



Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

Module 07: Programming in C++

Reference & Pointer

Instructors: Abir Das and Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

`{abir, sourangshu}@cse.iitkgp.ac.in`

Slides taken from NPTEL course on Programming in Modern C++
by **Prof. Partha Pratim Das**



Module Objectives

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives & Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

- Understand References in C++
- Compare and contrast References and Pointers



Module Outline

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives & Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

- Reference variable or Alias
 - Basic Notion
 - Call-by-reference in C++
- Example: Swapping two number in C
 - Using Call-by-value
 - Using Call-by-address
- Call-by-reference in C++ in contrast to Call-by-value in C
- Use of const in Alias / Reference
- Return-by-reference in C++ in contrast to Return-by-value in C
- Differences between References and Pointers



Reference

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

- A reference is an **alias** / **synonym** for an existing variable

```
int i = 15; // i is a variable
```

```
int &j = i; // j is a reference to i
```

Anything you do to j affects i, and vice versa, because they are the same memory location.

i ← variable

15 ← memory content

200 ← address **&i = &j**

j ← alias or reference

```
int i = 15; // i is a variable
```

```
int *j = &i; // j is a pointer to i
```

Here, j stores the address of i.

To access or modify i, you have to use *j (dereferencing)

```
*j=20;
```



Program 07.01: Behavior of Reference

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C
Swap in C++
const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

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

int main() {
    int a = 10, &b = a; // b is reference of a

    // a and b have the same memory location
    cout << "a = " << a << ", b = " << b << ". " << "&a = " << &a << ", &b = " << &b << endl;

    ++a; // Changing a appears as change in b
    cout << "a = " << a << ", b = " << b << endl;

    ++b; // Changing b also changes a
    cout << "a = " << a << ", b = " << b << endl;
}
```

```
a = 10, b = 10. &a = 002BF944, &b = 002BF944
a = 11, b = 11
a = 12, b = 12
```

- a and b have the *same memory location* and hence *the same value*
- Changing one changes the other and vice-versa



Pitfalls in Reference

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C
Swap in C++
const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

Wrong declaration	Reason	Correct declaration
<pre>int& i; int& j = 5; int& i = j + k;</pre>	<p>no variable (address) to refer to – must be initialized</p> <p>no address to refer to as 5 is a <i>constant</i></p> <p>only temporary address (result of <code>j + k</code>) to refer to</p>	<pre>int& i = j; const int& j = 5; const int& i = j + k;</pre>
<pre>#include <iostream> using namespace std; int main() { int i = 2; int& j = i; const int& k = 5; // const tells compiler to allocate a memory with the value 5 const int& l = j + k; // Similarly for j + k = 7 for l to refer to cout << i << ", " << &i << endl; // Prints: 2, 0x61fef8 cout << j << ", " << &j << endl; // Prints: 2, 0x61fef8 cout << k << ", " << &k << endl; // Prints: 5, 0x61fefc cout << l << ", " << &l << endl; // Prints: 7, 0x61ff00 }</pre>		



C++ Program 07.02: Call-by-reference

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C
Swap in C++
const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

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

```
void Function_under_param_test( // Function prototype
    int&, // Reference parameter
    int); // Value parameter
```

```
int main() { int a = 20;
    cout << "a = " << a << ", &a = " << &a << endl << endl;
    Function_under_param_test(a, a); // Function call
}
```

```
void Function_under_param_test(int &b, int c) { // Function definition
    cout << "b = " << b << ", &b = " << &b << endl << endl;
    cout << "c = " << c << ", &c = " << &c << endl << endl;
}
```

----- Output -----

a = 20, &a = 0023FA30

b = 20, &b = 0023FA30 // Address of b is same as a as b is a reference of a

c = 20, &c = 0023F95C // Address different from a as c is a copy of a

- Param **b** is *call-by-reference* while param **c** is *call-by-value*
- Actual param **a** and formal param **b** get the *same value* in called function
- Actual param **a** and formal param **c** get the *same value* in called function
- Actual param **a** and formal param **b** get the *same address* in called function
- However, actual param **a** and formal param **c** have *different addresses* in called function



C Program 07.03: Swap in C

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

Call-by-value – wrong

