

# Scope and Parameter Passing

Lecture 16

Sections 6.5, 6.10, 6.13

Robb T. Koether

Hampden-Sydney College

Mon, Oct 7, 2013

## 1 The Scope of an Object

- Statement Scope
- Block Scope
- Function Scope
- Global Scope

## 2 Parameter Passing

- Value Parameters
- Reference Parameters

## 3 Examples

## 4 Assignment

# Outline

## 1 The Scope of an Object

- Statement Scope
- Block Scope
- Function Scope
- Global Scope

## 2 Parameter Passing

- Value Parameters
- Reference Parameters

## 3 Examples

## 4 Assignment

# The Scope of an Object

- The **scope** of an object is the part of the program over which that object is recognized.
- The scope of an object may be
  - Statement scope – A single statement.
  - Block scope – A block within a function.
  - Function scope – A function.
  - Class scope – A class (to be discussed soon).
  - Global scope – The entire program.
- An object cannot be accessed outside of its scope.

# The Scope of an Object

- When a variable **passes out of scope**, it ceases to exist.
- The memory that was allocated to it is deallocated.
- A variable whose scope is the entire program is called **global**.
- All other variables are **local** (to the statement, block, function, etc.).

# Outline

## 1 The Scope of an Object

- Statement Scope
- Block Scope
- Function Scope
- Global Scope

## 2 Parameter Passing

- Value Parameters
- Reference Parameters

## 3 Examples

## 4 Assignment

# Statement Scope

- A **volatile**, or **unnamed**, object is an object that occurs as the value of an expression or a subexpression.
- For example, `a + b` is a volatile object in the statement

`c = a + b;`

- The scope of a volatile object is the statement in which it occurs.
- When the statement is completely evaluated, the volatile object passes out of scope.

# Outline

## 1 The Scope of an Object

- Statement Scope
- **Block Scope**
- Function Scope
- Global Scope

## 2 Parameter Passing

- Value Parameters
- Reference Parameters

## 3 Examples

## 4 Assignment

# Block Scope

- If an object is constructed (declared) within a block `{ }` in a function, its scope is limited to that block.
- The scope extends from the point of declaration to the end of the block in which the variable is declared.
- When execution leaves the block, the object passes out of scope.

# Block Scope

## Block Scope

```
if (a > b)
{
    int max = a;    // Block scope
}
else
{
    int max = b;    // Block scope
}
cout << "The max is " << max << endl;
```

- This will not work. Why?

# Block Scope

## Block Scope

```
if (a > b)
    int max = a;
else
    int max = b;
cout << "The max is " << max << endl;
```

- This will not work either. Why?

# Block Scope

## Block Scope

```
int max;           // Function scope  
if (a > b)  
    max = a;  
else  
    max = b;  
cout << "The max is " << max << endl;
```

- This will work. Why?

# Outline

## 1 The Scope of an Object

- Statement Scope
- Block Scope
- **Function Scope**
- Global Scope

## 2 Parameter Passing

- Value Parameters
- Reference Parameters

## 3 Examples

## 4 Assignment

# Function Scope

- Variables declared within a function are recognized only within that function.
- The scope extends from the point of declaration to the end of the function.
- When execution returns from the function, the object passes out of scope.

# Function Scope

## Function Scope

```
int max(int a, int b)
{
    int m;           // Function scope
    if (a > b)
        m = a;
    else
        m = b;
    return m;
}
```

- The variable `m` is not accessible from anywhere outside of this function.

# Outline

## 1 The Scope of an Object

- Statement Scope
- Block Scope
- Function Scope
- **Global Scope**

## 2 Parameter Passing

- Value Parameters
- Reference Parameters

## 3 Examples

## 4 Assignment

# Global Scope

- A variable is global if it is declared outside of any function.
- Global variables are recognized in all functions.
- Good programming practice strongly discourages the use of global variables.

