Chapter 4

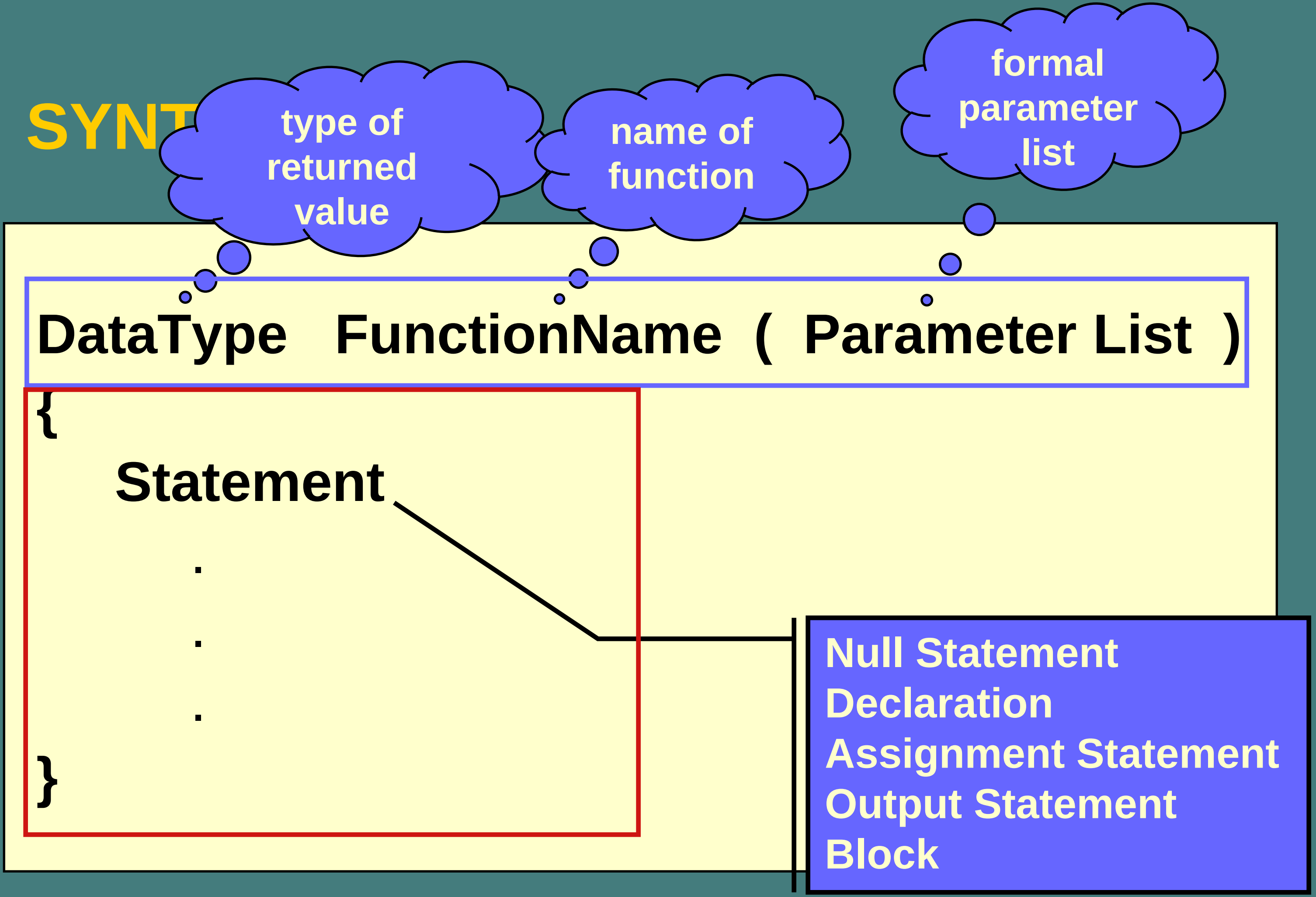
Functions

Chapter 4 Topics(part 1)

- Function Declaration
  - Function Definition
  - Function Prototype
- Function Call
  - Value-returning Function
  - void Function
- Parameters
  - Value Parameter
  - Reference Parameter
  - Function Parameters with Default Values

Function Definition

SYNT



```
DataType   FunctionName  (  Parameter List  )
{
     Statement
          .
          .
          .
}
```

Null Statement
Declaration
Assignment Statement
Output Statement
Block

What is a prototype?

