C++ Concepts – Assignment 2

1. Array of Objects

Definition: An array of objects stores multiple instances of the same class together, allowing processing using loops instead of declaring them one by one.

Example: Storing multiple rectangles in an array and calculating their areas.

```
class Rectangle { // Class definition starts
   int length, breadth; // Statement
public: // Access specifier
   void setData(int l, int b) { length = l; breadth = b; } // Initialization
or assignment
   int area() { return length * breadth; } // Statement
};

int main() { // Function definition
   Rectangle rect[3]; // Array of 3 objects // Statement
   rect[0].setData(5, 4); // Statement
   rect[1].setData(6, 3); // Statement
   rect[2].setData(7, 2); // Statement
}
```

Constructor

Definition: A constructor is a special member function that is automatically called when an object is created.

- It has the same name as the class and no return type.
- Constructors are used to initialize data members.
- They can be default (no parameters) or parameterized (take arguments).
- Constructors can also be overloaded to provide multiple ways to initialize objects.

Example: Used to initialize 'length' in the Box class

```
class ClassName { // Class definition starts
public: // Access specifier
  ClassName() {  // Function definition
    // default constructor body
  }
  ClassName(int x) {  // Function definition
    // parameterized constructor body
  }
};
```

---

2. Passing and Returning Objects

Definition: Objects can be passed to functions by value (copy) or by address (pointer). They can also be returned from functions.

Example: Changing the length of a box using both pass-by-value and pass-by-address.

```
class Box { // Class definition starts
  int length; // Data Member
public: // Access specifier
  Box(int l=0) { length = l; } // Initialization or assignment
  void setLength(int l) { length = l; } // Initialization or assignment
  int getLength() { return length; } // Statement
};

// Pass by Value
Box changeValue(Box b) { // Function definition
  b.setLength(20); // Statement
  return b; // Statement
}

// Pass by Address
void changeValueByAddress(Box *b) { // Function definition
  b->setLength(30); // Statement
}
```

- Passing by value makes a copy (may be slower for large objects).
- Passing by address (pointer) changes the original object and is more efficient for big objects.

3. Inline Function

Definition: An inline function is declared with inline keyword.

It is expanded in place at the call location to reduce the overhead of a function call.

Best for short functions.

Syntax:

Example: Calculating the cube of a number quickly.

```
inline int cube(int x) { // Function definition
  return x * x * x; // Statement
}
```

---

4. Static Variables

Definition: Static local variables retain their values between function calls. Static class members are shared by all objects of the class.

Syntax:

```
int num = 100; // Global // Initialization or assignment

class Demo { // Class definition starts
public: // Access specifier
  static int value; // Static member // Statement
};

int Demo::value = 50; // Initialization or assignment
```

---

5. Static Member Function

Definition: A static member function belongs to the class rather than any object. It can only access static data members.

Example: Displaying a shared count value for all objects.

```
class Sample { // Class definition starts
public: // Access specifier
   static int count; // Statement
   static void showCount() { // Function definition
      std::cout << count; // Statement
   }
};

int Sample::count = 10; // Initialization or assignment
```

---

6. Const Member Function

Definition: A const member function guarantees it will not modify any data members of the class.

Syntax:

```
class MyClass { // Class definition starts
public: // Access specifier
   void display() const { // Function definition
      std::cout << "This function will not modify the object."; //
Statement
   }
};
```

Example: Displaying details of an object without changing it.

7. Classes and Objects

Definition: A class is a blueprint for creating objects. Objects are instances that hold actual data.

Syntax:

```
class Car { // Class definition starts
public: // Access specifier
  void start() { std::cout << "Car started"; } // Statement
};

int main() { // Function definition
  Car c1; // object creation // Statement
  c1.start(); // Statement
}
```

Example: Example: Creating a Car object and calling its start function.

8. Access Specifiers

Definition: Control the accessibility of class members: public, private, and protected.

Syntax:

```
class Example { // Class definition starts
public: // Access specifier
  int a; // accessible anywhere // Statement
private: // Access specifier
  int b; // accessible only inside class // Statement
};
```

Example: Example: Variable 'a' is accessible in main, but 'b' is not.

9. Scope Resolution Operator (::)

Definition: The scope resolution operator is used to define member functions outside the class, access global variables, and access static class members.

Syntax:

```
int num = 10; // Global variable // Initialization or assignment

class Test { // Class definition starts
public: // Access specifier
  static int count; // Statement
  void showGlobal(); // Statement
};

int Test::count = 5; // static member initialization // Initialization
or assignment

void Test::showGlobal() { // Function definition
  int num = 20; // Local variable // Initialization or assignment
  std::cout << "Global num: " << ::num << std::endl; // Statement
}
```

Example: Example: Using :: to access a global variable when a local variable has the same name.

10. Namespaces

Definition: Namespaces prevent name conflicts by grouping identifiers under a name.

Syntax:

```
namespace MySpace { // Statement
  int value = 10; // Initialization or assignment
}

int main() { // Function definition
  std::cout << MySpace::value; // Statement
}
```

Example: Example: Accessing a variable from a user-defined namespace.