

# CS2313 Computer Programming

## LT7 – Function



香港城市大學  
City University of Hong Kong

專業 創新 胸懷全球  
Professional • Creative  
For The World

# Problem

```
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
cout << "Sum from 1 to 10 is " << sum << endl;

sum = 0;
for (int i = 20; i <= 37; i++)
    sum += i;
cout << "Sum from 20 to 37 is " << sum << endl;

sum = 0;
for (int i = 35; i <= 49; i++)
    sum += i;
cout << "Sum from 35 to 49 is " << sum << endl;
```

# Problem

```
int sum = 0;  
for (int i = 1; i <= 10; i++)  
    sum += i;
```

```
cout << "Sum from 1 to 10 is " << sum << endl;
```

```
sum = 0;  
for (int i = 20; i <= 37; i++)  
    sum += i;
```

```
cout << "Sum from 20 to 37 is " << sum << endl;
```

```
sum = 0;  
for (int i = 35; i <= 49; i++)  
    sum += i;
```

```
cout << "Sum from 35 to 49 is " << sum << endl;
```

# Solution

```
int sum(int i1, int i2)
{
    int sum_value = 0;
    for (int i = i1; i <= i2; i++)
        sum_value += i;
    return sum_value;
}
```

```
int main()
{
    cout << "Sum from 1 to 10 is " << sum(1, 10) << endl;
    cout << "Sum from 20 to 37 is " << sum(20, 37) << endl;
    cout << "Sum from 35 to 49 is " << sum(35, 49) << endl;
    return 0;
}
```

# Syntax Summary

- Keywords
  - `return, void.`
- Punctuators
  - `() , , .`

# Outcomes

- Function declaration.
- Parameter passing, return value.
- Passing array to a function.
- Function Prototype.
- Recursive Function.

# What is Function?

- A **collection** of statements that perform a specific task.
- Functions are used to break a problem down into **manageable** pieces.
- Write several small functions to solve a **specific** part of the problem.
- A function can be invoked **multiple** times. No need to repeat the same code in multiple parts of the program file.

# Structured Programming Guidelines

- Flow of control in a program should be as simple as possible
- Construction of program should embody top-down stepwise refinement design.
  - Stepwise refinement is an approach of **decomposing a problem into small problems** repeatedly until they are simple enough to be coded easily.
  - From another perspective, each problem can be viewed from **different levels of abstraction** (or details).
  - Top-down approach of problem solving is well exercised by human beings.



# Function in C++

- The C++ standard library provides a rich collection of functions for:
  - Mathematical calculations (`#include <cmath>`)
  - String manipulations (`#include <cstring>`)
  - Input/output (`#include <iostream>`).
  - etc.
- Note - some functions are defined in multiple library in some platform, e.g. function `sqrt` is defined in both `cmath` and `iostream` in VS2017

# How to Use Function Written by Others?

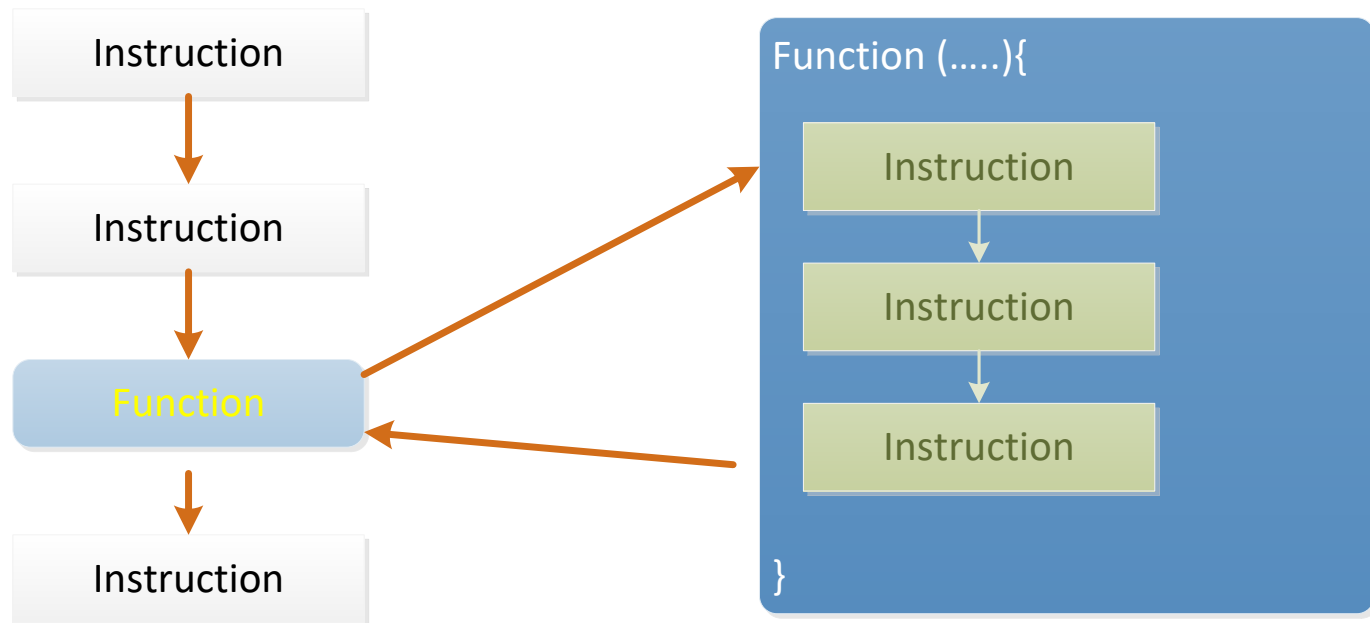
```
#include <iostream>
using namespace std;
void main()
{
    float area, side;
    cout << "Enter the area of a square:";
    cin >> area;
    side=sqrt(area);
    cout << "The square has perimeter: " << 4*side;
}
```

Tell compiler that you are going to use functions defined in `iostream` package.

Pass area to the function `sqrt` which will return the square root of area.

# Function Invocation

- During program execution, when a function name followed by parentheses is encountered:
  - the function is invoked and the program control is passed to that function;
  - when the function ends, program control is returned to the statement immediately after the function call in the original function.



# Write Your Own Function (User Defined Functions)

- ***Example:***
  - Define a function `printHello` which accepts an integer `n` as input.
  - The function should print "Hello" `n` times, where `n` is an integer.

# Function Components

Return type

Function name

Input (Parameter/Argument)



```
void printHello (int n) {  
    int i;  
    for (i=0; i<n; i++)  
        cout <<"Hello\n";  
}
```

Function body

`n` is defined as input, therefore there is no need to declare `n` in the function body again

# Calling A Function (I)

To make a function call, we only need to specify a function name and provide parameter(s) in a pair of ().

Function name	Input
	
<code>printHello</code>	<code>(3);</code>

Print "hello" 3 times.

We don't need the return type when calling a function.

**Syntax error:**

```
int printHello (3);
```

# Calling A Function (II)

```
int x=4;  
printHello (x);  
printHello (x+2);
```

Print "hello" 4 times and then 6 times.

