CS 102
Introduction to Programming Using C++

Chapter 5

Functions

Homework

- Written homework
- P. 233, R5.1, 2, 4, 5, 7, 8, 9, 12 (parts e, f, g), 17, 20

- Programs
- p. 237, choose one of P5.3, 4, 5
- Also, choose one of P5.27-30

Functions

- A C++ function is similar to a mathematical function
- It has some inputs
- It may produce an output
  - Unlike math, a function does not have to return a result

Examples of Functions

int main ()

int twice (int x)

void print_int (int num)

double sin (double x)

A Look at a Function

```
double rectangle_area (double rect_length,
                       double rect_width)
{
    double area = rect_length * rect_width;
    return area;
}
```

Analyzing the Example

Why Do We Need Functions?

- Why should we not just make everything one big program?
- Why not put all the code into the main () function so it does everything?
- It would seem to be easier than managing a lot of pieces
- It's actually more difficult, for several reasons

Benefits of Using Functions

- They simplify the coding process and speed up the development process
- If you have to use the same code in two or three different places, you don't have to rewrite it
- There is value to cut/paste, but maintenance (making changes) is a nightmare
  - Maintaining the same code in several places is difficult
  - That makes it error-prone
- They make testing easier
  - You can test each module separately, and then assemble them
- They make it easier for a team to design a large project
  - Each person/team can focus on an individual module

The Textbook's Example- Creating the Function

```
double cube_volume (double side_length)
{
    double volume = side_length * side_length *
                                side_length;
    return volume;
}
```

The Textbook's Example- Using the Function

```
vol = cube_volume (3);
cout << "The volume is " << cube_ volume (12);
double_vol = 2*cube_volume (side1);
```

Vocabulary

- Parameter
- Argument
- Function header
- Function body

Modifying Parameter Values

- In general, it's not a good idea to modify parameter values
- Here is a bad example

```
//  Function to convert dollars and cents to only cents
int total_cents (int dollars, int cents)
{
    cents = dollars * 100  +  cents;
    return cents;
}
```

A Better Example

- Here is a better version of that function that doesn't change the parameter

```
//  Function to convert dollars and cents to only cents
int total_cents (int dollars, int cents)
{
    int new_cents = dollars * 100  +  cents;
    return new_cents;
}
```

Explaining How to Use Functions (A Commenting Standard)

- There are special comments we put immediately before functions
- Their purpose is to tell how to call the function
- These are from Java; they follow a style called JavaDoc
- Here is an example for the cube_volume function

```
/**  Computes the volume of a cube
     @param side_length The side length of the cube
     @return  The volume of the cube  */
```

The return Statement

- All C++ functions exit when they see the return statement
- Code after a return statement is not executed
  - This code is called unreachable code
- The cout after the return statement below is not executed

```
double cube_volume (double side_length)
{
  double volume = side_length * side_length *  side_length;
  return volume;
  cout <<  "The volume is " << volume;
}
```

Types of Functions

- Some functions don't return a value
  - These functions typically do a task
  - Their type is void
  - void print_line (string line_to_print)
- Other functions return a value
  - Their type is based on what they return
  - int twice (int x)
  - double raise_to_power (int x)
  - int read_integer ()

Void vs. Non-Void Functions

- A void function should not have a return statement
  - The g++ compiler will catch this
- A non-void function must return a value
  - The g++ complier won't catch this
  - You will get a run-time error
  - Of course, the type returned must match the function's type

Helping the Compiler Find Functions

- This code is OK

```
double cube_volume (double side_length)
{
   double volume = side_length * side_length *  side_length;
   return volume;
}


int main ()
{
   cout << cube_volume (4);
}
```

Where Is cube_volume?

- This code won't compile
  - The compiler can't find cube_volume when it needs it

```
int main ()
{
   cout << cube_volume (4);
}


double cube_volume (double side_length)
{
   double volume = side_length * side_length *  side_length;
   return volume;
}
```

Function Prototypes

- Another approach is to copy a function's header to the beginning of the program
  - Add a semi-colon after the header
- If you do not do this, the compiler expects all functions to be defined before being used
- The header that you copied to the beginning of the program is called a function prototype
- Some programmers prefer to put all functions at the beginning of the program

Steps to Writing a Function

- First, you must decide what the function should do
- Decide on the function's inputs
- Decide on what will be returned
- Code the function
  - Use pseudocode or a flowchart to design
  - Then write the code
- Test the function

Keep Functions Short

- All functions (including main() itself) should be short
  - Typically, this means the function's code must be no more than two or three screens long
- A function should do only one task
- If it does more than that, it should be redesigned
  - It is probably two or more functions

A Book in the Library

- Suppose we are writing a program to track a book's history at a library
- What can happen to the book?
  - A patron requests the book (R)
  - The library purchases the book (P)
  - The book is placed on a shelf (S)
  - The book can be checked in and out (I, O)
  - The book might be lost or ruined (L)

The Code

- main () contains a loop that asks what should happen to the book
- The user enters R, P, S, I, O, or L
- The code reads the user's choice into the character variable user_request
- The code has a switch statement that processes the user's request

The switch Statement

```
switch (user_choice)
{
    case 'R':  // Lots of code
    case 'P':  // Lots of code
    case 'S':  // Lots of code
    case 'I':   // Lots of code
    case 'O': // Lots of code
    case 'L':  // Lots of code
}
```

A Better Version

```
switch (user_choice)
{
    case 'R':   request (book);
    case 'P':   purchase (book);
    case 'S':   shelve (book);
    case 'I':   check_in (book)
    case 'O':  check_out (book);
    case 'L':   remove_book (book);
}
```

Explaining the switch Statement

- Each line contains a function call
  - This version is more readable!
- We have to write the functions
- We would like to test our code after we write each individual function
- To do this we create stubs which are minimal functions

Some Stubs

- Suppose we have written the code for request () and want to test it
- We can't because the program won't compile until we have every function
- So we code stubs

```
void purchase (string book)
{
}

string shelve (string book)
{
    return "";
}
```

- We have to code stubs for every function

The Scope of Variables

- We already saw this
- Scope means where a variable can be used
- There is an interesting thing that can happen to variables when we use functions
  - It can happen in regular code too
  - It's more common in functions
- It's called shadowing

An Example of Shadowing

```
1. int i = 100, posn = –1;
2. string str = "hello there";
3. for (int i=0;  posn < 0  &&  i<str.length (); i++)
4.      if  (str [i] == ' ')
5.          posn = i;
6. cout << i << endl;
```

That Was Confusing!

- There are two variables with the same name
  - The variable i is declared in Line 1
  - The variable i is also declared in Line 3
  - Will the real i please stand up?
- This is not good!
- We call this shadowing
- We say the variable i in the for loop shadows the variable i in the main() function

Scope

- The i declared in Line 1 is local to the function, that is:
  - it belongs to the entire function
  - and only to the function
- The i in the for in Line 3 belongs only to the for
- When the for ends (Line 5), the variable i doesn't exist anymore
- This solves the mystery:
  - The i in Lines 4 and 5 is the i declared in Line 3
  - The i in Line 6 is the i declared in Line 1