Tutorial 19 -Function Overloading in C++

Definition:

Function overloading allows multiple functions with the same name but different parameters (type, number, or order) within the same scope.

Examples

1. Sum Function Overloading

- **Code**:

```
int sum(float a, int b) {
    cout << "Using function with 2 arguments" << endl;
    return a + b;
}
int sum(int a, int b, int c) {
    cout << "Using function with 3 arguments" << endl;
    return a + b + c;
}
int main() {
    cout << "The sum of 3 and 6 is " << sum(3, 6) << endl;
    cout << "The sum of 3, 7, and 6 is " << sum(3, 7, 6) << endl;
    return 0;
}

```

- **Explanation**:
  - Two functions with the same name `sum` but different parameters:
    - 1st: `sum(float a, int b)` adds two numbers.
    - 2nd: `sum(int a, int b, int c)` adds three numbers.
  - Function is selected based on the number and type of arguments.
- **Output**:

```
Using function with 2 arguments
The sum of 3 and 6 is 9
Using function with 3 arguments
The sum of 3, 7, and 6 is 16

```

---

2. Volume Function Overloading

- **Code**:

```
// Cylinder
int volume(double r, int h) {
    return 3.14 * r * r * h;
}
// Cube
int volume(int a) {
    return a * a * a;
```

```
}
// Rectangular Box
int volume(int l, int b, int h) {
    return l * b * h;
}
int main() {
    cout << "The volume of cuboid of 3, 7, and 6 is " << volume(3, 7, 6) << endl;
    cout << "The volume of cylinder of radius 3 and height 6 is " << volume(3, 6) << endl;
    cout << "The volume of cube of side 3 is " << volume(3) << endl;
    return 0;
}

```

- **Explanation**:
  - Three `volume` functions:
    - **Cylinder**: Takes radius `r` and height `h`.
    - **Cube**: Takes side `a`.
    - **Rectangular Box**: Takes length `l`, breadth `b`, and height `h`.
  - Function is selected based on the number and type of arguments.
- **Output**:

```
The volume of cuboid of 3, 7, and 6 is 126
The volume of cylinder of radius 3 and height 6 is 169.56
The volume of cube of side 3 is 27

```

Key Points to Note

1. Function name remains the same, but parameter types, count, or sequence differ.
2. The compiler determines which function to call based on arguments provided.
3. Examples:
   - **Sum Functions**: Different argument counts.
   - **Volume Functions**: Different argument types and counts.
4. Overloading enhances code readability and reusability.
5. Overloading cannot differ solely by return type.

Short Notes

1. **Function Overloading**:
   - Multiple functions with the same name but different parameters.
   - Differentiated by type, count, or order of arguments.
2. **Examples**:
   - **Sum**: `sum(a, b)` and `sum(a, b, c)`.
   - **Volume**: Cylinder `volume(r, h)`, Cube `volume(a)`, Box `volume(l, b, h)`.
3. **Benefits**: Improves code clarity and flexibility.
4. **Compiler Role**: Selects the correct function based on arguments at compile time.