

# Chapter 6 Functions



# Opening Problem

Find the sum of integers from 1 to 10, from 20 to 37, and from 35 to 49, respectively.



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

```
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;
}
```



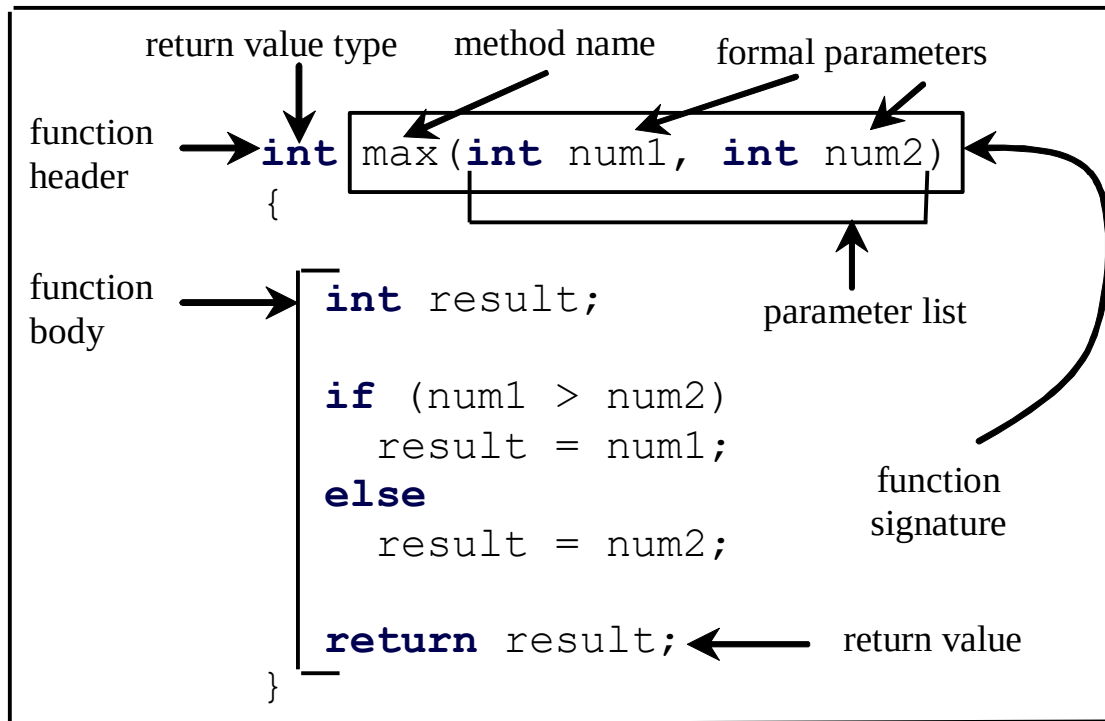
# Objectives

- ❑ To define functions with formal parameters (§6.2).
- ❑ To define/invoke value-returning functions (§6.3).
- ❑ To define/invoke void functions (§6.4).
- ❑ To pass arguments by value (§6.5).
- ❑ To develop reusable code that is modular, easy to read, easy to debug, and easy to maintain (§6.6).
- ❑ To use function overloading and understand ambiguous overloading (§6.7).
- ❑ To use function prototypes to declare function headers (§6.8).
- ❑ To define functions with default arguments (§6.9).
- ❑ To improve runtime efficiency for short functions using inline functions (§6.10).
- ❑ To determine the scope of local and global variables (§6.11).
- ❑ To pass arguments by reference and understand the differences between pass-by-value and pass-by-reference (§6.12).
- ❑ To declare const parameters to prevent them from being modified accidentally (§6.13).
- ❑ To write a function that converts a hexadecimal number to a decimal number (§6.14).
- ❑ To design and implement functions using stepwise refinement (§6.15).

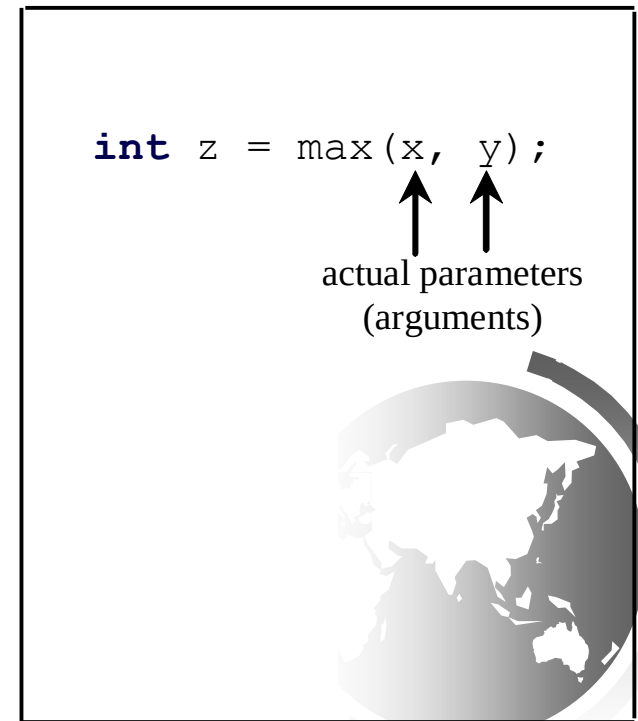
# Defining a Function

A function is a collection of statements that are grouped together to perform an operation.

Define a function



Invoke a function



# Defining Functions, cont.

- *Function signature* is the combination of the function name and the parameter list.
- The variables defined in the function header are known as *formal parameters*.
- When a function is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.



# Defining Functions, cont.

- A Function may return a value. The returnValueType is the data type of the value the function returns. If the function does not return a value, the returnValueType is the keyword void.