- ❖A prototype looks like a heading but must end with a semicolon(;)
- ❖and its parameter list just needs to contain the type of each parameter.

SYNTAX

```
DataType   FunctionName  (  Parameter List  );
```

To Compile Successfully,

- before a function is called in your program, the compiler must previously process either the function's prototype, or the function's definition (heading and body)

Chapter 4 Topics(part 1)

- Function Declaration
  - Function Definition
  - Function Prototype
- Function Call
  - Value-returning Function
  - void Function
- Parameters
  - Value Parameter
  - Reference Parameter
  - Function Parameters with Default Values

Function Calls

- One function calls another by using the name of the called function next to ( ) enclosing an argument list.
- A function call temporarily transfers control from the calling function to the called function.

Function Call Syntax

```
FunctionName (  Argument List  )
```

- The argument list is a way for functions to communicate with each other by passing information.
- The argument list can contain 0, or more arguments, separated by commas(,).

When a function is called,

- temporary memory is set up (for its value parameters and any local variables, and also for the function’s name if the return type is not void).
- Then the flow of control passes to the first statement in the function’s body. The called function’s body statements are executed until one of these occurs:

  return statement (with or without a return value),

  or,

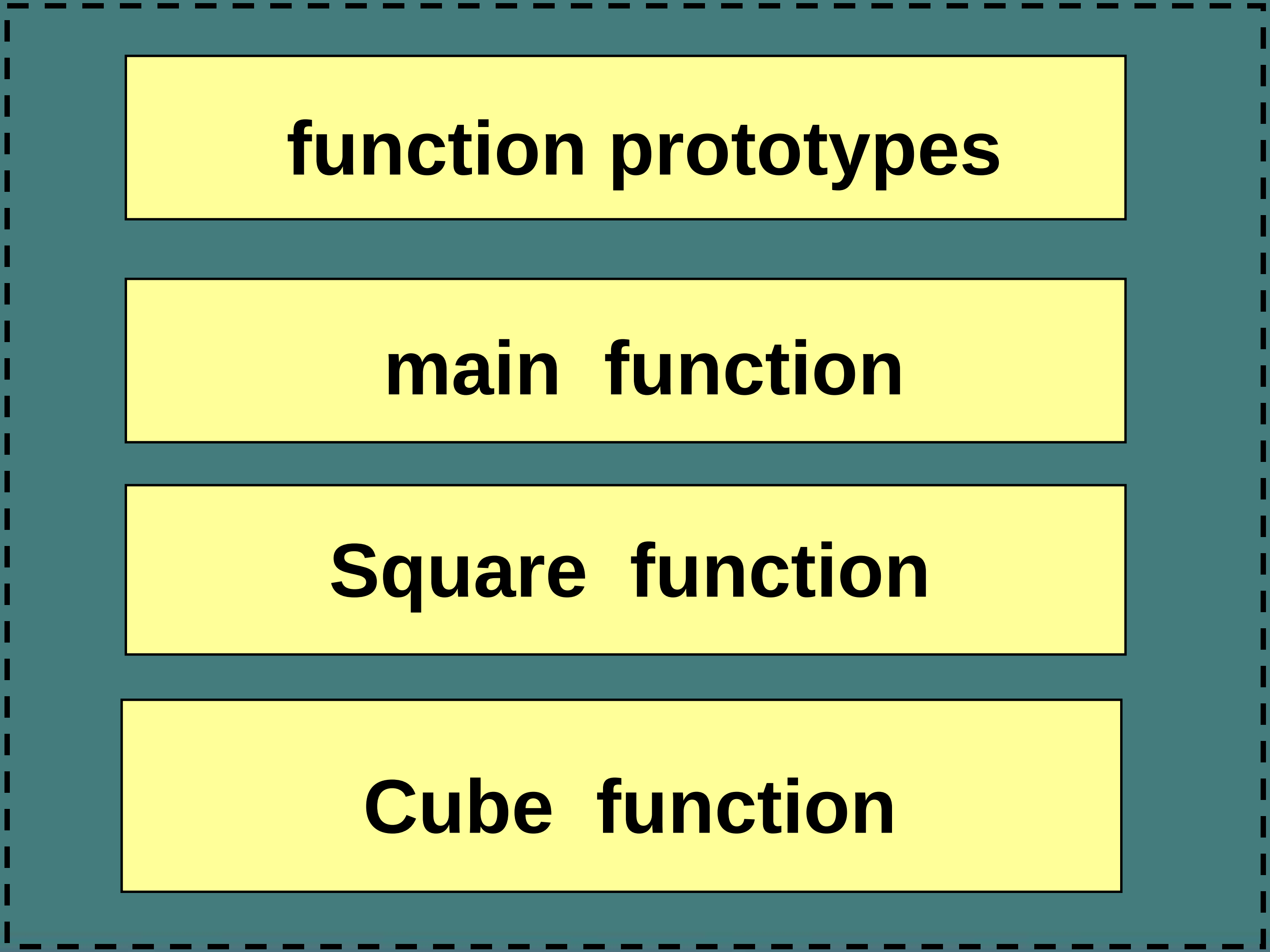
  closing brace of function body.
- Then control goes back to where the function was called.

