

This worksheet is for your use during and after lecture. It will not be collected or graded, but I think you will find it a useful tool as you learn C++ and study for the exams. Explain all false answers for the “True or False” questions; in general, show enough work and provide enough explanation so that this sheet is a useful pre-exam review. I will be happy to review your answers with you during office-hours, via Email, or instant messaging.

1. Consider the snippet of code below, what does it print out to the console?

```
1 #include <iostream>
2 using namespace std;
3
4 int left_to_right( int& x, int y, int z )
5 {
6     z = y = x;
7     return y*x;
8 }
9
10 int right_to_left( int& x, int y, int z )
11 {
12     x = y;
13     y = z;
14     return y/z;
15 }
16
17 void display( int x, int y, int z )
18 {
19     cout << "x=" << x << "\n";
20     cout << "y=" << y << "\n";
21     cout << "z=" << z << endl;
22 }
23
24 int main()
25 {
26     int x(8), y(9), z(x);
27
28     display( x, y, z );
29
30     z = left_to_right( y, x, z );
31     display( x, y, z );
32
33     y = right_to_left( z, x, y );
34     display( x, y, z );
35
36     x = left_to_right( y, x, z );
37     display( x, y, z );
38
39     return 0;
40 }
```

**Solution:**

```
x=8 y=9 z=8
x=8 y=9 z=81
x=8 y=1 z=8
x=1 y=1 z=8
```

2. Consider the source listing at the right.

(a) How many times is line 14 traversed by the CPU?

**Solution:** Line 14 is in function `g`. `g` is called once, its input from the return value of the first call to `f`, and `g`'s return value is provided as input to the *second* call of `f`. Since `g` is called once, and line 14 is not in a looping structure, line 14 is executed once by the CPU.

(b) What are the values of `B` just before line 8 is executed the first time?

- A. the value `-1`
- B. the value `0` ← **answer**
- C. the value of variable `a`
- D. “garbage”, or an unknown value

(c) How many times is line 8 traversed by the CPU?

**Solution:** Line 8 is in function `f`. As described in the solution to part a, `f` is called twice. Since line 8 is not in a looping structure, line 8 is executed twice by the CPU.

(d) What is the value of `B` printed to the console when the program ends?

- A. the value `-1`
- B. the value `0`
- C. the value `1`
- D. the value `2` ← **answer**
- E. more than 3
- F. the value of variable `a`
- G. “garbage”, or an unknown value

```

1  #include <iostream>
2  using namespace std;
3
4  int B(0);
5
6  int f( int a )
7  {
8      B = a;
9      return a;
10 }
11
12 int g( int a )
13 {
14     a += B;
15     return -a;
16 }
17
18 int main()
19 {
20     f( g( f( -1 ) ) );
21     cout << "B=" << B << endl;
22     return 0;
23 }

```

3. Match the variable storage and scope types at the left to their singular description on the right.

A auto or “local” variables (1)

B Global variable (3)

1. Variables declared on the stack *each* time a function is called.

2. Variables held in main memory that guarantee “1-cycle” manipulations by CPUs.

3. Variables available to all functions in a source file.

4. Variables representing a complex number  $a + bi$ .

The remaining worksheet questions ask you to write several different functions, plenty of additional blank pages have been provided for your work.

4. Write a function named `inSquare` that returns a Boolean value indicating whether a point (given by two double coordinates  $(x,y)$ ) is within a square with one corner at  $(a,b)$  and the opposite corner at  $(c,d)$ .  $x$ ,  $y$ ,  $a$ ,  $b$ ,  $c$ , and  $d$  should be parameters to the function.

5. Write a function named `pointDistance` that takes two points specified by  $(x,y)$  and  $(u,v)$  and returns the distance

between them as a double value.

6. Now, use function overloading to write a second `pointDistance` function that not only returns the distance between the two points but *also* sets two other double parameters ( $s, t$ ) to the midpoint between the two points.

Hints:

- You will need to use some call-by-reference variables!
- Don't re-write the logic in question 5! *Call* the previous `pointDistance` function from this one!
- The equation for the  $x$ -coordinate midpoint would be  $s = \frac{1}{2}(x + u)$ .

7. Use one of your `pointDistance` functions to write

```
bool inCircle( double h, double k, double r, double x, double y );
```

that returns true if point  $(x, y)$  is inside the circle centered at  $(h, k)$  with radius  $r$ .

Recall that we have shown you how to generate psuedo-random numbers in the labs for this course:

```
/**
 * pRNG Example
 */

#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
    // seed the random number generator with the clock
    srand( time(0) );

    cout << "A random number: " << rand() << endl;

    return 0;
}
```

8. Write a function named `outOf100` that accepts a single parameter (either double or int) called `percent`. `outOf100` should return true if

```
rand() % 100 < percent
```

This is a useful function to have in your “toolbox:” on average, it returns true `percent` times out of every 100 calls.

