Virtual

- The virtual keyword tells the system to expect to override the behavior of a class by its derived type.

Example:

```
Cube.cpp
RubikCube.cpp

Cube::print_1() {
    cout << "Cube" << endl;
}

Cube::print_2() {
    cout << "Cube" << endl;
}

virtual Cube::print_3() {
    cout << "Cube" << endl;
}

virtual Cube::print_4() {
    cout << "Cube" << endl;
}

// In .h file:
virtual print_5() = 0;

RubikCube::print_2() {
    cout << "Rubik" << endl;
}

RubikCube::print_4() {
    cout << "Rubik" << endl;
}

RubikCube::print_5() {
    cout << "Rubik" << endl;
}
```

Method Dispatch Rules

1) 

2) 

3) 

Polymorphism

Object-Orientated Programming (OOP) concept that a single object may take on the type of any of its base types.

- A RubikCube may polymorph itself to a Cube
- A Cube cannot polymorph to be a RubikCube (base types only)

Why Polymorphism? Suppose you’re managing an animal shelter that adopts cats and dogs:

Option 1 – No Inheritance

```
animalShelter.cpp

1 | Cat & AnimalShelter::adopt() { ... }
2 | Dog & AnimalShelter::adopt() { ... }
3 | ...
```

Option 2 – Inheritance

```
animalShelter.cpp

1 | Animal & AnimalShelter::adopt() { ... }
```

Pure Virtual Methods

In Cube, print_5() is a pure virtual method:

```
Cube.h

1 | virtual Cube::print_5() = 0;
```

A pure virtual method does not have a definition and makes the class and abstract class.

Abstract Data Types (ADT):

<table>
<thead>
<tr>
<th>List ADT - Purpose</th>
<th>Function Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
List Implementation
What types of List do we want?

C++ Templates:
1.
2.
3.

Templated Functions:

functionTemplate1.cpp

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

Where to put templated code?

Templated Classes:

List.h

```
#pragma once

class List {
    public:

    private:

};
```

Two Basic Implementations of List:
1.
2.

Linked Memory:

List.h

```
class ListNode {
    T & data;
    ListNode * next;
    ListNode(T & data) : data(data), next(NULL) {} 
};
```

CS 225 – Things To Be Doing:
1. mp_stickers extra credit due Today!
2. Daily POTDs