# Calling a Function

## Listing 6.1 Testing the `max` Function

This program demonstrates calling a Function `max` to return the largest of the `int` values

TestMax

TestMax



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

// Return the max between two numbers
int max(int num1, int num2)
{
    int result;

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

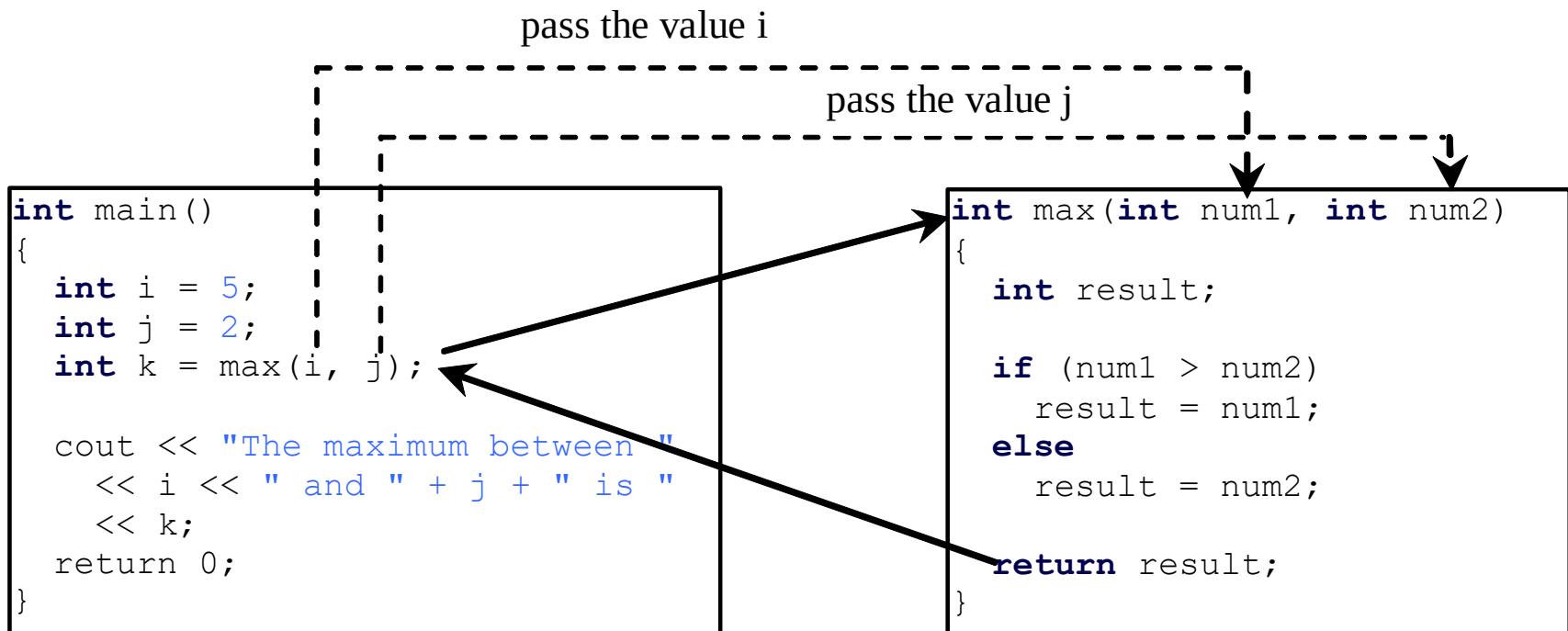
    return result;
}

int main()
{
    int i = 5;
    int j = 2;
    int k = max(i, j);
    cout << "The maximum between " << i <<
        " and " << j << " is " << k << endl;

