**Interviewing Questions**

**C Programming**:

1. **Why C does not support function overloading**?

Because the compiler for C doesn’t support it

We can achieve the smilimar functionality in C indirectly by using void\* type of pointer

For example:

int casting\_type(void\* arg, int arg2);

this function would have 2 arguments:

first is the value for casting

second is the type for casting (which we can define)

1. **Recursion**:

This is a technique of making a function call itself.

This technique provides a way to **break complicated problems down into simple problems** which are easier to solve

**C++ Programming**:

1. **What is function overloading?**

Function overloading is a feature of a programming language that allows one to have many functions with same name but with different signatures/argurments.

This feature is present is most of the Object Oriented Languages such as C++ and Java.

1. **OOP:**
   1. **Class definitions**: Basic building blocks OOP and a single entity which has data and operations on data togethers.
   2. **Object**: The instance of a class which are used in real functionality – its variables and operations
   3. **Abstraction**: Specifying what to do but not how to do; a flexible feature for having a overall view of an object’s functionality.
   4. **Encapsulation**: Binding data and operations of data together in a single unit.

-> Easy to understand is – we does not allow other access the data directly and we just get or set data through the method or functionalities Or basically, We hide data from outside.

-> Example for C++: we have methods { **set**(), **get**() value } to manipulate with data and the way for encapsulation is using access specifiers {**public**, **protected** and **private**}.

* 1. **Role of access specifier**: Determine the access of variables and functions in class.

The data members should be labeled as private using the private access specifiers

The member function which manipulates the data members should be labeled as public using the public access specifier

* 1. **Inheritance**: Reusability and extension of existing classes
  2. **Polymorphism**:

Multiple definitions for a single name - functions with same name with different functionality; saves time in investing many function names Operator and Function overloading

1. **Memory Architecture**:

|  |
| --- |
| **high address**  **Low address**  **Command-line arguments  and environtment variables**  **stack**  **heap**  **Initialized zero by exec**  **Uninitialized data (bss)**  **Read from program file by exec**  **Initialized data**  **text**  **Low address** |

Text Segment: also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.

the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

Initialized Data Segment: Initialized data segment, usually called simply the Data Segment.

A data segment is **a portion of the virtual address space of a program**, which **contains** the **global variables and static variables that are initialized by the programmer**.

The values of the variables can be altered at run time.

Uninitialized Data Segment: Uninitialized data segment often called the “**bss - block started by symbol**” segment.

Data in this segment **is initialized by the kernel to arithmetic 0** **before** the program **starts executing uninitialized data** starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

Stack: The stack area traditionally adjoined the heap area and grew in the opposite direction. when the stack pointer met the heap pointer, free memory was exhausted (we called stackoverflow).

The stack area contains the program stack, a LIFO structure, typically located in the higher parts of memory.

A **“stack pointer” register** tracks the **top of the stack**; it is adjusted each time a value is “pushed” onto the stack. **The set of values pushed for one function call** is termed a “**stack frame**”; A stack frame consists at minimum of a return address.

Heap: Heap is the segment where dynamic memory allocation usually takes place.

The heap area begins at the end of the BSS segment and grows to larger addresses from there.

The Heap area is managed by malloc, realloc, and free, which may use the brk and sbrk system calls to adjust its size.

The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

1. Lambda:

a lambda expression—often called a lambda—is **a convenient way of defining an anonymous function object** (a closure) right at the location where it's invoked or passed as an argument to a function.

Typically lambdas are used to encapsulate a few lines of code that are passed to algorithms or asynchronous functions.

Syntax: [=] () {}

* Capture clause:

1. Thread and Multi-Threads:
2. Process: