Computer Programming
Lecture 9

Hung-Yu Wei
Department of Electrical Engineering
National Taiwan University

Chapter 7: Pointers

- Basic pointers
  - Declaration
  - Initialization
  - Operations
- Pointers and arrays
- Pointers and strings
- Pointers and functions
  - Pass-by-reference

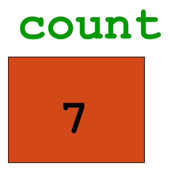
- Function pointer
- Example: Selection sort

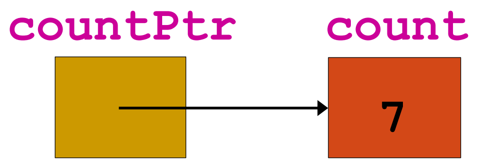
Pointers

- Powerful (but difficult to master) C/C++ feature
- Create pass-by-reference
- Close relationship with arrays and strings
  - C++ *string* class objects
  - C char pointer as string
- Used to create many data structures
  - Linked lists
  - Queues
  - Stacks
  - Trees

Pointer Variable

- Pointer variables
  - Contain memory addresses as values
  - Normally, variable contains specific value (direct reference)
  - Pointers contain address of variable that has specific value (indirect reference)
- Indirection
  - Referencing value through pointer





Pointer Declaration

- Pointer declarations
  - *
    - Known as "indirection operator" or "dereferencing operator"
    - indicates variable is pointer

      ```
      int *myPtr;
      ```
    - declares pointer to **int**
      - pointer of type **int ***
  - Multiple pointers require multiple *

    ```
    int *myPtr1, *myPtr2;
    ```

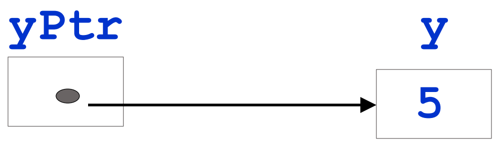
Pointer Variable Declarations and Initialization

- Can declare pointers to any data type
- Pointer initialization
  - Initialized to **`0`**, **`NULL`**, or address
    - **`0`** or **`NULL`** points to nothing

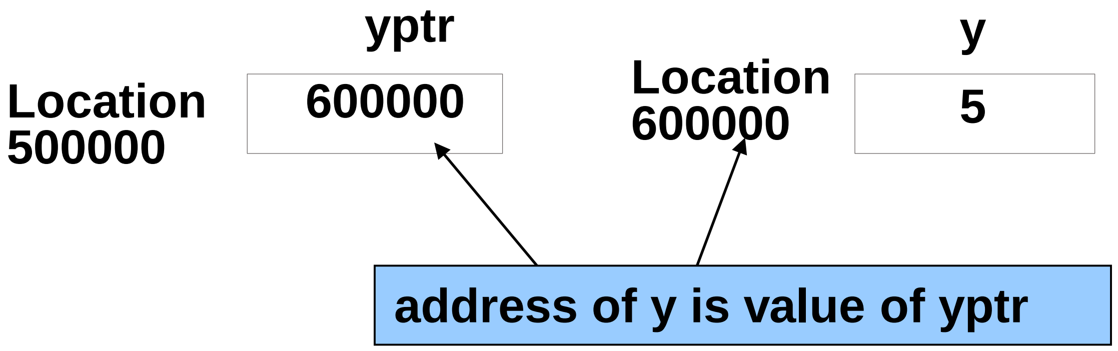
7.3 Pointer Operators

- & (address operator)
  - Returns memory address of its operand
  - Example

```
int y = 5;
int *yPtr;
yPtr = &y;     // yPtr gets address of y
```



  - yPtr "points to" y



Pointer Operators

- `*` (indirection/dereferencing operator)
  - Returns synonym for object its pointer operand points to
  - `*yPtr` returns `y` (because `yPtr` points to `y`).

```
*yptr = 9;      // assigns 9 to y
```