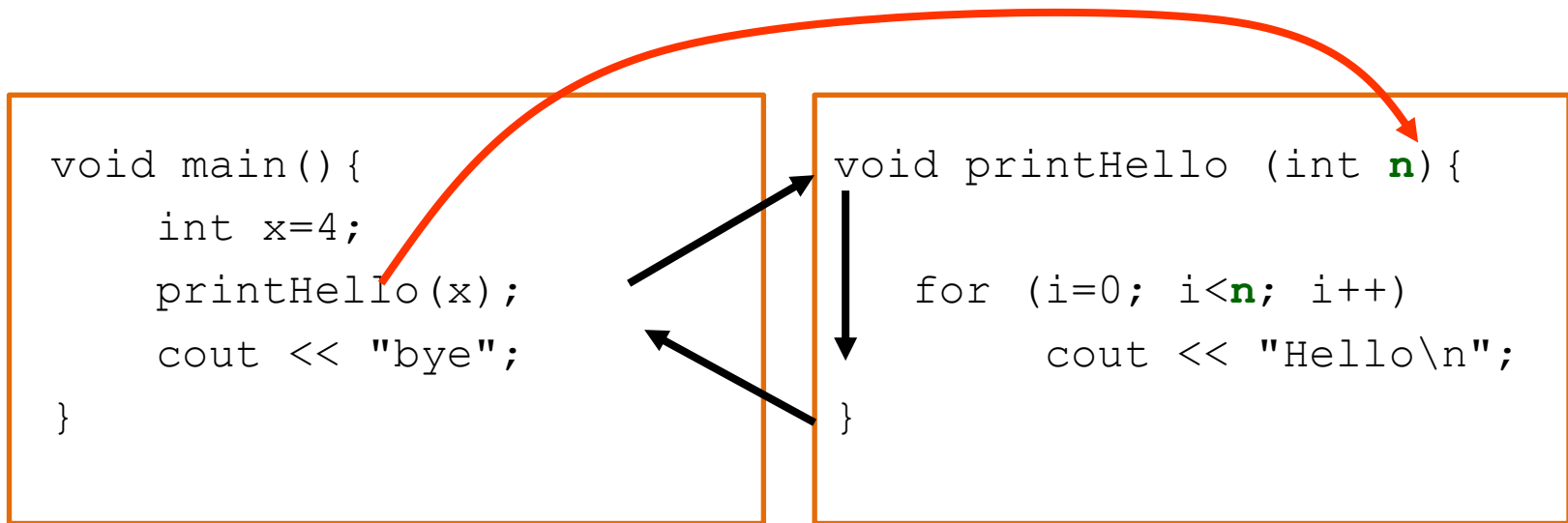
We don't need the parameter type when calling a function.

**Syntax error:**

```
printHello (int x);
```

Advantage of using a function: we don't need to write two loops, one to print "Hello" 4 times and the other to print "Hello" 6 times.

# Flow of Control



1. The program first start execution in `main()`.
2. `printHello(x)` is called.
3. The value of `x` is copied to the variable `n`. As a result, `n` gets a value of 4.
4. The loop body is executed.
5. After executing all the statements within `printHello()`, control go back to `main()` and "bye" is printed.



# Function with No Input

```
void printHello () {  
    cout << "hello";  
}
```

# Examples: Function w/o Return Value

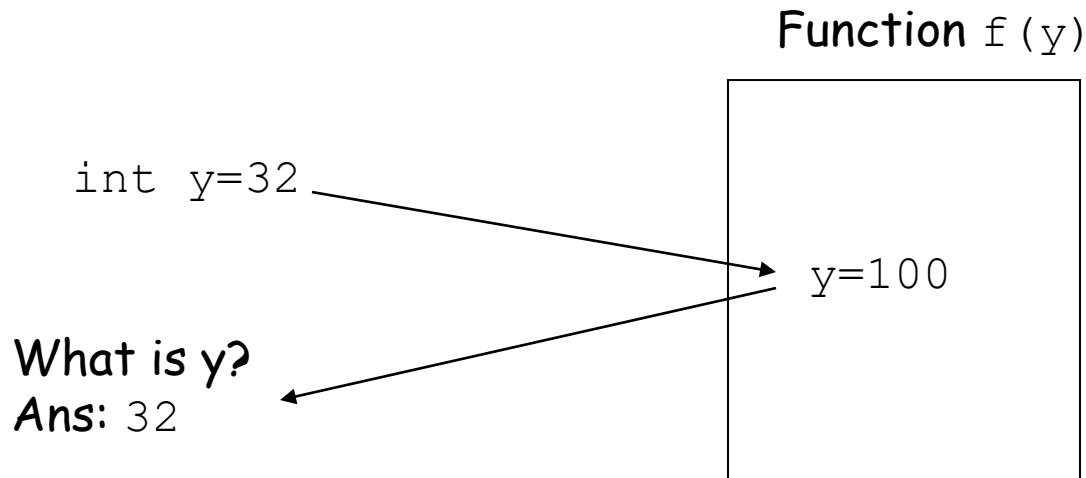
Function	Parameter	Return Value	Examples:
printHello	nil	nil	<pre>void printHello() {     cout &lt;&lt; "Hello\n"; }</pre>
printHellos	int n	nil	<pre>void printHello(int n) {     int i;     for (i=0;i&lt;n;i++)         cout &lt;&lt; "Hello\n"; }</pre>
printMax	float n1 float n2	nil	<pre>void printMax(float n1,float n2){     cout &lt;&lt; ((n1&gt;n2)?n1:n2)&lt;&lt; endl; }</pre>
printFloats	int n float data	nil	<pre>void printFloats(int n, float data){     ..... } or void printFloats(float data,int n){     ..... }</pre>

# Examples: Function with Return Value

Function	Parameter	Return Value	Examples:
getX	nil	int	<pre>int getX () {     int d;     cin &gt;&gt; d;     return d; }</pre>
calMax	float f1 float f2	float	<pre>float calMax (float f1,float f2){     if (f1&gt;f2)         return f1;     else         return f2; }</pre>
getInput	int n1, float n2	char	<pre>char getInput(int n1,float n2){     char c;     .....     return c; }</pre>

# Parameters Passing: Call-by-Value

- When a function is invoked, the arguments within the parentheses are passed using a *call-by-value*.
- Each argument is evaluated, and its value is used locally in place of the corresponding formal parameter.



# Parameters Passing: Call-by-Value

```
void f (int x){  
    x=4; //we modify the value x to 4  
    //Do we modify y at the same time? NO  
    y=4; //syntax error: y is local to main  
}  
  
void main(){  
    int y=3;  
    f(y);  
    cout << y; //print 3, y remains unchanged  
}
```

The variable `x` and `y` are local variables.

`x` is local to `f()`, so we cannot use `x` in `main()`.

`y` is local to `main()`, so we cannot use `y` in `f()`.

`x` and `y` are two independent variables. When `x` is modified, `y` will not be affected.

# What if We Change $x$ to $y$ in $f()$ ?

```
void f (int y) {  
    y=4; //modify y in f(), not the one in main()  
}  
  
void main() {  
    int y=3;  
    f(y);  
    cout << y; //print 3, y remains unchanged  
}
```

In this program, there are two variables called  $y$ .

One is defined in  $f()$  and one is defined in  $main()$ .

In  $f()$ ,  $y$  in  $f()$  is modified.

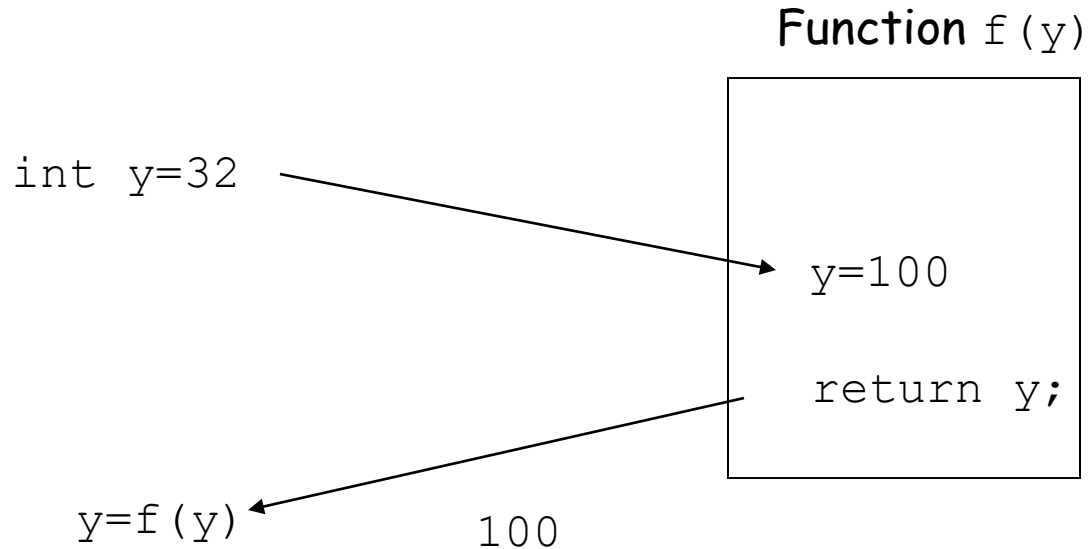
However,  $y$  in  $main()$  is not affected.

