CS 7A - Spring 2016 - Midterm 1. 3/17/16

Write all responses on separate paper. Do not use a computer.

1. Write C++ code that declares two variables, `x` and `y`, of type `int`, initializes them to 4 and 7, respectively, then declares a third variable, `sum`, of type `int` and assigns to it the sum of `x+y` and then prints `sum` to the console.

   ANS: Note: you could also write `#include <iostream>;` and embed this in `main()`, but it's not explicitly required.

```
{ int x{4}, y{7};
  int sum = x + y;
  cout << "\nsum = " << sum;
```

2. Write C++ code that in a loop, prompts the user to `"Enter a string."` and if the string is `"Code in C"`, responds with `"Whispered words of wisdom!"`. Include line feeds (`endl or '\n'`) as needed.

```
int main() {
  string str1, str;
  int i = 0;
  cout << "\nEnter a string: ";
  while(cin >> str1) {
    ++i;
    if(str == "Code in C")
      cout << "whisperer words of wisdom";
    else {
      str += str1;
      if(i<2) str += " "; //assume string has two whitespaces
      ++i;
    }
  }
}
```

   Or, if you know about `getline()`, there's this much simpler approach:

```
int main() {
    string str;
    cout << "\nEnter a string: ";
    while(getline(cin,str))
        if(str=="Code in C.")
            cout << "\nWhisper words of wisdom.";
}
```

3. What is the output of the following code?

```
int a{3};
int b{8};
b=a++;
cout<<++b;
```

   ANS: The tricky thing here is to differentiate "post-increment" from "pre-increment", which only matters if you're doing more that one thing in a single line. In the line `b=a++;` for instance, two things happen: an assignment and an increment. Which happens first? Since this is a "post-increment" situation, the increment happens after the assignment. So the value of `a` (which is 3) is assigned to `b` and then `a` is incremented to 4. Then the line `cout<<++b;` also does two things in a single line, but this time the "pre-increment operator executes first, setting b to the value 4 before printing it to the screen.
   Note that while you can get the meaning of the increment operators reversed and still produce the same output– though two wrongs are still two wrongs.

4. What is the output of the following code?

```
int main() {
    int n = 3;
    while (n >= 0)   {
        cout << n * n << endl;
        --n;
    }
    cout << n << endl;
    while (n < 4)
        cout << ++n << endl;
    cout << n << endl;
    while (n >= 0)
        cout << (n /= 2) << endl;
    return 0;
}
```

ANS: `n` starts at 3 and while it's non-negative, its square and a line feed are printed to the console. Then when $n = -1$, we print out `n (-1)` and while it's strictly less than 4 we print out the value one larger than it, while incrementing it. When `n` reaches 4, we print out n and a line feed. Then, while n is zero or more, it prints out the result of integer division by 2 and replaces n with the result of integer division by 2 with a line feed. Thus the output is

```
9
4
1
0
-1
0
1
2
3
4
2
1
0
000000000000000....//oo loop
```

5. What is the output of the following code?

```
#include <iostream>
int main(){
    int n;
    cout << (n = 4) << endl;
    cout << (n == 4) << endl;
    cout << (n > 3)  << endl;
    cout << (n < 4)  << endl;
    cout << (n = 0)  << endl;
    cout << (n == 0) << endl;
    cout << (n > 0)  << endl;
    cout << (n && 4) << endl;
    cout << (n || 4) << endl;
    cout << (!n)     << endl;
    return 0;
}
```

ans: Here are the outputs with commentary

```
4  //the value assigned to the variable n
1  //true, that
1  //n is bigger than 3, since n is 4
0  //false, 4 is not less than itself
0  //n is now 0
1  //true n is zero
0  //false, n is not greater than zero
0  //n is 0, so the first condition is false, making the && false
1  //false or true is true
1  //not false is true
```

6. Each of the following programs has error(s). Locate the error(s), classify each error as either a syntax error, a logical error, an overflow error, an underflow error, or a narrowing error.:

(a)
```
int average(int a, int b) {
    return a + b / 2;
}
```

ANS: The formula for computing the average is wrong, a logical error

(b)
```
int main() {
   vector<int> iVector(10);
   myvector[10]=5;
}
```
No such variable as "`myvector`". Syntax error.

