BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Programming and Computer Applications

Classes
A Deeper Look

Instructor : PhD, Associate Professor Leyla Muradkhanli

Using the `this` Pointer

- How do member functions know *which* object's data members to manipulate? Every object has access to its own address through a pointer called **this** (a C++ keyword).
- The `this` pointer is not part of the object itself.
  - The `this` pointer is passed (by the compiler) as an implicit argument to each of the object's non-`static` member functions.
- Objects use the `this` pointer implicitly or explicitly to reference their data members and member functions.
- The type of the `this` pointer depends on the type of the object and whether the member function in which `this` is used is declared `const`.

```cpp
// Using the this pointer to refer to object members.
#include <iostream>
using namespace std;
class Test
{
public:
   Test( int = 0 );
   void print() const;
private:
   int x;
};
Test::Test( int value )
   : x( value ) { }
void Test::print() const
{
   cout << "        x = " << x;
   cout << "\n  this->x = " << this->x;
   cout << "\n(*this).x = " << ( *this ).x << endl;
}
```

```
int main()
{
    Test obj( 12 );
    obj.print();
}
```

Output

x = 12

this->x = 12

(*this).x = 12

Using the `this` Pointer (cont.)

- Another use of the `this` pointer is to enable cascaded member-function calls
  - invoking multiple functions in the same statement
- The program modifies class `Time`'s *set functions* ***`setTime`***, ***`setHour`***, ***`set-Minute`*** *and* ***`setSecond`*** *such that each returns a reference to a* ***`Time`*** *object to enable cascaded member-function calls.*

```
// Time.h
// Cascading member function calls.
// Time class definition.
#ifndef TIME_H
#define TIME_H

class Time
{
public:
   Time( int = 0, int = 0, int = 0 ); // default constructor

   Time &setTime( int, int, int );
   Time &setHour( int ); // set hour
   Time &setMinute( int ); // set minute
   Time &setSecond( int ); // set second
```

```
// get functions (normally declared const)
   int getHour() const; // return hour
   int getMinute() const; // return minute
   int getSecond() const; // return second

   // print function
void print() const; // print time
private:
   int hour; // 0 - 23 (24-hour clock format)
   int minute; // 0 - 59
   int second; // 0 - 59
}; // end class Time

#endif
```

```cpp
// Time.cpp
// Time class member-function definitions.
#include <iostream>
#include <iomanip>
#include "Time.h"
using namespace std;
// constructor function to initialize private data;
// calls member function setTime to set variables;
Time::Time( int hr, int min, int sec )
{
   setTime( hr, min, sec );
}
// set values of hour, minute, and second
Time &Time::setTime( int h, int m, int s )
{
   setHour( h );
   setMinute( m );
   setSecond( s );
   return *this; // enables cascading
}
```

```
// set hour value
Time &Time::setHour( int h )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
   return *this; // enables cascading
}

// set minute value
Time &Time::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
   return *this; // enables cascading
}

// set second value
Time &Time::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
   return *this; // enables cascading
}
```

```
// get hour value
int Time::getHour() const
{
   return hour;
}

// get minute value
int Time::getMinute() const
{
   return minute;
}

// get second value
int Time::getSecond() const
{
   return second;
}
```

```
// print Time in universal-time format (HH:MM:SS)
void Time::print() const
{
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
      << setw( 2 ) << minute << ":" << setw( 2 ) << second;
}
```

```cpp
// Source.cpp
// Cascading member-function calls with the this pointer.
#include <iostream>
#include "Time.cpp"
using namespace std;
int main()
{
   Time t; // create Time object
   // cascaded function calls
   t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );
   cout << “Time: ";
   t.print();
   cout<<"\nNew time : ";
   // cascaded function calls
   t.setTime(23,15,20).print();
   cout<<endl;
}
```

Using the `this` Pointer (cont.)

- Why does the technique of returning `*this` as a reference work? The dot operator (`.`) associates from left to right, so line 10 first evaluates `t.setHour(18)`, then returns a reference to object `t` as the value of this function call.
- The remaining expression is then interpreted as
  - `t.setMinute( 30 ).setSecond( 22 );`
- The `t.setMinute( 30 )` call executes and returns a reference to the object `t`.
- The remaining expression is interpreted as
  - `t.setSecond( 22 );`

`static` Class Members

- In certain cases, only one copy of a variable should be shared by all objects of a class.
- A **static** data member is used for these and other reasons.

`static` data member in C++

- When we declare a normal variable (data member) in a class, different copies of those data members create with the associated objects.
- In some cases when we need a common data member that should be same for all objects, we cannot do this using normal data members. To fulfill such cases, we need static data members.

`static` data member in C++

- It is a variable which is declared with the static keyword, it is also known as class member, thus only single copy of the variable creates for all objects.
- Any changes in the static data member through one member function will reflect in all other object’s member functions.

`static` data member in C++

Declaration

static data_type member_name;

Defining the static data member

It should be defined outside of the class following this syntax:

data_type class_name :: member_name =value;

If you are calling a static data member within a member function, member function should be declared as static (i.e. a static member function can access the static data members)

static data member in C++

Consider the example, here static data member is accessing through the static member function:

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

class Demo
{
private:
static int X;

public:
static void fun()
{
cout <<"Value of X: " << X << endl;
}
};
```

static data member in C++

```
//defining
int Demo :: X =10;

int main()
{
Demo X;

X.fun();

return 0;
}
```

Output

Value of X: 10

Accessing static data member without static member function

A static data member can also be accessed through the class name without using the static member function (as it is a class member), here we need a scope resolution operator :: to access the static data member without static member function.

Syntax:

class_name :: static_data_member;

Accessing static data member without static member function

```
#include <iostream>
using namespace std;
class Demo
{
public:
static int A;
};
//defining
int Demo :: A=15;

int main()
{
cout<<"\nValue of A: "<<Demo::A;
return 0;
}
```

Output

Value of A: 15

Accessing static data member without static member function

In this program A is a class member (static data member), it can directly access with help on scope resolution operator.

Note: The const data member of class is static by default.

Static member function

A static member function is a special member function, which is used to access only static data members, any other normal data member cannot be accessed through static member function. Just like static data member, static member function is also a class function; it is not associated with any class object.

We can access a static member function with class name, by using following syntax:

class_name:: function_name(parameter);

Static member function

```
#include <iostream>
using namespace std;
class Demo
{
private:
//static data members
static int X;
static int Y;
public:
//static member function
static void  Print()
{
cout <<"Value of X: " << X << endl;
cout <<"Value of Y: " << Y << endl;
}
};
```

Static member function

```
//static data members initializations
int Demo :: X =10;
int Demo :: Y =20;
int main()
{
Demo OB;
cout<<"Printing through object name:"<<endl;
OB.Print();

cout<<"Printing through class name:"<<endl;
Demo::Print();
return 0;
}
```

Static member function

Output

Printing through object name:
Value of X: 10
Value of Y: 20
Printing through class name:
Value of X: 10
Value of Y: 20