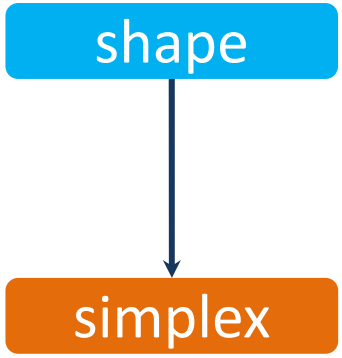
CS100
Introduction to Programming

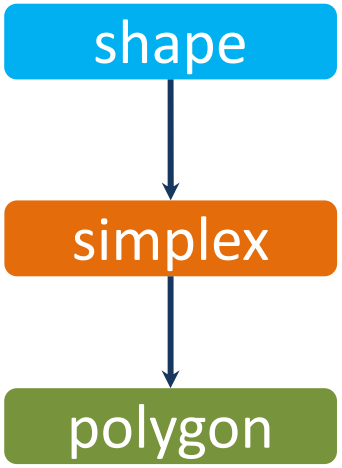
Lecture 10. Multiple Inheritance and Polymorphism

Recall on Inheritance

- **What is inheritance**
  - A way to derive (augment) a new class based on existing class(es)
  - A way to reuse the existing implementations



One-level inheritance



Multi-level inheritance

Another example of inheritance

- **How can we represent books with classes?**
  - "enum" data type
    - A user defined data type
    - Used to assign names to integral constants
    - The names make a program easy to read and maintain

```
enum State {Working, Failed};
enum State {Working=1, Failed};
enum State {Working=1, Failed=0};
```

Another example of inheritance

- **How can we represent books with classes?**
  - A parent class “Book”
  - Define general data and operations

```
enum BOOK_TYPE
{
    BOOK_STORY,
    BOOK_SCIENCE
};
```

```
class Book
{
public:
    void print_content();
    BOOK_TYPE get_type();

    Book();
    ~Book();
protected:
    char* m_content;
    int m_len;

    BOOK_TYPE m_type;
};
```

Another example of inheritance

- **How can we represent books with classes?**
  - Implementing the general parent class

```
book::book()
{
    m_content = NULL;
    m_len = 0;
}

book::~book()
{
    if (m_content != NULL)
        delete []m_content;
}
```

```
BOOK_TYPE book::get_type()
{
    return m_type;
}

void book::print_content()
{
    if (m_content != NULL)
        printf("%s\n", m_content);
}
```

Another example of inheritance

- **How can we represent books with classes?**
  - With this general class, we can define more specific book classes : story book

```
class story_book : public book
{
public:
    void set_content(const char* content);
    story_book(const char* content=NULL);
};
```

The story book inherits from “book”, with specific contents.

Another example of inheritance

- **How can we represent books with classes?**
  - Implementation of story book

```
story_book::story_book(const char* content)
{
    const char* prefix_char = "[Story]:";
    int len = int(strlen(content)+strlen(prefix_char));

    if (m_content == NULL)
        m_content = new char[len + 1];

    memcpy(&m_content[0], prefix_char, strlen(prefix_char));
    strcpy(&m_content[strlen(prefix_char)], content);

    m_len = len + 1;
    m_type = BOOK_STORY;
}
```

Another example of inheritance

- **How can we represent books with classes?**
  - Implementation of story book

```
void story_book::set_content(const char* content)
{
    const char* prefix_char = "[Story]:";
    int len = int(strlen(content) + strlen(prefix_char));

    if (m_content != NULL && m_len <= len)
    {
        delete[]m_content;
        m_content = new char[len + 1];
    }

    memcpy(&m_content[0], prefix_char,
    strlen(prefix_char));
    strcpy(&m_content[strlen(prefix_char)], content);
    m_len = len + 1;
}
```

Another example of inheritance

- **How can we represent books with classes?**
  - With this general class, we can define more specific book classes : science book

```
class science_book : public book
{
public:
    void set_content(const char* content);
    science_book(const char* content = NULL);
};
```