(c)
```
int  main(void) {
    int x{7777.7};
    char c2{7777};
    cout << c2 << " == " << char(x%256);
}
```
ANS: These are narrowing errors. The decimal part of 7777.7 will be ignored when it is assigned to `x`, and a `char` can only take on values 0 through 255. What will be assigned to c2 is the remainder when 7777 is divided by 256, and since 7777 = 30*256 + 97, so we'll see the correct statement "a == a".

(d)
```
int  foo(int x) {
    return(x+1);
}

int  main() {
    cout << foo(INT_MAX);
}
```
This will cause an overflow error. `INT_MAX` is the largest `int` value the compiler is designed to handle, so if you add 1 to it, you'll get a wrap around effect and the output will be `-INT_MAX`.

(e)
```
int  main(void) {
    unsigned int a{2};
    unsigned int b{3};
    a -=b;
}
```
ANS: Since $b == 3$, the statement `a -= b;` assigns $2 - 3 = -1$ to `a`, which is of type `unsigned int`, thus causing a type error. Because of wrap-around, $a = 4294967295 = 2^{32} - 1$.

7. Explain what the following program does:

```
1  void foo(int a, int& b) {
2    a*=2;
3    cout << "\na = " << a;
4    b*=2;
5  }
6
7  int main() {
8    int x=1, y=3;
9    foo(x, y);
10   cout << "x=" << x << ", y=" << y;
11 }
```

ANS: Variables `x` and `y` are defined to be `int`s with initial values 1 and 3, respectively. These variables are then passed to the function `foo()`–but while `x` is passed by value and renamed `a`, `y` is passed by reference and given the local name b. So a copy of `a` is doubled, and this doubled value is printed to the console as "`a = 2`", but the actual contents of the memory location of `y` is doubled, so that when control is returned to `main()` `x` is still 1, but `y` is now 6, and, all together, the output to the console is

```
a = 2
x = 1, y = 6
```

8. What is the output when the following code fragment is executed?

```
int found = 0, count = 5;
if (!found || --count == 0)
    cout << "danger" << endl;
cout << "count = " << count << endl;
```

ANS: Since `!found` will be `true`, the second condition is not even checked, which means the decrement to `count` doesn't happen. Thus the output is

```
danger
count = 5
```

9. What is the output when the following code fragment is executed?

```
char ch;
string title = "Titanic";
ch = title[1];
title[3] = ch;
cout << title << endl;
cout << ch << endl;
```

The variable `ch` gets the value 'i' and this is assigned to the fourth character of the string `title`, so the output is

```
Titinic
i
```

10. Write a program that implements the Babylonian algorithm. Use the flow chart at right as a guide, if you like. Remember to use proper syntax and style and to define your variables.
ANS: I dunno, this is how I remember it:

```
int main() {
  double A, x, temp;
  cout << "\nEnter a value to see its square root: ";
  cin >> A;
  temp = A/2;
  x = (A/temp + temp)/2;
  double toler = A*1e-15;
  while(x - temp > toler || temp - x > toler) { // one difference will be neg
      temp = x;
      x = (A/x + x)/2;
  }
  cout << "\nA square root of " << A << " is " << x;
}
```

11. Write a C++ program to implement the following pseudo-code for Euclid method of finding the greatest common divisor of `m` and `n`.

```
if m < n, swap(m,n)
while n does not equal 0
   r = m modulo n
   m = n
   n = r
endwhile
output m
```

ANS:

```
int gcd(int m, int n) {
    int temp, r;
    if(m<n) {  //swap
      temp = m;
      m = n;
      n = temp;
    }
    while(n != 0) {
        r = m%n;
        m = n;
        n = r;
    }
    return m;
}
```

12. What is the output when the following code fragment is executed?

```
int i = 5, j = 6, k = 7, n = 3;
cout << i + j * k - k % n << endl;
cout << i / n << endl;
```

ANS: The order of operations say the * and % operations take precedence (from left to right) so `i + j * k - k % n = 5 + 6 * 7 - 7 % 3 = 5 + 42 - 1` and the +/- operations are then performed left to right: `5 + 42 - 1 = 46`. Then `5 / 3 = 1`, since this is integer division. Thus the output will be

```
46
1
```

13. The nested conditional statement shown below has been written by an inexperienced C++ programmer. The behavior of the statement is not correctly represented by the formatting.

```
if (n < 10)
    if (n > 0)
        cout << "The number is positive." << endl;
    else    cout << "The number is ____________." << endl;
```

(a) What is the output of the statement if the variable `n` has the value 7 ? If `n` has the value 15 ? If `n` has the value `3` ?
ANS: If `n = 7` then the output is `The number is positive`
If `n = 15` then there is no output. This is because the `else` binds with the nearest `if`. But 15 is positive, so why not say so?
If `n = -3` then the output is
`The number is _____.`

(b) Correct the *syntax* of the statement so that the *logic* of the corrected statement corresponds to the formatting of the original statement. Also, replace the blank with an appropriate word or phrase.
ANS: The way the indentation goes, it's clear the coder intended the `else` to go with the second `if`.