```
#include <stdio.h>

void swap(int, int); // Call-by-value
int main() { int a = 10, b = 15;
    printf("a= %d & b= %d to swap\n", a, b);
    swap(a, b);
    printf("a= %d & b= %d on swap\n", a, b);
}

void swap(int c, int d) { int t;
    t = c; c = d; d = t;
}
```

- a= 10 & b= 15 to swap
- a= 10 & b= 15 on swap // No swap

- Passing values of a=10 & b=15
- In callee; c = 10 & d = 15
- Swapping the values of c & d
- No change for the values of a & b in caller
- Swapping the value of c & d instead of a & b

Call-by-address – right

```
#include <stdio.h>

void swap(int *, int *); // Call-by-address
int main() { int a=10, b=15;
    printf("a= %d & b= %d to swap\n", a, b);
    swap(&a, &b); // Unnatural call
    printf("a= %d & b= %d on swap\n", a, b);
}

void swap(int *x, int *y) { int t;
    t = *x; *x = *y; *y = t;
}
```

- a= 10 & b= 15 to swap
- a= 15 & b= 10 on swap // Correct swap

- Passing Address of a & b
- In callee x = Addr(a) & y = Addr(b)
- Values at the addresses is swapped
- Desired changes for the values of a & b in caller
- It is correct, but C++ has a better way out



Program 07.04: Swap in C & C++

Module 07

Instructors: Abir Das and Sourangshu Bhattacharya

Objectives & Outlines

Reference variable

Call-by-reference

Swap in C

Swap in C++

const Reference Parameter

Return-by-reference

I/O of a Function

References vs. Pointers

Summary

C Program: Call-by-value – wrong

```
#include <stdio.h>

void swap(int, int); // Call-by-value
int main() { int a = 10, b = 15;
    printf("a= %d & b= %d to swap\n",a,b);
    swap(a, b);
    printf("a= %d & b= %d on swap\n",a,b);
}

void swap(int c, int d) { int t ;
    t = c; c = d; d = t;
}
```

- a= 10 & b= 15 to swap
- a= 10 & b= 15 on swap // No swap

- Passing values of a=10 & b=15
- In callee; c = 10 & d = 15
- Swapping the values of c & d
- No change for the values of a & b in caller
- Here c & d do not share address with a & b

C++ Program: Call-by-reference – right

```
#include <iostream>
using namespace std;
void swap(int&, int&); // Call-by-reference
int main() { int a = 10, b = 15;
    cout<<"a= "<<a<<" & b= "<<b<<"to swap"<<endl;
    swap(a, b); // Natural call
    cout<<"a= "<<a<<" & b= "<<b<<"on swap"<<endl;
}

void swap(int &x, int &y) { int t ;
    t = x; x = y; y = t;
}
```

- a= 10 & b= 15 to swap
- a= 15 & b= 10 on swap // Correct swap

- Passing values of a = 10 & b = 15
- In callee: x = 10 & y = 15
- Swapping the values of x & y
- Desired changes for the values of a & b in caller
- x & y having same address as a & b respectively



Program 07.05: Reference Parameter as const

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

- A reference parameter may get changed in the called function
- Use **const** to stop reference parameter being changed

const reference – bad

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

int Ref_const(const int &x) {
    ++x;      // Not allowed
    return (x);
}

int main() { int a = 10, b;
    b = Ref_const(a);
    cout << "a = " << a << " and"
         << " b = " << b;
}
```

- **Error:** Increment of read only Reference 'x'

- **Compilation Error:** Value of x cannot be changed
- Implies, a cannot be changed through x

const reference – good

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

int Ref_const(const int &x) {
    return (x + 1);
}

int main() { int a = 10, b;
    b = Ref_const(a);
    cout << "a = " << a << " and"
         << " b = " << b;
}
```

a = 10 and b = 11

- **No violation**



Program 07.06: Return-by-reference

- A function can return a value by reference (**Return-by-Reference**)

- C uses **Return-by-value**

In C++, a non-const reference (int &b) can only bind to:

A variable with a name (like a), not

A temporary object (like the returned copy).

So this fails because we are trying to make b a reference to something that will be destroyed imm

Return-by-value

```
#include <iostream>
using namespace std;
int Function_Return_By_Val(int &x) {
    cout << "x = " << x << " &x = " << &x << endl;
    return (x);
}
int main() { int a = 10;
    cout << "a = " << a << " &a = " << &a << endl;
    const int& b = // const needed. Why?
        Function_Return_By_Val(a);
    cout << "b = " << b << " &b = " << &b << endl;
}
```

