**Lab4: Object Arguments, Dynamic Memory, Static Members, and Friend Functions**

**Objective:**

* To understand how to pass and return objects in functions.
* To implement arrays of objects and pointers to objects.
* To allocate memory for objects dynamically using dynamic constructors.
* To use static data members and static member functions.
* To apply constant member functions and create constant objects.
* To demonstrate the use of friend functions and friend classes for accessing private data.

**Theory:**

**1. Object as Function Arguments**

* Objects can be passed to functions so the function can use or modify their data.
* Passing by value sends a copy of the object, so the original is not changed.
* Passing by reference sends the actual object, allowing the function to modify it.  
  **Syntax:** void functionName(ClassName obj); // Pass a copy of the object  
   void functionName(ClassName &obj); // Pass the object by reference  
   void functionName(const ClassName &obj); // Pass object by reference without allowing changes

**2. Returning Objects from Functions**

* Functions can return objects back to the caller.
* Returning by value returns a copy of the object.
* Returning by reference returns the original object (careful with this).  
  **Syntax:**  
   ClassName functionName(); // Function returns an object by value  
   ClassName& functionName(); // Function returns an object by reference  
   return object; // Returning an object from a function

**3. Array of Objects**

* You can create an array that holds many objects of the same class.
* Each object is accessed by its index number starting from 0.
* Useful for managing groups of similar data like many students or employees.  
  **Syntax:** ClassName arrayName[arraySize]; // Declare an array of objects  
   arrayName[index].memberFunction(); // Access a member function of the object at position index

**4. Pointer to Objects**

* A pointer can hold the address of an object.
* This allows indirect access and dynamic handling of objects.
* Use -> to access members of the object through the pointer.  
  **Syntax:**  
   ClassName \*ptr; // Declare a pointer to an object  
   ptr = &object; // Store address of object in pointer  
   ptr->memberVariable; // Access data member via pointer  
   ptr->memberFunction(); // Call function via pointer

**5. Dynamic Memory Allocation for Objects**

* Objects can be created while the program is running using new.
* This is useful when the number of objects needed is unknown before running the program.
* Dynamically created objects should be deleted using delete to free memory.  
  **Syntax:**  
   ClassName \*ptr = new ClassName(); // Create a single object dynamically  
   ClassName \*arr = new ClassName[size]; // Create an array of objects dynamically  
   delete ptr; // Delete a dynamically created single object  
   delete[] arr; // Delete a dynamically created array of objects

**6. Dynamic Constructors**

* Constructors can be used to allocate memory or initialize dynamic data when an object is created.
* Typically, new is used inside constructors to allocate memory for data members.
* A destructor should be written to free this memory when the object is destroyed.  
  **Syntax:** ClassName(parameters) {  
       pointerMember = new dataType[size]; // Allocate memory in constructor

}  
 ~ClassName() {  
     delete[] pointerMember; // Free memory in destructor

}

**7. Static Data Member and Static Member Function**

* Static data members are shared among all objects of the class (only one copy exists).
* Static member functions belong to the class itself, not any specific object.
* Static functions can only access static data members.
* Static functions can be called using the class name without creating an object.  
  **Syntax:** class ClassName {  
       static dataType staticVariable; // Declare static data member  
       static returnType staticFunction(); // Declare static member function  
   };  
   dataType ClassName::staticVariable = initialValue; // Define static variable outside class  
   returnType ClassName::staticFunction() {  
       // Function code  
   }

**8. Constant Member Function and Constant Objects**

* Constant member functions do not change any member variables of the object.
* Constant objects can only call constant member functions.
* This prevents accidental changes to objects that should remain unchanged.  
  **Syntax:** returnType functionName() const; // Constant member function declaration  
   const ClassName obj; // Constant object declaration  
   obj.functionName(); // Constant object can call only constant member functions

**9. Friend Function and Friend Class**

* Friend functions have permission to access private and protected members of a class.
* Friend classes can access all private and protected members of the class that declared them friends.
* It is granted explicitly and is not inherited.  
  **Syntax (Friend Function):**  
   class ClassName {  
       friend returnType functionName(ClassName &obj); // Declare friend function  
   };  
  **Syntax (Friend Class):** class ClassName {

    friend class FriendClassName; // Declare friend class  
 };