# Examples of Scope

- Example
  - `ScopeExample.cpp`

# Outline

- 1 The Scope of an Object
  - Statement Scope
  - Block Scope
  - Function Scope
  - Global Scope
- 2 **Parameter Passing**
  - Value Parameters
  - Reference Parameters
- 3 Examples
- 4 Assignment

# Methods of Passing Parameters

- Parameters may be passed by one of two methods.
  - By value.
  - By reference.

# Outline

- 1 The Scope of an Object
  - Statement Scope
  - Block Scope
  - Function Scope
  - Global Scope
- 2 **Parameter Passing**
  - **Value Parameters**
  - Reference Parameters
- 3 Examples
- 4 Assignment

# Value Parameters

- When an object is **passed by value**, the value of the actual parameter is *copied* to the formal parameter.
- The function uses the local copy of the parameter.
- The **copy constructor** is used to make the copy (to be discussed later).
- The actual parameter is left unchanged, no matter what is done to the local copy.

# Value Parameters

```
int main()  
{  
    :  
    f(a, b)  
    :  
}
```



```
int f(int c, int d)  
{  
    :  
    sum = c + d;  
    :  
}
```

a and b are local to main()

# Value Parameters

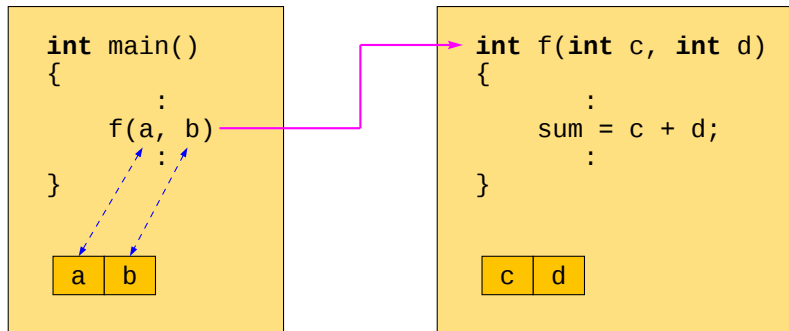
```
int main()  
{  
    :  
    f(a, b)  
    :  
}
```



```
int f(int c, int d)  
{  
    :  
    sum = c + d;  
    :  
}
```

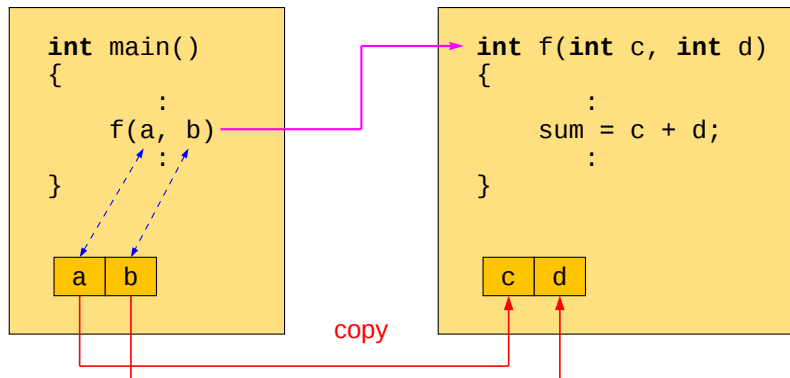
Their values are stored locally

# Value Parameters



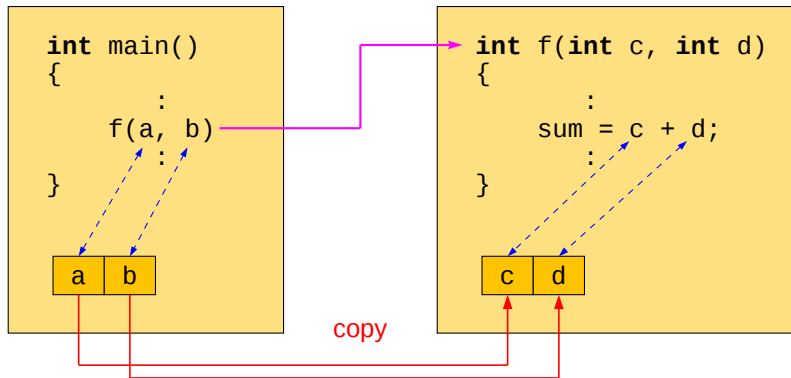
Memory is allocated in `f()` for the copies

