Practice Questions: Intro to C++

Data Structures and Algorithms

6 March, 2025

So why C++?

C is a procedural programming language focused on structured code and functions, while C++ enhances it with object-oriented programming (OOP), introducing classes, objects, inheritance, and polymorphism for improved code organization and reusability. C++ offers additional flexibility through function overloading for intuitive coding, templates for generic programming, and exception handling for robust error management. The Standard Template Library (STL) further boosts efficiency with pre-built tools, making C++ more versatile than C.

C++ is favored for Data Structures and Algorithms (DSA) over C due to its superior memory management features. With constructors and destructors, C++ enables automatic initialization and cleanup of resources, significantly lowering the chances of memory leaks that are more common with manual management in C. Furthermore, smart pointers, such as `unique_ptr` and `share_ptr`, enhance safety by automatically managing memory deallocation, reducing risks like dangling pointers or memory corruption. This streamlined memory handling makes C++ a more robust choice for implementing and maintaining efficient DSA solutions compared to C.

1 A Person Class - Easy

Implement a Person class with private member variables name (a char array of size 50) and age (an int). Include public member functions:

- A `constructor` that initializes name and age with given values. (remember OOPS in python, an object of a class calls the constructor function to initialize)
- `void setName(const char* newName)` to update the name (hint: use `strcpy`)
- `void setAge(int newAge)` to update the age
- `void print() const` to display the name and age

In the main function, make the person object `obj` with a certain name and age, and the call `obj.print()` to display the name and age in the terminal.

2 A Node Class - Easy

In the same veins as the question above, create a C++ class named Node that represents a node in a singly linked list. The class should include:

- A private integer member `data` to store the node's value.
- A private pointer `next` to the next `Node`.
- A constructor that initializes data to 0 and next to `nullptr`.

The class should have the following public methods:

- `int get_data()` to return the data value.

- `void set_data(int value)` to set the data value.
- `Node* get_next()` to return the next pointer.
- `void set_next(Node* ptr)` to set the next pointer.

In the main function, create two Node objects, set the next pointer of the first node to point to the second node (assign different integer values to their data (e.g., 10 and 20)) and print their data values using `get_data()`.

3 Vehicle Hierarchy - Medium

Create a hierarchy with a base class Vehicle and two derived classes, Car and Bicycle.

3.1 Base Class: Vehicle

```
- Private member: speed (double).

- Public member functions:
    - Constructor: Initializes speed to 0.
    - virtual void accelerate(double amount): Increases speed by amount.
    - double getSpeed() const: Returns speed.
```

3.2 Derived Class: Car

```
- Private member: fuelLevel (double).

- Public member functions:
    - Constructor: Initializes fuelLevel to a given value and speed to 0.
    - Override accelerate(double amount): Increases speed by amount and decreases fuelLevel
                                          by amount * 0.1.
    - double getFuelLevel() const: Returns fuelLevel.
```

3.3 Derived Class: Bicycle

```
- No additional members.
- Public member functions:
    - Override accelerate(double amount): Increases speed by amount (no fuel involved).
```

In the main function, create a Car with initial `fuelLevel` of 100.0 and a Bicycle. Call `accelerate(20.0)` on both, then print their speeds and, for the Car, its fuel level.

4 Linked List Node with Inheritance - Medium

Create a base class named `SinglyLinkedListNode` with:

```
- A protected integer member data.
- A protected pointer next to the next node.
- A constructor that initializes data to 0 and next to nullptr.

- Public methods:
    - int get_data() to return the data value.
    - void set_data(int value) to set the data value.
    - Node* get_next() to return the next pointer.
    - void set_next(Node* ptr) to set the next pointer.
    - A virtual method void print() that outputs the data value and whether
      next is nullptr (e.g., "Data: 10, Next: null" or "Data: 10, Next: not null").
```

Create a derived class named `DoublyLinkedListNode` that inherits from `SinglyLinkedListNode` and adds:

```
- A protected pointer prev to the previous node.
- A constructor that initializes data to 0 and both next and prev to nullptr.

- Public methods:
    - Node* get_prev() to return the previous pointer.
    - void set_prev(Node* ptr) to set the previous pointer.
    - An overridden print() method that also outputs whether prev is nullptr
      (e.g., "Data: 20, Next: null, Prev: null").
```

In the main function, create one `SinglyLinkedListNode` object and one `DoublyLinkedListNode` object (set their data to different values (e.g., 10 and 20)) and call the `print()` method on both objects to demonstrate method overriding.

5 Function Overloading - Easy

Write a C++ program with two overloaded functions named `max_value`:

- One version takes an integer array (`int arr[]`) and its size (`int size`) as parameters and returns the maximum integer value in the array.
- Another version takes a float array (`float arr[]`) and its size (`int size`) as parameters and returns the maximum float value in the array.

In the main function, create an integer array (e.g., 3, 1, 4, 2) and a float array (e.g., 1.5, 2.7, 0.9) and use the `max_value` functions to find and print the maximum value in each array.

6 Generic Stack Implementation - Hard

Implement a generic `Stack<T>` class using templates.

```
- Private Members:
    - T* data: Pointer to a dynamically allocated array.
    - int capacity: Maximum number of elements.
    - int topIndex: Index of the top element (-1 if empty).

- Public Member Functions:
    - Constructor: Takes an initial capacity, allocates data, and sets topIndex to -1.
    - Destructor: Frees data.
    - void push(T value): Adds value to the top if not full (do nothing if full).
    - T pop(): Removes and returns the top element if not empty (return a
               default value like 0 if empty).
    - bool isEmpty() const: Returns true if topIndex is -1.
```

In the main function, test out the robustness and the correctness for the stack implementation with any kind of data type.
