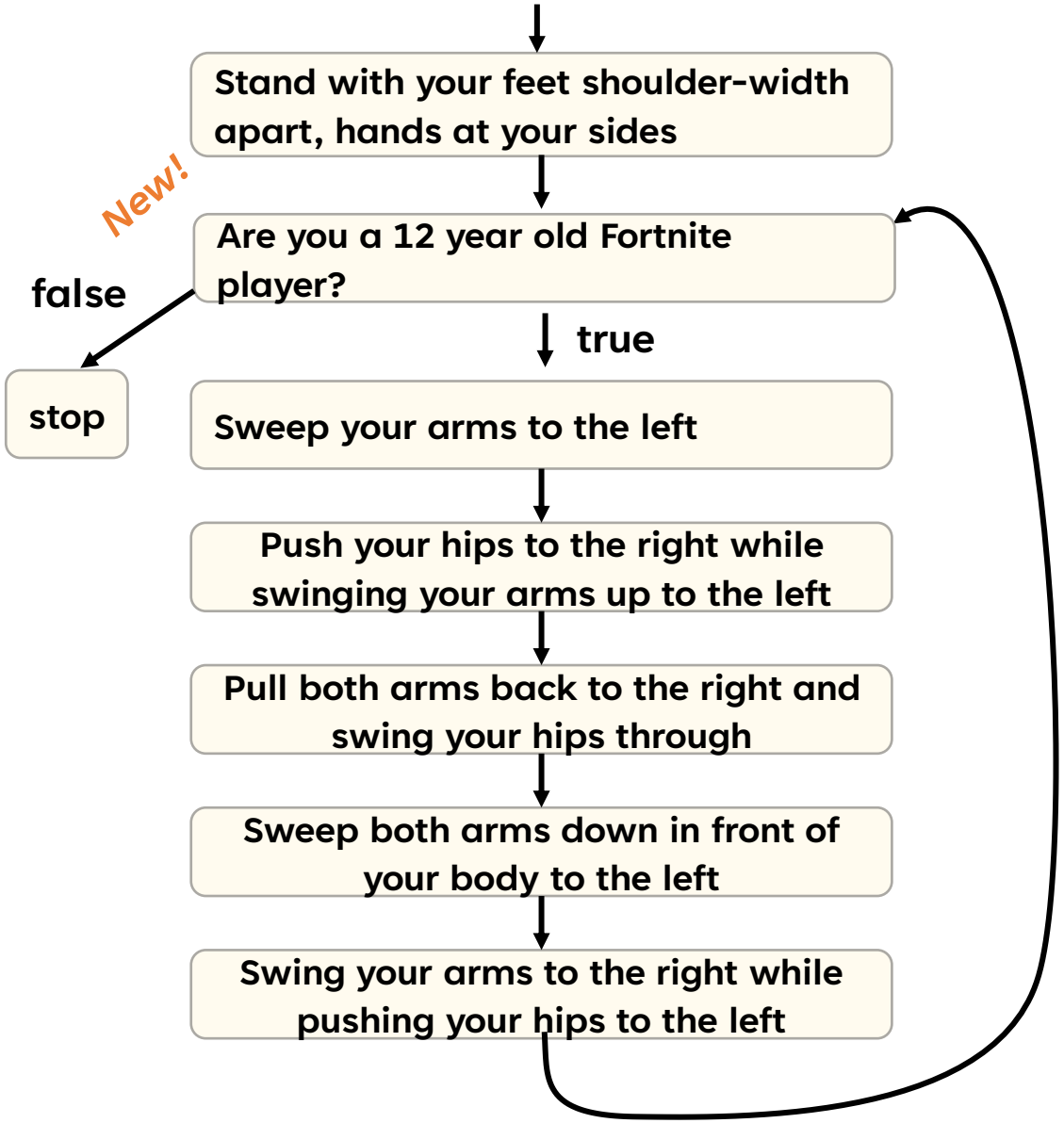
(DISAMBIGUATED WITH CONDITIONS)

OPERATION

EXECUTE CURRENT INSTRUCTION

PROCEED TO SUCCESSOR NODE

(ACCORDING TO CONDITION)



FLOWCHARTS – FOR CODE?!?!?!!

