Procedural Abstraction and Functions that Return a Value

CS 211

Adam Sweeney

September 9, 2017

Wichita State University, EECS

Procedural Abstraction and Functions that Return a Value

CS 211

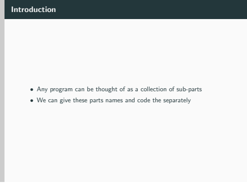
Adam Sweeney

September 9, 2017

Wichita State University, EECS

Introduction

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


Introduction

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

1. Taking in information, processing information, delivering information

Agenda

- Chapter 4 (4.1 - 4.3)
- Top Down Design
- Predefined functions
- Programmer defined functions

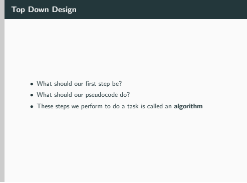
Agenda

- Chapter 4 (4.1 - 4.3)
- Top Down Design
- Predefined functions
- Programmer defined functions

Top Down Design

Top Down Design

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



1. first step? Make a plan
2. pseudocode? should resemble instructions in "plain" language that a clerk could carry out
3. algorithm design is extremely important, it's what makes us the engineers

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

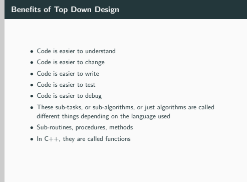
A Solid Plan of Attack

- 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

1. Also called stepwise refinement, divide and conquer

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
- These sub-tasks, or sub-algorithms, or just algorithms are called different things depending on the language used
- Sub-routines, procedures, methods
- In C++, they are called functions


Benefits of Top Down Design

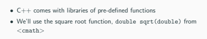
- Code is easier to understand
- Code is easier to change
- Code is easier to write
- Code is easier to test
- Code is easier to debug
- These sub-tasks, or sub-algorithms, or just algorithms are called different things depending on the language used
- Sub-routines, procedures, methods
- In C++, they are called functions

1. Are we sensing a pattern?
2. Before looking at designing our own functions, we'll look at using functions that have been writtenf or us

Predefined Functions

Predefined Functions

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


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

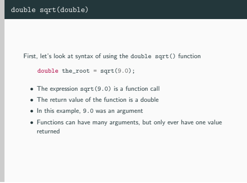
1. What pre-defined functions have we encountered already?

`double sqrt(double)`

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

```
double the_root = 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` was an argument
- Functions can have many arguments, but only ever have one value returned


double sqrt(double)

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

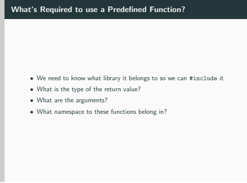
double the_root = sqrt(9.0);

- The expression sqrt(9.0) is a function call
- The return value of the function is a double
- In this example, 9.0 was an argument
- Functions can have many arguments, but only ever have one value returned

1. About arguments, they can be constant, like 9.0, or a variable (very common),or a more complicated expression
2. Arguments can be considered as input to a function
3. The value returned can be considered as the function's output

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 arguments?
- What namespace to these functions belong in?

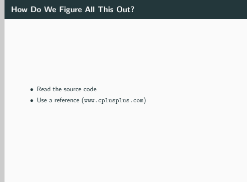

What’s Required to use a Predefined Function?

- 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 arguments?
- What namespace to these functions belong in?

1. Fun fact. The absolute value of an integer actually resides in the library `<cstdlib>`
2. We typically want to store the output of a function, so it would be good if we what type of variable we need
3. We can’t use the function properly if we don’t give it the inputs it expects, the way it expects them
4. Just about anything we will use in the class belongs in the `std` namespace, so we’re covered by the line `using namespace std;`

How Do We Figure All This Out?

- Read the source code
- Use a reference (`www.cplusplus.com`)

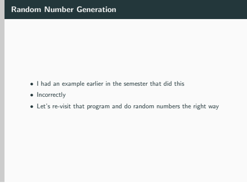

How Do We Figure All This Out?

- Read the source code
- Use a reference (`www.cplusplus.com`)

1. The code for these functions exist on our systems
2. This is much easier, and what I recommend

Random Number Generation

- I had an example earlier in the semester that did this
- Incorrectly
- Let's re-visit that program and do random numbers the right way

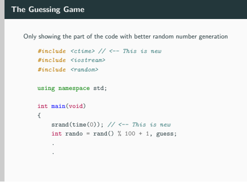

Random Number Generation

- I had an example earlier in the semester that did this
- Incorrectly
- Let's re-visit that program and do random numbers the right way

The Guessing Game

Only showing the part of the code with better random number generation

