Preliminary 75 examination questions for final exam (Algorithms and Data Structures, *Fall 2024*)

1. Name the last element in array of size n:

2. Name the data structures following FIFO principle.

3. Function call is better done with which data structure?

4. What data structure suitable for file management?

5. What data structure suitable for task management?

6. What data structure is suitable for storing pack of banknotes?

7. How do we store class in header file? Do we separate class signature from implementation?

8. Name the data structures following LIFO principle:

9. Which search algorithm(s) require the elements to be sorted?

10. Which search algorithm(s) doesn’t require the elements to be sorted?

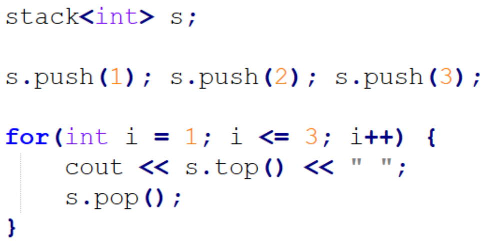
11. What are the common functions of stack data structure?

12. What are the common functions of queue data structure?

13. What are the common functions of linked list data structure?

14. How different is circular queue from common queue data structure?

15. Output of the following code snippet:

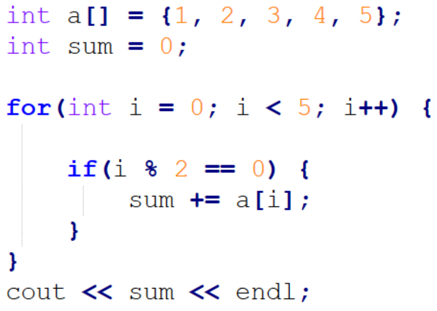


```
stack<int> s;

s.push(1); s.push(2); s.push(3);

for(int i = 1; i <= 3; i++) {
    cout << s.top() << " ";
    s.pop();
}
```

16. Output of the following code snippet:

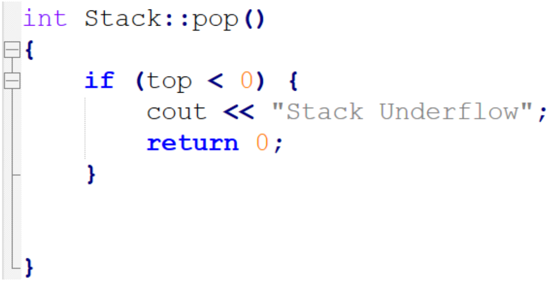


```
int a[] = {1, 2, 3, 4, 5};
int sum = 0;

for(int i = 0; i < 5; i++) {

    if(i % 2 == 0) {
        sum += a[i];
    }
}
cout << sum << endl;
```

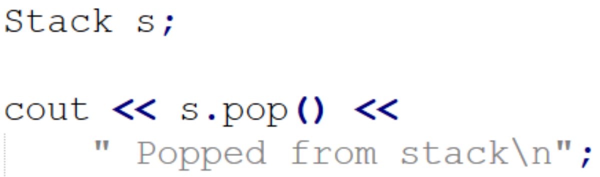
17. Write how else part of pop function in stack using array:



```
int Stack::pop()
{
    if (top < 0) {
        cout << "Stack Underflow";
        return 0;
    }


}
```

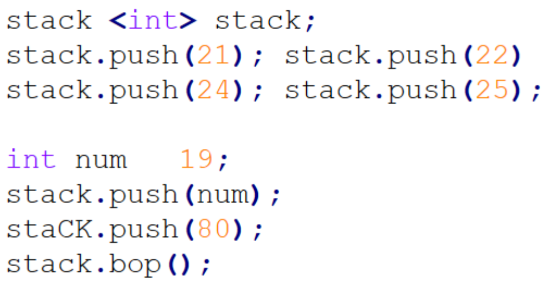
18. Below is source code and write the source code to add numbers ***4, 5, 6*** to stack in ***between the lines*** stack created and first item popped from it.



```
Stack s;

cout << s.pop() <<
    " Popped from stack\n";
```

19. What are the syntax errors from the following source code:



```
stack <int> stack;
stack.push(21); stack.push(22)
stack.push(24); stack.push(25);

int num   19;
stack.push(num);
staCK.push(80);
stack.bop();
```

20. Using STL stack library write a source code that creates stack, adds few numbers and prints first item on stack on console. Please, include iostream and other necessary source in your written answer:

21. Using STL queue library write a source code that creates queue, adds few numbers and prints front and rear item on queue on console. Please, include iostream and other necessary source in your written answer:

22. What is an array, and what index out of bound exception means?

23. Can array be resized in runtime?

24. How linear search and binary search different from one another?

25. Which sorting algorithm is more efficient, insertion sort or selection sort? Explain your arguments.

26. What is a linked list, and how does it differ from an array?

27. Describe the structure of a node in a linked list. What are the typical components of a node?

28. What are the key types of linked lists? Explain the differences between singly linked lists, doubly linked lists, and circular linked lists.

