National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Lab Manual - 04

Course Code: CL-217	Course : Object Oriented Programming Lab
Instructor(s)	Aqsa Zahid

CONTENTS

1. CLASSES (REVISITED)

A **class** is a collection of a fixed number of components. The components are called **members** of the class.
The general syntax for the class is as follow:

```
class classIdentifier
{
    classMembersList
};
```

You can access class members using **class objects**.

2. CLASS MEMBERS

Class member list can be either a **variable** or a **function**.

- If a member of a class is a variable, you declare it just like any other variable. Also, in the definition of the class, you cannot initialize a variable when you declare it.
- If a member of a class is a function, you typically use the function prototype to declare that member.
- If a member of a class is a function, it can (directly) access any member of the class—member variables and member functions. That is, when you write the definition of a member function, you can directly access any member variable

of the class without passing it as a parameter. The only obvious condition is that you must declare an identifier before you can use it.

ACCESS MODIFIERS

The members of a class are classified in three categories: **private, public,** and **protected.**

Following are some facts about **public** and **private.**

- By default, all members of a class are **private**.
- If a member of a class is **private**, you cannot access it outside of the class
- A **public** member is accessible outside of the class.
- To make a member of a class **public**, you use the member access specifier public with a colon ': '.

The following statements define the **class** `clockType`.

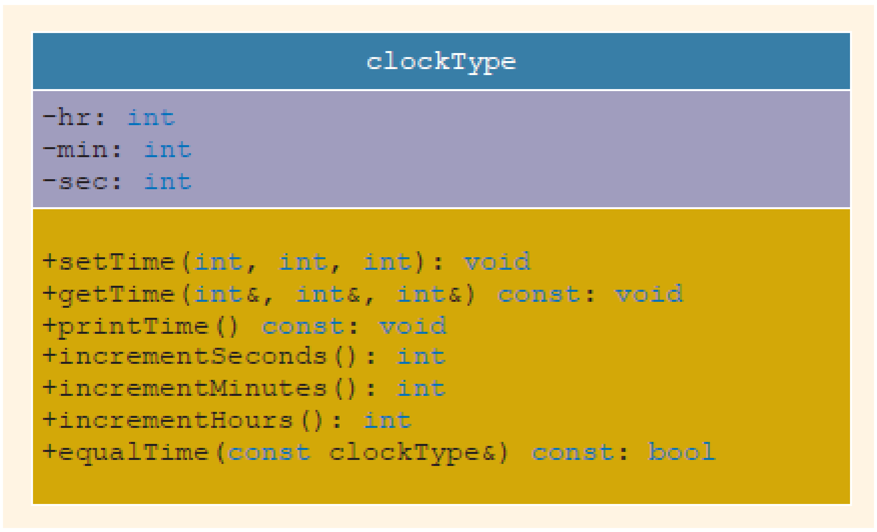
```
class clockType
{
      public:
            void setTime (int, int, int);
            void getTime (int&, int&, int&);
            void printTime ();
            void incrementSeconds ();
            void incrementMinutes ();
            void incrementHours ();
            bool equalTime (clockType&);

      private:
            int hr;
            int min;
            int sec;
};
```

In this definition:

- The class clockType has seven member functions: **setTime, getTime, printTime, incrementSeconds, incrementMinutes, incrementHours,** and **equalTime**. It has three member variables**: hr, min,** and **sec**.
- The three member variables — **hr, min,** and **sec** — are **private** to the class and cannot be accessed outside of the class.
- The seven member functions — **setTime, getTime, printTime, incrementSeconds, incrementMinutes, incrementHours,** and **equalTime** — can directly access the member variables (hr, min, and sec). In other words, when we write the definitions of these functions, we do not pass these member variables as parameters to the member functions.
- In the function equalTime, the formal parameter is a reference parameter. That is, in a call to the function **equalTime**, the formal parameter receives the address of the actual parameter. You could have declared the formal parameter as a value parameter, but that would require the formal parameter to copy the value of the actual parameter, which could result in poor performance.

3. UML FOR CLASS DIAGRAMS

A class and its members can be described graphically using a notation known as the Unified Modeling Language (UML) notation. For example, Figure 1 shows the UML class diagram of the class clockType.



4. ACCESSING CLASS MEMBERS:

Once the object of a class is declared, it can access the members of the class. The general syntax for an object to access a member of a class is:

```
classObjectName.memberName
```

The class members that a class object can access depend on where the object is declared.

- If the object is declared in the definition of a member function of the class, then the object can access both the public and private members.
- If the object is declared elsewhere (for example, in a user's program), then the object can access only the public members of the class.

Suppose we have the following declaration (say, in a user's program):

```
clockType myClock;
clockType yourClock;
```

Consider the following statements:

```
myClock.setTime(5, 2, 30);
myClock.printTime();
yourClock.setTime(x, y, z);    //assume x, y, and z are
                               //variables of type int
```

5. CONSTRUCTORS

To guarantee that the member variables of a class are initialized, you use **constructors.** There are two type of constructors: with parameters and without parameters. The constructor without the parameters is called **default constructor.**

Following are the properties of a constructor.

