**CPP Questions**

1. **Explain the different types of containers available in the standard library, what are their differences? Explain use cases for each.**
2. **Sequence containers:** These containers store arbitrary data elements in data structures that can be accessed sequentially by their position, the main sequence containers are:

* **Array:** This data structure implements a fix-sized array of elements.
* **Vector:** This data structure implements a dynamic array, in other words, it can be resized.
* **List:** This data structure is a double linked list, it is characterized by its data, which is sorted non-contiguously in memory as the previous data structures.

1. **Associative containers:** These containers store arbitrary data elements in data structures that can be accessed by their associated key, the main associative containers are:

* **Map:** This data structure provides a collection of paired elements, the first element is the key, and the second elements is the value corresponding to that key.
* **Set:** This data structure provides a set of elements, which must also have a key; in this case, keys must be unique.

1. **Container Adaptors:** These containers are constructed by taking the sequential containers and adapting its interface to provide a desired behavior, the main associative containers are:

* **Stack:** This data structure adapts a sequence container to provide a LIFO data structure.
* **Queue:** This data structure adapts a sequence container to provide a FIFO data structure.

**Differences:**

Each type of container differentiate from the other by using different structures and holding data and different algorithmic time for their operations, e.g. Adapters differentiate from the other containers by allowing only push/pop operations and no random insert, only allowing FIFO or LIFO behaviors.

**Use cases:**

* Sequential adapters should be used especially when data needs to be accessed sequentially.
* Associative containers should be used to access data through a key instead of an index, just like a dictionary.
* Container adapters are used for specific linear requirements (a pile or a stack) that have particular restricted functionalities such as FIFO (a line for a concert) or LIFO (a pile of plates).

1. **What are some sorting algorithms, list at least 3? What are their differences and give use cases for each.**

|  |  |  |
| --- | --- | --- |
| **Sorting Method** | **Differences** | **Use cases** |
| **Bubble Sort** | * It’s adaptive:   Minimum time is O(n)  Maximum time is O(n^2)   * It’s stable. | *Bubble sort can be used to get the k largest numbers in an array of length n with n>k.* |
| **Insertion Sort** | * Requires shifting elements * Cannot be used to give k largest numbers in array as bubble sorting. * It’s adaptive:   Minimum time is O(n)  Maximum time is O(n^2)   * It’s stable. | *Insertion sort is the best sorting algorithm when it comes to a linked list.* |
| **Quick Sort** | * Uses partitioning position for sorting elements. * It is recursive. * It’s not adaptive:   Minimum time is O(nLogn)  Maximum time is O(n^2) | *Quick sort is used for information searching.* |

1. **What is the purpose of virtual destructors? What types of issues can arise if not used correctly.**

Virtual destructors are used to properly delete or free up memory space allocated by a pointer to a derived class through the base class, normally the derived class destructor will not be called, only the base class destructor.

In other words, if we don’t set the base class destructor as virtual, then only the resources acquired from the base class code will be released, and memory from the derived destructor won’t be freed.

1. **Explain the keyword: static. What does it mean in each context?**

In C++ the static keyword allows an element to adopt a certain behavior, this elements are allocated in static storage area in memory, the static keyword can be used with the following elements:

* **Static Variables inside Functions:**

Static keyword inside functions can be used to initialize a variable only once, and then that variable will hold its value in static storage area, persisting through new function calls.

* **Static Class Objects:**

Static keyword will make class instances persist in the static storage area too, so if the class is instanced from a function, that instance will not be destroyed until the program ends or until it is destroyed manually.

* **Static Data Members in Class:**

Static keyword will allow static data members of a class to be shared by all the class instances. These data members are not dependent on object initialization, they must be initialized outside the class.

* **Static Member Functions in Class:**

Static keyword in a member function of a class allows the function to be called directly from another scope, only by using the class name and scope resolution operator ::.

1. **When are static member variables initialized?**

They occur in the static initialization and this occurs in compile time.

Initial values for static variables are tested during compilation and burned into the data section of the .exe file, this is called constant initialization.

If the initial value of a static element could not be evaluated at compile time, the compiler will perform zero-initialization. Therefore, after the static initialization, all static elements will be either const-initialized or zero-initialized.

1. **What is the difference between R-Values and L-Values?**

The main difference plays a role in the writing and understanding of expressions.

An L-value is an expression that could represent an object reference, such as an array, a variable name, a pointer, or anything that points to a specific memory location. On the other hand, an R-value in general means the data that will be stored in the space of memory allocated by the L-value.

We can think of L-values as containers and R-values as things contained in the containers. Without a container, they would expire.

1. **Is this code safe? If so why? If not why?**

**std::string foo(){**

**std::string something = “avalue”;**

**return something;**

**}**

Yes, by doing “return something”, it means that the string is being returned by value, in other words, it’s returning a copy of the string “something”, if the function were returning a string pointer, the function calls would be dealing with unstable references in the future.

1. **Why would you use new rather than malloc when allocating an object? Likewise, what’s the difference between free and delete?**

We use “new” when it is necessary to call the constructor of the class instance that will be allocated in heap; malloc will not call the constructor.

The same way, we use “delete” when it is necessary to call the destructor of the class, “free” will not call the destructor.

1. **What are the some of the principles of object-oriented programming?**

**Encapsulation:**

We accomplish encapsulation when a class maintains a private state for its attributes, and the only way to get access to modify them is by calling their public functions getters and setters.

**Abstraction:**

Abstraction means to provide only essential and required information about data to the outside world hiding particular details and implementation.