# How to Modify $y$ in $f()$ ?

```
int f (int x){  
    x=4; //we modify the value x to 4  
    //Do we modify y at the same time? NO  
    return x;  
}  
  
void main() {  
    int y=3;  
    y=f(y) ;  
    cout << y; //print 4  
}
```

By assigning the return value of  $f(y)$  to  $y$ ,  
After the function call,  $y$  gets a value of 4 .

# Return Value



We assign the return value of  $f(y)$  to the variable  $y$ .  
What is  $y$ ? Ans: 100.



# The `return` Statement

- When a `return` is encountered, the value of the (optional) expression after the keyword `return` is sent back to the calling function.
- The returning value of a function will be converted, if necessary, to the type of function as specified in the header to the function definition.
- Syntax:

```
return expression;  
return;
```

- ❑ Example:

```
return (a+b*2) ;
```

# Exercise

- Write a function `area`
  - Input: length of the sides of a square.
  - Output: the area of the square.

# Example: findMax

- We can define a function `findMax`, which accepts two integers as input.
- The function **returns** the larger value of the two integers.
- E.g.
  - When  $x > y$ , the expression `findMax(x, y)` should evaluate to a value of  $x$ .

```
cout << findMax(4, 3); //print x (4)
```

- When  $y > x$ , the expression `findMax(x, y)` should evaluate to a value of  $y$ .

```
cout << findMax(3, 4); //print y (4)
```

# Function Definition

```
int findMax (int n1, int n2) {  
    if (n1 > n2)  
        return n1;  
    else  
        return n2;  
}
```

The return type of the variable is **int**

When there are more than one arguments, they are separated by a comma.


The type of each variable should be specified individually.

**Error:**

```
int findMax (int a, b);
```

# Calling findMax ( )

```
int max;  
int x=3;  
int y=4;  
max=findMax(x, y);
```



The value of this expression is 4. Assign 4 to `max`

The variable `max` will hold the value of `x` when  $x > y$ .  
Otherwise, `max` will hold the value of `y`.

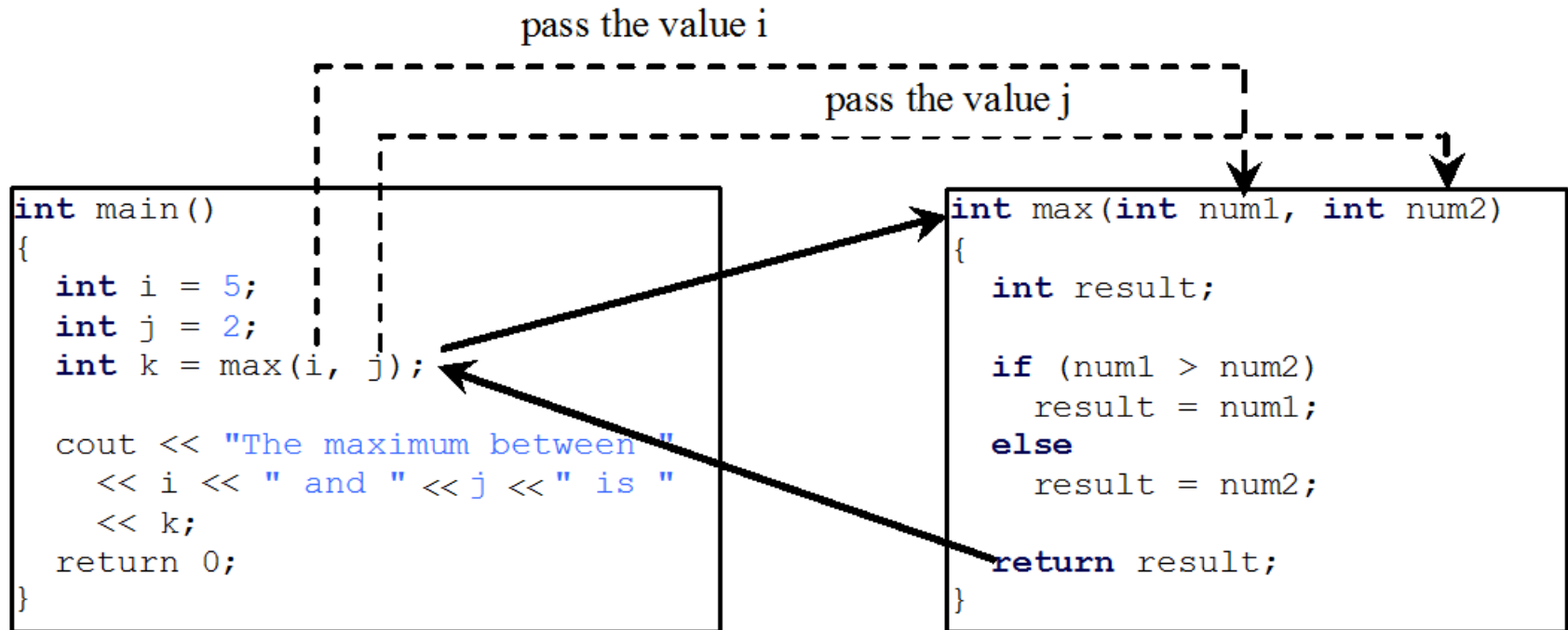
# Flow of Control

```
void main() {  
    int max;  
    int x=3;  
    int y=4;  
    max=findMax(x, y);  
}
```

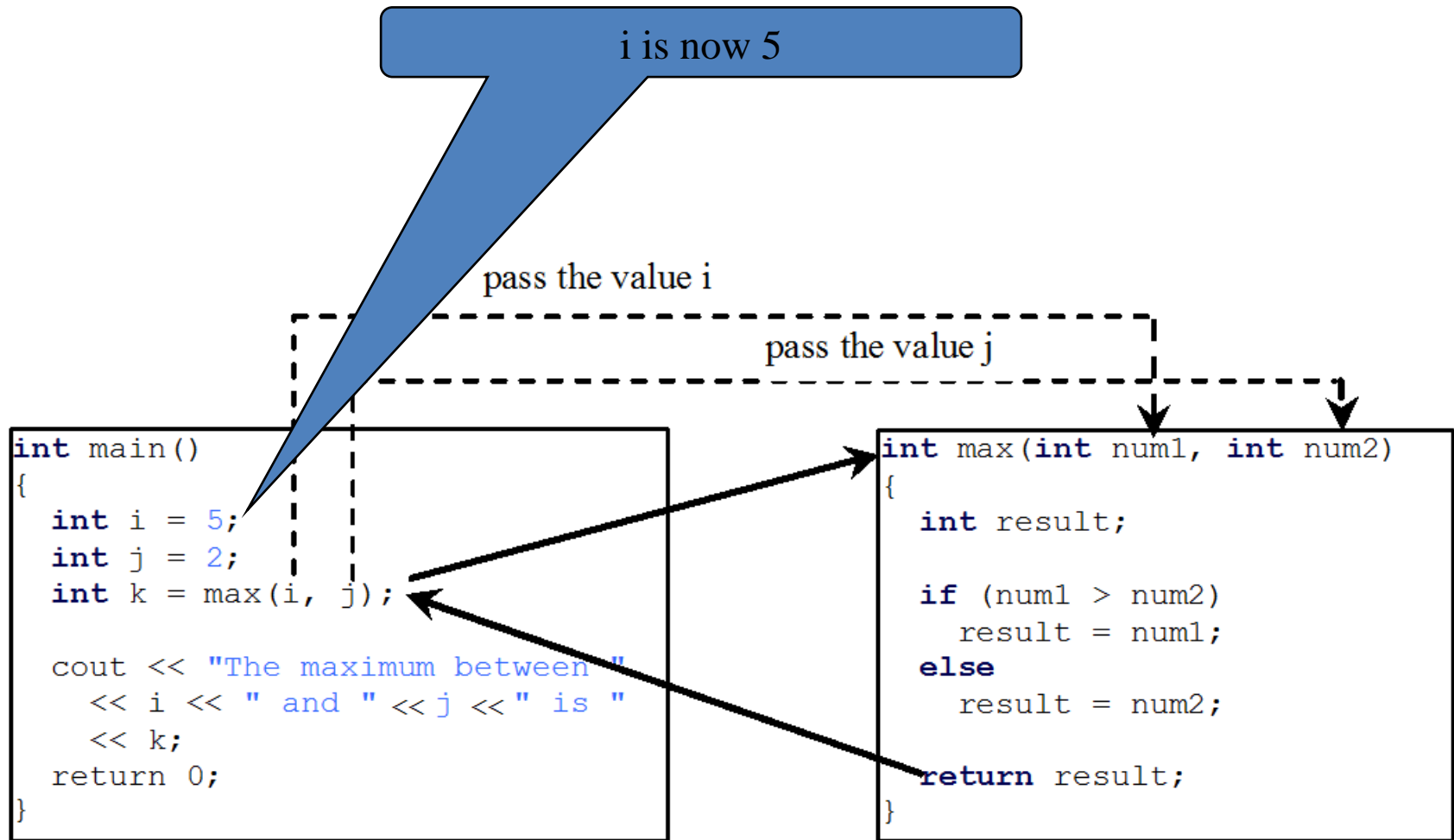
```
int findMax (int n1, int n2) {  
    if (n1>n2)  
        return n1;  
    else  
        return n2;  
}
```

