EXERCISE #15

SIDE CHANNEL REVIEW

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

Provide an instance of a function with a sensitive argument v and leaks a bit of v via a timing side channel

EXERCISE #15: SOLUTION

SIDE CHANNEL REVIEW

ADMINISTRIVIA AND ANNOUNCEMENTS

DEPENDENCE RELATIONS

EECS 677: Software Security Evaluation

Drew Davidson

LAST TIME: SNEAKY DATAFLOW

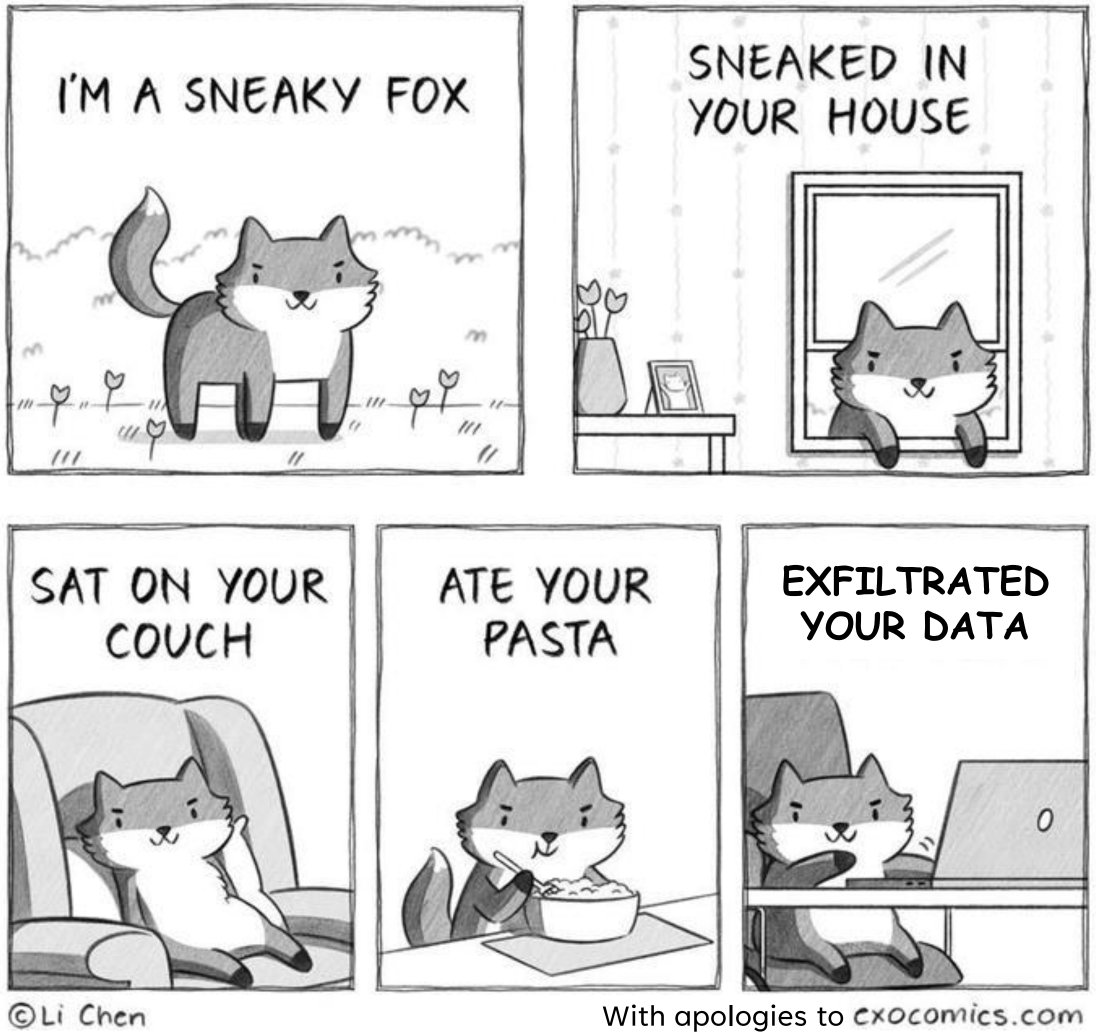
REVIEW: SIDE CHANNELS

General Side-Channel

- General side-channel: using a predictable phenomenon outside of the semantics of the program
- Covert channel: special instance of a side channel that is used intentionally by the program
- Either case: subverts the guarantee of a (naïve) static dataflow

Implicit Flow

- Launder a data dependency through a control dependency



With apologies to exocomics.com

LAST TIME: TIMING SIDE CHANNELS

REVIEW: SIDE CHANNELS

A REAL-WORLD THREAT!

