Introduction to User-defined Functions

Larry Caretto
Computer Science 106
Computing in Engineering and Science

March 21, 2006

California State University Northridge

Outline

- Why we use functions
- Writing and calling a function
- Header and body
- Function prototype
- Passing information to a function
- Returning values in the function name

Introduction to Functions

- Library functions like pow, `atan` and `sqrt` used previously
- Statement to set x = yz^3:
  - `x = y * pow( z, 3 );`
  - Note order of arguments important; the call `pow( 3, z )` gives 3^z
  - Use #include <cmath> for this function
- You can write your own functions
  - Why do we write functions?
  - How do we write code to use functions?

Why do we write functions?

- First use of functions was for code like mathematical function calculations
  - Specialized calculation done repeatedly
  - Want to write code only one time
  - Want to be able to pass values of parameters to code and get value back
- As programs got more complex, breaking code into functions provided a way to organize complex code

How do we write functions?

- C++ code is a collection of functions
- Each function, including main, has the same level of importance in writing code
  - Complete code for each function before starting a new function

```
int main()
{       // body of main
}
int myFunction( …… )
{       // body of myFunction
}
```


Using Functions

- User-defined functions are used in the same way as library functions
  - Functions are used (“called”, “invoked”) by placing the name of the function at the correct location in the code
  - The function name is usually followed by a list of variables or constants in parentheses whose values are passed to the function
  - The result of the function is returned in the function name

Operation of Functions

- Any function (the caller) can call another function by using the name of the function being called in an expression
- The statement calling the function sends information from the caller
- Execution control is transferred to the function being called
- The function being called returns control and (usually) results to the caller

Operation of Functions

- The function being called uses the information it receives to do a set of calculations or procedures
- In the usual case, the function being called returns a result to the caller in the location of the function name
- Example: d = pow(b,2) – 4 * a * c;
  - calls the pow function with values of b and 2
  - and the result b^2 is returned in function name, pow, for further use in an expression

Writing and Using Functions

- Organize the program into individual functions that are called by main
  - Simple example: main calls three functions: (1) input function, (2) calculation function and (3) output function
- Write code for each function (and main)
  - Write function header to specify information received from calling function
  - Write function body to calculate results and return them to “calling” function
- Write function calls that pass data and get results

Writing and Using Functions II

- Library functions, such as pow(x, y) to compute x^y, transfer information based on the order of the variables
- This is true for user-defined functions as well
  - Information transferred from a list of variables in the calling function to a list of variables in the function called
  - Correspondence based on order of variables in function header and statement calling the function

Writing Functions

- The code in a function is works in the same way that the code you have prepared for a single main function
- All the conventional structures can be used
- What is new?
  - The first line of the function, the function header, contains a declaration of the variables whose values are passed into the function when it is called

Function Basics

- Each function has a header and a body
  - Header specifies
    - Name of function
    - Type of value returned by the function name
    - List of variables in the function whose values are determined by the calling program
  - Body gives code executed by the function
- Function prototypes at start of code provide information to compiler
  - Same as header except a semicolon is added at the end
  - Can omit variable names

Function Basics II

- Example of header and body

```
double myPow ( double number,
  double power )  // header
{
     // Body,in braces contains
     // actual code for function
}
```

- Header declares data type for function value, function name, and variables whose values will be received from calling function

Function Header

- Header has following syntax:
  - ***<type>*** ***<name>*** (***<argument list>***)
  - ***<type>*** specifies the type of value returned by the function
  - ***<name>*** is the name you choose for your function; this name is used to call the function from another function
  - ***<argument list>*** specifies type and names of variables in function whose values come from the calling program
  - There is no semicolon at the end of the function header

Function Header II

- Example of function, `myPow`, a user-defined function to replace pow
  - Pass the number and the power to the function as type double
  - Return the result as type double
  - *General header syntax from previous chart*
    <type> ***<name>*** (***<argument list>***)
  - `double myPow ( double number, double power )`

Argument List

- Example on previous chart had two parameters in argument list
  - ***<type>*** ***<name>*** (***<argument list>***)
  - `double myPow ( double number, double power )`
- Function will use `number` and `power` as type double variables
- Values for these variables set by other function that calls (uses) `myPow`

Passing arguments

- Based on order of arguments in function header and in calling statement
- Recall the library pow function was called as pow(number, power)
  - pow(3,4) = 3^4 but pow(4,3) = 4^3
  - What is result of following code

```
double number = 3, power = 4;
cout << pow( power, number)
```

Result is 4^3; only the order counts!

The `return` Statement

- We have used this statement in main as `return EXIT_SUCCESS;`
- The general syntax of this statement is `return` ***<value>***`;`
- ***<value>*** may be a constant, a variable or an expression
- This is value returned to calling program in function name
- `return` always transfers control to calling function

Organization of Function Code

```
double myPow( double number,
              double power )
{
    double result = exp( power *
                    log( number ) );
    return result;
}
```

- Place following prototype at top of code

```
double myPow( double number,
              double power );
```


Alternative Function Code

```
double myPow( double number,
              double power )
{
    return exp( power *
                    log( number ) );
}
```

- Can use following prototype without variable names at top of code

```
double myPow( double, double );
```


Exercise

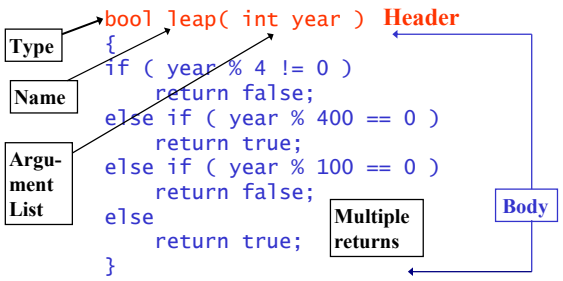
- Write a statement that uses the function myPow(double number, double power) to compute $z = x + yz^3$

```
z = x + y * myPow( z, 3 );
```

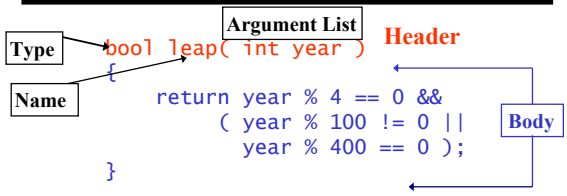
- Write the header for a type bool function named equal that whose first parameter is x, an int variable and whose second parameter, a, is type double

```
bool equal( int x, double a )
```


Another Function Example



Yet Another Function Example



- This function will have same behavior as one on previous chart
- User of function does not have to know its internal code, only its input and output

Use of bool leap(int year)

```
bool leap ( int year );// prototype
int main()  // examples of use
{  cout << "Enter a year: ";
   int y; cin >> y;
   bool cond = leap( y );
   if ( leap( y ) ) {…}
   if ( leap( y ) && month == 2 ) {…}
   return EXIT_SUCCESS
}
// leap and other functions go here
```

Exercise

- Write a function that takes two type int arguments and returns their difference

```
int diff( int a, int b)
{    return a – b; }
```

- Use this function to compute and print the difference 3 – 5

```
cout << diff( 3, 5 );
```

- Write the prototype

```
int diff( int a, int b);
```


Exercise

- Write a function that takes two type double arguments and returns their quotient
- Use this function to compute 5/3
- Write the prototype

```
double div( double a, double b)
{    return a / b; }
//use:  cout << div( 5, 3 );
// prototype: double div(double a,
                         double b);
```