When `findMax()` is called,  
the value of `x` is copied to the variable `n1` .  
the value of `y` is copied to the variable `n2` .

# Calling Functions

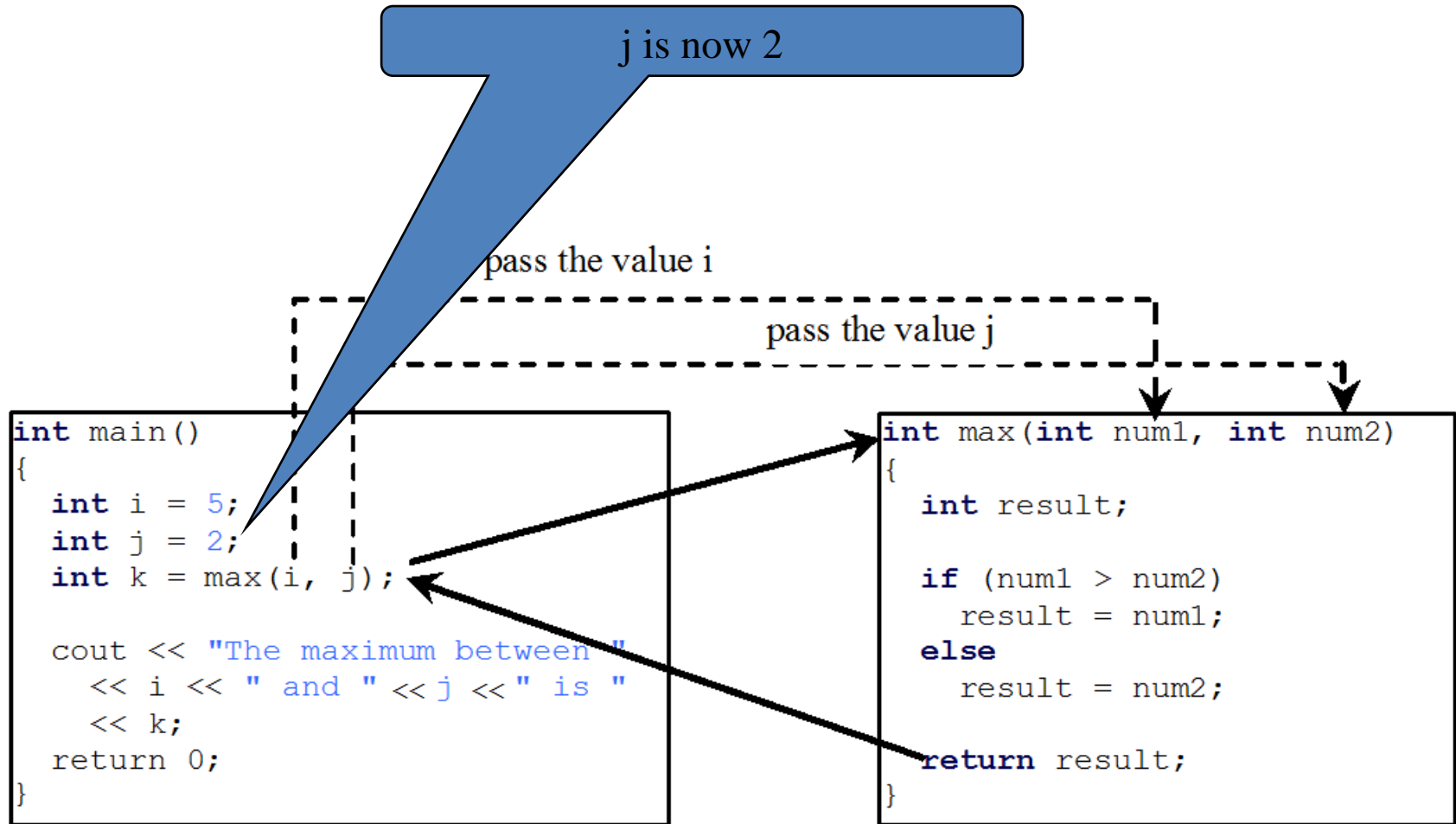


# Calling Functions, cont.

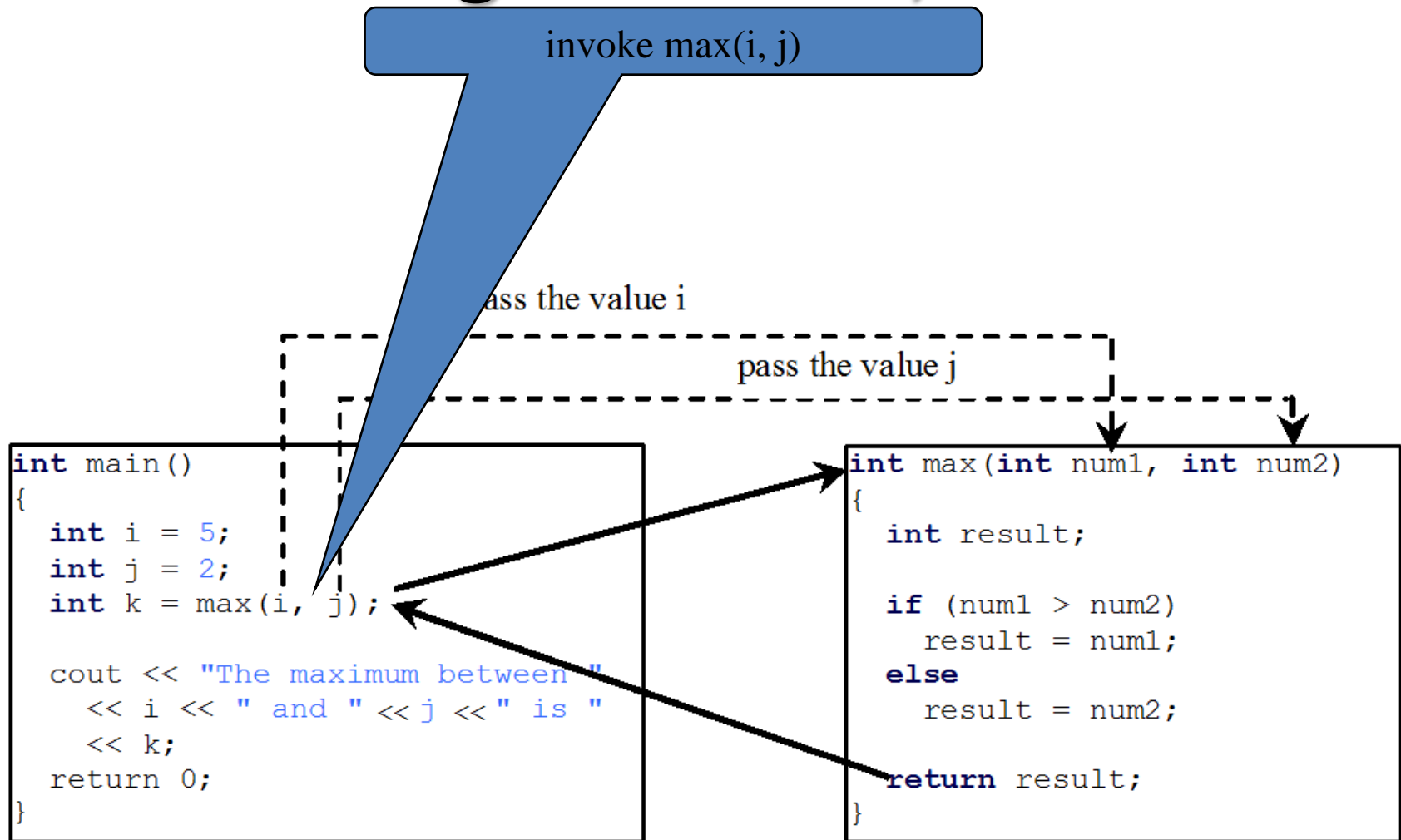




# Calling Functions, cont.

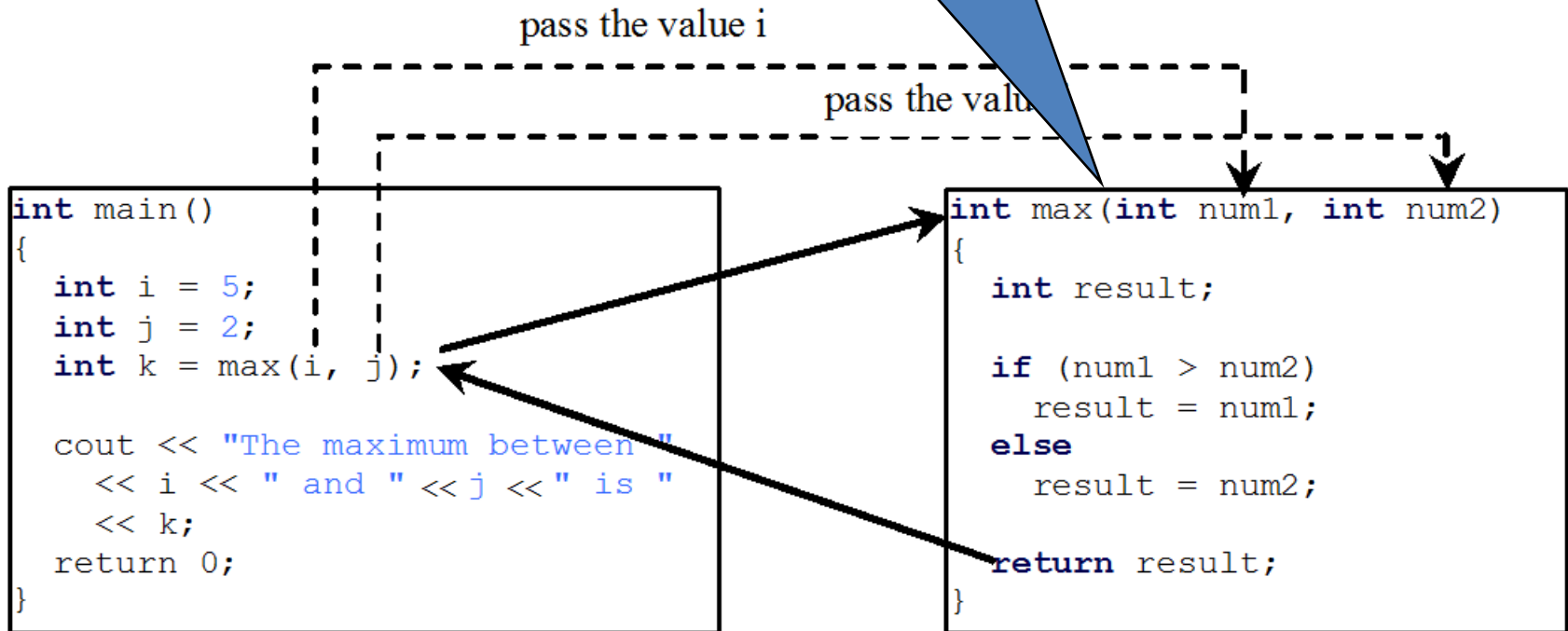


# Calling Functions, cont.



# Calling Functions cont.

invoke max(i, j)  
Pass the value of i to num1  
Pass the value of j to num2



# Calling Functions, cont.

declare variable result

