

IVI.

Functions

- Allows for modular programming
 - Write the function once, call it many times
 - Group similar things together
- Parameters allow values from calling function to be used within the function
- May, or may not, return a value to calling function
- · General function template:

```
return_type function_name(formal parameter list)
{
  function code...
}
```

Andrew M. Morgan

M

I VI

Function Prototype

- A function prototype "declares a function"
- Provides user with information about the function.
 - Name of function, what values (and types) need to be passed in, what type to expect the function to return
- Here is a function prototype for a factorial function int computeFactorial(int num);

- The function is called "computeFactorial"
 - Takes in one integer value from the calling function as a parameter
 - Returns an integer value to the calling function
- Note: The function name should be *descriptive* of its purpose!

Andrew M. Morgan

M

```
Function Definition

• 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:

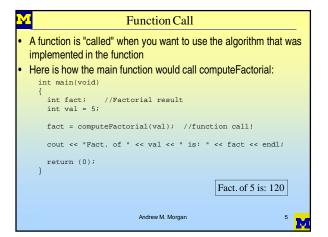
int computeFactorial(int num) //Function header

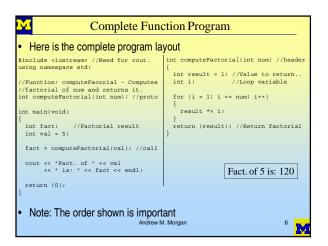
{
int result = 1; //Value to return..
int i; //Loop variable

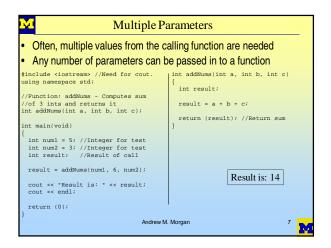
for (i = 1; i <= num; i++)

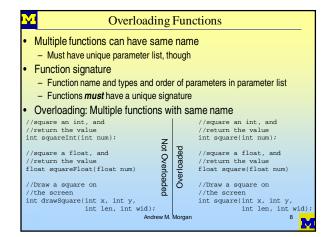
{
   result *= i;
   }
   return (result); //Returns an integer, as expected
}

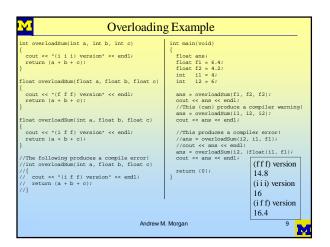
• Function header matches function prototype (no; though)
```





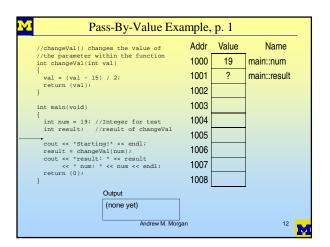


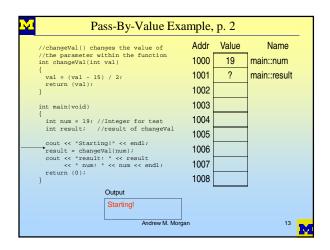


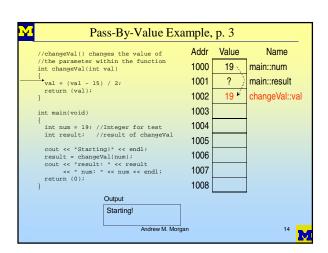


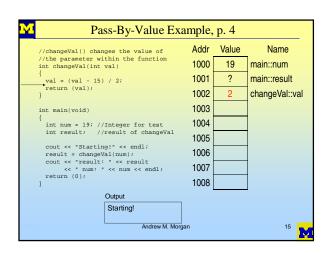
Some Words On Scope Any variable declared in a function is "local" to that function It only exists from the time it's declared until the end of the function - The function add4() can NOT access int add4(int foo) the variables bar, or result, from main(). int result; - The function main() can NOT access the variables foo, or result, from add4(). result = foo + 4; return (result); - Even though both functions have a variable called result - they are unique int main(void) variables, in unique addresses, with unique scopes. int bar = 7; int result; result = add4(bar); return (0); Andrew M. Morgan 10

