Computer Programming

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

Session: Inline Member Function and Template

Quick Recap of Relevant Topics

- Object-oriented programming with structures and classes
- Self-contained definitions of classes
  - All member functions defined within class definition

Overview of This Lecture

- Inlined and non-inlined member functions of classes
- Further use of scope resolution operator (::)
- Template classes and functions

Acknowledgment

- Much of this lecture is motivated by the treatment in

  An Introduction to Programming Through C++

  by Abhiram G. Ranade

  McGraw Hill Education 2014

- Examples taken from this book are indicated in slides by the citation **AGRBook**

Self-contained Class Definition

```
class V3 {
   private:
      double x, y, z;
      double length() const { return sqrt(x*x + y*y + z*z); }
  public:
      V3(double p = 0.0, double q = 0.0, double r = 0.0) {
          x = p; y = q; z = r; return;
     }
     ~V3() {return; }
     … Some more member functions (on next slide) …
};
```

Self-contained Class Definition

```
class V3 {
  private:   … Data and member functions (from previous slide) ...
  public:
     … Constructor and destructor (from previous slide) …
    V3 operator+ (V3 const &b) const {
         return V3(x + b.x, y + b.y, z + b.z);
    }
    V3 operator* (double const factor) const {
         return V3(x*factor, y*factor, z*factor);
    }
};
```

Can We Always Write Self-Contained Classes?

- **Inline member function**
  - Member function defined inside the class definition (all member functions we have seen so far)
  - Convenient if function definition contains a few lines
- With long and complicated member functions, defining member functions in class definition is cumbersome
- C++ allows member functions to be declared in a class definition, but defined outside the class definition
  - Very useful if different member functions are developed by different members of team

Member Functions Outside Class Definition

```
class V3 {
  private: double x, y, z;
           double length() const;
  public:
           V3(double p=0.0, double q=0.0, double r=0.0);
           V3 operator+(V3 const &b) const;
           V3 operator*(double factor) const;
          ~V3() {return;}
};
```

(Not inline) Member function declaration

Inline member function definition

Non-inline Member Function Definition

```
 V3::V3(double p, double q, double r) {
    x = p; y = q; z = r; return;
 }
V3  V3::operator+(V3 const &b) const {
   return V3(x+b.x, y+b.y, z+b.z);
 }
V3 V3::operator*(double factor) const {
  return (x*factor, y*factor, z*factor);
}
double V3::length() const {return sqrt(x*x + y*y + z*z);}
```

Note use of scope resolution operator ::

Motivating Template Class [Ref. AGRBook]

Class for implementing queue of integers (car identifiers)

```
  class IntQueue {
    private: int front, nWaiting;
                int elements[100];
    public:
                IntQueue() { front = 0; nWaiting = 0; return; }
               ~IntQueue() {return;}
                bool insert (int value);
                bool remove (int &value);
};
```

Motivating Template Class

```
 bool IntQueue::insert(int value) {
     if (nWaiting == 100) { cout << “Q Full!” << endl;  return false; }
     else {
      elements[(front + nWaiting)%100] = value; nWaiting++; return true;
     }
}
bool IntQueue::remove(int &value) {
    if (nWaiting == 0) { cout << “Q Empty!” << endl; return false; }
    else {
        value = elements[front]; front = (front + 1)%100;  nWaiting--; return true;
    }
}
```

Motivating Template Class: Queue of 3-D Vectors

```
class V3Queue {
    private: int front, nWaiting;
                V3 elements[100];
    public:
                V3Queue() { front = 0; nWaiting = 0; return; }
               ~V3Queue() {return;}
                bool insert (V3 value);
                bool remove (V3 &value);
};
```

Motivating Template Class: Queue of 3-D Vectors

```
 bool V3Queue::insert(V3 value) {
   if (nWaiting == 100) { cout << “Q Full!” << endl;  return false; }
   else {
    elements[(front + nWaiting)%100] = value; nWaiting++; return true;
   }
}
bool V3Queue::remove(V3 &value) {
  if (nWaiting == 0) { cout << “Q Empty!” << endl; return false; }
  else {
    value = elements[front]; front = (front + 1)%100;  nWaiting--; return true;
  }
}
```

Motivating Template Class

IIT Bombay

- Differences between IntQueue and V3Queue only with respect to data types of some members
- Wouldn’t it be nice to be able to define a template for a Queue class with a generic data type T?
- The template can then be instantiated by specifying T to give Queue classes for different data types.

C++ provides a mechanism to do this: Template Class

Template Classes

- Foundation of **generic programming**
  - Programming independent of specific types

- Template is a schema for creating several classes that differ only in data types of members and some parameters

  Abstract definition of a class (complete with data members and member functions) with generic data types of members

Template Class

```
template <class T> class Queue {
    private: int front, nWaiting;
             T elements[100];
    public:
             Queue() { front = 0; nWaiting = 0; return; }
             ~Queue() {return;}
              bool insert (T value);
              bool remove (T &value);
};
```

Template Member Function

```
template <class T> bool Queue<T> ::insert(T value) {
    if (nWaiting == 100) {
         cout << “Q Full!” << endl;  return false;
    }
    else {
         elements[(front + nWaiting)%100] = value;
         nWaiting++; return true;
    }
}
```

Template Member Function

```
template <class T> bool Queue<T>::remove(T &value) {
    if (nWaiting == 0) {
        cout << “Q Empty!” << endl;
        return false;
    }
    else {
        value = elements[front];
        front = (front + 1)%100;  nWaiting--;   return true;
    }
}
```

Instantiating Template Classes

```
Queue<int> myIntQueueObject;
Queue<V3> myV3QueueObject;
```

- Helps reduce repetition of code
- Facilitates generic programming, thinking at an abstract level
- Crucial in C++ Standard Library … to be studied next

Concluding Note About Template Class

Template class definition allows using more than one generic data type and other parameters using a comma-separated list

Class definition:

```
 template <class T, int QueueSize> class Queue {
    private: int front, nWaiting;
                  T elements[QueueSize];
    public:  …
};
```

Instantiation:

```
Queue<V3, 10> myV3QueueObject;
Queue<int, 100> myIntQueueObject;
```

Summary

IIT Bombay

- Inline and non-inline member functions of C++ classes
- Use of scope resolution operator in defining non-inline member functions
- Template classes and functions