today

- homework #1 — a brief overview of educational games
- data type conversion
- a few language basics
- libraries

a brief overview of educational games

- history
- anatomy
- motivation
- controversy

educational games: Interactive Learning Systems (ILS)

- definition:

  "Ultimately, all learning is interactive in the sense that learners interact with content to process, tasks to accomplish, and/or problems to solve. However...I refer to a specific meaning of interactive learning as involving some sort of technological mediation between a teacher/designer and a learner. [Thomas Reeves, 1999]

- two major approaches to ILS:
  - *instructive*: learn "from"
  - *constructive*: learn "with"

educational games: Instructive Learning Systems

- computer-aided instruction (CAI)
- intelligent tutoring systems (ITS)
- frame-based tutoring systems:
  - trace problem solving sessions
  - customize for individuals
  - select next step dynamically
  - coach users
- examples:
  - memory modeling (Roger Schank)
  - representation of misconceptions (Kurt VanLehn)
  - construction of rules (John Anderson)

educational games: John Anderson

- PhD Stanford, at CMU since 1978
- production systems: ACT, ACT*, ACT-R
  - Algebra tutor
  - Geometry tutor
  - LISP tutor
- acquisition of cognitive skill
- 3 stages of development:
  1. declarative stage
  2. knowledge compilation stage
  3. procedural stage

educational games: Constructive Learning Systems

- *Constructivism* — Jean Piaget
- *Constructionism* — Seymour Papert
- LOGO (circa 1970's)
- Star*LOGO (1990's)
- Mindstorms (1999)

educational games: Seymour Papert

- at MIT since early 1960's, by way of South Africa, Cambridge and Geneva
- Perceptrons (1970)
- Mindstorms:Children Computers and Powerful Ideas (1980)
- The Children's Machine: Rethinking School in the Age of the Computer (1992)
- The Connected Family: bridging the digital generation gap (1996)
- http://www.mamamedia.com

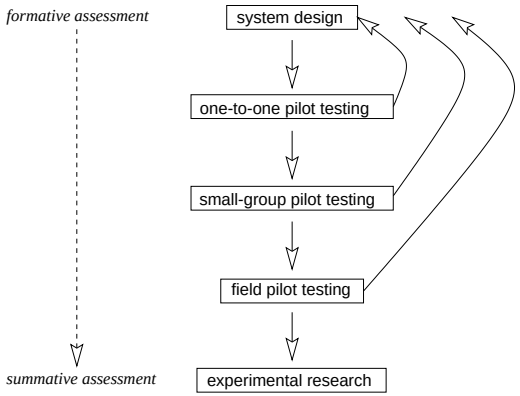
educational games: Anatomy of an ILS

- components
- system development cycle
- evaluation methodologies

educational games: ILS components

- *domain knowledge*
- *teaching component*
- *user interface*
- *student knowledge*
- *system adaptivity*
- *control component*

educational games: system development cycle



educational games: evaluation methodologies

- pre and post tests
- surveys
- session logs
- system products
- observations

educational games: motivation

- "Oh, if kids were only as motivated in school as they are in playing Nintendo." [Elliot Soloway, 1991]
- "Children assimilate information and acquire skills with astonishing speed when playing video games. Although much of this gain is of dubious value, the phenomenon suggests a potent medium for learning more practical things." [Herb Brody, 1993]

educational games: Thomas Malone

- PhD Stanford, at MIT since early 1980's (?)
- comprehensive experimental research: "What makes computer games fun?"
- challenge
- fantasy
- curiosity

educational games: Toys vs Tools

- Malone makes an important distinction
- Toys:
  - exist for their own sake, with no external goals
  - difficult to play, to increase challenge
- Tools:
  - exist because of their external goals
  - easy to use, to expedite the user's goal
- Educational Software:
  - Toy *and* Tool

educational games: Internet learning communities

- CoVis [Roy Pea, 1993] (http://www.covis.nwu.edu/, http://www.letus.org/)
- Belvedere [Dan Suthers, 1997] (http://lilt.ics.hawaii.edu/belvedere/index.html)
- KIE, WISE [Marcia Linn, 1995] (http://wise.berkeley.edu/)
- Concord Consortium (www.concord.org)
- EOE (www.eoe.org)

