CSCI 390 – Special Topics in C++

Lecture 14 (10/4/18)

Time To Turn Off Cell Phones

Example != operator

```
#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; }
  bool operator!=(const sPoint &rhs)
  { return (x != rhs.x) || (y != rhs.y); }


  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 != Point) << endl;
  return 0;
}
```

```
Expression: Origin != Point, Type: bool, Length: 1, Value: 1
```

Example += operator

```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; }
  sPoint &operator+=(const sPoint &rhs)
  { x+=rhs.x; y+=rhs.y; return *this; }

  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};
  Origin += Point;
  cout << DUMPOBJ(Origin) << endl;
  return 0;
}
```

```
Object: Origin, Type: sPoint, Length: 16, Address: 0x7fffc04fc4e0,
Value: (1, 1)
```

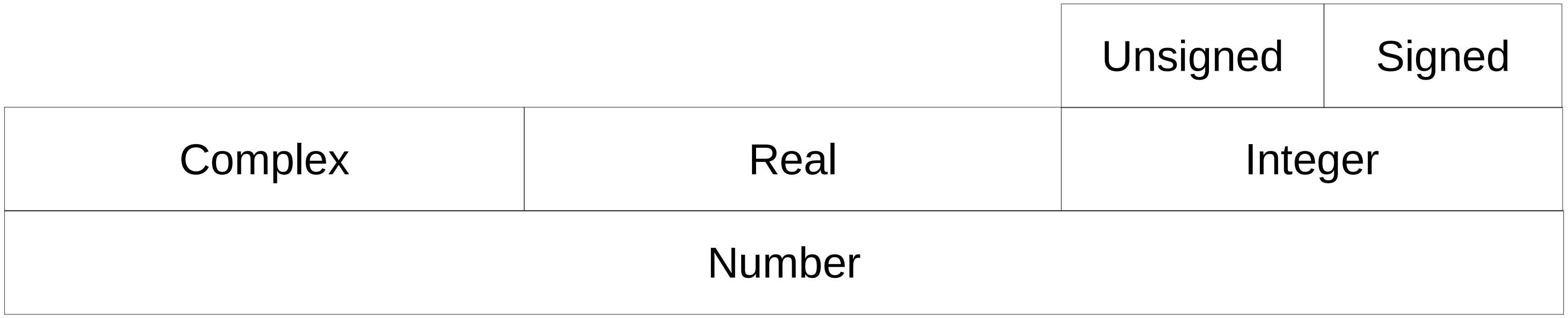
Example += operator

```
#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; }
  sPoint &operator+=(const sPoint &rhs)
  { x+=rhs.x; y+=rhs.y; return *this; }
  sPoint &operator+=(const double &rhs)
  { x+=rhs; y+=rhs; return *this; }

  double x;
  double y;
};

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

int main()
{
  sPoint Origin;
  Origin += 11.0;
  cout << DUMPOBJ(Origin) << endl;
  return 0;
}
```

```
Object: Origin, Type: sPoint, Length: 16, Address: 0x7ffcad7f3cf0,
Value: (11, 11)
```

Object Inheritance

- Sometimes what you are trying to model is hierarchical:
  - If hierarchical relationship is “has a”, it is a member.
  - If hierarchical relationship is “is a”, it is derived



Example

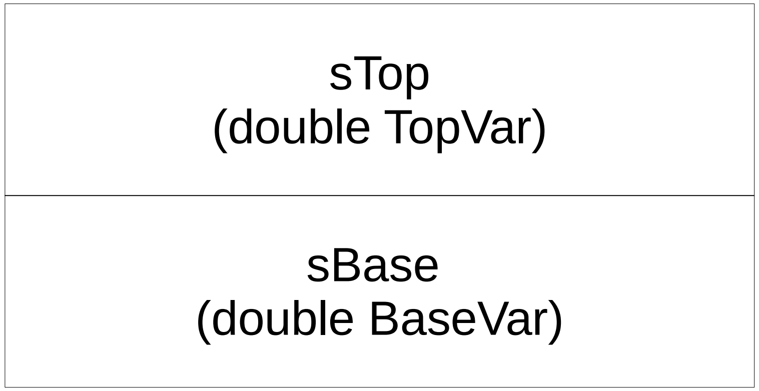
```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"
struct sBase
{
  sBase(void) : BaseVar(0.0) { return; }
  sBase(double _b) : BaseVar(_b) { return; }
  double BaseVar;
};

ostream &operator<<(ostream &s, const sBase &b)
{ s << b.BaseVar; return s; }

struct sTop : sBase
{
  sTop(void) : TopVar(1.0), sBase(1.0) { return; }
  sTop(double _t) : TopVar(_t), sBase(_t)
{ return; }
  double TopVar;
};

ostream &operator<<(ostream &s, const sTop &t)
{ s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
return s; }
```

```
int main()
{
  sBase Base;
  sTop Top;
  cout << Base << endl;
  cout << Top << endl;
  return 0;
}
```

```
0
<1, 1>
```

Example/**`protected`**

- Protected members are only visible to parents and friends.
- We will use this model for the next several slides:



Example

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"
struct sBase
{ protected:
  sBase(void) : BaseVar(0.0) { return; }
  sBase(double _b) : BaseVar(_b) { return; }
  double BaseVar;
};

ostream &operator<<(ostream &s, const sBase &b)
{ s << b.BaseVar; return s; }

struct sTop : sBase
{
  sTop(void) : TopVar(1.0), sBase(1.0) { return; }
  sTop(double _t) : TopVar(_t), sBase(_t)
{ return; }
  double TopVar;
};

ostream &operator<<(ostream &s, const sTop &t)
{ s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
return s; }
```

```
int main()
{
  sBase Base;
  sTop Top;
  cout << Base << endl;
  cout << Top << endl;
  return 0;
}
```

```
main.cpp: In function ‘std::ostream&
operator<<(std::ostream&, const sBase&)’:
main.cpp:10:10: error: ‘double sBase::BaseVar’ is
protected
   double BaseVar;
          ^
main.cpp:13:10: error: within this context
 { s << b.BaseVar; return s; }
          ^
main.cpp: In function ‘std::ostream&
operator<<(std::ostream&, const sTop&)’:
main.cpp:10:10: error: ‘double sBase::BaseVar’ is
protected
   double BaseVar;
          ^
main.cpp:21:37: error: within this context
 { s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
                                     ^
main.cpp: In function ‘int main()’:
main.cpp:8:3: error: ‘sBase::sBase()’ is
protected
   sBase(void) : BaseVar(0.0) { return; }
   ^
main.cpp:26:9: error: within this context
```

The Fix

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"
struct sTop;
struct sBase
{
  friend ostream &operator<<(ostream &s, const
sBase &b);
  friend ostream &operator<<(ostream &s, const
sTop &t);
  sBase(void) : BaseVar(0.0) { return; }
  sBase(double _b) : BaseVar(_b) { return; }
protected:
  double BaseVar;
};
ostream &operator<<(ostream &s, const sBase &b)
{ s << b.BaseVar; return s; }

struct sTop : sBase
{
  sTop(void) : TopVar(1.0), sBase(1.0) { return; }
  sTop(double _t) : TopVar(_t), sBase(_t)
{ return; }
  double TopVar;
};
ostream &operator<<(ostream &s, const sTop &t)
{ s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
return s; }
```

```
int main()
{
  sBase Base;
  sTop Top;
  cout << Base << endl;
  cout << Top << endl;
  return 0;
}
```

```
0
<1, 1>
```

Construction/Destruction Order

- Construction starts at the bottom and works its way up.
- Destruction starts at the top and works its way down.

Example

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

struct sBase
{
  sBase(void) : BaseVar(0.0)
    { cout << "sBase(void)\n"; return; }
  sBase(double _b) : BaseVar(_b)
    { cout << "sBase(double)\n"; return; }
    double BaseVar;
};

ostream &operator<<(ostream &s, const sBase &b)
{ s << b.BaseVar; return s; }

struct sTop : sBase
{
  sTop(void) : TopVar(1.0), sBase(1.0)
    { cout << "sTop(void)\n"; return; }
  sTop(double _t) : TopVar(_t), sBase(_t)
    { cout << "sTop(double)\n"; return; }
  double TopVar;
};

ostream &operator<<(ostream &s, const sTop &t)
{ s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
return s; }
```

```
int main()
{
  sTop Top;
  cout << Top << endl;
  return 0;
}
```

```
sBase(double)
sTop(void)
<1, 1>
```

Example With **new/delete**

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"
struct sBase
{
  sBase(void) : BaseVar(0.0)
    { cout << "sBase(void)\n"; return; }
  sBase(double _b) : BaseVar(_b)
    { cout << "sBase(double)\n"; return; }
    double BaseVar;
  ~sBase(void)
    { cout << "~sBase(void)\n"; return; }
};
ostream &operator<<(ostream &s, const sBase &b)
{ s << b.BaseVar; return s; }
struct sTop : sBase
{
  sTop(void) : TopVar(1.0), sBase(1.0)
    { cout << "sTop(void)\n"; return; }
  sTop(double _t) : TopVar(_t), sBase(_t)
    { cout << "sTop(double)\n"; return; }
  ~sTop(void)
    { cout << "~sTop(void)\n"; return; }
  double TopVar;
};
ostream &operator<<(ostream &s, const sTop &t)
{ s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
return s; }
```

```
int main()
{
  sTop *Top = new sTop(3.0);
  cout << *Top << endl;
  delete Top;

  return 0;
}
```

```
sBase(double)
sTop(double)
<3, 3>
~sTop(void)
~sBase(void)
```