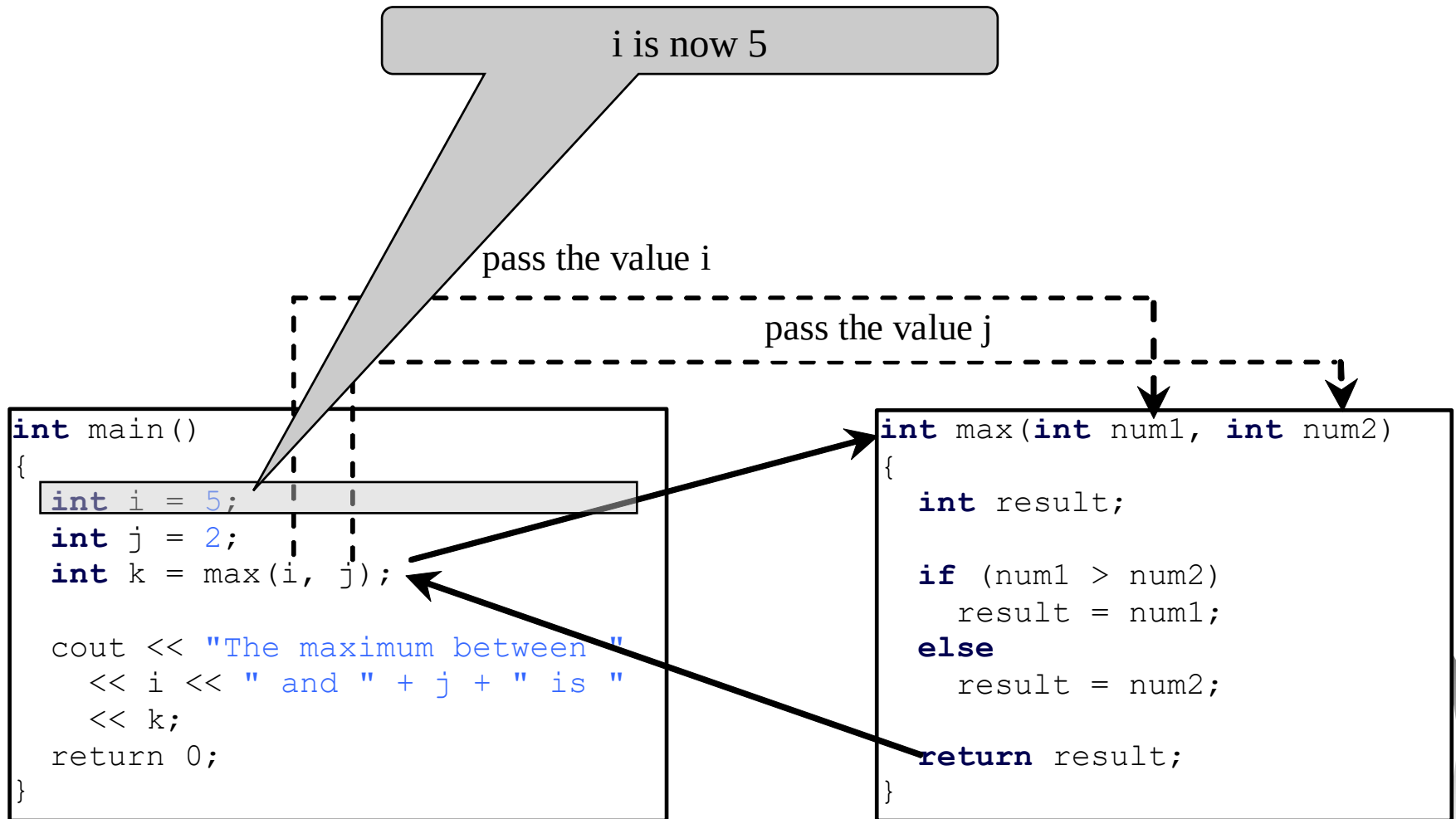
    return 0;
}
```



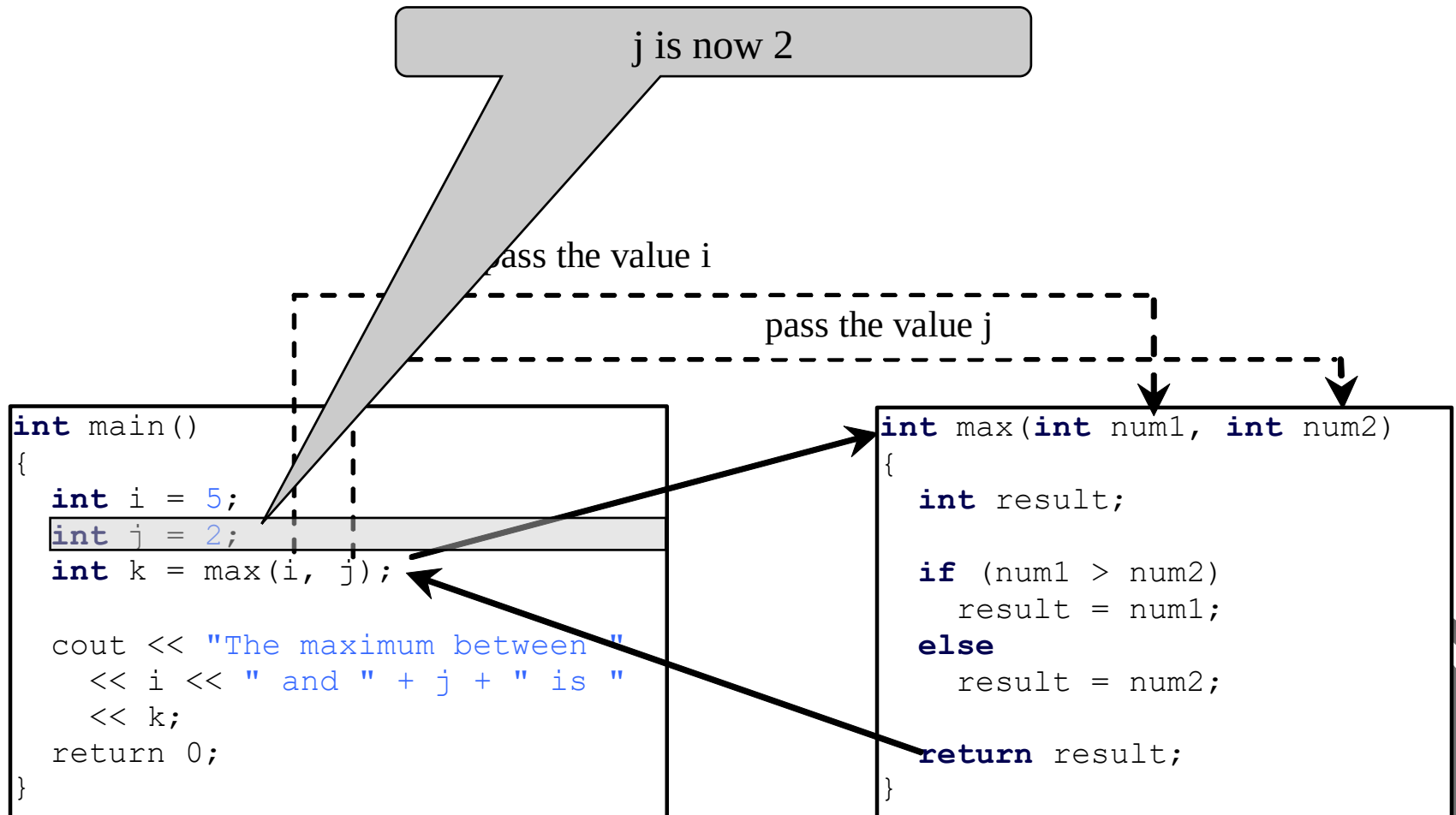
# Calling Functions, cont.



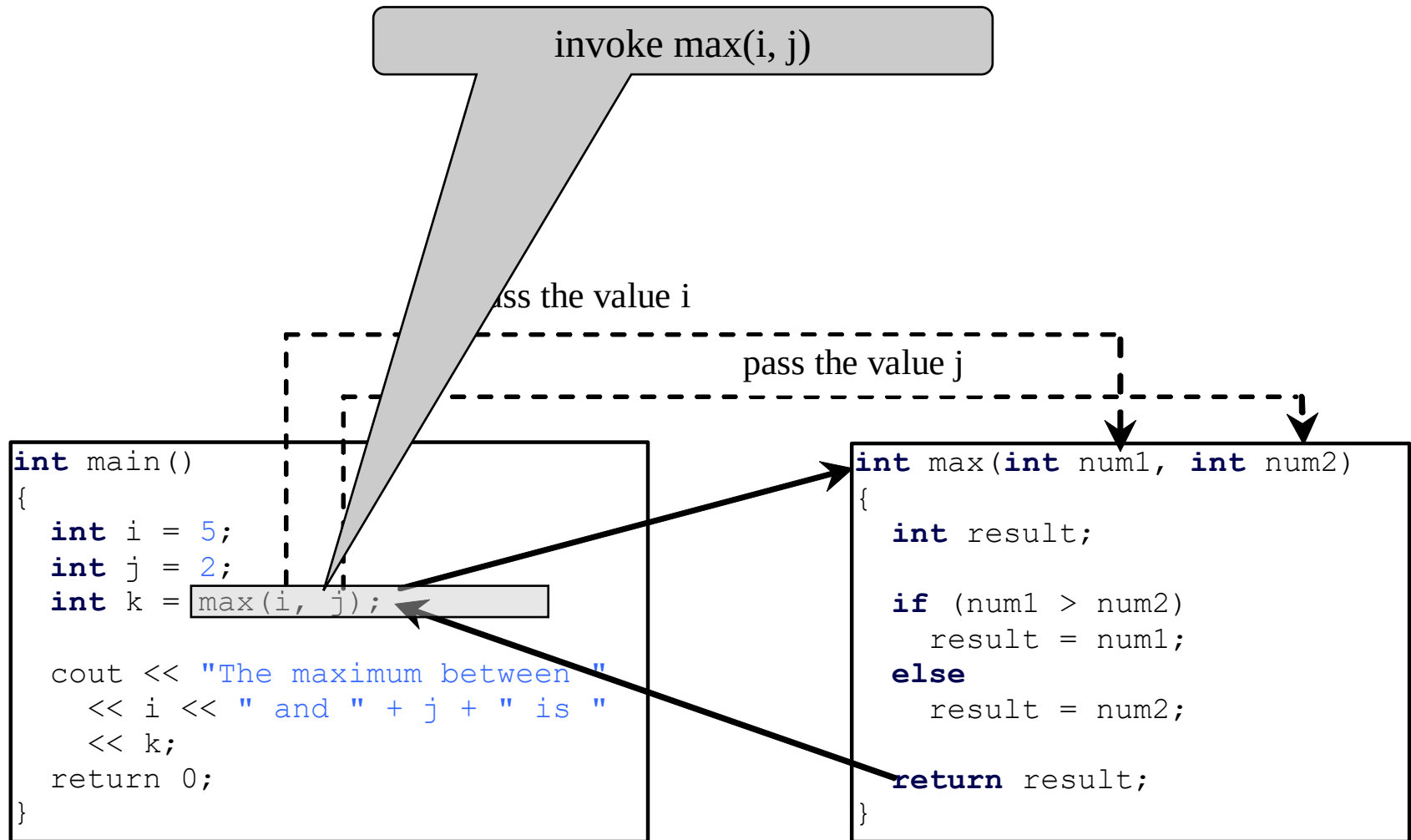
# Trace Function Invocation



# Trace Function Invocation



# Trace Function Invocation



# Trace Function Invocation

