

# Computer Science & Engineering 150A

## Problem Solving Using Computers – Laboratory

### Lecture 06 – Modular Programming

Derrick Stolee

Spring 2009

# Break it up!

CSCE150A

Modularization

Methods to  
Help

Examples

- Big programs get BIG.
- Most software is many millions of lines of code.
- Organization is important

# Let's think a bit smaller

CSCE150A

Modularization

Methods to  
Help

Examples

- Separate functionality into smaller chunks
  - Use functions whenever possible
  - Avoids repeated code
  - Sometimes requires more “communication”
- Organize connections between chunks
  - How does this statement get this value?
  - Should this loop contain a function call, or should the function contain the loop?
  - Should I group these together, or do them separately?

- Usually, functions have a single return value.
- This is the only way to tell the “caller” any information\*
- Parameters are *pass by value*, that is: values are copied.
- Consider the following definition:

```
int func(int a) { a = 5; return 0; }
```

- What happens with the following code in main():

```
int b=1; int c = func(b);
```

- We can pass values back to the caller more than just return!
- A parameter given by “type \* name” is called a *pointer*.
- This is called *pass by reference*.
- We can set pointer values using  $(*name) = \underline{\hspace{2cm}}$
- We pass an *address* to the function in the call using `&variable`.  
(see drawing on board)

# Example: Getting multiple values

- See code sample `ch6sam1.c`
- Calculates two properties of a rectangle: perimeter and area.
- Can't use a single return value!
- Solution: Use pointers!

# Example: Changing multiple values

- See code sample `ch6sam2.c`
- The `buyCoffee` method returns the charge.
- Updates the total amount of coffee purchased (`numCups`)
- Checks for a coupon, or adds a coupon depending on number of cups.

# Example: Changing multiple values

CSCE150A

Modularization

Methods to  
Help

Examples

- See code sample `ch6sam3.c`
- The `swapper` method swaps the values if out of order.
- A sequence of calls to `swapper` sorts five variable values.

Defining Pointers:

```
type* name; int* ptri; double* ptrd;
```

Setting Pointers:

```
int* point_to_i = &i;
```

Passing Pointers:

```
function(&var);
```

Accessing Pointer Values:

```
(*ptrToI) = 5;  
int ival = (**ptrToPtrToI);
```