Never Use **malloc/free**

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"
struct sBase
{
  sBase(void) : BaseVar(0.0)
    { cout << "sBase(void)\n"; return; }
  sBase(double _b) : BaseVar(_b)
    { cout << "sBase(double)\n"; return; }
    double BaseVar;
  ~sBase(void)
    { cout << "~sBase(void)\n"; return; }
};
ostream &operator<<(ostream &s, const sBase &b)
{ s << b.BaseVar; return s; }
struct sTop : sBase
{
  sTop(void) : TopVar(1.0), sBase(1.0)
    { cout << "sTop(void)\n"; return; }
  sTop(double _t) : TopVar(_t), sBase(_t)
    { cout << "sTop(double)\n"; return; }
  ~sTop(void)
    { cout << "~sTop(void)\n"; return; }
  double TopVar;
};
ostream &operator<<(ostream &s, const sTop &t)
{ s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
return s; }
```

```
int main()
{
  sTop *Top = (sTop *)malloc(sizeof(sTop));
  free(Top);

  return 0;
}

<No output>
```

The constructor/destructor does not run!

The Compiler is ALWAYS Type Aware

- The Compiler is ALWAYS Type Aware!
  - You can be fooled.
- Virtual functions work their way up the hierarchy to the highest level that matches the function name and parameters. It then executes that function.
  - The hierarchy is stored in a “vtab”.
    - All you need to know is that if an error mentions “vtab” the compiler thinks you are using virtual functions or you need to make something virtual
  - Always make destructors virtual.

Example

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"
struct sBase
{
  sBase(void) : BaseVar(0.0)
    { cout << "sBase(void)\n"; return; }
  sBase(double _b) : BaseVar(_b)
    { cout << "sBase(double)\n"; return; }
    double BaseVar;
  ~sBase(void)
    { cout << "~sBase(void)\n"; return; }
};
ostream &operator<<(ostream &s, const sBase &b)
{ s << b.BaseVar; return s; }
struct sTop : sBase
{
  sTop(void) : TopVar(1.0), sBase(1.0)
    { cout << "sTop(void)\n"; return; }
  sTop(double _t) : TopVar(_t), sBase(_t)
    { cout << "sTop(double)\n"; return; }
  ~sTop(void)
    { cout << "~sTop(void)\n"; return; }
  double TopVar;
};
ostream &operator<<(ostream &s, const sTop &t)
{ s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
return s; }
```

```
int main()
{
  sBase *Obj = new sTop();
  cout << *((sTop *)Obj) << endl;
  delete Obj;

  return 0;
}
```

```
sBase(double)
sTop(void)
<1, 1>
~sBase(void)
```

~sTop did not run!

The Fix

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"
struct sBase
{
  sBase(void) : BaseVar(0.0)
    { cout << "sBase(void)\n"; return; }
  sBase(double _b) : BaseVar(_b)
    { cout << "sBase(double)\n"; return; }
    double BaseVar;
  virtual ~sBase(void)
    { cout << "~sBase(void)\n"; return; }
};
ostream &operator<<(ostream &s, const sBase &b)
{ s << b.BaseVar; return s; }
struct sTop : sBase
{
  sTop(void) : TopVar(1.0), sBase(1.0)
    { cout << "sTop(void)\n"; return; }
  sTop(double _t) : TopVar(_t), sBase(_t)
    { cout << "sTop(double)\n"; return; }
  virtual ~sTop(void)
    { cout << "~sTop(void)\n"; return; }
  double TopVar;
};
ostream &operator<<(ostream &s, const sTop &t)
{ s << "<" << t.TopVar << ", " << t.BaseVar <<
">";
return s; }
```

```
int main()
{
  sBase *Obj = new sTop();
  cout << *((sTop *)Obj) << endl;
  delete Obj;

  return 0;
}
```

```
sBase(double)
sTop(void)
<1, 1>
~sTop(void)
~sBase(void)
```

~sTop DID run!

As part of the destruction of sTop, it works its way down the hierarchy destroying each object along the way.