Another example of inheritance

- **How can we represent books with classes?**
  - Implementation of science book

```
science_book::science_book(const char* content)
{
    const char* prefix_char = "[Science]:";

    int len = int(strlen(content) + strlen(prefix_char));
        if (m_content == NULL)

    m_content = new char[len + 1];
    memcpy(&m_content[0], prefix_char, strlen(prefix_char));
    strcpy(&m_content[strlen(prefix_char)], content);

    m_len = len + 1;
    m_type = BOOK_SCIENCE;
}
```

Another example of inheritance

- **How can we represent books with classes?**
  - Implementation of science book

```
void science_book::set_content(const char* content)
{
    const char* prefix_char = "[Science]:";
    int len = int(strlen(content) + strlen(prefix_char));

    if (m_content != NULL && m_len <= len)
    {
        delete[]m_content;
        m_content = new char[len + 1];
    }

    memcpy(&m_content[0], prefix_char, strlen(prefix_char));
    strcpy(&m_content[strlen(prefix_char)], content);
    m_len = len + 1;
}
```

Another example of inheritance

- **Use of the child classes**

```
int main()
{
    story_book story("This is a story book.");
    story.print_content();

    science_book science("This is a science book.");
    science.print_content();

    return 0;
}
```

```
[Story]:This is a story book.
[Science]:This is a science book.
```

Another example of inheritance

- **Now we further augment the child classes**
  - Each augmented with a “print_content” function
  - They have their own specific implementations

```
class story_book : public book{
public:
    void print_content();
    void set_content(const char*
    content);
    story_book(const char* content=NULL);
};

class science_book : public book{
public:
    void print_content();
    void set_content(const char* content);
    science_book(const char* content = NULL);
};
```

Another example of inheritance

- **And we can add specific implementations**
  - Each augmenting the parent class function
  - Call parent's function within the overridden functions

```
void story_book::print_content(){
    printf("Book Type 1 ");
    book::print_content();
}
```

```
void science_book::print_content(){
    printf("Book Type 2 ");
    book::print_content();
}
```

Another example of inheritance

- **Use of the child classes again**

```
int main()
{
    story_book story("This is a story book.");
    story.print_content();

    science_book science("This is a science book.");
    science.print_content();

    return 0;
}
```

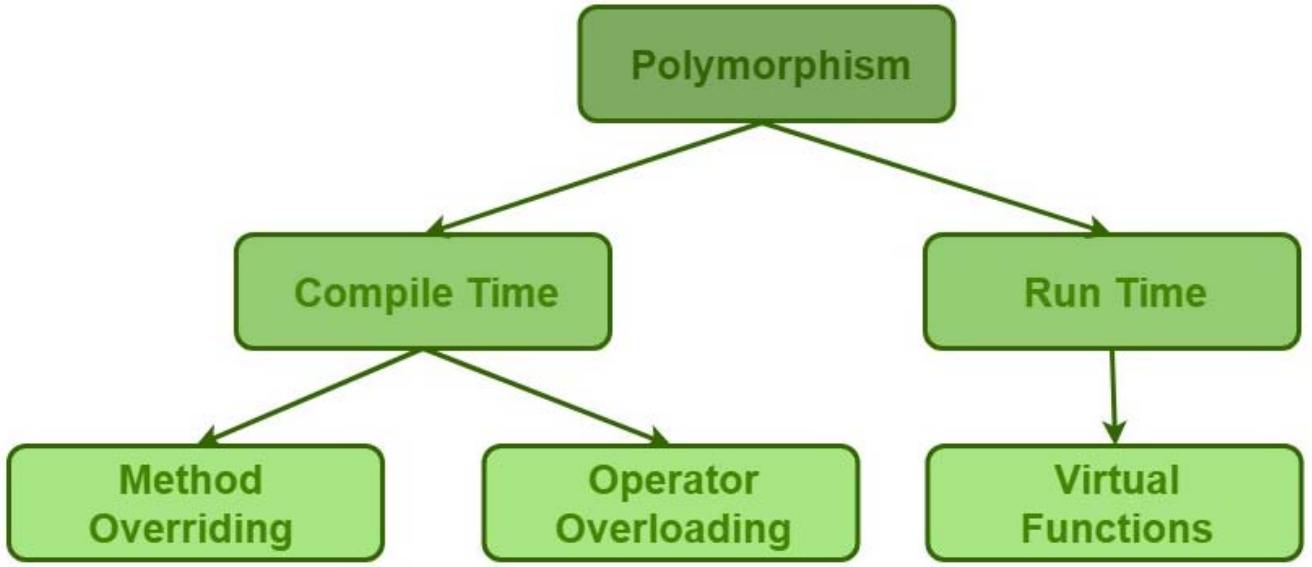
```
Book Type 1 [Story]:This is a story book.
Book Type 2 [Science]:This is a science book.
```