A C++ function can return

- in its identifier at most 1 value of the type which was specified (called the return type) in its heading and prototype

- but, a void-function cannot return any value in its identifier

Program with Several Functions



Value-returning Functions

```
#include <iostream>

int  Square ( int ) ;                                         // prototypes
int  Cube ( int ) ;
using namespace std;

int  main (  )
{
     cout  <<  “The square of 2 is “
              <<   Square (2)    <<    endl;        // function call

     cout  <<  “The cube of 2 is “
             <<  Cube (2)      <<    endl;          // function call
    return 0;
}
```

Rest of Program

```
int Square ( int  n )                    // header and body
{
     return   n * n;
}


int Cube ( int  n )                      // header and body
{
    return  n * n * n;
}
```

A void function call stands alone

```
#include  <iostream>

void  DisplayMessage ( int ) ;                    // prototype

using namespace std;

int  main (  )
{
    DisplayMessage(15);                            // function call

     cout  <<  “Good Bye“  <<    endl;

    return 0;
}
```

argument

A void function does NOT return a value

parameter

```
void  DisplayMessage ( int  n )
{
     cout  <<  “I have liked math for  “
             <<  n  <<  “ years”  << endl ;

}
```

Chapter 4 Topics(part 1)

- ❖ Function Declaration
  - ❖ Function Definition
  - ❖ Function Prototype
- ❖ Function Call
  - ❖ Value-returning Function
  - ❖ void Function
- ❖ Parameters
  - ❖ Value Parameter
  - ❖ Reference Parameter
  - ❖ Function Parameters with Default Values

Parameter List

- is the means used for a function to **share information** with the block containing the call

Classified by Location

Arguments	Parameters
Always appear in a function call within the calling block.	Always appear in the function heading, or function prototype.

4000

25

age

Argument in Calling Block

Value Parameter	Reference Parameter
The value (25) of the argument is passed to the function when it is called.	The memory address (4000) of the argument is passed to the function when it is called.
In this case, the argument can be a variable identifier, constant, or expression.	In this case, the argument must be a variable identifier.

By default, parameters

- (of simple types like int, char, float, double) are always value parameters, unless you do something to change that.

- To get a reference parameter you need to place & after the type in the function heading and prototype.