pass the value i

pass the value j

```
int main()
{
    int i = 5;
    int j = 2;
    int k = max(i, j);

    cout << "The maximum between "
          << i << " and " << j << " is "
          << k;
    return 0;
}
```

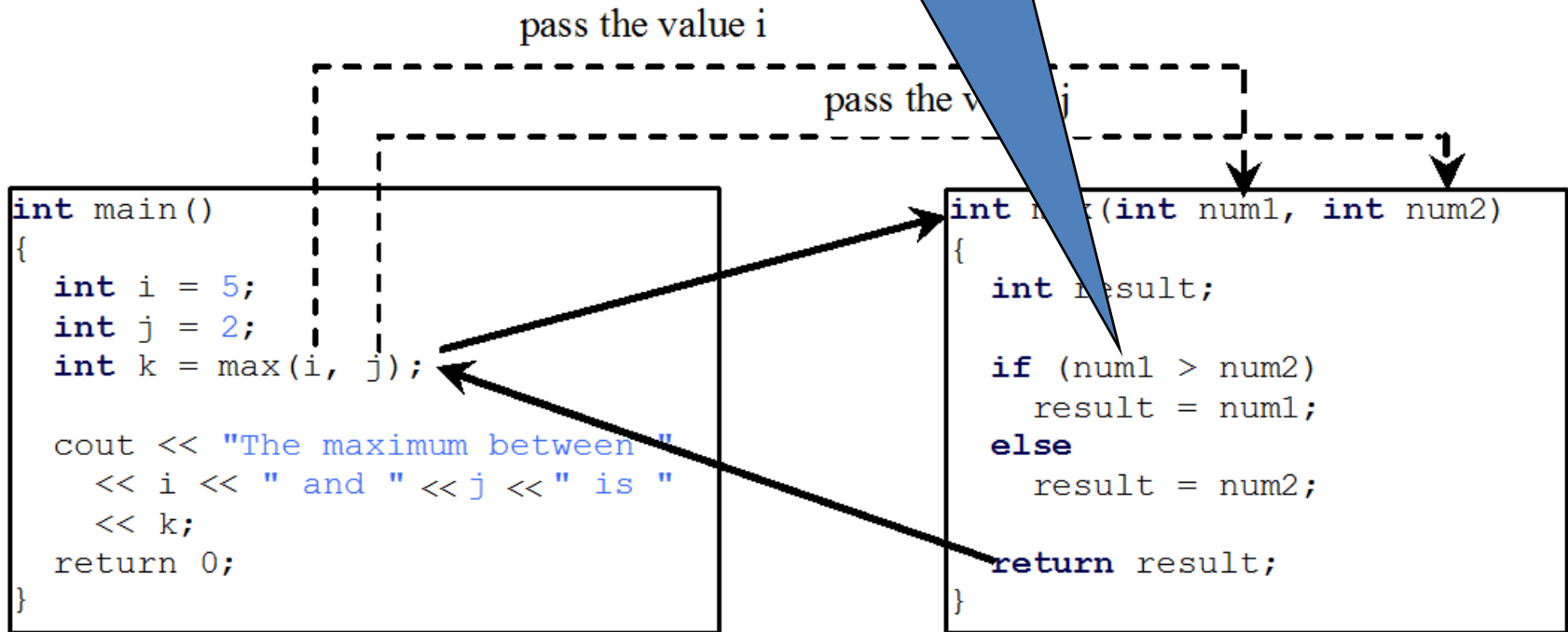
```
int max(int num1, int num2)
{
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

# Calling Functions, cont.

(num1 > num2) is true since num1 is 5 and num2 is 2



# Calling Function Cont.

result is now 5

pass the value i

pass the value j

