Notes on Object-Oriented System Design

Unit IV: C++ Basics

Overview:

- C++ is a general-purpose programming language with object-oriented features.
- **Program Structure:** Contains headers, namespaces, main function, and additional functions/classes.

Namespaces:

- Used to organize code and prevent name conflicts.
- Example:

```cpp
#include <iostream>
using namespace std;

namespace Math {
    int add(int a, int b) {
        return a + b;
    }
}

int main() {
    cout << Math::add(3, 4); // Output: 7
    return 0;
}
```

```

#### Identifiers and Variables:

- **Identifiers:** Names used to identify variables, functions, etc.
- **Variables:** Store data, e.g., `int age = 20;`.

#### Constants and Enum:

- **Constants:** Immutable values declared with `const` or `#define`.
- **Enum:** Represents a list of named integer constants.

```cpp
enum Day { Monday, Tuesday, Wednesday };
Day today = Monday;
```

Operators and Typecasting:

- Operators: Arithmetic, logical, bitwise, relational, etc.
- Typecasting: Convert one type to another.

```cpp
float f = 10.5;
int i = (int)f; // Explicit typecasting
```

Control Structures:

- Includes `if`, `else`, `switch`, `for`, `while`, and `do-while` loops.

```cpp
for (int i = 0; i < 5; i++) {
    cout << i << " ";
```
```

}

```

---

### C++ Functions

#### Simple Functions:

- Blocks of reusable code.

```cpp
int add(int a, int b) {
    return a + b;
}
```

Call and Return by Reference:

- **Call by Reference:** Passing variables by reference.

```cpp
void increment(int &x) {
    x++;
}
```

Inline Functions:

- Substitutes the function call with its body during compilation.

```cpp
inline int square(int x) {

```
    return x * x;
}
```

Macros vs Inline Functions:

- **Macros:** Preprocessor directives, e.g., `#define SQUARE(x) (x * x)`.
- Inline functions are safer and offer type checking.

Function Overloading and Default Arguments:

- Overloading: Same function name, different parameters.

```cpp
int add(int a, int b) {
    return a + b;
}
float add(float a, float b) {
    return a + b;
}
```

- Default arguments: Predefined values for parameters.

```cpp
void greet(string name = "Guest") {
    cout << "Hello, " << name;
}
```

Friend Functions:

- Allow external functions to access private members.

```cpp
class Box {
    private:
        int length;
    public:
        Box(int l) : length(l) {}
        friend void printLength(Box);
};

void printLength(Box b) {
    cout << b.length;
}
```

Virtual Functions:

- Enable runtime polymorphism.

```cpp
class Base {
    public:
        virtual void show() { cout << "Base class"; }
};

class Derived : public Base {
    public:
        void show() override { cout << "Derived class"; }
};
```

Unit V: Objects and Classes

Basics of Object and Class in C++:

- **Object:** Instance of a class.
- **Class:** Blueprint for creating objects.

```cpp
class Car {
    private:
        string color;
    public:
        Car(string c) : color(c) {}
        void displayColor() { cout << color; }
};
```

Private and Public Members:

- **Private:** Accessible only within the class.
- **Public:** Accessible from outside the class.

Static Data and Function Members:

- Shared across all instances of a class.

```cpp
class Counter {
    public:
```

```
        static int count;
        Counter() { count++; }
};
int Counter::count = 0;
```

Constructors and Destructors:

- **Constructor:** Initializes objects.
- **Destructor:** Cleans up resources.

```cpp
class Example {
    public:
        Example() { cout << "Object created!"; }
        ~Example() { cout << "Object destroyed!"; }
};
```

Operator Overloading:

- Redefine operators for custom types.

```cpp
class Complex {
    public:
        int real, imag;
        Complex operator + (Complex const &obj) {
            Complex res;
            res.real = real + obj.real;
            res.imag = imag + obj.imag;
```

```
            return res;
        }
};
```

... (content truncated for this example)