Computer Science 1MC3

Lab 10 – State Diagrams / Scope and Visibility

A state diagram requires three things; states, an initial state, and move conditions. States are like flow charts in the sense that we have boxes (states) that are connected by arrows. However, there are some key differences. Unlike flow charts we do not require a start or an end box, this implies that state diagrams can not represent a legal algorithm (an algorithm must terminate); furthermore a state box may have as many arrows pointing to or from it as needed. State diagrams are good models for things like ATM or pop machines, which indefinitely wait for a conditions *debit card* or *cash* respectively.

States

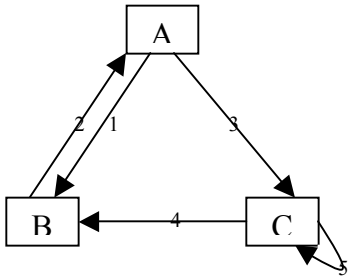
At any given time you must be at a state. A state can represent an action or a change, a change of state occurs when a given condition is met, in which case you must go to another state. In our ATM example we may have states like *ask for pin* or *dispense cash*.

Now, since we do not have a start or an end it is necessary to declare an initial state. The initial state tells you were to "step into" the sate diagram where you then wait for instructions.

Moving through State Diagrams

In flow charts, the only way to have a choice of a direction you must have a decision diamond, even then you may only go down the YES or NO arrow. In state diagrams we eliminate decision diamonds putting the condition right on the arrow while unrestricting the amount of arrows that may be drawn from a state. However, your conditions may not be ambiguous, that is, the condition is fulfilled or not fulfilled. *Pin entered* would be a good state, whereas *grass is green* is not.

Now imagine that you standing in this state diagram.



See the states as islands and arrows as bridges between them. You are instructed to stand at island A and to walk across a bridge if you hear a its number called. So, if you heard [1-2-3-5-4] you would end up at island B.

Questions:

1. Draw a state diagram to model a ATM machine.
2. Draw a state diagram to model a POP machine.
3. Can a state diagram be drawn where you get "stranded" at a state?

Scope and Visibility

The scope of a name is the portion of the program in which it exists. The rules for defining where a variable declared in some declaration can be used within the program is called its visibility.

Scope

The three levels of scope are **local**, **global**, and **modular**; respectively meaning that access to a variable is restricted to a single function, the whole program, or many programs. More specifically,

- parameters and variables created in a function are local
- variables declared before the main program are global
- static variables declared before the main program are modular global variables

---

Visibility

The visibility of a variable is the range in which it may be referenced in the actual program. The visibility rules follow from the scope rules.

- local variable may only be accessed from the same function it is declared in
- global variables may be accessed by any function in the program
- modular global variables may be accessed by other programs or modules

It is incorrect to try to access a variable from outside its visibility. Trying to access the local variables of function A from function B is not legal. This said it is also desirable to give a variable the least amount of visibility required for what you need. That is, it is preferable for a function to be local.

It should also be noted that you may name variables that fall outside of each other's visibility whatever you want. So it would be possible to have five local variables named `x` in several different functions. Conversely you are not allowed to name two variables with overlapping visibility the same thing, meaning that if you have a global variable `x` you may not have a local variable `x`.

Static

When you add the static modifier to a variable in a function this prevents said variable from being erased off the stack when the function terminates. A good example where static is necessary, is when a function requires to know how many times it was invoked (called). In this case the count variable would only need to be in the visibility of the function but not erased when the function terminates.

Problems

```
#include <stdio.h>

#define PI 3.14

int func(int x2, y2, z2);

int x,y,x;


int main (void) {

      int x1, y1, z1;

}

int func(int x2, y2, z2) {

      int   x1, y1, z1;
      static int  x4;

}
```

1. Determine the visibility and scope of this program.
2. Is the visibility of any variables overlapping?
3. Is overlapping visibility a problem? Why or why not?