Instructions: Submit your answers to these questions to the Curator as `OQ03` by the posted due date and time. No late submissions will be accepted.

For the next three questions, assume that the following pointer definitions are in place:

```
int* P = NULL;
int* Q = NULL;
int* R = NULL;
```

And, assume that some unknown code has been executed that has given each of these pointers a valid target, so no pointer is `NULL`, but that you have no further information about what those targets might be.

1. Suppose that `P == Q`. Which of the following statements must also be true?

1) `&P == &Q`
2) `*P == *Q`
3) `P - Q == NULL`

4) All must be true
5) All could be false
6) 1 and 2 only

7) 1 and 3 only
8) 2 and 3 only

2. Suppose that `*Q == *R`. Which of the following statements must also be true?

1) `Q == R`
2) `&Q == &R`
3) `Q - R == NULL`

4) All must be true
5) All could be false
6) 1 and 2 only

7) 1 and 3 only
8) 2 and 3 only

3. Suppose that `&*P == &*R`. Which of the following statements must also be true?

1) `&P == &R`
2) `P == R`
3) `*P == *R`

4) All must be true
5) All could be false
6) 1 and 2 only

7) 1 and 3 only
8) 2 and 3 only

For the questions 4 through 7, assume that we have a C function whose stack frame currently looks like this:

symbolic name	address	value
	0xFFA144CC	old frame pointer
X	0xFFA144C8	10
Y	0xFFA144C4	20
	. . .	
P	0xFFA144C0	0
	. . .	
Q	0xFFA144B8	0

Note: addresses are shown in hexadecimal and values are shown in base-10. Some irrelevant variables/values are not given. (If you write code to try to test your answers, there is no reason to expect you'll get the same addresses I did, and if you get a different memory layout, the behavior may be different.)

Assume the following code is executed at the in the function body:

```
P = &X;   // statement 1
Q = &Y;   // statement 2
```

The following questions are independent. That is, each assumes things are initially in the state following the execution of the two statements above.

4. Suppose the following code is executed after statements 1 and 2:

```
*P = *Q;
```

What values would the four variables have now?

	X	Y	P	Q
1)	10	10	0xFFA144C0	0xFFA144B8
2)	10	20	0xFFA144C0	0xFFA144B8
3)	20	10	0xFFA144C0	0xFFA144B8
4)	20	20	0xFFA144C0	0xFFA144B8
5)	10	10	0xFFA144C8	0xFFA144C4
6)	10	20	0xFFA144C8	0xFFA144C4
7)	20	10	0xFFA144C8	0xFFA144C4
8)	20	20	0xFFA144C8	0xFFA144C4
9)	None of these			

5. Suppose the following code is executed after statements 1 and 2:

```
Y = *P + 10;
*Q += 20;
```

What values would the four variables have now?

	X	Y	P	Q
1)	10	20	0xFFA144C8	0xFFA144C4
2)	10	30	0xFFA144C8	0xFFA144C4
3)	10	20	0xFFA144C8	0xFFA144C4
4)	10	30	0xFFA144C8	0xFFA144C4
5)	10	40	0xFFA144C8	0xFFA144C4
6)	20	40	0xFFA144C8	0xFFA144C4
7)	None of these			

6. Suppose the following code is executed after statements 1 and 2:

```
*(Q)++;
*(P)--;
```

What values would the four variables have now?

	X	Y	P	Q
1)	10	20	0xFFA144C4	0xFFA144C8
2)	11	19	0xFFA144C4	0xFFA144C8
3)	9	21	0xFFA144C4	0xFFA144C8
4)	10	20	0xFFA144C8	0xFFA144C4
5)	11	19	0xFFA144C8	0xFFA144C4
6)	9	21	0xFFA144C8	0xFFA144C4
7)	None of these			

7. Suppose the following code is executed after statements 1 and 2:

```
Q++;
P--;
*(Q)++;
*(P)--;
```

What values would the four variables have now?

	X	Y	P	Q
1)	10	20	0xFFA144C4	0xFFA144C8
2)	11	19	0xFFA144C4	0xFFA144C8
3)	10	20	0xFFA144C8	0xFFA144C4
4)	11	19	0xFFA144C8	0xFFA144C4
5)	9	21	0xFFA144C8	0xFFA144C4
6)	9	21	0xFFA144C4	0xFFA144C8

7) None of these

Suppose that `P` is an `int*` whose target is the first byte in an array of `Size` 32-bit `int` values. A programmer wants to use a while loop to traverse the array and do something (what is not important) with the values in the array. So far, he has written the following code and comments:

```
int  A[Size];

int* P = &A[0];       // *P is A[0]

int* Limit = ??;      // How do I set Limit to make the loop below work?

while ( P < Limit ) {

   // I'll write the code to process the int *P here.

   // How should I update P here?

}
```