```
int main()
{
    int i = 5;
    int j = 2;
    int k = max(i, j);

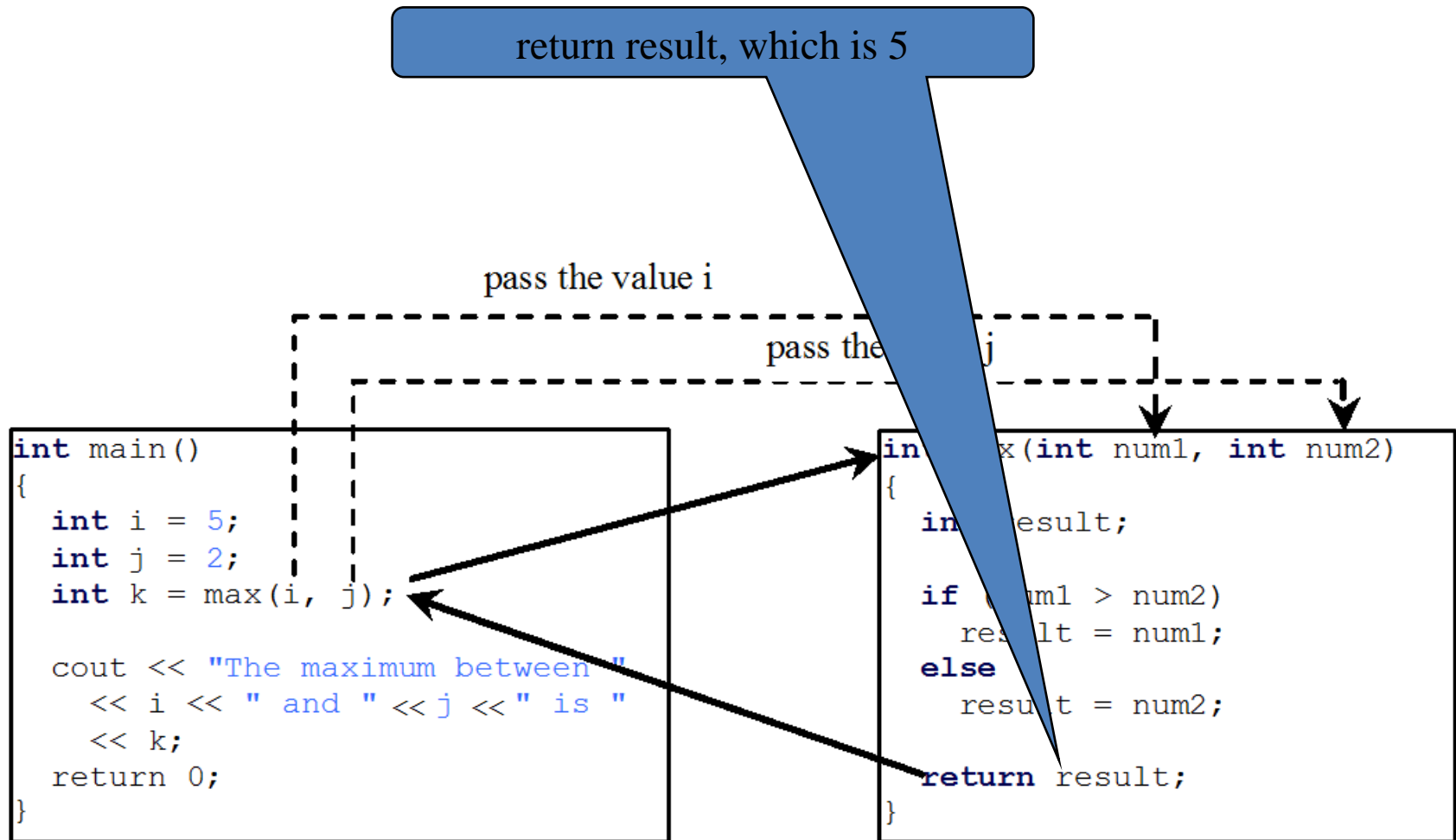
    cout << "The maximum between "
          << i << " and " << j << " is "
          << k;
    return 0;
}
```

```
int max(int num1, int num2)
{
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

# Calling Functions, cont.



# Calling Functions, cont.

return max(i, j) and assign the  
return value to k

the value i

pass the value j

```
int main()
{
    int i = 5;
    int j = 2;
    int k = max(i, j);

    cout << "The maximum between "
          << i << " and " << j << " is "
          << k;
    return 0;
}
```

```
int max(int num1, int num2)
{
    int result;

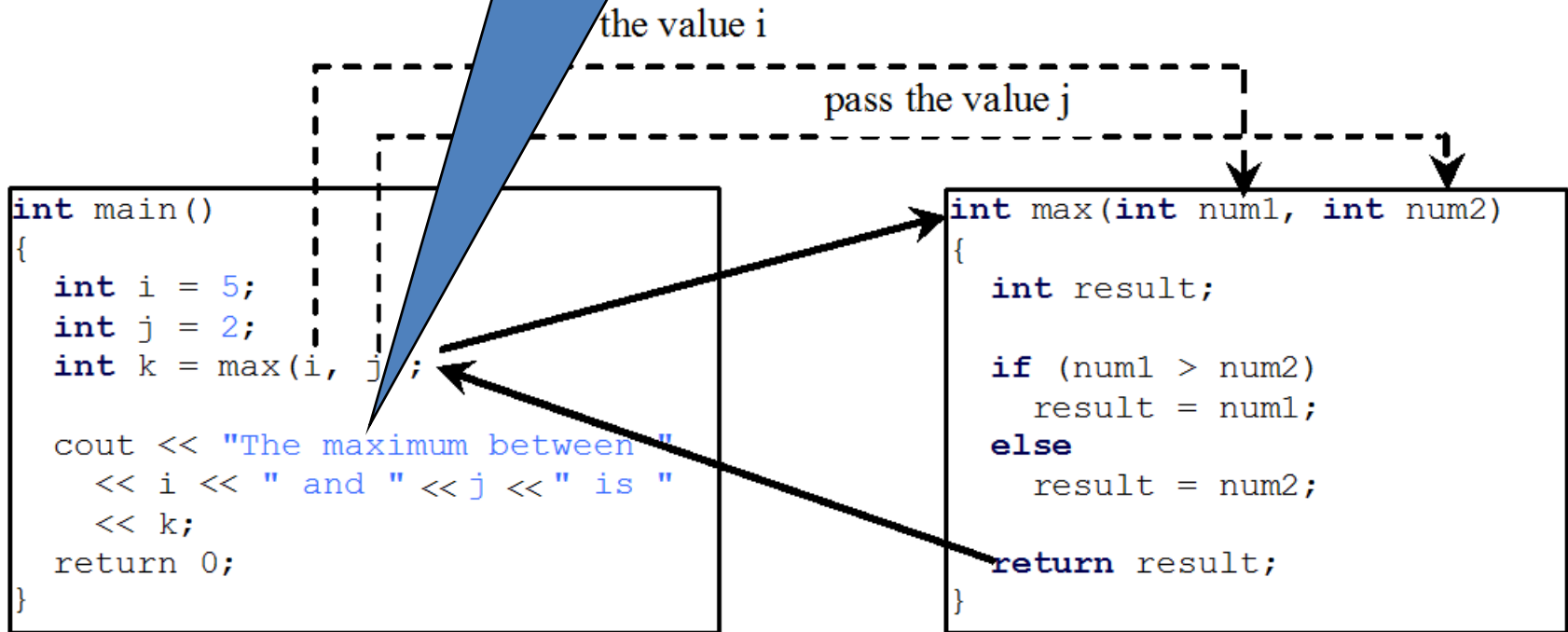
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```



# Calling Functions cont.

Execute the print statement



# Finding the Max of 3 Numbers

```
int i,j,k;  
int max;  
cin >> i >> j >> k;  
  
//find the max of i, j, k  
____ = findMax(____, ____);  
____ = findMax(____, ____);  
  
cout << "max is " << max;
```

# Answer

```
int i,j,k;  
int max;  
cin >> i >> j >> k;  
  
//find the max of i, j, k  
max = findMax(i, j);  
max = findMax(max, k);  
  
cout << "max is " << max;
```

# What is the Output of the Following Program?

