The language you will be analyzing is similar to C, but it is a specialized language developed to study memory pointers. alloc() allocates memory locations, * dereferences a pointer to store to or read from a memory location, and assignments and if / else statements act as expected.

Every program will contain one single print statement that always appears at the end of the program. Your task will be to state whether that print statement ALWAYS, SOMETIMES, or NEVER prints SENSITIVE. You will be trying to give accurate answers while completing each task as QUICKLY AS POSSIBLE.

For some of the problems, you may be given a model of the code that provides information about the value of variables and memory locations throughout the program. For some problems, this value information will be precise, including exactly all the values that could be there, but no more. In other cases, the value information will over-approximate, containing the precise values plus possibly some extra values as well. This value information might help you solve problems more efficiently than just looking at the code.

Press any button on the mouse to continue to the next page of instructions.

Let's look at an example:

```
1   var1 = alloc();
2   var2 = alloc();
3   *var1 = PUBLIC;
4   *var2 = SENSITIVE;
5   print(*var2);
```

alloc() creates a new memory region, gives it a unique name, and returns a pointer to it. So on line 1, alloc creates a region we'll call Mem1, and var1 stores a pointer to Mem1. The value information would represent this as var1: ->Mem1. On line 2, var2 stores a pointer to a new memory region, Mem2, represented as var2: ->Mem2. On line 3, we write PUBLIC into the memory pointed to by var1, i.e., we write PUBLIC into Mem1. The value information would represent this as Mem1: PUBLIC. On line 4 we write SENSITIVE into Mem2 (the memory pointed to by var2), represented as Mem2: SENSITIVE. Finally, on line 5, we print the data from the memory pointed to by var2.

If a precise set of value information is given along with the program, you will see the program on the left and the model showing the value information on the right:

```
var1 = alloc();                                *****Model*****
var2 = alloc();                               var1: ->Mem1
*var1 = PUBLIC;                               var2: ->Mem2
*var2 = SENSITIVE;
print(*var2);                                 Mem1: PUBLIC
                                              Mem2: SENSITIVE
```

One expert method for answering this kind of question works backwards from the print statement, or the sink. We will use this method in our examples. You can use the value information solve this problem as follows. If you look at the end of the code, you see that the print is printing *var2. Looking at the model, var2 points to Mem2. Further down the model, Mem2 is holds SENSITIVE. Since models always include all the values that might occur at the print statement, the print statement will only every print *var2 = *->Mem2 = SENSITIVE. Again, these models always have _at least_ the right information, the answer for this problem would be that the program ALWAYS prints sensitive data.

Press any button on the mouse to continue to the next page of instructions.

Instructions Page 3 of 7

Suppose instead that you were given a slightly different program, still with the precise model:

```
var1 = alloc();                                    *****Model*****
*var1 = PUBLIC;                                    var1: ->Mem1, ->Mem2
var2 = var1;                                       var2: ->Mem1
var1 = alloc();
*var1 = SENSITIVE;                                 Mem1: PUBLIC
print(*var2);                                      Mem2: SENSITIVE
```

Looking at the print statement, the program is printing *var2. The value information says var2 must point to Mem1. It also says that Mem1 must point to PUBLIC. Thus *var2 = *->Mem1 = PUBLIC. Because no other values are possible, this program will NEVER print sensitive data.

If a value could point to anything, the value information will use a '?':

```
var1: ?
```

Press any button on the mouse to continue to the next page of instructions.

Let's explore this with the imprecise model:

```
// select is unknown on entry to this fragment        *****Model*****
var = alloc();                                        select: ?
*var = SENSITIVE;                                     var: ->Mem1, ->Mem2
var = alloc();
if (select == 1) {                                    Mem1: SENSITIVE, PUBLIC
   *var = PUBLIC;                                     Mem2: SENSITIVE, PUBLIC
} else {
   *var = PUBLIC;
}
print(*var);
```

The print is printing *var, and the value information says that var can point to Mem1 or Mem2. Mem1 and Mem2 are both listed as pointing to SENSITIVE or PUBLIC. You might think that this program SOMETIMES prints SENSITIVE, but that answer would be incorrect. By checking the code, the first allocation assigned to var is used to write SENSITIVE to Mem1, and both branches of the if/else write PUBLIC to Mem2. Even though you don't know the value of select, since only the second allocation assigned to var reaches the print, you can conclude that this program NEVER prints SENSITIVE.