8. How could the variable `Limit` be initialized to make the loop terminate after the last element of `A` is processed?

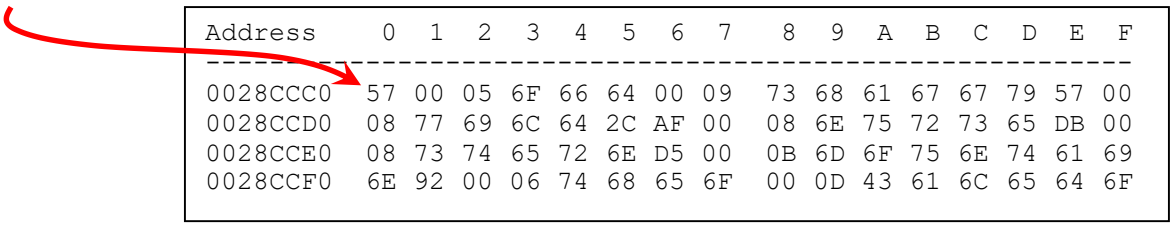
1) `P + Size`
2) `P + Size - 1`
3) `&A[Size]`
4) `&A[Size - 1]`
5) 1 or 3 only
6) 2 or 4 only
7) None of these

9. How should the variable `P` be updated so that all the elements of `A` are processed?

1) `P = P + 1;`
2) `P++;`
3) `P = P + 4;`
4) `P = P + sizeof(int);`
5) 1 or 2 only
6) 3 or 4 only
7) None of these

For questions 10 through 15, assume that p is a pointer of type `uint8_t*` and points to the beginning of a block of memory initialized as shown in the hexdump below:

```
uint8_t *p;
```



```
Address    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
-------------------------------------------------------------
0028CCC0  57 00 05 6F 66 64 00 09  73 68 61 67 67 79 57 00
0028CCD0  08 77 69 6C 64 2C AF 00  08 6E 75 72 73 65 DB 00
0028CCE0  08 73 74 65 72 6E D5 00  0B 6D 6F 75 6E 74 61 69
0028CCF0  6E 92 00 06 74 68 65 6F  00 0D 43 61 6C 65 64 6F
```

So, `p` would have the value `0x0028CCC0` and `*p` would have the value `0x57`, or `87` in base-10.

For each of the following questions, determine the value of the pointer expression.

Answers are all given in hexadecimal. Multi-byte integers are stored in memory in *little-endian* order. The answers offered below are expressed in human-friendly *big-endian* form.

10. `*(p + 4)`

1) 0x57
2) 0x00
3) 0x05
4) 0x6F
5) 0x66
6) 0x73

11. `*(p + 0x10)`

1) 0x09
2) 0x0F
3) 0x6F
4) 0x73
5) 0x5F
6) 0x08

12. `*(uint16_t*)(p + 0xA)`

1) 0x0057
2) 0x6779
3) 0x6167
4) 0x5700
5) 0x0900
6) 0x6761

13. `*(uint32_t*)(p)`

1) 0x6F050057
2) 0x00000057
3) 0x5700056F
4) 0x6F000000
5) 0xF6500075
6) 0x5708086E

14. `(uint32_t)(*p)`

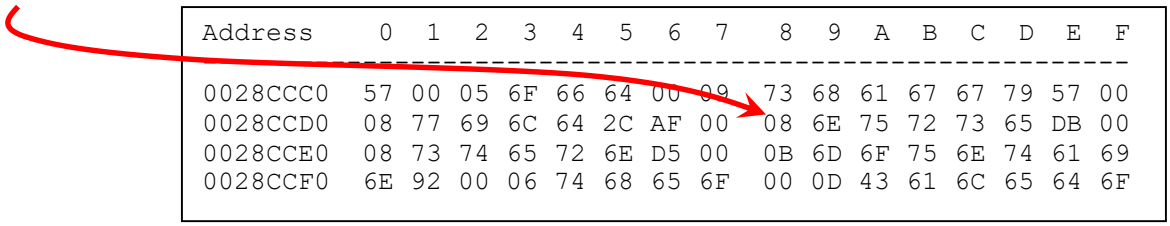
1) 0x6F050057
2) 0x00000057
3) 0x5700056F
4) 0x57000000
5) 0xF6500075
6) 0x5708086E

15. `*(uint32_t*)(p + 8)`

1) 0x6F050057
2) 0x00646F05
3) 0x73686167
4) 0x6F650005
5) 0x67616873
6) 0x09006466

For questions 16 through 20, assume that p is a pointer of type `uint8_t*` and points into a block of memory initialized as shown in the hexdump below:

```
uint8_t *p;
```



```
Address    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
-------------------------------------------------------------
0028CCC0  57 00 05 6F 66 64 00 09  73 68 61 67 67 79 57 00
0028CCD0  08 77 69 6C 64 2C AF 00  08 6E 75 72 73 65 DB 00
0028CCE0  08 73 74 65 72 6E D5 00  0B 6D 6F 75 6E 74 61 69
0028CCF0  6E 92 00 06 74 68 65 6F  00 0D 43 61 6C 65 64 6F
```

So, `p` would have the value `0x0028CCD8` and `*p` would have the value `0x08`. For each question, select the C expression that would yield the given value.