ABSTRACTING CODE: INSTRUCTION FLOWCHARTS

NOTATION

NODES ARE INSTRUCTIONS

EDGES GO TO SUCCESSOR NODES

(DISAMBIGUATED WITH CONDITIONS)

OPERATION

EXECUTE CURRENT INSTRUCTION

PROCEED TO SUCCESSOR NODE

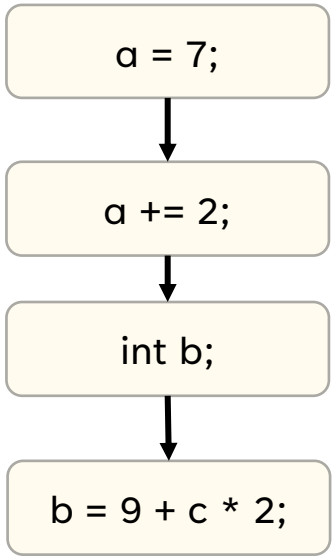
(ACCORDING TO CONDITION)

Source Code

```
a = 7;
a += 2;
int b;
b = 9 + c * 2;
```

Should this be just 1 instruction?

Instruction Flowchart



CODE FLOWCHARTS

INSTRUCTION FLOWCHARTS

NOTATION

NODES ARE INSTRUCTIONS

EDGES GO TO SUCCESSOR NODES UNDER APPROPRIATE CONDITION

OPERATION

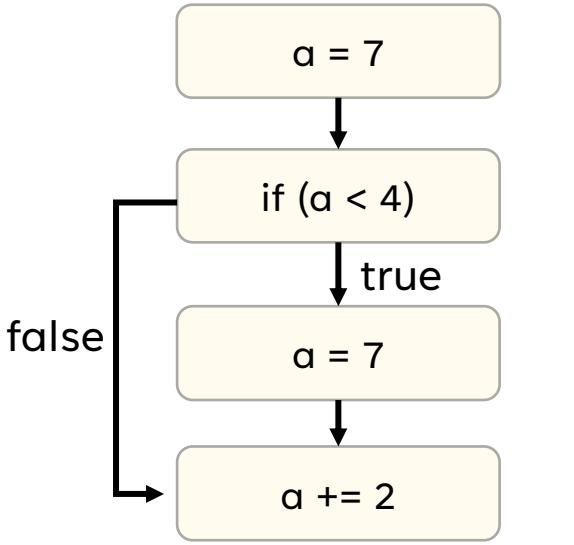
EXECUTE CURRENT INSTRUCTION

PROCEED TO THE RIGHT SUCCESSOR

source code

```
a = 7;
if (a < 4){
  a = 7;
}
a += 2;
```

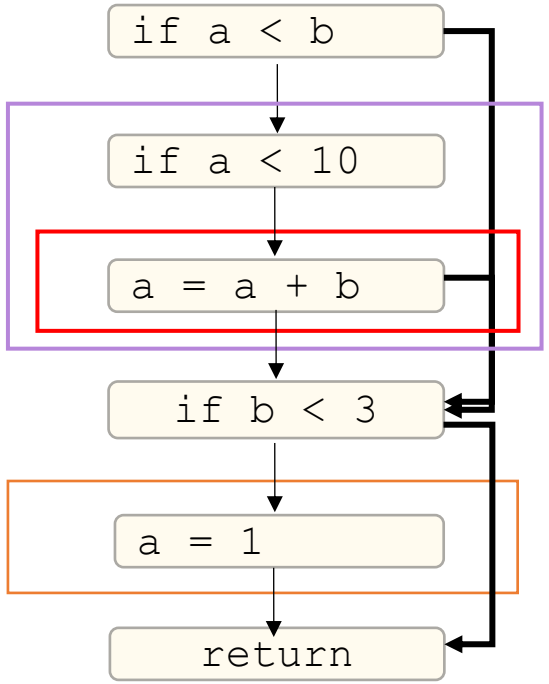
Instruction Flowgraph



FLOWCHARTS: VISUALIZING CONTROL

ABSTRACTING CODE: INSTRUCTION FLOWCHARTS

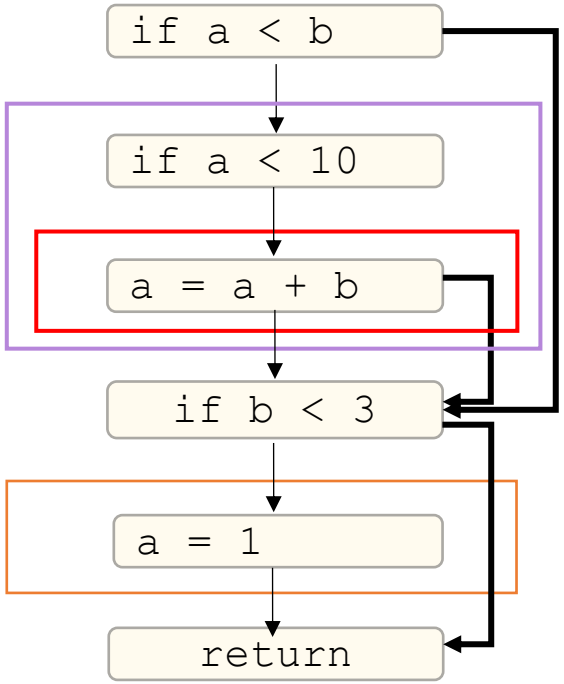
```
void funk(int a, int b){
    if (a < b){
        if (a < 10){
            a = a + b;
        }
    }
    if (b < 3){
        a = 1;
    }
    return;
}
```

