CSCI 390 – Special Topics in C++

Lecture 15 (10/9/18)

Time To Turn Off Cell Phones

Example Pointer To Object
Array Like

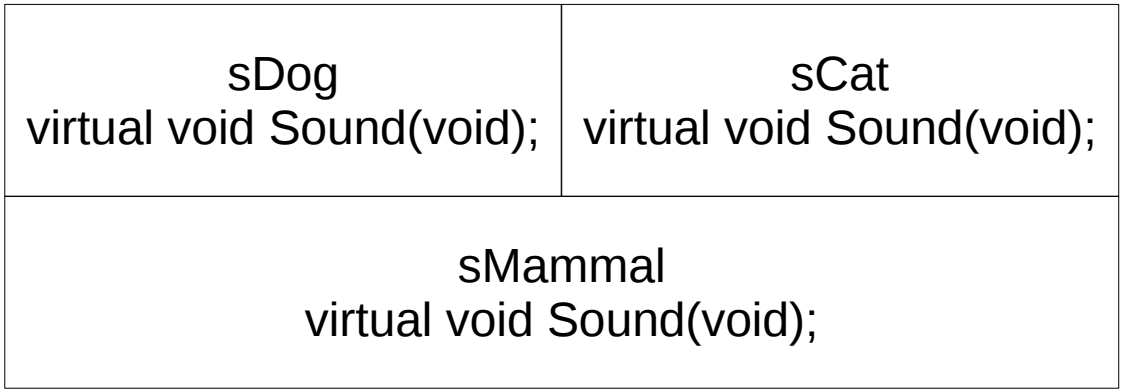
```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;
#include "Helper.h"
struct sBase
{
  sBase(void) : BaseVar(0.0)
    { cout << "sBase(void)\n"; return; }
  sBase(double _b) : BaseVar(_b)
    { cout << "sBase(double)\n"; return; }
    double BaseVar;
  ~sBase(void)
    { 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[2];
  Top[1].TopVar = 23.0;
  Top[1].BaseVar = 31.0;
  cout << Top[0] << endl;
  cout << Top[1] << endl;
  delete [ ] Top;

  return 0;
}
```

```
sBase(double)
sTop(void)
sBase(double)
sTop(void)
<1, 1>
<23, 31>
~sTop(void)
~sBase(void)
~sTop(void)
~sBase(void)
```

Virtual Functions

- Once a virtual function is defined, all parent classes must also define the function.
- Virtual functions must be inside an object.
- We will use this model for the discussion on virtual functions:



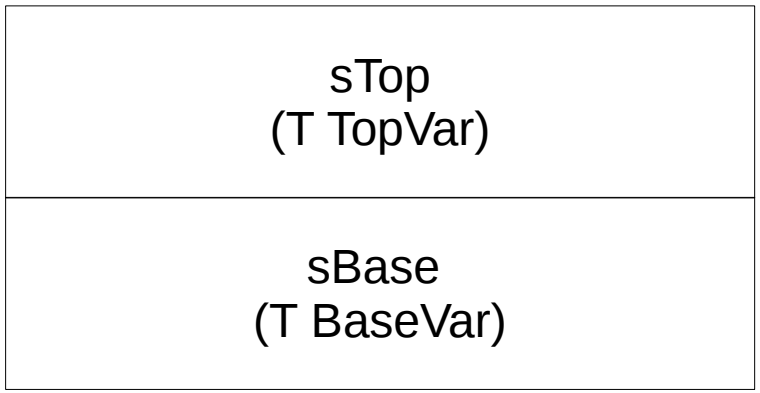
Virtual Function Example

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>

struct sMammal
{
  sMammal(void)  { return; }
  ~sMammal(void) { return; }

  virtual void Sound(void)
    { cout << "Error" << endl; return; }
};

struct sCat : sMammal
{
  sCat(void)  { return; }
  ~sCat(void) { return; }

  virtual void Sound(void)
    { cout << "Meow"; return; }
};

struct sDog : sMammal
{
  sDog(void)  { return; }
  ~sDog(void) { return; }

  virtual void Sound(void)
    { cout << "Woof"; return; }
};
```

```
int main()
{
  sMammal *Dog = new sDog();
  sMammal *Cat = new sCat();

  cout << "Dog goes: ";
  Dog->Sound();
  cout << endl;

  cout << "Cat goes: ";
  Cat->Sound();
  cout << endl;