```
a = 10 &a = 00DCFD18
x = 10 &x = 00DCFD18
b = 10 &b = 00DCFD00 // Reference to temporary
```

- Returned variable is **temporary**
- Has a **different address**

CS20202: Software Engineering

Return-by-reference

```
#include <iostream>
using namespace std;
int& Function_Return_By_Ref(int &x) {
    cout << "x = " << x << " &x = " << &x << endl;
    return (x);
}
int main() { int a = 10;
    cout << "a = " << a << " &a = " << &a << endl;
    const int& b = // const optional
        Function_Return_By_Ref(a);
    cout << "b = " << b << " &b = " << &b << endl;
}
```

```
a = 10 &a = 00A7F8FC
x = 10 &x = 00A7F8FC
b = 10 &b = 00A7F8FC // Reference to a
```

- Returned variable is **an alias of a**
- Has the **same address**

Instructors: Abir Das and Sourangshu Bhattacharya



Program 07.07: Return-by-reference can get tricky

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

Return-by-reference

```
#include <iostream>
using namespace std;
int& Return_ref(int &x) {

    return (x);
}

int main() { int a = 10, b = Return_ref(a);
    cout << "a = " << a << " and b = "
        << b << endl;

    Return_ref(a) = 3; // Changes variable a
    cout << "a = " << a;
}
```

a = 10 and b = 10
a = 3

- Note how *a value is assigned to function call*
- This can change a local variable

Return-by-reference – **Risky!**

```
#include <iostream>
using namespace std;
int& Return_ref(int &x) {
    int t = x;
    t++;
    return (t);
}

int main() { int a = 10, b = Return_ref(a);
    cout << "a = " << a << " and b = "
        << b << endl;

    Return_ref(a) = 3; // Changes local t
    cout << "a = " << a;
}
```

a = 10 and b = 11
a = 10

- We expect a to be 3, *but it has not changed*
- It returns reference to local. This is *risky*



I/O of a Function

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

- In C++ we can change values with a function as follows:

I/O of Function	Purpose	Mechanism
Value Parameter	Input	Call-by-value
Reference Parameter	In-Out	Call-by-reference
<code>const</code> Reference Parameter	Input	Call-by-reference
Return Value	Output	Return-by-value Return-by-reference <code>const</code> Return-by-reference

- In addition, we can use the **Call-by-address** (**Call-by-value** with pointer) and **Return-by-address** (**Return-by-value** with pointer) as in C
- But it is neither required nor advised



Recommended Mechanisms

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

- **Call**

- Pass parameters of **built-in types** *by value*
 - ▷ Recall: *Array parameters* are passed *by reference* in **C** and **C++**
- Pass parameters of **user-defined types** *by reference*
 - ▷ Make a *reference parameter* **const** if it is not used for output

- **Return**

- Return **built-in types** *by value*
- Return **user-defined types** *by reference*
 - ▷ Return value *is not copied back*
 - ▷ May be *faster* than returning a value
 - ▷ **Beware:** Calling function *can change returned object*
 - ▷ **Never return a local variables by reference**



Difference between Reference and Pointer

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

Pointers

- Refers to an *address (exposed)*
- Pointers can point to **NULL**

```
int *p = NULL; // p is not pointing
```

- Pointers can point to *different variables* at *different times*

```
int a, b, *p;
```

```
p = &a; // p points to a
```

```
...
```

```
p = &b; // p points to b
```

- **NULL** checking *is required*
- *Allows* users to *operate on the address*
- diff pointers, increment, etc.
- *Array of pointers* can be *defined*

References

- Refers to an *address (hidden)*
- References cannot be **NULL**

```
int &j ; // wrong
```

- For a reference, its *referent is fixed*

```
int a, c, &b = a; // Okay
```

```
...
```

```
&b = c // Error
```

- *Does not require* **NULL** checking
- Makes code *faster*
- *Does not allow* users to *operate on the address*
- All operations are interpreted for the referent
- *Array of references* *not allowed*



Module Summary

Module 07

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

Reference
variable

Call-by-reference

Swap in C

Swap in C++

const Reference
Parameter

Return-by-
reference

I/O of a Function

References vs.
Pointers

Summary

- Introduced reference in C++
- Studied the difference between call-by-value and call-by-reference
- Studied the difference between return-by-value and return-by-reference
- Discussed the difference between References and Pointers