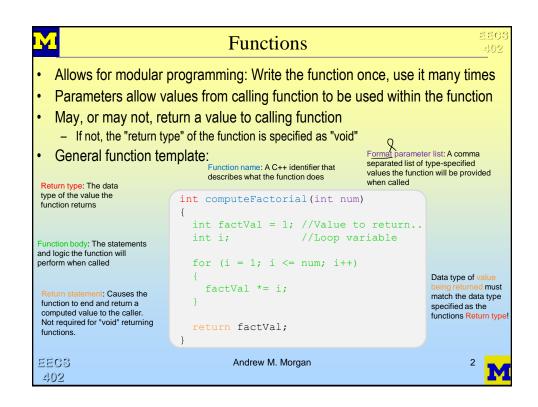


EECS402 Lecture 02

Andrew M. Morgan

Savitch Ch. 3-4
Functions
Value and Reference Parameters

Andrew M. Morgan





Function Prototype

크EC8

- 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 what it does
- 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!

EECS 402 Andrew M. Morgan

3



M

Function Definition

크로CS

- A function definition provides the implementation (in C++) of an algorithm
- Here is a function definition for computing factorial of a number passed in by the user:

• Function header matches function prototype (no; though)

402

Andrew M. Morgan

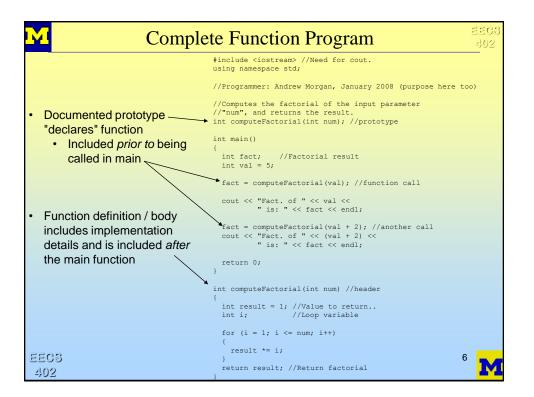
ı



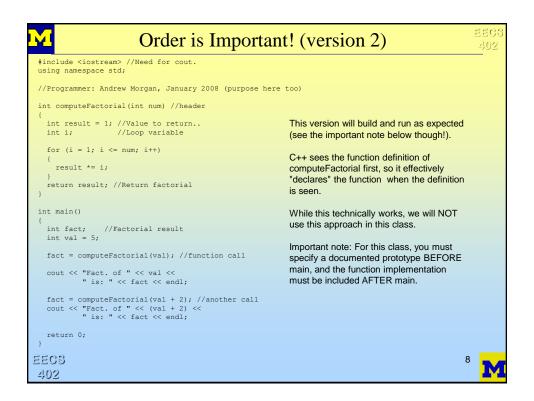
```
Function Call

    A function is "called" when you want to use the algorithm that was

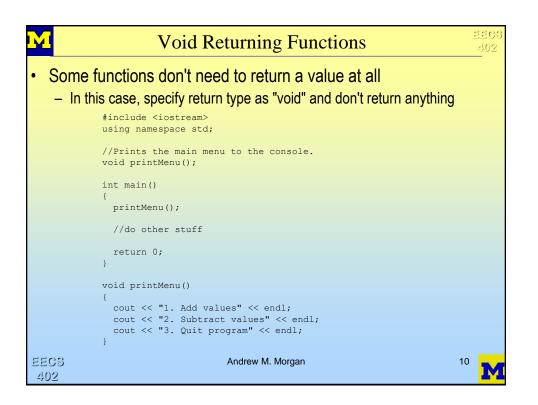
  implemented in the function
  Here is how the main function would call computeFactorial:
    int main()
      int fact;
                  //Factorial result
      int val = 5;
      fact = computeFactorial(val); //function call
      cout << "Fact. of " << val <<
             " is: " << fact << endl;
      fact = computeFactorial(val + 2); //another call
      Fact. of 5 is: 120
      return 0;
                                                 Fact. of 7 is: 5040
EECS
                            Andrew M. Morgan
403
```



```
Order is Important!
 #include <iostream> //Need for cout.
using namespace std;
 //Programmer: Andrew Morgan, January 2008 (purpose here too)
                                                       error: 'computeFactorial' was not declared in this scope
   int fact;
int val = 5;
                                                         fact = computeFactorial(val); //function call
   fact = computeFactorial(val); //function call
   cout << "Fact. of " << val <<
           " is: " << fact << endl;
   fact = computeFactorial(val + 2); //another call
   cout << "Fact. of " << (val + 2) << " is: " << fact << endl;
                                                          Without a prototype before main, C++ gets
                                                          to the line with the function call and says
  return 0;
                                                          "hmmm, looks like a typo since I've never
                                                          heard of "computeFactorial" before
int computeFactorial(int num) //header
   int result = 1; //Value to return..
                   //Loop variable
  for (i = 1; i <= num; i++)
    result *= i;
   return result; //Return factorial
EECS
403
```

