Fundamentals: Solutions to Drill Problems

Writing and running a new program

Question 1:

Part a:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, world!" << endl;
}
```

Part b:

It won't work because the library iostream contains the cout object. If we don't include it, the code cannot compile.

Part c:

It will not work without modification since the cout object lies within the namespace std.

Part d:

The modification is:

```
#include <iostream>
using namespace std;
int main()
{
    std::cout << "Hello, world!" << std::endl;
}
```

Question 2:

This can be done on multiple lines or a single line. Two such examples are shown below. There are many other ways to solve this problem.

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Alice" << endl << "Bob" << endl << "Clarice" << endl;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Alice" << endl;
    cout << "Bob" << endl;
    cout << "Clarice" << endl;
}
```

Declaring variables

Question 1:

Part a:

It depends on the character. Underscores are allowed as characters to start a variable name. Numbers are not allowed and will not compile. Other special characters (aka: punctuation and math operations) will not compile.

Part b:

Keywords are not valid variable names and the program will not compile. However, keywords can be included in a variable name. For example, the name intX is allowed, although it may be considered stylistically questionable.

Question 2:

Part a:

```
int main()
{
    double books = 5.0, gpa = 4.6, creditsThisTerm = 2.75;
}
```

Part b:

There are several ways to approach this. The code below demonstrates three possibilities.

```
int main()
{
    //declare variables of the same type on one line
    double x,y;
    int a,b,c;
    char t,s;

    //declare on separate lines
    double x;
    double y;
    int a;
    int b;
    int c;
    char t;
    char s;

    //declare and initialize
    double x = 0, y = 1;
    int a = 5, b = 0, c = 10;
    char t = 'x', s = 65;

}
```

Question 3:

```
#include <iostream>
```

```
using namespace std;
int main()
{
    int x;
    cout << x;
}
```

Students should see randomized values, but this depends on their computer's current state. Many students may see a value of 0 and assume that the computer initializes variables to zero for them. **This is not the case!** The computer does not auto-initialize variables!

Working with variables: changing their values with math

Question 1

```
#include <iostream>
using namespace std;
int main()
{
    //Part a: using only one line
    int x,y,z;
    cin >> x >> y >> z;

    //Part b: using three separate statements
    cin >> x;
    cin >> y;
    cin >> z;
}
```

Notice that using a single line makes for more compact code that takes less time to write, but reading on multiple lines makes for code that might be more clear.

Question 2

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    const double RAD_TO_DEG = 180/M_PI;
    const double DEG_TO_RAD = M_PI/180;

    double rads;
    cout << "Enter a value in radians";
    cin >> rads;

    double degs = rads*RAD_TO_DEG;
    double backToRads = degs*DEG_TO_RAD;

    cout << degs << " " << backToRads << endl;
}
```

Note: it's entirely possible that you always got the result you expected. It's actually quite difficult to come up with test cases that break this. Hint: what happens if the number of decimal places is large, say above 8?

Question 3

```
#include <iostream>
using namespace std;
int main()
{
    double a, b, c, x, y;
    cout << "Enter values for the coefficients a, b, and c," << endl;
    cout << "Then a value for the variable x" << endl;
    cin >> a >> b >> c >> x;

    y = a * x * x + b * x + c;
    cout << y;
}
```

Note: you could have chosen to use the pow function. In this case, I chose not to since my variable names were simple enough and I was just squaring a value. It would not be incorrect to use the pow function, and indeed it may make the code more clear.

Here are some suggested test cases to try:

- Let one of the coefficients be zero
- Let some combination of the coefficients be zero
- Let some coefficients be positive, some negative
- Test x = 0, -1, 1
- Test x = M_PI (requires #include <cmath>), or some other approximation of an irrational number. (why do you think this was important?)

All of your test cases should be checked through hand calculations, otherwise, how will you know if you're right?

Question 4:

```
#include <iostream>
using namespace std;
int main()
{
    double x, y;
    cout << "Enter a value for the variable x" << endl;
    cin >> x;

    y = (2.0/3)*(x+4)/(x-10); //Remember to avoid integer division!
    cout << y;
}
```

Note: this function introduces potential issues with integer division. If you enter only 2/3, rather than 2.0/3 (or some other way to cast to a double), then you will encounter an error in that y = 0 for any input of x.

Here are some suggested test cases:

- Test x = 0, -1, 1
- Test x = M_PI (requires #include <cmath>), or some other approximation of an irrational number.

- Testing x = 10 should produce a value of “inf” or “nan”, depending on your computer, but the code still runs. This is known as a “silent failure”, and can be very bad! It means that the code didn’t alert you to the failure, it just kept running and produced output.

All test cases should be confirmed by a hand calculation.

Question 5

The variable “sum” contains the value 0. Sum is not updated when a and b are updated. Instead, sum retains the value that it had when it was created. Since the value is never changed again, the variable itself is unchanged.

The unique problems and opportunities with integers and mathematics

Question 1

Part a:

