Reference Variables

Chapter 8

Page 383 - 408

Reference Variables

- Safer version of C/C++ pointer.
- "Refers to" a variable.
  - Like a pointer.
  - Effectively an *alias* for the variable to which it refers.
  - Used exactly like the the variable to which it refers.
    - No *
- & after type in a declaration makes it a reference.

Reference Variables

- You can declare reference *variables*
  - See pages 340 – 344
  - Must be initialized where declared.
    - Another name for an existing variable.
  - Not very useful!

- Function parameters can be references.
  - Reference parameter is an alias for the argument specified in the function call.
    - Not a copy!
  - Very useful!
  - "Call by reference"

Reference Variables as Function Parameters

- Example:
  - `void swap(int& a, int& b)`

- When a function parameter is a reference, the function has direct access to the variable passed by the caller.
  - *As if* the caller had passed a pointer.
  - But you don't dereference the parameter.
    - Use the parameter as if were the caller's variable.

Swap Example

- Let's implement a "swap" function using reference parameters
- Create a new empty project in Visual Studio.
- Add new item main.cpp

Reference Example: Swap

```
#include <iostream>

using namespace std;

void swap(int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

Arguments for Reference Parameters

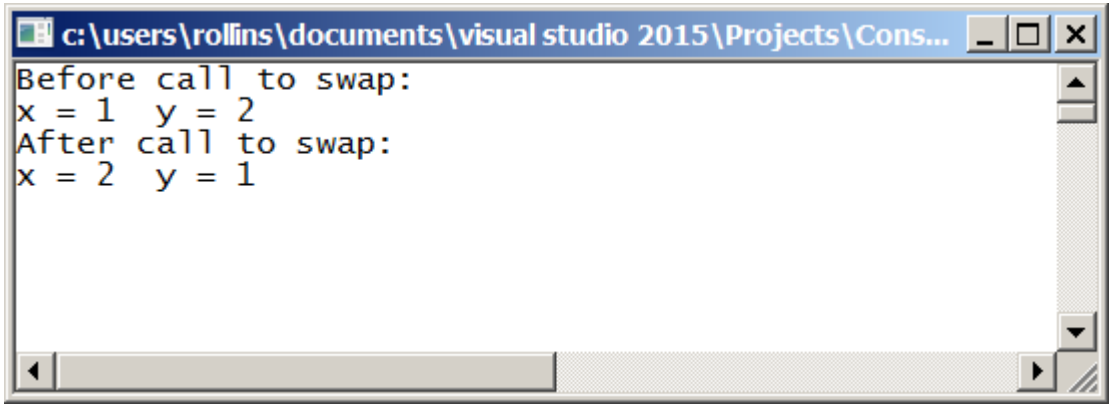
- Write the call to a function with reference parameters as if you were passing the values of the variables that you want to swap.
  - No “&”

- Note: You can't tell by looking at the call whether it is passing values or references.
  - You have to look at the function signature.

Reference Example: Swap

```
int main()
{
    int x = 1;
    int y = 2;

    cout << "Before call to swap: " << endl;
    cout << "x = " << x << "  y = " << y  << endl;

    swap (x, y);

    cout << "After call to swap: " << endl;
    cout << "x = " << x << "  y = " << y  << endl;

    cin.get();
    return 0;
}
```

Program swap Running



Things to Notice

- Reference parameters are initialized on the function call.
- Function acts on caller's variables.
  - as with pointers
- No dereferencing operator
  - Compare * for pointers
- Reference variable cannot be made to refer to a different memory location.
  - Consider: **`a = b;`**

References to Objects

- Download project Circle_Demo

http://www.csee.usf.edu/~turnerr/Object_Oriented_Design/Downloads/2016_01_29/

  - Expanded and improved version of our original Circle program.

- Extract All
- Open project in Visual Studio

Circle.h

```
#pragma once
#include <string>
using namespace std;
class Circle
{
    private:
        string name;
        double radius;

    public:
        Circle(string Name, double Radius);
        string Name() const { return name; };
        double Radius() const { return radius; };
        void Display() const;
        double Area() const;
};
```

Circle.cpp

```
#include "Circle.h"
#include <iostream>

Circle::Circle(string name_, double radius_)
{
    name = name_;
    radius = radius_;
}
```

Circle.cpp

