Writing Functions

Chapter 9

For Next Time

- Read Chapter 9

Reminder

- Test #2 is Wednesday, March 18
- That's one week from today!
- Emphasizes Chapters 5–10
  - Conditional statements (`if/else`, `switch`)
  - Loops (`while`, `do/while`, `for`)
  - Calling and writing functions
  - Call by value vs. call by reference
  - Recursion, pointers, etc.

Why Functions?

- Better manage the complexity of larger programs
- Reuse the same code from multiple places within a program
- Encapsulate functionality into "black boxes"
- Programmer focus on smaller, easier subproblems solved in isolation

Definition vs. Invocation

- **Definition**—provides the code that determines the function's behavior
- **Invocation**—is the execution of the function by client code

```
void count(int n)
{
    for ( int i = 0; i < n; i++ )
        cout << i << endl;
}
```

```
count(5);
```

Function Definition

- Type
- Name
- Formal Parameters
- Body

```
type name(parameters)
{
    body
}
```

Function Type

- Type can be any C++ type plus **`void`**
- The **`void`** type means the function does not return a value to the client

```
type name(parameters)
{
    body
}
```

Function Name

- An identifier
- Should accurately describe the purpose of the function

```
type name(parameters)
{
    body
}
```

Function Parameters

- Comma-separated list of parameter declarations
- May be empty

```
type name(parameters)
{
    body
}
```

Function Parameters

- List of statements the function is to execute
- May declare its own local variables

```
type name(parameters)
{
    body
}
```

Function Parameters

- Formal parameters
  - Parameters in the function definition
- Actual parameters
  - Parameters (or arguments) provided by the client
  - May be variables, literal values, constants, and expressions

Call by Value

- Actual parameters are assigned to the formal parameters when the client calls the function
  - This assignment is performed automatically and does not appear in your code
- Since the function works on copies of the actual parameters, the function cannot directly affect the actual parameters
- A variable local to the client may be modified via the function's return type

Local Variables

- Variables declared within a function are **local** to that function
- Local variables are not visible to code outside that function definition
- The name of a local variable can be the same as the name of a local variable in another function—they are different variables
- Formal parameters are local to function definitions

SOUTHERN ADVENTIST UNIVERSITY School of Computing

Function Documentation

- Essential information
  - Purpose
  - Role of each parameter
  - Nature of return value
- Other helpful information
  - Author
  - Date
  - References

SOUTHERN ADVENTIST UNIVERSITY School of Computing

Next . . .

More on functions

SOUTHERN ADVENTIST UNIVERSITY School of Computing

15