When to Use Reference Parameters

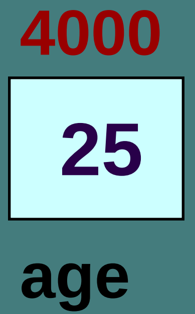
❖ reference parameters should be used when you want your function to give a value to, or change the value of, a variable from the calling block without an assignment statement in the calling block

Using a Reference Parameter

- is like giving someone the key to your home

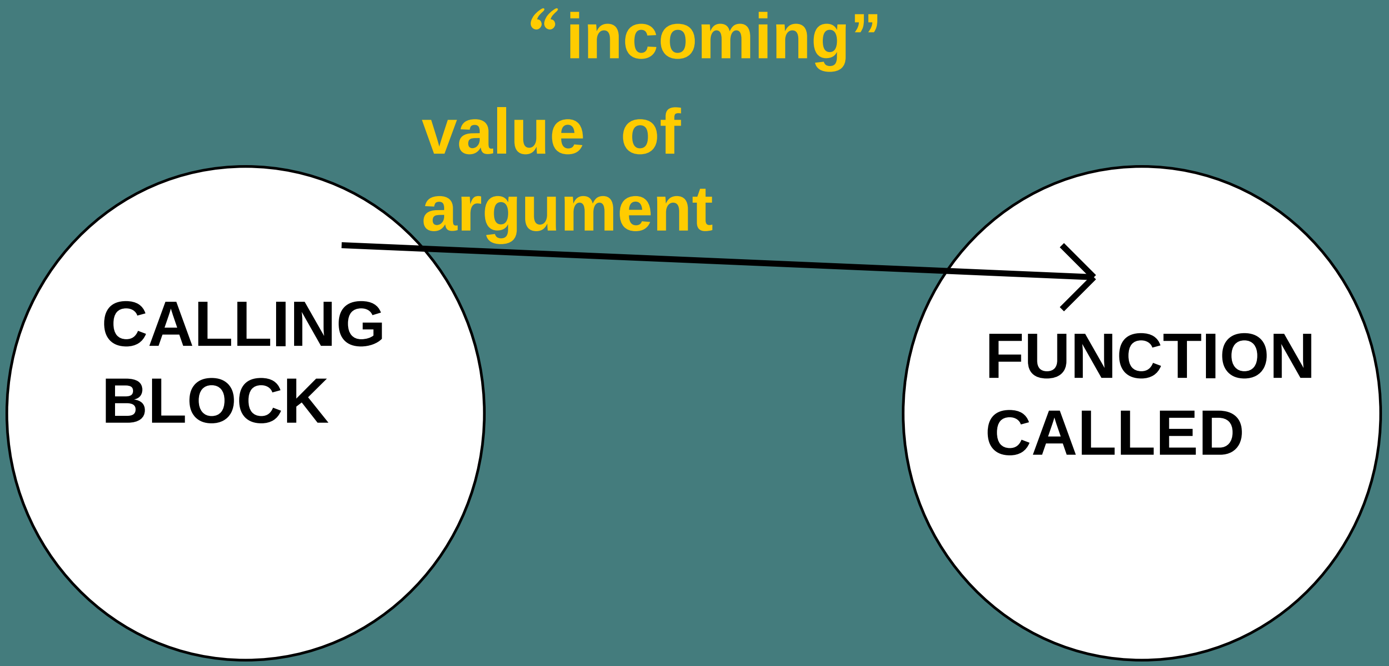
- the key can be used by the other person to change the contents of your home!

pass-by-value

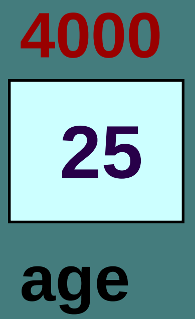


- If you pass only a copy of 25 to a function, it is called **“pass-by-value”** and the function will not be able to change the contents of age. It is still 25 when you return.

