

Computer Programming

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

Session: Object-oriented Programming using Member Functions



- Brief introduction to object-oriented programming
 - Program as a collection of interacting objects
- Structures representing objects
 - Groups of related variables, arrays, other structures
 - Accessing members of structures
 - Pointers to structures
 - Dynamic allocation and de-allocation of structures



- Member functions of structures
 - Interfaces of objects in object-oriented programming
- Accessing member functions





- 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**



- Identify entities (physical or conceptual) involved in the working of the system
 - Entities also called **objects**
- Think of system functionality in terms of operations on and interactions between objects
- Abstract away (hide) details not necessary for an operation
- Implement system modularly by focusing on entities, their interfaces and their interactions





- Contains information specific to the object
 - "Fixed" information usually doesn't change as objects interact
 - "State" information can change as objects interact
- Unambiguous, well-defined boundaries
 - Clear specification of what information is part of an object
- Ideally, every interaction between two objects should happen through well-defined interfaces

Focus of this lecture: interfaces of objects

Example: Vectors in 3 Dimensions [Ref. AGRBook]



- We want to write a program 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

The V3 Structure



What functions might we need to operate on objects of type V3?

- Adding two vectors
- Scaling a vector by a scalar constant
- Euclidean length of a vector ... and several more

Functions on V3 Objects



```
V3 sum (V3 const &a, V3 const &b) {
 V3 v;
 v.x = a.x + b.x;
 v.y = a.y + b.y;
 v.z = a.z + b.z;
 return v;
```

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

Note the manner in which parameters are passed.

Functions on V3 Objects



V3 scale (V3 const &a, double const factor) { V3 v;

```
v.x = a.x * factor;
```

```
v.y = a.y * factor;
```

```
v.z = a.z * factor;
```

return v;

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

Note the manner in which parameters are passed.

Functions on V3 Objects



```
double length (V3 const &a) {
```

double temp;

$$temp = a.x^*a.x + a.y^*a.y + a.z^*a.z;$$

return sqrt(temp);}

Assume "sqrt" function available from a library (e.g. cmath)

Note the manner in which parameters are passed.

- Let's take a closer look at the functions sum, scale and length
 - sum (**V3 const &a,** V3 const &b)
 - scale (V3 const &a, double const factor)
 - length (V3 const &a)

Each of these functions can be thought of as doing some computation on an object "a" of type "V3"

sum (V3 const &a, V3 const &b)
scale (V3 const &a, double const factor)
length (V3 const &a)

Why not associate these functions with the object "a" itself?

- When adding "b" to "a", call function "sum" associated with "a" and pass "b" as parameter
- When scaling "a" by "factor", call function "scale" associated with "a" and pass "factor" as parameter
- When finding the Euclidean length of "a", call function "length" associated with "a"

Helps define interfaces for interaction with the object "a"

• In C++, structures can have member functions

```
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;
```


Closer Look at a Member Function

Recall how we accessed member data values of structures

Recall how we accessed member data values of structures

V3 p, *ptrP; cin >> p.x; ptrP = &p; cout << ptrP->x;

Access using "->" operator

scale: Member function of receiver object

Parameter of member function

Accessing Member Functions of Structures

Member functions can be accessed in the same way

- Member functions as interfaces of structures
- Accessing member functions of structures