- The name of a constructor is the same as the name of the class.
- A constructor, even though it is a function, has no type. That is, it is neither a value-returning function nor a void function.
- A class can have more than one constructor. However, all constructors of a class have the same name.
- If a class has more than one constructor, the constructors must have different formal parameter lists. That is, either they have a different number of formal parameters or, if the number of formal parameters is the same, then the data type of the formal parameters, in the order you list, must differ in at least one position.
- Constructors execute automatically when a class object enters its scope. Because they have no types, they cannot be called like other functions.
- Which constructor executes depends on the types of values passed to the
- class object when the class object is declared.

```
class clockType
{
public:
    void setTime(int, int, int);
    void getTime(int&, int&, int&) const;
    void printTime() const;
    void incrementSeconds();
    void incrementMinutes();
    void incrementHours();
    bool equalTime(const clockType&) const;
    clockType(int, int, int);  //constructor with parameters
    clockType();  //default constructor

private:
    int hr;
    int min;
    int sec;
};
```

INVOKING DEFAULT CONSTRUCTOR

The syntax to invoke the default constructor is:

```
className classObjectName;
```

For example, the statement:

```
clockType yourClock;
```

declares **yourClock** to be an object of type **clockType.** In this case, the default constructor executes.

INVOKING CONSTRUCTOR WITH PARAMETERS

Suppose a class contains constructor with parameters. The syntax to invoke a constructor with a parameter is:

```
className classObjectName(argument1, argument2, ...);
```

in which the **argument1, argument2,** and so on are either a variable or an expression.

Note the following:

- The number of arguments and their type should match the formal parameters (in the order given) of one of the constructors.
- If the type of the arguments does not match the formal parameters of any constructor (in the order given), C++ uses type conversion and looks for the best match. For example, an integer value might be converted to a floating-point value with a zero decimal part. Any ambiguity will result in a compile-time error.

STATIC MEMBERS OF A CLASS

Similar to **static variables** a class can have **static members**. Let us note the following about the static members of a class.

- If a function of a class is static, in the class definition it is declared using the keyword static in its heading.
- If a member variable of a class is static, it is declared using the keyword static.
- A public static member, function, or variable of a class can be accessed using the class name and **the scope resolution operator ' :: '**.

LAB TASKS

TASK – 01:

Design and implement a class **dayType** that implements the day of the week in a program. The class **dayType** should store the day, such as **Sun for Sunday**. The program should be able to perform the following operations on an object of type **dayType**:

- Set the day.
- Print the day.
- Return the day.
- Return the next day.
- Return the previous day.
- Calculate and return the day by adding certain days to the current day. For example, if the current day is Monday and we add 4 days, the day to be returned is Friday. Similarly, if today is Tuesday and we add 13 days, the day to be returned is Monday.
- Add the appropriate constructors.

TASK – 02:

Redefine the class **personType** so that, in addition to what the existing class does, you can:

- Set the first name only.
- Set the last name only.
- Store and set the middle name.
- Check whether a given first name is the same as the first name of this person.
- Check whether a given last name is the same as the last name of this person. Write the definitions of the member functions to implement the operations for this class. Also, write a program to test various operations on this class.

TASK – 03:

a. Some of the characteristics of a book are the title, author(s), publisher, ISBN, price, and year of publication. Design a class **bookType** that defines the book as an ADT.

i. Each object of the class **bookType** can hold the following information about a book: title, up to four authors, publisher, ISBN, price, and number of copies in stock. To keep track of the number of authors, add another member variable.

ii. Include the member functions to perform the various operations on objects of type **bookType**. For example, the usual operations that can be performed on the title are to show the title, set the title, and check whether a title is the same as the actual title of the book. Similarly, the typical operations that can be performed on the number of copies in stock are to show the number of copies in stock, set the number of copies in stock, update the number of copies in stock, and return the number of copies in stock. Add similar operations for the publisher, ISBN, book price, and authors. Add the appropriate constructors and a destructor (if one is needed).

b. Write the definitions of the member functions of the class **bookType**.
c. Write a program that uses the class **bookType** and tests various operations on the objects of the class **bookType**. Declare an array of 100 components of type **bookType**. Some of the operations that you should perform are to search for a book by its title, search by ISBN, and update the number of copies of a book.

TASK – 04:

In this exercise, you will design a class **memberType**.
a. Each object of **memberType** can hold the name of a person, member ID, number of books bought, and amount spent.
b. Include the member functions to perform the various operations on the objects of **memberType** — for example, modify, set, and show a person's name. Similarly, update, modify, and show the number of books bought and the amount spent.
c. Add the appropriate constructors.
d. Write the definitions of the member functions of **memberType**.
e. Write a program to test various operations of your class **memberType**.

TASK – 05:

Using the classes designed in Lab Task 03 and 04, write a program to simulate a bookstore. The bookstore has two types of customers: those who are members of the bookstore and those who buy books from the bookstore only occasionally. Each member has to pay a $10 yearly membership fee and receives a 5% discount on each book purchased. For each member, the bookstore keeps track of the number of books purchased and the total amount spent. For every eleventh book that a member buys, the bookstore takes the average of the total amount of the last 10 books purchased, applies this amount as a discount, and then resets the total amount spent to 0. Write a program that can process up to 1000 book titles and 500 members. Your program should contain a menu that gives the user different choices to effectively run the program; in other words, your program should be user driven.