Pass-by-value

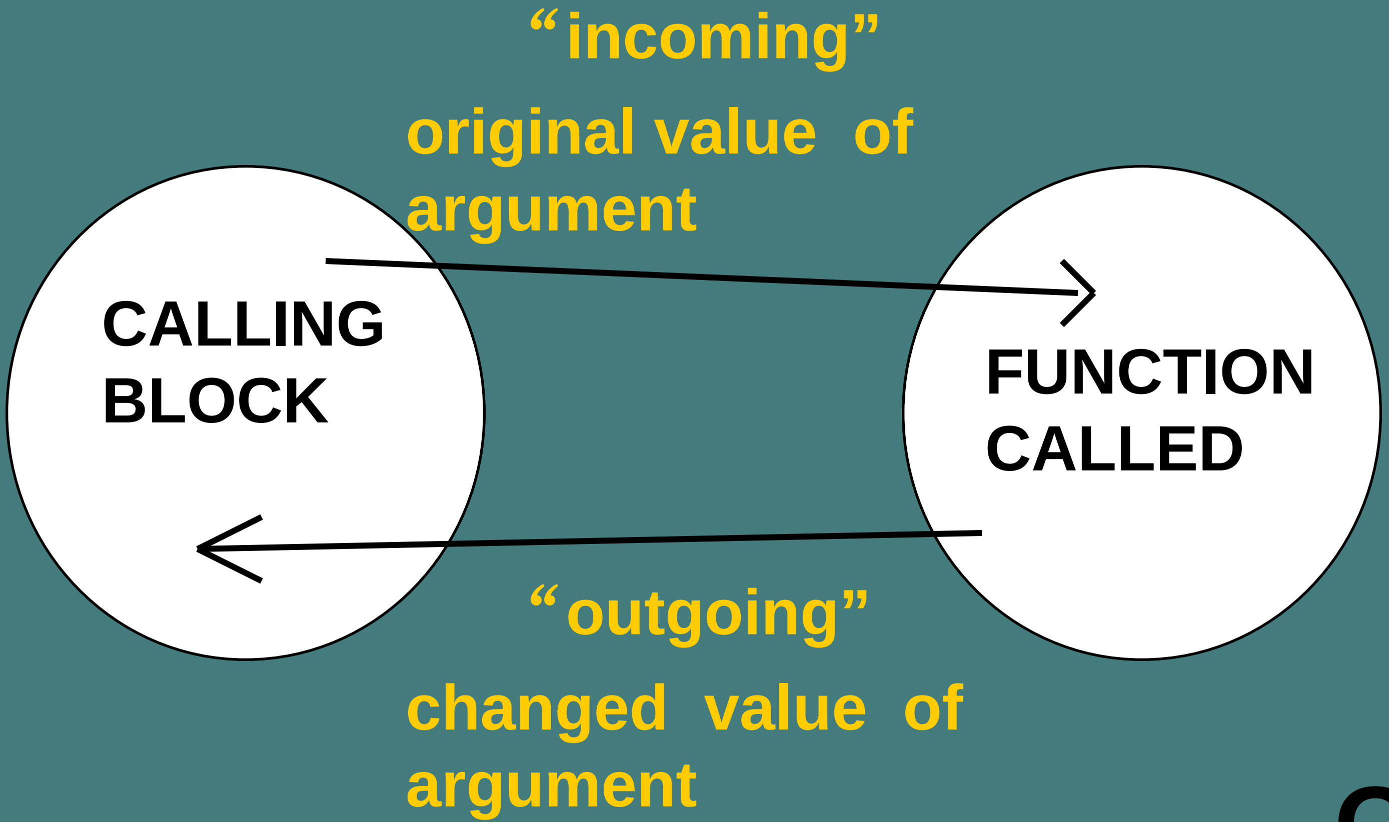


pass-by-reference



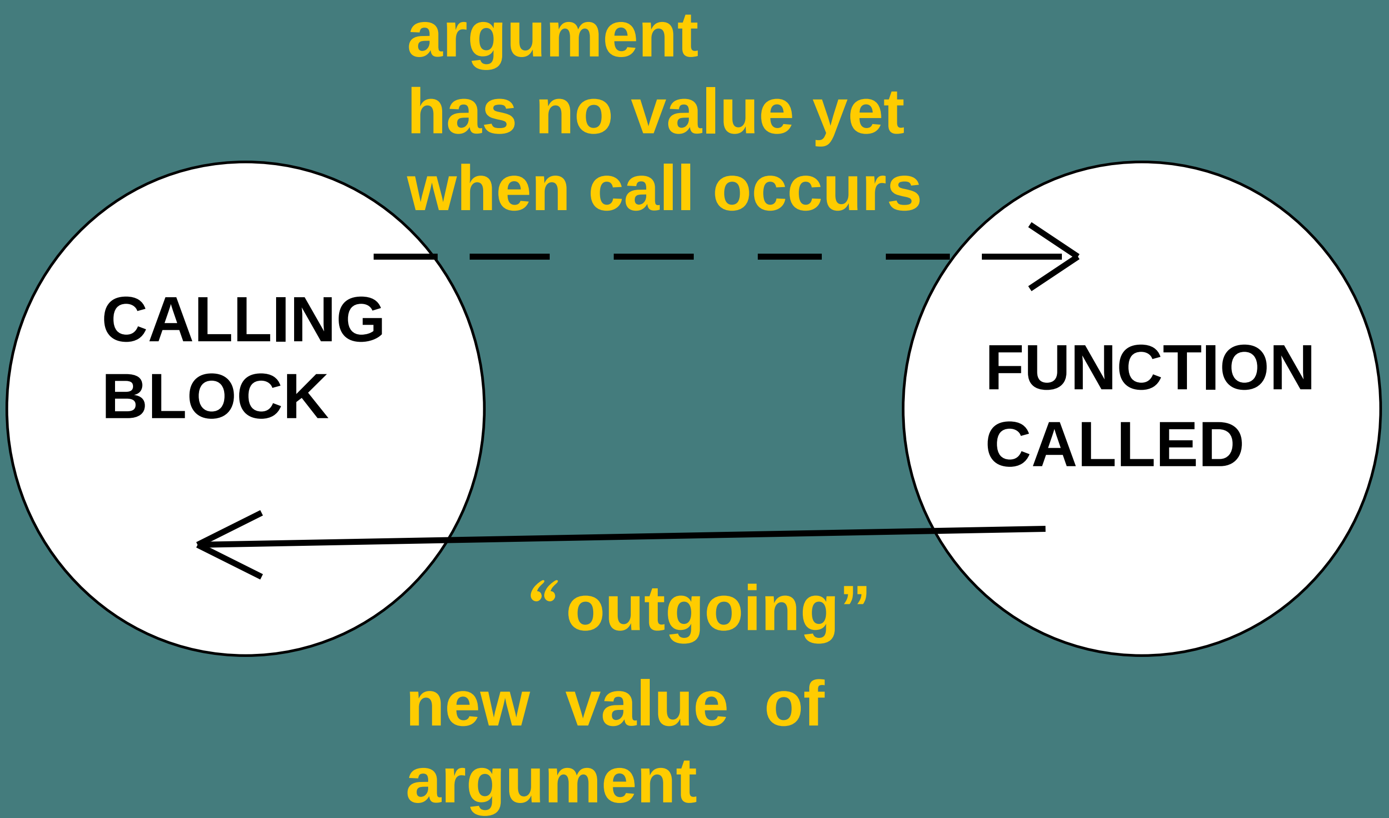
- BUT, if you pass 4000, the address of age to a function, it is called "pass-by-reference" and the function will be able to change the contents of age. It could be 23 or 90 when you return.
- pass-by-reference is also called pass-by-address or pass-by-location

Pass-by-reference



OR,

Pass-by-reference



Data Flow Determines Passing-Mechanism

Parameter Data Flow	Passing-Mechanism
Incoming /* in */	Pass-by-value
Outgoing /* out */	Pass-by-reference
Incoming/outgoing /* inout */	Pass-by-reference

Function Parameters with Default Values

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

// one parameter has default value 0
int sum (int a, int b, int c=0) { return a + b + c; }

int main( )
{
    int i=10, j=20, k=30;
    //sum up three int numbers
    cout<<i<< ' + ' <<j << ' + ' <<k<< ' = ' <<sum(i, j, k)<<endl;
    //sum up two int numbers
    cout<<i<< ' + ' <<j<< ' = ' <<sum(i, j)<<endl;
    return 0;
}
```