- `*` and `&` are inverses of each other

- Declare

  ```
  int *   MyPointer;
  ```

- *

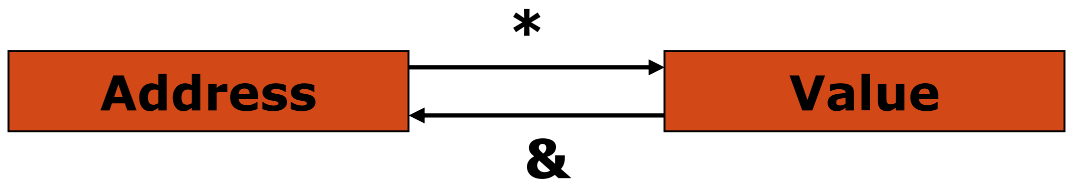
  ```
  * MyPointer
  ```

- &

  ```
  & MyVariable
  ```

Example

- Show how “*” and “&” works
- They are the same
  - &*myPtr
  - *&myPtr
  - Remember “*” and “&” are inverse operations



```cpp
// Fig. 7.4: fig07_04.cpp
// Pointer operators & and *.
#include <iostream>
using namespace std;

int main()
{
   int a; // a is an integer
   int *aPtr; // aPtr is an int * which is a pointer to an integer

   a = 7; // assigned 7 to a
   aPtr = &a; // assign the address of a to aPtr

   cout << "The address of a is " << &a
      << "\nThe value of aPtr is " << aPtr;
   cout << "\n\nThe value of a is " << a
      << "\nThe value of *aPtr is " << *aPtr;
   cout << "\n\nShowing that * and & are inverses of "
      << "each other.\n&*aPtr = " << &*aPtr
      << "\n*&aPtr = " << *&aPtr << endl;
} // end main
```

Fig. 7.4 | Pointer operators & and *. (Part 1 of 2.)

7.4 Pass-By-Reference with Pointer

- Three methods to pass arguments to functions
  - Pass-by-value
    - The typical one
    - Use this unless the function need to directly modify the argument variables
      - Principle of the least privilege
  - Pass-by-reference with reference arguments
    - Chapter 5
  - Pass-by-reference with pointers
    - This Chapter
- Examples in the next 3 pages
  - A function that takes X and computes 2X
  - Compare these 3 cases carefully

Pass-by-value

- Pass-by-value
  - Prototype

    ```
    int FunctionByValue(int);
    ```
  - Implementaiton

    ```
    int FunctionByValue(int X) {
    return X*2;}
    ```
  - Invoke/call the function

    ```
    int Y=10;
    cout << FunctionByValue(Y);
    ```

Pass-by-reference with reference arguments

- Pass-by-reference with reference arguments (Chapter 5)
  - Prototype

    ```
    void FunctionByRef(int &);
    ```
  - Implementation

    ```
    void FunctionByRef(int &X) {
    X=X*2; }
    ```
  - Invoke/call the function

    ```
    int Y=10;
    FunctionByRef(Y);
    cout << Y;
    ```

Pass-by-reference with pointers

- Pass-by-reference with pointers
  - Prototype

    ```
    void FunctionByPointer(int *);
    ```
  - Implementation

    ```
    void FunctionByPointer(int *Xptr) {
    *Xptr= *Xptr * 2; }
    ```
  - Invoke/call the function

    ```
    int Y=10;
    FunctionByPointer(&Y);
     cout << Y;
    ```

Calling Functions by Reference

- Pass-by-reference with pointer arguments
  - Emulate pass-by-reference
    - Use pointers and indirection operator
  - Pass address of parameters using `&` operator
  - Arrays not passed with `&` because array name is already a pointer
  - `*` operator used as alias/nickname for variable inside of function

Why do we use pass-by-reference?

