Passing Parameters to User-defined Functions

Larry Caretto
Computer Science 106
Computing in Engineering and Science

March 23, 2006

California State University Northridge

Outline

- Review Introduction to Functions
  - Header and body
  - Function prototype
  - Passing information to a function
  - Returning values in the function name
- `void` functions with no return value
- Use pass-by-reference to change variables passed from the calling program to the function

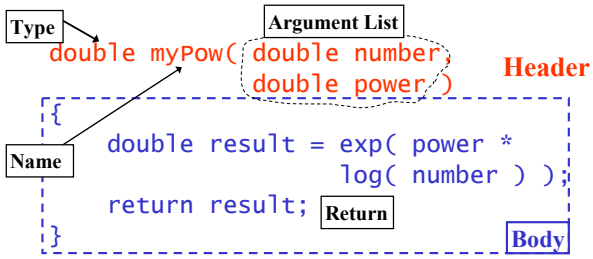
Review Function Introduction

- A C++ program is a collection of functions
  - Each function is written as a unit
  - Complete code for one function before starting to write a new one
  - Execution starts in main function
- Upon calls to a function, information and control is transferred to the function
- Value returned in function name

Review Parts of a Function

- Function header is first line of function
  - Gives type of function, name of function and argument list
    - Name is used to "call" function
    - Argument list specifies variables whose values are defined by the unit that calls the function
- Function body, enclosed in braces { }, gives function code
- Prototype is header with a semicolon that appears before main function

Function Example



- Place following prototype at top of code

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


Information Transfer

- Function header has argument list
- Variables in that list (called dummy parameters or dummy arguments) are determined by call to function
- Call to function has actual arguments or actual parameters in same order that dummy arguments appear
  - Order is all that matters in transferring information to a function

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

California State University Northridge 7

Exercise

- A function and that has two type int arguments and returns the larger of the two; write its code and prototype

```
int larger( int a, int b)
{    if ( a > b )
         return a;
     else
         return b;
}
```

California State University Northridge 8

Exercise Continued

- Prototype for the function is

`int larger ( int a, int b)` or
`int larger ( int, int )`

- How would you use the larger function to return the larger of two variables x and y in a new variable z?

`int z = larger( x, y )`

- Does the order of x and y matter?
  - Not for this function: max(x,y) = max(y,x)

California State University Northridge 9

`void` functions

- The type void used for functions that do not return a value
- Example: error message function
  - Receives a numerical code from the calling program
  - Prints the correct error message for the given error code
  - Does not have to return any information to the calling program

California State University Northridge 10

Error Message Function

```
void printError( int code )
{
  if ( code == 1)
    cout << "Type one error\n";
  else if ( code == 2 )
    cout << "Error two is …
  else if ( code == 2 )
    cout << "Third error message …
  else if      // additional code
}  // no return needed here
```

California State University Northridge 11

Using `void` Functions

- Simply put function name and argument(s) as one statement in code
- Since void functions do not return a value they cannot be used in an expression or have their value assigned to a variable
- Example

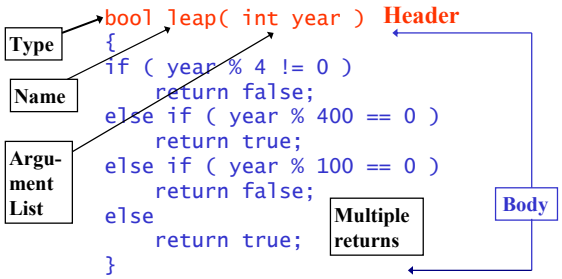
```
int errCode = 6;
if ( error )
    printError( errCode );
```

California State University Northridge 12

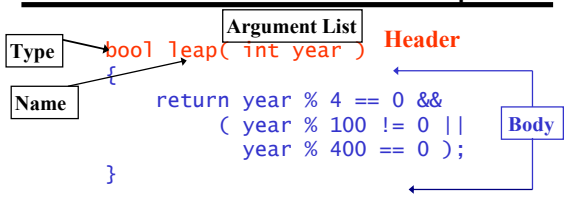
The `return` Statement

- The `return` statement returns control and a value to the calling program
  - Functions, other than void functions use the syntax `return` *<value>* to return a value to the calling function in the function name
  - Void functions may have a simple `return` statement without a value to return control to the calling program at some point before the end of the function
- Functions may have more than one return statement
  - `return` transfers control immediately

Multiple Return Example



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 argument list and meaning of the answer returned