```
void f(int y, int x) {  
    cout << "x=" << x << endl;  
    cout << "y=" << y << endl;  
}  
  
void main() {  
    int x=3, y=4;  
    f(x, y);  
}
```

# Answer

```
x=4 ;
```

```
y=3 ;
```

# Parameters Passing: Call-by-Reference

- Parameters pass to a function can be updated inside the function.
- Add '&' in front of the parameter that to be called by reference
- More detail will be given in further Lecture (Pointer).

```
void swap(int &a, int &b) {  
    int temp;  
    temp=a;  
    a=b;  
    b=temp;  
}  
  
void main() {  
    int x=1,y=3;  
    cout << "x:"<<x <<" ,y:"<<y<<endl;  
    swap(x,y);  
    cout << "x:"<<x <<" ,y:"<<y<<endl;  
}
```

# Parameters Passing: Arrays

- When passing an array to a function, we only need to specify the array name.
- The following example is **invalid**:

```
void f(int x[20]){  
    ...  
}  
  
void main(){  
    int y[20];  
    f(y[0]); //invalid, type mismatch  
}
```

# Parameters Passing: Arrays

The size of array is optional.

```
void f(int a[])
```

if the content of a[i] is modified in the function, the modification will persist even after the function returns (Call by reference).

```
void f(int a[3]) {  
    cout << a[0] << endl; //1 is printed  
    a[0]=10;  
}  
  
void main (void) {  
    int a[3]={1,2,5}; //an array with 3 elements  
    f(a); //calling f() with array a  
    cout << a[0] << endl; //10 is printed  
}
```

Only need to input the array name!



# Defining and Calling Functions

## Correct

```
void f() {  
}  
  
void main() {  
    f();  
}
```

## Syntax Error

```
void main() {  
    f(); //f() is undefined  
}  
  
void f() {  
}
```

A function should be defined before use.

# Defining and Calling Functions

- Suppose there are 10 functions `func1`, `func2`, `func3`, ..., `func10`
- `func1` will call `func2`, `func2` will call `func3`, `func5` will call `func6`, `func7` will call `func1`
- In what sequence should the functions be defined?

# Function Prototype

- C++ language allows us to define the function **prototype** and then call the function.
- Function prototype:
  - Specifies the **function name**, **input** and **output type**.
  - The following statement specifies that `f` is a function name, there are no input and no return value:  

```
void f(void);
```
- The function can be defined later.

# Function Prototype

```
void f (void);
```

```
void main() {  
    f();  
}
```

```
void f() {  
    //define f() here  
}
```

```
int findMax (int, int);
```

```
void main() {  
    int x=findMax (3,4);  
}
```

```
int findMax (int n1, int n2) {  
    //define findMax() here  
}
```

# Function Prototype

The prototype

```
int findMax (int, int);
```

specifies that `findMax` is a function name.

The return type is `int`.

There are two arguments and their types are `int`.

Another way to write the prototype is:

```
int findMax (int n1, int n2);
```

However, the variable names are optional.

# Function Prototype

- In C++, function prototype and definition **can** be stored separately.
- Header file (**.h**):
  - With extension .h, .e.g stdio.h, string.h .
  - Contain function prototype only.
  - To be included in the program that will call the function.
- Implementation file (**.cpp**):
  - Contain function implementation (definition).

# Function Prototype

main.cpp

```
#include "mylib.h"
void main(){
    int x,y=2,z=3;

    .....
    x=calMin(y,z);
    .....
}
```

mylib.h

```
int calMin(int,int);
```

mylib.cpp

```
int calMin(int a,int b){
    if (a>b)
        return b;
    else
        return a;
}
```

# Exercise

Rewrite the example `radius.cpp` by defining 2 additional functions.

`areaOfCircle` : Given a radius `r`, return the area of a circle.

`PrintResult` : Given a radius `r` and area `a`, generate the output.



# radius.cpp

```
#include <iostream>
#include <iomanip>
using namespace std;
#define PI 3.141592653589793
void main()
{
    int radius;
    float area;
    cout << "Radius:";
    cin >> radius;
    area = PI * radius * radius;
    cout << fixed << setprecision(2);
    cout << "Area = " << PI << "*" << radius << "*" << radius;
    cout << "=" << area << endl;
}
```

## Sample input/output

```
Radius:5
Area = 3.14 * 5 * 5 = 78.54
```

# Answer

```
#include <iostream>
#include <iomanip>
using namespace std;
#define PI 3.141592653589793
//define the function areaOfCircle

//define the function printResult

void main(){
    int radius;
    float area;
    cout << "Radius:";
    cin >> radius;
}
```

# Answer

```
#include <iostream>
#include <iomanip>
using namespace std;
#define PI 3.141592653589793
float areaOfCircle(int r){ /*r=radius*/
    float result;
    result= PI * r* r;
    return result;
}

void printResult(int r, float a){ /*r=radius; a=area*/
    cout << "Area = " << PI << "*" << r << "*" << r << "=" << a <<endl;
}

void main(){
    int radius;
    float area;
    cout << "Radius:";
    cin >> radius; /*input 5*/
    area=areaOfCircle(radius);
    printResult(radius, area);
}
```

# Function Prototype (cont'd)

- `void` is used if a function takes no arguments.
- Prototypes allow the compiler to check the code more thoroughly.
- Values passed to function are coerced where necessary, e.g., `printDouble (4)` where the integer 4 will be *promoted* as a double data type 4.

```
#include <iostream>
using namespace std;
void printDouble(double d) {
    cout <<fixed;
    cout <<d <<endl;
}
void main()
{
    int x=4;
    printDouble(x);
}
```

# Recursions

- One basic problem solving technique is to break the task into **subtasks**.
- If a subtask is a smaller version of the original task, you can solve the original task using a **recursive** function.
- A recursive function is one that invokes itself, either directly or indirectly.

# Example: Factorial

- The factorial of  $n$  is defined as:

$$0! = 1$$

$$n! = n * (n-1) * \dots * 2 * 1 \quad \text{for } n > 0$$

- A recurrence relation:

$$n! = n * (n-1)! \quad \text{for } n > 0$$

- E.g.:

$$\begin{aligned} 3! &= 3 * 2! \\ &= 3 * 2 * 1! \\ &= 3 * 2 * 1 * 0! \\ &= 3 * 2 * 1 * 1 \end{aligned}$$

# Iterative vs. Recursive

## Iterative

```
int factorial(int n)
{
    int i, fact = 1;

    for (i = 1; i <= n; i++)
    {
        fact = i * fact;
    }

    return fact;
}
```

## Recursive

```
int factorial(int n)
{
    if (n==0) return 1;
    return n*factorial(n-1);
}
```

# Example: Vertical Number

- Input: one (nonnegative) integer.
- Output: integer with one digital per line.

Input	Output
12345	1 2 3 4 5
7894	7 8 9 4
4	4



# Example: Vertical Number

- How to break down a number into separated digits?

