National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2023, Lab Manual - 07

Course Code: CL-1004	Course : Object Oriented Programming Lab
Instructor(s) :	Shazia Paras Shaikh

Contents:

- Working with static functions
- Constant
- Constant Functions
- Inline Functions
- Member initializer list
- Has - a relationship

Working with Static Functions:

A class's static member function is a function that has been designated as static and as a result a static member function is unrelated to any class object.

Frequently, shared data between all objects in a class is stored in static members.

For instance, you could use a static data member as a counter to keep track of the number of newly created objects of a particular class type. To track the total number of objects, this static data member can be increased each time an item is created.

The scope resolution operator also allows calling static member functions using the class name. A static member function can call static member functions inside or outside of the class, as well as static data members although the scope of static member functions is restricted to the class and they are unable to access the current object pointer.

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

class Student
{
   private:
   static int ID;
   static int SemesterNumber;
   static float CGPA ;

   public:

   static void DisplayDetails()
   {
      cout << "The ID of the student is: " << ID << endl;
     cout << "The current semester of the student is: " << SemesterNumber  <<
endl;
      cout << "The CGPA of the student is: " << CGPA << endl;
   }
};

// initialize the static data members

int Student :: ID = 101;
int Student :: SemesterNumber = 3;
float Student :: CGPA = 3.56;


int main()
{

   Student stu;
   b.print();
   Box::print();

cout << "\nStatic member function can be called by object or by
class reference: \n"  << endl;
   return 0;
}
```

const Keyword in C++

Constant is something that doesn't change. In C language and C++ we use the keyword const to make program elements constant. **const** keyword can be used in many contexts in a C++ program. It can be used with:

1. Variables
2. Function arguments and return types
3. Class Data members
4. Class Member functions
5. Objects

Constant Variables in C++

```
int main
{
   const int i = 10;
   const int j = i + 10;     // works fine
   i++;    // this leads to Compile time error
}
```

If you make any variable as constant, using const keyword, you cannot change its value. Also, the constant variables must be initialized while they are declared.

In the above code we have made i as constant, hence if we try to change its value, we will get compile time error. Though we can use it for substitution for other variables.

Constant Functions in C++

Functions declared constant are those that are prohibited from altering the data members of their class. The function prototype and the function definition header both receive the keyword "const" to designate a member function as a constant.

Class objects can also be defined as const, just like member functions and member function arguments. Because a const object cannot be changed, only const member functions may be called on it because they guarantee that the object won't be changed.

By adding the const keyword before the object definition, a const object can be made. A compile-time error occurs whenever a const object's data member is attempted to be changed.

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

class ConstTest
{
    double var;

      public:

    void set_var(int var)
    {
        this->var=var;
    }

     int get_var() const        //The function is now
constant
     {
         ++var;            // Error (cannot change the
member now)
         return var;
     }

};


main()
{
    ConstTest objC;
    objC.set_var(80);
    cout<<endl<<objC.get_var();

    return 0;
}
```

INLINE FUNCTIONS

A programming construct known as an "inline function" executes the function code directly within the code that uses it, as opposed to doing so via a separate function call. This indicates that rather than creating a distinct function object, the function code is copied and pasted into the calling code during programme compilation.

For small, frequently called functions with little overhead, like getters and setters or straightforward arithmetic tasks, inline functions are frequently used. Inline functions can improve efficiency by cutting down on the time required for function call setup and tear-down by getting rid of the function call overhead.

The keyword "inline" is used to signal that a function should be treated as inline. However, if compilers believe it would boost speed, they may also inline functions automatically.

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

inline int cube(int x) {
    return x * x * x;
}

int main() {
    int num = 5;
    cout << "The cube of " << num << " is " <<
square(num) << endl;
    return 0;
}
```

In this above example, the inline keyword is used to define the square() function. The compiler will substitute the function call with the real code of the square() function when the function is invoked in main(). By removing the overhead of a function call, this can improve speed.

It should be noted that the compiler may decide not to inline the function if it decides that doing so would not improve performance. The use of inline is merely a recommendation to the compiler.

MEMBER INITIALIZATION LIST

Initializer List is used in initializing the data members of a class. The list of members to be initialized is indicated with the constructor as a comma-separated list followed by a colon. Following is an example that uses the initializer list to initialize x and y of Point class.

Uses:

- For initialization of non-static const data members.
- For initialization of reference members.
- For initialization of member objects which do not have a default constructor.
- For initialization of base class members.
- When the constructor's parameter name is the same as the data member.
- For Performance reasons.