The precise model would look like this:

```
// select is unknown on entry to this fragment        *****Model*****
var = alloc();                                        select: ?
*var = SENSITIVE;                                     var: ->Mem1, ->Mem2
var = alloc();
if (select == 1) {                                    Mem1: SENSITIVE
   *var = PUBLIC;                                     Mem2: PUBLIC
} else {
   *var = PUBLIC;
}
print(*var);
```

Take a minute to make sure that you know how to combine your knowledge of the program and the model to get to the correct answer.

Press any button on the mouse to continue to the next page of instructions.

Sometimes, a variable may be assumed to be a certain value on entry. For example:

```
// assume cond is 1 on entry
var1 = alloc();
var2 = alloc();
*var1 = PUBLIC;
*var2 = PUBLIC;
if (cond == 1) {
   *var2 = SENSITIVE;
} else {
   *var1 = SENSITIVE;
}
if (cond == 1) {
   var = var1;
} else {
   var = var2;
}
print(*var)
```

```
*****Model*****
var1: ->Mem1
var2: ->Mem2
cond: 1
var: ->Mem1

Mem1: PUBLIC
Mem2: PUBLIC, SENSITIVE
```

In this case, the print is looking at *var. The model says var can only point to Mem1, and it says that Mem1 can only point to PUBLIC. So this program NEVER prints SENSITIVE data.

Why is this the case? Well, given that cond is 1, the if condition of the first if/else has to be taken, so Mem2 is updated to contain SENSITIVE or PUBLIC. Since that first else can't be taken when cond is 1, Mem1 can only contain PUBLIC. In the second if/else, only the if is taken, so var can only point to Mem1. If this doesn't make sense, stop and ask for further training.

Press any button on the mouse to continue to the next page of instructions.

Sometimes the code will be incomplete and you will not have enough information to tell whether the code ALWAYS or NEVER prints SENSITIVE. For example:

```
// select is unknown on entry to this fragment          *****Model*****
var = alloc()                                           var: ->Mem1
if (sel == 0) {
   *var = PUBLIC;                                      Mem1: PUBLIC, SENSITIVE
} else {
   *var = SENSIITVE;
}
print(*var);
```

The print is looking at *var, which points only to Mem1. But Mem1 points to PUBLIC or SENSITIVE. Since this model may or may not be precise, you need to use the code to verify that the correct answer is that this program SOMETIMES prints SENSITIVE, depending on the value of sel.

Finally, memory can hold pointers to other memory. For example:

```
var = alloc();                                           *****Model*****

*var = alloc();                                         var: ->Mem1

**var = SENSITIVE;

print (**var);                                         Mem1: ->Mem2

                                                       Mem2: SENSITIVE
```

This program would ALWAYS print SENSITIVE. Just verify again that you can use the model alone, without the code, to answer this question.

Press any button on the mouse to continue to the next page of instructions.

Again, in this test, you will see several examples of code that contain a single print at the end, and you will answer whether the code ALWAYS, SOMETIMES, or NEVER prints SENSITIVE. Some of these examples will have more information provided than others. You will be asked to use the available information to answer the questions as quickly as possible.

While you are answering the questions, we will use an eye tracker to assess which pieces of information you are looking at while figuring out your answers. To keep the eye tracker calibrated, we will ask you to look at a fixation point (a '+' in the center of the screen) for two seconds prior to the start of each question. Please stare at the + without blinking when it appears on the screen prior to each question.

Please ask the experimenter if you have any questions. Otherwise, press the mouse button to begin the experiment.