Mitigating Information Leakage Based on Variable Timing

Operations such as message authentication code (MAC), RSA signature padding, and password processing are especially susceptible to timing side channel attacks. These operations include a step that compares two values. If the comparison time is dependent on the inputs, malicious actors can use the timing differences to learn valuable information. This type of attack, known as an *oracle attack*[6], can target processes that are not vulnerable to speculative execution side channels and can operate at an API level.

intel

Developers / Topics & Technologies / Software Security Guidance / Best Practices / Side Cha

Guidelines for Mitigating Timing Side Channels Against Cryptographic Implementations

ID	758403
Updated	6/29/2022
Version	2.1
Public	

LAST TIME: TIMING SIDE CHANNELS

REVIEW: SIDE CHANNELS

A real-world Threat!

How to fix (in software)?

- Best idea (that I know of): an elaboration on the dataflow facts

  Ensure uniform operation between flows

LAST TIME: TIMING SIDE CHANNELS

REVIEW: SIDE CHANNELS

A REAL-WORLD THREAT!

HOW TO FIX (IN SOFTWARE)?

- Best idea (that I know of): an elaboration on the dataflow facts

  Ensure uniform operation between flows

```
bool checkPW(const char * given){
  const char * expected = "12345";
  int gLen = strlen(given);
  int eLen = strlen(expected);
  if (gLen != eLen){ return false; }
  for (int i = 0; i < eLen; i++){
    if (given[i] != expected[i]){
      return false;
    }
  }
  return true;
}
```

volatile

```
bool checkPW(const char * given){
  const char * expected = "12345";
  int gLen = strlen(given);
  int eLen = strlen(expected);
  bool ok = true;
  if (gLen != eLen){ ok = false; }
  for (int i = 0; i < eLen; i++){
    int gIdx = math.min(gLen - 1, i);
    if (given[gIdx] != expected[i]){
      ok = false;
    }
  }
  return ok;
}
```

