FUNCTIONS FOR ALL SUBTASKS

CS 211

WICHITA STATE UNIVERSITY

INTRODUCTION

- Wrap up discussion of functions
- Go over some basic debugging

AGENDA

- Void functions
- Pass by reference
- Testing and debugging functions
- General Debugging

VOID FUNCTIONS

VOID

- A return type is required when writing functions, but we don’t always want to return something
- If the return type is void, we are saying that the function returns nothing
- Because the function returns nothing, we no longer have to assign the function call to anything

VOID FUNCTION EXAMPLE

```
#include <iostream>
void say_hello() {
    std::cout << “Hello!\n”;
    return;
}

int main() {
    say_hello();
}
```

PASS BY REFERENCE

GETTING VARIABLES INTO FUNCTIONS

- Every function we've written so far has passed the variables "by value"
  - A copy of the variable was given to the function
- Sometimes we don't want to give a copy, we want to give the original
- This is called pass "by reference"

HOW TO PASS BY REFERENCE

- Use the ‘&’ operator when specifying the parameter
  - Called ‘address of’ operator
- Use of an extra operator implies that pass by value is the default behavior
- Mix & match of pass by value and pass by reference is perfectly fine

REASONS TO PASS BY REFERENCE

- Avoid expensive copies
  - Some objects are considered "heavy"
  - Code runs much faster if copies are avoided
- Output parameters
  - A way to get more than one output from a function is to pass parameters intended strictly for output
  - Generally the last 'n' parameters, and are named to identify as output

TESTING AND DEBUGGING FUNCTIONS

MAKING DIVISION WORK FOR US

- Dividing problems into sub-tasks have many benefits
- Another benefit is test-ability
- Breaking sub-tasks into functions means that once a function is tested (thoroughly), any issues we run into must be in untested code
  - Helps us find issues quickly
- How can we test our code?

DRIVERS

- A driver is a separate program whose intent is solely to test your code
- Can conduct unit tests and/or functional tests
  - Unit tests ensure that individual chunks of code behave correctly
  - Functional tests bring in multiple pieces of code to ensure that they interact with other correctly
- Frameworks exist to do this for us
  - doctest
  - catch2
- We can write simple tests ourselves as well

AND STUBS

- Stubs allow us to test one piece of code while another isn’t finished yet
- The unfinished piece is written just to return a dummy value
- This allows us to test early
- Once the stub gets fleshed out, the existing tests should remain valid

GENERAL DEBUGGING

RECURRING TOPIC

- This will be brought up from time to time
- As we learn new things, we will learn how to debug them
- Debugging is extremely important
  - And not always easy

STARTING OUT ADVICE

- Don’t assume the bug has to be where you initially think it is
- Do NOT throw code into your program to see what happens
  - Debugging is a controlled process
- When the decision comes down to taking a break and getting frustrated, take the break

CHECK THE BASIC STUFF

- Uninitialized variables
- Off by one
- Exceeding data boundary
  - Very important with arrays
- Automatic type conversion
- Using = instead of ==

LOCALIZE THE ERROR

- Can’t fix a bug if we don’t know where it is!
- Comments can be used as debugging tools as well
  - Comment out a line or block
  - Stub it if needed
  - Deleting code is a cleanup step, not a debugging step
- Trace variables with `std::cout`

ASSERT()

- Asserts are a great way to check if conditions are being met
- They have the form `assert(BOOLEAN_EXPRESSION);`
- If the Boolean expression evaluated to false, the program terminates
  - Also shows you which assert failed
- If you can check conditions at compile-time, `static_assert()` can be used
  - Program will fail to compile if Boolean expression evaluates to false
  - Able to add a message