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

pass the value i

pass the value j

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

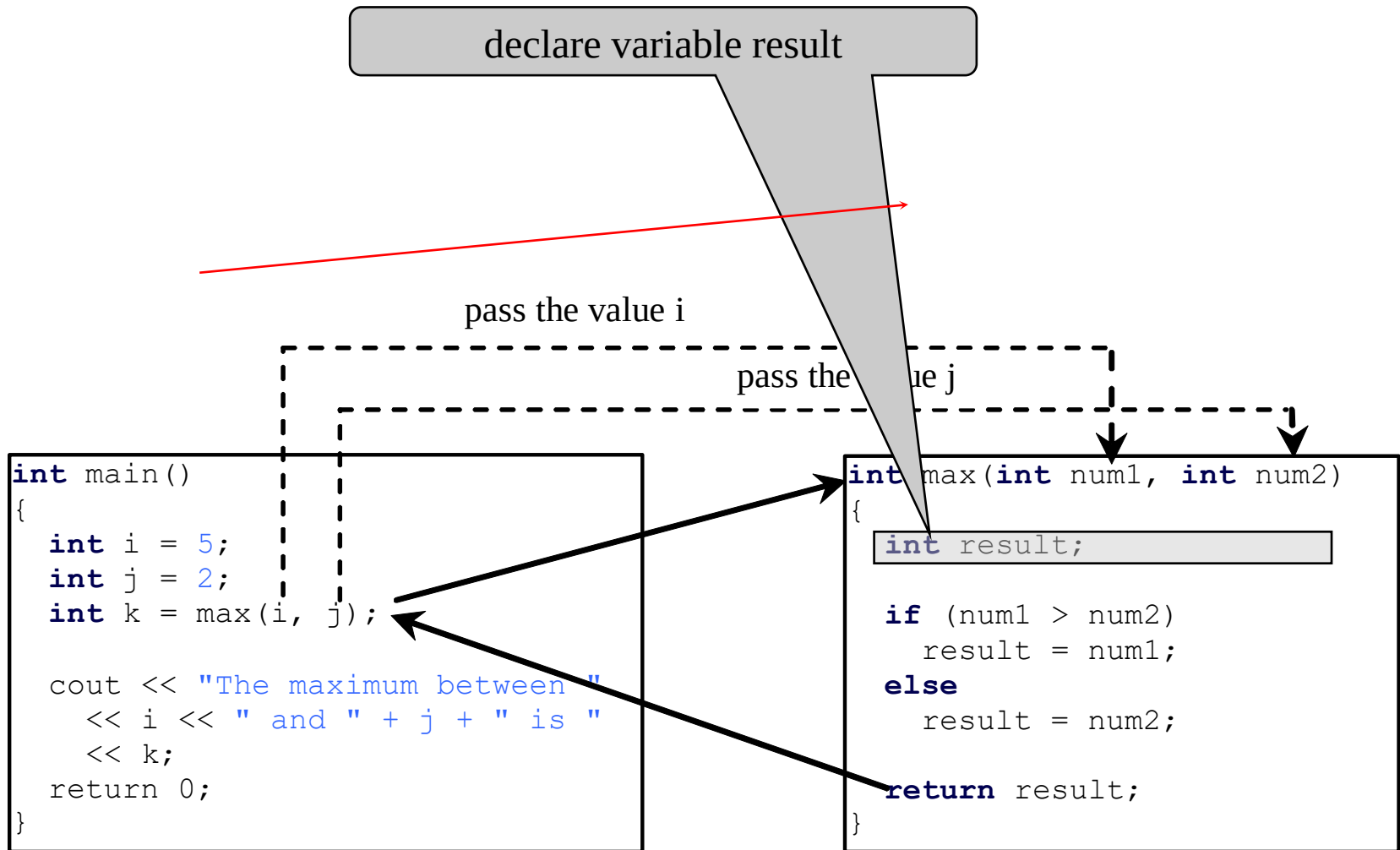
    cout << "The maximum between "
          << i << " and " + j + " is "
          << k;
    return 0;
}
```