Polymorphism

- **The word polymorphism means**
  - Having many forms
  - Real life example of polymorphism
    - A person at the same time can have different characteristic
    - Like a man at the same time is a father, a husband, an employee

Polymorphism

- **In C++ polymorphism is mainly divided into two types**
  - Compile time polymorphism
  - Runtime polymorphism



Polymorphism

- **Function overloading**
  - Recall our previous example
  - The same function name, different input parameters

```
int add(int x, int y)
{
      return x + y;
}

long add(long x, long y)
{
      return x + y;
}
```

```
float add(float x, float y)
{
      return x + y;
}

double add(double x, double y)
{
      return x + y;
}
```

Polymorphism

- **Polymorphism over class hierarchies**
  - Member function (method) overriding

```
class Book
{
public:
    void print_content();
    BOOK_TYPE get_type();

    Book();
    ~Book();
protected:
    char* m_content;
    int m_len;

    BOOK_TYPE m_type;
};
```

```
class story_book : public book{
public:
    void print_content();
    void set_content(const char*
    content);
    story_book(const char* content=NULL);
};

class science_book : public book{
public:
    void print_content();
    void set_content(const char* content);
    science_book(const char* content = NULL);
};
```

What's the problem for method overriding?

- **Let's look at the "book" example again**
  - "book" is the shared parent class
  - We want to use "print_content" to print each content

```
int main(){
    story_book story
                ("This is a story book.");
    science_book science
                ("This is a science book.");

    book* p_book = (book*)&story;
    p_book->print_content();
    p_book = (book*)&science;
    p_book->print_content();

    return 0;
}
```

```
[Story]:This is a story book.
[Science]:This is a science book.
```

```
void book::print_content(){
    if (m_content != NULL)
        printf("%s\n", m_content);
}

void story_book::print_content(){
    printf("Book Type 1 ");
    book::print_content();
}

void science_book::print_content(){
    printf("Book Type 2 ");
    book::print_content();
}
```

What's the problem for method overriding?

- **Let's look at the "book" example again**
  - Our goal is not achieved

```
story_book story("This is a story book.");
science_book science("This is a science book.");

book* p_book = (book*)&story;
p_book->print_content();
```

When we call this function, we hope to call story_book::print_cotent, NOT book::print_cotent, but this is not achievable by a "book" pointer

```
p_book = (book*)&science;
p_book->print_content();
```

When we call this function, we hope to call science_book::print_cotent, NOT book::print_cotent, but this is not achievable either

Run-Time Polymorphism

- **This needs run-time polymorphism**
  - The program can recognize which child class a pointer converted to a parent class comes from
  - What we want to achieve?

```
story_book story("This is a story book.");
science_book science("This is a science book.");

book* p_book = (book*)&story;
p_book->print_content();

p_book = (book*)&science;
p_book->print_content();
```

```
Book Type 1 [Story]:This is a story book.
Book Type 2 [Science]:This is a science book.
```

Virtual Function

- **A virtual member function in classes**
  - Declared within a parent class and overridden by one or more child classes
  - Declared with a **virtual** keyword in parent class
    - If we change the "book" definition as:

```
class Book
{
public:
    virtual void print_content();
    BOOK_TYPE get_type();

    Book();
    ~Book();
protected:
    ...
};
```

This definition will achieve our goal!