LAST TIME: SNEAKY DATAFLOW

REVIEW: SIDE CHANNELS

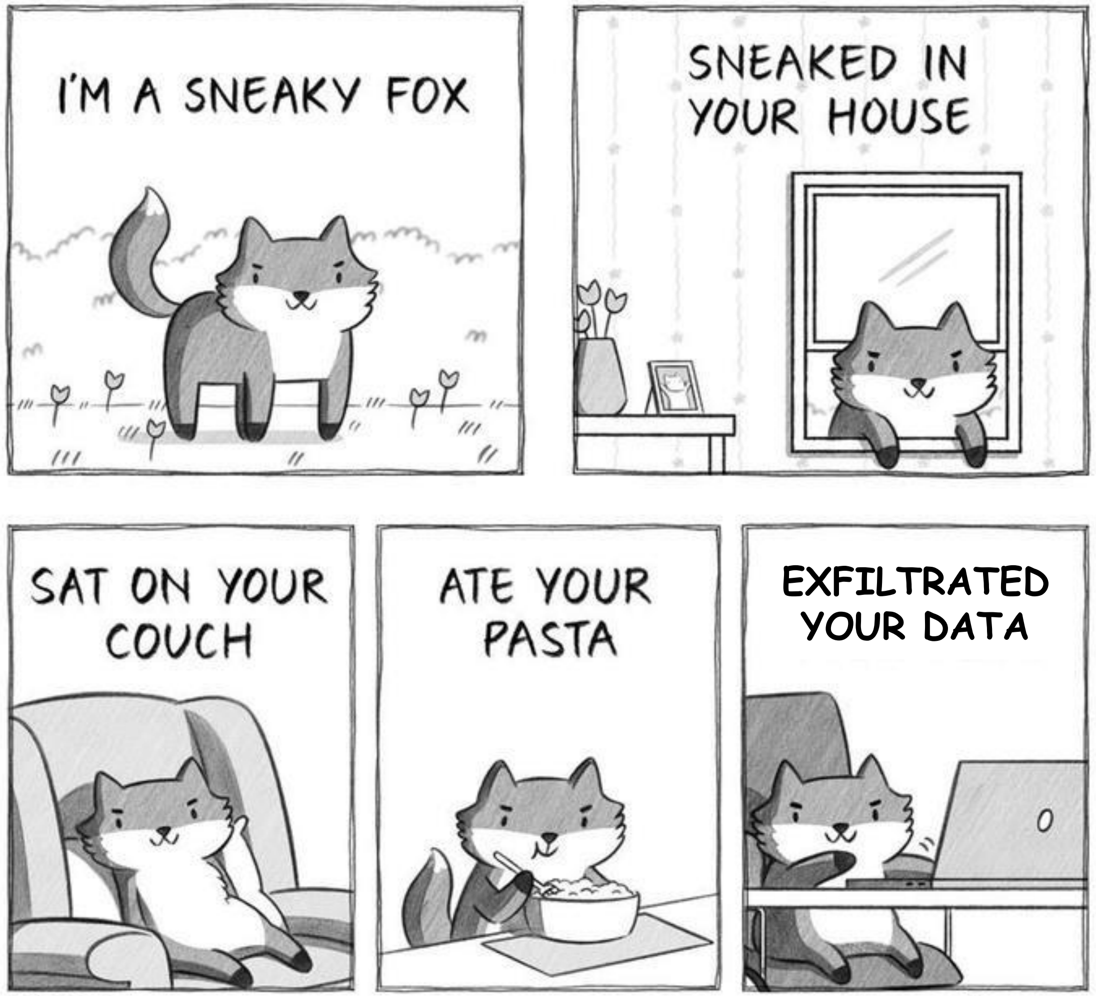
General Side-Channel

- General side-channel: using a predictable phenomenon outside of the semantics of the program
- Covert channel: special instance of a side channel that is used intentionally by the program
- Either case: subverts the guarantee of a (naïve) static dataflow

Implicit Flow

- Launder a data dependency through a control dependency

```
if(x){
  y=true;
}else{
  y=false;
}
```



©Li Chen

With apologies to exocomics.com

LAST TIME: SNEAKY DATAFLOW

REVIEW: SIDE CHANNELS

General Side-Channel

- General side-channel: using a predictable phenomenon outside of the semantics of the program
- Covert channel: special instance of a side channel that is used intentionally by the program
- Either case: subverts the guarantee of a (naïve) static dataflow

Implicit Flow

- Launder a data dependency through a control dependency

Commonality: we don't care about particular values, we care about dependency

THIS LECTURE

DELVING INTO DATA ABSTRACTIONS
THAT INDICATE DEPENDENCY

LECTURE OUTLINE

- Dependence relations
- Control Dependence
- Data Dependence

WHY DOES STATEMENT X DO THING Y?

DEPENDENCE RELATIONS

Often interested in a subset of program behavior

What "influenced" statement X?