29. What is the head of a linked list, and why is it important?

30. How do you traverse a linked list, and what are some challenges involved in this process?

31. Explain the process of inserting a new node at the beginning, middle, and end of a linked list.

32. How is a node deleted from a linked list? Discuss the challenges of deleting a node in different scenarios (e.g., at the head, in the middle, or at the end).

33. What are the advantages of using a linked list over an array? Provide examples where a linked list is more efficient.

34. What are the disadvantages of linked lists compared to arrays?

35. How do circular linked lists differ from regular linked lists? What are some applications of circular linked lists?

36. Explain the concept of a doubly linked list. How does having two pointers per node affect operations and performance?

37. What is the time complexity of basic operations in a singly linked list (e.g., traversal, insertion, deletion)?

38. What is the role of the "null" or "nil" pointer in linked lists?

39. How can linked lists be used to implement other data structures like stacks or queues?

40. What is a self-referential structure in the context of linked lists, and why is it fundamental for their implementation?

41. What is a graph data structure, and what are its key components?

42. Differentiate between directed and undirected graphs with examples.

43. What are weighted graphs, and how are weights represented? Provide a real-world example of their use.

44. Explain the difference between a sparse graph and a dense graph. How is this distinction useful in graph analysis?

45. What is a complete graph, and how many edges does a complete graph with nnn vertices have?

46. What is a path in a graph? Differentiate between a simple path and a cycle.

47. Explain the difference between connected and disconnected graphs. What is a strongly connected component in a directed graph?

48. What is a tree, and how is it a special case of a graph?

49. Compare and contrast adjacency lists and adjacency matrices for graph representation. What are the advantages and disadvantages of each?

50. What is the breadth-first search (BFS) algorithm? Describe its steps and typical use cases.

51. What is the depth-first search (DFS) algorithm? How does it differ from BFS in terms of implementation and use cases?

52. Define a graph traversal. What is the significance of graph traversal in applications like web crawling or social network analysis?

53. What is Dijkstra’s algorithm? Explain how it is used to find the shortest path in a weighted graph.

54. What is a minimum spanning tree (MST)? Name two algorithms used to find an MST and compare their approaches.

55. What is a bipartite graph, and how can you determine if a given graph is bipartite? Provide an example.

56. What is a map data structure, and what are its key components?

57. How does a map differ from other data structures like arrays and linked lists?

58. Explain the difference between a hash map and a tree map. How does their underlying implementation affect performance?

59. What are the typical operations supported by a map (e.g., insertion, deletion, lookup), and what are their time complexities?

60. What is a hash function, and why is it critical for the implementation of a hash map?

61. What are hash collisions, and how are they handled in a hash map? Compare techniques like separate chaining and open addressing.

62. What is the significance of key-value pairs in a map, and how does the choice of key affect the map’s efficiency?

63. How do ordered maps, such as a tree map, maintain their order, and why is this feature useful?

64. Discuss the differences between mutable and immutable maps, and give examples of use cases for each.

65. What are some real-world applications of the map data structure, and why is it preferred in those scenarios?

66. What is an iterator, and what role does it play in traversing a data structure?

67. Explain the difference between internal and external iteration. Which is more commonly used in modern programming languages?

68. How do iterators ensure that data structures can be accessed sequentially without exposing their internal implementation details?

69. What is the difference between a forward iterator, a bidirectional iterator, and a random-access iterator? Provide examples of data structures where each type might be used.

70. What are the advantages of using iterators compared to traditional looping constructs (e.g., for and while loops) for data structure traversal?

71. What is a tuple in data structures, and how does it differ from other data types like arrays or lists?

72. What is a pair, and how is it typically used in programming? Provide examples of common use cases.

73. Explain the immutability of tuples in programming languages like Python. How does this characteristic affect their use?

74. Compare and contrast tuples and pairs. When would you choose one over the other?

75. Discuss the advantages and limitations of using tuples or pairs to store related data compared to using custom classes or structs.

~ ~ END OF DOCUMENT ~~