CS 7A – quiz 4 – Chapters 1–4 - Fall 2015 - . Show work for credit. Write on separate paper.

1. Find as many errors in the following code as you can. Classify each error as a syntax error or a logic error and describe how to fix it so it works properly.

```
1 int main() {
      int num, bigNum, power, count;
3     cout << "\nEnter␣an␣integer:␣";
      cin >> num; bigNum = num;
5     cout << "\nWhat␣power␣do␣you␣want␣it␣raised␣to?␣";
      cin >> power;
7     while(count++ < power);
          bigNum *= num;
9     cout << "\nThe␣result␣is␣" << bigNum << endl;
  }
```

If you compile and run this code (as shown) below, you might, at first, thing there are no errors.

```
Enter an integer: 10
What power do you want it raised to? 2
The result is 100
```

However, another run bring this into doubt:

```
Enter an integer: 10
What power do you want it raised to? 3
The result is 100
```

In fact, the results are completely unpredictable. The variable `count` was never defined since it was not initialized. Thus it's value is whatever happens to be in the memory that was allocated for this variable. In these instances (output above) it seems that the memory happens to contain, 0, which was lucky, but 0 is not a terribly unusual value for random, uninitialized memory. It was a logic error to not initialize `count`.

Also, there is a semicolon at the end of the condition for the while loop. This means that, despite the indentation of the next line, the while loop has no body. But the post-increment operator (++) on `count` means that it is incremented as the otherwise empty while loop executes and so the while loop does terminate, after which the (wrongly) indented next line does execute once, so `bigNum` is multiplied by `num` exactly once. And since `bigNum` was initialized to `num`, the program will print `num*num` to the console. That's why the first run above gave a proper output, but the cube didn't give 1000.

To fix it initialize and update the variables like so:

```
1 int main() {
      int num, bigNum, power, count=0;
3     cout << "\nEnter␣an␣integer:␣";
      cin >> num;
5     cout << "\nWhat␣power␣do␣you␣want␣it␣raised␣to?␣";
      cin >> power;
7     bigNum = num;
      while(++count < power) {
9         bigNum *= num;
      }
11    cout << "\nThe␣result␣is␣" << bigNum << endl;
  } //It may clarity to separate update of count from condition of while loop.
```

2. Consider the following C++ program. Is there an error? If so, what kind of error? What will be the result of attempting to compile and execute the program?

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

int main()
{
    for(int i = 0; i < 2; ++i)
        for(int j = 0; j < 2; ++j)
        {
            cout << "i = " << i << " ";
            --i;
            cin.get();
        }
    return 0;
}
```

The program self-tabulates values of `i` but not values of `j`. Owing to the test restriction against using a electro-mechanical computing device, we must tabulate both by hand to be sure of what's going on.

Though the program pauses for a keypress, it is none-the-less caught in an infinite l∞p, a sort of runtime logic error.

i	j
0	0
-1	1
-1	0
-2	1
-2	0
-3	1
-3	0
-4	1

3. Determine the value of the following expressions, assuming `a = 5, b = 2, c = 4,` and `d = 5`;
   (a) `a == 5` evaluates to 0, or `false`.
   (b) `b + d == c * c`. Substituting, we see this is `2 + 7 == 4 * 4`, which evaluates to `false`.
   (c) `d % b * c > 5 || c % b * d < 7` evaluates to `4 > 5 || 0 < 7 = false or true = true = 1`. To verify these, you could (after the paper and pencil exam) write code like this:

```
int main() {
    int a{5}, b{2}, c{4}, d{5};
    int x = (d % b * c) > 5;
    cout << "\nd % b * c = " << d%b*c << "\n(d % b * c > 5) = " << x;
    int y = (d % b * c > 5) || (c % b * d < 7);
    cout << "\nc % b * d  = " << c % b * d;
    cout << "\n(d % b * c > 5) || (c % b * d < 7) = " << y;
}
```

   which produces this verifying output:

```
d % b * c = 4
(d % b * c > 5) = 0
c % b * d  = 0
(d % b * c > 5) || (c % b * d < 7) = 1
```

4. What is the output of the following code fragment?