Virtual Function

- **Now we can write a generic print function**
  - Print the desired content by the same “book” pointer

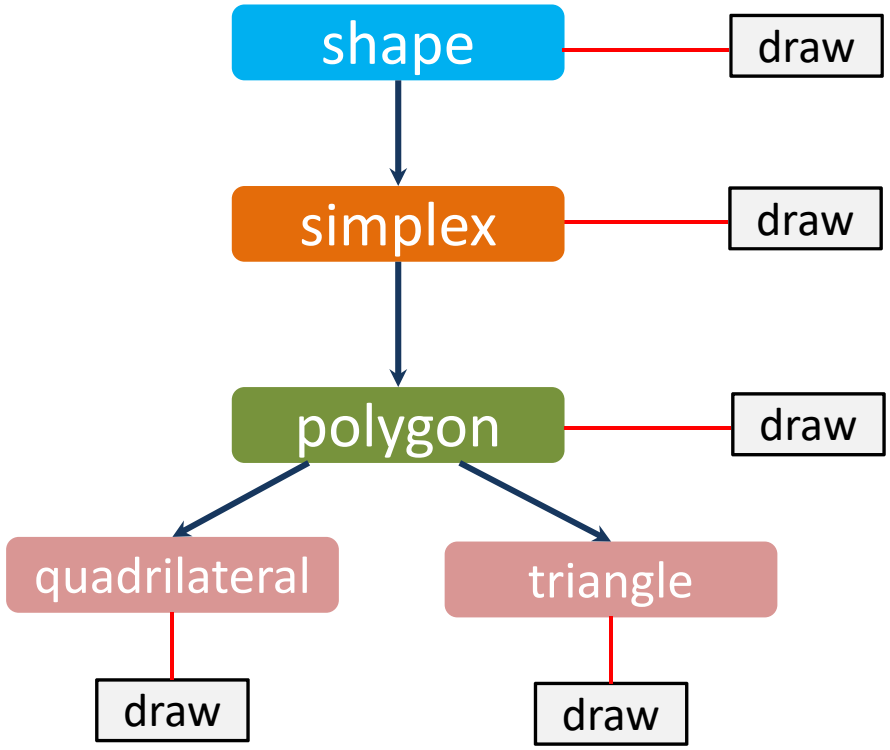
```
void generic_book_print(book* p_book){
    p_book->print_content();
}

int main(){
    story_book story("This is a story book.");
    science_book science("This is a science book.");

    generic_book_print((book*)&story);
    generic_book_print((book*)&science);

    return 0;
}
```

```
Book Type 1 [Story]:This is a story book.
Book Type 2 [Science]:This is a science book.
```

See the run-time polymorphism here!

More Examples on Virtual Function

- **Generic draw for shapes**
  - Suppose you want to draw different shapes



More Examples on Virtual Function

- **Generic draw for shapes**
  - Definition/implemention of draw() in shape

```
class shape2D
{
public:
    virtual void draw();

    float get_boundary_length();
    float get_area();
    shape2D();
protected:
    float m_boundary_length;
    float m_area;
};
```

```
void shape2D::draw()
{
    printf("\n--->\n");
    printf("You have drawn from shape.\n");
    printf("<---\n");
}
```

More Examples on Virtual Function

- **Generic draw for simplex**
  - Definition/implemention of draw() in simplex

```
class simplex2D : public shape2D
{
public:
    void draw();

    int get_vertex_count();
    int get_edge_count();

    simplex2D();
protected:
    int m_vertex_count;
    int m_edge_count;
};
```

```
void simplex2D::draw()
{
    printf("\n--->\n");
    printf("You have drawn from simplex.\n");
    printf("<---\n");
}
```