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

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

    return result;
}
```

# Trace Function Invocation



# Trace Function Invocation

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

pass the value i

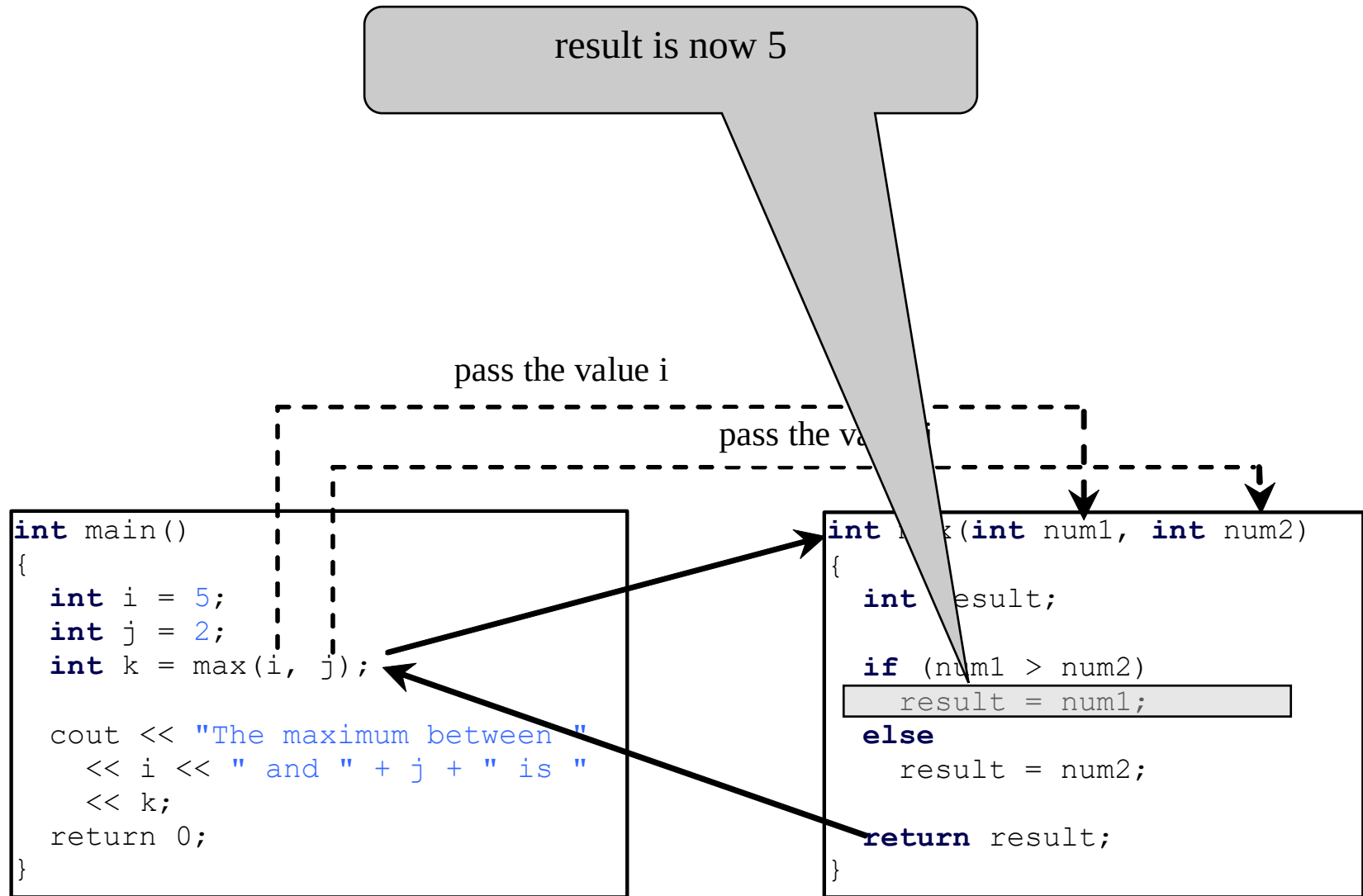
pass the value j

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

