Instructions

For a each topic (if applicable) there are 5 sections:

Short Answer

Answer each question as well as you can.

Circle the Variables' Scopes

Circle the scope of the given variables – i.e., circle the part of the code in which each variable "exists". A good test, if you're not sure, is: if you inserted a statement to `cout` the variable at a given point in the code, would any errors be produced?

Fill in the Memory

Fill in (or draw) a picture representing the computer memory, showing how the variables might be allocated, and whether they have an assigned value, or are uninitialized (use `?` for "uninitialized").

Fix the Broken Code

Modify the code in the cleanest (and shortest) way possible, so that it will compile and run without errors or warnings.

Trace the Working Code

Trace through the execution of the code, keeping track of the value of each variable at each point in time, and the final output that the program produces.

1 Variables and Assignment

Short Answer

Question

Describe undeclared variables, uninitialized variables, and initialized variables.

Answer

Undeclared variables don't really exist. This term usually shows up in error messages when you try to use a variable that you haven't declared yet (either because you forgot, or you misspelled or mis-capitalized it, etc.).

Uninitialized variables are declared, but have not yet been given a value. This means that their value is undefined (i.e., garbage, or whatever was in that location in memory before it was assigned to that variable). Variables should not be used before being given a value.

Initialized variables are both declared an initialized. This means that they have been assigned a location in memory, and that that memory has been set to some desired value.

Question

Describe what we mean when we refer to a variable's scope.

Answer

A variable's scope is the region in the code where the variable "exists". Inside this scope, after the variable is declared, the variable may be used. Outside this scope, or before the variable has been declared, attempting to use the variable will result in a compile time error (in C++).

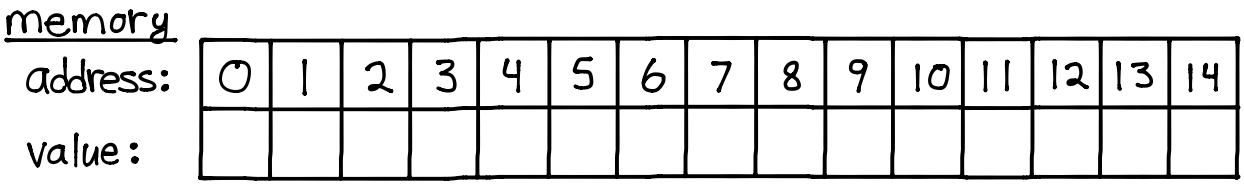
In practice, the scope of most variables is determined by the innermost pair of curly braces containing it.

It is possible for a variable in an inner scope to have the same name as a variable in an outer scope. If this is the case, the variable in the inner scope is said to "mask" the variable from the outer scope, beginning from the point at which it is declared. Once execution proceeds outside the inner scope, the variable from the outer scope will be "visible" (i.e., usable) again.

Circle the Variables' Scopes

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

int main() {
    int a = 3, b = 3, c = 3;

    {
        int a = 5, b = 5;
        c = b;

        {
            int a = 7;
            b = a;
        }
    }

    return 0;
}
```

Fill in the Memory




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

int main() {
    int a = 3, b = 3, c = 3;

    {
        int a = 5, b = 5;
        c = b;

        {
            int a = 7;
            b = a;
        }
    }

    return 0;
}
```

Fix the Broken Code

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

int main() {
    int a = 5;
    int b = 7;

    cout << "before swapping: " << a << " " << b << endl;

    // swap
    int temp = a;
    a = b;
    b = temp;

    cout << "after swapping: " << a << " " << b << endl;

    return 0;
}
```

```
--- code/1-fix-1.cpp        2014-04-30 22:09:04.000000000 -0700
+++ code/1-fix-1.answer.cpp        2014-04-30 22:26:24.000000000 -0700
@@ -8,9 +8,9 @@
     cout << "before swapping: " << a << " " << b << endl;