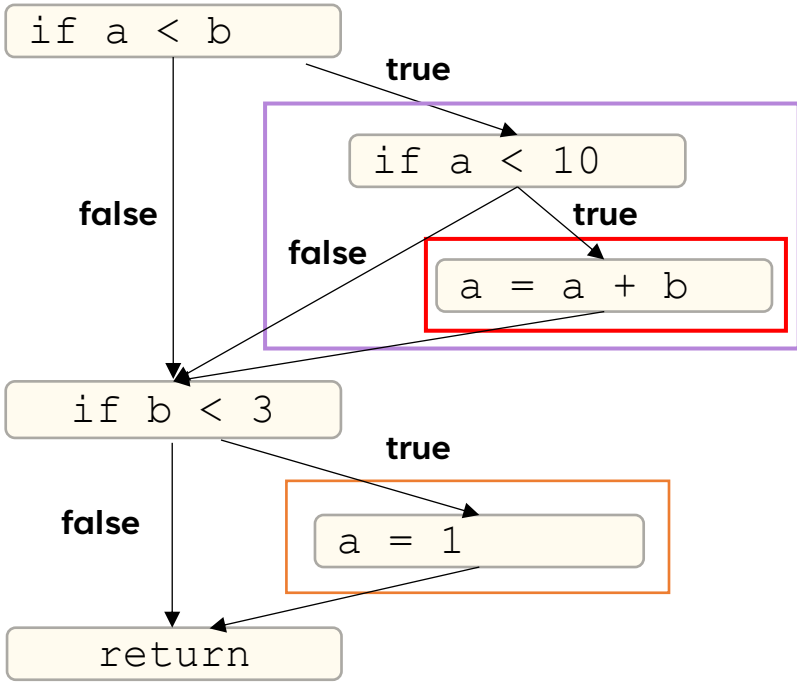


FLOWCHARTS: VISUALIZING CONTROL

ABSTRACTING CODE: INSTRUCTION FLOWCHARTS

```
void funk(int a, int b){
    if (a < b){
        if (a < 10){
            a = a + b;
        }
    }
    if (b < 3){
        a = 1;
    }
    return;
}
```





