Scope

Scope is an *attribute* (or property) of an *identifier* (or *name)*.

The *scope* of an identifier is the region of the program code in which it can be used.

local scope
An identifier declared within a block (braces { })
scope extends from declaration to end of block.
parameter scope extends to the whole function definition.

global scope
An identifier declared outside all blocks.
scope extends from declaration to the end of the compilation unit (source file).

As a property of names, it applies to

- variable names
- constant names
- function names
- class names

as well as to other names.

Scope Examples

A do-nothing example with many of the names declared multiple times.

Mouse Instructions
To find out what the scope of a name is roll the mouse over where the name is first declared. Note that if you click on the names you can freeze the scope marking, letting you scroll around the program. Click on the same name again to turn it off.

scope_demo.cpp

```
/*************************************************************
 * Memorial University of Newfoundland
 * Engineering 2420 Structured Programming
 * scopeDemo.cpp -- A silly demo of scope
 *
 * Input: none
 * Output: none
 *
 * Author: Michael Bruce-Lockhart
 *
 *************************************************************/
#include <iostream>
```



```
using namespace std;

int john = 3;
int mary = 4;

void functionFoo();
void functionFaa();

int main(){
 int john = 2;
 int mary = 1;

    { int jules = 0;
        mary += jules;
        cout << "Inside block: jules is " << jules << " and mary is " << mary;
        cout <<endl;
    }
    cout << "\nAfter block: john is " << john << " and mary is " << mary;
    cout <<endl;
    functionFoo();
    cout << "\nAfter function: john is " << john << " and mary is " << mary;
    cout <<endl;
    return 0;
 }

void functionFoo(){
 int john = 17;
 int mary = 13;
    cout << "\nInside functionFoo: john is " << john << " and mary is " << mary
    cout << endl;
 }

void functionFaa(){
 int mary = 29;
    cout << "\nInside functionFaa: john is " << john << " and mary is " << mary
    cout << endl;
}
```

Visibility

Scope & visibility are slightly different

It is possible for a name to be in scope but not be visible

There are lots of john's and mary's in the example above. Like families, the scope of many of them overlap. How do we know which is which?

We repeat the same example below, only now we have marked it for visibility. Note that the first john, which is declared at the *external level*, is only visible at the external level as well as inside functionFoo. This is because main and functionFoo have their own johns. We say these johns *occlude* the other one.

```
/*************************************************
   a (silly)demonstration of Visibility
```

visibility_demo.cpp

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

int john = 3;
int mary = 4;

void functionFoo();
void functionFaa();

int main(){
    int john = 2;
    int mary = 1;

    for (int jules = 0; jules < 5; jules++){
        mary += jules;
        cout << "Inside loop: jules is " << jules << " and mary is " << mary;
        cout <<endl;
    }
    cout << "\nAfter loop: john is " << john << " and mary is " << mary;
    cout <<endl;
    functionFoo();
    cout << "\nAfter function: john is " << john << " and mary is " << mary;
    cout <<endl;
    return 0;
 }

void functionFoo(){
    int john = 17;
    int mary = 13;
    cout << "\nInside functionFoo: john is " << john << " and mary is " << mary
    cout << endl;
 }

void functionFaa(){
    int mary = 29;
    cout << "\nInside functionFaa: john is " << john << " and mary is " << mary
    cout << endl;
}
```

- Try to keep the scope of variables small—it can make your code easier to read and modify.
- Prefer local variables over global variables.
- Use parameters rather than global variables to pass information to functions.
- If your parameter list is too long maybe your function is doing too much.

A more detailed discussion of the concept of scope is available here.

This page last updated on Monday, January 26, 2004