PROCEDURAL ABSTRACTION AND FUNCTIONS THAT RETURN A VALUE

CS 211

WICHITA STATE UNIVERSITY

INTRODUCTION

- Any program can be thought of as a collection of sub-parts
- We can give these parts names and code them separately

AGENDA

- Top Down Design
- Predefined functions
- Programmer defined functions

TOP DOWN DESIGN

HOW NOW?

- What should our first step be?
- What should our pseudocode do?
- The steps we perform to do a task are called an **algorithm**

A SOLID PLAN OF ATTACK

- Break down our task into smaller sub-tasks
- Break those sub-tasks down into sub-sub-tasks
- Continue to do so until these pieces are trivial to code
- This is Top Down Design

BENEFITS OF TOP DOWN DESIGN

- Code is easier to understand
- Code is easier to change
- Code is easier to write
- Code is easier to test
- Code is easier to debug

PREDEFINED FUNCTIONS

AS IN ALREADY MADE FUNCTIONS

- C++ comes with libraries of pre-defined functions
- We'll use the square root function, `double sqrt(double)`, from `<cmath>`

DOUBLE SQRT(DOUBLE)

- First, let's look at syntax of using the `sqrt()` function

```
double theRoot = sqrt(9.0);
```

- The expression `sqrt(9.0)` is a function call
- The return value of the function is a double
- In this example, 9.0 is the **argument** to the function
- Functions may have many arguments, but only one value can be returned

WHAT'S REQUIRED TO USE A PREDEFINED FUNCTION?

- We need to know what library it belongs to so we can #include it
- What is the type of the return value?
- What are the parameters?
- What namespace do these functions belong in?

HOW DO WE FIGURE ALL THIS OUT?

- Read about it
- Use a reference like cppreference.com
  - Can be tough to decipher at first

A TANGENT TOPIC APPEARS!

TYPE CASTING

- From Homework 1, integer division was sufficient
- How could we have got an exact division result?
- While still using integers?
- So, how can we get an exact result using just integers?

MORE TYPE CASTING

- Slide titles can be helpful, the answer is type casting
- We can “cast” a variable of one type into the role of another
- Two forms of type casting, implicit and explicit
- We should know how they both work to a degre

IMPLICIT TYPE CASTING

- Implicitly type casting is done for you by the compiler
- Programmer implies that a cast should happen, and it does
- Example: `9 / 2 = 4`
- But wait: `9 / 2.0 = 4.5`
- What happened, and why?

BUT VARIABLES

- We can't add a '.0' to a variable name
- An example

```
int input1, input2;
double result;
result = input1 / input2;
```

- This another example of implicit type casting
- But it doesn't work! Why?

EXPLICIT TYPE CASTING

- We can tell a variable to assume the role of another type
- That's why it's called explicit type casting
- It looks like this: static_cast<NEW_TYPE>(ARGUMENT)
- Using the previous example:

```
int input1, input2;
double result = static_cast<double>(input1) / input2;
```

- This does work, buy why?

PROGRAMMER DEFINED FUNCTIONS

THE GOOD STUFF

- We've seen how to use functions that have already been written
- Now let's discuss creating our own

BEFORE WE START WRITING FUNCTIONS

- Where do these functions go?
- More specifically, the compiler must know about them before they are ever called/used (just like variables)
- This means that we must either place the entire function ahead of our main function, or declare it ahead of our main() function

WHAT IS REQUIRED?

- Three things
  - The data type of function output
  - A name for the function
  - The section for our parameters
- Seem familiar?

DISSECTING A FUNCTION DECLARATION

- `double calcTotalCost(int quantity, double price);`

- The very first part of a function declaration is **always** the type of the function return
- The second part is **always** the name of the function
  - *Remember*: Functions should have an action word in their name
  - Functions do things
- The third part is **always** required, BUT the parentheses can be left empty

FUNCTION BODY

- We've seen a function declaration (the semicolon identifies it)
- Now we'll look at a definition

```
double calcTotalCost(int quantity, double price)
{
    const double TAX_RATE = 0.05;
    double subtotal = quantity * price;
    return subtotal + (subtotal + TAX_RATE);
}
```

WHAT DOES A FUNCTION KNOW?

- What it's given, and what it figures out on its own
  - Scope
- Arguments provided to the function are the only way to get outside information into the function

NOTES ON FUNCTIONS

- Consider them "small programs"
- They **only** know what they are given (parameters), what they can do (function body)
- The **only** way to give functions 'outside' knowledge is through the parameters
- Function definitions are strict
- Utilizing functions properly makes our code more modular, which in turn makes it easier to write, debug, maintain, change, etc.