Empty Argument List

- If a function does not need any values from the calling program an empty set of parentheses is required
- Example is function with several output statements to describe purpose of code

```
void describeCode()
{
    cout << "This code …..
    cout << "Still more output
  // No return needed for type void
}
```


Kinetic Energy Function

- Write a function that takes two type double parameters, mass and velocity and computes kinetic energy = $mV^2/2$

```
double KE( double m, double v )
{
    return m * v * v / 2;
}
```

- Possible calls to this function

```
totalE = KE( 4, 3 ) + PE;
```

- What is result of this call

```
result = 50 + KE( 5, 2 );
```

50 + 5 * 2^2 / 2 = 60

Kinetic Energy Function II

```
double KE( double m, double v )
{
    return m * v * v / 2;
}
```

- What is output from these calls?

```
double mass = 5, velocity = 2, PE = 0;
cout << KE( velocity, mass );25
double e = PE + KE( 2*pow( velocity,
                        2), velocity);
double total = KE( mass * velocity,
                                mass );
```

e = 0 + KE(2*2^2,2) = KE(8,2)=8*2^2/2 = 16

total = KE(5*2,5) = KE(10,5)=10*5^2/2 = 125

Passing Information to Functions

- Parameters in function header: formal parameters or dummy parameters (also called formal or dummy arguments)
- Values sent to function by calling program: actual parameters or actual arguments
- Pass by value is default process: when a function is called a copy of the value of the argument is passed to the function

More on Information to Functions

- In pass-by-value, the values of the actual arguments in the calling program are not changed
- The alternative to pass by value is pass by reference
  - The memory address of the actual parameter is passed to the function
  - Changes to the dummy parameter in the function change the actual parameter in the calling program

Pass-by-Value Example

```
//calling program
double x = 8, y = 2;
cout << "fake = " << fake( x, y );
cout << "x = " << x << ", y = "
     << y;  // what is printed?
//function
double fake( double x, double y )
{
    x +=10; y *= x;  return 3 * y;
}
```

fake = 60 x = 8 y = 2

x = 8 + 10 = 18

y = 2 * 18 = 20

return 3 * 20 = 60

Pass-by-value Operation

- The code on the previous chart does not change the x and y values in the calling program
- Only values of x and y from the calling program are passed to the function
- Functions cannot changed values of variables that are passed by value
- How do we use pass by reference to change the values of parameters passed into a function?

Pass-by-reference

- To use pass by reference place an ampersand (&) between the type and the parameter name in the function header: `int f1( int& x, int& y )`
  - Not a preferred programming style
  - Used only when we have to change more than one parameter (*e.g.*, input routine, vector components, *etc.*)
  - Exercise seven uses an input function which must have pass by reference

Pass-by-reference II

- Default is pass-by-value where changes to parameters do not affect variables in the calling program

```
double fake1 ( int x, double y )
{  x++;  y += x;  return x * y; }
```

- Ampersand (&) gives pass by reference that changes program variables

```
double fake2 ( int& x, double& y )
{  x++;  y += x;  return x * y; }
```

Pass-by-Value Example

```
//calling program segment
double u = 5, v = 2;
cout << "fake = " << fake( u, v );
cout << "\nu =" << u << ", v =" << v;
   // what is printed?
//function
double fake( double x, double y )
{
    x +=10; y *= x;  return 3 * y;
}
```

fake = 90
u = 5, v = 2

x = 5 + 10 = 15 y = 2 * 15 = 30 fake(u, v) = 3 * 30 = 90

California State University Northridge 25

Pass-by-Reference Example

```
//calling program segment
double u = 3, v = 4;
cout << "fake = " fake( u, v );
cout << "\nu =" << u << ", v =" << v;
     // what is printed?
//function
double fake(double& x, double& y )
{
    x +=10; y *= x;  return 3 * y;
}
```

fake = 156
u = 13, v = 52

x = 3 + 10 = 13 y = 13 * 4 = 52 fake(u, v) = 3 * 52 = 156

California State University Northridge 26

Pass-by-Reference Example II

```
double u = 3, v = 4;
cout << "fake = " fake( u, v );
cout << "\nu =" << u << ", v =" << v;
double fake( double& x, double& y )
{ x +=10; y *= x;  return 3 * y; }
// at start fake has x = 3, y = 4
// fake code sets x = x + 10 = 13
// and y = y * x = 4 * 13 = 52
// fake returns 3 * 52 = 156 and
// changes u to 13 and v to 52
```

California State University Northridge 27