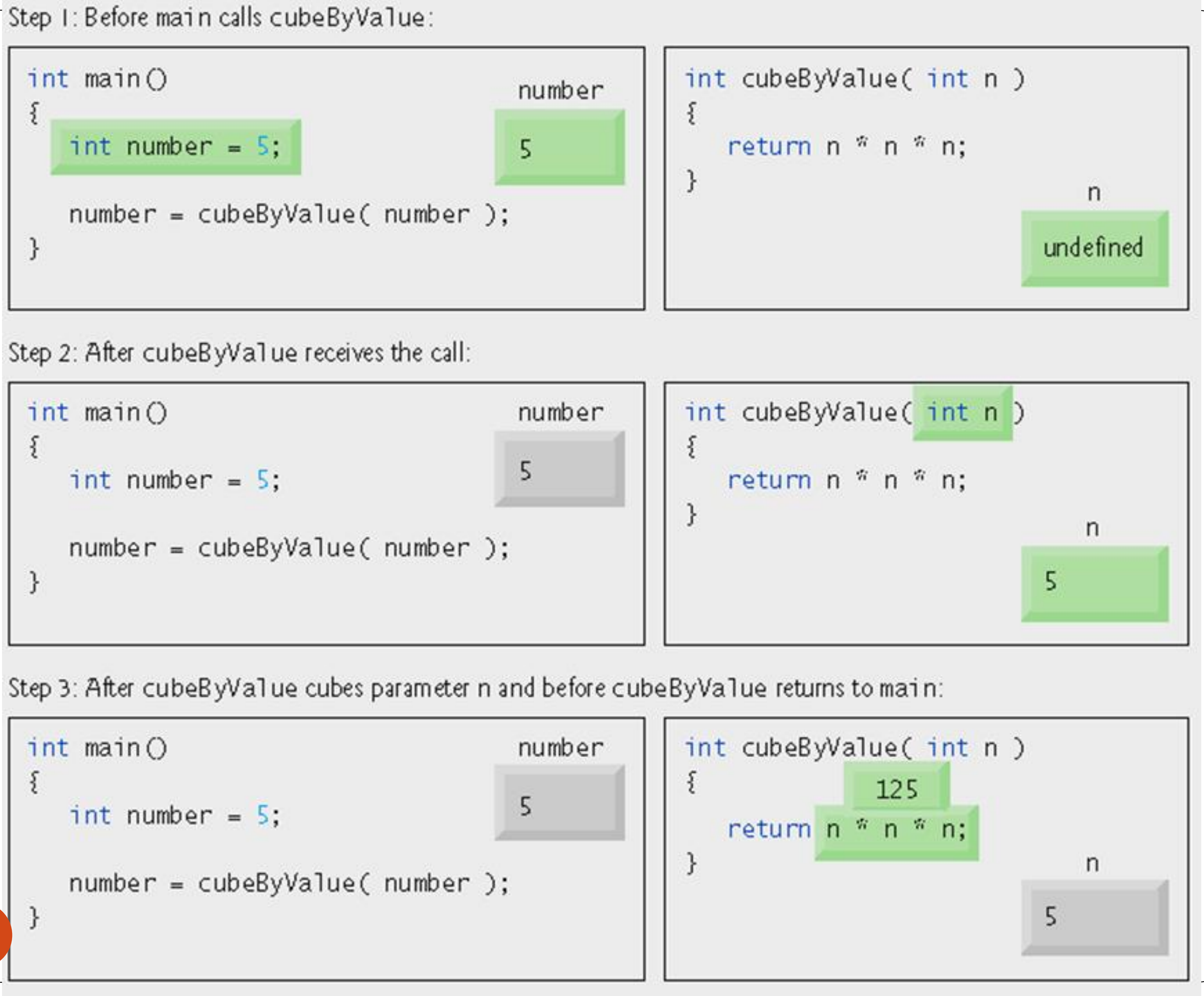
- *"return"* can only return one value from function
- Arguments passed to function using reference arguments
  - Modify original values of arguments
  - More than one value "returned"

Two more examples

- Compute n^3
  - Pass-by-value
    - Fig7.6
    - Fig7.8 --- memory operation
  - Pass-by-reference with a pointer argument
    - Fig7.7
    - Fig7.9 --- memory operation

```cpp
// Fig. 7.6: fig07_06.cpp
// Pass-by-value used to cube a variable's value.
#include <iostream>
using namespace std;

int cubeByValue( int ); // prototype

int main()
{
   int number = 5;

   cout << "The original value of number is " << number;

   number = cubeByValue( number ); // pass number by value to cubeByValue
   cout << "\nThe new value of number is " << number << endl;
} // end main

// calculate and return cube of integer argument
int cubeByValue( int n )
{
   return n * n * n; // cube local variable n and return result
} // end function cubeByValue
```

Fig. 7.6 | Pass-by-value used to cube a variable's value. (Part 1 of 2.)



