Multiple Choice Questions for DS

# Questions on C++

**Question 1:** Consider the following four ways to write the “for” loop in C++.

/\* (1) \*/ for (unsigned i = 0; i < n; i++) { /\* body \*/ }  
/\* (2) \*/ for (unsigned i = 0; i < n; ++i) { /\* body \*/ }  
/\* (3) \*/ for (int i = 0; i < n; i++) { /\* body \*/ }  
/\* (4) \*/ for (int i = 0; i < n; ++i) { /\* body \*/ }

Select one correct statement.

1. Using either data type (unsigned or int) is the same behavior of the loop.
2. **Using “unsigned i” works better for very large number of iterations**
3. Using “int i” works better for very large number of iterations
4. Using “++i” means that in the loop body “i” takes values from 1 to n (inclusive); not from 0 to n-1.

a = 3;

b = ++a; //

c = a++; //

**Question 2:** Compare the following code fragments to take different actions depending on the values of a digit variable d (0,...,9):

/\* IF-ELSE CONSTRUCT ... Linear search \*/  
  
if (d == 0) { name = “zero”; }  
else if (d == 1) { name = “one”; }  
/\* ... \*/  
else if (d == 9) {name = “nine”; }  
else { name = “unknown; }

/\* SWITCH-CASE STATEMENT \*/

switch (d) {  
 case 0: name = “zero”; break;  
 case 1: name = “one”; break;  
 /\* ... \*/  
 case 9: name = “nine”;   
 default: name = “unknown”;   
}

What is the main difference between the “switch-case” statement and the “if-else” statements in this context? Select one.

1. The “switch-case” statement is more readable;
2. **The “switch-case” notation is more efficient.**
3. The “if-else” notation is more efficient.
4. The “switch-case” notation allows the programmer to combine related branches by writing them next to each other.

**Question 3:** Are there any situations when it is advisable to write “**std::cout**” all the time (instead of adding a line “**using namespace std**” at the beginning, and then writing “**cout**”).

1. No advantages; both notations are fine in all situations.
2. **One should prefer “std::cout” in C++ header files.**
3. One should prefer “std::cout” when implementing class object methods (rather than functions outside any class).
4. Notation “std::cout” is preferred if you need interoperability of C++ with C.

**Question 4:** Under which circumstances you need to use the **#include guard** of the following kind:   
#infndef MYCLASS\_H   
#define MYCLASS\_H

/\* Header of a C++ class. \*/

#endif

Select one statement.

1. The #include guards are mandatory syntax for all C++ header files.
2. **The #include guards are necessary, if the same header file can be included multiple times**
3. The #include guards should surround custom C++ template code
4. The #include guards should be used for the header files that define global variables and constants.

**Question 5:** A programmer considers that it is necessary to write some include directives in the middle of a C++ source file like this:

/\* Some C++ statements and **functions** \*/   
#include “myclass.h”  
/\* More C++ statements and functions \*/

What would you say to this programmer? Select one answer.

1. This is sometimes necessary, if you want to define things in certain order.
2. The programmer should move the include to the top of his C++ file.
3. **The programmer should separate declarations from definitions in these header files, then reorder.**
4. If such includes are necessary, then the content of the header file contents should be inserted in this location.

**Analysis:** Include command just copy-pastes the content of the file in some location – BEFORE that file is compiled; it does not have any intelligent behavior.   
Moving it to the beginning may break a few things (if header files are improperly constructed – if they contain code that depends on earlier declarations; if they do not contain

**Question 6:** Executable code is usually not written in C++ header files (MyClass.h, etc.). What are the main reasons why source code should be in the CPP files.   
Select 3 answers

1. Some C++ code cannot be compiled unless it is in a CPP file (rather than H file)
2. **In C++ any variable or function should be declared before they are used; and the header files let us separate declaration from definition.**
3. **The source code changes in a header file would trigger recompilation of all the classes, where it is included.**
4. **Placing only declarations in a header allows two classes to reference each other.**

# Questions on Data Structures

**Question 1:** Assume that you need to create an English-to-English dictionary that explains various words of Indian origin (Avatar, Bandana, Guru, Jungle, Karma, etc.); provides their explanations. You need to store them in a data structure provided by STL – and you need to iterate over them in the shortlex order.