FLOWCHARTS: A USEFUL TOOL

ABSTRACTING CODE: INSTRUCTION FLOWCHARTS

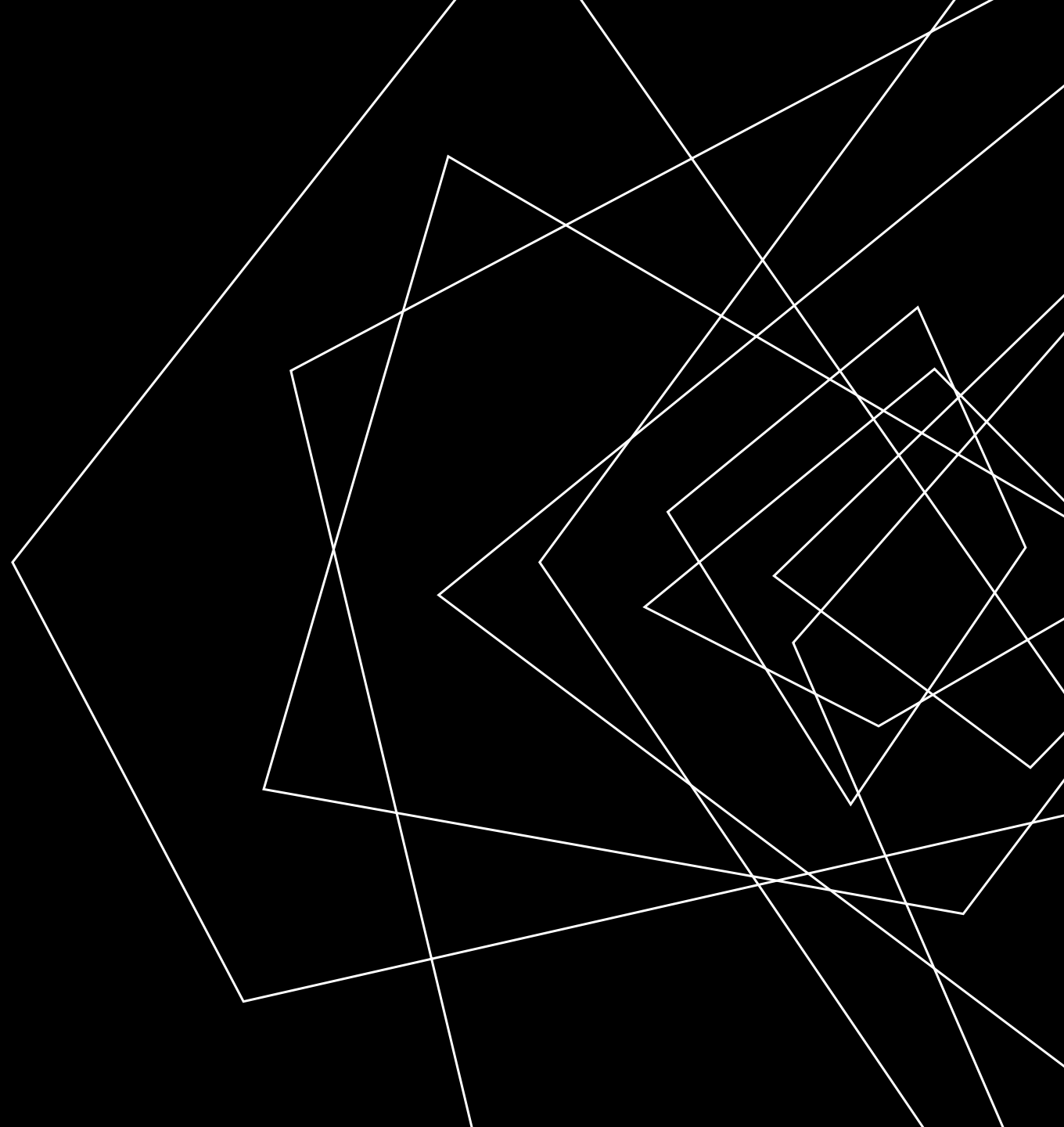
MAYBE THIS IS HOW YOU LEARNED TO THINK ABOUT CODE!

IT'S A NICE WAY TO VISUALIZE THE *CONTROL FLOW* OF THE PROGRAM

WE CAN EXTEND THIS INTUITION FOR PROGRAM ANALYSIS

LECTURE OUTLINE

- Instruction Flowcharts
- Control Flow Graphs



COMPACTING THE FLOW CHART

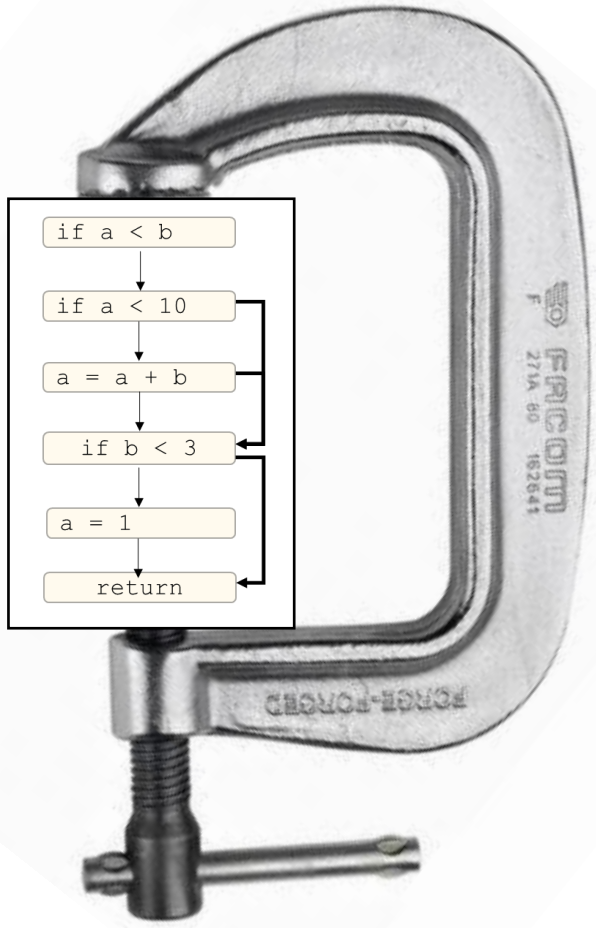
ABSTRACTING CODE: CONTROL-FLOW GRAPHS

FROM FLOWCHARTS TO CONTROL FLOW GRAPHS

- This graph is needlessly verbose
- Too many nodes that communicate nothing

WHAT IF WE ELIMINATE THE 1 INSTRUCTION PER NODE CONSTRAINT?

- Attempt to use as few edges as possible



BASIC BLOCKS

ABSTRACTING CODE: CONTROL-FLOW GRAPHS

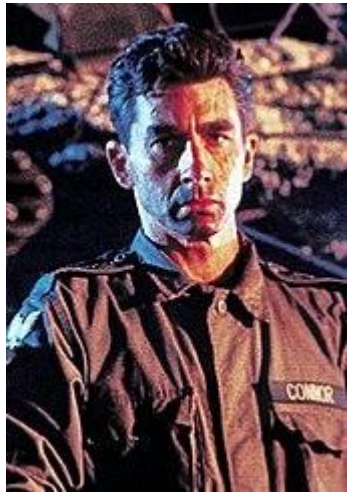
DEFINITION: SEQUENCE OF INSTRUCTIONS GUARANTEED TO EXECUTE WITHOUT INTERRUPTION

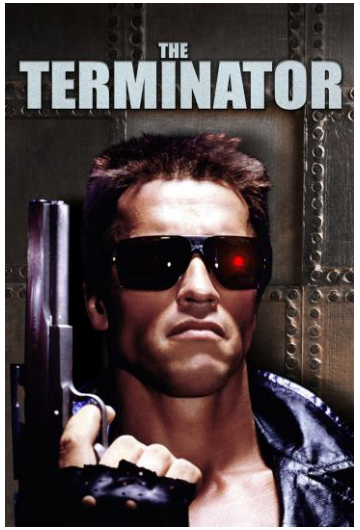
BASIC BLOCKS BOUNDARIES

ABSTRACTING CODE: CONTROL-FLOW GRAPHS

TWO DISTINGUISHED INSTRUCTIONS IN A BLOCK (MAY BE THE SAME INSTRUCTION)

- Leader: An instruction that begins the block
- Terminator: An instruction that ends the block





BASIC BLOCKS BOUNDARIES

ABSTRACTING CODE: CONTROL-FLOW GRAPHS

TWO DISTINGUISHED INSTRUCTIONS IN A BLOCK (MAY BE THE SAME INSTRUCTION)

- Leader: An instruction that begins the block

  The first instruction in the procedure

  The target of a jump

  The instruction after an terminator

- Terminator: An instruction that ends the block

  The last instruction of the procedure

  A jump (goto, if statement, loop construct)

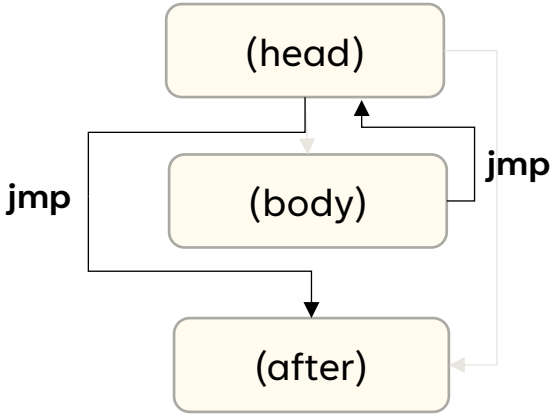
  A call (We'll use a special LINK edge)