```
int main() {
    for(int ct = 1; ct <= 3; ct++) {
```

```
        cout << ct;
4       for(int i = 3; i <=4; i++) {
            cout << ct << "␣" << i;
6       }
        cout << "do␣re␣mi";
8   }
}
```

The output is `11 31 4do re mi22 32 4do re mi33 33 4do re mi`

5. 9. What is the output produced by the following program if the user enters 4?

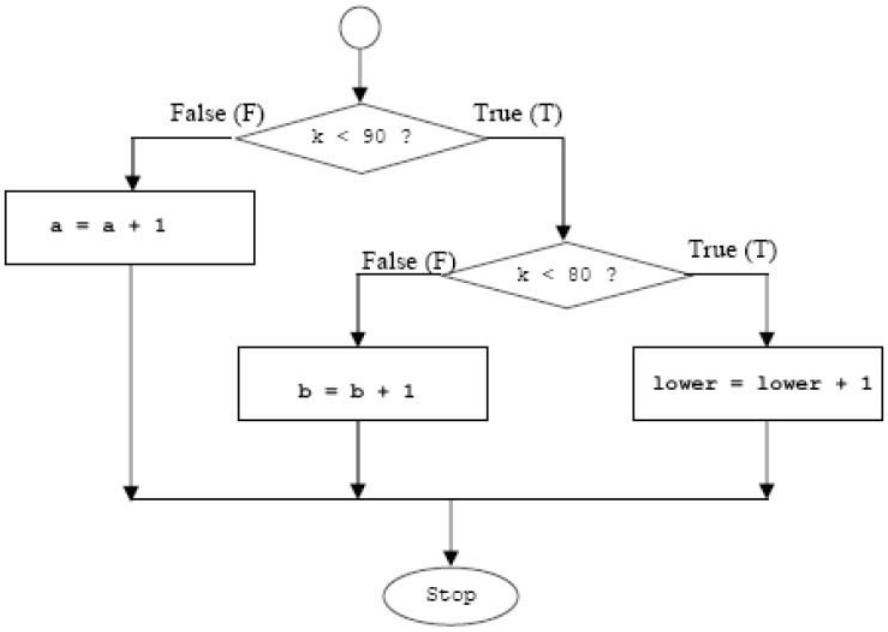
```
1  #include <iomanip>
   #include <iostream>
3  using namespace std;

5  int main()
   {
7          setiosflags(ios::right);
           int n;
9          cout << "How␣many␣columns?␣(1-16):␣";
           cin >> n;
11         for (int x=1; x <= n; x++)
           {
13                 for (int y=1; y <= n; y++)
                           cout << setw(5) << x*y;
15                 cout << endl;
           }
17 }
```

SOLN:

```
    1    2    3    4
    2    4    6    8
    3    6    9   12
    4    8   12   16
```

6. Write code to implement the following flow chart.



```
int main() {
    int a = 0, b = 0, lower = 0;
    for(int i = 0; i < 10; ++i)
    {
        cin >> k;
        if(k >= 90) ++a;
        else if(k >= 80) ++b;
        else ++lower;
    }
    cout << "\nThere␣are␣" << a
         << "␣90␣or␣more.\n" << b
         << "␣in␣[80,90)␣and␣\n"
         << lower << "␣less␣than␣80.";
}
```

The above l∞ps the algorithm in the flowchart 10 times, because why not?

7. 10. Write a program to

   (a) Find the square root of the sum of the first 100 even numbers, correct to 10 significant digits. Don't use not the `cmath` library. SOLN:

```
#include <iomanip>
#include <iostream>
using namespace std;
double root(double);

int main()
{
        double sum = 0;
        cout << setprecision(10);
        for (double n = 2; n < 100; n += 2)
                sum += n;
        cout << "\nsqrt(0␣+␣2␣+␣...␣+␣98)␣=␣" << root(sum) << endl;
}

double root(double A)
{     // Use the babylonian algorithm to compute the square root
        double x1 = A, x2 = x1/2;
        const double tolerance = 1e-14;
        while(x2 - x1 > tolerance || x1 - x2 > tolerance)
        {
                x1 = x2;
                x2 = (x1 + A/x1)/2;
        }
        return x2;
}
```

   Output:
   `sqrt(0 + 2 + ... + 98) = 49.49747468`

   (b) Find the reciprocal of the product of the first 10 odd numbers correct to 10 significant digits. SOLN:

```
1 int main()
{
3         double product = 1;
          cout << setprecision(10);
5         for (double n = 1; n < 20; n += 2)
                  product *= n;
7         cout << "\nThe␣reciprocal␣of␣1*3*5*␣...␣*␣19␣=␣" << 1/product << endl;
}
```

Output:

The reciprocal of 1*3*5* ... * 19 = 1.527349309e-009