Practice test for midterm 1 – Solutions

September 17, 2020

1 Basics of C++

► How many comments, directives, declarations, definitions, and statements occur in the following program?

```
/*
 * myprogram.cpp
 */
#include <iostream>
#include <string>
using namespace std;

int main() {
    cout << "Hello"; cout << " to you" << endl;
    int i;
    cout << "Enter your age: ";
    cin >> i;
    cout << "TMI, pal" << endl;
}
```

- Comments: 1
- Directives: 2
- Declarations: 2 (`using` and the variable declaration)
- Definitions: 1
- Statements: 5 (two statements on the first line of `main`)

► Circle all the mistakes (errors) in the following program, and mark what changes you'd have to make to fix them.

```
#include <iostreams>                 // iostream
using namespace std                  // Missing ;

int main()                           // Missing {
    cout << "Hello, sir! << endl;    // Missing " after `sir!`
    int i;
    cin >>> i;                       // >> not >>>
    cout << "OK!" >> endl            // << not >>, missing ;
}
```

► Write the output that the following program will produce.

```
int main() {
    cout << "Hello!" << endl;/*
    cout << "Goodbye!" << endl;*/
    cout << "Hello again." << endl;
    return 0;
    cout << "Goodbye for good!" << endl;
}
```

Output:

```
Hello
Hello again.
```

(`"Goodbye!"` is commented out, and `"Goodbye for good!"` is after the `return 0;` which ends the program.)

► Label each of the following either *declaration* or *statement*:

- `int x = 12 * 2;` *Declaration*
- `string s = "hello";` *Declaration*
- `cout << x << s;` *Statement*
- `string t = s.substr(0,3);` *Declaration*
- `using namespace std;` *Declaration*
- `return 0;` *Statement*

2 Expressions, variables, and types

► For the following code fragment, trace through it to determine the final values of the variables a and b.

```
int a = 0, b = 1;
a = a + b;              // a = 1
{
    int c = a = b;      // a = 1, c = 1
    c = a + b;          // c = 2
    a = 2 * c;          // a = 4
}                       // End of c's scope
b = a + b * 2;          // b = 4 + 1 * 2 = 6
```

Final values:

```
a = 4
b = 6
```

► There are *three* problems with variable scoping in the following program. Circle them, and describe each one briefly.

```
int main() {
    int a = 1;
    {
        int b = 2;
        cout << a << b;
        {
            int c = 4;
            cout << c;
        }
    }
    cout << b << c;       // b not in scope, c not in scope
    int c = 3;
    cout << a << b << c;  // b not in scope
}
```

► Write a program which reads an entire line of input from the user, and then prints it back out, as if it was a quote from Michael Scott. For example, if the user entered

```
I don't see the point of this.
```

the program should print

```
"I don't see the point of this."
    ~Michael Scott
```

Solution:

```
string s;
getline(cin,s);

cout << "\"" << s << "\"" << endl;
cout << "    ~Michael Scott" << endl; // Could also use \t to indent
```

► Describe what each of the following *character escapes* will produce when printed (sent to `cout`):

- `\n` Newline (`endl`)
- `\\` Backslash
- `\t` Tab
- `\"` Double-quote
- `\b` Backspace (maybe)

► Here is a sample program that reads three numbers from the user, and then prints out one of the three. Describe, in English, which of the three it prints. (You might try tracing through it with some inputs to see what it does.)

```
int main() {
    cout << "Enter three integers: ";
    int a, b, c;
    cin >> a >> b >> c;

    int d = b < c ? b : c;

    cout << (a < d ? a : d);
}
```

Answer: this prints the *smallest* of the three numbers entered. d will become the smaller of b and c, which is then compared to a, giving the smallest of all three.

3 Conditional and looping statements

► Describe what the following program will print out, depending on the user's input:

```
int x;
cin >> x;
if(x < -10) {
    cout << -10;
}
else if(x > 10) {
    cout << 10;
}
else {
    cout << x;
}
```

Answer: If the user enters a number between -9 and +9 then their number is printed unchanged. If the user enters a number $<$ -9, then `-10` is printed; if the user enters a number $>$ +`10`, then `10` is printed. (This is known as "clamping", restricting a value to a given range by stopping it at the ends of that range.)

► Simplify the following if-else statements:

-
  ```
  if(x < 0 && x > 0)
      cout << "OK";
  else
      cout << "Not cool";
  ```

- Answer: `x < 0 && x > 0` is impossible (always false), so the `else` part will always run.

  ```
  cout << "Not cool";
  ```

-
  ```
  if(!(x < y || x > y))
      cout << "Yes";
  else if(x == y)
      cout << "No";
  else
      cout << "Maybe?";
  ```

- Answer: `!(x < y || x > y)` is equivalent to `x >= y && x <= y`, which in turn is equivalent to `x == y`. Since an `if-else` chain will run the *first* branch that is true, the second `x == y` will never happen. Thus, we simplify to just:

  ```
  if(x == y)
      cout << "Yes";
  else
      cout << "Maybe?";
  ```

-
  ```
  string s = ...;
  if(s.empty() == (s.length() == 0))
      cout << "PoTAYto";
  else
      cout << "PoTAHto";
  ```
- Answer: `s.empty()` and `s.length() == 0` will always have the same true/false value (because `s.empty()` *means* “is the length of s equal to 0”), so the condition simplifies to just `true`, which means the `else` part never happens, thus we get

  ```
  cout << "PoTAYto";
  ```

▶ Write a `while` loop which will print out all the even numbers from 2 to 20.

Answer: there are several equally-correct ways to do this:

```
int i = 2;
while(i <= 20) {
    cout << i << endl;
    i += 2;
}
```

```
int i = 1;
while(i < 21) {
    if(i % 2 == 0)
        cout << i << endl;
    ++i;
}
```

```
int i = 1;
while(i <= 10) {
    cout << i * 2 << endl;
    ++i;
}
```

and so forth.

▶ Write a `do-while` loop which will count *down* to 1 from the number entered by the user. If the user enters a number < 1 it should print 1 and stop.

```
int n;    // Count down from n
cin >> n;
// Your loop goes here

// Answer:
// This is the easiest way to get the "just print 1 and stop" behavior
if(n < 1)
    n = 1;

do {
    cout << n << endl;
    --n;
} while(n >= 1);
```

▶ Translate the following `for` loop into a `while` loop that does the same thing:

```
for(int x = 0; x < 100; x = x * x + 1) {
```

```
    cout << x;
    if(x % 7 == 0)
        break;
}
```

Answer:

```
int x = 0;            // Initialization
while(x < 100) {      // Condition

    cout << x;
    if(x % 7 == 0)    // Body
        break;

    x = x * x + 1;    // Update
}
```

(Bonus question: what will this loop print out?)

Answer: it will print `0` and then stop (because `0 % 7 == 0`).