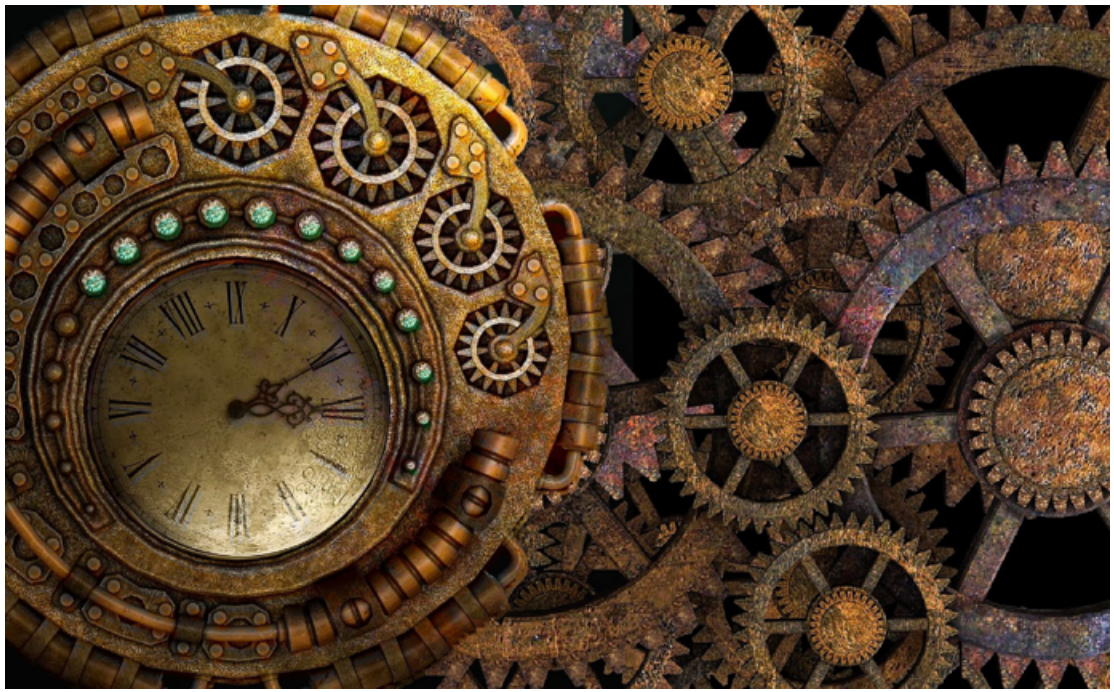
CS 161
Introduction to CS I
Lecture 8

- What do we do when things go wrong?
- How can we use the same code in multiple places?

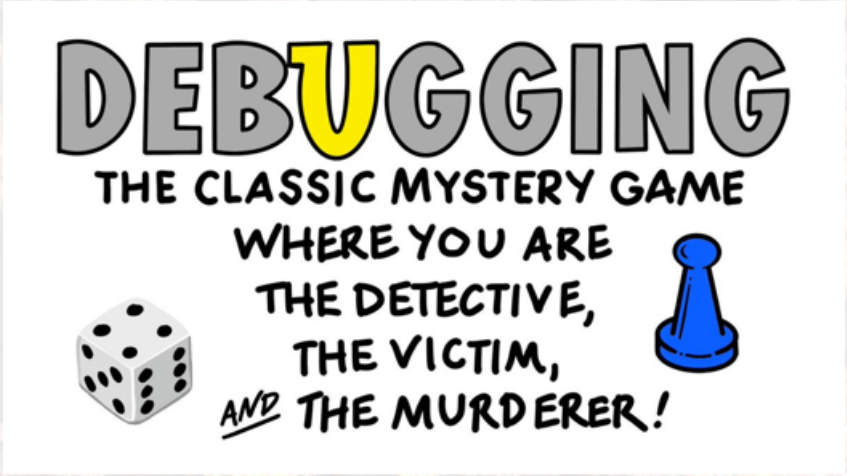


More on testing

- What is an ”edge” case?
  - `if (age < 25)`
  - `if (0 < age && age < 25)`
  - `if (dice_roll % 2 == 0)`
- Utility of test cases
  - `if (age < 25)`
  - Test: 1, 2, 3, 4, 5…?

Coding error (bug) types

- Syntax error
  - Incorrect use of C++ (grammar)
  - **How do you find these?**
- Logic error
  - Program does not do the task correctly
  - **How do you find these?**
- Execution/runtime error
  - Program stops unexpectedly
  - **How do you find these?**



Error examples

- Syntax
  - Missing main()
  - Missing semi-colon
  - Misspelled identifiers: myVariable vs. myvariable
  - Missing **or extra** quotation mark, curly brace, parenthesis
  - Use of single quotes instead of double quotes: 'CS 161'
- Logic
  - Incorrect loop conditions – e.g., unintended infinite loop
  - Increment past largest value that can be stored (overflow)
  - Missing ‘else’ or ’default’ case

Error examples

- Runtime
  - Segmentation fault, core dump (memory access failure)
  - Read from a file that doesn't exist
  - Divide by zero

Bug detection tools: Is something wrong?

- Visual inspection
- Syntax: Read and interpret compiler messages
  - Search the web for the exact error
- Logic: Create test cases and check that output matches input
- Logic: Trace through the code (read it out loud)
- Runtime: Notice that it crashed ☺

Bug localization tools: Where is it?

- Syntax: Look at line numbers identified by the compiler
- Logic:
  - Inspect program state (also useful for runtime errors)
    - Use `cout` to print variables and see what is happening during execution
    - Use `cin` to pause the program
  - Check your assumptions explicitly with `assert(<expr>)`
  - Trace through the code (read it out loud)
  - Comment out problematic code to isolate it

Debugging example

```
float altitude;

if (altitude > 60000)
  cout << "Up high!" << endl
  cout << altitude << " is way too high! << endl

cout << "Enter another altitude:";
cin << altitude;

if (altitude > 60000)
  cout << "Up high!" << endl
  cout << altitude << " is way too high! << endl

return 0;
```

Course map

Divide and conquer part 2 (recursion)

Structured data (arrays)

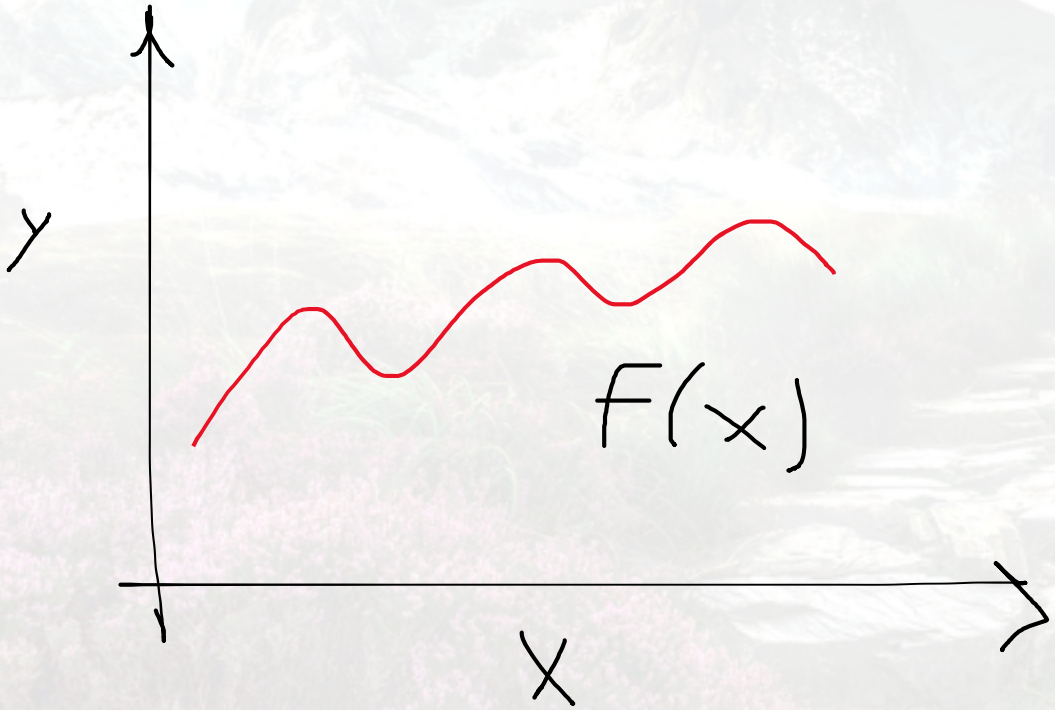
Basics
Storing data, calculations, interacting with users

Decision making (adaptation) and **repetition** (write once, repeat forever!)

Divide and conquer (modularization and code re-use in functions)

Dynamic growth (memory allocation and management)

Functions!



Functions allow us to…

- Divide and conquer
  - Break problem/task into subtasks: **decomposition**
- Make programs easier to design
- Make code easier to understand
  - Abstract away from details
- Reduce replicated (repeated) code
  - Why does this matter?