(c) Correct the *formatting style* of the (original) statement so that the new format reflects the logical behavior of the original statement. Also, replace the blank with an appropriate word or phrase.

14. The loop shown below has been written by an inexperienced C++ programmer. The behavior of the loop is not correctly represented by the formatting style.

```
int n = 10;
while (n > 0)
    n /= 2;
    cout << n * n << endl;
```

(a) What is the output of the loop as it is written?
ANS:
0

(b) Correct the syntax of the loop so that the logic of the corrected loop corresponds to the formatting of the original loop. What is the output of the corrected loop?
ANS: The body of the while loop is clearly indented so as to indicate and intention that both lines are part of the while loop. But the curly braces are missing, so it should look like this:

```
int n = 10;
while (n > 0) {
    n /= 2;
    cout << n * n << endl;
}
```

25
4
1

(c) Correct the formatting of the (original) loop so that the new format reflects the logical behavior of the original loop.

```
int n = 10;
while (n > 0)
    n /= 2;
cout << n * n << endl;
```

15. Remove all the unnecessary tests from the nested conditional statement below.

```
float income;
cout << "Enter your monthly income: ";
cin  >> income;
if (income < 0.0)
    cout << "You are going farther into debt every month." << endl;
else if (income >= 0.0  &&  income < 1200.00)
    cout << "You are living below the poverty line." << endl;
else if (income >= 1200.00  &&  income < 2500.00)
    cout << "You are living in moderate comfort." << endl;
else if (income >= 2500.00)
    cout << "You are well off." << endl;
```

ANS:

```
float income;
cout << "Enter your monthly income: ";
cin  >> income;
if (income < 0.0)
    cout << "You are going farther into debt every month." << endl;
else if (income < 1200.00)
    cout << "You are living below the poverty line." << endl;
else if (income < 2500.00)
    cout << "You are living in moderate comfort." << endl;
else cout << "You are well off." << endl;
```

16. Answer the questions below concerning the following fragment of code.

```
int n;
cout << "Enter an integer: ";
cin  >> n;
```

```
if (n < 10)
    cout << "less than 10" << endl;
else if (n > 5)
    cout << "greater than 5" << endl;
else    cout << "not interesting" << endl;
```

(a) What will be the output of the fragment above if the interactive user enters the integer value 0 ?
ANS: `less than 10`

(b) What will be the output of the fragment above if the interactive user enters the integer value 15 ?
ANS: `greater than 5`

(c) What will be the output of the fragment above if the interactive user enters the integer value 7 ?
ANS: `less than 10`

(d) What values for `n` will cause the output of the fragment above to be "not interesting"?
ANS: That's just not possible!

17. Rewrite the following code fragment so that it uses a "do...while..." loop to accomplish the same task.

```
int n;
cout << "Enter a non-negative integer: ";
cin  >> n;
while (n < 0) {
    cout << "The integer you entered is negative." << endl;
    cout << "Enter a non-negative integer: ";
    cin  >> n;
}
```

ANS:

```
int n;
cout << "Enter a non-negative integer: ";
cin  >> n;
if(n<0)
    do {
        cout << "The integer you entered is negative." << endl;
        cout << "Enter a non-negative integer: ";
        cin  >> n;
    } while (n < 0);
```

18. In the code fragment below, the programmer has almost certainly made an error in the first line of the conditional statement.

