WYSE - Academic Challenge
Computer Science Test (State) - 2019

1. **DLNS** In Two's Complement, the 'decimal' value of each bit is the same as with ordinary binary numbers (i.e., each bit is equal to 2^i, where i is the i-th zero-based bit from the LSB position. The only difference is that the MSB is $-1 \cdot 2^n$. So for a 10 bit binary value, we have: $-1 \cdot 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$. To get the most negative value, we simply take: $-1 \cdot 2^9 = -512$. For the upper bound, we just set all the other bits to 1, and the MSB to 0: $2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 = 511$

2. **DSA** Stable sorting algorithms maintain the relative position of equal elements. Heapsort is not a stable sorting algorithm as it does not guarantee the resulting list will maintain the relative position of equal elements. Heapsort and Mergesort are sorting algorithms, not searching algorithms. Mergesort can be implemented as a stable sorting algorithm. Heapsort has a worst-case Big-Oh time complexity of $O(n \log n)$.

3. **OOP** When a class is instantiated, the base class constructor is always called (along with any of that class's base classes), followed by the constructor of the class being directly instantiated. The fact that `Tree` is privately inherited by `Pine` has no bearing on what constructors are called.

4. **DSA** An algorithm is stable if it maintains the relative position of two elements which are considered equal. The time complexity of a sorting algorithm can be used to describe how efficient an algorithm can sort an input list.

5. **DLNS** We first need to find how long it takes for a bit to travel from the router to the client. We know that the signal propogates at roughly the speed of light, which is $3.0 \times 10^8 \text{m/s}$, and the bit has to travel 10m. So using the formula $d = r \cdot t$, and plugging in known values for d and r, we get: $10\text{m} = 3.0 \times 10^8 \text{m/s} \cdot t$. Rearranging, we get: $t = \frac{10\text{m}}{3.0 \times 10^8 \text{m/s}} = (10/3) \times 10^{-8}\text{s}$.

   Next, we can calculate how many bits are transmitted in that time, if the router is transmitting at a rate of 54mbps. We have: $54 \times 10^6 \text{bps} \cdot (10/3) \times 10^{-8}\text{s} = (540/3) \times 10^{-2}\text{bits}$.

6. **GEN** A routing table is used to determine where to send a packet. In this case, we have two different interfaces: wlan0 and br-lan. Since there isn't an entry in the routing table that has a destination and mask that contains the address 10.17.0.5, we would use the default destination entry, which points to 10.36.0.1 on interface wlan0. In other words, the IP address we are sending data to isn't in our network, so we send it to our gateway.

7. **OOP** Template classes and template functions allow easy re-use of code when we need functions or classes to work on different data types. Conceptually, a list is a good example of this: the basic operations that a list would need (add, remove, get) are agnostic to the specific data type that each item in the list is. In other words, the code to retrieve an item from a list should be basically the same, regardless of whether the items in the list are integers, strings, or some other data type. This is an example of Polymorphism. The Standard Template Library (STL) is an example of a library that is made up of many template classes. Template classes can be used with inheritance. It is possible to create a class which inherits from a template class, for example.

8. **GEN** The return value for `int main()` is used as the exit code for the program. This is commonly used by the operating system to determine if the program exited successfully or encountered an error.

9. **PL** The `setw` function, which is available through the `iomanip` library, pads the next value to be printed with the number of spaces specified in the function call. The padding is done on the left.

10. **PL** Here is a trace of the code:

a[0]	a[1]	a[2]	i	j	comment
			0		
1, ?, ?	?, ?, ?	?, ?, ?	0		
1, ?, ?	?, ?, ?	?, ?, ?	1		
1, 1, ?	1, ?, ?	?, ?, ?	1		
1, 1, ?	1, ?, ?	?, ?, ?	2		
1, 1, 1	1, ?, ?	1, ?, ?	2		
1, 1, 1	1, ?, ?	1, ?, ?	1		
1, 1, 1	1, ?, ?	1, ?, ?	1	1	
1, 1, 1	1, 2, ?	1, ?, ?	1	1	
1, 1, 1	1, 2, ?	1, ?, ?	1	2	
1, 1, 1	1, 2, 3	1, ?, ?	1	2	
1, 1, 1	1, 2, 3	1, ?, ?	2	2	
1, 1, 1	1, 2, 3	1, ?, ?	2	1	
1, 1, 1	1, 2, 3	1, 3, ?	2	1	
1, 1, 1	1, 2, 3	1, 3, ?	2	2	
1, 1, 1	1, 2, 3	1, 3, 6	2	2	

11. **PL** Since arrays store data in sequential memory locations, and since an array variable points to the first element in the array, we can use pointers to access specific elements in the array.

