**C++ QUESTIONS WITH SHORT ANSWERS**

**1. WHAT IS C++?**

**C++ is an object-oriented programming language, which was developed by Bjarne Stroustrup.**

**2. WHAT ARE THE DIFFERENCES BETWEEN C AND C++?**

**C is a procedure-oriented programming language and does not have some features that C++ supports such as function and operator overloading, namespaces, virtual and friend functions, data hiding etc. Unlike C, C++ provides a high level of resources and memory management.**

**3. DATA TYPES IN C++?**

**We divide C++ data types into 3 groups:**

**3.1 PRIMITIVE TYPES:**

**Integer [(short int) 2 - (int) 4 - (long int) 8 bytes]: Whole numbers.**

**Character [(char) 1 byte]: Single-character.**

**Boolean [(bool) 1 byte]: True (1) or false (0).**

**Floating-point [(float) 4 bytes]: Numbers with decimals.**

**Double floating-point [(double) 8 bytes]: Numbers with decimals.**

**Void: Empty value, only pointers can be declared as void.**

**Wide Character [(wchar\_t) 2 bytes]: Stores more than one character.**

**3.2 DERIVED TYPES:**

**Function: It is derived from basic data types and performs some operations.**

**Pointer: It holds the memory address of another variable.**

**Array: It stores multiple values which are the same type.**

**Reference: It is the alias of an already existing variable.**

**3.3 USER-DEFINED TYPES:**

**Class: It is the blueprint of an object.**

**Structure: It is used to group several related variables.**

**Enumeration: It contains a fixed set of constants.**

**Union: Similar to “Structure” but its variables share the same memory location.**

**Typedef: It is simply giving another name to an existing data type.**

**4. WHAT IS NAMESPACE DIRECTIVE?**

**Namespace prevents name collision, especially when your code includes multiple libraries. The entire namespace can be declared as well as all the names in a namespace can be used individually with the “using” directive. For example;**

**using namespace std; or using std::endl;**

**5. WHAT ARE THE DIFFERENCES BETWEEN POINTER AND REFERENCE?**

**Unlike pointers(\*p), a reference(&r)**

* **must be declared and initialised at the same time,**
* **is not allowed to re-assign,**
* **does not need extra space on the stack, it shares its original variable memory location,**
* **cannot be NULL, it has to be a valid value,**
* **cannot be used with the arithmetic operators,**
* **is just an alias of an existing variable, not its memory address,**

**6. WHAT IS PASSING BY (VALUE, REFERENCE, POINTER, CONST REFERENCE, CONST POINTER)?**

**Passing by value means that a copy of the actual variable is passed into the function as a parameter, the actual variable does not get affected by the function.**

**Passing by reference means that the actual variable is passed into the function as a parameter and this variable can be modified when the function is called because we deliberately refer to the actual one by using reference. If we use “const” reference, the function cannot change the variable's value.**

**Passing by pointer is similar to passing by reference, so the pointer’s value can be modified permanently in the function. Unlike reference, we also can re-assign our pointer in the function but it is passed by value, which means that our pointer is copied before passing into the function, so the actual one, which is outside of the function, does not get affected. If we use “const” pointer, the function cannot change the memory address that the pointer points to.**

**6. WHAT IS AN INLINE FUNCTION?**

**If a function is inline, the compiler changes the normal calling process and that reduces the execution time and the program size because a separate location is not allocated on the memory for controlling the function code. Inline keyword is a suggestion, which means that the compiler can ignore it.**

**7. WHAT ARE STATIC\_CAST AND REINTERPRET\_CAST?**

**Static\_cast is used to convert one type into another. It is a compile-time conversion.**

**Reinterpret\_cast is used to the conversion of unrelated types (pointer to pointer, pointer to function, pointer to integral types etc.). It should not be used unless you do not have another option as it is inherently an unsafe cast type.**

**8. WHAT IS THE DIFFERENCE BETWEEN PREFIX AND POSTFIX?**

**If we use prefix, the value is incremented/decremented before being evaluated; this process works in reverse for postfix.**

**There is no difference between them, if they are used alone. (i++;) and (++i;) have the same effect.**

**9. WHAT ARE THE STACK AND THE HEAP?**

**Local variables and function parameters are stored contiguously on the stack and the allocation is done automatically by the compiler. The stack memory can also be described as temporary storage.**

**The heap is a dynamic memory which allows programmers to allocate/de-allocate memory manually and it also has more storage space than the stack.**

**10. WHAT IS THE DIFFERENCE BETWEEN NEW AND MALLOC()?**

**Both new and malloc() are used to allocate space on the heap. New is an operator which calls the constructor; however, malloc() is a C standard library function. In addition, new is faster than malloc().**

**11. WHAT ARE ACCESS MODIFIERS AND WHAT ARE THEY USED FOR?**

**Access modifiers (public, private, protected) indicate the accessibility of class members.**

**If a member is public, it is accessible to other classes or functions.**

**Private means that members are accessible only from inside the same class or friend functions.**

**Protected is similar to private; in addition, protected members can be accessed by the derived class members.**

**If a member’s accessibility is not specified, it is private by default.**

**12. WHAT IS THE DIFFERENCE BETWEEN STRUCT AND CLASS?**

**We can simply say that struct is a value type whereas class is a reference type. Hence, struct members are public and are stored on the stack while class members are stored on the heap. Unlike struct, class have 3 access modifiers which are public, protected and private.**

**13. WHAT ARE THE ADVANTAGES AND DISADVANTAGES OF USING POINTERS AS A CLASS MEMBER?**

**The advantages of using a pointer as a class member are lazy initialisation (to delay the creation of an object until it is used), reduction of header dependencies and giving more control to programmers.**

