



# COMP2012 Object-Oriented Programming and Data Structures

## Review: References

Dr. Desmond Tsoi

Department of Computer Science & Engineering  
The Hong Kong University of Science and Technology  
Hong Kong SAR, China



# What are References?

- A **reference** is **an alternative / another name (alias)** for a variable or an object
- Reference variables are usually used in parameter passing to functions

## Syntax:

```
<type>& <variable name> = <variable>;  
<type> &<variable name> = <variable>; OR
```

where **<type>** is the type of the variable address that the pointer variable can store (e.g. int, char, double, user-defined type), **<variable name>** is the name of the reference variable, **<variable>** is a variable of the same type as **<type>**

```
<type>& p = q;  
<type> &p = q;
```



# Declaration of Reference Variables

- Recall, the syntax for declaring a reference variable:

```
<type>& <variable name> = <variable>;
```

```
// Declare an int variable
```

```
int j = 1; // the value of j is 1
```

```
// Declare a reference variable r, which is another name of j
```

```
int& r = j; // now r = j = 1
```

```
// Assign r to x
```

```
int x = r; // now x = 1
```

```
// As r is just j, r changes to 2 means j changes to 2 as well
```

```
r = 2; // now r = j = 2
```

```
j = 10; // now r = j = 10
```

A reference allows indirect manipulation of a variable / object, somewhat like a pointer, without requiring the complicated pointer syntax ;)

# Important Points about Reference

- A reference **MUST** always bound to a variable / an object (similar to constant pointer)
- It must therefore be **initialized when it is created**

```
int j = 1;  
int& r1 = j; // ok  
int& r2;      // error!
```

- A reference cannot bound to another object once it initially was **initialized to**. But what does the following mean?

```
int j = 10;  
int& r = j;  
int k = 50;  
r = k;      // what does this mean?
```

Assignment from a variable, say k, to a reference variable, say r, means r get assigned with the value of k

# The Different Uses of Operator &

- Do not confuse the use of operator & in declaring reference variable versus the use of & as the address of operator for pointer
- Example

```
int j;
```

```
// The following statement declares a reference variable
```

```
// "&" is a part of reference type
```

```
int& i = j;
```

```
// The following statement is used to get the address
```

```
// of j and assign it to p
```

```
int* p = &j;
```



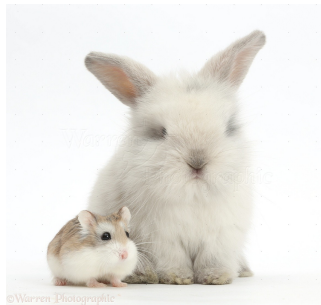
# Questions

- The following is wrong. Why?

```
int j;  
int& i = &j;
```

- The following is correct. What does it mean?

```
int j;  
int* p = &j;  
int*& ref = p;  
// a reference of "int pointer"
```



## Example

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

int main() {
    int j = 1;
    // pi is an int pointer initialized to the address of j
    int* pi = &j;
    int*& ref = pi; // ref is a reference variable of type int*
    cout << "j = " << j << ", ";
    cout << "*pi = " << *pi << ", *ref = " << *ref << endl;

    int k = 2;
    pi = &k;
    cout << "j = " << j << ", ";
    cout << "*pi = " << *pi << ", *ref = " << *ref << endl;
    return 0;
}
```

### Output:

```
j = 1, *pi = 1, *ref = 1
j = 1, *pi = 2, *ref = 2
```

# Call by Reference

- Reference arguments are special case of reference variable

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

int f(int& i) {
    ++i;
    return i;
}

int main() {
    int j = 7;
    cout << f(j) << endl;
    cout << j << endl;
    return 0;
}
```



**Output:**

8  
8



# Call by Reference

- Variable `i` is a local variable in the function `f`. Its type is “int reference” and it is created when `f` is called
- In the call `f(j)`, `i` is created similarly to the construction:  
`int& i = j;`
- So within the function `f`, `i` will be an alias of the variable `j`, and `i` cannot be binded to another variable
- But every time the function is called, a new variable `i` is created and it can be a reference to a different variable / object



# Why Call by Reference?

There are **two reasons**:

1. The function caller wants the function to be able to change the value of passed arguments
2. For efficiency:
  - ▶ If you pass a function argument by value, the function gets a local copy of the argument
  - ▶ For large objects, copying is expensive; on the other hand, passing an object by reference does not require copying, only a memory address (since reference is similar to pointer)



# Example

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

void swap(char& y, char& z) {
    char temp = y;
    y = z;
    z = temp;
}

int main() {
    char a = 'y';
    char b = 'n';
    swap(a, b);
    cout << a << b << endl;
    return 0;
}
```

Output:  
ny



## const: References as Function Arguments

- You can express your intention to leave a reference argument of your function unchanged by making it `const`
- This has two advantages:
  - ▶ First, if you accidentally try to modify the argument in your function, the compiler will catch the error!

```
void cbr(int& i) {  
    i += 10; // Fine  
}
```

```
void cbcr(const int& j) {  
    j += 10; // Error!  
}
```

## const: References as Function Arguments (Cont'd)

- Second, you can call a function that has a const reference parameter with both const and non-const arguments
- Conversely, a function that has a non-const reference parameter can only be called with non-const arguments



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

void cbr(int& i) {
    cout << i << endl;
}

void cbcr(const int& i) {
    cout << i << endl;
}

int main() {
    int i = 50;
    const int j = 100;
    cbr(i);
    cbcr(i);
    cbr(j); // Error
    cbcr(j);
    return 0;
}
```

# Pointer vs. Reference

A reference can be thought of as a special kind of pointer, but there are **3 big differences to remember!**

- A pointer can point to nothing (nullptr), but a reference is always bound to a variable / object
  - ▶ No need to check nullptr when using reference ;)
- A pointer can point to different variables / objects at different times (through assignments). A reference is always bound to the same variable / object.
  - ▶ Assignments to a reference do NOT change the variable / object it refers to but only the value of the referenced object
- The name of a pointer refers to the pointer variable. The \* or – > (details of operator – > will be covered later) operators have to be used to access the object. The name of a reference always refers to the same object. There are no special operators. ;)

# Example - Pointers and References

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

void func1(int* p1) { (*p1)++; }
void func2(int& ri) { ri++; }

int main() {
    int i = 1;
    cout << "i = " << i << endl;
    // call using address of i
    func1(&i);
    cout << "i = " << i << endl;
    // call using i
    func2(i);
    cout << "i = " << i << endl;
    return 0;
}
```

Output:

```
i = 1
i = 2
i = 3
```



## Further Reading

- Read Chapter 6 & 8 of "C++ How to Program" or Chapter 3 of "C++ Primer" textbook





That's all!

Any question?

