Union

- Union is user defined data-type.
- Like struct it is collection of similar or non-similar data elements.
- All members of union share same memory space i.e. modification of an member can affect others too.
- Size of union = Size of largest element
- When union is initialized at declaration, the first member is initialized.
- Application:
  - System programming: to simulate register sharing in the hardware.
  - Application programming: to use single member of union as per requirement.

```
union test {
    int num;
    char arr[2];
}u = { 65 };
printf("%d, %c, %s\n", u.num, u.arr[0], u.arr);
```

Word aligned memory access

- refers to storing and accessing data where the starting address of a data element (like an integer or a structure) is a multiple of its size, making the access efficient for the processor.

- Unaligned access occurs when data isn't at a word boundary, forcing the system to split the access across multiple memory words, which is slower and can even cause exceptions on some architectures.

- Alignment speeds up data transfer by allowing the CPU to fetch an entire data element in a single memory cycle.

- What is Word Alignment?

  - Word Boundary:
    - An address is considered "word-aligned" if it's a multiple of the word size. For example, on a system with 4-byte integers (words), an aligned address would be 0x1000, 0x1004, 0x1008, and so on.
  - Data Elements:
    - When a data type is aligned, its storage begins at an address that is a perfect multiple of its size.
  - CPU Memory Operations:
    - Modern CPUs are designed to access memory efficiently in chunks called words.

- Why is it Important?

  - Performance:
    - Aligned access allows the CPU to retrieve the entire data element in one memory read operation.
  - Efficiency:

        - Unaligned data requires the CPU to perform multiple memory reads, fetch different parts of the data from separate memory locations, and then reassemble them, which is slow and resource-intensive.
    - Hardware Support:
        - Many CPU architectures are optimized for aligned data access, and some may not support unaligned access at all.

- Consequences of Unaligned Access
    - Performance Degradation:
        - The CPU has to execute multiple read/write operations to gather the unaligned data, leading to significant performance penalties.
    - Exceptions/Faults:
        - On some processor architectures, attempts to perform unaligned memory accesses can result in hardware exceptions (like a HardFault on Cortex-M0) or segmentation faults, causing the program to terminate.

- Example
    - Imagine a 4-byte integer needs to be stored.
    - Aligned Access:
        - If the memory starts at address 0x1000, the integer will be stored from 0x1000 to 0x1003, because 0x1000 is a multiple of 4.
    - Unaligned Access:
        - If the data needs to start at address 0x1001, it is unaligned
        - The CPU would need to:
            - Read the memory from 0x1000 to 0x1003 to get the first 3 bytes.
            - Read the memory from 0x1004 to 0x1007 to get the last 1 byte.
            - Combine these pieces into the complete integer.

- In essence, word alignment ensures data is positioned in memory in a way that matches the CPU's natural fetch boundaries, optimizing performance and preventing errors.

File IO

- File is collection of data and information on storage device.
- Each file have data (contents) and metadata (information).
- File IO can enable read/write file data.
- File Input Output
    - Low Level File IO
        - Use File Handle.
    - High Level File IO
        - Use File Pointer.
        - Formatted (Text) IO
            - fprintf(), fscanf()
        - Unformatted (Text) IO
            - fgetc(), fputc(), fgets(), fputs()
        - Binary File IO
            - fread(), fwrite()

- File must be opened before read/write operation and closed after operation is completed.
  - FILE * fp = fopen("filepath", "mode"); – to open the file
    - File open modes:
      - w: open file for write. If exists truncate. If not exists create.
      - r: open file for read. If not exists, function fails.
      - a: open file for append (write at the end). If not exists create.
      - w+: Same as "w" + read operation.
      - r+: Same as "r" + write operation.
      - a+: Same as "a" + append (write at the end) operation.
    - File can be opened as text file (default or suffix "t") or binary (suffix "b").
    - Return FILE* when opened successfully, otherwise return NULL.
  - fclose(fp);
    - Close file and release resources.
- Character IO
  - fgetc(), fputc()
- String (Line) IO
  - fgets(), fputs()
- Formatted IO
  - fscanf(), fprintf()
- Binary (record) IO
  - fread(), fwrite()
- File position
  - fseek(), ftell()