**Inheritance**

It refers to the capacity of a class to acquire certain properties from another class, the class that provides its properties is called parent or base class, and the class that inherits properties from the parent class is called child or derived class.

**Polymorphism**

It refers to the capacity of a base class to provide its properties to one or multiple derived classes, allowing them to have their own implementations of the properties, in this case the base class would be just acting as an interface that lists the properties that derived classes will implement.

1. **Explain inheritance vs composition vs aggregation**

**Inheritance**: This property allows new classes from existing classes, the derived class can be a new base class, and can have a new derived class, having a hierarchy.

**Aggregation:** Aggregation is a type of association that includes a class as a part of another class, the properties can be shared between those classes.

**Composition:** It is a strong way to compose classes, where the lifetime of the contained class should coincide with the base class lifetime.

1. **Should you always initialize variables?**

Uninitialized variables are a common cause of problems in a C or C++ software, it is important to initialize variables as a coding standard. Initializing variables has many advantages not just because a program that works correctly is infinity faster than one that does not, but also the development team spends less time finding and fixing defects that should not be there.

1. **Write a program (or multiple) in a known programming language to do the following:**
2. **Query for installed windows patches.**

**In python:**

from windows\_tools.updates import get\_windows\_updates

for update in get\_windows\_updates(filter\_duplicates=True):

print(update)

**It is required to install windows\_tools.updates:**

pip install windows\_tools.updates

1. **Write a program (or multiple) in a known programming language to do the following:**
2. **Query for windows system information.**

#include <windows.h>

#include <stdio.h>

#pragma comment(lib, "user32.lib")

void main()

{

SYSTEM\_INFO siSysInfo;

GetSystemInfo(&siSysInfo);

printf("Hardware information: \n");

printf(" OEM ID: %u\n", siSysInfo.dwOemId);

printf(" Number of processors: %u\n",

siSysInfo.dwNumberOfProcessors);

printf(" Page size: %u\n", siSysInfo.dwPageSize);

printf(" Processor type: %u\n", siSysInfo.dwProcessorType);

printf(" Minimum application address: %lx\n",

siSysInfo.lpMinimumApplicationAddress);

printf(" Maximum application address: %lx\n",

siSysInfo.lpMaximumApplicationAddress);

printf(" Active processor mask: %u\n",

siSysInfo.dwActiveProcessorMask);

}

1. **Name some tools and/or techniques that you personally find to be the most helpful surrounding development.**

* **Scrum** as an agile development methodology based on iterative processes.
* **Jira** as an adequate Agile tool for Agile methodologies for a team software development.
* **Clean** **code** for implementing the best practices of coding in software development.
* **Udemy** as a great source of learning with its multiple programming video courses.
* **StackOverflow** as a good source of answers to common problems in the development community.
* **GitHub/Gitbucket** as tools to review code, manage projects, and build software.
* **Visual studio code/community** as good IDEs for compiling and executing C++ code.
* **Eclipse** as a good IDE for compiling and executing Java code.

1. **Name some tools and/or techniques that you personally find to be the most helpful surrounding code maintenance.**

* Adding useful comments
* Using descriptive names in variables, methods, classes, etc.
* Reading some code from a more experienced developer.
* Git

1. **Scenario: You are dealing with legacy code containing no test suites nor documented requirements, and are required to make a change. Describe your expected process for how you may approach the solution.**
2. **Consider both a long term and short-term solutions.**

In a long-term scenario, I would make a thorough analysis, debug and test of the legacy code to see what is actually happening; if possible, I would send burned data to the compiler and check out the different results it could give me. After different tests, the code starts speaking by itself, giving me an approach of where I have to make the changes. I would repeat the same process to finally know what changes I have to make.

A short-term scenario, wouldn’t be too different from the other one, but the time that I would spend making a thorough analysis of the code, would rather be spent in checking out in the internet for similar code or similar changes to the ones I have to make, If there was the possibility to contact one the legacy code programmers, I would definitely get advantage of that opportunity.

1. **What concerns do you had supporting legacy operating systems? (If any)**

Just as most of technology, legacy operating systems get old and the costs can get out of control because fewer and fewer programmers know that required programing language, and bugs become harder to fix and more difficult to understand, causing costs increase.

1. **Tell us about a project you worked on that you found to be interesting or unusual.**

I worked in a 3D racing videogame made with Unreal Engine 4 and C++, when talking about videogames, it is very important to properly manage memory and to have a good knowledge of OOP and pointers, objects in this case would have a graphic representation. In my game I had to use many classes to make a wheeled vehicle with a suspension system, each vehicle wheel was a class that needed to be handled from the vehicle class.

This vehicle class inherits from a class already made by Unreal Engine called Wheeled Vehicle, which provides many variables, methods and functions that allows modifying Wheel objects and physics that will allow an object to act just like a vehicle. The Wheeled Vehicle class supports 3D objects and animations that allows giving the car a 3D visualization.



The C++ that Unreal Engine uses is a modification of the normal C++, for example, a class is named UClass, a *std::vector* in Unreal Engine is called *TArray*, every class that controls a character in the game should start with the prefix A, such as *AWheeledVehicle.cpp* and so forth, but the programming language and its rules is still C++, that was definitely some interesting fact that I found creating my videogame, and makes sense, because it is a game engine, all C++ was modified to facilitate the programmer from doing a lot of work and focus more in the work of developing the game itself, of course, the engine won’t dismiss the fact that we have to handle memory ourselves, that’s one of the main points of C++.