(a) What is the output of this code fragment as it is written?
ANS:

```
n is zero
The square of n is 0.
```

(b) How can it be corrected to do what is the programmer surely intended?
ANS: Simply change `if (n = 0)` to `if (n == 0)`.

```
int n = 5;
if (n = 0)   // NOTE THE OPERATOR!!!
    cout << "n is zero" << ".\n";
else
    cout << "n is not zero" << ".\n";
cout << "The square of n is " << n * n << ".\n";
```

19. What is the output when the following code fragment is executed?

```
int n, k = 5;
n = (100 % k ? k + 1 : k - 1);
cout << "n = " << n << "    k = " << k << endl;
```

ANS: `n = 4 k = 5`

20. What is the output when the following code fragment is executed?

```
int    n;
double x = 3.8;
n = int(x);
cout << "n = " << n << endl;
```

ANS: `n = 3`

21. What is the output when the following code fragment is executed? Rewrite the fragment to obtain an equivalent code fragment in which the body of the loop is a simple statement instead of a compound statement.

```
int i = 5;
while (i > 0) {
    --i;
    cout << i << endl;
}
```

ANS:

```
4
3
2
1
0
```

```
int i = 5;
while (i > 0) {
    cout << --i << endl;
}
```

22. The following loop is an endless loop: when executed it will never terminate. What modification can be made in the code to produce the desired output?

```
cout << "Here's a list of the ASCII values of all the upper"
     << " case letters.\n";
char letter = 'A';
while (letter <= 'Z')
    cout << letter << "  " << int(letter) << endl;
```

ANS: Change `int(letter)` to `int(letter++)`

23. Write a function named "sum_from_to" that takes two integer arguments, call them "first" and "last", and returns as its value the sum of all the integers between first and last inclusive. Thus, for example,

```
cout << sum_from_to(4,7) << endl;  // will print  22  because 4+5+6+7 = 22
cout << sum_from_to(-3,1) << endl; // will print  5  'cause (-3)+(-2)+(-1)+0+1 = 5
cout << sum_from_to(7,4) << endl;  // will print  22  because 7+6+5+4 = 22 c
out << sum_from_to(9,9) << endl;  // will print  9
```

```
int sum_from_to(int a, int b) {
    int sum{0};
    if(a > b) {
      while(b <= a) {
        sum += b;
        ++b;
      }
      return sum;
    else {
        while(a <= b) {
            sum += a;
            ++a;
        }
        return a;
    }
}
```

24. Write a function named "enough" that takes one integer argument, call it "goal" and returns as its value the smallest positive integer `n` for which $1 + 2 + 3 + \cdots + n$ is at least equal to `goal` . Thus, for example,

```
cout << enough(9) << endl; // will print  4  because  1+2+3+4>9 but 1+2+3<9
cout << enough(21) << endl;// will print  6  because 1+2+...+6>21 but 1+2+...5<21
cout << enough(-7) << endl;// will print  1  because  1>  7 and 1 is the smallest
                           // positive integer
cout << enough(1) << endl; // will print  1  because 1=1 and 1 is the smallest
                           // positive integer
```

ANS:

```
int enough(int x) {
    int start = 1, sum = 1;
    while(sum < x) {
        ++start;
        sum += start;
    }
    return start;
}
```

25. Write a function named "reduce" that takes two positive integer arguments, call them "num" and "denom", treats them as the numerator and denominator of a fraction, and reduces the fraction. That is to say, each of the two arguments will be modified by dividing it by the greatest common divisor of the two integers. The function should return the value `0` (to indicate failure to reduce) if either of the two arguments is zero or negative, and should return the value `1` otherwise. Thus, for example, if `m` and `n` have been declared to be integer variables in a program, then

```
m = 25;
n = 15;
if (reduce(m,n))
    cout << m << '/' << n << endl;
else
    cout << "fraction error" << endl;
```

will produce the following output:

```
5/3
```

Note that the values of `m` and `n` were modified by the function call. Similarly,

```
m = 63;
n = 210;
if (reduce(m,n))
    cout << m << '/' << n << endl;
else
    cout << "fraction error" << endl;
```

will produce the following output:

```
3/10
```

Here is another example.

```
m = 25;
n = 0;
if (reduce(m,n))
    cout << m << '/' << n << endl;
else
    cout << "fraction error" << endl;
```

will produce the following output:

```
fraction error
```

The function `reduce` is allowed to make calls to other functions that you have written.
ANS: The tricky part here is to realize that the variables that are passed must be changed by the function and those changes have to stick after the function returns. This means that the variables must be passed by reference:

```
bool reduce(int &a, int &b) {
  if(a <= 0 || b <= 0) return 0;
  else {
    a /=gcd(a,b);
    b /=gcd(a,b);
    return 1;
  }
}
```