Components of a Function

• Must have these 3 parts
  – Function Declaration/Prototype
    • Informs the compiler about the general structure of the function
  – Function Call
    • Instructs the function to execute
  – Function Definition
    • Contains the code that the function executes

Function Prototype

• Placed before any function calls
• Must contain
  – Name of the function
  – Return type
  – Parameter(s) type
• May contain parameter names but this is ignored by the compiler

```
int calculateAge(int, int, int);  // OR
int calculateAge(int day, int month, int year);
```

Chapter 3
Functions

• Divides a program into manageable units
• Main function contains overall structure of program
  – Details are contained in functions
• Main performs 3 actions
  – Calls a function to perform a task
  – Gives the function any information it needs to perform the task
  – Receives the results returned by the function
Function Call

- This is the statement that invokes the function
  - Function can be defined but never called
- Functions can call other functions
- Must pass required data
  - Can pass literals and values in variables
- Use assignment to store return values (if they exist)

```java
salary = calcPay(60, 25, 37.5);
```

Function Definition

- Placed after main function (not inside)
  - Never place a function inside of another
- Variables defined in a function are only visible to that function
  - Called local variables
  - Can have the same variable names in different functions
- Return statement sends data back to the calling function
  - Some functions have nothing to return (void)

Void Functions

- Performs a task but does not send any information back
  - Display information on screen
- } causes the function to terminate
  - Use a return statement to terminate prior to the end of the function

```java
void printTemp(int temp) {
    cout << "The current temperature is " << temp << " and if you're in Houghton it's probably snowing.";
    cout << endl;
}
```

Function Definition

```java
double calcPay(double hours, double rate, double overRate) {
    double overtimeHrs = 0.0;
    double pay = 0.0;
    if (hours > 40.0) {
        overtimeHrs = hours - 40.0;
        hours -= overtimeHrs;
    }
    pay = (hours * rate) + (overtimeHrs * overRate);
    return pay;
}
```
Global Variables

- Declared outside all functions
  - Usually at the top of the file
- Dangerous in large software applications
  - No control over usage
- Commonly used for constants
  - All functions have access to global constants

```cpp
#include <iostream>
using namespace std;
const double TAX_RATE = 0.05;
int main() {
    ...
}
```

Boolean Functions

- Returns true or false
  - Can be in the form of a boolean expression
    - return ((rate == 0) || (rate < 10))
- Return value can directly be used in an if statement or while loop

```cpp
if (isValidDate(month, day, year))
    cout << “Valid Date\n”;
else
    cout << “Invalid Date\n”;
```

Block Scope

- A block is a piece of code enclosed in { }
  - Still a block if the {} are omitted e.g. loop with one statement in the body
- Any variable declared in a block is visible only in that block
  - Called “block scope”

```cpp
for (int i = 1, sum = 0; i < 3; ++i) {
    sum += i;
}
cout << sum << endl;
```

main() - “The Special Function”

- Recall main() is a function
- Special because
  - Only one main function can exist in a program
- Who calls main()? 
  - Operating System
- Return type can be int or void
  - Int is preferred
  - 0 means everything went well
  - 1 means there was an error e.g. `exit(1);`
**Predefined Functions**

- Libraries full of functions you can use
- Must "#include" appropriate library

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>TYPE OF ARGUMENTS</th>
<th>TYPE OF VALUE RETURNED</th>
<th>EXAMPLE</th>
<th>VALUE</th>
<th>LIBRARY HEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt</td>
<td>Square root</td>
<td>double</td>
<td>double</td>
<td>sqrt(4.0)</td>
<td>2.0</td>
<td>cmath</td>
</tr>
<tr>
<td>pow</td>
<td>Powers</td>
<td>double</td>
<td>double</td>
<td>pow(2.0, 3.0)</td>
<td>8.0</td>
<td>cmath</td>
</tr>
<tr>
<td>abs</td>
<td>Absolute value for int</td>
<td>int</td>
<td>int</td>
<td>abs(-7)</td>
<td>7</td>
<td>cstdlib</td>
</tr>
<tr>
<td>labs</td>
<td>Absolute value for long</td>
<td>long</td>
<td>long</td>
<td>labs(-70000)</td>
<td>70000</td>
<td>cstdlib</td>
</tr>
<tr>
<td>fabs</td>
<td>Absolute value for double</td>
<td>double</td>
<td>double</td>
<td>fabs(-7.5)</td>
<td>7.5</td>
<td>cmath</td>
</tr>
</tbody>
</table>

**Random Numbers**

- Return "randomly chosen" number
- Commonly used in simulations & games
  - rand()
    - Takes no arguments
    - Returns value between 0 & RAND_MAX - 1
  - Scaling
    - Squeezes random number into smaller range
    - rand() % 6
    - Returns random value between 0 & 5
  - Shifting
    - (rand() % 6) + 1
    - Shifts range between 1 & 6 (e.g., die roll)

**Procedural Abstraction**

- Need to know “what” a function does, not “how” it does it
- Think “black box”
  - You know how to use it, not how it's designed
- Implement functions like black boxes
  - User of function only needs declaration
  - Does not need to know function definition
    - Called information hiding
    - Hide details of how function works

**Commenting Functions**

- Each function definition should be preceded by its
  - Preconditions
    - Things that must be true for the function to work properly
  - Postconditions
    - Effects of calling the function

```c
// double sqrt(double x)
// Precondition - x is a non-negative number
// Postcondition - Returns the sqrt of x
double sqrt(double x) {
    ...
```
**Random Number Seeds**

- Pseudo random numbers
  - Calls to `rand()` produce given "sequence" of random numbers

- Use "seed" to alter sequence
  - `srand(seed_value);`
    - void function
    - Receives one int argument, the "seed"
    - Can use any seed value, including system time: `srand(time(0));`
    - `time(0)` returns system time as numeric value
    - Library `<ctime>` contains time() functions

**Parameter vs. Argument**

- Terms often used interchangeably
- For our purposes:
  - Parameters
    - Types specified in function prototype
    - Variables declared in function definition's header
  - Argument
    - Data passed in function call
- Parameter is “formal” piece
- Argument is “actual” piece