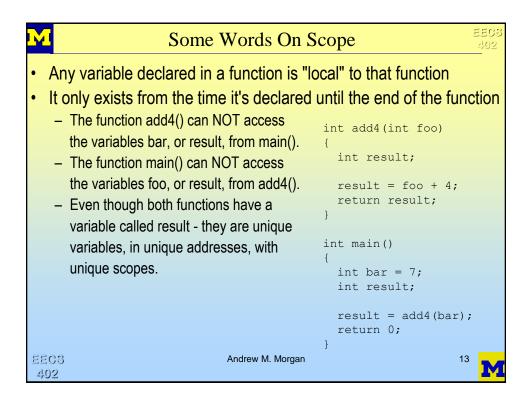


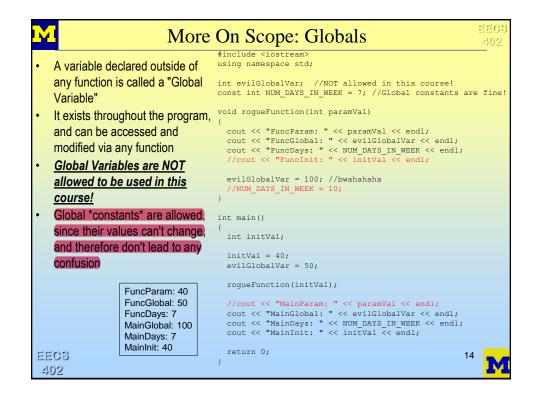
```
Multiple Parameters
 Often, multiple values from the calling function are needed
  Any number of parameters can be passed in to a function
#include <iostream> //Need for cout.
                                             int addNums(int valA, int valB, int valC)
using namespace std;
                                               int sumOfVals;
//Computes sum of all 3 provided values
int addNums(int valA, int valB, int valC);
                                               sumOfVals = valA + valB + valC;
                                               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;</pre>
 cout << endl;</pre>
                                                            Result is: 14
 return 0;
EECS
                                    Andrew M. Morgan
403
```

