C++ Features

Overview

- Static Members
- Default Values

2

Static Member Data

- All objects share the same data
- Global default data for a class
- Examples of static Data
  - Counting the number of objects created
  - Direction a flotilla of ships has

3

STATIC MEMBER DATA

The data members we declared in the classes where just a description for the data members later to be created. Only if an object is created these data members exist in memory.

The static data members can be seen as data members of the class instead of the object. The data members will have corresponding memory addresses. They exist beyond the lifetime of the objects of the class. All the objects share the same static data members.

These data members are often used in classes as default data for data members of the objects. Take for example the Date class and suppose the default Date constructor initialises the day, month and year to 01-01-1990. This is likely to be hard coded. Whenever you want to change these values you have to recompile the code of the class again to get different default values. During run-time you cannot change these default settings again.
One way to solve this problem is by creating three static data members for the default dates. Whenever the default values have to change you only have to change these static data members and all the default values from that point on are changed, because all the objects share the same static data members.

Defining Static Member Data

- Solves namespace pollution
- Use of 'static' specifier

```
class Point
{
public:
  static int s_counter;
};
```

- Must be initialised outside the class declaration (outside header file, in a source file)

```
int Point::s_counter = 0;
```

4

Defining Static Member Data

As mentioned earlier these static data members exist without having to create an object. Static data members live from program start. For variables to exist during the lifetime of a program they have to be defined globally. The global definition of the static data members is placed in a source file. It is custom to define static members in the code file belonging to the class.

The static members first need to be declared in the header file of the class using the 'static' keyword. This keyword is only used in the header file. The normal accessing rights of the class are used for these members.

Declaring Static Member Data

```
class Date
{
public:
       static int mdef_day;
       static int mdef_month;
       static int mdef_year;
};

class Line
{
public:
       static int ocounter;
       static Point origin;
};
```

The definition of these data members must be done in some source file we choose for the source file of the class. When defining these data members you have to use the class name and scope resolution operator.

Defining Static Member Data

```
int Date::mdef_day = 1;
int Date::mdef_month = 1;
int Date::mdef_year = 1990;

int Line::ocounter = 0;
Point Line::origin = Point();
```

These data members can be used with or without an object using the object or the scope resolution operator.

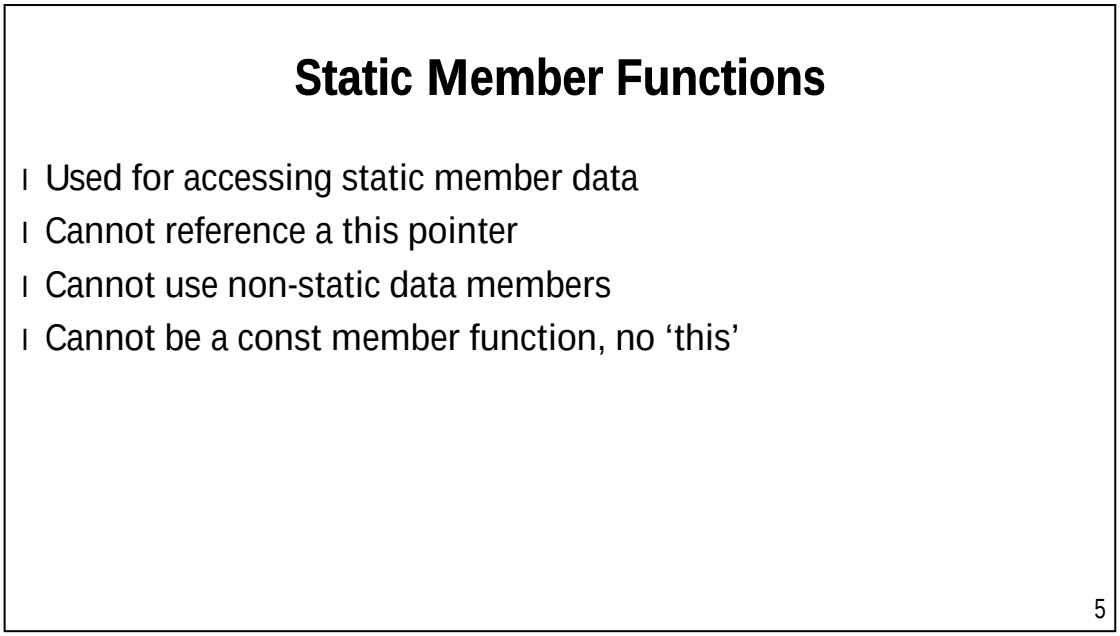
Using Static Member Data

```
void main
{
	// With an object
	Date d1;

	d1.mdef_day = 24;

	// Without an object
	Date::mdef_day = 31;
}
```