BENEFITS OF BASIC BLOCKS

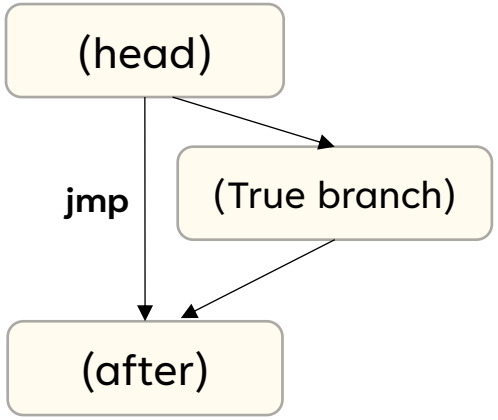
ABSTRACTING CODE: CONTROL-FLOW GRAPHS

CHARACTERISTIC STRUCTURE OF COMMON CONTROL CONSTRUCTS

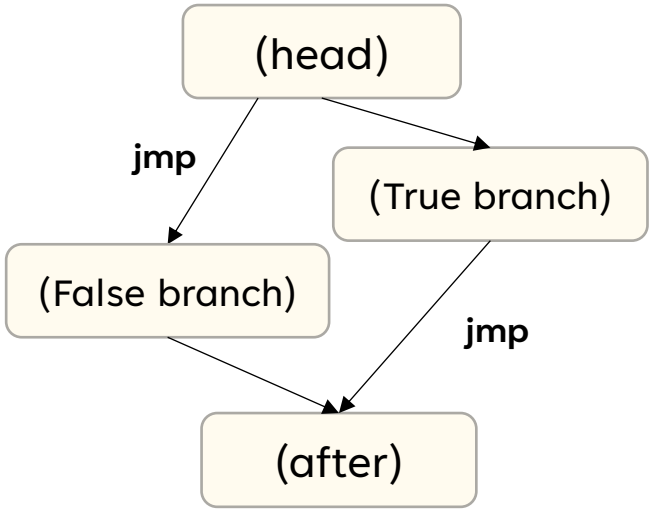
Loops



If-stmt



If-else



NOTE CFGS ARE PER-FUNCTION OBJECTS

ABSTRACTING CODE: CONTROL-FLOW GRAPHS

```
int foo(int c){
  a = 1;
  b = 2;
  if (c > 5){
    c = 1;
  }
  return 0;
}

int main(){
  int local = 1;
  int ret = foo(local);
  if (ret > 1){
    return 1;
  }
  return 2;
}
```

