Time Class Case Study

In the preceding section, we introduced many basic terms and concepts of C++ object oriented programming. In this section, we take a deeper look at classes. In this Time class case study we will demonstrate several class construction features. We begin with a Time class that reviews several of the features presented in the preceding section.

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

class Time
{
	private :
		int hour;
		int minute;
		int second;
	public :
		//constructor with default value 0
		Time(int h = 0, int m  = 0, int s = 0);
		//setter function
		void setTime(int h, int m, int s);
		//print description of object in hh:mm:ss
		void print();
		//compare two time object
		bool equals(Time);
};

Time :: Time(int h, int m, int s)
{
	hour = h;
	minute = m;
	second = s;
}

void Time :: setTime(int h, int m, int s)
{
	hour = h;
	minute = m;
	second = s;
}
```

```
void Time :: print()
{
	cout << setw(2) << setfill('0') << hour << ":"
		<< setw(2) << setfill('0') << minute << ":"
		<< setw(2) << setfill('0') << second << "\n";
}

bool Time :: equals(Time otherTime)
{
	if(hour == otherTime.hour &&
		minute == otherTime.minute &&
		second == otherTime.second)
		return true;
	else
		return false;
}

int main()
{
	Time t1(10, 50, 59);
	t1.print();  // 10:50:59
	Time t2;  //object created with default value
	t2.print();  // 00:00:00
	t2.setTime(6, 39, 9); //set the new time in object
	t2.print();  // 06:39:09

	if(t1.equals(t2))
		cout << "Two objects are equals\n";
	else
		cout << "Two objects are not equals\n";

	return 0;
}
```

Output :

```
10:50:59
00:00:00
06:39:09
Two objects are not equals
```

Let's us discuss our Time class and add some new concepts of programming

Constructors with Default Arguments

In Circle class we have created explicit default constructor and parameterized constructor. Compiler overloads constructor based on match.

You can combine both statements in one as in Time class example

```
//constructor with default value
Time(int h = 0, int m = 0, int s = 0);
```

It works same way just matter of styling code.

Constant Function

In some cases, you will need that the member function should not change any private member variables of the calling object. You can do this by adding const to the end of the function declaration (prototype) and in the function definition.

Let's make member functions in our Time class const if appropriate:

```
class Time
{
        ...
        ...
        //print description of object in hh:mm:ss
        void print() const;
        ...
        ...

};

....
....
void Time :: print() const
{
        cout << setw(2) << setfill('0') << hour << ":"
                << setw(2) << setfill('0') << minute << ":"
                << setw(2) << setfill('0') << second << "\n";
}

        ....
```

Constant Parameters

You can make a parameter in a function as being a *const* parameter by preceding its type with *const*. This tells the compiler to disallow that parameter changing its value inside that function. This mechanism protects you from making inadvertent mistakes.

Let’s make parameter constant in our Time class, if appropriate:

```
class Time
{
        .....
        //constructor with default value 0
        Time(const int h = 0, const int m  = 0, const int s = 0);
        //setter function
        void setTime(const int h, const int m, const int s);
        .....
};

Time :: Time(const int h, const int m, const int s)
{
        hour = h;
        minute = m;
        second = s;
}

void Time :: setTime(const int h, const int m, const int s)
{
        hour = h;
        minute = m;
        second = s;
}

......
```

Passing Objects to Functions

Object can be passed by value, by reference or by pointer. In our Time class we passed object by value.

```
bool Time :: equals(Time otherTime)
{
        if(hour == otherTime.hour &&
                minute == otherTime.minute &&
                second == otherTime.second)
                return true;
        else
                return false;
}
```

This means that equals() receives a copy of object t2 with name otherTime.
If a function needs to store or change data in an object’s member variables, the object must be passed to it by reference.

Constant Reference Parameters

Passing by const reference is the preferred way to pass objects as an alternative to pass-by-value. When you pass by const reference, you take the argument in by reference, but cannot make any changes to the original object.

```
        //object passing by reference with const parameter
        bool equals(const Time&);
```