C++

Static members

Introduction

- There are two types of static members:
  - Static data member
  - Static method member
- Static members are shared between all the instances of a class: they do not belong to the objects but to the class

Static methods

- Static methods are like global function
- Static methods can access only static data (static methods do not have a **this** pointer)
- Calling a static method is done by prefixing its name by the class name and "**::**"
  - For a class A and a method f(): A::f();
- Like any other methods, static methods can access private data of the class

Example of usage

```
class A {
private:
  int _a;
public:
 A() : _a(0) {}
 static void f() {
   std::cout << “static method f” << std::endl;
 }
};

int main() {
  A::f(); // will print “static method f”
}
```

Static data

- Static variables (static data members) are shared among the instances of the class
- Static data members are similar to global variables but they have a better protection (provided by the class)
- A global variable is accessed by prefixing its name by the name of the class and “::”
  - Example: for a class A and a static var _a, it can be accessed by A::_a;
- Static data members have the same protection mechanism as other data members. A private static data member can not be accessed from outside of the class
- Static data / method members are also called **class data / method**
- Static data members can not be initialized in the class definition. They must be defined outside of the class definition (for example in the class implementation (.cpp) file)

Example

```
// In file Cars.h
class Cars {
private:
  static int num_produced;

public:
  Cars() { num_produced++; }
  static int get_num_produced() {
     return num_produced;
  }
};
```

```
// In file Cars_main.cpp
#include “Cars.h”
int Cars::num_produced = 0;

int main() {
  cout << Cars::get_num_produced() << endl;
  Cars c1;
  cout << Cars::get_num_produced() << endl;
  Cars c2;
  cout << Cars::get_num_produced() << endl;
}
```

Example of usage

- Static methods can be used in the named constructor idiom
  - Ctors always have the same name of the class
  - Ctors differentiated by the parameter list
  - Sometimes we want to have different ctors but with the same number of parameters and same type

Named constructor idiom

We would like to have a class Point with two ctors:
- one to construct a point from Cartesian coord
- one to construct a point from Polar coord

```
class Point {
 public:
  Point (float x, float y); // Cartesian coord
  Point (float r, float t); // Polar coord
  // Error: ambiguous ctor
};

int main() {
 Point p(1.0, 3.0); // ambiguous: which ctor ?
}
```

Named constructor idiom

```
class Point {
public:
 // construct a point from Cart coord
 static Point createFromCart(float x, float y);
 // construct a point from Polar coord
 static Point createFromPolar (float r, float t);

private:
 Point(float x, float y); // Cart coord
 float x, y; // Cart coord
};

int main() {
  Point p1 = Point::createFromCart(1.0, 3.0);
}
```