FUNCTION CALLING: PARAMETER PASSING

Function circle_area Call

```
double circle_area(double radius);

int main()
{
   double area, radius(5);
   area = circle_area(radius);
   . . .
}

double circle_area(double radius)
{
   double area;

   area = M_PI * radius * radius;
   return area;
}
```

Function Prototype

Function Call

Actual Parameter

Formal Parameter

Function Call

- The *actual parameter* can is an *expression:*

- Function **call**

```
double area, x(5);
area = circle_area(x);
```

- Function **header**

```
double circle_area(double radius);
```

Function Call

- The *actual parameter* can be a *const*
- Function **call**

```
double area;
const double MY_RADIUS(5);
area = circle_area(MY_RADIUS);
```

- Function **header**

```
double circle_area(double radius);
```

Function Call

- The *actual parameter* can be the *result of a calculation*

- Function **call**

```
double area, a(2), b(3);
area = circle_area(a + b - 1);
```

- Function **header**

```
double circle_area(double radius);
```

Pass By Value

- The *actual parameters* are *copied* into the matching *formal parameter* from left-to-right

- Function **call**

```
int x(10), y(5);
maxInt(x - y + 1, 2 * x + y)
```

- Function **header**

```
int maxInt(int a, int b)
```

Pass By Value

```
double circle_area(double radius);

int main()
{
   double area, radius(5);
   area = circle_area(radius);
   cout << radius << endl;
   . . .
}

double circle_area(double radius)
{
   double area;

   area = M_PI * radius * radius;
   radius = 10;

   return area;
}
```

Function Prototype

Function is called

What is displayed?

Consider this

Pass By Value

```
double circle_area(double radius);

int main()
{
   double area, radius(5);
   area = circle_area(radius);
   cout << radius << endl;
   . . .
}

double circle_area(double radius)
{
   double area;

   area = M_PI * radius * radius;
   radius = 10;

   return area;
}
```

Calls circle_area which sets radius to 10

Displays 5

Formal parameters are local variables.

Changing one has no affect outside the function

PASS BY REFERENCE

Pass By Reference

```
double circle_area(double & radius);

int main()
{
   double area, radius(5);
   area = circle_area(radius);
   cout << radius << endl;
   . . .
}

double circle_area(double & radius)
{
   double area;

   area = M_PI * radius * radius;
   radius = 10;

   return area;
}
```

Pass By Reference

Function is called

Displays 10

Pass By Reference

Pass By Reference

- The actual parameter must be a variable
  - Cannot be just any expression!
- For example, given prototype:

```
double circle_area(double & radius);
```

- The following function calls are ***illegal***:

```
circle_area(5);
circle_area(2 * 5);
circle_area(x + y - 1);
const double MY_RADIUS(25.3);
circle_area(MY_RADIUS); // MY_RADIUS is a constant
```

Notes on Pass By Reference

- The syntax for pass-by-reference parameters:
  `data-type & reference-name`
- The function *call* gives no indication whether a parameter is passed by reference
  - You need to look at the function definition to make sure
- If you do not pay attention to this, variables could be changed that you were not expecting to change!
- If a variable's value is not supposed to change, then it should (usually) be passed by value

Pass By Reference Example

- Suppose you want to write a function to compute and return the **area** and **perimeter** of a rectangle given its *length* and *width*

- What is the function prototype?

```
// not:
??? calc_rect(double l, double w);
```

- A function can only return **at most a single value**!

Pass By Reference Example

- Write a function to compute and return the **area** and **perimeter** of a rectangle given its *length* and *width*

- How about?

```
void calc_rect(double l, double w,
               double & area, double & perim);
```

