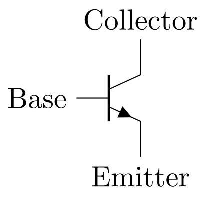
# Academic Challenge
# Computer Science Test (Regional) - 2019

1. **GEN** Google Cloud, Amazon Web Services, Microsoft Azure, and IBM Cloud are all public cloud providers. Github is an online source control provider.

2. **DLNS** The circuit forms a NOT gate. When A is high, the transistor allows current to flow from the collector to the emitter poles (see below for pole labeling). When this happens, B is effectively grounded. The inverse is true when A is low; the path between the collector and emitter resists current resulting in B being high.



3. **GEN** HTTP is a protocol by which documents can be accessed over the Internet. Documents can also be accessed by other protocols, such as FTP. Email is sent between mail hosts most commonly with SMTP (Simple Mail Transfer Protocol). There are several printing protocols, one of which is IPP (Internet Printing Protocol). HTML (HyperText Markup Language) is used most commonly as a means to create web pages viewable by web browsers.

4. **PL** Tracing the code yields:

| $i$ | output | comment |
|---|---|---|
| ? | ? | |
| 0 | ? | Initialize and check $i < 5$, executes loop |
| 0 | 0 | Prints $i$ |
| 1 | 0 | Increment and check $i < 5$, executes loop |
| 1 | 01 | Prints $i$ |
| 2 | 01 | Increments and check $i < 5$, executes loop |
| 2 | 012 | Prints $i$ |
| 3 | 012 | Increments and check $i < 5$, executes loop |
| 3 | 0123 | Prints $i$ |
| 4 | 0123 | Increments and check $i < 5$, executes loop |
| 4 | 01234 | Prints $i$ |
| 5 | 01234 | Increments and check $i < 5$, exits for-loop since $5 \not< 5$ |
| 5 | 012345 | Prints $i$ (outside of for-loop) |

Therefore, the output would be 012345.

5. **PL** Using the trace from the previous solution, the conditional is executed a total of 6 times.

6. **GEN** RISC CPUs generally require more instructions to achieve the same result as one instruction in a CISC CPU. An ALU is one part of a CPU and is used to perform arithmetic and logic operations. In general, the RISC architecture is used more prevalently today, and the CISC architecture used more prevalently in the earlier days of computers.

7. **GEN** A kernel is the core of an operating system, but is not an example of an operating system. Microsoft Windows, Ubuntu Linux, and macOS are all examples of modern operating systems.

8. **DSA** To find the Post-Order Traversal of a tree, recursively visit the left subtree, then the right subtree, then the current node. For leaf nodes (nodes without any subtrees), simply visit that node.

9. **DSA** The tree is a Binary Tree. By definition, a Binary Tree is a tree in which the degree of every node is either 0, 1, or 2. All trees can also be classified as graphs (though not all graphs are trees). A tree is defined as a graph with no cycles. Node A is the only root node in the tree; a root node has no parent. The degree of a node is the number of subtrees of a node.

10. **DLNS** Converting the binary value to decimal can be calculated using this formula:

$$\begin{aligned}01000111_2 &= 0\cdot 2^7 + 1\cdot 2^6 + 0\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0\\ &= 64+4+2+1\\ &= 71\end{aligned}$$

11. **PL** Tracing the code for the first function call yields:

| $a$[] | $b$ | $c$ | comment |
|---|---|---|---|
| {1, 2, 3, 4, 5} | 5 | 0 | |
| | 4 | 0 | $b \neq 0$, decrement $b$ |
| | 4 | 5 | $c = c + 5$ |
| | 3 | 5 | $b \neq 0$, decrement $b$ |
| | 3 | 9 | $c = c + 4$ |
| | 2 | 9 | $b \neq 0$, decrement $b$ |
| | 2 | 12 | $c = c + 3$ |
| | 1 | 12 | $b \neq 0$, decrement $b$ |
| | 1 | 14 | $c = c + 2$ |
| | 0 | 14 | $b \neq 0$, decrement $b$ |
| | 0 | 15 | $c = c + 1$ |
| | 0 | 15 | $b = 0$, exit `while` loop, return $c = 15$ |