educational games: Multi-player educational games

- MUD's
- Pueblo [Jim Walters & Billie Hughes] (http://oldpueblomoo.arizona.edu/, http://bessie.englab.slcc.edu/)
  - start-up
  - help
  - reality check
- MOOSE Crossing [Amy Bruckman] (http://www.cc.gatech.edu/elc/moose-crossing/)
  - Internet

educational games: Controversy

- "No matter what we do, a huge infusion of technology is coming to education. It doesn't matter if it works or not, whether we make mistakes or not. It's coming because so much money is behind it. And because that infusion of technology is inevitable, it would be nice to start adding some new perspectives about technology in the schools. It's just possible our decisions about technology in schools are not being guided by the instincts of our best teachers. Right now, we run the risk of being blinded by science." [Tom Snyder, 1994]

educational games: Jane Healy

- Failure to Connect: How Computers Affect Our Children's Minds (1998)
- not enough pedagogy
- not enough research
- too much, too soon?
- are computers better than TV?

Now back to C...

C preprocessor — question from last class

- what is output?

```
#include <stdio.h>

#define linux 1
#define windows 2
#define OS osx

int main() {

  printf( "hello world!\n" );

#if OS == linux
  puts( "good for you for running linux!" );
#else
  puts( "oh no -- you're *not* running linux!!!" );
#endif

#if OS == windows
  puts( "oh no -- you're running windows!!!" );
#else
  puts( "yippee! you *not* running windows!!!" );
#endif

}
```

...and the answer is:

- the output is:

```
hello world!
oh no -- you're *not* running linux!!!
yippee! you *not* running windows!!!
```

- what if we had put `#define OS linux` ???

data type conversion (1)

- also called *type casting*
- also called *coercion*
- in C, data types can be converted either *implicitly* or *explicitly*
- here's a reminder of the types:

type	**size in bytes** (on CUNIX)	**range**
char	8	$-128...127$
short	16	$-32,768...32,767$
int	32	$-2,147,483,648...2,147,483,647$
long	32	$-2,147,483,648...2,147,483,647$
float	32	$10^{-38}... 3 * 10^{38}$
double	64	$2 * 10^{-308}... 10^{308}$

- conversion from floating point type (i.e., float or double) to integer type (i.e., short, int, long) will result in loss of precision

data type conversion (2)

- what is the output of the following?

```
int    a = 3;
float  x = 97.6;
double y = 145.987;
int    i = 100000;
short  s;
y = (double)x * y;
x = x + (float)a;
s = i;
printf("a=%d x=%f y=%lf i=%d s=%d\n",a,x,y,i,s );
```


...and the answer is:

- the output is:

```
a=3 x=100.599998 y=14248.330977 i=100000 s=-31072
```

- almost any conversion does something —
  but not necessarily what you intended!!

integers to reals (1)

- example:

```
#include <stdio.h>

int main() {

  float f1 = 12.34;
  float f2 = 12.99;
  int   j, k;

  printf( "original values: f1=%f f2=%f\n",f1,f2 );
  j = (float)f1;
  k = f1;
  printf( "f1 ---> explicit j=%d, implicit k=%d\n",j,k );

  j = (float)f2;
  k = f2;
  printf( "f2 ---> explicit j=%d, implicit k=%d\n",j,k );

}
```

- output:

```
original values: f1=12.340000 f2=12.990000
f1 ---> explicit j=12, implicit k=12
f2 ---> explicit j=12, implicit k=12
```


integers to reals (2)

- example:

```
#include <stdio.h>
#include <math.h>

int main() {
  float f1 = 12.34;
  float f2 = 12.99;
  int   j, k, m, n;

  j = (float)f1;
  k = f1;
  m = ceil(f1);
  n = floor(f1);
  printf( "%f ---> explicit=%d, implicit=%d, ceil=%d, floor=%d\n",f1,j,k,m,n );

  j = (float)f2;
  k = f2;
  m = ceil(f2);
  n = floor(f2);
  printf( "%f ---> explicit=%d, implicit=%d, ceil=%d, floor=%d\n",f2,j,k,m,n );

}
```