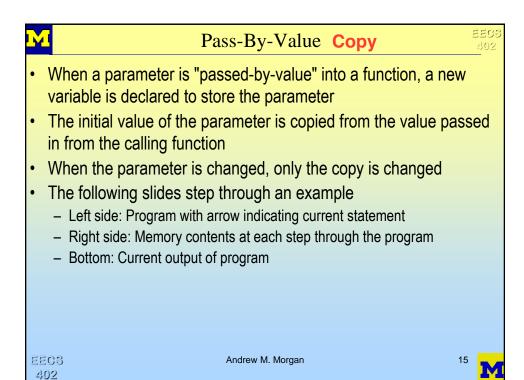


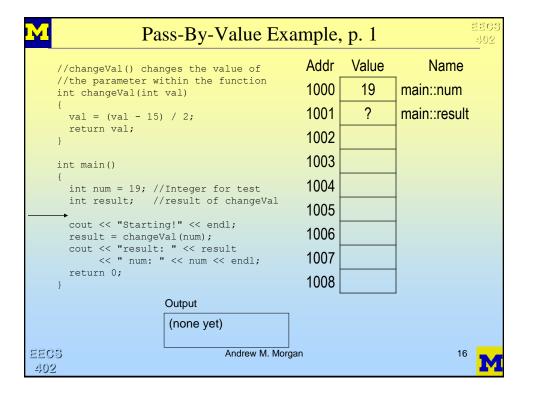
Overloading Functions Multiple functions can have same name - Must have unique parameter list, though Function signature - Function name and types and order of parameters in parameter list Functions must have a unique signature Overloading: Multiple functions with same name //square an int, and //square an int, and //return the value //return the value int squareInt(int num); int square(int num); Not Overloaded Overloaded //square a float, and //square a float, and //return the value //return the value float squareFloat(float num) float square(float num) //Draw a square on //Draw a square on //the screen //the screen int drawSquare(int x, int y, int square(int x, int y, int len, int wid); int len, int wid); EECS Andrew M. Morgan 11 703

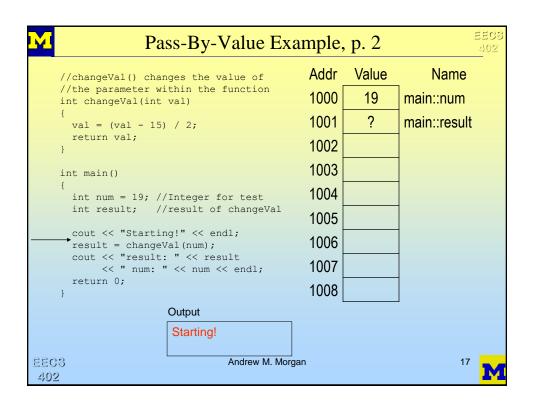
```
Overloading Example
                                                int main()
int overloadSum(int a, int b, int c)
 cout << "(i i i) version" << endl;
                                                  float ans;
  return (a + b + c);
                                                  float f1 = 6.4;
                                                  float f2 = 4.2;
                                                  int i1 = 4;
int i2 = 6;
float overloadSum(float a, float b, float c)
                                                  int
 cout << "(f f f) version" << endl;</pre>
                                                  ans = overloadSum(f1, f2, f2);
 return (a + b + c);
                                                  cout << ans << endl;
                                                  ans = overloadSum(i1, i2, i2);
float overloadSum(int a, float b, float c)
                                                  cout << ans << endl;
 cout << "(i f f) version" << endl;</pre>
                                                  ans = overloadSum(i2, (float)i1, f1);
 return (a + b + c);
                                                  cout << ans << endl;
                                                  return 0;
                                                                           (f f f) version
                                                                            14.8
                                                                           (i i i) version
                                                                           (i f f) version
                                                                           16.4
EECS
                                       Andrew M. Morgan
402
```