```
class Point {
private:
    int x;
    int y;
public:
    Point(int i = 0, int j = 0):x(i), y(j) {}
    int getX() const {return x;}
    int getY() const {return y;}
};

int main() {
  Point t1(10, 15);
  cout<<"x = "<<t1.getX()<<", ";
  cout<<"y = "<<t1.getY();
  return 0;
}
```

HAS-A RELATIONSHIP

In Object-Oriented Programming (OOP), a has-a relationship is a type of association between classes where one class has a member variable of another class type. This relationship is also known as composition or aggregation.

Association

A type of relationship between classes in C++ called association explains how objects of one class are connected to objects of another class. An instance of association is one in which one class makes use of or interacts in some manner with another class.

In C++, there are primarily two kinds of associations:

Aggregation (Independent)

Aggregation is a type of association where objects from different classes are referenced by one another but can still live separately.

In implementation below, the Book class has attributes for its title, author, and year published. The Library class has an array of Book pointers, and a method for adding Book objects to the library. When a Book is added to the library, the library prints a message indicating that the book has been added.

This implementation demonstrates an aggregation relationship between the Library and Book classes, since the Library "has-a" collection of Book objects, but the Book

```
#include <iostream>
#include <string>

using namespace std;

class Book {
private:
    string title;
    string author;
    int year_published;
public:
    Book(string t, string a, int y) {
        title = t;
        author = a;
        year_published = y;
    }
    string get_title() {
        return title;
    }
};

class Library {
private:
    Book* books[100]; // Maximum 100 books in library
    int num_books;
public:
    Library() {
        num_books = 0;
    }
    void add_book(Book* book) {
        if (num_books < 100) {
            books[num_books] = book;
            num_books++;
            cout << book->get_title() << " added to
library" << endl;
        } else {
            cout << "Library is full" << endl;
        }
    }
};

int main() {
    Book book1("The Catcher in the Rye", "J.D.
Salinger", 1951);
    Book book2("To Kill a Mockingbird", "Harper Lee",
1960);
    Book book3("1984", "George Orwell", 1949);

    Library library;
    library.add_book(&book1);
    library.add_book(&book2);
    library.add_book(&book3);

    return 0;
}

};
```

Composition (Dependent)

This form of association involves the composition of two classes, in which case the containing object determines the lifetime of the composed object. In other terms, the composed object is also destroyed when the containing object is.

```
class Engine {
  private:
    int horsepower;
  public:
    Engine(int hp) : horsepower(hp) {}
};

class Car {
  private:
    Engine engine;
  public:
    Car(int hp) : engine(hp) {}
};
```

In this example, we have two classes Engine and Car. The Engine class has a single member variable horsepower, which represents the horsepower of the engine. The Car class has a single member variable engine, which is an object of type Engine.

In the constructor of the Car class, we initialize the engine member variable using a member initialization list. We pass the hp argument to the constructor of the Engine class to create an Engine object with the specified horsepower, and then assign the resulting object to the engine member variable.

With this composition, a Car object can be created with a specific engine, and the engine object is owned by the car object. If the Car object is destroyed, the engine object is also destroyed.

Lab tasks:

1) Write a program that simulates a group of people using a class called Person that has a static member variable count, which holds the number of people in the group. The Person class also has static member functions to get the count and reset it to zero. The program creates three instances of the Person class with different names and increments the count each time a new Person object is created. The program then prints out the total number of people and each person's name using a non-static member function.

2) Make a program that consists of three classes: Person, Car, and Garage.

   Add the details as following:

   The Person class represents a person and has a single member variable name.
   The Car class represents a car and has two member variables: make and owner. make represents the make of the car and owner is a pointer to a Person object, which creates an aggregation relationship between the Car and Person classes.
   The Garage class represents a garage and has an array of Car pointers. This creates a composition relationship between the Garage and Car classes.
   In the main() function, the program creates some Person and Car objects and adds them to a Garage object. The program then prints out the details of the cars in the garage, including their make and owner.

3) Suppose you're creating a Rectangle class to represent rectangles in a 2D coordinate system. The Rectangle class should have two member variables: width and height. Write a program that defines the Rectangle class using member initialization list to initialize the member variables to default values of 0.

   In the main() function, create two Rectangle objects and prompt the user to enter the width and height of each rectangle. After the user enters the values, calculate and display the area of each rectangle.

4) Suppose you are working on a program to calculate the distance between two points in 2D space. You decide to use a class called Point to represent a point, with two private member variables x and y representing the coordinates of the point.

   Using inline functions, write a program that prompts the user to enter the coordinates of two points, creates Point objects for each point, calculates the distance between the two points using an inline function, and displays the result to the user.

Hint: To calculate the distance between two points (x1, y1) and (x2, y2), use the formula sqrt((x2 - x1)^2 + (y2 - y1)^2).

5) Create a program that models a university department using classes for Instructor, Course, and Department. The Department class should contain an array of Instructor objects and the instructor class should contain an array of Course objects. Each Course object should contain a pointer to the instructor who teaches it. Implement appropriate constructors, destructors, and member functions for each class. Use the main() function to create a Department object, add Instructor and Course objects to it, and display the department's instructors and their courses.