```
double Circle::Area() const
{
    return 3.141592 * radius * radius;
}

void Circle::Display() const
{
    cout << name << " is a Circle of radius "
         << radius << std::endl;
}
```

Circle_Test.cpp

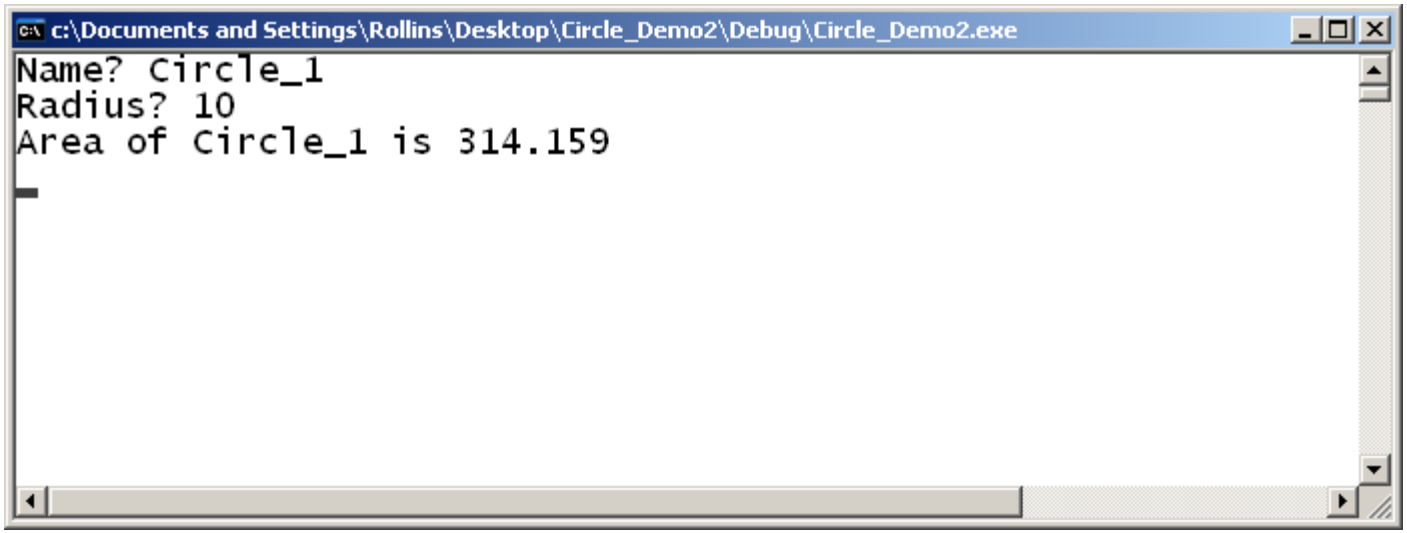
```
#include <iostream>
#include <string>
#include "Circle.h"

Circle* Create_Circle()
{
    string name;
    double radius;
    cout << "Name? ";
    getline(cin, name);
    cout << "Radius? ";
    cin >> radius;
    cin.get();              // Clear keyboard input buffer
    return new Circle(name, radius);
}
```

Circle_Test.cpp

```
int main()
{
    Circle* c1 = Create_Circle();
    double c1_area = c1->Area();
    cout << "Area of " << c1->Name() << " is " << c1_area << endl;

    cin.get();   // Hold window open
    cin.get();
    return 0;
}
```

Build and run.

Program Running



Comparing Circles

- Let's add a method to class Circle to permit users to compare Circles.
- Pass Circle to compare by reference
- In Circle.h:

```
bool Is_Greater_Than(const Circle& other) const;
```

Comparing Circles

In Circle.cpp

```
bool Circle::Is_Greater_Than(const Circle& other) const
{
    return this->radius > other.radius;
}
```

Comparing Circles

```
int main()
{
    Circle* c1 = Create_Circle();
    double c1_area = c1->Area();
    cout << "Area of " << c1->Name() << " is " << c1_area << endl;

    Circle* c2 = Create_Circle();
    double c2_area = c2->Area();
    cout << "Area of " << c2->Name() << " is " << c2_area << endl;
```

Comparing Circles

```
if (c1->Is_Greater_Than(*c2))
{
    cout << c1->Name() << " is greater than "
         << c2->Name() << endl;
}
else if (c2->Is_Greater_Than(*c1))
{
    cout << c2->Name() << " is greater than "
         << c1->Name() << endl;
}
else
{
    cout << c1->Name() << " and " << c2->Name()
         << " are the same size\n";
}
```

Program Running

c:\Documents and Settings\Rollins\Desktop\Circle_Demo2\Debug\Circle_Demo2.exe

```
Name? Circle_1
Radius? 10
Area of Circle_1 is 314.159
Name? Circle_2
Radius? 5
Area of Circle_2 is 78.5398
Circle_1 is greater than Circle_2
```

Summary

- References are good!
  - Use them as function parameters instead of pointers in C++.
- Make reference parameters **`const`** whenever possible.
  - Also the function itself!