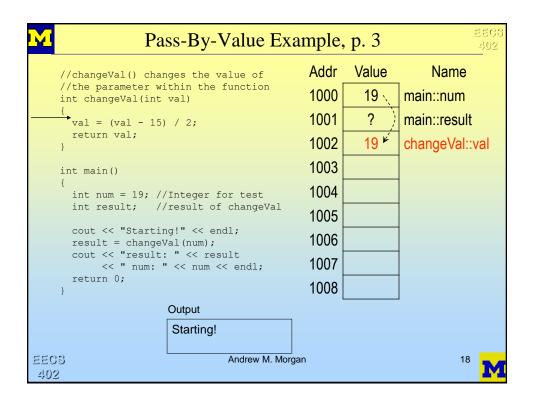


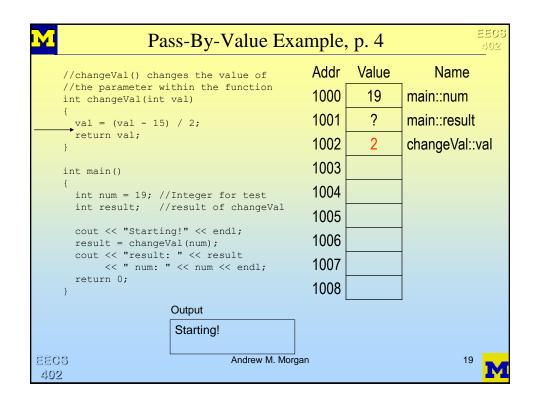


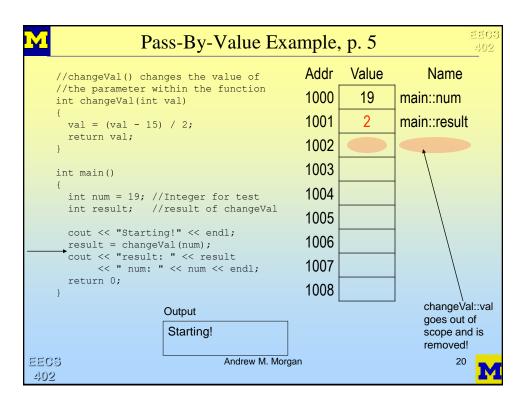


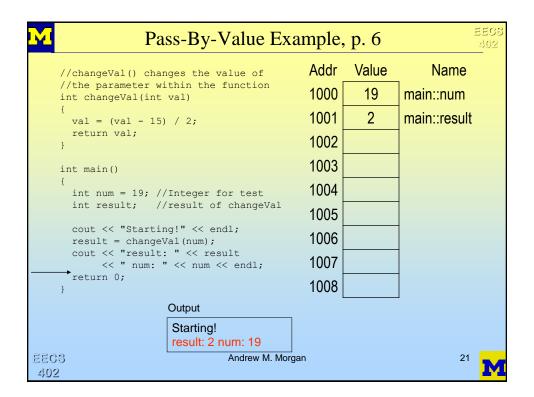












M

Pass-By-Reference

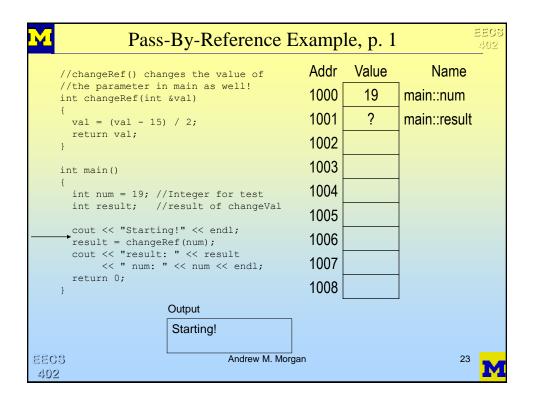
로로() 402

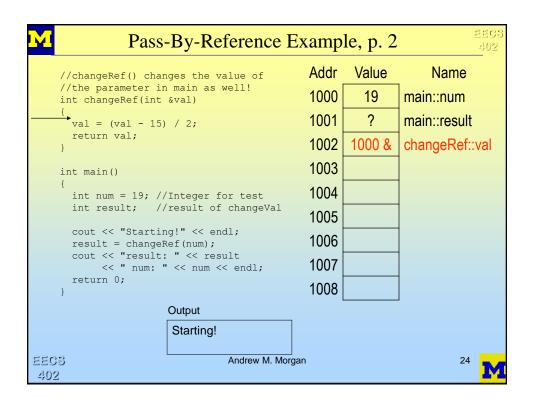
- C++ Only (Not available in C)
- Unlike pass-by-value, parameter "references" the same memory location (no copy is made)
- Accomplished by including an '&' before the parameter name in function prototype and header
- Changing the value of a reference parameter in a function changes the value of the variable in the calling function (since the same memory is referenced)
- Argument in function call MUST be a variable
 - Can not be a literal or a constant (since it could be changed)
- Allows for multiple values to be "returned" from a function
- An example is traced on the following slides