More Examples on Virtual Function

- **Generic draw for simplex**
  - Definition/implemention of draw() in polygon

```
class polygon2D : public simplex2D
{
public:
    void draw();

    void calc_boundary_length();

    polygon2D();
    polygon2D(vertex2D* p_vertex, int
    vertex_count);
    ~polygon2D();
protected:
    vertex2D* m_vertex;
    int m_vertex_count;
    int m_edge_count;
};
```

```
void polygon2D::draw()
{
    printf("\n--->\n");
    printf("You have drawn from polygon.\n");
    printf("<---\n");
}
```

More Examples on Virtual Function

- **Generic draw for simplex**
  - Other draw() in further derived classes can be declared and implemented similarly

```
void triangle2D::draw()
{
    printf("\n--->\n");
    printf("You have drawn from triangle.\n");
    printf("<---\n");
}


void quad2D::draw()
{
printf("\n--->\n");
printf("You have drawn from quadrilateral.\n");
printf("<---\n");
}
```

More Examples on Virtual Function

- **A generic draw class**
  - You can define a class to draw a set of objects

```
class generic_draw
{
public:
    ...
    void draw_shape(shape2D* p_shape);
    ...
};

void generic_draw::draw_shape(shape2D* p_shape)
{
    p_shape->draw();
}
```

More Examples on Virtual Function

- **Use the generic draw**
  - Draw different kinds of shapes with the same interface

```
int main()
{
    generic_draw draw;

    polygon2D p;
    triangle2D t;
    quad2D q;

    draw.draw_shape((shape2D*)&p);
    draw.draw_shape((shape2D*)&t);
    draw.draw_shape((shape2D*)&q);

    return 0;
}
```

```
--->
You have drawn from polygon.
<---

--->
You have drawn from triangle.
<---

--->
You have drawn from quadrilateral.
<---
```

Virtual Destructor

- **Consider a two-level derived class**
  - Dynamic allocation is required in the class
  - The parent class

```
class string_base
{
public:
    virtual char* get_string() { return NULL; };
    virtual void print_string() {};

    string_base() {};
    ~string_base() {};
};
```

Virtual Destructor

- **Consider a two-level derived class**
  - Dynamic allocation is required in the class
  - The child class

```
class string : public string_base
{
public:
    char* get_string();
    void print_string();

    string(const char* str);
    ~string();
private:
    char* str;
    int num;
};
```

```
string::string(const char* str){
    int len = strlen(str);
    this->str = new char[len + 1];
    strcpy(this->str, str);
}

string::~string(){
    if (str != NULL)
        delete []str;
    printf("The string has been deleted.\n");
}

char* string::get_string(){
    return str;
}

void string::print_string(){
    printf("%s\n", str);
}
```

Virtual Destructor

- **What we want to achieve**
  - If we only have the pointer of a parent class
  - But we want to delete the pointer object

```
int main()
{
    string*p_str=new string("This is a test string.");

    string_base* p_str_base = (string_base*)p_str;
    p_str_base->print_string();

    delete p_str_base;

    return 0;
}
```

This is a test string.

This is not achievable!

Virtual Destructor

- **What we want to achieve**
  - If we only have the pointer of a parent class
  - But we want to delete the pointer object
    - How to change?

```
class string_base
{
public:
    virtual char* get_string() { return NULL; };
    virtual void print_string() {};

    string_base() {};
    virtual ~string_base() {};
};
```

Virtual Destructor

- **What we want to achieve?**
  - If we only have the pointer of a parent class
  - But we want to delete the pointer object
    - Now we repeat again

```
int main()
{
    string*p_str=new string("This is a test string.");

    string_base* p_str_base = (string_base*)p_str;
    p_str_base->print_string();

    delete p_str_base;

    return 0;
}
```

```
This is a test string.
The string has been deleted.
```

