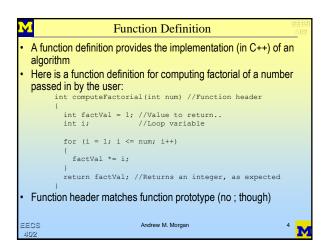
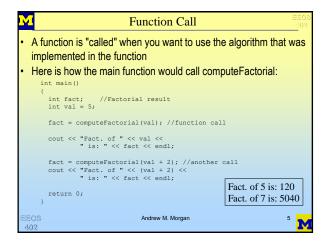


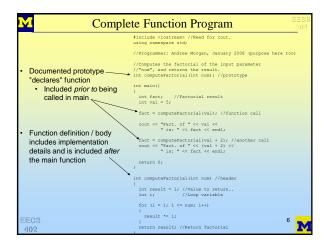
```
Function Prototype
   A function prototype "declares a function"
       C++ must see the prototype before the function call can be checked for proper syntax!
   Prototype is what the user will look at when they want to call the function

    Therefore, function prototypes must include a comment to help user understand the function and

   Here is the function prototype for the factorial function
            //Computes the factorial of the input parameter
            //"num", and returns the result.
           int computeFactorial(int num);
  The function is called "computeFactorial"
       Takes in one integer value from the calling function as a parameter
       Returns a computed integer value to the calling function
    - Prototype is documented
   Style: The function name must be descriptive of its purpose!
   Style: The function name must be named with a verb!
   Style: The function prototype must be clearly documented with comments!
                                      Andrew M. Morgan
FFCS
                                                                                      3
```







```
Order is Important! (version 2)
 int result = 1; //Value to return..
int i; //Loop variable
                                                                          This version will build and run as expected
                                                                         (see the important note below though!).
 for (i = 1; i <= num; i++)
                                                                         C++ sees the function definition of
                                                                         computeFactorial first, so it effectively 
"declares" the function when the definition is seen.
   result *= i;
  return result; //Return factorial
                                                                         While this technically works, we will NOT use this approach in this class.
int main()
 int fact; //Factorial result
int val = 5;
                                                                         Important note: For this class, you must specify a documented prototype BEFORE main, and the function implementation must be included AFTER main.
 fact = computeFactorial(val); //function call
 cout << "Fact. of " << val << " is: " << fact << end);
```

```
Multiple Parameters
   Often, multiple values from the calling function are needed
  Any number of parameters can be passed in to a function
                                                     int addNums(int valA, int valB, int valC)
#include <iostream> //Need for cout.
using namespace std;
                                                        int sumOfVals;
//Computes sum of all 3 provided values
int addNums(int valA, int valB, int valC);
                                                       sumOfVals = valA + valB + valC;
int main()
                                                       return sumOfVals;
 int num1 = 5; //Integer for test
int num2 = 3; //Integer for test
int result; //Result of call
 result = addNums(num1, 6, num2);
 cout << "Result is: " << result;
cout << endl;</pre>
                                                                      Result is: 14
 return 0;
                                          Andrew M. Morgan
                                                                                               9
```

```
Void Returning Functions

• Some functions don't need to return a value at all

- In this case, specify return type as "void" and don't return anything

#include <iostream>
using namespace std;

//Prints the main menu to the console.
void printMenu();

int msin()
{
  printMenu();

  //do other stuff
  return 0;
}

void printMenu()
{
  cout < "1. Add values" << end1;
  cout < "2. Subtract values" << end1;
  cout < "3. Quit program" << end1;
}

EECS

Andrew M. Morgan

10
```

```
Overloading Functions
  Multiple functions can have same name
   - Must have unique parameter list, though
  Function signature
   - Function name and types and order of parameters in parameter list
    - Functions must have a unique signature
   Overloading: Multiple functions with same name
                                                    //square an int, and //return the value
//square an int, and
//return the value
int squareInt(int num);
                                                    int square(int num);
                                         Not
//square a float, and
//return the value
float squareFloat(float num)
                                                   //square a float, and
//return the value
float square(float num)
                                         Overloaded
 //Draw a square on
                                                    //Draw a square on
 //the screen
                                                    //the screen
int drawSquare(int x, int y, int len, int wid);
                                                    Andrew M. Morgan
                                                                              11
```

```
Overloading Example
   overloadSum(int a, int b, int c)
                                                           int main()
loat overloadSum(float a, float b, float c)
                                                             ans = overloadSum(f1, f2, f2);
cout << ans << end1;</pre>
cout << "(f f f) version" << endl;
return (a + b + c);</pre>
                                                             ans = overloadSum(i1, i2, i2);
cout << ans << end1;</pre>
float overloadSum(int a, float b, float c)
cout << "(i f f) version" << endl;
return (a + b + c);</pre>
                                                             ans = overloadSum(i2, (float)i1, f1);
cout << ans << endl;</pre>
                                                             return 0;
                                                                                            (f f f) version
                                                                                            14.8
                                                                                            (i i i) version
                                                                                            16
                                                                                            (i f f) version
                                                                                           16.4
                                               Andrew M. Morgan
                                                                                                        12
```

