Greedy Alignment, Padding, and Compiler Memory Layout in C++

1. What is Greedy Alignment?

Greedy alignment is a compiler behavior used to place variables and data in memory while satisfying alignment requirements.
It applies to all data types — including individual variables, arrays, and structs.

Each data type requires alignment based on its size or the target architecture. For example:

- int typically requires 4-byte alignment
- double often requires 8-byte alignment

So if a variable needs to start at a memory address that's a multiple of its alignment, but the next free address isn't aligned properly, the compiler inserts padding bytes.

This strategy is called "greedy" because the compiler places each item in the declared order, adding only the necessary padding without rearranging anything for optimization.

2. Examples of Greedy Alignment

Example 1: Integer Alignment

```
char a;   // occupies 1 byte
int b;    // requires 4-byte alignment
```

In memory layout:

- a is placed at offset 0
- Compiler adds 3 bytes padding
- b is placed at offset 4 to 7

b is now correctly aligned at offset divisible by 4.

Example 2: Double Alignment

```
char a;
double d;   // requires 8-byte alignment
```

Memory layout:

- a at offset 0

- Compiler adds 7 bytes padding
- d is placed at offset 8 to 15

Now, d is aligned at an address divisible by 8.

Example 3: Struct Alignment

```
struct Example {
  char a;   // offset 0
  int b;    // needs 4-byte alignment
  char c;   // offset 8
};
```

Layout:

- a at offset 0
- 3 bytes padding → b at offset 4
- b occupies offset 4–7
- c at offset 8
- Compiler adds 3 bytes padding at the end
- Total struct size = 12 (a multiple of the largest alignment = 4)

This extra padding ensures correct alignment when an array of such structs is created.

Example 4: Array of Structs

```
struct Demo {
  char a;
  int b;
};
```

Demo arr[2];

Each struct instance layout:

- a at offset 0
- 3 bytes padding
- b at offset 4
- Total size = 8 bytes

Now:

- arr[0] occupies offset 0–7
- arr[1] starts at offset 8 — correctly aligned for its int member at offset 4

Without the padding at the end of each struct, b in the second struct would begin at offset 5, which breaks its 4-byte alignment requirement.

3. Summary

- Greedy alignment is a memory layout strategy where the compiler places data in declared order.
- Padding is inserted before or after data to satisfy alignment rules.
- This applies to variables, structs, and arrays.
- The total struct size is also aligned to ensure consistent layout in arrays.
- Proper alignment ensures performance, correctness, and hardware compatibility.