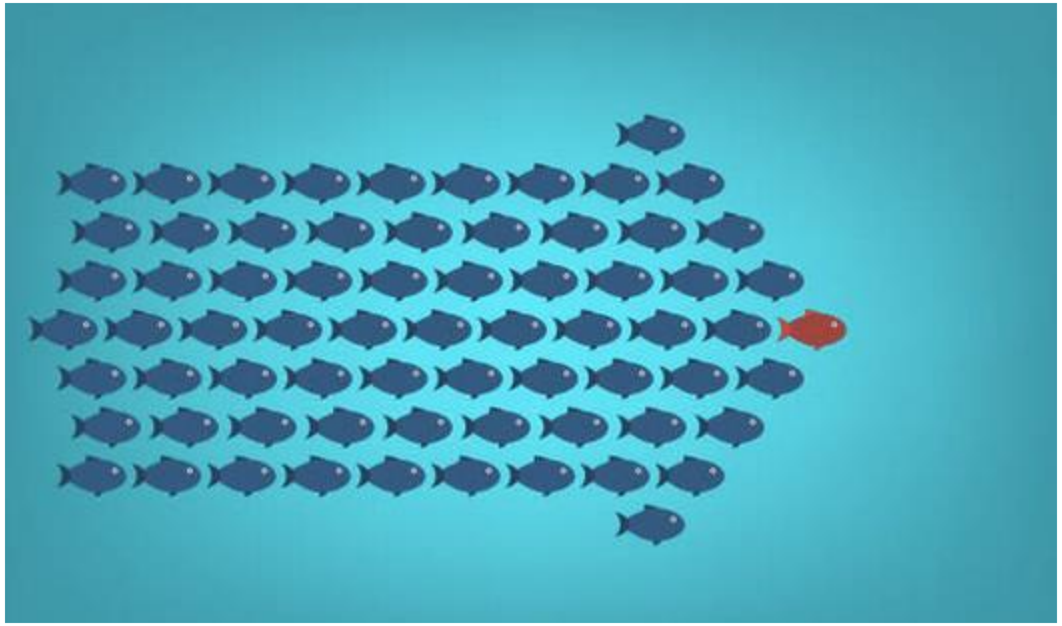
What did statement X "influence"?

Useful in a variety of contexts

Consider a pointer… what might make it null?

Assisting Scalability

Don't get lost in details unrelated to my pointer / bug



APPLICATIONS

DEPENDENCE RELATIONS

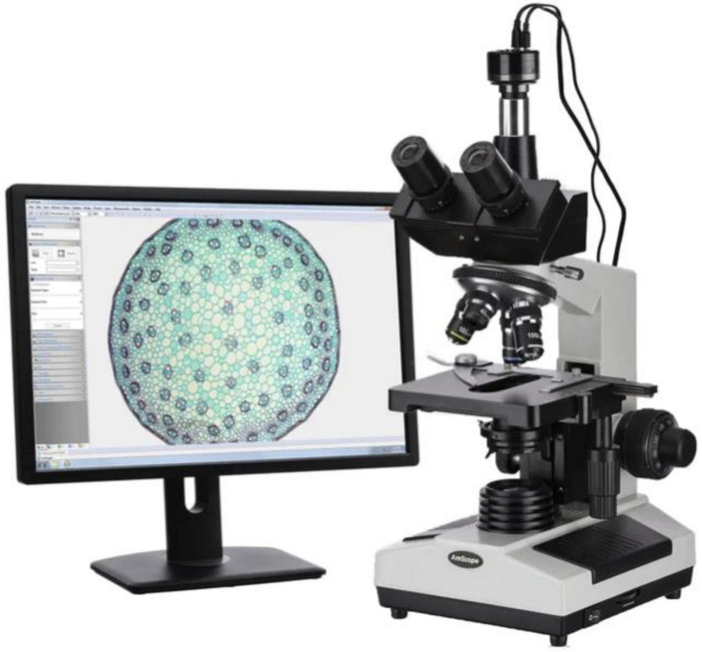
PROACTIVE

What causes my program to crashing?

Does this statement leak data?

REACTIVE

Zoom in on a suspicious operation



LECTURE OUTLINE

- Dependence relations
- Control Dependence
- Data Dependence

"CONTROL RELIANCE" INTUITION

DEPENDENCE RELATIONS

Consider the following procedure…

What other statements ***decide*** whether a given statement executes?

```
void foo(){
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: }
```

The outcome of Line 2 decides on whether Line 3 is executed

The outcome of Line 2 decides on whether Line 4 is executed

The outcome of Line 1 does **not** decide on whether Line 2 is executed

The outcome of Line 2 does **not** decide on whether Line 5 is executed

"CONTROL RELIANCE" INTUITION: IMMEDIACY

DEPENDENCE RELATIONS

MANY INSTRUCTIONS MAY CAUSE A SKIP-OVER

Line 5 relies on Line 1 and Line 2 and Line 3!

Also convenient to say that every line in a procedure relies on the entry to that procedure

We'd say Line 4 "most closely" relies on Line 3 because there is no instruction between line 3 and 4 that decides if Line 4 executes

```
void foo(){
1: if ( i == 1){
2:   if (j == 1)  {
3:      if (k == 1){
4:         PRINT "hi ";
5:         PRINT "there!";
6:      }
7:   }
8: }
```

CONTROL DEPENDENCE

Informally, an instruction X has a control dependence on Y if:
Statement Y decides whether X executes with no intervening decider

Related concept: MUST a statement A be executed for B to execute?