  delete Dog;
  delete Cat;

  return 0;
}

Dog goes: Woof
Cat goes: Meow
```

Template Objects

- We will use this model for this discussion:



Example With Pointers

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;

template<typename T>
struct sBase
{
  sBase(void) : BaseVar(T(0.0)) { return; }
  sBase(T _b) : BaseVar(_b) { return; }
  virtual ~sBase(void) { return; }
  virtual std::ostream &ID(std::ostream &f) const
{ f << BaseVar; return f; }
  T BaseVar;
};

template<typename T>
std::ostream &operator<<(std::ostream &f, const
sBase<T> &Obj)
  { return Obj.ID(f); }

template<typename T>
struct sTop : sBase<T>
{
  sTop(void) : TopVar(T(1.0)), sBase<T>(T(1.0))
{ return; }
  sTop(T _t) : TopVar(_t), sBase<T>(_t)
{ return; }
  virtual ~sTop(void) { return; }
  virtual std::ostream &ID(std::ostream &f) const
{ f << "(" << TopVar << ", " << sBase<T>::BaseVar
<< ")"; return f; }
  T TopVar;
};

template<typename T>
std::ostream &operator<<(std::ostream &f, const
sTop<T> &Obj)
  { return Obj.ID(f); }

int main()
{
  sBase<double> *Base = new sBase<double>();
  sBase<double> *Top = new sTop<double>();

  cout << "Base: " << *Base  << endl;
  cout << " Top: " << *Top   << endl;

  delete Base;
  delete Top;

  return 0;
}
```

```
Base: 0
 Top: (1, 1)
```

Example

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;

template<typename T>
struct sBase
{
  sBase(void) : BaseVar(T(0.0)) { return; }
  sBase(T _b) : BaseVar(_b) { return; }
  virtual ~sBase(void) : { return; }
  T BaseVar;
};

template<typename T>
struct sTop : sBase<T>
{
  sTop(void) : TopVar(T(1.0)), sBase<T>(T(1.0))
{ return; }
  sTop(T _t) : TopVar(_t), sBase<T>(_t)
{ return; }
  virtual ~sTop(void) { return; }
  T TopVar;
};
```

```
int main()
{
  sBase<double> Base;
  sTop<double> Top;
  cout << Base.BaseVar << endl;
  cout << Top.TopVar << ", " << Top.BaseVar <<
endl;

  return 0;
}

0
1, 1
```

Example With **operator<<**

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;

template<typename T>
struct sBase
{
  sBase(void) : BaseVar(T(0.0)) { return; }
  sBase(T _b) : BaseVar(_b) { return; }
  virtual ~sBase(void) { return; }
  T BaseVar;
};

template<typename T>
std::ostream &operator<<(std::ostream &f, const
sBase<T> &Obj)
  { f << Obj.BaseVar; return f; }
```

```
template<typename T>
struct sTop : sBase<T>
{
  sTop(void) : TopVar(T(1.0)), sBase<T>(T(1.0)) {
return; }
  sTop(T _t) : TopVar(_t), sBase<T>(_t) { return;
}
  virtual ~sTop(void) { return; }
  T TopVar;
};

template<typename T>
std::ostream &operator<<(std::ostream &f, const
sTop<T> &Obj)
  { f << "(" << Obj.TopVar << ", " << Obj.BaseVar
<< ")"; return f; }

int main()
{
  sBase<double> Base;
  sTop<double> Top;

  cout << "Base: " << Base  << endl;
  cout << " Top: " << Top   << endl;

  return 0;
}

Base: 0
 Top: (1, 1)
```

Example With Virtual Functions

```
#include <iostream>
using std::cout; using std::endl;
#include <ostream>
using std::ostream;

template<typename T>
struct sBase
{
  sBase(void) : BaseVar(T(0.0)) { return; }
  sBase(T _b) : BaseVar(_b) { return; }
  virtual ~sBase(void) { return; }
  virtual std::ostream &ID(std::ostream &f) const
{ f << BaseVar; return f; }
  T BaseVar;
};

template<typename T>
std::ostream &operator<<(std::ostream &f, const
sBase<T> &Obj)
  { return Obj.ID(f); }
```

```
template<typename T>
struct sTop : sBase<T>
{
  sTop(void) : TopVar(T(1.0)), sBase<T>(T(1.0)) {
return; }
  sTop(T _t) : TopVar(_t), sBase<T>(_t) { return;
}
  virtual ~sTop(void) { return; }
  virtual std::ostream &ID(std::ostream &f) const
{ f << "(" << TopVar << ", " << sBase<T>::BaseVar
<< ")"; return f; }
  T TopVar;
};

template<typename T>
std::ostream &operator<<(std::ostream &f, const
sTop<T> &Obj)
  { return Obj.ID(f); }

int main()
{
  sBase<double> Base;
  sTop<double> Top;

  cout << "Base: " << Base  << endl;
  cout << " Top: " << Top   << endl;

  return 0;
}
```

```
Base: 0
 Top: (1, 1)
```