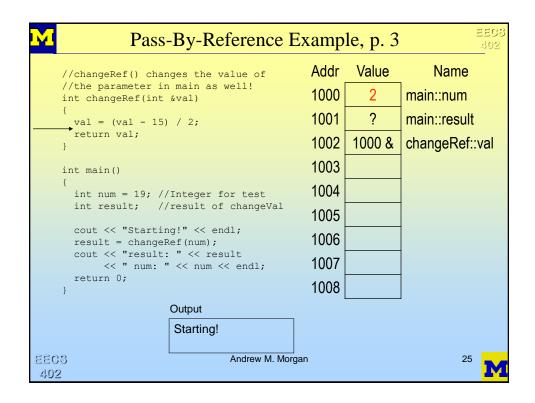
402

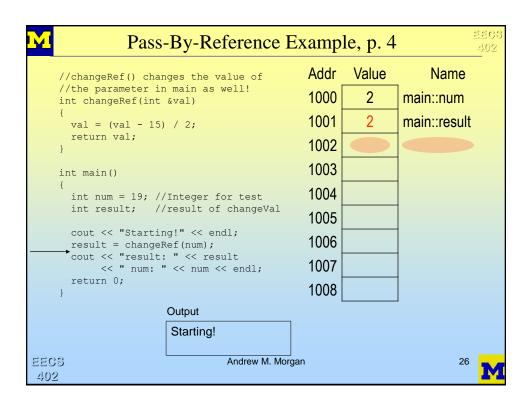
Andrew M. Morgan

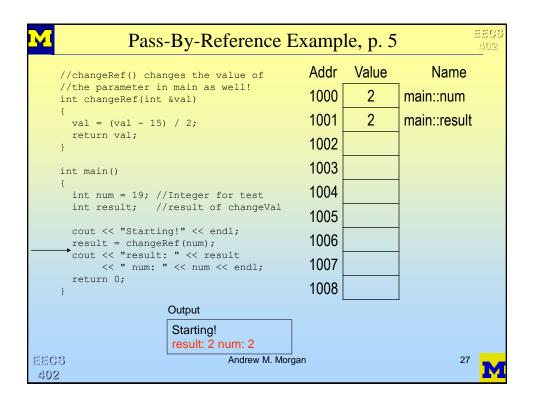












```
Swap Example, Multiple Reference Params
  void swap(int &valA, int &valB) //Pass-by-reference!
    int temp;
    temp = valA;
    valA = valB;
    valB = temp;
  int main()
    int n1 = 5;
    int n2 = 10;
    cout << "Before swap - n1: " << n1 << " n2: " << n2 << endl;</pre>
    swap(n1, n2);
    cout << "After swap - n1: " << n1 << " n2: " << n2 << endl;</pre>
    return 0;
                                              Before swap - n1: 5 n2: 10
  }
                                              After swap - n1: 10 n2: 5
EECS
                              Andrew M. Morgan
402
```



Advantages of Modularity

550: 402

- Cleaner code
 - A call to a function named "computeFactorial()" is compact and essentially selfdocumenting
 - 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)

402 402 Andrew M. Morgan



```
Remember Short-Circuiting??
bool isLessThan20(int inVal)
                                             int main()
 return inVal < 20;
                                               int aValue = 60;
                                               int bValue = 30;
bool changeValTo100(int &valToChange)
                                               if (isLessThan20(aValue) || changeValTo100(bValue))
                                                 cout << "AB if expression was true!!!" << endl;</pre>
 bool didChangeIt;
  if (valToChange == 100)
                                               cout << "bValue is: " << bValue << endl;
   didChangeIt = false;
                                               int cValue = 10;
                                               int dValue = 30;
                                               if (isLessThan20(cValue) || changeValTo100(dValue))
   valToChange = 100;
didChangeIt = true;
                                                 cout << "CD if expression was true!!!" << endl;</pre>
                                               cout << "dValue is: " << dValue << endl;</pre>
 return didChangeIt;
                                               return 0;
                                           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
    dValue is: 30
                                           changeValTo100 function doesn't even get called in that case!
EECS
                                          Andrew M. Morgan
 402
```