"CONTROL RELIANCE" INTUITION: IMMEDIACY

DEPENDENCE RELATIONS

Many instructions may cause a skip-over

Line 5 relies on Line 1 and Line 2 and Line 3!

Also convenient to say that every line in a procedure relies on the entry to that procedure

We'd say Line 4 "most closely" relies on Line 3 because there is no instruction between line 3 and 4 that decides if Line 4 executes

```
void foo(){
1: if ( i == 1){
2:   if (j == 1) {
3:     if (k == 1){
4:       PRINT "hi ";
5:       PRINT "there!";
6:     }
7:   }
8: }
```

Control Dependence

Informally, an instruction X has a control dependence on Y if:
Statement Y decides whether X executes with no intervening decider

CONTROL DEPENDENCE

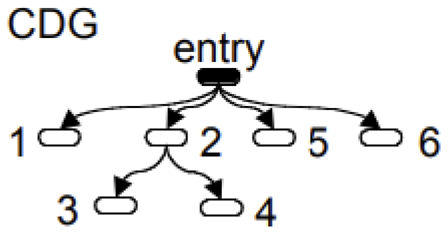
DEPENDENCE RELATIONS

Capture Control Dependence in a Data structure

The control dependence graph

```
void foo(){
1: READ i;
2: if ( i == 1)
3:     PRINT "hi!"
   else
4:     i = 1;
5: PRINT i;
6: }
```

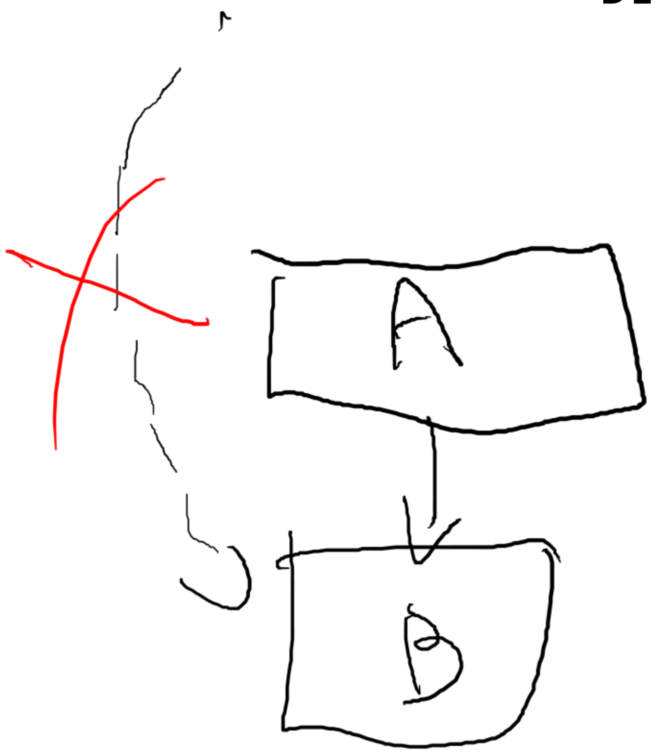
Related concept: MUST a statement A be executed for B to execute?




Domination

DOMINATORS

DEPENDENCE RELATIONS



POSTDOMINATION

DEPENDENCE RELATIONS

INTUITION ON CONTROL DEPENDENCE

What is the closest statement are you guaranteed to execute?

POSTDOMINATION

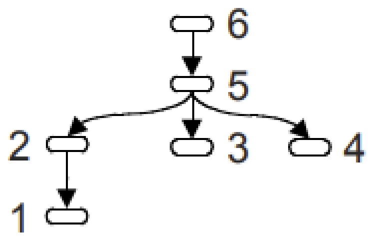
A Statement Y **postdominates** X ⇔ every path from X is guaranteed to go through Y, denoted X in PDOM(Y)

