

## CSCI1320 – Test #1 Answers

You will have until the end of the normal class period (50 minutes if you start on time) to finish this test. It consists of 10 equally weighted questions and an extra credit question. It is possible to receive partial credit on all the problems, but only if you write something down.

1. Why does pass-by-reference exist in C++? Give an example of a function where you would want to use it.

Pass by reference allows you to change the value of an actual argument by changing the formal argument inside a function. This is helpful in a number of situations in C++. The most notable is when a function needs to “return” multiple values. Since functions can only return one thing this allows you to pass multiple entities as arguments and change their values so that they are effectively returned to the calling program. Many of you used this in assignment #2 for counting up the number of students with different letter grades.

2. Trace the following code and write the output that it produces. Use the table provided for partial credit.

```
int main(void) {
    cout << recursiveFunc(5) << endl;
    cout << recursiveFunc(20) << endl;
}

int recursiveFunc(int argument) {
    if(argument==1) return 1;
    if(argument%2==0) { // number is even
        return recursiveFunc(argument/2)+1;
    } else {
        return recursiveFunc(3*argument+1)+1;
    }
}
```

Write output here:

6  
8

Argument value for recursiveFunc call	Scratch work for call	Value returned from call
5	You could put anything here.	6
16		5
8		4
4		3
2		2
1		1
20		8
10		7
5		6
16		5
8		4
4		3
2		2
1		1

3. Write a function that takes an integer argument and returns the product of all the positive even numbers up to that number. (So if you pass in 7 it returns 2\*4\*6=48.)

```
int prod(int n) {
    int ret=1;
    for(int i=2; i<=n; i+=2) ret*=i;
    return ret;
}
```

4. Write a function that squares all the elements of an array of doubles that is passed in as an argument. (So if the array [2.0, 3.0] were passed in it would be [4.0, 9.0] when it returned.)

```
void square(double a[],int len) {
    for(int j=0; j<len; j++) {
        a[j]*=a[j];
    }
}
```

5. Trace the following code and write the output that it produces. Use the table provided for partial credit. Show the steps in your trace for partial credit. Label columns with variable names.

```
void foo(int &n1,int &n2) {
    if(n1/n2<3) {
        n1+=2;
    } else {
        n2++;
    }
}

int bar(int a,int b,int c) {
    int ret=1;
    for(int j=a; j<b; j+=c) {
        foo(j,ret);
    }
    return ret;
}

int main() {
    cout << bar(1,10,1) << endl;
    cout << bar(3,18,2) << endl;
}
```

Write output here:

4

6

Call 1	j	ret	Call 2	j	ret
	1	1		3	1
	4	1		5	2
	5	2		9	2
	8	2		11	3
	9	3		13	4
	10	4		15	5
				17	6
				19	6

This problem proved to be quite challenging as it exhibits non-normal behaviors (like changing the loop variable in a place other than the iterator). Some people were also thrown by the fact that the work for the loop happens in a function call. This can be simplified if you notice that the function takes both arguments by reference. Because of this, the code in foo can basically be pasted right into the loop in bar with minor variable name adjustments so you get this.

```
int bar(int a,int b,int c) {
    int ret=1;
    for(int j=a; j<b; j+=c) {
        if(j/ret<3) {
            j+=2;
        } else {
            ret++;
        }
    }
    return ret;
}
```

While this doesn't actually change anything in what the code does, it might make it easier for you to read and to trace.

6. Write Boolean conditional expressions to perform each of the following checks. (2 points each)  
a. Given airTempC as the ambient temperature in Celsius, will water be in a liquid form there at 1 atm of pressure (0 is the freezing point and 100 is the boiling point)?

```
(airTempC>0) && (airTempC<100)
```

b. Test if a, b, or their sum is zero.

```
(a==0) || (b==0) || (a+b==0)
```

c. Test if the double grade is a passing grade.

```
grade>=60
```

d. Given a and b are doubles, test if a is outside the range 0..100 and b is outside the range 70..100.

```
((a<0) || (a>100)) && ((b<70) || (b>100))
```

e. Test if a is evenly divisible by 2 and 3, but not 5 (remember that if it is evenly divisible the remainder of division is zero).

```
(a%2==0) && (a%3==0) && (a%5!=0)
```

This can be simplified to

```
(a%6==0) && (a%5!=0)
```

7. Explain what a class is and what advantages they provide for us in programming. This is a broad and quite open question. I've mentioned a few things in class from simple ways they could have helped you on a previous assignment to more complex reasons for using them. The description of what it is gives you half the points. Different advantages of using them can get you the other half.