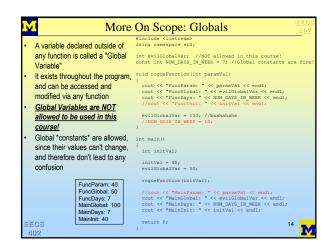
```
Some Words On Scope
Any variable declared in a function is "local" to that function
It only exists from the time it's declared until the end of the function
 - The function add4() can NOT access
                                            int add4(int foo)
   the variables bar, or result, from main().
                                               int result:

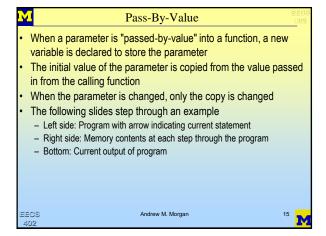
    The function main() can NOT access

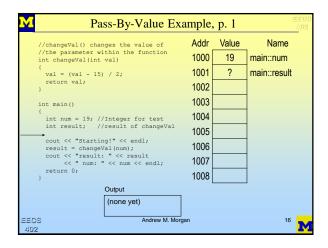
   the variables foo, or result, from add4().
                                               result = foo + 4;
                                              return result;

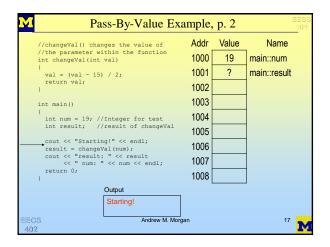
    Even though both functions have a

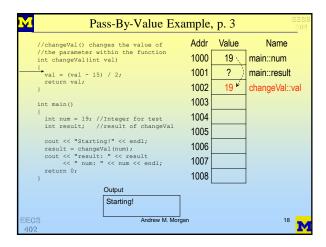
   variable called result - they are unique
   variables, in unique addresses, with
                                            int main()
   unique scopes.
                                               int bar = 7;
                                              int result;
                                              result = add4(bar);
                                              return 0;
                             Andrew M. Morgan
                                                                  13
```

