Computer Programming

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

Session: An Example Program using Member Functions

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
  - Pointers to structures
  - Dynamic allocation and de-allocation of structures

Overview of This Lecture

- An interesting program using member functions
  - Two object-oriented implementations

Acknowledgment

- Some examples in this lecture are from

  An Introduction to Programming Through C++

  by Abhiram G. Ranade

  McGraw Hill Education 2014

- All such examples indicated in slides with the citation **AGRBook**

Recap: Vectors in 3 Dimensions [Ref. AGRBook]

- We want to reason about motion in 3-dimensional space
- Must deal with 3-dimensional vectors representing
  - Position
  - Velocity
  - Acceleration
- 3-dimensional vectors are basic entities (objects) in this program
  - Need to define a C++ structure to represent a vector
  - For simplicity, we will use Cartesian coordinates

Recap: Struct V3 with Member Functions

IIT Bombay

```
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;
    }
};
```

A Simple Motion Simulator [Ref AGRBook]

IIT Bombay

- Given
  - Initial position and velocity of a solid body as 3-dimensional vectors
  - Acceleration of the body as a 3-dimensional vector **(assume this doesn't change with time)**
  - Granularity of simulation time (Δ)
  - Total elapsed time (T)
- Find the positions (as 3-dimensional vectors) every Δ units of time for the entire elapsed time (T)

Augmenting V3

IIT Bombay

Let's add a member function to print out the x, y and z co-ordinates of a V3 object

```
struct V3 {
    double x, y, z;
    … Member functions "sum", "scale" and "length" …
    void print() {
      cout << "x: " << x << " y: " << y << " z: " << z << endl;
      return;
    }
};
```

A Motion Simulator Program

IIT Bombay

```
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
   cout << “Give x, y and z components of initial velocity: ”;
   cin >> vel.x >> vel.y >> vel.z;
   cout << “Give x, y and z components of acceleration: “;
   cin >> acc.x >> acc.y >> acc.z;
   cout << “Give x, y and z components of initial position: “;
   cin >> pos.x >> pos.y >> pos.z;
   cout << “Give total simulation time: “; cin >> totalT;
   cout << “Given simulation time granularity: “; cin >> deltaT;
   … Rest of code …
}
```

A Motion Simulator Program

IIT Bombay

```
 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 values …
   if ((totalT < 0)  || (deltaT <= 0)) {
     cout << “Invalid input!” << endl; return -1;
   }
   … Rest of code …

}
```

A Motion Simulator Program

IIT Bombay

```
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;
}
```

An Alternative Implementation

Define a MotionSimulator Object

```
struct MotionSimulator {
    V3 initPos, initVel, acc;
    V3 currPos, currVel;
    double deltaT;
    void initializeSimulator() {
            currPos = initPos; currVel = initVel; return;
    }
    … Rest of member functions …
};
```

An Alternative Implementation

IIT Bombay

Define a MotionSimulator Object

```
struct MotionSimulator {
  … Member declarations …
  void initializeSimulator() { … }
  void simulateAStep() {
    // Calculate updated position as currPos + currVel*deltaT
    // Calculate updated velocity as currVel + acc*deltaT
    currPos = currPos.sum(currVel.scale(deltaT));
    currVel = currVel.sum(acc.scale(deltaT));
    return;
  }
  … Rest of member functions …
};
```

An Alternative (Not Equivalent) Implementation

Define a MotionSimulator Object

```
struct MotionSimulator {
    … Member declarations …
    void initializeSimulator() { … }
    void simulateAStep() { … }
    void printPosition() {
        currPos.print();
        return;
    }
};
```

A Motion Simulator Program


```
 int main()
{  MotionSimulator mSim;
   double t = 0.0, deltaT, totalT;  //  t: time elapsed so far
   cout << “Give x, y and z components of initial velocity: “;
   cin >> (mSim.initVel).x  >> (mSim.initVel).y >>  (mSim.initVel).z;
   cout << “Give x, y and z components of acceleration: “;
   cin >> (mSim.acc).x >>  (mSim.acc).y >>  (mSim.acc).z;
   cout << “Give x, y and z components of initial position: “;
   cin >> (mSIm.initPos).x >> (mSim.initPos).y >> (mSim.initPos).z;
   cout << “Give total simulation time: “; cin >> totalT;
   cout << “Given simulation time granularity: “; cin >> deltaT;
   mSim.deltaT = deltaT;
   … Rest of code …
}
```

A Motion Simulator Program

IIT Bombay

```
int main()
{  MotionSimulator mSim;
   double t = 0.0, deltaT, totalT;  //  t: time elapsed so far
   … Reading in and validating values …
   mSim.initializeSimulator();
   while (t <= totalT) {
        mSim.simulateAStep();
        cout << “Time “ << t << “ “;  mSim.printPosition();
        t = t + deltaT;
   }
   return 0;
}
```

An Interesting Question

IIT Bombay

- Are the results computed by the two simulations the same?
- If not, why?
  - Try with various values of simulation time granularity
  - For what values of simulation time granularity, do the results computed by the two simulations agree (by and large)?

- Implementation choices are important when solving a problem

Summary

- A simple motion simulator program using object-oriented programming
- Use of member functions
- Glimpse of data hiding, and effect of simulation time granularity