Tracing the code for the second function call yields:

| $a$[] | $b$ | $c$ | comment |
|---|---|---|---|
| { -1, 0, 1, 2} | 4 | 0 | |
| | 3 | 0 | $b \neq 0$, decrement $b$ |
| | 3 | 2 | $c = c + 2$ |
| | 2 | 2 | $b \neq 0$, decrement $b$ |
| | 2 | 3 | $c = c + 1$ |
| | 1 | 3 | $b \neq 0$, decrement $b$ |
| | 1 | 3 | $c = c + 0$ |
| | 0 | 3 | $b \neq 0$, decrement $b$ |
| | 0 | 2 | $c = c + (-1)$ |
| | 0 | 2 | $b = 0$, exit `while` loop, return $c = 2$ |

12. **PL** Declaring a variable within a function as `static` means that the value of that variable is maintained across function calls. As a result, we just need to trace the second function call, `fun1(b, 4)`, as the first function's trace wouldn't change.

| `a[]` | $b$ | $c$ | comment |
|---|---|---|---|
| { -1, 0, 1, 2} | 4 | 15 | |
| | 3 | 15 | $b \neq 0$, decrement $b$ |
| | 3 | 17 | $c = c + 2$ |
| | 2 | 17 | $b \neq 0$, decrement $b$ |
| | 2 | 18 | $c = c + 1$ |
| | 1 | 18 | $b \neq 0$, decrement $b$ |
| | 1 | 18 | $c = c + 0$ |
| | 0 | 18 | $b \neq 0$, decrement $b$ |
| | 0 | 17 | $c = c + (-1)$ |
| | 0 | 17 | $b = 0$, exit `while` loop, return $c = 17$ |

13. **PL** A trace of the commands yields:

| list | operation | comment |
|---|---|---|
| 2 | `PUSH 2` | |
| 2, 4 | `PUSH 4` | |
| 2, 4, 8 | `PUSH 8` | |
| 2, 4, 8, 3 | `PUSH 3` | |
| 8, 3, 6 | `ADD` | Add 2, 4, push result to end of list |
| 6, 24 | `MULT` | Multiply 8, 3, push result to end of list |

14. **DSA** The data structure being utilized is a queue. This is evidenced by the fact that new items are added to the end of the list, and items being removed or processed are taken from the front of the list. A stack would insert or remove items from the front of the list. Queues and stacks could both be implemented using either Linked Lists or Arrays.

15. **PL** The function capitalizes a lowercase letter. This is possible because characters can also be represented by their ASCII character codes, and because the the ASCII code for each letter is one more than the ASCII code of the previous letter in the alphabet. In other words, 'b' = 'a' + 1.

$$\begin{aligned} &= a - \text{'a'} + \text{'A'} \\ &= (\text{'a'} - \text{'a'}) + \text{'A'} \\ &= 0 + \text{'A'} \\ &= \text{'A'} \end{aligned}$$

16. **PL** We can compute the result by realizing that by subtracting `'a'` from any lowercase letter, we are left with the index of that letter in the lowercase alphabet (e.g., $a = 0$, $b = 1$, $c = 2$, $\ldots$). Then by adding that index to `'A'`, we are left with the character code for the capitalized version of that letter.

$$\begin{aligned} &= a - \text{'a'} + \text{'A'} \\ &= \text{'g'} - \text{'a'} + \text{'A'} \\ &= (\text{zero-based index of g in lowercase alphabet}) + \text{'A'} \\ &= \text{'G'} \end{aligned}$$

17. **DLNS** One's complement notation can be found by inverting each bit in a binary number. As a result, zero can be represented as $0000_2 = 0_{10}$ and as $1111_2 = -0_{10} = 0_{10}$.

18. **GEN** HTTP is a part of the Application Layer of the OSI Model. The layers of the OSI Model are as follows:

    1. Application layer
    2. Presentation layer
    3. Session layer
    4. Transport layer
    5. Network layer
    6. Data link later
    7. Physical layer

19. **DSA** A linear search works by sequentially comparing each item in a list of values with the value being searched for. The worst-case scenario for finding the desired element occurs when the search value is at the end of the list, requiring that every element in the list be checked.

20. **PL** The code on line 4 is called a Function Prototype. It is necessary because the compiler is not aware of the function (which starts on line 17) when it encounters a call to it (lines 9 and 12).

21. **PL** A trace of the code follows:

| `main` | | `addSum` | | |
|---|---|---|---|---|
| a | b | a | &b | comment |
| 4 | 5 | | | |
| 4 | 5 | 4 | 5 | call `addSum` |
| 4 | 9 | 4 | 9 | $b = 4 + 5$, $b$ is passed by reference, return 9 |
| 3 | 9 | | | |
| 3 | 9 | 3 | 9 | call `addSum` |
| 3 | 12 | 3 | 12 | $b = 3 + 9$, $b$ is passed by reference, return 12 |

Therefore, `9 12` will be printed.

