Queens College, CUNY, Department of Computer Science

Object Oriented Programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

June 29, 2018

Static data & methods

- In this lecture we shall learn about the keyword `static`, as applied to a C++ class.
- We shall learn about **static data members** in a class.
- We shall also learn about **static methods** in a class.
- The keywords "`const`" and "`static`" can be applied togerher.
- A data member and/or method of a C++ class can be tagged both `static` and `const`.

1 Example: class `Day`

- Consider the following very simple class and C++ program.
- The class just contains an array of strings with the names of the days of the week.

```
#include <iostream>
#include <string>
using namespace std;

class Day {
public:
  Day() : numDays(7),
          day({"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"})
          {}

  int num() const { return numDays; }
  string getDayOfWeek(int n) const
  {
    if ((n < 0) || (n >= numDays)) { return string(""); }
    return day[n];
  }
private:
  int numDays;
  string day[7];
};

int main()
{
  Day dw;
  for (int n = 0; n < dw.num(); ++n) {
    cout << dw.getDayOfWeek(n) << endl;
  }
  return 0;
}
```

- The significant feature is that ***the data is the same for all the objects of the above class.***
- There is no real need to instantiate a different array for every instance of objects of the class.
- By the same logic, the value of `numDays` is the same constant number for all the instances of objects of the class.
- **We only need one array of strings, and one `int`.**
- **They are shared by all the instances of objects of the class.**
- **They are effectively "global variables" in the class.**

2 Static data members

- We introduce the concept of **static data members** in a C++ class.
- Static data members are members which **belong to the class** (not to individual objects).
- Static data members are therefore effectively "global variables" in the class.
- **Static data members are indicated by the tag `static`.**
- Let us make a class `SDay` ("static day") with all static data members.

```
class SDay {
public:
  SDay() {}

  // static data
  static int numDays;                    // static data
  static string day[7];                  // static data
};

int SDay::numDays = 7;
string SDay::day[] = {"Sunday", "Monday", "Tuesday", "Wednesday",
                      "Thursday", "Friday", "Saturday"};
```

- The data members are public so outside applications can access them directly.
- It is also possible for static data members to be private. We shall do so later.
- Note that the class constructor *does not initialize the values of the static data members.*
- This is because a constructor is invoked to *create an object*, and the static data members do not belong to an individual object.
- A static data member is initialized in a statement **outside the class declaration.**

```
int SDay::numDays = 7;          // static data non-inline initialization statement
string SDay::day[] = {\dots};   // static data non-inline initialization statement
```

- The syntax is similar to the non-inline statements we have seen for C++ classes.
  1. The initialization statement is placed in a cpp file (not a header file).
  2. The initialization statement must also be written in **only one cpp file of the project.**
  3. Otherwise the same initialization would be duplicated in multiple places in the project.
- **The compiler does not automatically generate an initialization statement for a static data member.**
- It is our responsibility to initialize the static data members.

3 Accessing static data members: example program

- Because static data members belong to the class as a whole, **we do not need an object to access them.**
- Here is a working C++ program to use the class `SDay`.

```
#include <iostream>
#include <string>
using namespace std;

class SDay {
  // etc
};

// non-inline initialization of static data members

int main()
{
  for (int n = 0; n < SDay::numDays; ++n) {    // static member "SDay::numDays"
     cout << SDay::day[n] << endl;             // static member "SDay::day[n]"
  }
  return 0;
}
```

- The program **does not instantiate an object of the class.**
- The prefix "`SDay::`" (i.e. name of the class) is used to access static data.
  1. "`SDay::numDays`" acceses the value of `numDays` without a class object.
  2. "`SDay::day[n]`" accesses the value of `day[n]` without a class object.

4 Private static data, public static methods

- Let us make the static data members private.
- To access them, we supply **public static methods.**
  1. Just as data members can be static, class methods can also be static.
  2. A static method is associated with the class, not with an individual object, hence it can be accessed without using an object.
- Obviously, static methods can also be private.
- Here is a class `PSDay` with private static data and public static methods.

```
class PSDay {
public:
  PSDay() {}

  static int num() { return numDays; }                  // public static method
  static string getDayOfWeek(int n)                     // public static method
  {
    if ((n < 0) || (n >= numDays)) { return string(""); }
    return day[n];
  }

private:
  // static data
  static int numDays;
  static string day[7];
};

int PSDay::numDays = 7;
string PSDay::day[] = {"Sunday", "Monday", "Tuesday", "Wednesday",
                       "Thursday", "Friday", "Saturday"};
```

- The class looks almost the same as `Day` but the data members are now static.
- The public methods `num` and `getDayOfWeek` are now static.
  1. Note that **static methods cannot be tagged `const`.**
  2. If a class method is tagged as `const`, it is a promise that the function will not change the values of the data members of the object.
  3. However, a static method is not associated with an object.
- Although there is no example in the above class, it should be clear that **a static method cannot access non-static data members.**
- This is because non-static data members belong to an individual object, hence a static method would not know how to manipulate a non-static data member.

4.1 Example program for private static data and public static methods

- Here is a working C++ program to use the class `PSDay`.
- It looks almost the same as the program in Sec. 1 (which used the class `Day`), but the program below invokes static methods.
  1. The static method `num` is accessed via `PSDay::num`.
  2. The static method `getDayOfWeek` is accessed via `PSDay::getDayOfWeek`.
  3. There is no object of the class `PSDay`.

```
#include <iostream>
#include <string>
using namespace std;

class PSDay {
  // etc
};

// non-inline initialization of static data members

int main()
{
  for (int n = 0; n < PSDay::num(); ++n) {          // public static method
    cout << PSDay::getDayOfWeek(n) << endl;         // public static method
  }
  return 0;
}
```

5 const static data

- It was stated above that a static method cannot be tagged `const`.
- However, **a static data member can be tagged `const`.**
- It is obvious that `numDays` and the array `day` are constants.
- They can both be tagged as `const` static data members.
- The syntax to use `const` is obvious.
- Here is a class `CPSDay` with `const` static data.
- The main program looks the same as in Sec. 4.1.

```
class CPSDay {
public:
  CPSDay() {}

  static int num() { return numDays; }
  static string getDayOfWeek(int n)
  {
    if ((n < 0) || (n >= numDays)) { return string(""); }
    return day[n];
  }

private:
  static const int numDays;                          // const static data
  static const string day[7];                        // const static data
};

const int CPSDay::numDays = 7;
const string CPSDay::day[] = {"Sunday", "Monday", "Tuesday", "Wednesday",
                              "Thursday", "Friday", "Saturday"};

int main()
{
  for (int n = 0; n < CPSDay::num(); ++n) {
    cout << CPSDay::getDayOfWeek(n) << endl;
  }
  return 0;
}
```

6 Copy, assign, `this` pointer

- The copy constructor and assignment operator **do not copy the values of static data.**
- The purpose of both the copy constructor and assignment operator is to *make a copy of an object*, and static data is not associated with an object.
- **The "`this`" pointer does not exist in static methods.**
- The "`this`" pointer *points to the current object*, and a static method has no associated object.