

Art 382: Introduction to Interactive Media

mcdo@umbc.edu

Lecture 23, May 5

Schedule

- Take Roll
- Final Exam
- Final Project
- Final notebook
- Loose ends
 - review "for" loops
 - while loops
 - fun with if

Final Exam

- Thursday, May 14th at 8AM
 - Traffic will be awful, so leave early, Ofer.
- Will be all about ActionScript
 - if it's on a slide, it could show up
 - that's why I make slides.

Final project

- Worth the same as all the other projects
- Due May 14th at 8AM
- The Circus Act, as described earlier
- rubric available
 - I'm grading the drawings

Final Notebook

- 20 more drawings by your
- 40 more samples
- Just like last time.
- Also due May 14th at 8AM

- Probably ought to start on some of this...

review: "for" loops

- four parts -- on the final
 - initializer: an assignment that sets the variable's first value
 - condition: and expression that tests the variable; if true, do the statements in the body
 - increment: an assignment that modifies the first variable-- presumably to get it closer to making the condition false
 - body: a pair of curly brackets holding some statements

review: example loops

- `for (i=0; i<10; i=i+1) { a[i] = 1; }`
 - `i` is "initialized" to 0
 - the loop continues until `(i<10)` is false
 - `i` increases by 1 each time
 - the loop modifies the array `a`
- `for (j=10; j>0; j=j-2) { a[j] = 1; }`
 - `j` is initialized to 10
 - the loop continues until `(j>0)` is false
 - `j` decreases by 2 each time

while loops

- "for" loops do something a certain number of times
 - look at all the numbers, change all the diapers
- "while" loops -- until something happens
 - find the first dog
 - wait until you push a button
 - wait for 10 seconds
- kind of interchangeable

while syntax: 3 parts

- `while (condition) { body(); }`
 - "while"
 - condition in parentheses
 - loops contents ("the body of the loop") in curly brackets
- very easy to make loops that don't stop
 - "infinite" loops
 - Flash times you out, but that takes 15 seconds

examples

- `while (x<0) { x=x+1; }`
 - adds 1 to x until x is ≥ 0
- `while (x==1) { y=1; }`
 - sets y to be 1 over and over, forever
 - an infinite loop; also an error
 - kinda useless, yes.
- `while (x>0) { y=y-1; x=x-1; }`
 - subtracts x from y in a silly way

more examples

- `while (a[i]!=0) { i=i+1 }`
 - finds an element of `a != 0`
 - many errors possible!
 - what is `i` initially? 394959539?
 - how many elements are in `a`?
 - "going off the end of an array"
 - » Flash says "null object reference"
 - what happens if all `a[i]`'s `== 0`?

why "for" and "while"?

- no reason; you could use only one or the other
- they are a hint about what the code is doing
 - code that is easier to read is better
 - for => always loop the same # times
 - while => loop a more variable # times
- innovations continued
 - do {} while (), loop {} until ();
 - iterator classes

