IIT Bombay

Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Access Control and Introduction to Classes

Quick Recap of Relevant Topics

- Structures representing objects
  - Groups of related variables, arrays, other structures
  - Member functions as interfaces for interaction
  - Accessing members (data and functions) of structures

No restrictions on accessing members of a structure from anywhere in a program

Overview of This Lecture

IIT Bombay

- Access control of members in structures
    - private and public members
- Classes in C++ programs

Acknowledgment

IIT Bombay

- Much of this lecture is motivated by the treatment in

An Introduction to Programming Through C++

by Abhiram G. Ranade

McGraw Hill Education 2014

Recap: Object-Oriented Programming Overview

- Identify **entities** or **objects** involved in the working of the system
- Think of system functionality in terms of **operations on and interactions between objects**
  - Member functions are interfaces for these operations
- **Abstract** away (hide) details of object not necessary to be exposed
  - Data hiding or encapsulation
  - More generally controlling access to information/interface of objects

Focus of this lecture

Recap: Struct V3

```
struct V3 {
    double x, y, z;
    double length() { return sqrt(x*x + y*y + z*z); }
    V3 sum (V3 const &b) {
       V3 v;
       v.x = x + b.x; v.y = y + b.y; v.z = z = b.z;  return v;
    }
    V3 scale (double const factor) {
       V3 v;
       v.x = x*factor; v.y = y*factor;  v.z = z*factor;  return v;
    }
 };
```

Accessing Data Members of V3

```
 int main()
{  V3 vel, acc, pos; // initial velocity, acceleration, initial position
    … Some more declarations …
   cout << “Give x, y and z components of initial velocity: “ << endl;
   cin >> vel.x >> vel.y >> vel.z;
   cout << “Give x, y and z components of acceleration: “ << endl;
   cin >> acc.x >> acc.y >> acc.z;
   … Rest of code …
}
```

Accessing Member Functions of V3

```
 int main()
{  V3 vel, acc, pos; // initial velocity, acceleration, initial position
   V3 currDispl, currPos; // current displacement & position
   double t = 0.0, deltaT, totalT;  //  t: time elapsed so far
  … Reading in and validating values …
  while (t < totalT) {
      // Calculate current displacement using vel*t + (0.5)*acc*t^2
      currDispl = (vel.scale(t)).sum(acc.scale(0.5*t*t));
      currPos = currDispl.sum(pos);
      cout << “Time “ << t << “ “;  currPos.print();  t = t + deltaT;
  }
  return 0;
}
```

No restrictions on accessing members of a structure from anywhere in a program

Controlling Access to Members

IIT Bombay

- C++ allows three types of access control for every member (data or function)
  - **private**: Member can be accessed only from member functions of same structure
  - **public**: Member can be accessed from anywhere in program
  - **protected**: Outside scope of current discussion …
- Crucial for data hiding or encapsulation
- **private**, **public**, **protected**: C++ keywords

Controlling Access to Members

IIT Bombay

All members of a structure are public by default

```
struct V3 {
      double x, y, z;

      double length() { return sqrt(x*x + y*y + z*z); }
      V3 sum (V3 const &b) {
            V3 v;  v.x = x + b.x; v.y = y + b.y; v.z = z = b.z;  return v;
      }
     V3 scale (double const factor) {
            V3 v;  v.x = x*factor; v.y = y*factor;  v.z = z*factor;  return v;
     }
 };
```

Controlling Access to Members

```
struct V3 {
    private:
        double x, y, z;
    public:
        double length() { return sqrt(x*x + y*y + z*z); }
        V3 sum (V3 const &b) {
            V3 v;  v.x = x + b.x; v.y = y + b.y; v.z = z = b.z;  return v;
        }
        V3 scale (double const factor) {
            V3 v;  v.x = x*factor; v.y = y*factor;  v.z = z*factor;  return v;
        }
};
```

C++ allows access-control of groups of members

Controlling Access to Members

IIT Bombay

```
struct V3 {
    private:
        double x, y, z;
    public:
      V3 sum (V3 const &b) {
        V3 v;  v.x = x + b.x; v.y = y + b.y; v.z = z = b.z;  return v;
      }
      V3 scale (double const factor) {
        V3 v;  v.x = x*factor; v.y = y*factor;  v.z = z*factor;  return v;
      }
    private:
        double length() { return sqrt(x*x + y*y + z*z); }
 };
```

C++ allows access-control of groups of members

Classes in C++

- A **class** is like a structure, except that all members are private by default.
  - More commonly used than structures in C++ programs

```
class V3 {
    private:
        double x, y, z;
    public:
        double length() {  …. }
        V3 sum(V3 const &b) {  …. }
        V3 scale(double const factor) { …. }
};
```

C++ keyword

Effect of Access Control

int main()

{ V3 vel, acc, pos; // initial velocity, acceleration, initial position

… Some more declarations …

cout << “Give x, y and z components of initial velocity: “ << endl;

cin >> ~~vel.x >> vel.y >> vel.z;~~

cout << “Give x, y and z components of acceleration: “ << endl;

cin >> ~~acc.x >> acc.y >> acc.z;~~

… Rest of code …

}

Compiler Error!

So How Do We Read/Write Data Members of V3?

- Make all data members public
  - Not preferred, defeats purpose of data encapsulation
  - Breaks modularity of code by exposing internal details
    - E.g., we chose Cartesian coordinates to represent 3-D vectors
    - What if we later decide to use cylindrical coordinates ?

```
class V3 {
    public:
      //  double x, y, z;
           double rho, phi, z;
     … Member functions …
};
```

```
int main() {
  V3 vel, acc, pos;
  // cin >> vel.x >> vel.y >> vel.z;
   cin >> vel.rho >> vel.phi >> vel.z;
  … Rest of code …
}
```

Should All Data Members Always Be Private?

- Not necessarily
- Expose and allow access to only those members that other functions need access to
- Hide and prevent access to

  book-keeping data members, implementation-specific

  data members, internal state recording data members, …
- Choice of what should be private/public affects quality and modularity of code
  - Relevant for data members and member functions
  - Careful thought process essential – more of an art!

So How Do We Read/Write Data Members of V3?

Accessor functions

Member functions that return values of only those data members that other functions are allowed to read

```
class V3 {
  private: double x, y, z;
  public:
      double getX() {return x;}
      double getY() {return y;}
      double getZ() {return z;}
      … Other member functions …
 };
```

So How Do We Read/Write Data Members of V3?

Mutator functions

Member functions that update values of data members that other functions are allowed to update

```
class V3 {
  private: double x, y, z;
  public:
      void setXYZ(double vx, double vy, double vz)
        { x = vx; y = vy; z = vz; return;}
      … Other member functions …
 };
```

So How Do We Read/Write Data Members of V3?

- Changing internal representation of a class requires changing only accessor/mutator function definitions

```
class V3 {
  private: double rho, phi, z;
  public:
      double getX() {return (rho* cos(phi));}
      double getY() {return (rho* sin(phi));}
      double getZ() {return z;}
      void setXYZ(double vx, double vy, double vz)
      { rho = sqrt(vx*vx + vy*vy); phi = arctan(vy/vx); z = vz; return;}
      … Other member functions …
 };
```

Summary

IIT Bombay

- Controlling access to members in structures through “public” and “private”
- A brief introduction to C++ classes