CSCI 390 – Special Topics in C++

Lecture 12 (9/27/18)
and
Lecture 13 (10/2/18)

Time To Turn Off Cell Phones

Some More Of **`this`**

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"

struct sPoint
{
  sPoint(void) : x(0), y(0)
  {cout << DUMPTHIS(this) << endl;
   cout << DUMPTYPE(*this) << endl;
   cout << DUMPVAR(x) << endl;
   return;}
  double x;
  double y;
};

ostream &operator<<(ostream &s, const sPoint &p)
{s << "(" << p.x << ", " << p.y << ")"; return s;}

int main()
{
  sPoint Point;
  sPoint *pPoint{&Point};
  cout << DUMPOBJ(Point) << endl;
  cout << DUMPVAR(pPoint) << endl;
  cout << DUMPVAR(Point.x) << endl;
  cout << DUMPVAR(pPoint->x) << endl;

  return 0;
}
```

```
Variable: this, Type: sPoint*, Length: 8, Value: 0x7fffbb196270

Type: sPoint, Length: 16

Variable: x, Type: double, Length: 8, Address: 0x7fffbb196270,
Value: 0

Object: Point, Type: sPoint, Length: 16, Address: 0x7fffbb196270,
Value: (0, 0)

Variable: pPoint, Type: sPoint*, Length: 8, Address:
0x7fffbb196268, Value: 0x7fffbb196270

Variable: Point.x, Type: double, Length: 8, Address:
0x7fffbb196270, Value: 0

Variable: pPoint->x, Type: double, Length: 8, Address:
0x7fffbb196270, Value: 0
```

class vs struct

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"

class sPoint
{
  sPoint(void) : x(0), y(0)
  {cout << DUMPTHIS(this) << endl;
   cout << DUMPVAR(x) << endl;
   return;}
  double x;
  double y;
};

ostream &operator<<(ostream &s, const sPoint &p)
{s << "(" << p.x << ", " << p.y << ")"; return s;}

int main()
{
  sPoint Point;
  sPoint *pPoint{&Point};
  cout << DUMPOBJ(Point) << endl;
  cout << DUMPVAR(pPoint) << endl;
  cout << DUMPVAR(Point.x) << endl;
  cout << DUMPVAR(pPoint->x) << endl;

  return 0;
}
```

```
main.cpp: In function ‘std::ostream& operator<<(std::ostream&,
const sPoint&)’:
main.cpp:13:10: error: ‘double sPoint::x’ is private
   double x;
          ^
main.cpp:18:16: error: within this context
 {s << "(" << p.x << ", " << p.y << ")"; return s;}
                ^
main.cpp:14:10: error: ‘double sPoint::y’ is private
   double y;
          ^
main.cpp:18:31: error: within this context
 {s << "(" << p.x << ", " << p.y << ")"; return s;}
                               ^
main.cpp: In function ‘int main()’:
main.cpp:9:3: error: ‘sPoint::sPoint()’ is private
   sPoint(void) : x(0), y(0)
   ^
main.cpp:22:10: error: within this context
   sPoint Point;
          ^
main.cpp:13:10: error: ‘double sPoint::x’ is private
   double x;
          ^
In file included from main.cpp:5:0:
main.cpp:26:25: error: within this context
   cout << DUMPVAR(Point.x) << endl;
                         ^
Etc.
```

`private`, `public`, `protected`

- Object constructors, destructors, member functions and member variables can be either: **`private`**, **`public`**, or **`protected`**:
  - **`private`**: The members declared after the specifier are only visible inside object. This is the default for **`class`** objects.
  - **`public`**: The members declared after the specifier are visible inside and outside the object. This is the default for **`struct`** objects.
  - **`protected`**: Covered with inheritance discussion.

The Fix

```cpp
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"

class sPoint
{
public:
  sPoint(void) : x(0), y(0)
  {cout << DUMPTHIS(this) << endl;
   cout << DUMPVAR(x) << endl;
   return;}
  double x;
  double y;
};

ostream &operator<<(ostream &s, const sPoint &p)
{s << "(" << p.x << ", " << p.y << ")"; return s;}

int main()
{
  sPoint Point;
  sPoint *pPoint{&Point};
  cout << DUMPOBJ(Point) << endl;
  cout << DUMPVAR(pPoint) << endl;
  cout << DUMPVAR(Point.x) << endl;
  cout << DUMPVAR(pPoint->x) << endl;

  return 0;
}
```

```
Variable: this, Type: sPoint*, Length: 8, Value: 0x7ffd630ea1f0

Variable: x, Type: double, Length: 8, Address: 0x7ffd630ea1f0,
Value: 0

Object: Point, Type: sPoint, Length: 16, Address: 0x7ffd630ea1f0,
Value: (0, 0)

Variable: pPoint, Type: sPoint*, Length: 8, Address:
0x7ffd630ea1e8, Value: 0x7ffd630ea1f0

Variable: Point.x, Type: double, Length: 8, Address:
0x7ffd630ea1f0, Value: 0

Variable: pPoint->x, Type: double, Length: 8, Address:
0x7ffd630ea1f0, Value: 0
```

Member Functions

- Member function work very much like regular functions, except **`this`** is silently added as the first parameter.
- static member functions do not add **`this`** as the first parameter. They cannot access member variables.
  - Static functions are often used to avoid naming conflicts by placing the function inside the scope of the object.

Example

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include <cmath>
#include "Helper.h"
class sPoint
{
public:
  sPoint(void) : x(0), y(0) { return; }
  sPoint(double _x, double _y) : x(_x), y(_y)
    { return; }
  static double Distance(const sPoint &p)
  { cout << this << endl;
    return 0.0; }

  double x;
  double y;
};

ostream &operator<<(ostream &s, const sPoint &p)
{s << "(" << p.x << ", " << p.y << ")"; return s;}

int main()
{
    return 0;
}
```

```
main.cpp: In static member function ‘static double
sPoint::Distance(const sPoint&)’:
main.cpp:14:13: error: ‘this’ is unavailable for static member
functions
  { cout << this << endl;
            ^
```

The Fix

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include <cmath>
#include "Helper.h"
class sPoint
{
public:
  sPoint(void) : x(0), y(0) { return; }
  sPoint(double _x, double _y) : x(_x), y(_y)
    { return; }
  static double Distance(const sPoint &p1,
    const sPoint &p2)
  { double dx = p1.x – p2.x;
    double dy = p1.y - p2.y;
    return std::hypot(dx, dy); }  double x;
  double y;
};

ostream &operator<<(ostream &s, const sPoint &p)
{s << "(" << p.x << ", " << p.y << ")"; return s;}

int main()
{
  sPoint Origin;
  sPoint Point{1.0, 1.0};
  cout << DUMPVAL(sPoint::Distance(Origin, Point))
<< endl;
  return 0;
}
```

```
Expression: sPoint::Distance(Origin, Point), Type: double, Length:
8, Value: 1.41421
```

Example

```cpp
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include <cmath>
#include "Helper.h"
class sPoint
{
public:
  sPoint(void) : x(0), y(0) { return; }
  sPoint(double _x, double _y) : x(_x), y(_y)
    { return; }
  static double Distance(const sPoint &p) const
  { double dx = x - p.x; double dy = y - p.y;
    return std::hypot(dx, dy); }

  double x;
  double y;
};

ostream &operator<<(ostream &s, const sPoint &p)
{s << "(" << p.x << ", " << p.y << ")"; return s;}

int main()
{
  sPoint Origin;
  sPoint Point{1.0, 1.0};
  cout << DUMPVAL(Origin.Distance(Point)) << endl;

  return 0;
}
```

```
main.cpp: In static member function ‘static double sPoint::Distance(const sPoint&)’:
main.cpp:14:17: error: invalid use of member ‘sPoint::x’ in static member function
  { double dx = x - p.x; double dy = y - p.y;
                ^
main.cpp:17:10: note: declared here
  double x;
         ^
main.cpp:14:38: error: invalid use of member ‘sPoint::y’ in static member function
  { double dx = x - p.x; double dy = y - p.y;
                                     ^
main.cpp:18:10: note: declared here
  double y;
         ^
```

`static` Functions

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include <cmath>
#include "Helper.h"
class sPoint
{
public:
  sPoint(void) : x(0), y(0) { return; }
  sPoint(double _x, double _y) : x(_x), y(_y)
    { return; }
  static double Distance(const sPoint &p1,
    const sPoint &p2)
  { double dx = p1.x – p2.x;
    double dy = p1.y - p2.y;
    return std::hypot(dx, dy); }  double x;
  double y;
};

ostream &operator<<(ostream &s, const sPoint &p)
{s << "(" << p.x << ", " << p.y << ")"; return s;}

int main()
{
  sPoint Origin;
  sPoint Point{1.0, 1.0};
  cout << DUMPVAL(sPoint::Distance(Origin, Point))
<< endl;
  return 0;
}
```

```
Expression: sPoint::Distance(Origin, Point), Type: double, Length:
8, Value: 1.41421
```

Operator Function Details

- Object operators:
  - First parameter is always **`this`**. For binary operators, it is the left hand operand.
  - Assignment operators must return a reference to the object, i.e., **`*this`**.
  - Other operators usually return an <rvalue>, but may return an in-scope <lvalue>.
- Non-object operators:
  - **`this`** is not available. <lhs> and <rhs> must be the only parameters in that order.

How + Works

- **`a + b + c`** evaluates left to right.
  - So, **`a + b`**, is evaluated first and the value returned is added to **`c`**.
- **`std::ostream &operator<< (std::ostream &`**`f`**`, const &`**`obj`**`)`** works the same way – it's just not so obvious.

How << Works For Stream Output Consider This

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include <cmath>
#include "Helper.h"
class sPoint
{
public:
  sPoint(void) : x(0), y(0) { return; }
  sPoint(double _x, double _y) : x(_x), y(_y)
    { return; }

  double x;
  double y;
};

ostream &operator<<(ostream &s, const sPoint &p)
{s << "(" << p.x << ", " << p.y << ")"; return s;}

int main()
{
  (0, cout << "Hello") << " World" << endl;

  return 0;
}
```

```
Hello World
```

How << Works For Stream Output

-
  ```
  ostream &operator<<(ostream &s, const sPoint &p)
  {s << "(") << p.x << ", " << p.y << ")"; return s;}
  ```
  - Write "(" to the ostream s and returns s.

-
  ```
  ostream &operator<<(ostream &s, const sPoint &p)
  {s << p.x << ", " << p.y << ")"; return s;}
  ```
  - Write p.x to the ostream s and returns s.

-
  ```
  ostream &operator<<(ostream &s, const sPoint &p)
  {s << ", ") << p.y << ")"; return s;}
  ```
  - Write ", " to the ostream s and returns s.
- Etc.