Pure Virtual Function

- **A special kind of virtual function**
  - A virtual function without implementation part
  - Look at the "string_base" class again

```
class string_base
{
public:
    virtual char* get_string() { return NULL; };
    virtual void print_string() {};

    string_base() {};
    virtual ~string_base() {};
};
```

Useless; act like an interface that provides functions for the child classes

Pure Virtual Function

- **A special kind of virtual function**
  - A virtual function without implementation part
  - Now we modify as:

```
class string_base
{
public:
    virtual char* get_string() = 0;
    virtual void print_string() = 0;

    string_base() {};
    virtual ~string_base() {};
};
```

Pure Virtual Function

- **A special kind of virtual function**
  - No that virtual functions should not be implemented (compile error)
  - It only has a pointer to the implementations in the child class

- **Abstract class**
  - A class containing pure virtual functions
  - Cannot be used to declare class object (variable)
  - Use pointers should be used

Abstract Class & Pure Virtual Function

- Look again the usage with abstract class pointer

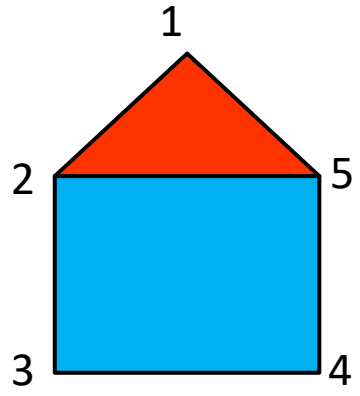
```
int main()
{
    string*p_str=new string("This is a test string.");

    string_base* p_str_base = (string_base*)p_str;

    p_str_base->print_string();

    delete p_str_base;

    return 0;
}
```

Now we use pointer of an abstract class

Multiple Inheritance

- **A class can be inherited from multiple parent classes**
  - Can specify different access when inheriting

```
class house2D :
        public triangle2D, public quad2D
{
public:
    void draw();
    house2D(const vertex2D v[5]);
};
```

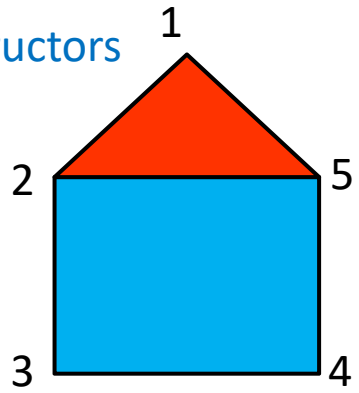


Multiple Inheritance

- **A class can be inherited from multiple parent classes**
  - Can specify different access when inheriting

```
house2D::house2D(const vertex2D v[5])
: triangle2D(v[0], v[1], v[4]),
  quad2D(v[1], v[2], v[3], v[4])
{
    //triangle2D::triangle2D(v[0], v[1], v[4]);
    //quad2D::quad2D(v[1], v[2], v[3], v[4]);
}

void house2D::draw()
{
    printf("\n--->\n");
    printf("You have drawn from house.\n");
    printf("<---\n");
}
```