     // swap
-    int temp = int a;
-    int a = int b;
-    int b = int temp;
+    int temp = a;
+    a = b;
+    b = temp;

     cout << "after swapping: " << a << " " << b << endl;
```

```
before swapping: 5 7
after swapping: 7 5
```

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

int main() {
    int    i = '5' + 4.7;  // '5' == 53
    char   c = 42;         // '*' == 42
    bool   b = 10;
    double d =  7;
    string s = "hello world!";

    int u = 42;  // otherwise we're using an uninitialized variable below

    cout << i << " " << c << " " << b << " "
         << d << " " << s << " " << u << endl;

    return 0;
}
```

```
--- code/1-fix-2.cpp        2014-04-30 22:21:38.000000000 -0700
+++ code/1-fix-2.answer.cpp        2014-05-02 11:27:14.000000000 -0700
@@ -9,7 +9,7 @@
     double d =  7;
     string s = "hello world!";

-    int u;
+    int u = 42;  // otherwise we're using an uninitialized variable below

     cout << i << " " << c << " " << b << " "
          << d << " " << s << " " << u << endl;
```

```
57 * 1 7 hello world! 42
```

Trace the Working Code

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

int main() {
    cout << "a b temp\n" << "--------\n";

    int a = 5;      cout << a << endl;
    int b = 7;      cout << a << " " << b << endl;

    int temp = a;   cout << a << " " << b << "  " << temp << endl;
    a = b;          cout << a << " " << b << "  " << temp << endl;
    b = temp;       cout << a << " " << b << "  " << temp << endl;

    return 0;
}
```

```
a b temp
--------
5
5 7
5 7  5
7 7  5
7 5  5
```

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

int main() {
    cout << "a b c d\n" << "-------\n";

    int a, b, c;

    a = b = c = 3;  cout << a << " " << b << " " << c << endl;
    b = c = 5;      cout << a << " " << b << " " << c << endl;
    c = 7;          cout << a << " " << b << " " << c << endl;

    int d = b;      cout << a << " " << b << " " << c << " " << d << endl;

    return 0;
}
```

```
a b c d
-------
3 3 3
3 5 5
3 5 7
3 5 7 5
```

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

int main() {
    int a = 3, b = 3;
    cout << "a: " << a << "  b: " << b << endl;
    {
        int a = 5;
        b = a;
        cout << "a: " << a << "  b: " << b << endl;
    }
    cout << "a: " << a << "  b: " << b << endl;

    return 0;
}
```

```
a: 3  b: 3
a: 5  b: 5
a: 3  b: 5
```

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

int main() {
    int a = 3, b = 3, c = 3;
    cout << "a: " << a << "  b: " << b << "  c: " << c << endl;
    {
        int a = 5, b = 5;
        c = b;
        cout << "a: " << a << "  b: " << b << "  c: " << c << endl;
        {
            int a = 7;
            b = a;
            cout << "a: " << a << "  b: " << b << "  c: " << c << endl;
        }
        cout << "a: " << a << "  b: " << b << "  c: " << c << endl;
    }
    cout << "a: " << a << "  b: " << b << "  c: " << c << endl;

    return 0;
}
```

```
a: 3  b: 3  c: 3
a: 5  b: 5  c: 5
a: 7  b: 7  c: 5
a: 5  b: 7  c: 5
a: 3  b: 3  c: 5
```

2 Data Types and Expressions

Circle the Variables' Scopes

Fill in the Memory

Fix the Broken Code

Trace the Working Code

3 If and If-Else

Circle the Variables' Scopes

Fill in the Memory

Fix the Broken Code

Trace the Working Code

4 Boolean Expressions

Circle the Variables' Scopes

Fill in the Memory

Fix the Broken Code

Trace the Working Code

5 Predefined Functions

Circle the Variables' Scopes

Fill in the Memory

Fix the Broken Code

Trace the Working Code

6 Loops

Circle the Variables' Scopes

Fill in the Memory

Fix the Broken Code

Trace the Working Code

7 Arrays

Circle the Variables' Scopes

Fill in the Memory

Fix the Broken Code

Trace the Working Code

8 Selection Sort

Circle the Variables' Scopes

Fill in the Memory

Fix the Broken Code

Trace the Working Code