

CSCE 120/121

Introduction to Program Design & Concepts

Software Development Process

Dr. Tim McGuire

Grateful acknowledgment to Dr. Philip Ritchey and Dr. Michael Moore for some of the material on which these slides are based.

Software Development Process

- Figure out **what** needs to be done (requirements/specifications)
- Break the problem into smaller problems to be solved
- What is known / given?
- What can/must be assumed?
- What is being asked for?
- Express requirements as tests

1. Analysis

2. Design

3. Implement

4. Test

5. Repeat

Software Development Process

- Decide **how** to solve the problem
- Overall structure of system
- How do the parts interact?
- What is the flow?
- What libraries/tools/frameworks can help solve the problem?

1. Analysis

2. Design

3. Implement

4. Test

5. Repeat

Software Development Process

- Focus on problem solving, *not* on coding details.
- Should be readable by anyone, even if they are not familiar with C++ (or any programming language).
- Algorithm
 - A detailed sequence of actions to perform to accomplish some task.
([Dictionary.com](https://www.dictionary.com))

1. Analysis

2. Design

3. Implement

4. Test

5. Repeat

Software Development Process



- Tools
 - Pseudocode
 - Simple English
 - Semi-formal, precise
 - Flowchart
 - Graphical
 - Formal

1. Analysis

2. Design

3. Implement

4. Test

5. Repeat

Software Development Process

- Know **what** to do and have plan for **how** to do it.
- Write tests (using test cases from Analysis phase)
- Write code to pass the tests

1. Analysis

2. Design

3. Implementation

4. Test

5. Repeat

Software Development Process

- Test
 - Run tests
- Debug
 - Fix code that fails tests

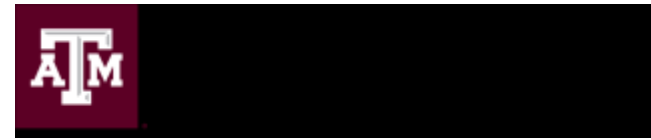
1. Analysis

2. Design

3. Implement

4. Test

5. Repeat



Software Development Process

- Most important thing to remember
- Repeat!!!
 - Go back to any step at any time

1. Analysis

2. Design

3. Implement

4. Test

5. Repeat

Problem Solving

Polya's Four Steps

Polya's 4-Step Problem Solving Process



1. Understand the problem
2. Make a plan
3. Execute the plan
4. Evaluate the result

1. Understand the Problem

- Do you understand all the words used in stating the problem?
- What are you asked to find or show?
- Can you restate the problem in your own words?
- Can you think of a picture or diagram that might help you understand the problem?
- Is there enough information to enable you to find a solution?

2. Make a Plan

- Choose an appropriate strategy for solving the problem.
- Skill at choosing strategies improves with practice
- Examples:
 - Guess and check
 - Look for a pattern
 - Make an orderly list
 - Draw a picture
 - Eliminate possibilities
 - Solve a simpler problem
 - Use symmetry
 - Use a model
 - Consider special cases
 - Work backwards
 - Use direct reasoning
 - Use a formula
 - Solve an equation
 - Be ingenious

3. Execute the Plan

- Requires patience and persistence (sometimes “grit”)
- Stick to the plan until it stops working, then choose a different plan
 - Don’t give up too soon.
- Check each step of the plan for correctness

4. Evaluate the Result

- Take time to reflect on what you have done
- What worked?
- What didn't work?
- Can you see a better way to solve it?
 - Simpler?
 - More elegant?
- Can you apply the solution to another problem?

A First Look at Functions

Declaring and Defining a Function

- General form:
 - `return_type name (formal parameters); // declaration`
 - `return_type name (formal parameters) body // definition`
- Formal parameters, format is
 - `type1 name1, type2 name2, ...`
- Make return type void if you don't want to return anything
- *body* is a block (or a try block)
 - Example:

```
double f(int a, double d) {  
    return a*d;  
}
```


Calling a Function

- Recall: `double f(int a, double d) { return a*d; }`
- To call a function: `name (actual arguments)`
- Actual arguments format is
argname₁, argname₂, ...

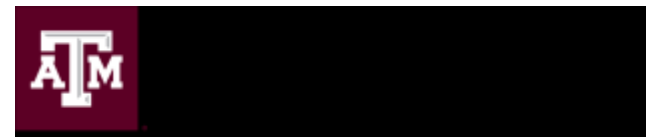
do not include types!

- Example:

```
int x = 2;  
double y = 5.0;  
cout << f(x,y); // prints out 10.0
```

Note on Terminology

Definition	Stroustrup	zyBook	McGuire
In the declaration/definition, the type and names of values to be passed into the function.	Formal Argument	Parameter	Formal Parameter
In the call, the values/variable actually passed into the function.	Actual Argument	Argument	Actual Argument



