More on Pass by Value and by Reference

Larry Caretto
Computer Science 106
Computing in Engineering and Science

March 28, 2006

California State University Northridge

Outline

- Review Operation of Functions
- Review last lecture on pass-by-reference and pass-by-value
- Look at example where we need pass-by-reference
- Coding exercises with pass-by-value and pass-by-reference

Review 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

Review 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

Review Other Functions

- `void` functions do not return information to the calling program
  - A void function does not require a `return` statement
  - It may have a `return` statement that does not return a value, only control to the calling function
- Functions with an empty argument list do not receive information from the calling function

Example: Function for n!

```
double fact (int n )
{
  if ( n < 0 )  // error
    return 0;
  else if ( n == 0 || n == 1 )
    return 1;
  else {
    double fact = 1;
    for ( int m = n; m > 1; m-- )
       fact *= m;
    return fact;
  }
}
double fact( int n );
double fact( int );
```

Header
Type
Name
Argu-ment List
Body
Multiple returns
Possible prototypes

Review 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

Review Information Transfer

- 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

Review 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

Review 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; }
```


Review 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

Review Pass-by-Reference

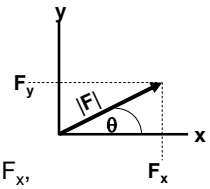
```
//calling program segment
double u = 3, v = 4;
cout << "fake = " fake( u, v );
cout << "\nu =" << u << ", v =" << v;
     // what is printed?
//function
double fake(double& x, double& y )
{
    x +=10; y *= x;  return 3 * y;
}
```

fake = 156
u = 13, v = 52

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

Example: Converting Vectors

- Convert different representations of two-dimensional vectors
  - Polar: magnitude, |**F**| and direction, θ
  - Rectangular: components, F_x, and F_y, along the x and y axes



- Conversion equations
  - $|\mathbf{F}| = (F_x^2 + F_y^2)^{1/2}$, $\theta = \tan^{-1}(F_y / F_x)$
  - $F_x = |\mathbf{F}|\cos\theta$, $F_y = |\mathbf{F}|\sin\theta$,

Converting Vectors II

- Write two functions to convert between the two different representations
  - Polar to rectangular function has magnitude and direction as inputs and returns x-component and y-component
  - Rectangular to polar function has x-component and y-component as inputs and returns magnitude and direction
  - Use atan2 function for $\theta = \tan^{-1}(F_y / F_x)$ to get full 2π result ($-\pi/2 < \text{atan result} < \pi/2$)

Converting Vectors III

- Each function has two input values and computes two results
- Use pass by reference to get results back to calling program
- Inputs to function are pass by value
- Function type can be void since function name need not return a value
  - Functions using pass by reference to return values sometimes return an error code in the function name

Converting Vectors IV

```
void polarToRectangular (
    double magnitude, double angle,
    double& xComponent,
    double& yComponent )
{
   xComponent = magnitude
              * cos( angle );
   yComponent = magnitude
              * sin( angle );
}
```


Converting Vectors V

```
void rectangularToPolar (
    double xComponent,
    double yComponent,
    double& magnitude, double& angle )
{
   magnitude = sqrt(
               pow( xComponent, 2 ) +
               pow( yComponent, 2 ) );
   angle = atan2 ( yComponent,
                   xComponent );
}
```


Use of Conversion Functions

```
const double PI = 4 * atan(1.0);
double x = 3, y = 4;
double A, theta;
rectangularToPolar(x, y, A, theta );
cout << x << "    " << y << "    "
     << A << "    " << theta << endl;
double size = 10, direction = PI / 8;
double xComp, yComp;
polarToRectangular( size, direction,
                    xComp, yComp );
cout << size << "  " << direction <<
      "  " << xComp << "  " << yComp;
```

Pass-by-Reference Exercise

- Write an input function that prompts the user to enter two type double variables, x and y, and returns these values to the calling program

```
void input( double& x, double& y)
{
    cout << "Enter x and y: ";
    cin >> x >> y;
}
```


Pass-by-Reference Exercise

```
void input( double& x, double& y)
{
    cout << "Enter x and y: ";
    cin >> x >> y
}
```

- Write the prototype for this function and a call to the function to get x and y

```
void input( double& x, double& y );
double x, y;
input ( x, y );
```

Optional

Use of Pass by Reference

- Calling programs use same approach for pass by reference and pass by value
- Variable or expression is placed as one of the arguments to the function
- Do not use a constant in a function call unless it is passed by value
- Data types (and ampersands for pass by reference) are not used in function call

What is Program Output?

```
void f1( int& x, int
  y )
{
   if (x >= y) {
      y = x + 1;
      cout << x << "
  "
           << y <<
  endl;
}
```

```
int x = 2,
    y = 3;
f1(y, x);
cout << x <<
<< y << endl;
f1( x, y );
cout << x <<
<< y << endl;
f1(y, x);
cout << x <<
<< y << endl;
f1( x, y );
cout << x;
```


What is Program Output? II

```
void f1( int& x, int
  y )
{
   if (x >= y) {
      y = x + 1;
      cout << x << "
  "
           << y <<
  endl;
}
```

```
in main
x = 2,y = 3;
f1(y, x);
f1(3, 2)
in f1
x = 3, y = 2
x >= y so
y = x + 1 = 4
y is pass-by-
value so there
is no change
in main
x = 2, y = 3
```


What is Program Output? III

```
void f1( int& x, int y )
{
   if (x >= y) {
      y = x + 1;
      cout << x << "   "
           << y << endl;
   }
   else {
      x = y + 1;
      cout << x << "   "
           << y << endl;
   }
}
```

```
in main
x = 2,y = 3;
f1(x, y);
f1(2, 3)
in f1
x = 2, y = 3
x < y so
x = y + 1 = 4
x is pass-by-
value so in
main
x = 4, y = 3
```

What is Program Output? IV

```
void f1( int& x, int y )
{
   if (x >= y) {
      y = x + 1;
      cout << x << "   "
           << y << endl;
   }
   else {
      x = y + 1;
      cout << x << "   "
           << y << endl;
   }
}
```

```
in main
x = 4,y = 3;
f1(y, x);
f1(3, 4)
in f1
x = 3, y = 4
x < y so
x = y + 1 = 5
x is pass-by-
value so in
main
y = 5, x = 4
```

California State University Northridge

25

What is Program Output? V

```
void f1( int& x, int y )
{
   if (x >= y) {
      y = x + 1;
      cout << x << "   "
           << y << endl;
   }
   else {
      x = y + 1;
      cout << x << "   "
           << y << endl;
   }
}
```

```
in main
x = 4,y = 5;
f1(x, y);
f1(4, 5)
in f1
x = 3, y = 4
x < y so
x = y + 1 = 6
x is pass-by-
value so in
main
x = 6, y = 5
```

California State University Northridge

26

Assignments

- Reading pages in text
  - Today: Pages 352-385
  - Thursday: Pages 63, 211-215, 341-351 1
  - Tuesday, April 4: none
- This week's homework problem
  - Page 216, checkpoint 4.27
- Exercise seven due Tuesday, April 4
- Lab quiz April 4 and Midterm April 6
  - Both cover up to and including looping

California State University Northridge

27