1. Consider a small array $a$ and large array $b$. Accessing the first element of $a$ takes more/less/the same amount of time as accessing the first element of $b$.
   (a) the same amount of time
   (b) less time
   (c) more time

2. Consider a small array $a$ and large array $b$. Accessing the last element of $a$ takes more/less/the same amount of time as accessing the first element of $b$.
   (a) the same amount of time
   (b) less time
   (c) more time

3. C arrays are:
   (a) both homogeneous and heterogeneous
   (b) neither homogeneous nor heterogeneous
   (c) homogeneous
   (d) heterogeneous

4. C arrays can be:
   (a) neither dynamically nor statically allocated
   (b) dynamically allocated only
   (c) dynamically and statically allocated
   (d) statically allocated only

Concept: static allocation of arrays

5. Why is the following array declaration not proper?
   ```c
   int z1[];
   ```
   (a) $z1$ is not a legal variable name
   (b) the size of the array is missing
   (c) it is proper
   (d) the square brackets are in the wrong place

6. Why is the following array declaration not proper?
   ```c
   int _[] = { 1 };
   ```
   (a) the size of the array is missing
   (b) arrays have to have more than one slot
   (c) _ is not a legal variable name
   (d) it is proper

7. Why is the following array declaration not proper?
   ```c
   int[1] x = { 1 };
   ```
   (a) it is proper
   (b) the brackets are in the wrong place
   (c) the size of the array is missing
8. Why is the following array declaration improper?
   ```
   int y[2] = { 1, 2, 3 };
   ```
   (a) it is proper, the last initializer is ignored
   (b) it is proper, the array is expanded to three elements
   (c) there are too many initializers
   (d) there are too few initializers

9. Why is the following array declaration improper?
   ```
   int w = { 1, 2, 3 };
   ```
   (a) there are too many initializers
   (b) there are too few initializers
   (c) the square brackets are missing
   (d) it is proper

10. Why is the following array declaration improper?
    ```
    double y[3] = { 1, 2, 3 };
    ```
    (a) there should be a comma after the 3
    (b) you cannot have integer initializers
    (c) you cannot give a size if you have initializers
    (d) it is proper

11. Why is the following array declaration improper?
    ```
    double y[1];
    ```
    (a) it is proper
    (b) arrays need more than one slot
    (c) there should be an empty initializer list
    (d) initializers are missing

12. Why is the following array declaration improper?
    ```
    double y[2] = {};
    ```
    (a) it is proper
    (b) the initializer list should be {},
    (c) there should be initializer values
    (d) the initializer list should be {,,}

13. Why is the following array declaration improper?
    ```
    double y[3] = { 1.1 2.2 3.3 };
    ```
    (a) it is proper
    (b) commas between initializers are missing
    (c) commas after each initializer are missing

14. Why is the following array declaration improper?
    ```
    double y[3] = { "1", "2", "3" };
    ```
    (a) it is proper
    (b) there should be a comma after the 3
    (c) there is a size when you have initializers
    (d) there are string initializers for a double array

15. Why is the following array declaration improper?
void y[2];
(a) void is not a valid type for arrays
(b) the initializer is missing
(c) it is proper

16. Why is the following array declaration improper?
   void *y[2];
(a) void * is not a valid type for arrays
(b) it is proper
(c) the initializer is missing

Concept: array indexing

17. T or F: Given the declaration int x[3]; and int y[3];, then y[0] = x[0]; is a logically correct assignment.
18. T or F: Given the declaration int x[3]; and int y[3];, then y[0] = x[1]; is a logically correct assignment.
19. T or F: Given the declaration int x[3]; and int y[3];, then y[0] = x[3]; is a logically correct assignment.
20. T or F: Given the declaration int x[3]; and int y[3];, then y[3] = x[2]; is a logically correct assignment.
21. An array has size \( n \). What is the index of the last element?
   (a) \( n \)
   (b) \( n - 2 \)
   (c) \( n + 1 \)
   (d) \( n - 1 \)

22. You wish to set change the first element in an array \( a \) to the value of the last element. Which of the following accomplishes that task?
   (a) \( a[0] = a[n-1] \);
   (b) \( a[1] = a[n-1] \);
   (c) \( a[1] = a[n] \);
   (d) \( a[0] = a[n] \);