- output:

```
12.340000 ---> explicit=12, implicit=12, ceil=13, floor=12
12.990000 ---> explicit=12, implicit=12, ceil=13, floor=12
```


using the math library (1)

- in the previous slide, the functions `ceil()` and `floor()` come from the C math library
- definitions:
  - `ceil( x )`: returns the smallest integer not less than `x`, as a `double`
  - `floor( x )`: returns the largest integer not greater than `x`, as a `double`
- in order to use these functions, you need to do two things:
  1. include the *prototypes* (i.e., function definitions) in the source code:
     `#include <math.h>`
  2. include the library (i.e., functions' object code) at link time:
     `unix$ gcc abcd.c -lm`
- exercise: can you write a program that *rounds* a floating point?

using the math library (2)

- some other functions from the math library (these are function *prototypes*):
  - `double sqrt( double x );`
  - `double pow( double x, double y );`
  - `double exp( double x );`
  - `double log( double x );`
  - `double sin( double x );`
  - `double cos( double x );`
- exercise: write a program that calls each of these functions
- questions:
  - can you make sense of `/usr/include/math.h`?
  - where are the definitions of the above functions?
  - what are other `math` library functions?

looping

- loops in C are just like in Java
- there are 2 methods for looping:
  - counter-controlled (loop for a fixed number of times)
  - sentinal-controlled (loop while a condition is true)
- there are 3 statements for implementing the 2 methodologies:
  - `for`
  - `while`
  - `do...while`
- as always: *beware the infinite loop!*
- `Cntrl-C` interrupts your executing C program
- exercise: can you write 6 loops, one for each method-statement combination?

branching

- branching in C is just like in Java
- there are 2 ways to do branching:
  - `if/else`
  - `switch`
- questions:
  - which is more flexible and powerful?
  - one can always be translated into the other, but not the other way around — which is which?

using the stdio library

- in order to use this library, you need to do one thing:
  1. include the *prototypes* (i.e., function definitions) in the source code:
     ```
     #include <stdio.h>
     ```
- note that you do NOT have to include the library at link time (unlike with and all other libraries):

```
unix$ gcc abcd.c
```


using the stdio library — printf (1)

- `int printf(const char *format, ...)` formatted output to stdout
- formatting:

conversion character	**argument**	**description**
c	char	prints a single character
d or i	int	prints an integer
u	int	prints an unsigned int
o	int	prints an integer in octal
x or X	int	prints an integer in hexadecimal
e or E	float or double	print in scientific notation
f	float or double	print floating point value
g or G	float or double	same as e,E,f, or f — whichever uses fewest characters
s	char*	print a string
p	void*	print a pointer
%	none	print the % character

using the stdio library — printf (2)

- some flags:

flag	description
-	left justify
+	print plus or minus sign
0	print leading zeros (instead of spaces)

- also specify field width and precision
- example:

```
printf( "i=%d s=%d f=6.3f m=43s",i,s,f,m );
```


using the stdio library — scanf (1)

- `int scanf(const char *format, ...)` formatted output to stdout
- formatting:

conversion character	argument	description
c	char*	reads a single character
d	int*	reads a decimal integer
i	int*	reads an integer in decimal, octal (leading 0) or hex (leading 0x)
u	int*	reads an unsigned int
o	int*	reads an integer in octal
x or X	int*	reads an integer in hexadecimal
e, E, f, F, g or G	float or double	reads a floating point value
s	char*	reads a string
p	void**	reads a pointer

- more next time ... POINTERS!

example

```
#include <stdio.h>

void main( void ) {
  int n = 0; /* initialization required */
  printf( "how much wood could a woodchuck chuck\n" );
  printf( "if a woodchuck could chuck wood?" ); /* prompt user */
  scanf( "%d",&n ); /* read input */
  printf( "the woodchuck can chuck %d pieces of wood!\n",n );
  return;
}

$ a.out
how much wood could a woodchuck chuck
if a woodchuck could chuck wood? 12345
the woodchuck can chuck 12345 pieces of wood!
```