Step 1: Before main calls cubeByValue:

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

Step 2: After cubeByValue receives the call:

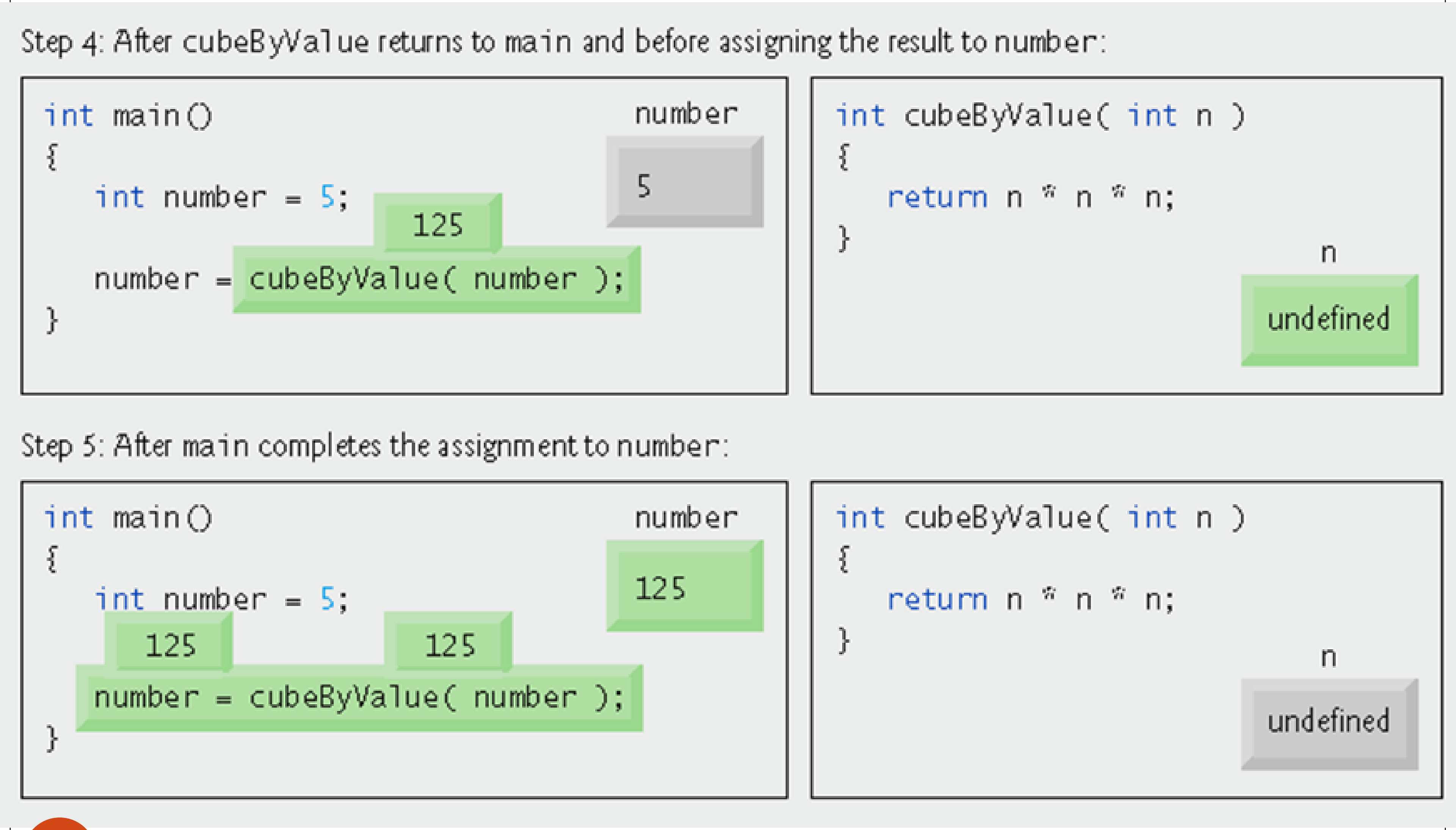
```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

Step 3: After cubeByValue cubes parameter n and before cubeByValue returns to main:

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```


Step 4: After cubeByValue returns to main and before assigning the result to number:

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

Step 5: After main completes the assignment to number:

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

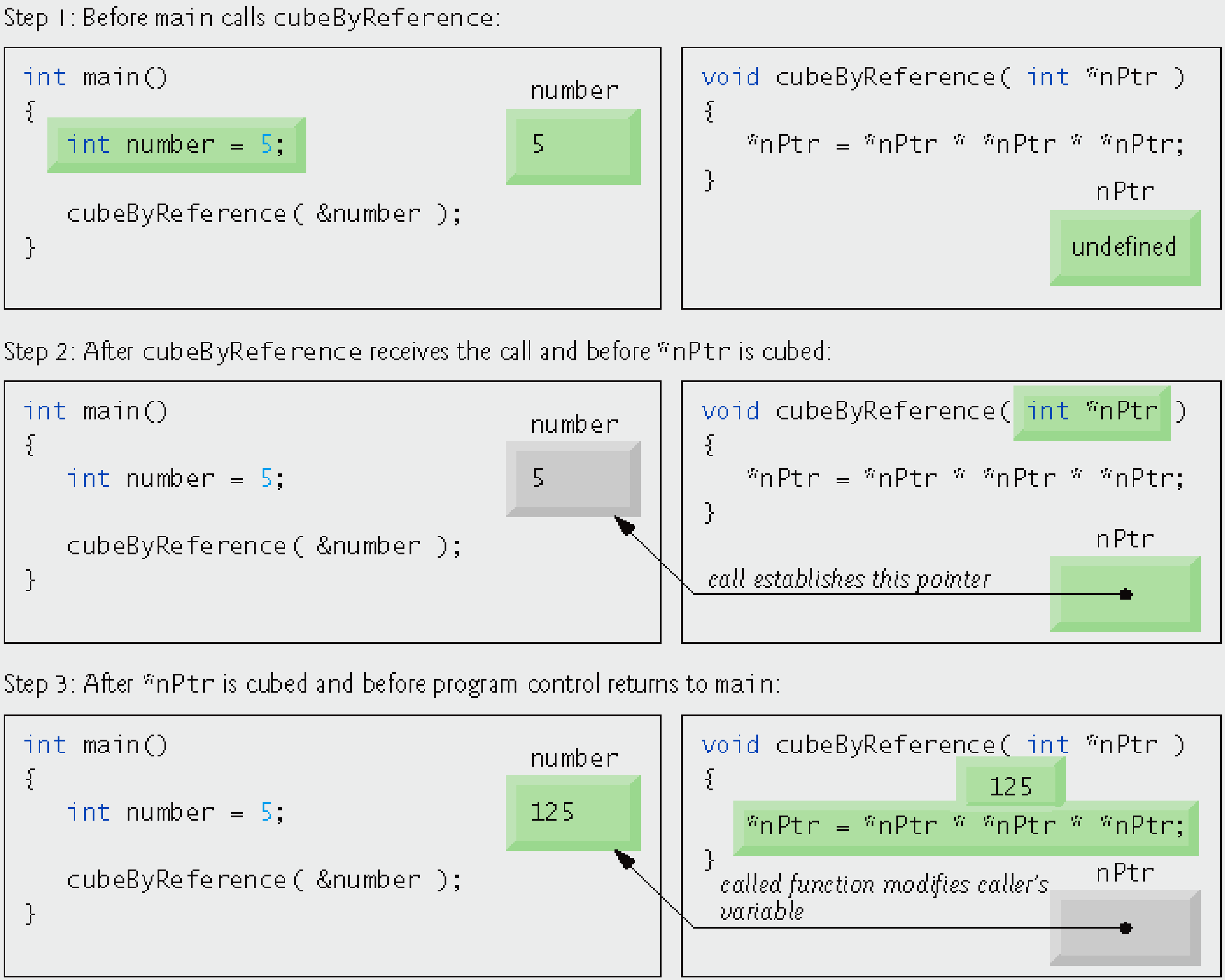
```cpp
// Fig. 7.7: fig07_07.cpp
// Pass-by-reference with a pointer argument used to cube a
// variable's value.
#include <iostream>
using namespace std;

void cubeByReference( int * ); // prototype

int main()
{
   int number = 5;

   cout << "The original value of number is " << number;

   cubeByReference( &number ); // pass number address to cubeByReference

   cout << "\nThe new value of number is " << number << endl;
} // end main

```

Fig. 7.7 | Pass-by-reference with a pointer argument used to cube a variable's value. (Part 1 of 2.)

```
// calculate cube of *nPtr; modifies variable number in main
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
} // end function cubeByReference
```

```
The original value of number is 5
The new value of number is 125
```

Fig. 7.7 | Pass-by-reference with a pointer argument used to cube a variable's value. (Part 2 of 2.)



Step 1: Before main calls cubeByReference:

```
int main()
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

Step 2: After cubeByReference receives the call and before *nPtr is cubed:

```
int main()
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

Step 3: After *nPtr is cubed and before program control returns to main:

```
int main()
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```