```
#include <iostream>
using namespace std;
int main()
{
    int x, y;
    cout << "Enter two integers!" << endl;
    cin >> x >> y;

    cout << x/y;
}
```

Part b:

There are many ways to write this code. Three suggestions are shown below.

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

int main()
{
    int x, y;
    cout << "Enter two integers!"<< endl;
    cin >> x >> y;

    //we can cast implicitly:
    cout << 1.0*x/y << endl;

    //we can cast explicitly
    cout << (double)x/y << endl;

    //we can store additional variables
    double dX = x, dY = y;
    cout << dX/dY;
}
```

Here are some suggested test cases:

- Testing any case where x is a multiple of y will produce the correct result. For instance, x = 10, y = 2, will produce 5.
- Testing any case where x is NOT a multiple of y will induce integer division errors. Test x = 10, y = 3, for example, to get the answer 3.
- Code for part b should be re-tested using the same test cases as for part a, to compare answers.

Question 2:

The student incorrectly assumed that because the variable "result" was a double, the computer would case **the calculation of the quotient** to a double. In reality, the computer will first evaluate the right hand side of the expression, in which both variables are integers. This will introduce integer division errors. The result will get cast to a double only after the computation of the right hand side is complete.

The student can test the code with many test cases. Two suggestions are:

x = 1, y = 12
x = 10, y = 3

This code can be modified in several ways to avoid the integer division. Ultimately, the result must be cast to a double before it the computation is carried out. One potential method to solve the problem is shown below:

```
#include <iostream>

using namespace std;

int main()
{
    int a, b;
    double result;
    cout << "Enter values for a and b!" << endl;
    cin >> a >> b;
    result = 1.0*a/b; //Note that I am implicitly casting here
    cout << result;
}
```

Question 3:

Part a:

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

int main()
{
    int x;
    cout << "Enter an integer: ";
    cin >> x;

    //using modulo
    cout << x % 10;

    //using integer division.  Note: this is not a good way to do this!
```

```
    int lastDigit = x - (x/10)*10;
    cout << lastDigit;
}
```

Part b:

We are going to use the modulo operator, which makes for far more compact code.

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

int main()
{
    int x;
    cout << "Enter an integer: ";
    cin >> x;

    int digit1, digit2, digit3;
    digit1 = x%10;
    x = x/10; //need to remove that last digit to simplify matters
    digit2 = x%10;
    x = x/10;
    digit3 = x;
    cout << digit1 << endl;
    cout << digit2 << endl;
    cout << digit3 << endl;
}
```

Part c:

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

int main()
{
    int x;
    cout << "Enter an integer: ";
    cin >> x;

    int digit1, digit2, digit3;
    digit1 = x%10;
    x = x/10; //need to remove that last digit to simplify matters
    digit2 = x%10;
    x = x/10;
    digit3 = x;
    int reversedX = 100*digit1 + 10*digit2 + digit3;
    cout << x + reversedX;
}
```

Question 4

In the expression a%b, the sign of the output of the expression is always the same as the sign of a.

The C++ math library

Question 1

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double x;
    cout << "Enter a value for x in radians: " << endl;
    cin >> x;
    double result = pow(sin(x),2) + pow(cos(x),2) - 1;
    cout << result;
}
```

Note: I chose to use both the pow function and the sin/cos functions to increase the number of cmath library functions I used.

Note also that you can test this all day and still see results that agree with what you expect, or you could stumble upon an input that makes it fail. It depends very strongly on your computer. Hint: try numbers with large numbers of digits after the decimal, like 8 or more.

Question 2

- The first line should output 1, since 0 rad = 0 degrees.
- The second line should output 1, since the argument to sin is in radians and I know that sin(M_PI/2) = 1 from high school math.
- The third line will output -0.45 or something close. If you predicted that you'd see a zero, remember that the argument to sin and cos must be in radians, so the input to the function was 90 rad, not 90 degrees.
- The fourth line will output zero, since the ½ term is evaluated using integers 1 and 2, and therefore this suffers from an integer division error.
- The fifth line will output 1 for the same reason. The ½ term is evaluated using integers 1 and 2, which evaluates to 0. Then the exponent is evaluated. Anything to the exponent 0 is 1.
- The sixth line will output 15 (or a suitably close rounding of that number, depending on student computer).

The seventh line will output "nan", or "not a number". This is also a silent failure, since the computer did not obviously indicate that the input was bad.