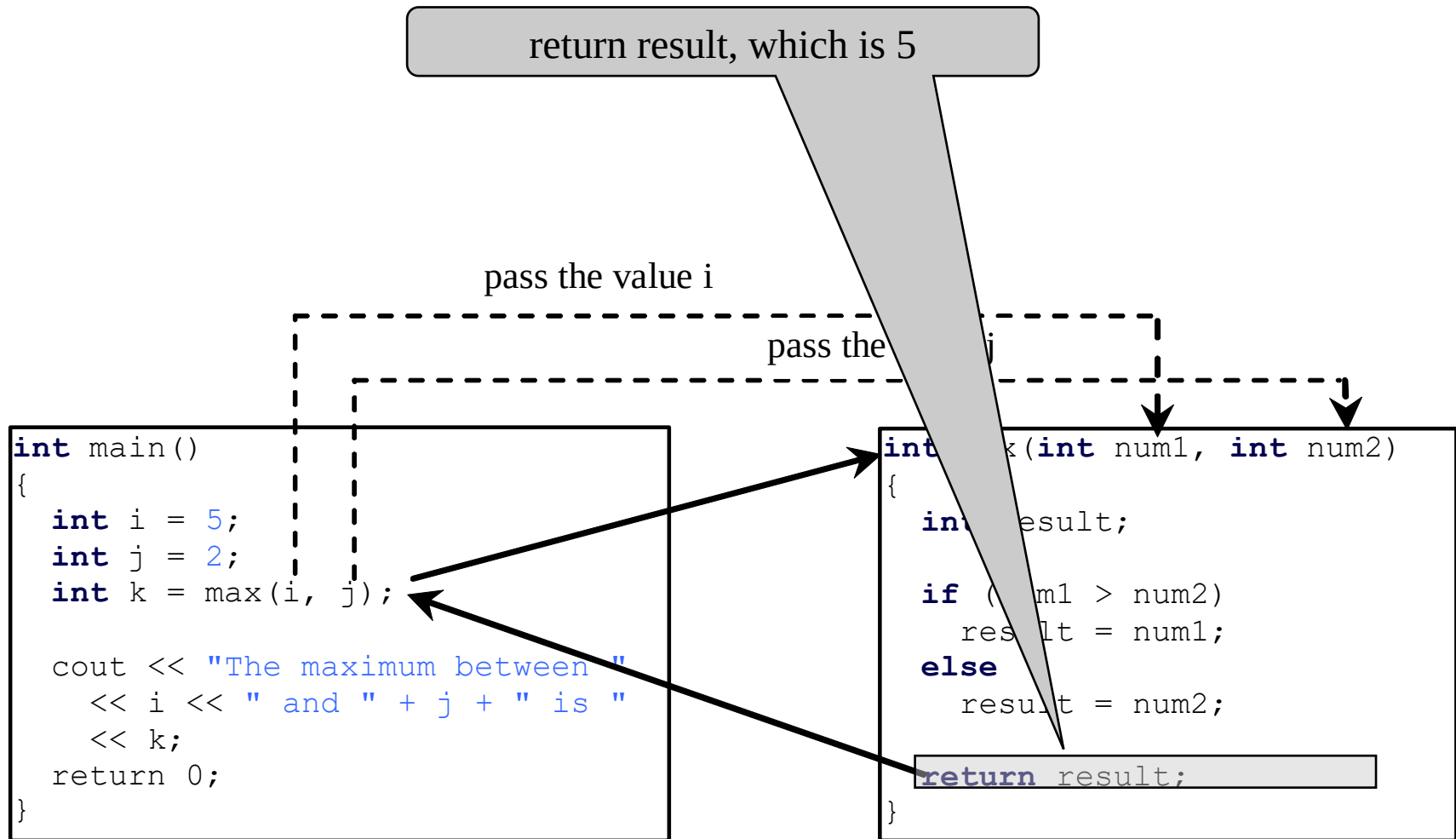
    cout << "The maximum between "
          << i << " and " + j + " is "
          << k;
    return 0;
}
```

```
int max(int num1, int num2)
{
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

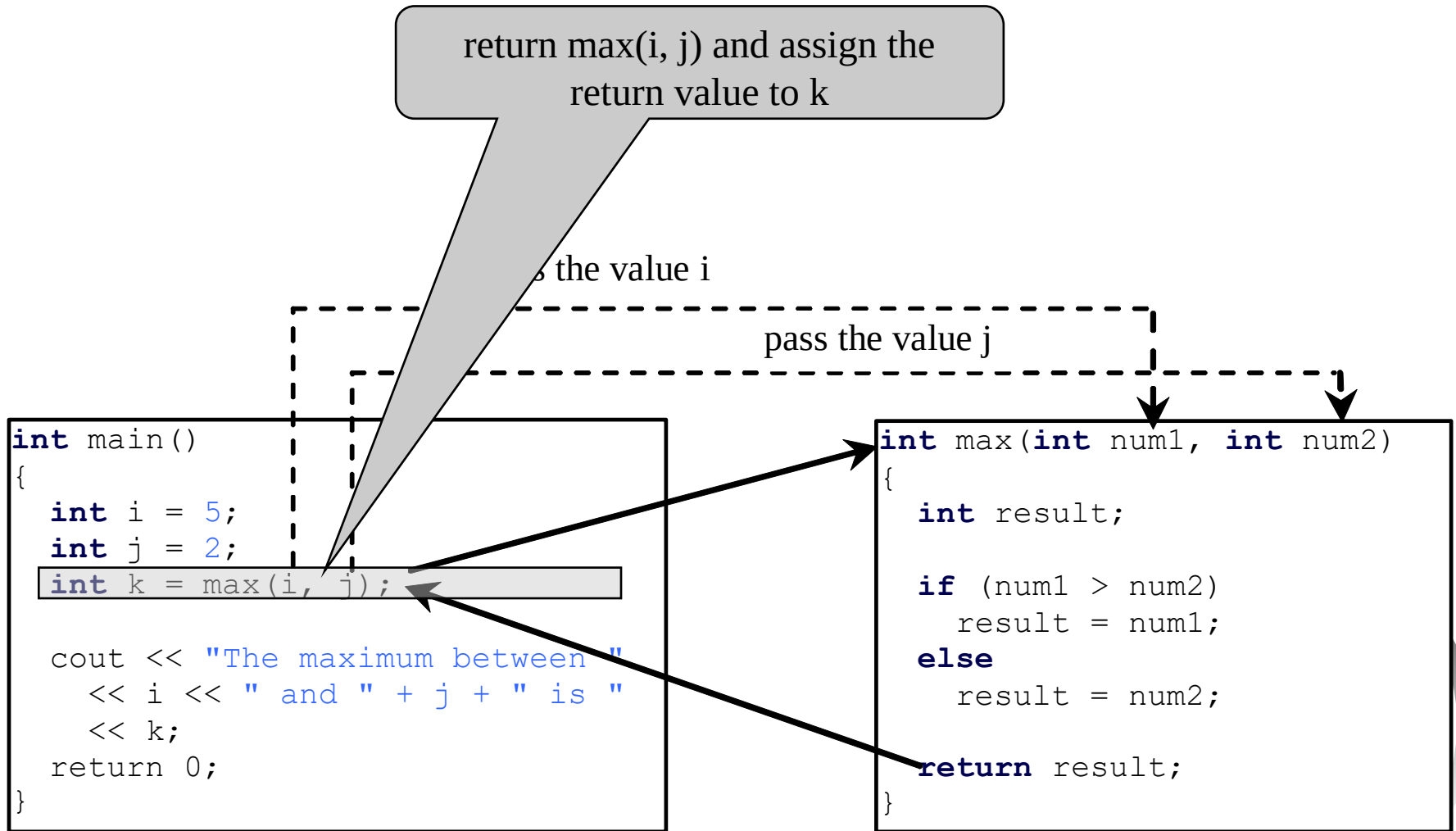
# Trace Function Invocation



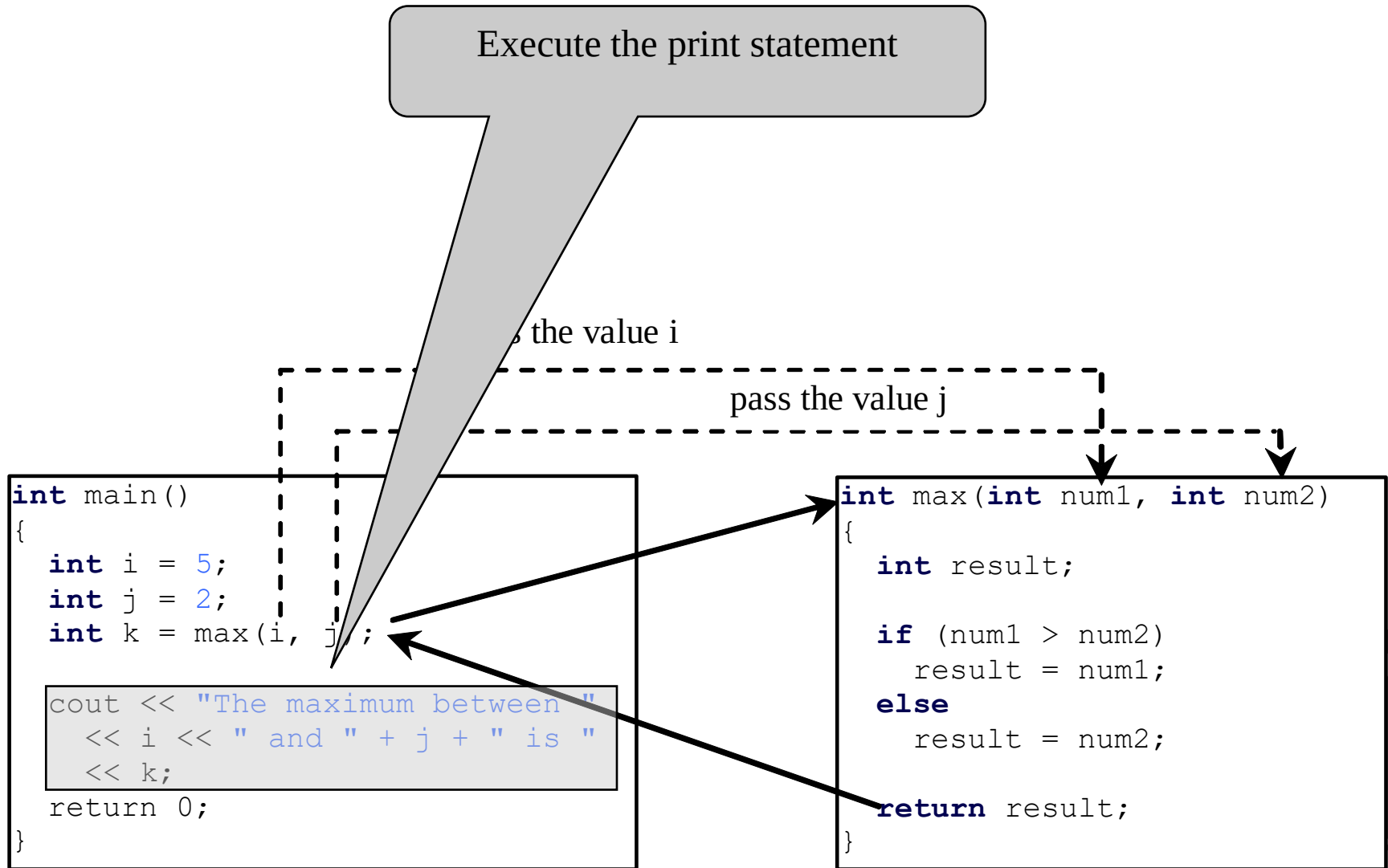
# Trace Function Invocation



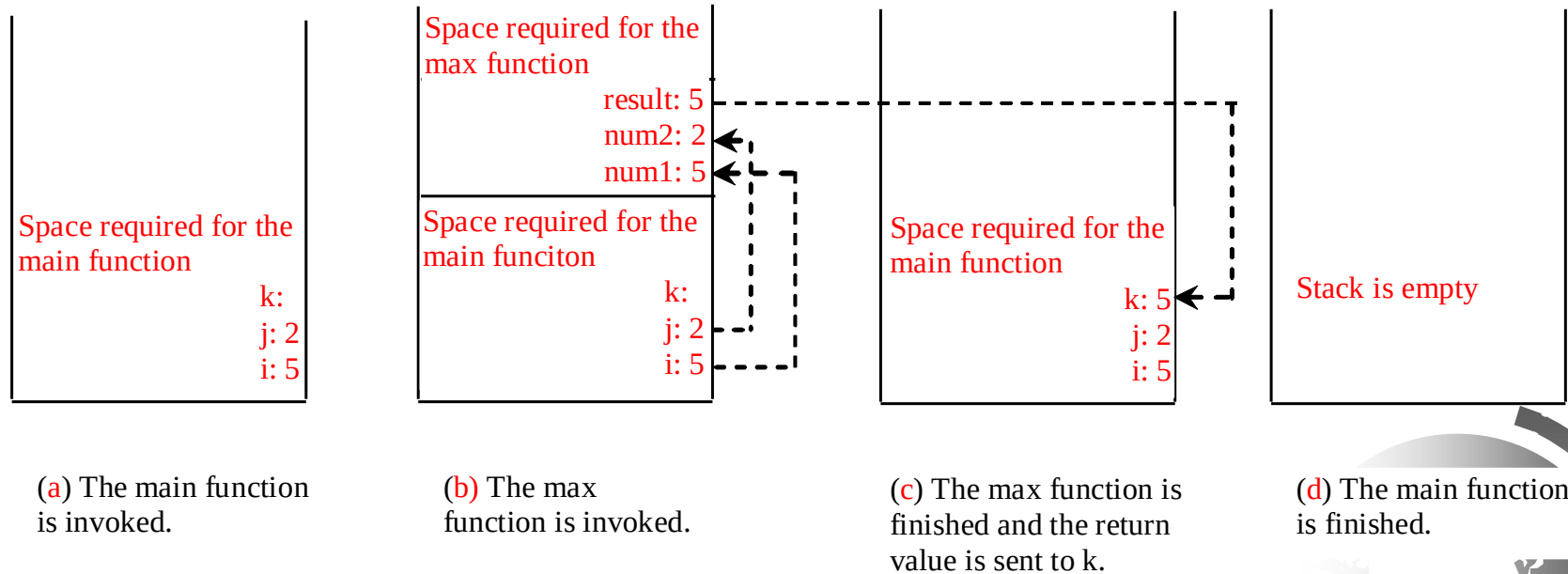
# Trace Function Invocation



# Trace Function Invocation



# Call Stacks



# Trace Call Stack

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

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

i is declared and initialized

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

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

    return result;
}
```