**The disadvantages of using a pointer as a class member are having to write our copy constructor and operators and that means more work, using a pointer can easily cause memory leak and accessing a pointer data member is slower than a non-pointer.**

**14. WHEN SHOULD WE DEFINE A DESTRUCTOR?**

**When we need to perform an action other than destructing class members, we define our destructor. For example, freeing memory.**

**When we destruct an object through a base class pointer, the destructor must be defined as virtual.**

**15. WHAT IS THE DIFFERENCE BETWEEN SHALLOW AND DEEP COPY?**

**Shallow copy can simply be defined as copying all data members of an object to create a new object but copied members still refer to the same location as its original; however, when we use deep copy, the object is copied and then located in a new allocated space in the memory. Unlike shallow copy, a deep copied object is independent of its origin.**

**16. WHAT ARE BASE CLASS AND ABSTRACT BASE CLASS?**

**A base class is the parent class, from which the other classes are derived and inherit members. An abstract base class is the base class which contains at least one pure virtual function.**

**17. WHAT IS RUN-TIME TYPE INFORMATION?**

**It is a mechanism that allows us to determine the type of an object at run-time. We can use dynamic\_cast (converts base class pointer to derived class pointer-downcasting) operator, type\_info (holds the type of an object) class and typeid (returns the type of an object) operator in this concept.**

**The base class must have one virtual function (run-time polymorphism) for the dynamic\_cast conversion to be successful.**

**18. WHEN SHOULD NOT WE CALL VIRTUAL FUNCTION?**

**Virtual functions should not be called in a constructor or a destructor because they are called in a certain order because of the nature of the inheritance, which means that virtual functions are not virtual during the construction and destruction processes.**

**19. WHAT IS THE DIFFERENCE BETWEEN CLASS AND TYPENAME IN TEMPLATE?**

**There is no difference between “class” and “typename” unless we use nested templates. We must use “class” in nested templates.**

**20. WHAT IS TEMPLATE-METAPROGRAMMING?**

**Template-metaprogramming is a technique that the program performs computations at compile-time in other words it runs at compile-time, to write a metaprogram, we use compile-time types and variables (const, constexpr) instead of run-time features (RTTI, virtual).**

**21. WHAT IS A METAFUNCTION?**

**A metafunction is actually a class/struct that can also return a value just as regular functions do and it is commonly used in the context of template metaprogramming.**

**For instance:**

template <int num1, int num2>

struct Add

{

    static constexpr int value = num1 + num2;

};

**“type\_traits” is a standard c++ library that provides many metafunctions.**

**22. WHAT IS EXCEPTION HANDLING?**

**When the program runs, unexpected problems may occur, in which case we use the exception handling technique to prevent the program from terminating with an error.**

**Exception handling consists of some keywords such as try, throw and catch:**

**“try” is a block of code which includes “throw” keywords, which means that the program may throw an exception inside the try block.**

**“catch” is used to catch the exception which is thrown and includes the next step which will be done if an error occurs.**

**23. WHAT IS ITERATOR?**

**An iterator is an object which points to elements in containers, thus they are used to move through the contents of containers.**

**24. WHAT IS LAMBDA EXPRESSION?**

**A lambda is a kind of function object which can capture variables in scope (a closure). It has value[=], reference [&] and both [v1, &v2] capture modes.**

**25. WHAT ARE SMART POINTERS?**

**Unlike raw pointers, smart pointers destroy themselves where they are out of scope so that memory leaks do not occur, in other words, programmers do not have to bother freeing memory.**

**There are 3 types of smart pointers:**

**unique\_ptr: It allows only one owner of the object.**

**shares\_ptr: It allows more than one owner that point to the object. Reference counter, which is accessed by using use\_count() method, includes the number of owners.**

**weak\_ptr: It is used with shares\_ptr but not added to reference counter.**

**26. WHAT IS THREAD?**

**Threads are parts of the process. We can create thread objects by using std::thread and pass callable objects(function pointers, function objects, lambdas, member functions) into them.**

**27. WHAT ARE SINGLE-THREADED AND MULTI-THREADED PROGRAMMING?**

**If threads execute in order, it is called single-threaded programming, which means that one thread waits to finish another thread before it runs; in contrast, detached threads run independently from other threads, if we divide one program into two or more separate parts by using detached threads, it is called multi-threaded programming.**

**We use join() and detach() methods to determine which way a thread works.**

**28. WHAT ARE MUTEX AND DEADLOCK?**

**Mutex prevents separate threads which share the same data resources from accessing the data at the same time.**

**In concurrent computing, if each thread waits for another waiting process to proceed, it is called deadlock because the program cannot make any progress.**

**29. WHAT IS RAII (RESOURCE ACQUISITION IS INITIALIZATION)?**

**It is a programming idiom which means that a resource is tied to the lifetime of the object, in other words, when the object goes out of the scope, its resources are released.**

**30. WHAT IS GENERIC PROGRAMMING?**

**Generic programming is to write algorithms which can work with all data types, because of which a generic program provides flexibility and reusability.**

**31. WHAT ARE TIME AND SPACE COMPLEXITY?**

**Time complexity helps us to understand the relationship between the runtime and input size of an algorithm.**

**Space complexity helps us to understand the relationship between the memory usage and input size of an algorithm.**

**Both are expressed with Big O notation. Best to worst time complexity (n: input number): Constant Time O(1), Linear Time O(n), Logarithmic Time O(log n), Quadratic Time O(n^2), Exponential Time O(2^n), etc…**

**We choose the best solution for our problem by considering the time and space complexity of algorithms so that we can write more efficient programs.**