

scope (of an identifier) the range of program statements within which the identifier is recognized as a valid name

C++ Scope Rules

1. Every identifier must be declared and given a type before it is referenced (used).
2. The scope of an identifier begins at its declaration.
3. If the declaration is within a compound statement, the scope of the identifier ends at the end of that compound statement. We say the identifier is local to that compound statement (or block).
4. If the declaration is not within a compound statement, the scope of the identifier ends at the end of the file. We say the identifier has global scope.

binding determining which declaration of an identifier corresponds to a particular use (reference) of that identifier

C++ Binding

Binding identifier references to declarations is the responsibility of the compiler. When the compiler finds a reference it searches for a matching declaration of the name. This search is conducted according to the following rules.

1. A declaration of a constant or variable must match the identifier name exactly.
2. A declaration of a function must also match the number and parameter types.
3. The search proceeds upward (backward) from the location of the reference.
4. If there is no matching declaration in the block containing the reference, and that block is contained within an "enclosing" block, then the search continues into that "enclosing" block.
5. The search will not enter a block enclosed within the block currently being searched. That is, a block is "private" when viewed from outside it.
6. If necessary the compiler will continue this process until global scope is reached and searched.
7. If no matching declaration is found, the reference to the identifier is invalid and an "undeclared identifier" message is issued.

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

// Constants with global scope:
const int QUITVALUE      = 0;
const string UserPrompt = "Please enter . . . (zero to quit): ";
const string NewMaxMsg  = "That beats the old maximum by: ";

int main() {

    int Max = INT_MIN;    // local to main()
    int UserEntry;

    cout << UserPrompt;  // declarations of cout and cin are . . .?
    cin  >> UserEntry;

    . . .
```

```
...
while (UserEntry != QUITVALUE) {

    if (UserEntry > Max) {
        int Increase;           // local to the if-body
        Increase = UserEntry - Max;
        cout << NewMaxMsg << Increase << endl;
        Max = UserEntry;
    }

    // couldn't refer to Increase here

    cout << UserPrompt;
    cin  >> UserEntry;
}

return 0;
}
```

One of the most contentious issues facing novice programmers is the use of identifiers that have global scope.

Since the publication of Edsger Dijkstra's "Use of Globals Considered Harmful" in 1966, there has been a growing consensus among software engineers that identifiers should be declared at the global level only reluctantly.

As a general policy in CS 1044, global scope is to be used only for:

- constants
- function declarations
- type definitions

The use of globally-scoped variables is expressly forbidden. We will discuss why this prohibition is enforced as we consider various examples. Regardless of your prior programming experience and attitudes, do not ignore this rule.

lifetime (of a variable) the period of time, during the execution of a program, when the variable has memory allocated to it

Local (or automatic) variables:

The lifetime begins when execution enters the function in which the variable is declared.

The lifetime ends when the function terminates (returns).

Global (or static) variables:

The lifetime begins when the program begins execution.

The lifetime ends when the the program terminates.