void = no return value

Pass By Reference allows us to simulate passing multiple return values back

Pass By Reference

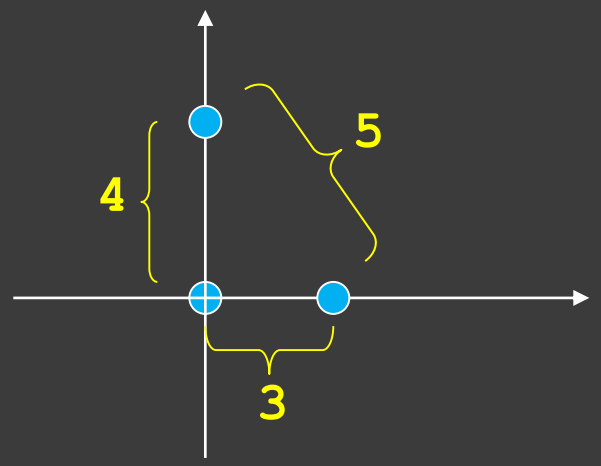
```
void calc_rect(double l, double w, double & area, double & perim);

int main()
{
  double a, p;
  calc_rect(10.0, 5.0, a, p);
  cout << "The area is " << a
       << " and the perimeter is " << p << endl;
  . . .
}

void calc_rect(double l, double w, double & area, double & perim)
{
  area = l * w;
  perim = 2 * (l + w);
}
```

Pass by reference supports returning 2 or more values back to the call

Remember: Compute distances between three points




```
> pointDist1.exe
Enter point 1 (2 floats): 0 0
Enter point 2 (2 floats): 3 0
Enter point 3 (2 floats): 0 4
Distance between (0,0) and (3,0) = 3
Distance between (0,0) and (0,4) = 4
Distance between (3,0) and (0,4) = 5
```

pointDist3.cpp

```
. . .
// main function
int main()
{
  double px, py, qx, qy, rx, ry;
  double dx, dy, dist_pq, dist_pr, dist_qr;

  // read input
  cout << "Enter point 1 (2 floats): ";
  cin >> px >> py;
  cout << "Enter point 2 (2 floats): ";
  cin >> qx >> qy;
  cout << "Enter point 3 (2 floats): ";
  cin >> rx >> ry;
```

Define a function for this code

pointDist5.cpp

```
. . .
// main function
int main()
{
  double px, py, qx, qy, rx, ry;
  double dist_pq, dist_pr, dist_qr;

  // read input
  inputCoord(1, px, py);
  inputCoord(2, qx, qy);
  inputCoord(3, rx, ry);
  . . .
}

void inputCoord(int i, double & x, double & y)
{
  cout << "Enter point " << i << " (2 floats): ";
  cin >> x >> y;
}
```

A CLOSER LOOK AT PASS-BY-REFERENCE

The `swap()` Function

- Write a function called `swap` with two input parameters (type double) that will exchange their values

- For example,

```
int main()
{
   double first(3.3), second(5.6);

   swap(first, second);

   // first now holds 5.6
   // second now holds 3.3
    // Must use pass by reference here!

    . . .
}
```

Function **swap()**

- Will this function definition work?

```
void swap(double & num1, double & num2)
{
   num1 = num2;
   num2 = num1;
}
```

NO! We need a 3rd variable to temporarily store **num1**'s original value.

Function swap()

- How about this solution?

```
void swap(double & num1, double & num2)
{
   double tmp;

   tmp = num1;
   num1 = num2;
   num2 = tmp;
}
```

Standard Template Library (STL): **swap()**

- The function swap is already defined in the library algorithm (part of STL)
  - *So just use that!*

```
#include <algorithm>
using namespace std;

int main()
{
   int a, b;
   double c, d;

. . .
   swap(a, b);  // swap 2 integers
   swap(c, d);  // swap 2 doubles
. . .
```

Pass by Reference Rule:

- Arguments of type int must be passed for parameters of type int
- Arguments of type double must be passed for parameters of type double
- Arguments of type string must be passed for parameters of type string
- etc.

Pass by Reference

- Function prototypes and headers (in the definition) must match. Variable names & parameter names do not need to match.

```
. . .
double calculate(double & a, int & b, double & c, int d);

int main()
{
  double e(0.0), g(2.5), result;
  int f(1), h(27);

  result = calculate(a, b, c, d);

  . . .

  return 0;
}

double calculate(double &j, int& k, double&m, int n)
{
  return j + k + m + n;
}
```