Call parents' constructors



Multiple Inheritance

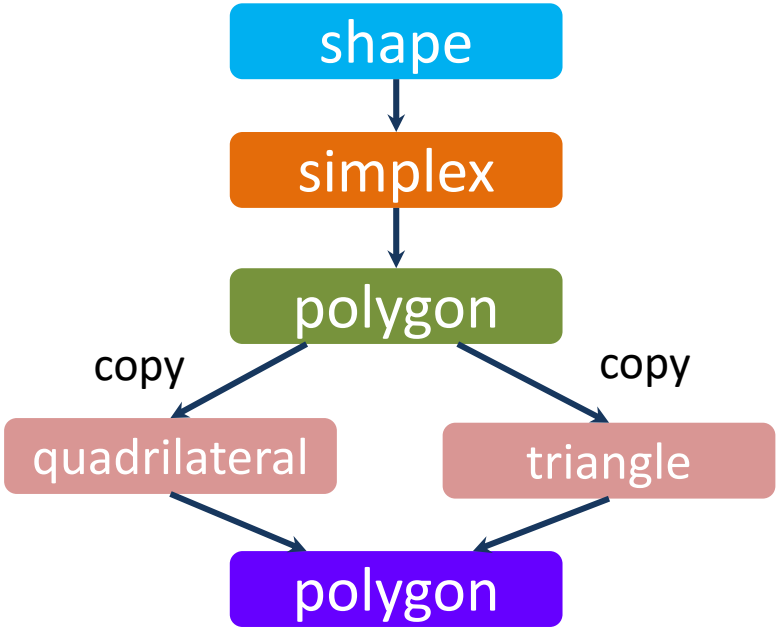
- **A class can be inherited from multiple parent classes**
  - But if we still want to have virtual function polymorphism?

```
int main(){
    vertex2D v[5];
    v[0] = vertex2D(0, 10);
    v[1] = vertex2D(-6, 5);
    v[2] = vertex2D(-6, -1);
    v[3] = vertex2D(6, -1);
    v[4] = vertex2D(6, 5);

    house2D h(v);
    shape2D* p_s = (shape2D*)&h;
    p_s->draw();

    return 0;
}
```

Whether there is a problem?

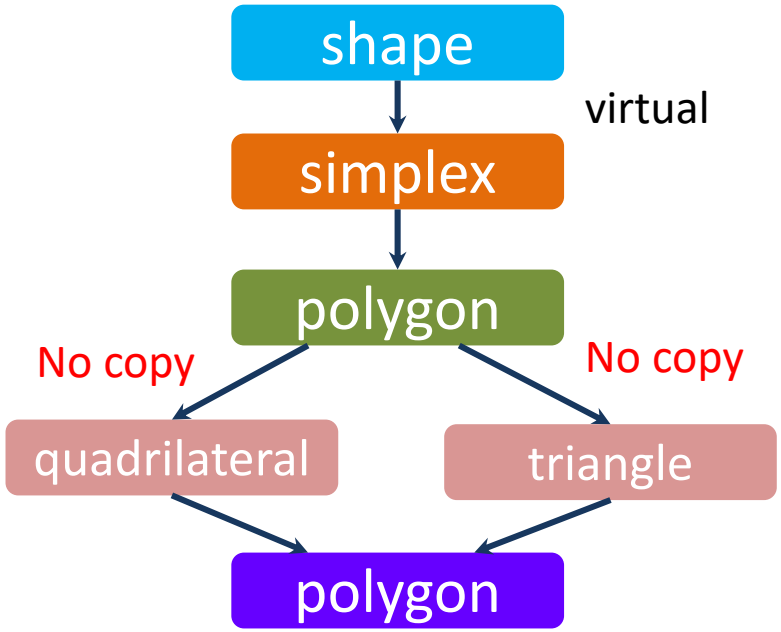
Multiple Inheritance

- **A class can be inherited from multiple parent classes**
  - Yes, the problem is that different child classes share the same parent class



Multiple Inheritance

- **A class can be inherited from multiple parent classes**
  - Solution: virtual inheritance




```
class simplex2D :
        public virtual shape2D
{
public:
    void draw();

    int get_vertex_count();
    int get_edge_count();

    simplex2D();
protected:
    int m_vertex_count;
    int m_edge_count;
};
```

Multiple Inheritance

- Look at our previous code with virtual function polymorphism again

```
int main(){
    vertex2D v[5];
    v[0] = vertex2D(0, 10);
    v[1] = vertex2D(-6, 5);
    v[2] = vertex2D(-6, -1);
    v[3] = vertex2D(6, -1);
    v[4] = vertex2D(6, 5);

    house2D h(v);
    shape2D* p_s = (shape2D*)&h;
    p_s->draw();

    return 0;
}
```

```
--->
You have drawn from house.
<---
```