Data Structures

February 10 – Inheritance and Templates
G Carl Evans
#include "Cube.h"

int main() {
    cs225::Cube c(10);
    c = c;
    return 0;
}
Example:

```cpp
#include "Cube.h"
...
40  Cube& Cube::operator=(const Cube &other) {
41      _destroy();
42      _copy(other);
43      return *this;
44  }
```
Inheritance
#pragma once

#include "Shape.h"

class Square : public Shape {

public:
    double getArea() const;

private:
    // Nothing!
};

class Shape {

public:
    Shape();
    Shape(double length);
    double getLength() const;

private:
    double length_;
};
Derived Classes

[Public Members of the Base Class]:

```cpp
5 int main() { 
6   Square sq; 
7   sq.getLength(); // Returns 1, the length init’ed
8   // by Shape’s default ctor
... ... 
... }
```

[Private Members of the Base Class]:
Polymorphism

The idea that a single interface may take multiple types or that a single symbol may be different types.

In Object-Orientated Programming (OOP) a key example is that a single object may take on the type of any of its base types.
Virtual
Cube.cpp

```cpp
1  Cube::print_1() {
2      cout << "Cube" << endl;
3  }
4
5  Cube::print_2() {
6      cout << "Cube" << endl;
7  }
8
9  virtual Cube::print_3() {
10     cout << "Cube" << endl;
11  }
12
13  virtual Cube::print_4() {
14     cout << "Cube" << endl;
15  }
16
17  // In .h file:
18  virtual print_5() = 0;
19
20
21
22```

RubikCube.cpp

```cpp
1  // No print_1() in RubikCube.cpp
2
3
4
5  RubikCube::print_2() {
6      cout << "Rubik" << endl;
7  }
8
9  // No print_3() in RubikCube.cpp
10
11
12
13  RubikCube::print_4() {
14      cout << "Rubik" << endl;
15  }
16
17
18  RubikCube::print_5() {
19      cout << "Rubik" << endl;
20  }
21
22```
## Runtime of Virtual Functions

<table>
<thead>
<tr>
<th></th>
<th>Cube c;</th>
<th>RubikCube c;</th>
<th>RubikCube rc; Cube &amp;c = rc;</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual-main.cpp</td>
<td>c.print_1();</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c.print_2();</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c.print_3();</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c.print_4();</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c.print_5();</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why Polymorphism?
class Animal {
    public:
    void speak() {
    }
};

class Dog : public Animal {
    public:
    void speak() {
    }
};

class Cat : public Animal {
    public:
    void speak() {
    }
};
Abstract Class:

[Requirement]:

[Syntax]:

[As a result]:
```cpp
virtual-dtor.cpp

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
</table>
| 15   | class Cube {
| 16   |     public: |
| 17   |       ~Cube(); |
| 18   | }; |
| 19   | |
| 20   | class RubikCube : public Cube {
| 21   |     public: |
| 22   |       ~RubikCube(); |
| 23   | }; |
```
class PNG {
    public:
        PNG();
        PNG(unsigned int width, unsigned int height);
        PNG(PNG const & other);
        ~PNG();
        PNG & operator=(PNG const & other);
        bool operator==(PNG const & other) const;
        bool readFromFile(string const & fileName);
        bool writeToFile(string const & fileName);
        HSLAPixel & getPixel(unsigned int x, unsigned int y) const;
        unsigned int width() const;
        // ...

    private:
        unsigned int width_;  
        unsigned int height_;  
        HSLAPixel *imageData_;  
        void _copy(PNG const & other);
};
Abstract Data Type
List ADT
What types of “stuff” do we want in our list?

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Templates
```cpp
template1.cpp

T maximum(T a, T b) {
    T result;
    result = (a > b) ? a : b;
    return result;
}
```
#pragma once

```cpp
class List {
    public:
```

```cpp
private:
```

```cpp`
};
```