Function prototype

Function definition

What's the error?

```
. . .
void rect_calc(double & l, double & w,
               double & area, double & perim);

int main()
{
  double rectArea(0.0), rectPerim(0.0),
         rectLength(2.0), rectWidth(3.0);

  rect_calc(rectLength, rectWidth, rectArea, rectPerim);

  cout << "Area: " << rectArea << endl;
  cout << "Perimeter: " << rectPerim << endl;

  return 0;
}

void rect_calc(double l, double w,
               double & area, double & perim)
{
  area = l * w;
  perim = 2 * (l + w);
}
```

```
void rect_calc(double & l, double & w,
               double & area, double & perim);

int main()
{
   double rectArea(0.0), rectPerim(0.0),
          rectLength(2.0), rectWidth(3.0);

   rect_calc(rectLength, rectWidth, rectArea, rectPerim);
...
void rect_calc(double l, double w,
               double & area, double & perim)
{
   area = l * w;
   perim = 2 * (l + w);
}
```

```
> g++ rectCalcError1.cpp
/tmp/ccJFlGrU.o: In function `main':
rectCalcError1.cpp:(.text+0x4e): undefined reference to `rect_calc(double&, double&,
      double&, double&)'
collect2: ld returned 1 exit status
>
```

What's the error?

```
. . .
void rect_calc(double l, double w,
               double & area, double & perim);

int main()
{
  int rectArea(0.0), rectPerim(0.0),
      rectLength(2.0), rectWidth(3.0);

  rect_calc(rectLength, rectWidth, rectArea, rectPerim);

  cout << "Area: " << rectArea << endl;
  cout << "Perimeter: " << rectPerim << endl;

  return 0;
}

void rect_calc(double l, double w,
               double & area, double & perim)
{
  area = l * w;
  perim = 2 * (l + w);
}
```

```
8.   void rect_calc(double l, double w,
9.                    double & area, double & perim);
10.
11.  int main()
12.  {
13.    int rectArea(0.0), rectPerim(0.0),
14.        rectLength(2.0), rectWidth(3.0);
15.
16.    rect_calc(rectLength, rectWidth, rectArea, rectPerim);
...
28.  void rect_calc(double l, double w,
29.                    double & area, double & perim)
30.  {
31.    area = l * w;
32.    perim = 2 * (l + w);
33.  }
```

```
> g++ rectCalcError2.cpp
rectCalcError2.cpp: In function ‘int main()’:
rectCalcError2.cpp:16: error: invalid initialization of reference of type ‘double&’ from
    expression of type ‘int’
rectCalcError2.cpp:8: error: in passing argument 3 of ‘void rect_calc(double, double,
    double&, double&)’
>
```

What's the error?

```
. . .
void rect_calc(int l, int w,  int & area, int & perim);

int main()
{
  double rectArea(0.0), rectPerim(0.0),
         rectLength(2.0), rectWidth(3.0);

  rect_calc(rectLength, rectWidth, rectArea, rectPerim);

  cout << "Area: " << rectArea << endl;
  cout << "Perimeter: " << rectPerim << endl;

  return 0;
}

void rect_calc(int l, int w, int & area, int & perim)
{
  area = l * w;
  perim = 2 * (l + w);
}
```

```
8.   void rect_calc(int l, int w,
9.                    int & area, int & perim);
10.
11.  int main()
12.  {
13.    double rectArea(0.0), rectPerim(0.0),
14.           rectLength(2.0), rectWidth(3.0);
15.
16.    rect_calc(rectLength, rectWidth, rectArea, rectPerim);
...
28.  void rect_calc(int l, int w,
29.                    int & area, int & perim)
30.  {
31.    area = l * w;
32.    perim = 2 * (l + w);
33.  }
```

```
> g++ rectCalcError2.cpp
rectCalcError2.cpp: In function ‘int main()’:
rectCalcError2.cpp:16: error: invalid initialization of reference of type ‘double&’ from
    expression of type ‘int’
rectCalcError2.cpp:8: error: in passing argument 3 of ‘void rect_calc(double, double,
    double&, double&)’
>
```

Other Notes on Functions

- If a calculation or process needs to *execute repeatedly*, then make that a function
- Keep your functions from doing too much
- If a function is long and complex, it is hard to debug
- Split long and complex functions into more functions

CONST PARAMETERS

The `const` Parameter

- You can declare a formal parameter with `const`

```
void rect_calc(const double l, const double w,
               double & area, double & perim);
```

- Indicates that the parameter cannot be modified in the function definition

```
void rect_calc(const double l, const double w,
               double & area, double & perim)
{
  area = l * w;
  perim = 2 * (l + w);
}
```

