Agenda

- 2D Array
- enum
- multiple files
- Hierarchy and its type.
- Association
- Inheritance
- ~~Type of Inheritance~~
- ~~Diamond problem~~
- ~~Virtual base class~~
- ~~Mode of Inheritance~~

Multi Dimension Array

```
// multi dimension array
int main()
{
    int arr[][3] = {10, 20, 30, 40, 50, 60};
    // int arr[2][3] = {10, 20, 30, 40, 50, 60};
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 3; j++)
            cout << arr[i][j] << ",";
    cout << endl;
    return 0;
}

// multi dimension array of ptrs (Dynamic memory allocation)
int main()
{
    int *arr[2][3];
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 3; j++)
            arr[i][j] = new int(i + j);

    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 3; j++)
            cout << arr[i][j] << endl;

    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 3; j++)
            delete arr[i][j];
    return 0;
}


// multi dimension array with Dynamic memory allocation
int main2()
{
```

```
    int **arr = new int *[2];
    arr[0] = new int[3]{10, 20, 30};
    arr[1] = new int[3]{40, 50, 60};

    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 3; j++)
            cout << arr[i][j] << ",";
    cout << endl;
    delete[] arr[0];
    delete[] arr[1];
    delete[] arr;
    return 0;
}
```

enum

- Enumeration (Enumerated type) is a user-defined data type that can be assigned some limited values. These values are defined by the programmer at the time of declaring the enumerated type.
- Enums provide a way to define symbolic names for sets of integers, making the code more readable and maintainable.

```
#include <iostream>

// Define an enum named Color
enum Color {
    RED,    // 0
    GREEN,  // 1
    BLUE    // 2
};

int main() {
    // Declare a variable of type Color
    Color myColor = GREEN;

    // Check the value of myColor
    if (myColor == GREEN) {
        cout << "The color is green." << endl;
    } else {
        cout << "The color is not green." << std::endl;
    }
    return 0;
}
```

Modularity (Multiple Files)

- "/usr/include" directory is called standard directory for header files.
- It contains all the standard header files of C/C++

- If we include header file in angular bracket (e.g #include<filename.h>) then preprocessor try to locate and load header file from standard directory only(/usr/include).
- If we include header file in double quotes (e.g #include“filename.h") then preprocessor try to locateand load header file first from current project directory if not found then it try to locate and load from standard directory.

```
// Header Guard
  #ifndef HEADER_FILE_NAME_H
  #define HEADER_FILE_NAME_H
    //TODO : Type declaration here
  #endif
```

Hierarchy

- It is a major pillar of oops.
- Level / order / ranking of abstraction is called hierarcy.
- Its main purpose is to achive reusability.
- Advantages of reusability:
    1. To reduce develoers efforts.
    2. To reduce development time and development cost.

Types of Hierarcy:

1. Has-a/Part-of Association/Containment
2. Is-a/Kind-of Inheritance/Generalization
3. Use-a Dependancy
    - This hierarchy represents how classes depend on each other.
    - Dependencies occur when one class relies on another class but does not own or control its lifetime.
    - For example, if Class A uses Class B as a method parameter or local variable, there's a dependency between A and B.
4. Creates-a Instantiation
    - This can often be seen in factory design patterns or in scenarios where one class encapsulates the creation logic of another class.

Association

- If has-a relationship exist between two types then we should use association.
- Example:
    1. Room has-a wall
    2. Room has-a chair
    3. Car has-a engine
    4. Car has-a music player
    5. Department has-a faculty
    6. Human has-a heart
- If object is part-of / component of another object then it is called association.

- Composition and aggregation are specialized form of association.
- If we declare object of a class as a data member inside another class then it represents association.

```
class Engine{
};
class Car{
    private:
    Engine e; //Association
};
int main( void ){
    Car car;
    return 0;
}

//Dependant Object : Car Object
//Dependancy Object : Engine Object
```

1. Composition

- If dependancy object do not exist without Dependant object then it represents composition.
- Composition represents tight coupling.
- If we create object of dependency class as data member inside the dependent class it represents composition.

2. Aggegration

- If dependancy object exist without Dependant object then it represents Aggregation.
- Aggregation represents loose coupling.