A typical example of a static member is the object counter. When an object is created the counter is incremented (in the constructor) and when it is destroyed the counter decrements (in the destructor).



STATIC MEMBER FUNCTIONS

Besides static member data it is also possible to create functions belonging to a class that can be used without creating an object.

These member functions are used for accessing the static members of a class. They can only be used for static members because they cannot access the non static members. A static member function can be used without specifying an object. This is why a static member function cannot access normal data members which only exist in an object.

The static member functions are member functions. However these functions have no concept of a 'this' pointer. These functions cannot access the non-static data members of the class.

The static member functions cannot be const member functions because they have no ‘this’ pointer because there is no concept of a current object.

Example Static Member Function

```
class Point
{
public:
  static int GetObjectCount();
};
```

- Usage:

```
void main()
{
  cout << Point::GetObjectCount();
}
```

6

EXAMPLE STATIC MEMBER FUNCTION

The static member functions are a lot like static member data and they can be used without having to create an object. The static member function is declared like a normal member function with the difference that the 'static' keyword is used only in the header file of the class.

Defining Static Member Function

```
class Date
{public:
       static void def_day(int newval);
};

void Date::def_day(int newval)
{
        mdef_day= newval;
}
```

Using Static Member Function

```
void main()
{
       // With an object
       Date d1;
       d1.def_day(10);

       // Without an object
       Date::def_day(10);
}
```

Default Values

- Assign default values to arguments
- Values only in header file
- Only last arguments can be default
- Upon calling, arguments should be filled left to right

```
class Point
{
public:
       Point(int x = 11, int y = 12);
};
```

7

DEFAULT VALUES

Member functions can be declared in such a way that certain default values can be assigned to certain arguments in the argument list. The impact for clients of the function is that not all parameters need to be given when calling the function; the 'missing' arguments will get the default values.

Default Values in Constructor

```
class Point
{
public:
       // Point(); Not possible any more
       Point(int xval = 11.0, int yval = 12.0);
};
```

The constructor of the Point class can be invoked with zero, one or two arguments of type double.

Default Values in Constructor

```
Point pt;                  // x=11, y=12
Point pt2(10);             // x=10, y=12
Point pt3(10, 20);         // x=10, y=20
```

Using default arguments is similar to using a number of overloaded functions. For example the function

Default Values in Functions

```
void foo(int a = 1, int b = 2, int c = 0)
```

is equivalent to four function declarations

```
void foo(int, int, int);
void foo(int, int);
void foo(int);
void foo();
```

The order of which function parameters have default values is important. Default values can only be given to function parameters from right to left. So giving the first parameter a default value but not the second is not possible. But it is possible to give only the last function parameter a default value but not the previous ones.

```
void foo2(int a = 1, int b = 2, int c);
```

Error with Default Values

will produce a compile error.

Similar, the order in you leave out values when calling a function with “optional parameters” is also important. You can only leave out arguments from right to left. Thus in case of the Point constructor defined above, you can omit passing “yval” while providing “xval” or you can omit both “xval” and “yval”. But it is not possible to omit only “xval”.