Remember that the hexdump above shows memory contents in little-endian order, while the answers below are shown in big-endian order.

16. `0x00DB6573`

1) `(uint32_t*)(*(p+4))`
2) `*(uint32_t*)(p + 4)`
3) `(uint32_t)(*p)`
4) All of them
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

17. `0x6E08`

1) `*(uint16_t*)(p)`
2) `(uint16_t)(*p)`
3) `(uint16_t)(*p) + ((uint16_t)(*(p+1)) << 4)`
4) All of them
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

18. `0x0C`

1) `(*p) + 4`
2) `*(p + 4)`
3) `(*p) >> 4`
4) All of them
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

19. `0x00AF`

1) `(*p) - 2`
2) `*(p - 2)`
3) `(*p << 16)`
4) All of them
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

20. `0x08`

1) `*p - 8`
2) `*(p - 8)`
3) `*(p + 8)`
4) All of them
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

NB: You can test your answers to questions 9 – 20 if you set up a suitable block of memory and then write `printf()` statements to check values. Here's how to get started:

```
int main() {

   uint8_t Data[64] = {0x57, 0x00, 0x05, 0x6F, 0x66, 0x64, 0x00, 0x09,
                       0x73, 0x68, 0x61, 0x67, 0x67, 0x79, 0x57, 0x00,
                       0x08, 0x77, 0x69, 0x6C, 0x64, 0x2C, 0xAF, 0x00,
                       0x08, 0x6E, 0x75, 0x72, 0x73, 0x65, 0xDB, 0x00,
                       0x08, 0x73, 0x74, 0x65, 0x72, 0x6E, 0xD5, 0x00,
                       0x0B, 0x6D, 0x6F, 0x75, 0x6E, 0x74, 0x61, 0x69,
                       0x6E, 0x92, 0x00, 0x06, 0x74, 0x68, 0x65, 0x6F,
                       0x00, 0x0D, 0x43, 0x61, 0x6C, 0x65, 0x64, 0x6F};

   // For questions 9-15
   uint8_t *p = Data;
   . . .
```

If you do this, and it's good practice with C, be careful about how you write your format specifiers in the `printf()` statements. The answers show the value in hexadecimal, with the appropriate number of leading zeros (if needed). You won't get that unless you use the right specifies (e.g., `"%X"` or `"%08x"` or something similar).

---

21. Which of the following statements are true? The values should be interpreted as <u>signed</u> 16-bit values.

1) `0x704F < 0x7053`
2) `0x704F > 0x7053`
3) `0xAFFF < 0xB00D`
4) `0xAFFF > 0xB00D`

5) 1 and 3 only
6) 1 and 4 only
7) 2 and 3 only
8) 2 and 4 only

22. Which of the following statements are true? The values should be interpreted as <u>unsigned</u> 16-bit values.

1) `0x704F < 0x7053`
2) `0x704F > 0x7053`
3) `0xAFFF < 0xB00D`
4) `0xAFFF > 0xB00D`

5) 1 and 3 only
6) 1 and 4 only
7) 2 and 3 only
8) 2 and 4 only

23. Which of the following statements are true? The values should be interpreted as <u>signed</u> 16-bit values.

1) `-0xF347 == 0x0CB8`
2) `-0xF347 == 0x0CB9`
3) `~0xF347 == 0x0CB8`
4) `~0xF347 == 0x0CB9`

5) 1 and 3 only
6) 1 and 4 only
7) 2 and 3 only
8) 2 and 4 only

24. Which of the following statements are true? The values should be interpreted as <u>unsigned</u> 16-bit values.

1) `0x704F & 0x7053 == 0x704F`
2) `0x704F & 0x7053 == 0x7053`
3) `0x704F & 0x7053 == 0x7043`
4) `0x704F | 0x7053 == 0x704F`
5) `0x704F | 0x7053 == 0x7053`
6) `0x704F | 0x7053 == 0x705F`
7) 1 and 4 only
8) 2 and 5 only
9) 3 and 6 only
10) None of these

25. Which of the following statements are true? The values should be interpreted as unsigned 16-bit values.

1) `0x704F ^ 0x7053 == 0x001C`
2) `0x704F ^ 0x7053 == 0xFFD3`
3) `0x704F ^ 0x704F == 0x0000`
4) `0x704F ^ 0x704F == 0xFFFF`

5) 1 and 3 only
6) 1 and 4 only
7) 2 and 3 only
8) 2 and 4 only
9) None of these