C++ Preprocessor

- Runs before actual compilation.
  - Preprocessor output is passed to compiler.
  - Preprocessor is a macro language.
    - Macros expand to C++ source.
    - Macros do not expand inside quotes.
    - Think of it as a meta language for C++.
  - Preprocessor can conditionally include source.
    - Often used to configue for an environment.

C++ Preprocessor Predefined Macros

__cplusplus denotes the version of C++ standard that is being used, expands to value 199711L(until C++11), 201103L(C++11), 201402L(C++14), or 201703L(C++17)

__STDC_HOSTED__ (C++11) expands to the integer constant 1 if the implementation is hosted (runs under an OS), 0 if freestanding (runs without an OS)

__FILE__ expands to the name of the current file, as a character string literal, can be changed by the #line directive

__LINE__ expands to the source file line number, an integer constant, can be changed by the #line directive

__DATE__ expands to the date of translation, a character string literal of the form "Mmm dd yyyy". The first character of "dd" is a space if the day of the month is less than 10. The name of the month is as if generated by std::asctime()

__TIME__ expands to the time of translation, a character string literal of the form "hh:mm:ss"

C++ Preprocessor-Like Predefined Variable

__func__ Within every function-body, the special predefined variable __func__ with block scope and static storage duration is available, as if defined immediately after the opening brace by:

```
static const char __func__[] = "function name";
```

Note: The is not implemented as a macro because the preprocessor does not compile and thus has no knowledge of C++ syntax. It does not even know what a function is. The compiler declares this static variable for each function, and delberately makes it look like a predefined preprocessor macro.

Example Predefined Macro Expansion

```
#include <iostream>
using std::cout;  using std::endl;

#include "Helper.h"

int main(void)
{
    cout <<"__cplusplus: " << __cplusplus << endl;
    cout << DUMPVAL(__cplusplus) << endl;

    cout <<"__STDC_HOSTED__: " <<
       __STDC_HOSTED__ << endl;
    cout << DUMPVAL(__STDC_HOSTED__) << endl;

    cout <<"__FILE__: " << __FILE__ << endl;
    cout << DUMPVAR(__FILE__) << endl;

    cout <<"__LINE__: " << __LINE__ << endl;
    cout << DUMPVAL(__LINE__) << endl;

    cout <<"__DATE__: " << __DATE__ << endl;
    cout << DUMPVAR(__DATE__) << endl;

    cout <<"__TIME__: " << __TIME__ << endl;
    cout << DUMPVAR(__TIME__) << endl;

    cout << "__func__: " << __func__ << endl;
    cout << DUMPVAR(__func__) << endl;

   return 0;
}
```

```
__cplusplus: 201103
Expression: __cplusplus, Type: long, Length: 8,
Value: 201103

__STDC_HOSTED__: 1
Expression: __STDC_HOSTED__, Type: int, Length:
4, Value: 1

__FILE__: main.cpp
Variable: __FILE__, Type: char [9], Length: 9,
Address: 0x401776, Value: main.cpp

__LINE__: 17
Expression: __LINE__, Type: int, Length: 4,
Value: 18

__DATE__: Oct  8 2018
Variable: __DATE__, Type: char [12], Length: 12,
Address: 0x4017be, Value: Oct  8 2018

__TIME__: 20:20:01
Variable: __TIME__, Type: char [9], Length: 9,
Address: 0x4017de, Value: 20:20:01

__func__: main
Variable: __func__, Type: char [5], Length: 5,
Address: 0x401804, Value: main
```

C++ Preprocessor
Defining Simple User Macros

- Define syntax:
  - **`#define`** <macro id> <macro text>
  - By convention, <macro ids> are uppercase so that the reader knows it is a macro.
- Once defined, works just like predefined macros.
- Undefine syntax:
  - **`#undef`** <macro id>