```
#include <ctime> // <-- This is new
#include <iostream>
#include <random>

using namespace std;

int main(void)
{
    srand(time(0)); // <-- This is new
    int rando = rand() % 100 + 1, guess;
    .
    .
```


The Guessing Game

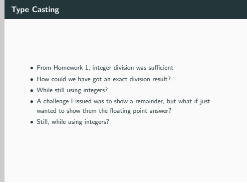
Only showing the part of the code with better random number generation

```
#include <ctime> // <-- This is new
#include <iostream>
#include <random>

using namespace std;

int main(void)
{
    srand(time(0)); // <-- This is new
    int rando = rand() % 100 + 1, guess;
    .
    .
```

1. This is just an actual demonstration of how predefined functions can make our lives easier when we use them properly

Type Casting

- From Homework 1, integer division was sufficient
- How could we have got an exact division result?
- While still using integers?
- A challenge I issued was to show a remainder, but what if just wanted to show them the floating point answer?
- Still, while using integers?

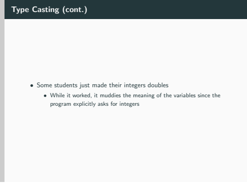

Type Casting

- From Homework 1, integer division was sufficient
- How could we have got an exact division result?
- While still using integers?
- A challenge I issued was to show a remainder, but what if just wanted to show them the floating point answer?
- Still, while using integers?

1. It's a surprisingly small change, but you have to know it exists

Type Casting (cont.)

- Some students just made their integers doubles
  - While it worked, it muddies the meaning of the variables since the program explicitly asks for integers

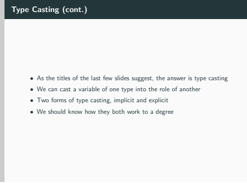


Type Casting (cont.)

- Some students just made their integers doubles
  - While it worked, it muddies the meaning of the variables since the program explicitly asks for integers

Type Casting (cont.)

- As the titles of the last few slides suggest, 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 degree

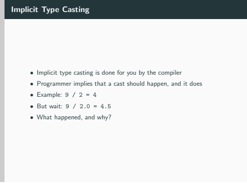


Type Casting (cont.)

- As the titles of the last few slides suggest, 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 degree

Implicit Type Casting

- Implicit 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?


Implicit Type Casting

- Implicit 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?

1. By changing 2 to 2.0, we implied that the divisor is a double
2. Integers don't have decimals, but a double does
3. Integers can safely be promoted to doubles because a double requires more bytes than an integer, i.e., an integer can safely fit in the space of a double
4. The compiler recognizes that you implied a double, so it promotes the dividend to be a double and performs double division instead of integer division
5. What if we're working with variable names?

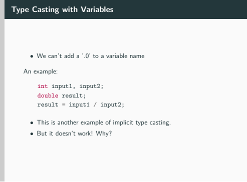
Type Casting with Variables

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

An example:

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

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


Type Casting with Variables

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

An example:

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

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

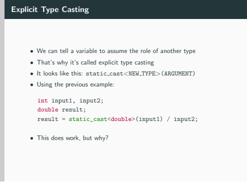
1. It doesn't work because the integer division is fully evaluated and then promoted to be a double
2. Meaning, the wrong answer gets stored as a double

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

- This does work, but 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;
result = static_cast<double>(input1) / input2;
```

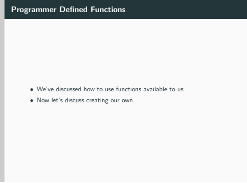
- This does work, but why?

1. This actually banks on both explicit and implicit type casting
2. The explicit type cast occurs when we tell the input1 variable be a double
3. The compiler sees that we are dividing a double by an integer, and performs an implicit type cast on the divisor, promoting it to double
4. Then, double division is done and the result is stored in the variable `result` directly
5. There is an older syntax for type casting. Don't use it.

