Lecture 12.1

Topics
1. `protected` Specification – Brief

---

1. `protected` Specification

Recall that there are three access specifications given inside a class definition as of,

`public`

`protected`

`private`

The **`public`** and **`private`** specifications have been introduced and used all of the classes presented in lectures so far.

And note that for **stand-alone classes** (that means classes to be used as they were defined without any kind of extension) just as the classes presented, the **`public`** and **`private`** specifications are sufficient.

2.1 Description

Let's consider the stand-alone classes first. In these classes, the `protected` specifier may be placed anywhere in class definition. Typically, it is placed right after the `private` members and before the `public` members.

The general form is as follows,

```
class ClassName {
public:
  // Public Members
protected:
  // Protected Members
private:
  //private Members
};
```

An example of a class with `protected` member is a good way to explain the idea and its use. Let's look at a simple example given next.

2.2 Example

```
//Program Name:  cis25L1211.cpp
//Discussion:    protected member
#include <iostream>
using namespace std;

class IntPoint {
public:
  IntPoint() {
    x = 1;
    y = 1;
  }
  IntPoint(int a, int b) {
    x = a;
    y = 1;
```

```
  }
  void setXY(int a, int b) {
    x = a;
    y = b;
    return;
  }
  void getXY(int& a, int& b) {
    a = x;
    b = y;
    return;
  }

protected:
  int x; // x-coordinate
  int y; // y-coordinate
};

int main() {
  IntPoint pObj(5, 10);

  int x, y;

  pObj.getXY(x, y);

  cout << "\nCoordinate of pObj,\n\tx : " << x
       << "\n\ty : " << y << endl;

  return 0;
}
```

OUTPUT

```
Coordinate of pObj1,
        x : 5
        y : 1
```

2.3 What Next?

When the ideas of code saving and reuse are brought into the design of classes, several things can be done such as

- Function (being called again and again at different stages of logical execution), or
- Making the objects as members of classes (e.g., a **`Point`** object has a member that is a **`Fraction`** object).

The next idea and concept are to extend the existing class(es) (if possible) to produce a new class – This is called inheritance.