Pre-Existing Functions



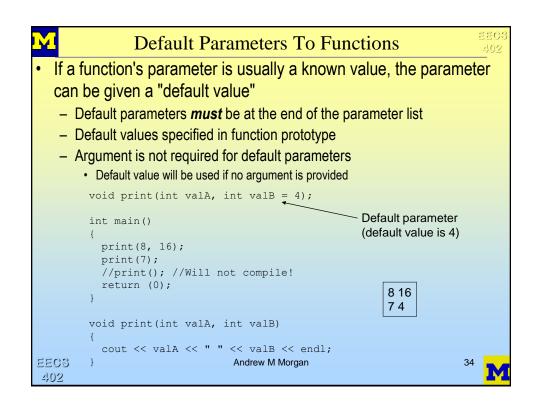
- C++ standard libraries contain many functions that you usually do not have to write algorithms for yourself
- Example: Many math related functions in a standard math library
 - Must #include <cmath> to access these functions (and include the "using namespace std;" line)
 - Once the library is #include'd, 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...

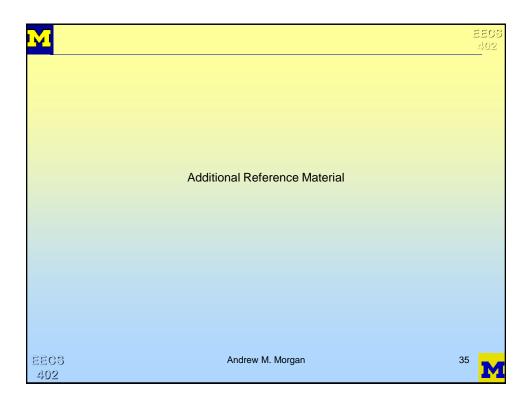
402 402 Andrew M Morgan



```
Using Standard Math Library, Example Output
  #include <iostream>
  #include <cmath> ←
                                                Note: this is *required*, even if you happen to use an
  using namespace std;
                                                IDE that allows you "get away without it"
  int main()
    double checkVal;
    double resultVal;
    checkVal = 4;
    resultVal = pow(checkVal, 3.0);
    cout << checkVal << "^3.0 = " << resultVal << endl;</pre>
    checkVal = 65;
    resultVal = sqrt(checkVal);
    cout << "sqrt of " << checkVal << " = " << resultVal << endl;</pre>
    return 0;
                                                  4^3.0 = 64
                                                  sqrt of 65 = 8.06226
EECS
                                    Andrew M Morgan
402
```

```
Using Standard Math Library, Example Output
  #include <iostream>
  //#include <cmath> +
                                                     Note: this is commented for this example!
  using namespace std;
  int main()
    double checkVal;
    double resultVal;
    checkVal = 4;
    resultVal = pow(checkVal, 3.0);
    cout << checkVal << "^3.0 = " << resultVal << endl;</pre>
    checkVal = 65;
    resultVal = sqrt(checkVal);
    cout << "sqrt of " << checkVal << " = " << resultVal << endl;</pre>
    return 0;
                                    LinuxPrompt> g++ mathstuff.cpp -o mathstuff
                                    mathstuff.cpp: In function 'int main()':
                                    mathstuff.cpp:11:32: error: 'pow' was not declared in this scope
 These functions aren't available unless
                                      resultVal = pow(checkVal, 3.0);
 you #include <cmath>!
                                    mathstuff.cpp:15:28: error: 'sqrt' was not declared in this scope
                                      resultVal = sqrt(checkVal);
EECS
                                       Andrew M Morgan
703
```





M

Modular Testing - Driver Programs

물론C3 402

- Driver programs allow you to test a newly written function
- The purpose of a driver program is simply to call your function and output some results to check correctness
- Most main programs in lectures so far have been driver programs to demonstrate the use of other functions
- Especially helpful when the function you are writing is buried deep in some million line project
 - If adding functionality to a simulation that takes 12 hours to run, you don't want to have to run 50 test cases (25 days) using the entire simulation just to test one function

402

Andrew M. Morgan



Modular Testing - Stubs

로**르**C8 402

- Stubs allow you to test a program that is unfinished
- If waiting for someone else to finish an important function, you would still want to do some testing.
- Provide the function prototype and a "dummy" body
 - Stub does not return actual value that the function will, but allows you to call the function as if it were complete, for testing.
- This simply allows you to have the function defined in some way so when the function is ready, the stub is simply replaced with the actual function.

三三CS 402 Andrew M. Morgan