Programmer Defined Functions

Programmer Defined Functions

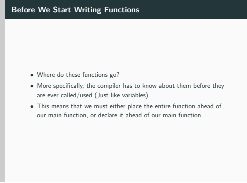
- We've discussed how to use functions available to us
- Now let's discuss creating our own


Programmer Defined Functions

- We've discussed how to use functions available to us
- Now let's discuss creating our own

Before We Start Writing Functions

- Where do these functions go?
- More specifically, the compiler has to 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

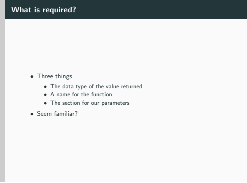

Before We Start Writing Functions

- Where do these functions go?
- More specifically, the compiler has to 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

1. They can be in the same file or in a separate file
2. For this class, they will always be in the same file
3. We can think of placing the entire function in front of main as declaring and initializing it at the same time
4. Where placing a declaration, or prototype, is simply declaring the function, and we give it a value later

What is required?

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


What is required?

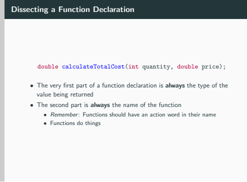
- Three things
    - The data type of the value returned
    - A name for the function
    - The section for our parameters
- Seem familiar?

1. It should seem familar, it's a lot like declaring a variable

Dissecting a Function Declaration

```
double calculateTotalCost(int quantity, double price);
```

- The very first part of a function declaration is **always** the type of the value being returned
- The second part is **always** the name of the function
  - *Remember*: Functions should have an action word in their name
  - Functions do things


Dissecting a Function Declaration

```
double calculateTotalCost(int quantity, double price);
```

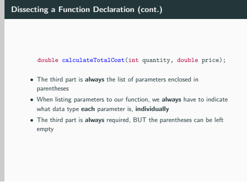
- The very first part of a function declaration is **always** the type of the value being returned
- The second part is **always** the name of the function
  - *Remember*: Functions should have an action word in their name
  - Functions do things

1. 1st part is the type of the value returned, in this case, the function is returning a double
2. Naming rules apply to functions as well as variables
3. Names should be succinct and unambiguous

Dissecting a Function Declaration (cont.)

```
double calculateTotalCost(int quantity, double price);
```

- The third part is **always** the list of parameters enclosed in parentheses
- When listing parameters to our function, we **always** have to indicate what data type **each** parameter is, **individually**
- The third part is **always** required, BUT the parentheses can be left empty



Dissecting a Function Declaration (cont.)

```
double calculateTotalCost(int quantity, double price);
```

- The third part is **always** the list of parameters enclosed in parentheses
- When listing parameters to our function, we **always** have to indicate what data type **each** parameter is, **individually**
- The third part is **always** required, BUT the parentheses can be left empty

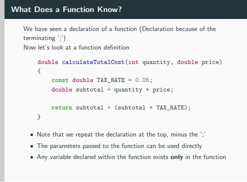
1. This part trips a lot of people up
2. When declaring a function, the parentheses are **ALWAYS** required
3. HOWEVER, if you are not giving your function any parameters, you may leave them empty

What Does a Function Know?

We have seen a declaration of a function (Declaration because of the terminating ';')
Now let's look at a function definition

```
double calculateTotalCost(int quantity, double price)
{
    const double TAX_RATE = 0.05;
    double subtotal = quantity * price;

    return subtotal + (subtotal * TAX_RATE);
}
```

- Note that we repeat the declaration at the top, minus the ';'
- The parameters passed to the function can be used directly
- Any variable declared within the function exists **only** in the function


What Does a Function Know?