a better example

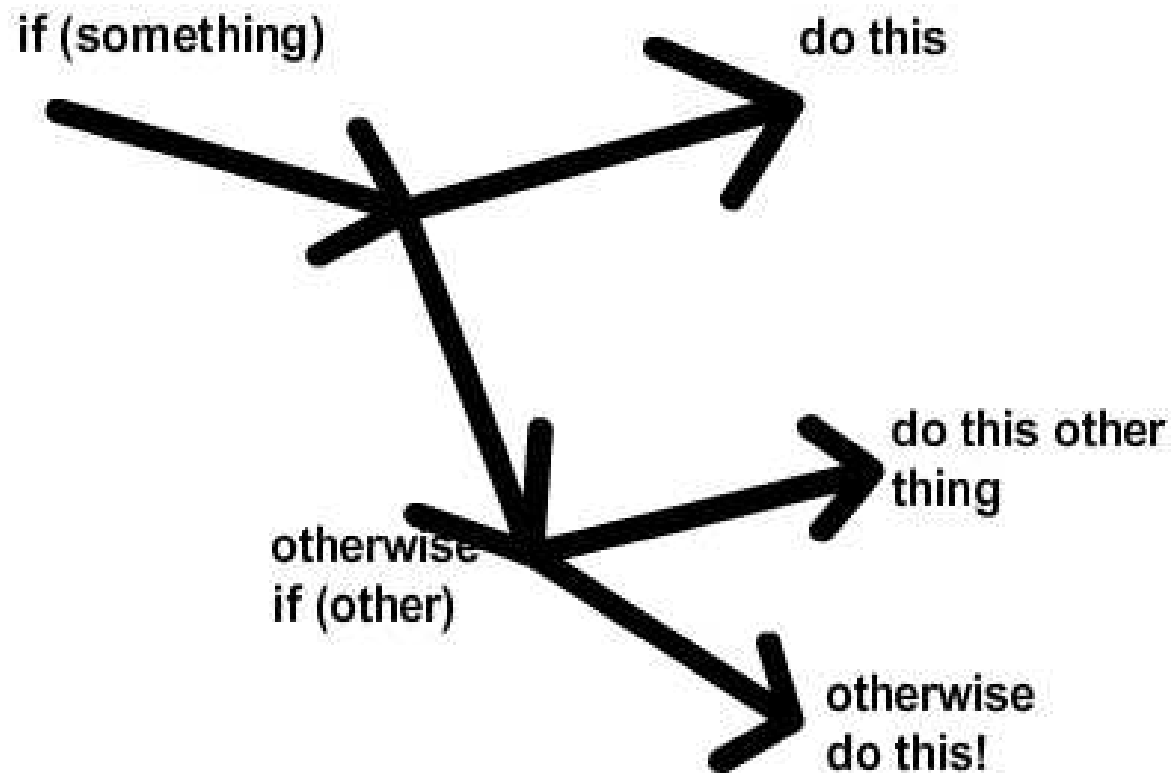
- // given an array, a, with 10 numbers in it
- i=0;
- while ((i<10)&&(a[i]!=0)) { ++i; }
- if (i==10) { trace("I got nothin"); }
- else { trace("a[", i, "]=0"); }

coding to catch mistakes

- if you work with data, some of it is always bad
- code should be "fault-tolerant"
 - easier said than done!
- usually, this just means putting in "if" statements that check the data for errors

more about if's

- originally called "branch" statements



- can pile up on each other!

more if: combining conditions

- if conditions can have many terms; use AND and OR operators to combine them
 - if `((a==1)&&(b==1)) { a=2; }`
 - a gets set to 2 only if a=1 and b=1
 - if `((a==1) || (b==1)) { a=2; }`
 - a gets set if either a or b is 1
- use NOT to negate
 - if `!(a==2) { a=2; }`
 - set a=2, if a is not equal to 2

if combinations

- many ways to say the same thing
 - if (a==2) { a=3; }
 - if ((a<3) && (a>1)) { a=3; }
 - if ((!(a>=3)) && (!(a<=1))) { a=3; }
 - if ((a<4) && (a!=3) && (a!=1) && (a>0)) {a=3;}
- the simplest way is the best

if combinations == "Boolean" algebra

- Charles Boole invented it
- Always gives a "true" or "false" answer

combining "if" statements

```
if (a>0) {  
    if (a<4) { trace("bing"); }  
    else { trace ("foo"); }  
else { trace("bar"); }
```

// same as

```
if ((a>0)&&(a<4)) { trace("bing"); }  
if ((a>0)&&(a>=4)) { trace("foo"); }  
if (a<=0) { trace("bar"); }
```

// which one is better?

else if

```
if (a==1) { trace("foo 1"); }  
else if (a==2) {trace("foo 2"); }  
else if (a==3) {trace("foo 3"); }  
else if (a==4) {trace("foo 4"); }  
else if (a==5) {trace("foo 5"); }
```

switch statements

treating a whole bunch of cases specially

```
switch (a) {  
    case 0: a=1; break;  
    case 1: a=2; break;  
}
```

// same as

```
if (a==0) { a=1; }
```

```
if (a==1) { a=2; }
```


a fault-tolerant function

```
function nameMyBaby(in:String, girl:Boolean) {  
  if (in=="Adolf") { trace("No."); return; }  
  if (girl && (in=="Bubba")) { trace("No."); return; }  
  if (in=="Sausage") { trace("No."); return; }  
  if (in=="Paris") { trace("No."); return; }  
  // 1000 other cases  
  trace("Oh, who cares; your family just has weird  
    names");  
}
```