Unit II

**Generic Programming-Function and class templates, Inheritance basics, base and derived classes,
inheritance types, base class access control, runtime polymorphism using virtual functions, abstracts
classes, streams I/O. Algorithms, performance analysis-time complexity and space complexity, review
of basic data structures-The list ADT, Stack ADT, Queue ADT.**

Generic Programming

1. **Function and Class Templates**:
    - **Reusable Code**: Templates for functions and classes to work with any data type.
    - **Parameterized Types**: Allow for the creation of generic algorithms.

Inheritance Basics

1. **Base and Derived Classes**:
    - **Inheriting Properties**: Derived classes inherit attributes and behaviors from base classes.
    - **Hierarchy**: Establishing a relationship between classes.
2. **Inheritance Types**:
    - **Single Inheritance**: Derived from a single base class.
    - **Multiple Inheritance**: Derived from multiple base classes.
    - **Hierarchical Inheritance**: Multiple derived classes from a single base class.
3. **Base Class Access Control**:
    - **Public, Protected, Private Inheritance**: Controlling access to base class members in derived classes.
4. **Runtime Polymorphism using Virtual Functions**:
    - **Dynamic Binding**: Resolving function calls at runtime.
    - **Virtual Functions**: Functions declared in base class and overridden in derived classes.
5. **Abstract Classes**:
    - **Pure Virtual Functions**: Classes that contain at least one pure virtual function.
    - **Cannot be Instantiated**: Used as a base for other classes.

Streams I/O

1. **Input/Output Streams**:
    - **iostream Library**: Handling input and output operations in C++.
    - **cin, cout, cerr**: Standard streams for input, output, and error messages.

Algorithms and Performance Analysis

1. **Algorithms**:
    - **Generic Methods**: A step-by-step procedure for solving computational problems.
    - **Standard Template Library (STL)**: Collection of algorithms and data structures.
2. **Performance Analysis**:
    - **Time Complexity**: Measure of the amount of time an algorithm takes to run.
    - **Space Complexity**: Measure of the amount of memory an algorithm uses.

Basic Data Structures

1. **List ADT**:
    - **Sequential Collection**: Collection of elements with linear order.
    - **Operations**: Insertion, deletion, traversal, etc.
2. **Stack ADT**:
    - **Last In, First Out (LIFO)**: Data structure where the last element added is the first one to be removed.
    - **Operations**: Push, pop, peek, etc.
3. **Queue ADT**:
    - **First In, First Out (FIFO)**: Data structure where the first element added is the first one to be removed.
    - **Operations**: Enqueue, dequeue, front, rear, etc.

1. **Generic Programming: Function and Class Templates**

Function Templates:
Function templates allow writing generic functions that work with any data type.

cpp
Copy code
template <class T>

```
T maxValue(T a, T b) {
return (a > b) ? a : b;}

int maxInt = maxValue(5, 10); // maxInt = 10
float maxFloat = maxValue(5.5f, 10.7f); // maxFloat = 10.7
```

Class Templates:

Class templates allow defining generic classes.

```
template <class T>
class Pair {
private:
T first, second;

public:
Pair(T a, T b) : first(a), second(b) {}

T getFirst() { return first; }
T getSecond() { return second; }
};

Pair<int> intPair(5, 10);
int firstValue = intPair.getFirst(); // firstValue = 5
```

2. Inheritance Basics, Base and Derived Classes

Base and Derived Classes:

Inheritance allows a new class to inherit properties and behavior from an existing class.

```
class Animal {
public:
void makeSound() {
cout << "Some generic sound\n";
}
};

class Dog : public Animal {
public:
void makeSound() {
cout << "Woof!\n";
}
```

```
};

Dog myDog;
myDog.makeSound(); // Output: Woof!
```

3. Inheritance Types, Base Class Access Control

Inheritance Types:

Public: Public members of the base class become public in the derived class.

Protected: Public and protected members of the base class become protected in the derived class.

Private: Public and protected members of the base class become private in the derived class.

```
class Base {
  public:
int publicVar;
  protected:
int protectedVar;
  private:
int privateVar;
};

class Derived : public Base {
// Access specifiers control visibility of inherited members
};
```

4. Runtime Polymorphism using Virtual Functions

Virtual Functions:

Virtual functions enable polymorphic behavior, allowing functions to be overridden in derived classes.

```
class Shape {
  public:
virtual void draw() {
cout << "Drawing a shape\n";
}
};

class Circle : public Shape {
  public:
void draw() override {
cout << "Drawing a circle\n";
}
```

```
};

Shape* shapePtr = new Circle();
shapePtr->draw(); // Output: Drawing a circle
```

5. Abstract Classes

Abstract Classes:

Abstract classes contain at least one pure virtual function and cannot be instantiated.

```
class AbstractShape {
public:
virtual void draw() = 0; // Pure virtual function
};

class ConcreteShape : public AbstractShape {
public:
void draw() override {
cout << "Drawing a concrete shape\n";
}
};

AbstractShape* abstractPtr = new ConcreteShape();
abstractPtr->draw(); // Output: Drawing a concrete shape
```

6. Streams I/O

Streams Input/Output:

C++ streams allow reading from and writing to various sources like standard input/output, files, etc.

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
int number;
cout << "Enter a number: ";
cin >> number;
cout << "You entered: " << number << endl;
```

```
ofstream file("output.txt");
file << "Writing to a file using C++ streams.\n";
file.close();
return 0;
}
```

7. Algorithms, Time Complexity, and Space Complexity

Time Complexity and Space Complexity:

Understanding algorithm efficiency in terms of time and space.

```
// Example of a simple linear search algorithm
int linearSearch(int arr[], int n, int target) {
for (int i = 0; i < n; ++i) {
if (arr[i] == target) {
return i; // Element found
}
}
return -1; // Element not found
}
```

8. Review of Basic Data Structures

Basic Data Structures:

List ADT: A collection of elements with sequential access.

Stack ADT: Last In, First Out (LIFO) data structure.

Queue ADT: First In, First Out (FIFO) data structure.

```
// Example of a stack implementation using C++ STL
#include <stack>

int main() {
stack<int> myStack;
myStack.push(5);
myStack.push(10);
int topElement = myStack.top(); // topElement = 10
myStack.pop();
return 0;
}
```