1 CFG for foo

1 CFG for main

Special “link edge” to connect call to its return site

EXERCISE: BUILD THE CFG

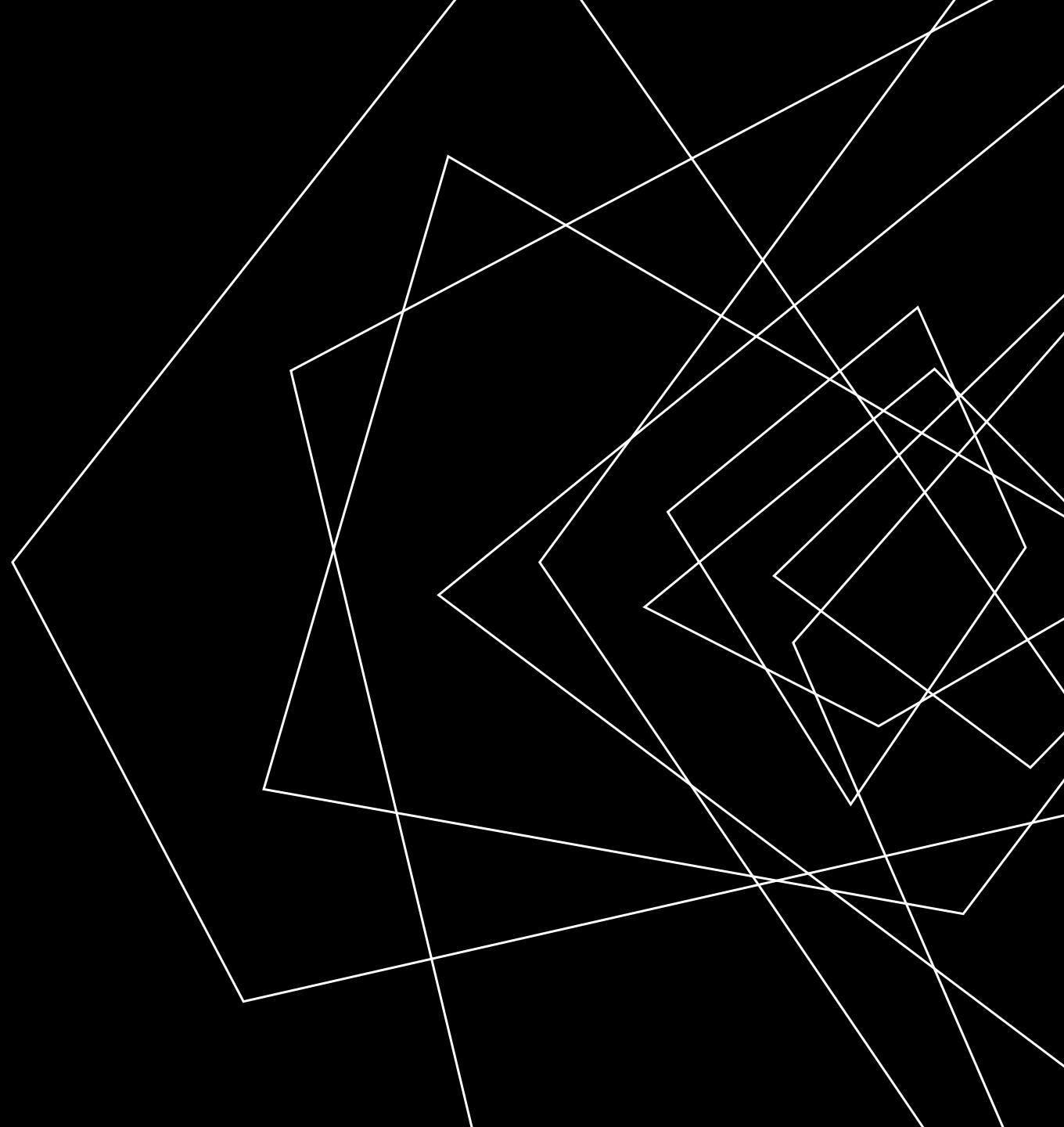
ABSTRACTING CODE: CONTROL-FLOW GRAPHS

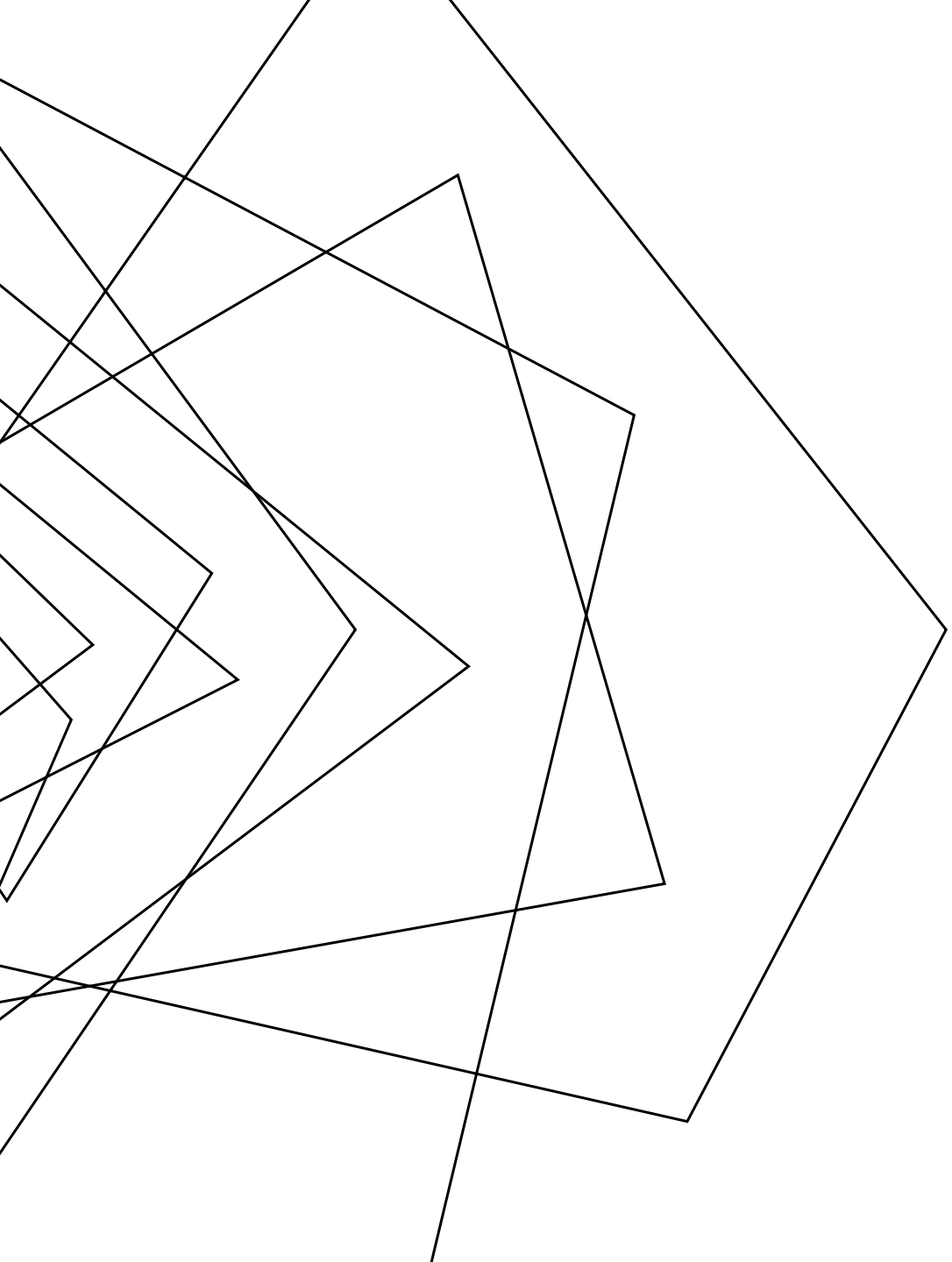
```
int foo(int c){
  a = 1;
  b = 2;
  if (c > 5){
    c = 1;
  }
  return 0;
}

int main(){
  int local = 1;
  int ret = foo(local);
  if (ret > 1){
    return 1;
  }
  return 2;
}
```

LECTURE END!