23. You know that the second element of array \( a \) holds a valid index for \( a \). You wish to change the element at that index to zero. Which of the following reliably accomplishes that task?
   (a) \( a[a[1]] = a[1] \);
   (b) \( a[a[2]] = 0 \);
   (c) \( a = a[a[2]] \);
   (d) \( a[a[1]] = 0 \);

24. Consider the declaration int y[3];. As an rvalue, the expression \( y[2] \):
   (a) may cause the program to crash
   (b) will always cause the program to crash
   (c) references the last element in the array
   (d) generates an out-of-bounds error message

25. Consider the declaration int y[3];. As an rvalue, the expression \( y[3] \):
   (a) generates an out-of-bounds error message
   (b) will always cause the program to crash
   (c) references the last element in the array
   (d) may cause the program to crash

26. Consider the declaration int y[3];. As an rvalue, the expression \( y[4] \):
   (a) references the last element in the array
   (b) will always cause the program to crash
   (c) generates an out-of-bounds error message
(d) may cause the program to crash

**Concept: pointers and pseudopointers**

27. Consider the declaration `int y[3];`. The proper declaration for a pointer that can point to array `y` is:
   
   (a) `int z;`
   
   (b) `int *z[3];`
   
   (c) `int *z;`
   
   (d) `int z(*[3]);`

28. Consider the declaration `char *y[3];`. The proper declaration for a pointer that can point to array `y` is:
   
   (a) `char **z;`
   
   (b) `char *z[3];`
   
   (c) `char *z;`
   
   (d) `char *z(*[3]);`

29. Suppose `z` points to array `a`. To access the first element of `a` using `z`, one would use the expression:
   
   (a) `*z[0]`
   
   (b) `*z[1]`
   
   (c) `z[0]`
   
   (d) `z+0`
   
   (e) `z[1]`
   
   (f) `z+1`

30. Consider the declaration `int *y[3];`. To make the assignment `p = y;` legal, the declaration of `p` would have to be:
   
   (a) `int ***p;`
   
   (b) `int *p;`
   
   (c) `int p;`
   
   (d) `int **p;`

31. Consider the declaration `int **y[3];`. To make the assignment `p = y;` legal, the declaration of `p` would have to be:
   
   (a) `int *p;`
   
   (b) `int p;`
   
   (c) `int ***p;`
   
   (d) `int **p;`

32. Consider the declaration `int y[3];`. To make the assignment `p = y;` legal, the declaration of `p` would have to be:
   
   (a) `int p;`
   
   (b) `int *p;`
   
   (c) `int ***p;`
   
   (d) `int **p;`

33. Consider the declaration `char *y[3];`. To make the assignment `p[0] = y;` legal, the declaration of `p` would have to be:
   
   (a) `char *p;`
   
   (b) `char **p;`
   
   (c) `char ***p;`
   
   (d) `char p;`

34. Consider the declaration `char *y[3];`. To make the assignment `*p = y;` legal, the declaration of `p` would have to be:
   
   (a) `char p;`
   
   (b) `char *p;`
   
   (c) `char ***p;`
   
   (d) `char **p;`
35. Consider the declaration `char y[3]`. To make the assignment `p[0] = y[0]`; legal, the declaration of `p` would have to be:
   (a) `char **p;`
   (b) `char ***p;`
   (c) `char *p;`
   (d) `char p;`

36. Consider the declaration `char y[3]`. To make the assignment `*p = y[0]`; legal, the declaration of `p` would have to be:
   (a) `char **p;`
   (b) `char p;`
   (c) `char ***p;`
   (d) `char *p;`

37. Consider the declaration `char **y[3]`. To make the assignment `p[0] = y[0]`; legal, the declaration of `p` would have to be:
   (a) `char **p;`
   (b) `char p;`
   (c) `char *p;`
   (d) `char ***p;`

38. Consider the declaration `char **y[3]`. To make the assignment `*p = y[0]`; legal, the declaration of `p` would have to be:
   (a) `char *p;`
   (b) `char ***p;`
   (c) `char p;`
   (d) `char **p;`

39. Consider the declaration `char *y[3]`. To make the assignment `p[0] = y`; legal, the declaration of `p` would have to be:
   (a) `char **p;`
   (b) `char *p;`
   (c) `char ***p;`
   (d) `char p;`

40. Consider the declaration `char *y[3]`. To make the assignment `*p = y`; legal, the declaration of `p` would have to be:
   (a) `char *p;`
   (b) `char ***p;`
   (c) `char p;`
   (d) `char **p;`