The `const` Parameter

- Formal parameters defined with pass by reference (`&`) can also be declared with `const`!
- Though, why would you want to do this?
- Answer:
  - Pass by reference is used to efficiently pass large “object” values (pass by value is expensive since it makes a copy)
  - The `const` declaration ensures that it is not modified regardless of how they are passed

Using `const` With Pass By Reference

- The function `read_input()` asks the user for two numbers using **ANY** provided prompt

```
#include <string>
using namespace std;
...
void read_input(const string & prompt,
                double & num1, double & num2)
{
  cout << prompt;

  cin >> num1;
  cin >> num2;
}
```

Note

- Use pass by reference because the passed in string may be large (using pass by value would make a **COPY**, which is inefficient)
  - Pass by reference uses much less memory

Pass By Const Reference

```
double circle_area(const double &);

int main()
{
   const double radius(5);
   double area;
   area = circle_area(radius);
   cout << area << endl;
   . . .
}

double circle_area(const double & r)
{
  double area;

  area = M_PI * r * r;
  //r = 10; // no longer valid

  return area;
}
```

What is the error?

```
double circle_area(double &);

int main()
{
  const double radius(5);
  double area;
  area = circle_area(radius);
  cout << area << endl;
  . . .
}

double circle_area(double & r)
{
  double area;

  area = M_PI * r * r;

  return area;
}
```

```
1:  // Program name: example.cpp
2:  double circle_area(double &);
3:  int main()
4:  {
5:    const double radius(5);
6:    double area;
7:    area = circle_area(radius);
8:    cout << area << endl;
  . . .
12: }
13: double circle_area(double & r)
14: {
15:   double area;
16:   area = M_PI * r * r;
17:   return area;
18: }
```

```
> g++ rectCalcError2.cpp
example.cpp: In function ‘int main()’:
example.cpp:7: error: binding reference of type ‘double&’ to ‘cost double’ discards qualifiers
   7 |    area = circle_area(radius);
                              ^~~~~
>
```

FUNCTION EXERCISES

passExample1.cpp

```
...
void f(int a, int & b);

int main()
{
  int x(5), y(10);

  f(x, y);

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;

  return 0;
}
```

```
void f(int a, int & b)
{
  a++;
  b = 2 * b;

  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
}
```

What is the output?

Output:
a = 6
b = 20
x = 5
y = 20

passExample2.cpp

```
...
void f(int & a);
void g(int & b);

int main()
{
  int x(4);

  f(x);

  cout << "x = " << x << endl;

  return 0;
}
```

```
void f(int & a)
{
  a++;
  g(a);
  cout << "a = " << a << endl;
}

void g(int & b)
{
  b = 2 * b;
  cout << "b = " << b << endl;
}
```

What is the output?

Output: b = 10
a = 10
x = 10

What is the problem?

```
...
void f(int & a);
void g(int & b);

int main()
{
  int x(4);

  f(x);

  cout << "x = " << x << endl;

  return 0;
}
```

```
void f(int & a)
{
  a++;
  g(a);
  cout << "a = " << a << endl;
}

void g(int & b)
{
  b = 2 * b;
  f(b);
  cout << "b = " << b << endl;
}
```

passExample3.cpp

```
...
void f(int & a);

int main()
{
  int a(3), x(8);

  f(x);

  cout << "a = " << a << endl;
  cout << "x = " << x << endl;

  return 0;
}
```

```
void f(int & a)
{
  int x;

  a = 2 * a;
  x = 7;

  cout << "a = " << a << endl;
  cout << "x = " << x << endl;
}
```

What is the output?

Output: a = 16
x = 7
a = 3
x = 16

passExample4.cpp

```
...
void f(int a, int b);

int main()
{
  int x(5);

  f(x, x);

  cout << "x = " << x << endl;

  return 0;
}
```

```
void f(int a, int b)
{
  a++;
  b = 2 * b;

  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
}
```

What is the output?

Output: a = 6
b = 10
x = 5

passExample5.cpp

```
...
void f(int & a, int & b);

int main()
{
  int x(5);

  f(x, x);

  cout << "x = " << x << endl;

  return 0;
}
```

```
void f(int & a, int & b)
{
  a++;
  b = 2 * b;

  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
}
```

What is the output?

Output: a = 12
b = 12
x = 12