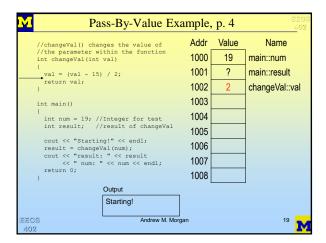


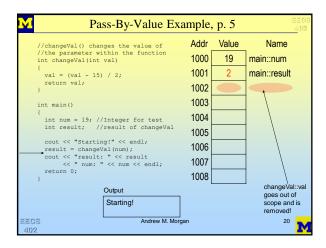


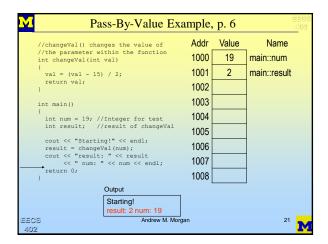


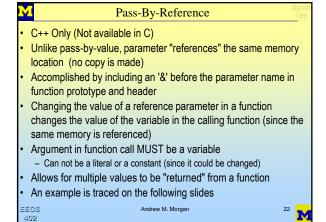


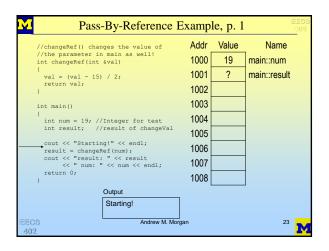


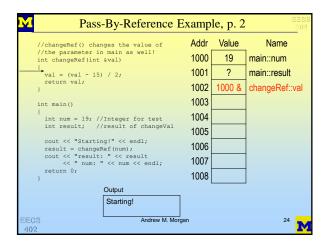


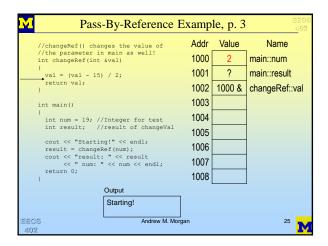


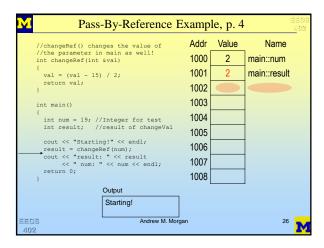


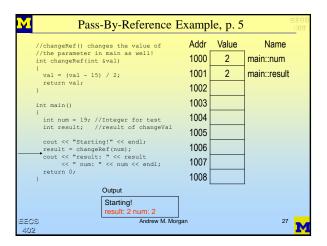












```
Swap Example, Multiple Reference Params

void swap(int &valA, int &valB) //Pass-by-reference!
{
  int temp;

  temp = valA;
  valA = valB;
  valB = temp;
}

int main()
{
  int n1 = 5;
  int n2 = 10;

  cout << "Before swap - n1: " << n1 << " n2: " << n2 << end1;
  swap(n1, n2);
  cout << "After swap - n1: " << n1 << " n2: " << n2 << end1;

  return 0;
}

Before swap - n1: 5 n2: 10
  After swap - n1: 10 n2: 5

EECS

Andrew M. Morgan

28
```

```
Advantages of Modularity

Cleaner code

A call to a function named "computeFactorial()" is compact and essentially self-documenting

A loop to compute the factorial would not be immediately clear

Non-duplication

Factorial algorithm implemented once, can be used simply by calling the function as needed

Breaks the program into smaller pieces

Real world: Write specifications and prototypes for needed functions, then distribute different functions to different people - parallel coding is faster

Easier testing

How to test one, huge, monolithic, 30,000 line program?

Modular program can be tested module by module (function by function, in this case)
```

```
Remember Short-Circuiting??
return inVal < 20;
                                              int aValue = 60;
int bValue = 30;
 ol changeValTo100(int &valToChange)
                                              if (isLessThan20(aValue) || changeValTo100(bValue)
                                                cout << "AB if expression was true!!!" << endl;
bool didChangeIt;
if (valToChange == 100)
                                              cout << "bValue is: " << bValue << endl;
  didChangeIt = false;
                                              if (isLessThan20(cValue) || changeValTo100(dValue)
                                                cout << "CD if expression was true!!!" << endl;
                                              cout << "dValue is: " << dValue << endl;
return didChangeIt;
                                         Note that dValue did NOT get updated!!
  AB if expression was true!!!
  bValue is: 100
                                          Since "isLessThan20(cValue)" evaluated to true, there's no need
  CD if expression was true!!!
                                         to evaluate "changeValTo100(dValue)" at all - the changeValTo100 function doesn't even get called in that case!
  dValue is: 30
                                         Andrew M. Morgan
                                                                                           30
```

```
M
                       Pre-Existing Functions
  C++ standard libraries contain many functions that you usually do
   not have to write algorithms for yourself
   Example: Many math related functions in a standard math library
    - Must #include <cmath> to access these functions (and include the "using
      namespace std;" line)
   - Once the library is #include'd, you may utilize functions from the library
        · If you don't #include <cmath>, then you should get an error when trying to use
          functionality from the math library!
   - Some available functions:

    double sin(double x)

    double cos(double x)

       · double pow(double base, double exponent)

    double sqrt(double x)

       • Etc...
                                                                        31
                                 Andrew M Morgan
```

```
Using Standard Math Library, Example Output

#include <iostream>
#include <cmath>
using namespace std;

int main()

{
    double checkVal;
    double checkVal;
    checkVal = 4;
    resultVal = pow(checkVal, 3.0);
    cout << checkVal << "^3.0 = " << resultVal << endl;

checkVal = 65;
    resultVal = sqtt(checkVal);
    cout << 'checkVal = ("^3.0 = " << resultVal << endl;

checkVal = 65;
    resultVal = sqtt(checkVal);
    cout << "sqrt of " << checkVal << " = " << resultVal << endl;

return 0;
}

4^3.0 = 64
sqt of 65 = 8.06226
```