Oregon State University
College of Engineering

Robot patrol (top-down design / decomposition)

- Goal: Patrol perimeter and check for intruders

Oregon State University
College of Engineering

Bottom-up design / composition

- Given only these functions:
  - `void forward(int steps);`
  - `void turn_right();`
- Create new functions:
  - `void turn_360_degrees();`
  - `void turn_left();`
  - `void backward(int steps);`

Functions you've already used

- `main()`
- `rand()`
- `time(NULL)`
- `srand(time(NULL))`

Functions

Semi-colon required

Function declaration or prototype

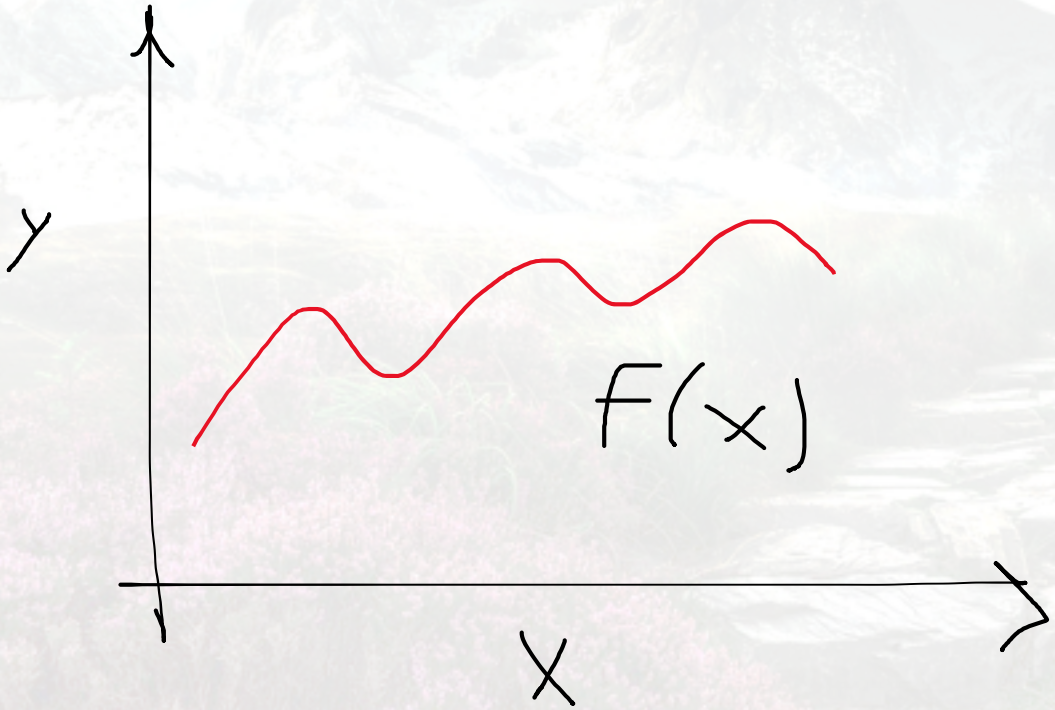
```
float circle_area(float radius);
```

Return type Name Parameters

Function definition

```
float circle_area(float radius)
{
  return 3.14159 * radius * radius;
}
```

Functions




```
float f(float x)
{
   y = ...;
   return y;
}
```

Functions

Function call

```
int main()
{
   float r;
   cin >> r;
   float area = circle_area(r);
   return 0;
}
```

Arguments

- Function must have been declared or defined earlier in the file
  - **Declaration**: "I will define this later."
  - **Definition**: "I'm defining it now."

Function headers

```
/*********************************************************
 ** Function:
 ** Description:
 ** Parameters:
 ** Pre-Conditions:
 ** Post-Conditions:
 *********************************************************/
return_type function_name(type param, type param, ...)
{
   ...;
}
```

Oregon State University
College of Engineering

Function header example

```
/*****************************************************
 ** Function: circle_area
 ** Description: Calculate area of circle, given radius.
 ** Parameters: radius (float)
 ** Pre-Conditions: radius is non-negative
 ** Post-Conditions: return area
 *****************************************************/
float circle_area(float radius)
{
  return 3.14159 * radius * radius;
}
```

Oregon State University
College of Engineering

Functions: multiple parameters

Function definition

```
float calc_BMI(float height, float weight)
{
  return weight / pow(height, 2);
}
```

- But only one return value
- Functions can call other functions

What vocabulary did we learn today?

- Testing: edge cases
- Function declaration vs. definition
- Function parameters vs. arguments
- Function call

What ideas and skills did we learn today?

- Error types: syntax, logic, runtime
- Strategies for detecting and locating bugs
- How functions can make programs easier to design and read
- Good function header style

Oregon State University
College of Engineering

Week 3 nearly done!

- ❑ Attend lab (laptop required)
- ❑ Read **Rao Lesson 7** (pp. 151-158) - functions
- ❑ Finish **Assignment 2 implementation** (due **Sunday, Jan. 26**)

See you Monday!

- ❑ Bring: name of a physical object that acts as a function