41. **T** or **F**: Given the declaration `int x[3];` and `int y[3];` then `x = y;` will cause a compiler error.
42. **T** or **F**: Given the declaration `int x[3];` and `int y[3];` then `y = x;` will cause a compiler error.
43. **T** or **F**: Given the declaration `int x[3];` and `int y[3];` then `y = x[0];` will cause a compiler error.
44. **T** or **F**: Given the declaration `int x[3];` and `int y[3];` then `y[0] = x;` will cause a compiler error.
45. Consider the declaration `int z[3];`. The name `z` is:
   (a) a pointer
   (b) neither a pointer nor a pseudopointer
   (c) a pseudopointer
46. **T** or **F**: You can assign a pointer to a pseudopointer.
47. **T** or **F**: You can assign a pseudopointer to a pseudopointer.
48. **T** or **F**: You can assign a pointer to a pointer.
49. **T** or **F**: You can assign a pseudopointer to a pointer.
50. Using the `sizeof` operator on a pseudopointer to an array gives you:
   (a) you are not allowed to use `sizeof` on a pseudopointer
(b) the size of the pseudopointer
(c) the number of slots in the array
(d) the number of bytes in the array

51. Using the `sizeof` operator on a pointer to an array gives you:
   (a) you are not allowed to use `sizeof` on a pointer
   (b) the number of bytes used to hold an address
   (c) the number of bytes in the array
   (d) the number of slots in the array

52. Consider:

   ```c
   int a[3];
   int *p = a;
   ```

   The value stored at the memory location associated with `p` is:
   (a) undefined, since the operation is illegal
   (b) the name `a`
   (c) the value of the first element of the array
   (d) the address of the first slot of the array

53. Consider:

   ```c
   int a[3];
   int *p = a;
   ```

   The value stored at the memory address found in `p` is:
   (a) the name `a`
   (b) the address of the first slot of the array
   (c) the value of the first element of the array
   (d) undefined, since the operation is illegal

54. The array access `a[3]` can be rewritten as:
   (a) *(a) + 3
   (b) *(a) + 2
   (c) *(a + 3)
   (d) *(a + 2)

55. The pointer access `*(a+4)` can be rewritten as:
   (a) *(a[3])
   (b) *(a[4])
   (c) a[3]
   (d) a[4]

56. The array access `a[0]` cannot be rewritten as:
   (a) *a
   (b) *(a + 2) - 2
   (c) *(a + 0)
   (d) *(a + 2 - 2)

57. Consider:

   ```c
   int a[] = { 10, 100, 100 };
   int *p = a;
   ```

   The expression `p + 1`:
   (a) points to the second slot of `a`
   (b) evaluates to 101
   (c) points to the first slot of `a`
58. Consider:

```c
int a[] = { 10, 100, 100 };
int *p = a;
```

The expression `p + 2`:
(a) evaluates to 1001
(b) evaluates to 101
(c) points to the third slot of `a`
(d) points to the first memory location beyond `a`

59. Consider:

```c
int a[] = { 10, 100, 100 };
int *p = a;
```

The expression `p + 3`:
(a) evaluates to 1001
(b) points to the third slot of `a`
(c) points to the first memory location beyond `a`
(d) would generate an error by the compiler

60. The standard library function that can be used to dynamically allocate an array is called:

(a) allocate
(b) dallocate
(c) dymalloc
(d) malloc

**Concept:** strings and pseudopointers

61. The string "dog" is stored as:
(a) three non-contiguous memory locations
(b) 4 contiguous locations in memory
(c) a single location in memory
(d) 3 contiguous locations in memory

62. A pointer to the string "rat" has the type:

(a) char *
(b) char [3]
(c) string *
(d) string
(e) char

63. Consider a pointer `p` to the string "bat". What is the rvalue of `p[1]`?
(a) the letter 'b'
(b) the first memory location in the string
(c) the second memory location in the string
(d) the letter 'a'

64. Consider a pointer `p` to the string "bat". What is the lvalue of `p[1]`?
(a) the letter 'a'
(b) the letter 'b'
(c) the first memory location in the string
(d) the second memory location in the string

65. Consider a pointer `p` to the string "bat". What is the rvalue of `p[3]`?
(a) the null character
(b) the letter 't'
(c) unknown
(d) the last memory location in the string

66. Consider a pointer \( p \) to the string “bat”. What is the lvalue of \( p[4] \)?

(a) the null character
(b) the second memory location beyond the string
(c) unknown
(d) the first memory location beyond the string