We have seen a declaration of a function (Declaration because of the terminating ';')
Now let's look at a function definition

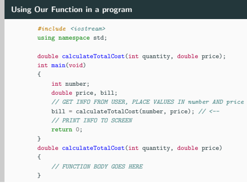
```
double calculateTotalCost(int quantity, double price)
{
    const double TAX_RATE = 0.05;
    double subtotal = quantity * price;

    return subtotal + (subtotal * TAX_RATE);
}
```

- Note that we repeat the declaration at the top, minus the ';'
- The parameters passed to the function can be used directly
- Any variable declared within the function exists **only** in the function

1. The use of parameters and passing parameters trips a lot of people up
2. Passing will be discussed next, let's just remember right now that a function parameter is like a declared variable that we have access to

Using Our Function in a program

```cpp
#include <iostream>
using namespace std;

double calculateTotalCost(int quantity, double price);
int main(void)
{
    int number;
    double price, bill;
    // GET INFO FROM USER, PLACE VALUES IN number AND price
    bill = calculateTotalCost(number, price); // <--
    // PRINT INFO TO SCREEN
    return 0;
}
double calculateTotalCost(int quantity, double price)
{
    // FUNCTION BODY GOES HERE
}
```



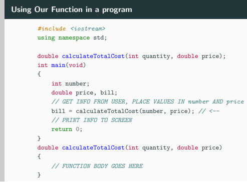
Using Our Function in a program

```cpp
#include <iostream>
using namespace std;

double calculateTotalCost(int quantity, double price);
int main(void)
{
    int number;
    double price, bill;
    // GET INFO FROM USER, PLACE VALUES IN number AND price
    bill = calculateTotalCost(number, price); // <--
    // PRINT INFO TO SCREEN
    return 0;
}
double calculateTotalCost(int quantity, double price)
{
    // FUNCTION BODY GOES HERE
}
```

1. Let's discuss what's happening. First, note the function declaration above the main function. You can either place a declaration as I have, or the entire function definition
2. Style dictates using a declaration. People generally want to know what your program does, and that means they want to read your main function
3. The declaration/prototype basically tells the compiler, "Hey, this is a thing, don't freak out when you see it"
4. In the main function, we store user input into the variables `number` and `price`. When we call the function, we are assigning its output to the variable `bill`

Using Our Function in a program

```
#include <iostream>
using namespace std;

double calculateTotalCost(int quantity, double price);
int main(void)
{
    int number;
    double price, bill;
    // GET INFO FROM USER, PLACE VALUES IN number AND price
    bill = calculateTotalCost(number, price); // <--
    // PRINT INFO TO SCREEN
    return 0;
}
double calculateTotalCost(int quantity, double price)
{
    // FUNCTION BODY GOES HERE
}
```



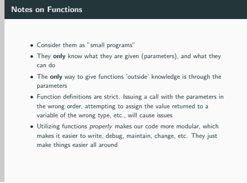
Using Our Function in a program

```
#include <iostream>
using namespace std;

double calculateTotalCost(int quantity, double price);
int main(void)
{
    int number;
    double price, bill;
    // GET INFO FROM USER, PLACE VALUES IN number AND price
    bill = calculateTotalCost(number, price); // <--
    // PRINT INFO TO SCREEN
    return 0;
}
double calculateTotalCost(int quantity, double price)
{
    // FUNCTION BODY GOES HERE
}
```

1. This part here is where people get tripped up. When the function was called, the variables `number` and `price` were given. The fact that the second parameter was called price is a coincidence
2. Remember scope. The variables in main do **NOT** exist inside the function `calculateTotalCost`
3. Instead, what is happening is the values of the variales we gave to `calculateTotalCost` were copied into new variables called `quantity` and `price` and given to the function
4. This is called "Pass by Value"

Notes on Functions

- Consider them as "small programs"
- They **only** know what they are given (parameters), and what they can do
- The **only** way to give functions 'outside' knowledge is through the parameters
- Function definitions are strict. Issuing a call with the parameters in the wrong order, attempting to assign the value returned to a variable of the wrong type, etc., will cause issues
- Utilizing functions *properly* makes our code more modular, which makes it easier to write, debug, maintain, change, etc. They just make things easier all around


Notes on Functions

- Consider them as "small programs"
- They **only** know what they are given (parameters), and what they can do
- The **only** way to give functions 'outside' knowledge is through the parameters
- Function definitions are strict. Issuing a call with the parameters in the wrong order, attempting to assign the value returned to a variable of the wrong type, etc., will cause issues
- Utilizing functions *properly* makes our code more modular, which makes it easier to write, debug, maintain, change, etc. They just make things easier all around