CSE 142
Programming I

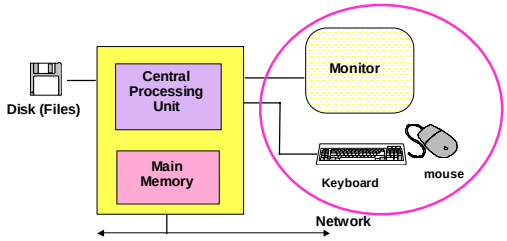
Terminal Input and Output (I/O)

And some miscellaneous topics

Writing Useful Programs

- If a tree falls in the forest…
- Programs aren't very useful if they don't interact with the outside world
- This interaction is called I/O
- I/O is messy in C
  - ➤Lots of terminology
  - ➤Lots of picky details

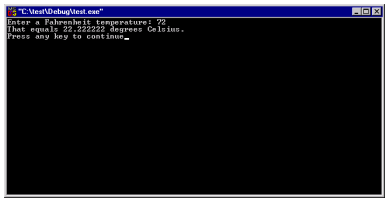
I'm sure you're tired of this slide, but here it is again...



Basic Definitions

- Input moves data into memory from the outside world (keyboard, files, network)
  - ➤Changes variables
  - ➤"read" operation
- Output moves data from memory to the outside world (monitor, files, network)
  - ➤Does not change variables
  - ➤"write" operation

ASCII Output



- What is this ASCII thing?

Example of I/O Statements

```
printf("Enter a Fahrenheit temperature: ");

scanf("%lf",  &fahrenheit);

celsius =  (fahrenheit - 32.0) * 5.0 / 9.0;

printf("That equals %f degrees Celsius.",
     celsius);
```

Terminal Input and Output

- `printf()` prints to the monitor
  - ➤Performs *output*
  - ➤Does *not* change variables
- `scanf()` reads from the keyboard
  - ➤Performs *input*
  - ➤Does change variables

`printf()` syntax

```
printf("control string", list of expressions);
```

- `printf()` has two parts:
  - ➤The control string contains all of the parts of the printf statement that are static, and has placeholders for the dynamic parts
  - ➤The list of expressions tells the computer how to fill in the placeholders

A `printf()` example

```
int numPushups;

numPushups = 5;
printf("Hello.  Do %d pushups.\n",
  numPushups);
```

`%d` is a placeholder for an int value

`\n` is an escape sequence for a newline character

What does the `\n` do?

```
printf("Hello,\nMy name is ");
printf("Isaac\n");
```

```
Hello,
My name is Isaac
```

The cursor is here now!

Placeholders and `printf()`

- Placeholders are used to print variables
  - ➤Each placeholder is matched *in order* with an expression
  - ➤The number of placeholders and expressions must match
  - ➤ The types of the placeholders and the expressions must match

`printf()` placeholders

- Each type has a corresponding placeholder:
  - ➤integer: %d (think "decimal")
  - ➤double: %f (think "floating-point")
  - ➤char: %c (this one makes sense)

`printf()` example

```
int numDogs;
double dogAge;
numDogs = 2;
dogAge = 7.6;
printf("My %d dogs are %f years
  old.\n", numDogs, dogAge);
```

My 2 dogs are 7.600000 years old.

Formatting Output

- We can format output is several ways:
  - Control the number of decimals:
    - (7.6 vs. 7.600000)
  - Control total width:
    - 7.6 vs. _____7.6
  - Use different number formats:
    - Scientific notation
    - Octal
  - etc.

Formatting Output

- Controlling the width:
  - %6d `_ _ _ _ _ 3`
  - %6f `7 . 6 0 0 0`
  - %6c `_ _ _ _ _ a`
- Controlling the number of decimals
  - %6.2f `_ _ 7 . 6 0`
  - %-6.2f `7 . 6 0 _ _`

`scanf()` syntax

```
scanff("control string", list of &variables);
```

- `scanf()` has two parts:
  - The control string contains placeholders for the values to be read
  - The list of &variables tells the computer where to put the read values

A `scanf()` example

```
int usersAge;

scanf("%d", &usersAge);
```

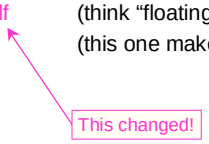
`%d` is a placeholder for an int value

If You Forget the '&'

It will compile, but when you run it...

`scanf()` placeholders

- Each type has a corresponding placeholder:
  - integer: %d (think "decimal")
  - double: %lf (think "floating-point")
  - char: %c (this one makes sense)



Whitespace

- The space (' '), tab ('\t'), and newline ('\n') are called whitespace
- Whitespace is skipped when reading integers and doubles
- Whitespace is not skipped when reading characters!
  - They *are* characters, after all

Multiple Inputs

- Just like multiple outputs:
  - The types of the placeholders and variables must match
  - The number and order must match

```
int age;
double height;
scanf("%d %lf", &age, &height);
```


Format Items Summary

Type	scanf()	printf()
char	%c	%c
int	%d	%d
double	%lf	%f

- If the types don't match:
  - printf: garbled output
  - scanf: unpredictable errors (and don't forget the &!)

What Happens If...

- What if the user enters the wrong type?
  - Things get screwed up!
  - We should handle these errors
  - We'll learn how later, for now we'll assume that the user makes no mistakes
    - Bad assumption! Users always make mistakes. (Or they're malicious.)

Syntax

- Syntax is what defines a valid program
  - If there's a syntax error, the compiler will catch it and complain
  - Punctuation
  - Reserved words
  - Undeclared variables
  - etc.

Semantics

- Semantics refers to what the program means
  - ➤Also called logic errors
  - ➤The compiler cannot catch these errors!
  - ➤Hard to find!

Find and Classify the Errors

```
#include <stdio.h>
int main(void){
  double far, cel;
  scanf('%lf', far);
  cel = (far - 32.0) * 5.0 / 9.0
  printf("Celsius is %f\n", far);
  return 0;
}
```


Find and Classify the Errors

```
#include <stdio.h>;
int main(void){
  int feet, inches;
  scanf("%lf", &feet);
  printf("That's %d inches\n",
     feet*13);
  return 0;
};
```


More on Initializing Variables

- We've seen
  - ➤Assignment statements
  - ➤scanf
- Can also initialize while declaring:

```
int foo = 7;
float bar = 6.5, blargh = 9.9;
```


Initialization Quiz

```
int main (void){           /*line 1*/
   int a, b, c, d=10;      /*line 2*/
   b=5;                    /*line 3*/
   d=6;                    /*line 4*/
   scanf("%d %d", &b, &c); /*line 5*/
   …
}
```

Q: Where is each of a, b, c, and d initialized?