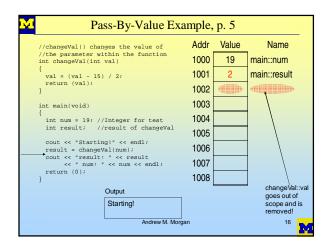
Pass-By-Value • When a parameter is "passed-by-value" into a function, a new variable is declared to store the parameter • The initial value of the parameter is copied from the value passed in from the calling function • When the parameter is changed, only the copy is changed • The following slides step through an example - Left side: Program with arrow indicating current statement - Right side: Memory contents at each step through the program - Bottom: Current output of program

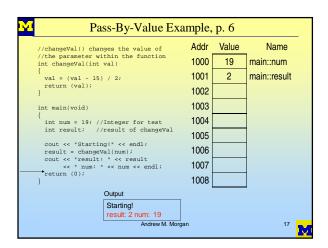




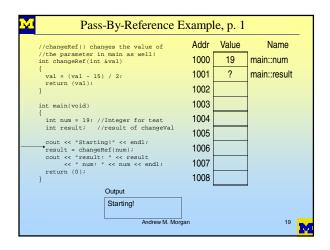






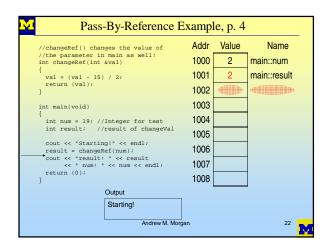


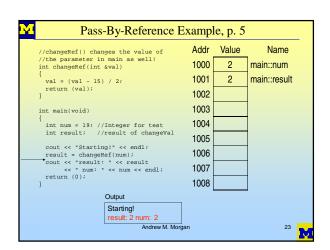
Pass-By-Reference C++ Only (Not available in C) Unlike pass-by-value, parameter "references" the same memory location (no copy is made) Accomplished by including an '&' before the parameter name in function prototype and header Changing the value of a reference parameter in a function changes the value 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



Pass-By-Reference Example, p. 2				
//changeRef() changes the value of	Addr	Value	Name	
<pre>//the parameter in main as well! int changeRef(int &val)</pre>	1000	19	main::num	
val = (val - 15) / 2;	1001	?	main::result	
return (val); }	1002	1000 &	changeRef::val	
int main(void)	1003			
{ int num = 19; //Integer for test	1004			
int result; //result of changeVal	1005			
<pre>cout << "Starting!" << endl; result = changeRef(num);</pre>	1006			
cout << "result: " << result	1007			
return (0); }	1008			
Output				
Starting!				
Andrew M. Morga	an		20	

Pass-By-Reference Example, p. 3				
//changeRef() changes the value of	Addr	Value	Name	
<pre>//the parameter in main as well! int changeRef(int &val)</pre>	1000	2	main::num	
val = (val - 15) / 2;	1001	?	main::result	
return (val); }	1002	1000 &	changeRef::val	
int main(void)	1003			
int num = 19; //Integer for test	1004			
int result; //result of changeVal	1005			
<pre>cout << "Starting!" << endl; result = changeRef(num);</pre>	1006			
cout << "result: " << result	1007			
return (0); }	1008			
Output				
Starting!				
Andrew M. Morg	an		21	





```
Swap Example, Multiple Reference Params

void swap(int &a, int &b) //Pass-by-reference!
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main(void)
{
    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

Andrew M. Morgan 24
```

Advantages of Modularity Cleaner code - A call to a function called "computeFactorial()" is compact and essentially self-documenting - A loop to compute the factorial would not be immediately clear 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) Andrew M. Morgan 25 Modular Testing - Driver Programs 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 Andrew M. Morgan Modular Testing - Stubs 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. Andrew M. Morgan