9. Write a function that “flips” a bias coin 10,000 times and after every 100 flips displays: the cumulative number of heads flipped, the cumulative number of tails flipped, the total number of flips so far, the longest “streak” of consecutive heads flipped, and the longest “streak” of consecutive tails so far. This function will need to take one parameter, the *bias* of the coin. A coin’s bias is its tendency to flip heads expressed as a percent or a probability. A coin with 50% bias is called a “fair coin,” and a coin with a 30% bias would flip heads about 30 out of 100 times.
10. In ultimate frisbee, it is common for two teams to decide who gets first possession by flipping *two* frisbees in the air and letting them land on the ground. One of the teams’ captains calls either “same” or “different” instead of

“heads” or “tails”. “Same” wins if both frisbees land in the same orientation (both rim up, or both rim down); “different” wins if they land in different orientations.

Suppose an Evil Captain uses a specially selected unfair disc that lands “up” 75% of the time. If the Honest Captain uses a fair disc, should she call “same” or “different” to win first possession — or does it not matter?

Write a `main()` routine that uses `outOf100()` and queries the user for the bias of the Evil Captain’s unfair disc. The program should run 10,000 simulated frisbee flips to test your answer.

## —Solutions to the Geometry Questions—

```
1  /**
2   * Solutions for worksheet II geometry problems
3   */
4
5  #include <cmath>
6  using namespace std;
7
8
9  /**
10 * returns true if point (x,y) is inside the square bounded
11 * by (a,b) and (c,d)
12 */
13 bool inSquare( double x, double y,
14               double a, double b, double c, double d )
15 {
16     /**
17      * we are not gauranteed what the relative orientation
18      * of (a,b) and (c,d) are. We can't assume (a,b) is the
19      * lower left and (c,d) is the upper right!
20      */
21     // if x is <= the smaller of a and c, the point is outside
22     if( a < c ) {
23         if( x <= a ) {
24             return false;
25         } else {
26             if( x < c ) {
27                 return false;
28             }
29         }
30     }
31
32     /**
33      * that is the long-winded way to write the logic. now I'll show you the
34      * ternary operator and how it greatly reduces the typing required. if
35      * need be, review the ternary operator from the lecture slides or the
36      * book!
37      */
38
39     // if x is >= the greater of a and c, the point is outside
40     // The a>c?a:c makes sure x is compared against the larger of a or c
41     // NOTE: we must use parens!
42     if( x >= (a>c?a:c) ) return false;
43
44     // now the same logic in the y direction
45     if( x <= (b<d?b:d) ) return false;
46     if( x >= (b>d?b:d) ) return false;
47
48     // if we are this far, the point is in the circle
49     return true;
50 }
51
52
```

```
53  /***
54  * returns distance between (x,y)->(u,v)
55  */
56  double pointDistance( double x, double y, double u, double v )
57  {
58      /***
59      * distance is the sqrt of the summed squares of the distances
60      */
61      double xdiff = x - u;
62      double ydiff = y - v;
63      // square them
64      xdiff *= xdiff;
65      ydiff *= ydiff;
66      // return sqrt of sum
67      return sqrt( xdiff + ydiff );
68  }
69
70
71  /***
72  * returns distance between (x,y)->(u,v) and the midpoint (s,t)
73  * is calculated.
74  */
75  double pointDistance( double x, double y, double u, double v,
76                      double& s, double& t )
77  {
78      // midpoint along x axis
79      s = (x+u)/2;
80      // midpoint along y axis
81      t = (y+v)/2;
82
83      // return the result of our other pointDistance function
84      return pointDistance( x, y, u, v );
85  }
86
87
88  /***
89  * return true if the point (x,y) is in the circle with radius r
90  * centered at (h,k)
91  */
92  bool inCircle( double h, double k, double r, double x, double y )
93  {
94      /***
95      * if the point is in the circle, its distance to (h,k) is
96      * less than the radius.
97      */
98      if( pointDistance( h, k, x, y ) < r ) return true;
99      return false;
100 }
```