Select one correct statement.

1. You would need to use Dictionary<string,string> instead of Map<string,string>
2. You cannot use “string” objects; you need to use custom class for dictionary entries with overloaded operator < (less than).
3. **You need a custom compare function that is antisymmetric and transitive, pass its pointer to a STL class constructor.**

**Question 2:** This class is similar to your Exercise02 (current data about a student)

class Student {  
public:  
 int birthDate; /\* in seconds after Jan 1, 1970 \*/  
 int height; /\* in centimeters \*/  
 int weight; /\* in grams \*/  
 string firstName;   
};

We want to store these students in a predefined C++ class unordered\_set. What is the best hash function to use in this case?

1. The default hash function
2. MD5 or similar secure hash is preferrred – as it also protects the student’s personal data
3. Compute the predefined hash values on all the 4 attribute values; join them with OR.
4. **Compute the predefined hash values on some of the 4 attribute values; join them with XOR.**

**Question 3:** We want to build a hash table to store a few numbers. The hash function raises each number to the 4th power and takes the last digit. The hash table has 10 buckets (one bucket per each digit). How many buckets will be filled?

Answer: 4

**Question 4:** A Sequence ADT is an abstract data type. It allows arbitrary access of any element in the sequence, allows to insert and erase them at every location (One implementation of it is STL class “vector”.)  
Sequence ADT can also be implemented either as a linked list or as a C++ array.   
What are the methods that are faster for a linked list implementation (compared to an array)?

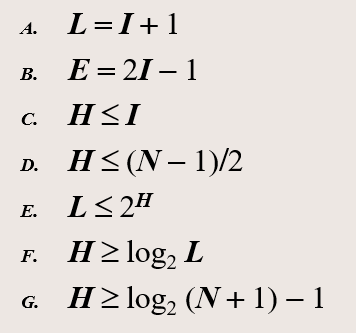
1. size(), empty() – getting the current length (and check, if it is empty)
2. **insertFront(e), eraseFront() – inserting and erasing elements at the front**
3. insertBack(e), eraseBack() – inserting and erasing elements at the back
4. begin(), end() – getting the first and the last element
5. at(i), set(i,e) – getting/updating an element at the i-th position.

**Question 5:** Sequence ADT can be implemented either as a linked list or as a C++ array.   
What are the methods that are faster for an array implementation?

1. size(), empty() – getting the current length (and check, if it is empty)
2. insertFront(e), eraseFront() – inserting and erasing elements at the front
3. insertBack(e), eraseBack() – inserting and erasing elements at the back
4. begin(), end() – getting the first and the last element
5. **at(i), set(i,e) – getting/updating an element at the i-th position.**

**Question 6:** For a full binary tree T introduce the following notation:   
N – the total number of nodes  
L – the number of leaves (external nodes)  
I – the number of internal nodes  
E – the number of edges  
H – the height of the tree

Identify, which statements are not necessarily true.



(Answer: ACDEFG)

**Question 7:** Four numbers 10,20,30,40 are inserted in a priority queue in some order (implemented as an array-based heap). It is a “minimum heap” (it supports the operation removeMin()).   
In how many ways the array of this heap can be filled with the values?

**Answer:** 3

**Question 8:** Brute force or exhaustive search paradigm is characterized by verifying every item, checking every item in a list. Identify which algorithm best matches this paradigm:

1. **Building the truth table for a Boolean logic formula.**
2. Kruskal’s algorithm to find the Minimum Spanning Tree in a weighted graph.
3. Matrix multiplication using Strassen’s algorithm
4. Computing Fibonacci number F(n) by filling in an array of length n+1.