12. **DLNS** Since the value starts with a 1, we know it is negative. To convert it to its positive binary representation, we invert the bits and add 1:
`11011100` (Two's complement)
`00100011` (Invert)
`00100100` (Add 1)

    To convert to One's Complement Notation, we simply invert:
`00100100`
`11011011` (Invert)

13. **PL** The expression `a % b` evaluates to zero if `a` is evenly divisible by `b`. If that expression evaluates to zero, it is considered a false value, which means it will execute the `else` clause on the `if` statement.

14. **PL** Since the first parameter to `fun1` is passed by reference, it cannot be a literal.

15. **PL** Here is an overview of the function calls:

Code	a	b	Comment
call fun1(a, 2)	3	2	Initial function call
a % b	3	2	Check if condition, a % b = 3, which is true
decrement a	2	2	
call fun1(a, b)	2	2	
a % b	2	2	Check if condition, a % b = 0, which is false
call fun2(a, b)	2	2	
a = a - b	0	2	
return a	0	2	fun2 returns

16. **PL** Please see the previous code trace.

17. **PL** A `for`-loop has three parts in the header: initialization, conditional, and increment/decrement. All three are optional, and in this case, the initialization is omitted. The order of execution is as follows:

    1. Execute initialization
    2. Check conditional is true
    3. Execute body of `for`-loop
    4. Execute increment/decrement
    5. Check conditional is true
    6. etc...

    To simplify this, we can look solely at the conditional. Translated to English, the conditional states: "If the variable `i` is not zero, continue looping. Decrement `i`." Since we are using a postfix decrement, the decrement happens after the `i` is checked to be true. Therefore, the result will be `-1`.

18. **PL** Here is a trace of the code:

i	comment
3	initialization
2	Check condition on if with i = 3, post-decrement
1	Decrement i inside of body of for loop
0	Decrement i inside of the for-loop header
-1	Check condition on if with i = 0, post-decrement

19. **PL** The for loop, in this case, has no body (due to the semicolon after the `for`-loop header). Here is a trace of the code:

i	comment
3	initialization
2	pre-decrement i, check condition on if, with i = 2
1	Decrement i inside of the for-loop header
0	pre-decrement i, check condition on if, with i = 0, exit

20. **PL** The code `a /= b` is equivalent to `a = a / b`. The left and right shift operators divide and multiply by a power of 2. In the case of `b >> 2`, the result is b divided by 2^2. The bitwise AND operator (`&`) performs a logical ANDing of corresponding bits from the two operands.

21. **GEN** When using Public-Key Encryption, two keys are used: a public key and a private key. Data that is encrypted with the public key can only be decrypted with the corresponding

private key, and vise versa. Due to this fact, supposing Bob encrypted data with their private key. Alice could use Bob's public key to decrypt the data, and would also know that since the data could be decrypted, it must have originated from Bob (authentication). Similarly, Alice could use Bob's public key to encrypt data, and then only Bob could decrypt it using their private key (Encryption).

22. **DSA** A min-heap is a complete binary tree. A binary tree is a tree in which the degree of any node is 0, 1, or 2. A min-heap is also a tree in which the data at any node is less than or equal to the data in any child nodes. In other words, the root node will contain the smallest value in the heap.

23. **GEN** In C++, it is possible to define a template struct, just as it is possible to define a template class or function. Here is an example:

```
1  template <class T>
2  struct linkedListItem
3  {
4    linkedListItem *next;
5    T data;
6  };
```

24. **OOP** It is possible for a class to have an Is-A relationshiop and a Has-A relationship with a base class. As an example, consider the following:

```
1  class Person
2  {
3    // details go here
4  };
5
6  class GamePlayer : public Person
7  {
8    private:
9      Person mentor;
10
11   // details go here
12 };
```

25. **DSA** When expressing the time complexity of a function using Big-Oh notation, we concern ourselves only with the fastest growing portion of the function. In this case, n^4 grows the fastest.

26. **DSA** A binary tree is a tree in which each node has a left subtree and a right subtree. A max heap is also a binary tree. A linked list could be created using the struct where the `left` and `right` properties point to the previous and next nodes in the list.

27. **DLNS** A single Hexadecimal digit requires up to 4 binary bits to represent. Hexadecimal, since it is base-16, requires 16 symbols to represent each possible digit: 0-9, A-F.

28. **OOP** Since `GrayscaleIcon` and `Icon` don't inherit from eachother, the `SetColorAt` on each class isn't directly related. They both implement the pure virtual `IconBase::GetColorAt` function.

29. **PL** The function returns the average of the red, green, and blue components of the color by adding all three values together, then dividing by 3.

30. **PL** If `data` were declared as `const` with no other modifications, a compile-time exception would be thrown. This is because there are methods which assign elements of that array. Declaring it as const would prevent us from being able to make legitimate updates to it.