Expressions: Solutions

Order of Operations

Question 1:

```
#include <iostream>
#include <cmath> //needed for certain functions
using namespace std;
int main()
{
    double a,b,c,d,e;
    cout << "Enter the five numbers: ";
    cin >> a >> b >> c >> d >> e;
    double x, y, z;
    x = pow(b,2.0) + pow(c,5.0) + pow(d,2.0)/3;
    y = a + b/2*8 + 90;
    z = 3*a + 4*b + 5*c;
    cout << "X: " << x << " Y: " << y << " Z: " << z;
}
```

Question 2:

```
#include <iostream> //typo here
#include <cmath> //sqrt lives here!

using namespace std; //using, not use

  int main() //main didn’t have a return type, int
  {
        double x, y; //semicolon!
        cin >> x;
        //lots of problems below!
        y = pow(x,2.0) + 5*x + sqrt(x + 2) - (1.0/2)*x;
        cout << y;
  }
```

Question 3

a) 6

b) 6.71

c) 64

d) 1

e) 1

f) 2

Combined Assignment

Question 1

The output of the cout statement will be shown below with a brief explanation

a) Output is 5. The line x=x is allowed but does absolutely nothing useful.

b) Output is 10. The right side is evaluated first, producing the value 10. This value is then stored in the variable x.

c) This will not compile. In mathematics, it's allowed to write 10=x to mean that the variable x stores the value 10. That is not permitted in C++. Anything to the LEFT of the equals sign must be something that can be changed, and the number 10 cannot be changed.

d) Output is 10. This code breaks down into the following:

```
int x = 5;
x = x + x;
cout << x;
```

e) Output is 6. This is counter-intuitive, but consider that the pre-increment operator happens before anything else. So this code breaks down into:

```
int x = 5;
x = x + 1; //the pre-increment operator happens first!
x = x; //totally useless line of code
cout << x;
```

f) Output is 12. This is also counter-intuitive, but the same reason as for part (e) still holds. The code becomes:

```
int x = 5;
x = x + 1; //the pre-increment operator happens first!
x = x + x; //no longer totally useless, but pretty much…
cout << x;
```

g) Output is 17. All expressions to the right of the equals sign are evaluated first, then the addition happens, then the equality. The following code will produce identical output:

```
int x = 5;
int temp = 7 + x; //temp stores the value 12
x = x + temp; //x now stores the value 17
cout << x;
```

h) Code will not compile. Once a symbol is declared as constant, it cannot be changed.

Logical Expressions

Question 1

a) Output is 1, or true.

b) Output is 1, or true.

c) Output is 1, or true. This counter-intuitive result can be determined by either following the order of operations for logical operators, or putting brackets in to clarify what happens first. The code is equivalent to:

```
int x = (5 < 4) < 3;
```

The first thing to be evaluated is the expression (5<4), which is false. This is represented as a 0 in the computer. Then the expression (0 < 3) is evaluated, which is a true statement. A key take-away here is that logical expressions follow the order of operations!

d) Output is 0, or false. Although it is not immediately obvious, the relational operators (<, >, >=, <=) take precedence over the logical operators (&&, ||) in the order of operations. Once again we can add brackets to make things less ambiguous. The code is equivalent to:

```
int x = (5 < 4) && (4 > 3);
```

You may notice that, although these brackets aren't strictly necessary from the perspective of what the code **does**, they are vital to aid us when we try to read the code. We suggest always adding brackets if there is any ambiguity.

e) Output is 1, or true. The order of operations still holds as it did in part (d).

f) Output is 1, or true. There is only one way to represent false in c++, which is 0. Anything that is not 0 is therefore true.

g) Output is 1, or true.