- Instruction Flowcharts
- Control Flow Graphs

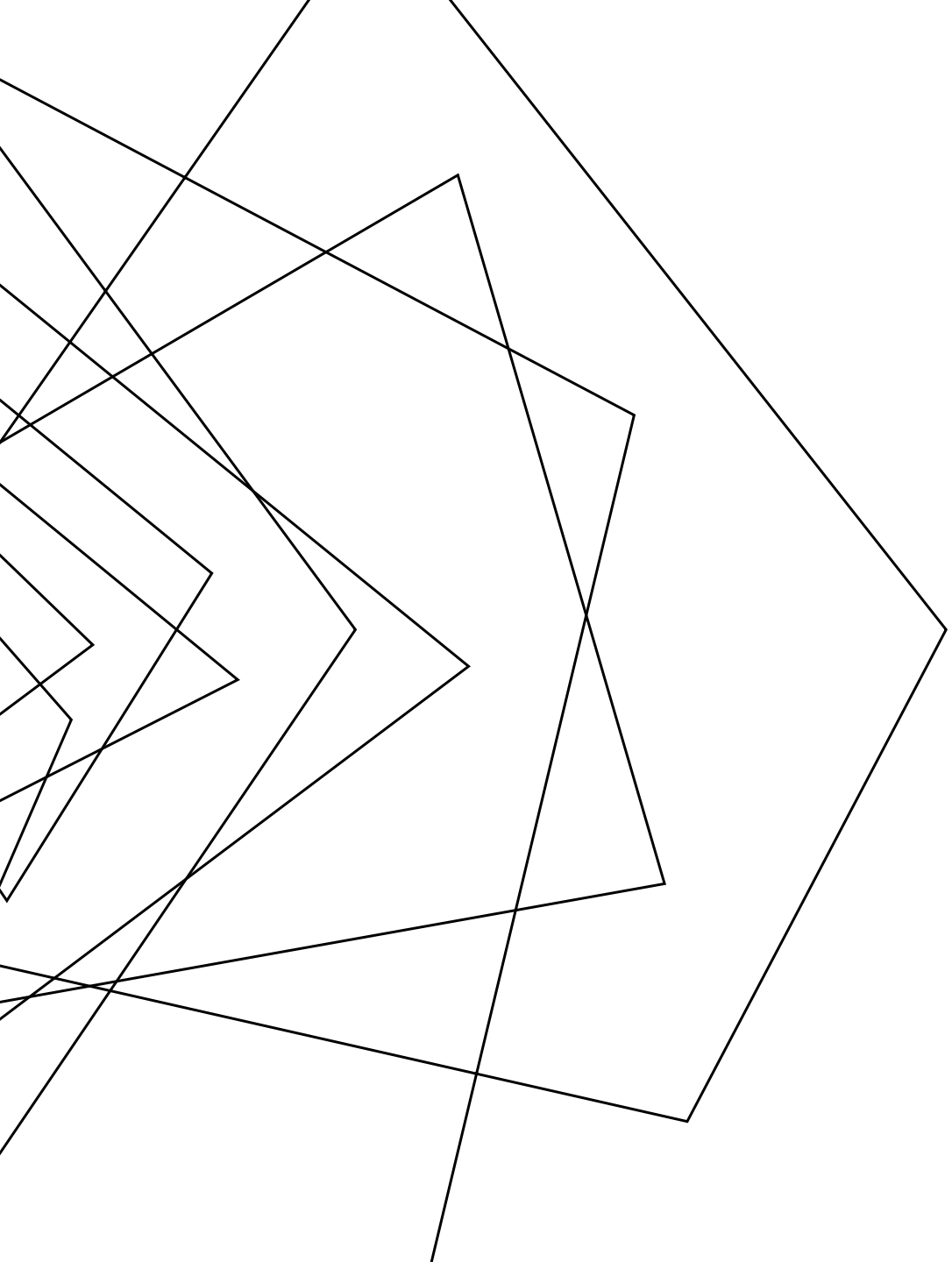




SUMMARY

DESCRIBED THE NEED TO VISUALIZE PROGRAMS IN WAYS OTHER THAN A FLAT LISTING OF SOURCE CODE

SHOWED ONE SUCH VISUALIZATION, THE CONTROL-FLOW GRAPH



NEXT TIME

SHOW ADDITIONAL PROGRAM ABSTRACTIONS TO SIMPLIFY ANALYSIS, IN PARTICULAR SSA FORM

USE THESE CONCEPTS TO INTRODUCE LLVM IR