# Value Parameters



Copies of `a` and `b` are made in `f()`

# Value Parameters



`c` and `d` are local to `f()`

# Outline

- 1 The Scope of an Object
  - Statement Scope
  - Block Scope
  - Function Scope
  - Global Scope
- 2 **Parameter Passing**
  - Value Parameters
  - **Reference Parameters**
- 3 Examples
- 4 Assignment

# Reference Parameters

- When an object is **passed by reference**, the *address* of the actual parameter is passed to the function.
- The formal parameter becomes a reference to the actual parameter, i.e., the actual parameter and the formal parameter are the same object.
- If the formal parameter is changed, so is the actual parameter.
- An ampersand & is used in the formal parameter list to indicate a reference parameter.

# Reference Parameters

```
int main()  
{  
    :  
    f(a, b)  
    :  
}
```



```
int f(int& c, int& d)  
{  
    :  
    sum = c + d;  
    :  
}
```

a and b are local to main()

# Reference Parameters

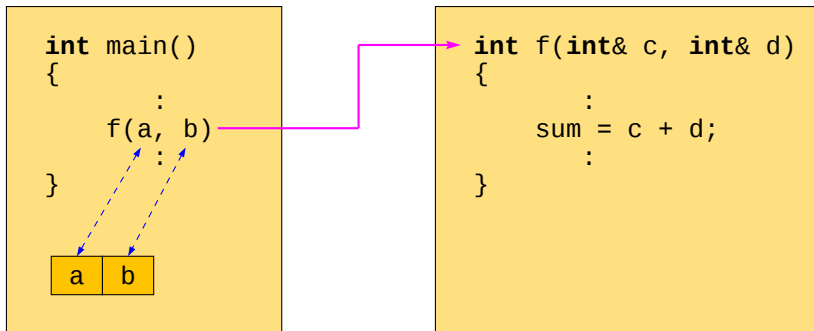
```
int main()  
{  
    :  
    f(a, b)  
    :  
}
```



```
int f(int& c, int& d)  
{  
    :  
    sum = c + d;  
    :  
}
```

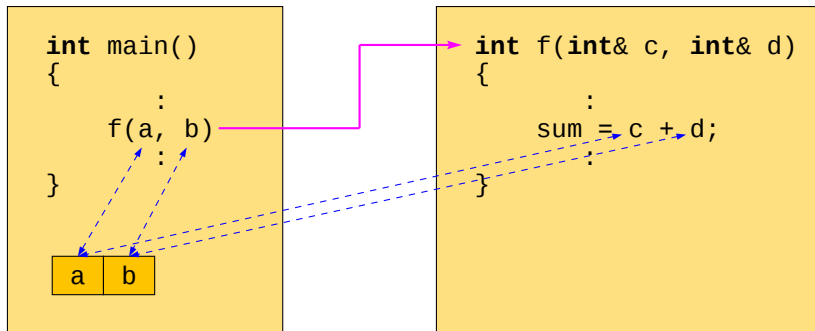
Their values are stored locally

# Reference Parameters



No memory is allocated in `f()`

# Reference Parameters



`c` and `d` refer to `a` and `b` in `main()`

# Outline

- 1 The Scope of an Object
  - Statement Scope
  - Block Scope
  - Function Scope
  - Global Scope
- 2 Parameter Passing
  - Value Parameters
  - Reference Parameters
- 3 Examples
- 4 Assignment

# Examples of Parameter Passing

- Examples

- `GCDExample.cpp`
- `GetWordFunc.cpp`

# Outline

- 1 The Scope of an Object
  - Statement Scope
  - Block Scope
  - Function Scope
  - Global Scope
- 2 Parameter Passing
  - Value Parameters
  - Reference Parameters
- 3 Examples
- 4 Assignment

# Assignment

## Assignment

- Read Sections 6.5, 6.10, 6.13.