A class is a collection of both data and functions under a single, complex type. The first, and simplest advantage of a class is that it allows us to group things together so that objects that have multiple pieces of data in them and group the functionality for that data as well. This means you don't have to do things like have multiple parallel arrays the way you did in assignment #3. More important than this though is the fact that classes allow you to hide what the class is actually doing from the rest of the world. This matters for two reasons. The simpler reason is that you can encapsulate the data in the class from the rest of the program. This means that you can make it so that the data is safe from other sections of code so those pieces of code don't screw up the data. In addition, it allows you to separate the interface of the class from the actual implementation. In some ways this protects the rest of the program from changes you might want to make in your class. Because the outside program only sees the public interface, it also only uses the public interface. You can change anything inside the class you want as long as you don't change the public interface or how it behaves, and your program will still work.

8. Trace the sorting of the following array using the minSort algorithm. I've put in minSort below in case you want to trace the actual code. [5, 8, 3, 11, 6, 2, 9]

```
void minSort(int a[],int len) {  
    for(int i=0; i<len-1; i++) {  
        int min=i;  
        for(j=i+1; j<len; j++) {  
            if(a[j]<a[min]) min=j;  
        }  
        if(min!=i) {  
            int tmp=a[i];  
            a[i]=a[min];  
            a[min]=tmp;  
        }  
    }  
}
```

Answer:

```
[5,8,3,11,6,2,9]
```

```
[2,8,3,11,6,5,9]
```

```
[2,3,8,11,6,5,9]
```

```
[2,3,5,11,6,8,9]
```

```
[2,3,5,6,11,8,9]
```

```
[2,3,5,6,8,11,9]
```

```
[2,3,5,6,8,9,11]
```

The key to remember here is that min sort builds a sorted array from one end by repeatedly moving the minimum value into the farthest over location. It moves the minimum into place using a swap, not a shift. A shift is much slower than a swap and should only be used when required like in insertion sort.

9. We discussed in class how you could do a search on sorted data more quickly than you can on unsorted data. Describe how you can do this and why it works.

The basic idea here is that when the data is sorted you can ignore parts of it. The most common way of doing this is through a binary search. In this search you look at the middle value and compare it to what you are looking for. If it is greater you know the value is in the lower half, otherwise it is in the upper half. You can repeat this for half you know it is in. In this process you are constantly throwing out half of the possibilities that are left without having to take the time to check them. As such it take logarithmic time to find something instead of linear time.

10. Write a loop that will print out the first 30 powers of 2, beginning with 1 ( $=2^0$ ).

```
for(int i=0, j=1; i<30; i++, j*=2) {
    cout << j << endl;
}
```

I should have specified that you should do this without the pow function because that made the problem too easy. The trick here is that you need two loop variables, one to count how many things you have printed and the other to calculate the powers of two you are going to print.

Extra Credit: Below are two functions to calculate Fibonacci numbers. You have seen both of these before. One is a direct application of the recursive definition of Fibonacci numbers and the other uses a loop and stores the last two values in variables. Following the functions is a table. Fill in that table with the number of additions and subtractions each of these functions will have to perform to calculate the given Fibonacci number. Use what this table shows you and your intuition of these functions to then tell me the order of each.

```
int recurFib(int n) {
    if(n<3) return(1);
    return recurFib(n-1)+recurFib(n-2);
}
int loopFib(int n) {
    int prev=1, cur=1;
    for(int i=2; i<n; i++) {
        int tmp=prev+cur;
        prev=cur;
        cur=tmp;
    }
    return cur;
}
```

N	RecurFib(n)	loopFib(n)
4	4	4
8	60	12
16	2958	28

Order of recurFib:  $O(2^n)$

Order of loopFib:  $O(n)$

For both functions you should pay attention to the fact that they don't do any math when the input value is less than 3. Also, keep in mind ALL the additions it is doing. That  $i++$  in the loop is an addition. Lastly, for the recursive call the basic formula is  $f(n)=3+f(n-1)+f(n-2)$ . Note that is almost like the Fibonacci numbers themselves except for that +3 in there. That makes a really big difference though. It is a plus three because you have  $n-1$ ,  $n-2$ , and the sum of the two recursive calls.