## —Solutions to the Probability Questions—

```
1  /***
2   * Solutions to worksheet II probability questions
3   */
4
5  #include <iostream>
6  #include <cstdlib>
7  #include <ctime>
8
9  using namespace std;
10
11 /***
12  * returns true percent/100 times on average
13  */
14 bool outOf100( double percent )
15 {
16     return rand() % 100 < percent;
17 }
18
19
20 /***
21  * simulates a lot of coin tosses without output
22  * to cout. bias is the coin's bias for flipping heads,
23  * it should be between 0 and 100 inclusive.
24  */
25 void flips_managed( double bias )
26 {
27     const int TRIALS( 10000 );
28     const int OUTPUT_INTERVAL( 100 );
29     int longest_heads_streak(0), longest_tails_streak(0);
30     int current_heads_streak(0), current_tails_streak(0);
31     int heads(0);
32     bool heads_this_flip;
33
34     for( int t(0); t<TRIALS; t++ ) {
35
36         heads_this_flip = outOf100(bias);
37
38         // since a bool true is 1 and false is 0, we can do:
39         heads += heads_this_flip;
40
41         if( heads_this_flip ) {
42             // did we just finish a long streak of tails?
43             if( current_tails_streak > longest_tails_streak ) {
44                 longest_tails_streak = current_tails_streak ;
45             }
46             current_tails_streak = 0;
47             current_heads_streak++;
48         } else {
49             // did we just finish a long streak of heads?
50             if( current_heads_streak > longest_heads_streak ) {
51                 longest_heads_streak = current_heads_streak;
52             }
53             current_heads_streak = 0;
54             current_tails_streak++;
55         }
56     }
```

```
57     // output every OUTPUT_INTERVAL times — our counter is
58     // zero based, so we add one (when == 99, we've flipped 100 times)
59     if( (t+1) % OUTPUT_INTERVAL == 0 ) {
60         cout << "heads_" << heads << "_";
61         cout << "tails_" << t+1-heads << "_";
62         cout << "total_" << t+1 << "_";
63         cout << "longest_heads_" << longest_heads_streak << "_";
64         cout << "longest_tails_" << longest_tails_streak << "_";
65         cout << endl;
66     }
67 }
68 }
69
70
71 /**
72  * ultimate frisbee flip simulation
73  */
74 int main()
75 {
76     // seed the random number generator with the clock
77     srand( time(0) );
78
79     double evil_bias;
80     do {
81         cout << "Enter the evil bias of the Evil Captain's disc [0-100]: ";
82         cout << flush;
83         cin >> evil_bias;
84     } while ( evil_bias < 0 || evil_bias > 100 );
85
86     const double FAIR_BIAS( 50.0 );
87     const int FLIPS( 10000 );
88
89     int same(0), diff(0); // counters or number of same and different flips
90
91     for( int f(0); f<FLIPS; f++ ) {
92         if( outOf100(FAIR_BIAS) == outOf100(evil_bias) ) {
93             same++;
94         } else {
95             diff++;
96         }
97     }
98
99     cout << "After_" << FLIPS << "_flips, _same=" << same;
100    cout << "_and_diff=" << diff << endl;
101
102    return 0;
103 }
```

## —Solutions to the Probability Questions—

After I wrote the previous answer for `flip`, I realized that there was a redundant piece of logic that really belongs in a separate function. The first answer to `flip` isn't wrong, but I like this one more. Can you see what I have changed? And how I have used pass-by-reference to consolidate some critical counter calculations?

```

1  /**
2   * A better solution flips() from worksheet II probability questions
3   */
4
5  #include <cstdlib>
6  #include <ctime>
7  #include <iostream>
8
9  using namespace std;
10
11 /**
12  * returns true percent/100 times on average
13  */
14 bool outOf100( double percent )
15 {
16     return rand() % 100 < percent;
17 }
18
19 /**
20  * consolidated logic for managing the streak counter
21  * variables
22  *
23  * current_flipped_streak is the counter for the current flip outcome
24  * current_other_streak and
25  * longest_other_streak are the counters for the other coin face
26  */
27 void managed_streaks( int& current_flipped_streak ,
28                     int& current_other_streak , int& longest_other_streak )
29 {
30     // did we just finish a long streak for the other face?
31     if( current_other_streak > longest_other_streak ) {
32         longest_other_streak = current_other_streak;
33     }
34     current_other_streak = 0;
35     current_flipped_streak++;
36 }
37
38 /**
39  * simulates a lot of coin tosses without output
40  * to cout. bias is the coin's bias for flipping heads,
41  * it should be between 0 and 100 inclusive.
42  */
43 void managed_flips( double bias )
44 {
45     const int TRIALS( 10000 );
46     const int OUTPUT_INTERVAL( 100 );
47     int longest_heads_streak(0), longest_tails_streak(0);
48     int current_heads_streak(0), current_tails_streak(0);
49     int heads(0);
50     bool heads_this_flip;
51
52     for( int t(0); t<TRIALS; t++ ) {

```

```
53
54     heads_this_flip = outOf100(bias);
55
56     // since a bool true is 1 and false is 0, we can do:
57     heads += heads_this_flip;
58
59     // manage the streak counter logic
60     if( heads_this_flip ) {
61         managed_streaks( current_heads_streak ,
62                         current_tails_streak , longest_tails_streak );
63     } else {
64         managed_streaks( current_tails_streak ,
65                         current_heads_streak , longest_heads_streak );
66     }
67
68     // output every OUTPUT_INTERVAL times — our counter is
69     // zero based, so we add one (when == 99, we've flipped 100 times)
70     if( (t+1) % OUTPUT_INTERVAL == 0 ) {
71         cout << "heads_" << heads << "_";
72         cout << "tails_" << t+1-heads << "_";
73         cout << "total_" << t+1 << "_";
74         cout << "longest_heads_" << longest_heads_streak << "_";
75         cout << "longest_tails_" << longest_tails_streak << "_";
76         cout << endl;
77     }
78 }
79 }
80
81
82 /**
83  * test managed flips logic
84  */
85 int main()
86 {
87     // seed the random number generator with the clock
88     srand( time(0) );
89
90     managed_flips( 50.0 );
91 }
```