i: 5

The main method  
is invoked.

# Trace Call Stack

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

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

j is declared and initialized

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

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

    return result;
}
```

j: 2  
i: 5

The main method  
is invoked.

# Trace Call Stack

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

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

Declare k

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

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

    return result;
}
```

Space required for the  
main method

k:  
j: 2  
i: 5

The main method  
is invoked.

# Trace Call Stack

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

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

Invoke max(i, j)

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

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

    return result;
}
```

Space required for the  
main method

k:  
j: 2  
i: 5

The main method  
is invoked.

# Trace Call Stack

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

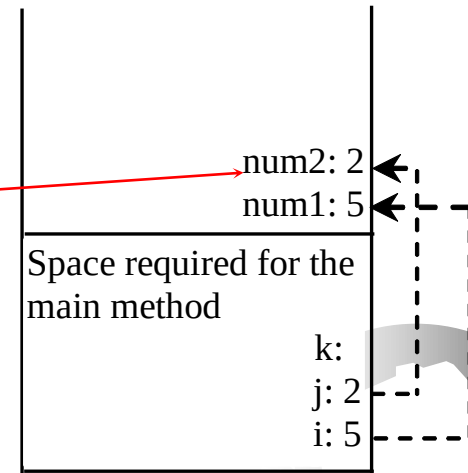
    cout << "The maximum between "
          << i << " and " + j + " is "
          << k;
    return 0;
}
```

pass the values of i and j to num1  
and num2

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

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

    return result;
}
```



The max method is  
invoked.

# Trace Call Stack

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

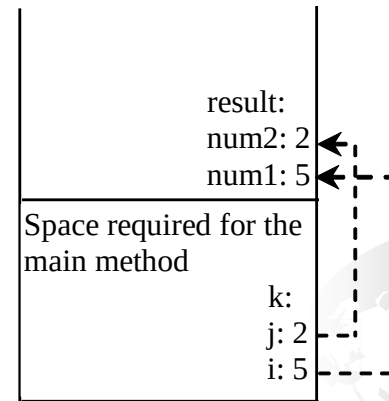
    cout << "The maximum between "
          << i << " and " + j + " is "
          << k;
    return 0;
}
```

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

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

    return result;
}
```

(num1 > num2) is true



The max method is invoked.

# Trace Call Stack

```
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;
}
```

Assign num1 to result

Space required for the  
max method

result: 5  
num2: 2  
num1: 5

Space required for the  
main method

k:  
j: 2  
i: 5

The max method is  
invoked.

# Trace Call Stack

```
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;
}
```

Return result and assign it to k

Space required for the  
max method

result: 5  
num2: 2  
num1: 5

Space required for the  
main method

k:5  
j: 2  
i: 5

The max method is  
invoked.

# Trace Call Stack

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

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

Execute print statement

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

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

    return result;
}
```

Space required for the  
main method

k:5  
j: 2  
i: 5

The main method  
is invoked.