```
void printDigit(int n)
{
    do{
        cout << n%10 <<endl;
        n/=10;
    }while (n>0);
}
```

Input	Output
7894	4 9 8 7

# Example: Vertical Number

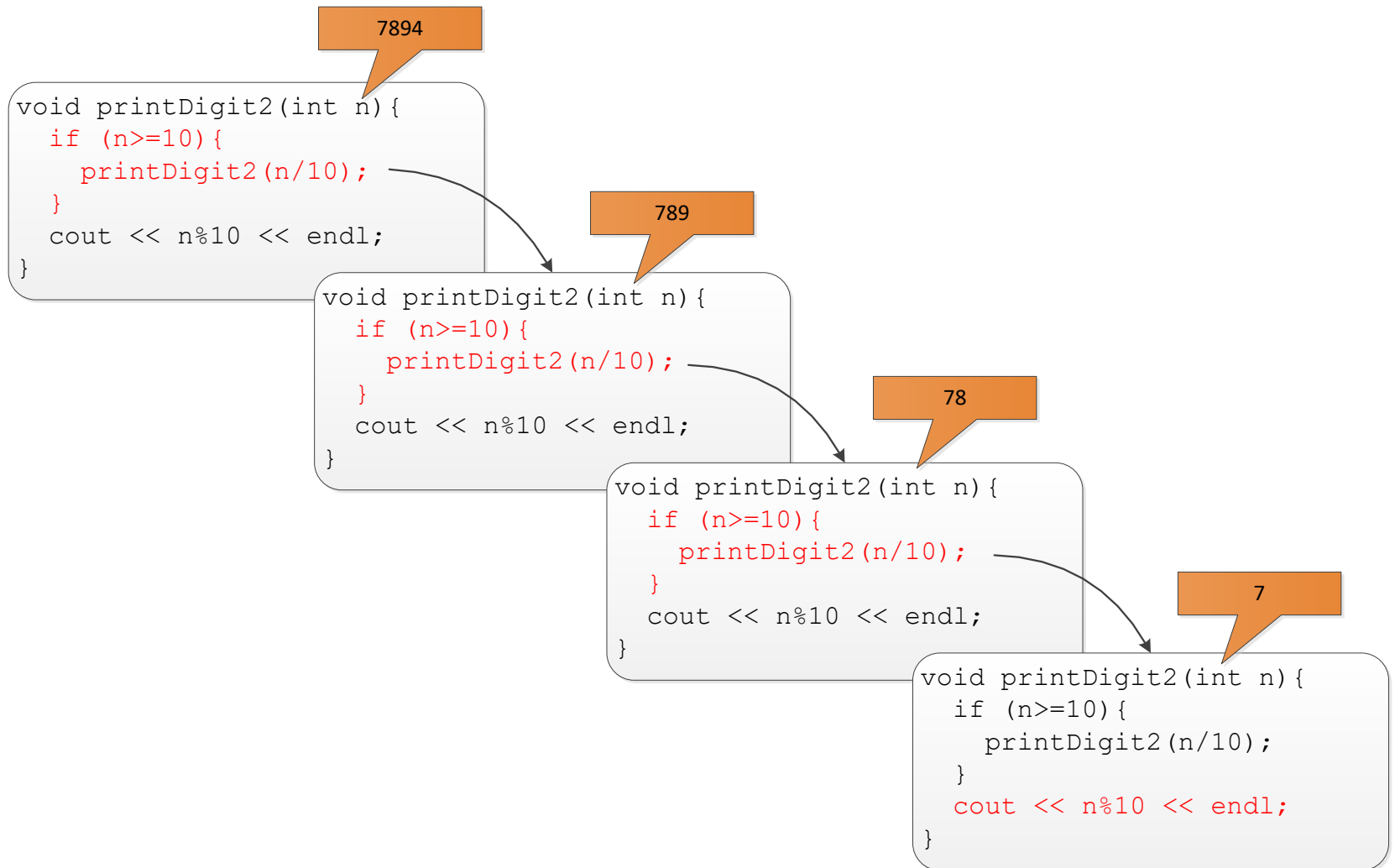
- How to print the digits in reverse order?

```
void printDigit(int n)
{
    do{
        cout << n%10 <<endl;
        n/=10;
    }while (n>0);
}
```

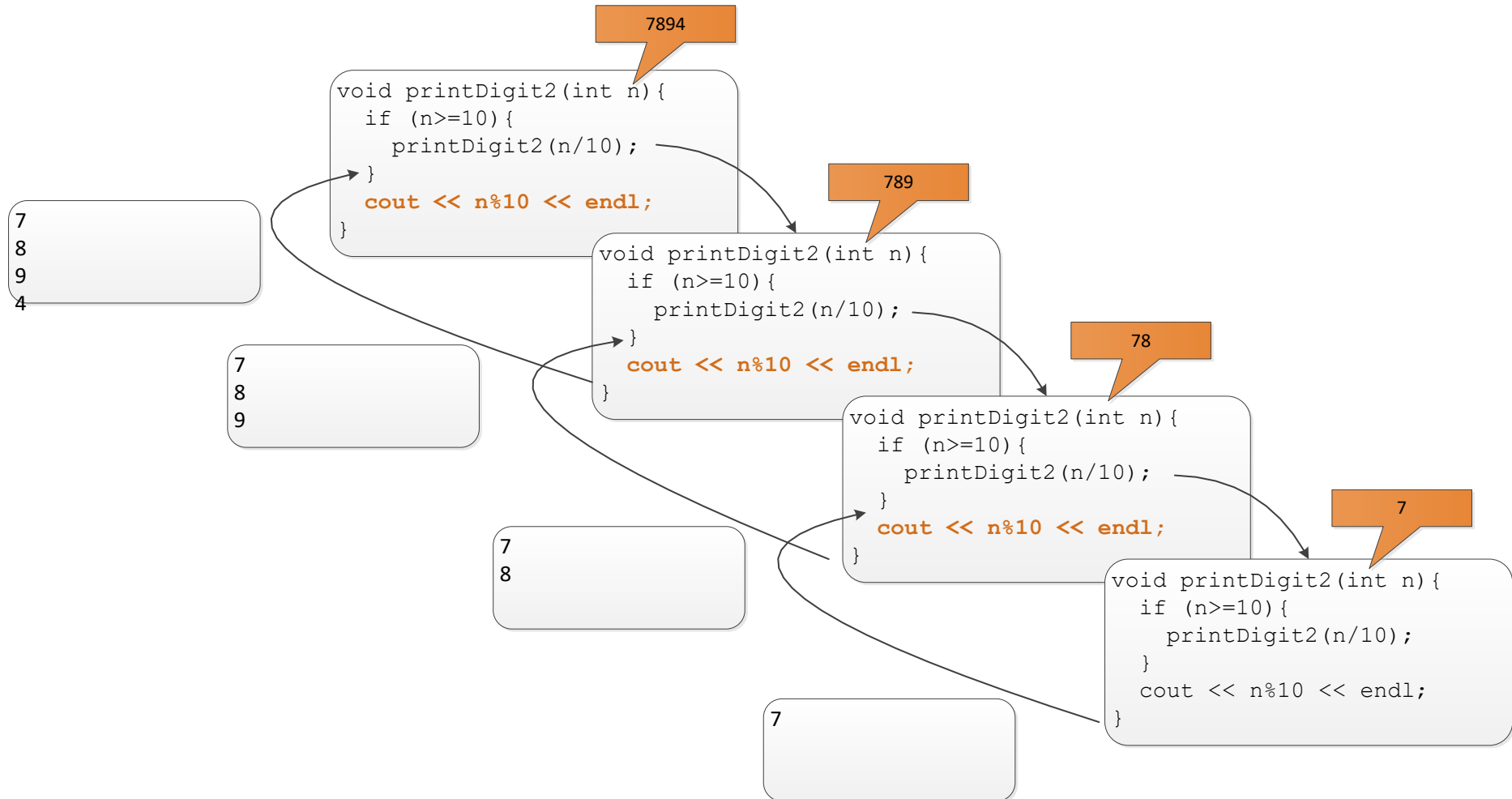
```
void printDigit2(int n){
    if (n>=10){
        printDigit2(n/10);
    }
    cout << n%10 << endl;
}
```

Input	Output
7894	7 8 9 4

# Recursive : Entering



# Recursive: Leaving



# Guidelines for Writing Recursive Functions

- Identify the parameters.
  - e.g.  $n$  in the factorial problem.
- Find out a recurrence relation between the current problem and smaller versions (in terms of smaller parameters) of the current problem.
  - e.g.  $\text{factorial}(n) = n * \text{factorial}(n-1)$ .
- Find out the base cases and their solutions.
  - e.g.  $\text{factorial}(0) = 1$ .
  - Omitting the base case is one of the common mistakes in writing recursive functions.

# Checkpoints

1. There is no infinite recursion (check exist condition).
2. Each stopping case performs the correct action for that case.
3. For each of cases that involve recursion, if all recursive calls perform their actions correctly, then the entire case performs correctly.

# Summary

- Functions help programmer write a more simple program and make the problem more easy to solve.
- `return_type Function_name(paramaters)`
- Function prototype must be declared before it can be used.
- Header file is used to store function prototype but not the body.
- Parameters can be call by value or call by reference.