Advanced Programming Concepts with C++ CSI2372 – Fall 2019

Jochen Lang &
Mohamed Taleb
EECS

This lecture

Java in C++

- **Basic Object-oriented C++**
  - Classes, Ch. 2.6 , (7.1)
    - Example: Point2D
  - Construction
  - Constructor types, Ch. 7.5
  - Destruction 7.1.5

Classes

- `class` defines a type `class` (no access modifiers)
- Classes in C++ implement the same concepts than classes in Java
- Access control in classes defaults to `private` but can be defined as `private`, `protected` and `public`
- Classes typically have attributes and methods
- Declaration and definition of a `class` is typically separate from the method definitions
- Definition of a class is typically done in a header file
- Definitions of class methods are typically done in a cpp source file

Class Example: Point2D

- **Define a class `Point2D`**

```
class Point2D {
    double d_x = 3.0;
    double d_y;
public:
    // Use compiler to generate a no args ctor
    Point2D() = default;
    // All args ctor
    Point2D( double _x, double _y );
};
```

Some Remarks

- Class `Point2D` is defined; methods (here: only two constructors) are declared; this snippet belongs in a header
- Methods (here: only a constructor) are defined in source file
- `d_x, d_y` default to private access
- Copy constructor, destructor and assignment operator are not explicitly defined – they are synthesized by the compiler!
- Class variables can be initialized in a constructor or in the member list (applies to any constructor)

Constructor

- **Same name than its class**
  - Public or protected method
  - Performs initialization of object attributes
  - No return type (not even void)
  - Executed every time when an instance of a class is created
  - A class can have several constructors with different argument lists
    - method or function overloading (**Overloading** means to define multiple methods with the same name but different signatures)
  - *In C++11 construction can be delegated (implementing a constructor in terms of another – like Java)*

Destructor

- **Same name than class but starts with ~**
  - Public method
  - No return value
  - No arguments
    - Only one destructor per class
  - Called whenever an object is destroyed
    - Auto variable gets out of scope (including function arguments at the end of a function)
    - Explicit call to delete for dynamically allocated objects
    - Program terminates
  - Destructor should free all resources associated with an object, e.g., dynamic memory, file descriptors etc.
  - **Example:**
    - ~Point2D(){ }
    - ~Employee(){ delete pEmp; }
- *More later*

Class Types

- **Use in the same way as fundamental types:**
  - Declare and create

```
class A;
```

```
A objA;
```

  - Initialize*)

```
A objA( myArg );
```

```
A objA = A( myArg );
```

  - Assign

```
A objB; objB = objA;
```

  - Destruct

```
{ A objA; }
```

  - Use as arguments (by value)

```
void myFunc( A objA );
```

  - Use as return type

```
A myFunc();
```

*) Objects of restricted type of classes can also be brace-initialized

Class Example: Calculation with 2D Vectors to Point Locations

- Add addition, subtraction and dot product to class Point2D

```
class Point2D {
    double d_x = 3.0;
    double d_y;
public:
    Point2D() = default;
    Point2D( double _x, double _y );
    Point2D add( Point2D _oPoint );
    Point2D subtract( Point2D _oPoint );
    double dot( Point2D _oPoint );
};
```

Definition of `Point2D` Methods

- Can access private attributes since it is the same class
- Define types and prototypes (including classes) in header files but methods in cpp files

```
Point2D::Point2D( double _x, double _y ) {
…
}
double Point2D::dot( Point2D _oPoint ) {
…
}
Point2D Point2D::add( Point2D _oPoint ) {
…
}
Point2D Point2D::subtract( Point2D _oPoint ) {
…
}
```

Definition of `Point2D` Methods

- **Can access private attributes since it is the same class**

```
// Use initializer list to initialize class variables
Point2D::Point2D( int _x, int _y ): d_x( _x), d_y( _y)
{}
double Point2D::dot( Point2D _oPoint ) {
    double res = _oPoint.d_x * d_x + _oPoint.d_y * d_y;
    return res;
}
Point2D Point2D::add( Point2D _oPoint ) {
    Point2D res;
    res.d_x = d_x + _oPoint.d_x;
    res.d_y = d_y + _oPoint.d_y;
    return res;
}
```

Let's test our methods: Objects of type `Point2D`

```
#include Point2d.h

int main( ) {
  Point2D pt1( 0.0, 1.0 ); // Define a Point2d
  Point2D pt2(pt1); // Make a copy
  Point2D pt3{1.0,0.0}; // Use brace initialization
  Point2 pt4 = pt3; // Call the copy constructor again
  Point2D pt5; // Default point
  pt1.dot( pt3 );
  Point2D res = pt1.add(pt3).subtract( pt4 );
  return 0;
}
```

File Structure

- **Three files**
  - 2 source files: `main.cpp` and `point2d.cpp`
  - 1 header file: `point2d.h`
  - 2 files to compile
    - `main.cpp` including `point2d.h`
    - `point2d.cpp` including `point2d.h`
    - Results are two object files `main.obj` and `point2d.obj`
  - 2 files to link into an executable
  - `main.obj` and `point2d.obj` linked to form executable `main.exe`
- **Note: All file extensions are system and compiler conventions and can be changed.**

Next Lecture

C-like C++

- **Data types and memory management**
  - Scope, modifiers, type conversions
  - Automatic type derivation and conversions
  - Pointers and arrays
  - Memory allocation: static, automatic and dynamic
  - Allocation and de-allocation, C++ vs. C
  - Pass by value, by reference, by pointer