Intuitively, X is "destined" to meet Y

A Statement Y **immediately postdominates** X ⇔ X in PDOM(Y) and there is no intervening postdominator, denoted X in IPDOM(Y)

```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```

Post dom tree



BUILDING THE CDG

DEPENDENCE RELATIONS

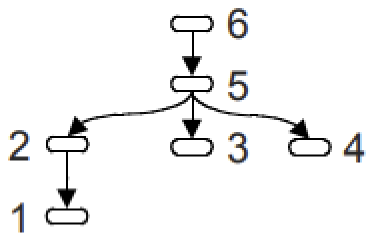
(Immediate) Forward Dominators

X IN IPDOM(Y) ⇔ Y in IFDOM(X)

```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```

2 in IPDOM 5

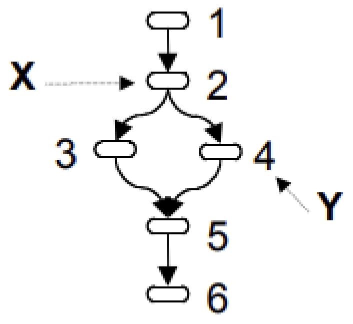
5 in IFDOM 2

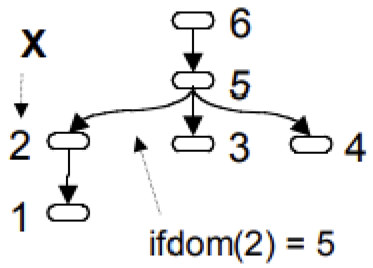


BUILDING THE CDG

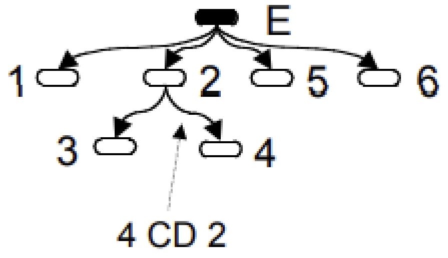
DEPENDENCE RELATIONS

Y is control dependent on X ⇔ there is a path in the CFG from X to Y that doesn't contain the immediate forward dominator of X






```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```



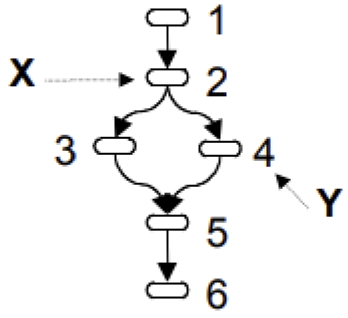
LECTURE OUTLINE

- Dependence relations
- Control Dependence
- Data Dependence

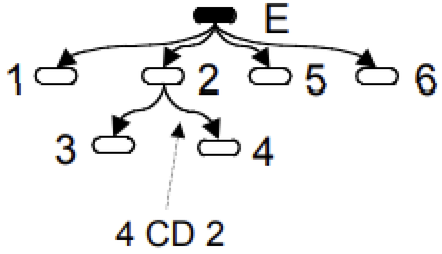
DATA DEPENDENCE

DEPENDENCE RELATIONS

Influence is more than control, it's also what values mattered to your behavior




```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```

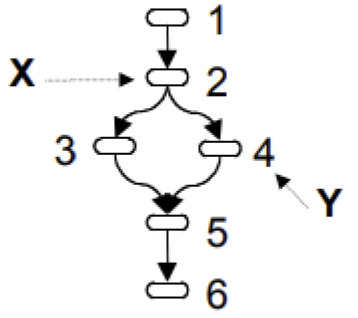


Note here: 1 might have set 5, but it's not control dependent!

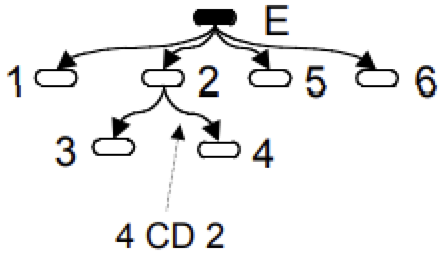
THE DATA DEPENDENCE GRAPH

DEPENDENCE RELATIONS

Depiction of the *reaching definitions* of each statement




```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```



NEXT TIME
DEPENDENCE RELATIONS

Consider the Program Slice

Forward Slice: the portions of the program a given statement influences

Backwards Slice: the portions of the program influenced by a give statement

WRAP-UP