22. **PL** A trace of the code follows:

| `main` | | `addSum` | | |
|---|---|---|---|---|
| a | b | a | &b | comment |
| 4 | 5 | | | |
| 4 | 5 | 4 | 5 | call `addSum` |
| 4 | 4 | 4 | 4 | Prefix decrement of `b` |
| 4 | 8 | 4 | 8 | $b = 4 + 4$, $b$ is passed by reference, return 8 |
| 3 | 8 | | | |
| 3 | 8 | 3 | 8 | call `addSum` |
| 4 | 7 | 4 | 7 | Prefix decrement of `b` |
| 3 | 11 | 3 | 11 | $b = 3 + 7$, $b$ is passed by reference, return 11 |

23. **PL** The parameter is being passed by reference. Variables being passed by reference, when updated, also update the corresponding variable in the calling function.

24. **PL** GetAverageAgeOfMammals expects an array of Mammals. However, since Dog inherits from Mammal, we can also pass an array of Dogs. The same reasoning allows us to assign a Dog to an instance of Mammal. When accessing properties of a class, the Dot-Operator is used (not the Right-Arrow operator, which is used with pointers). GetFurColor is a function on the Dog class, not Mammal.

25. **OOP** A default constructor is one in which there are no parameters. Both `Dog` and `Mammal` have constructors that require parameters be passed. Since `name` is a protected member of `Mammal` it can be accessed directly from the `Dog` class. If it were declared as private, then it could not be accessed directly. Since `Dog` is-a `Mammal`, functions within `Dog` can access public or protected members of `Mammal`. However, members of `Mammal` cannot access anything within the `Dog` class. Because `Dog` inherits from `Mammal`, this means that all instances of the `Dog` class are also instances of the `Mammal` class.

26. **OOP** Inheritance allows one object to inherit another object, which allows for easy reuse of code. Polymorphism allows us to write a function which accesses a particular function on an object, and will also work on descendant objects. Encapsulation hides the implementation details from the outside world. Requiring getters and setters for accessing private data is known as Abstraction.

27. **DLNS** Converting from decimal to binary can be done by repeatedly dividing the decimal number by 2, and recording the remainder. The reversed remainders represent the binary equivalent:

| Decimal number | Divided by 2 | Remainder |
|---|---|---|
| 19 | 9 | 1 |
| 9 | 4 | 1 |
| 4 | 2 | 0 |
| 2 | 1 | 0 |
| 1 | 0 | 1 |

The result in binary is `1 0011`, padded to 8 bits is: `0001 0011`.

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

| a[0] | a[1] | a[2] | i | j | output | comment |
|---|---|---|---|---|---|---|
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | | | | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | | | initialize i |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 0 | | initialize j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 0 | 1 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 1 | 1 | increment j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 1 | 14 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 2 | 14 | increment j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 2 | 147 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 3 | 147 | increment j, exit inner loop |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 3 | 147 | increment i |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 0 | 147 | initialize j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 0 | 1472 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 1 | 1472 | increment j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 1 | 14725 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 2 | 14725 | increment j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 2 | 147258 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 3 | 147258 | increment j, exit inner loop |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 3 | 147258 | increment i |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 0 | 147258 | initialize j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 0 | 1472583 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 1 | 1472583 | increment j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 1 | 14725836 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 2 | 14725836 | increment j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 2 | 147258369 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 3 | 147258369 | increment j, exit inner loop |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 3 | 3 | 147258369 | increment i, exit inner loop |

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

| a[0] | a[1] | a[2] | i | j | output | comment |
|---|---|---|---|---|---|---|
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | | | | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | | | initialize i |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 0 | | initialize j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 0 | 1 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 0 | 1 | increment j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 0 | 14 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 0 | 2 | 14 | increment j, exit inner loop |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 2 | 14 | increment i |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 1 | 14 | initialize j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 1 | 142 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 1 | 142 | increment j |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 1 | 1425 | |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 1 | 2 | 1425 | increment j, exit inner loop |
| {1, 2, 3} | {4, 5, 6} | {7, 8, 9} | 2 | 2 | 1425 | increment i, exit inner loop |

30. **PL** Examples of unary operators include:

1. `++` (prefix/postfix increment)
2. `--` (prefix/postfix decrement)

3. `!` (logical negation operator)
4. `&` (address-of operator)