Function Placement

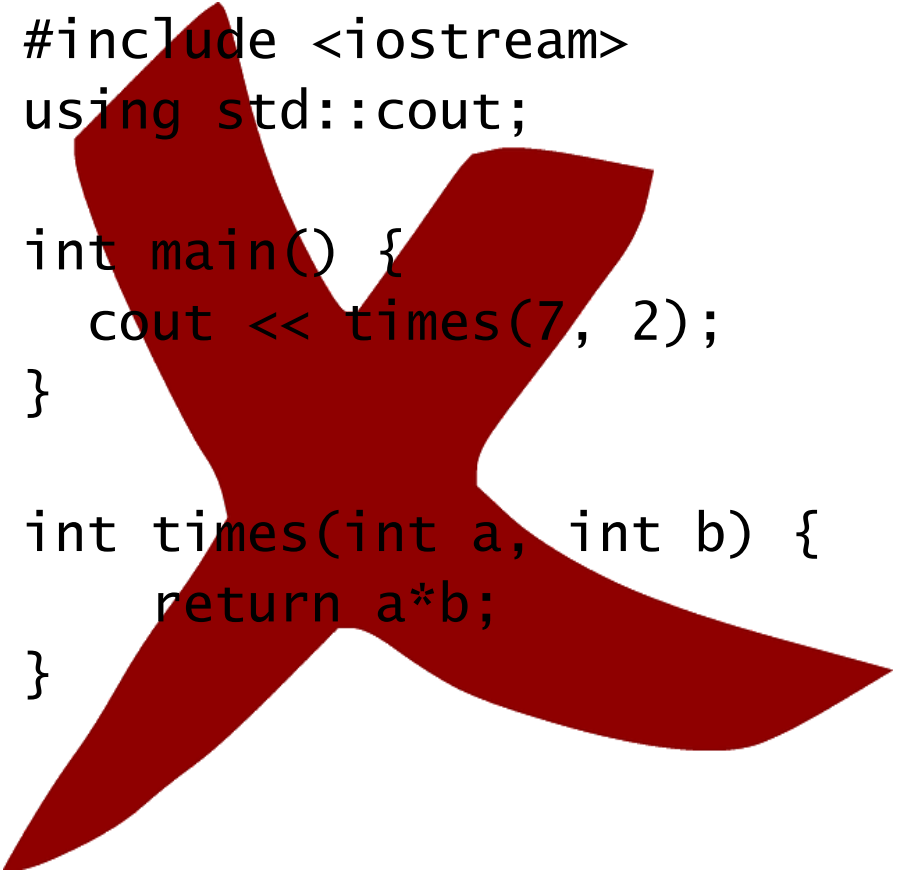
- Functions must be declared in the .cpp file before they are called.
 - So if a function is called in the main function, then it must be declared before main.
 - It can be *defined* later, as long as it is *declared* before it is called.
- You cannot define functions inside other functions
- We'll talk about defining functions inside classes later

Function Placement



}

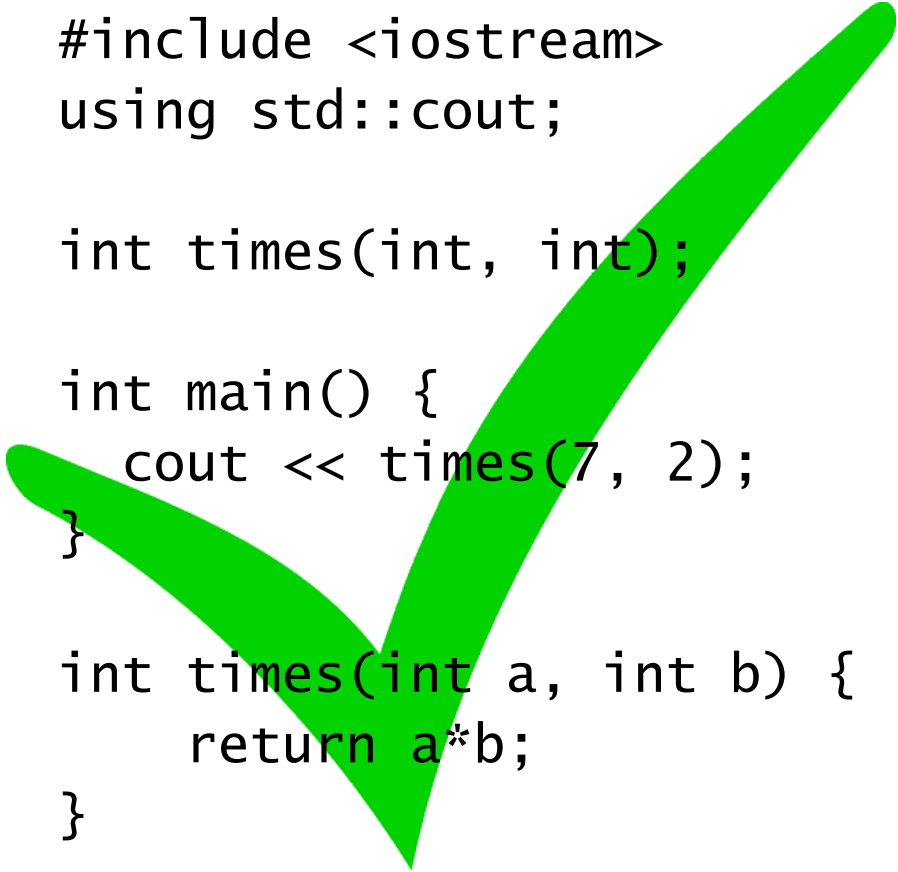
Function Prototype (Alternative)



```
#include <iostream>
using std::cout;

int main() {
    cout << times(7, 2);
}

int times(int a, int b) {
    return a*b;
}
```



```
#include <iostream>
using std::cout;

int times(int, int);

int main() {
    cout << times(7, 2);
}

int times(int a, int b) {
    return a*b;
}
```