# CS256

## Chapter 15: Inheritance, Polymorphism, and Virtual Functions

Hanoi University of Science and Technology

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Eighth Edition, Pearson, 2015
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

# Early Binding

```cpp
#include <iostream>
using namespace std;

// Base class
class BaseClass {
public:
    // Normal method, NO virtual keyword
    void display() {
        cout << "This .. method of Base class." << endl;
    }
};

// Derived class inheriting from Base class
class DerivedClass : public BaseClass {
public:
    // Derived class redefines the display() method
    void display() {
        cout << "This ... method of Derived class." << endl;
    }
};
```

# Early Binding

```
int main() {
    DerivedClass derivedObj;

    // Base class pointer pointing to
    // the derived class object
    BaseClass* basePtr = &derivedObj;

    // Early Binding: The compiler calls
    // the method of BaseClass
    // based on the pointer's type,
    // not the actual object it points to.
    basePtr->display();

    return 0;
}
```

# Early Binding

**Static Typing (Early Binding)**

- A base class pointer can hold the address of any derived class object.
- **Limitation:** Method calls are resolved based on the *pointer's type*, not the actual object it points to.

# Early Binding

**Static Typing (Early Binding)**

- A base class pointer can hold the address of any derived class object.
- **Limitation:** Method calls are resolved based on the *pointer's type*, not the actual object it points to.

**The Need for Dynamic Behavior**

- To execute the correct method, the program must identify the actual object's type at runtime.
- A single pointer might point to different object types dynamically during execution.

# Polymorphism and Virtual Functions

**Polymorphism**: allows an object reference variable or an object pointer to reference objects of different types and to call the correct member functions, depending upon the type of object being referenced.

**Virtual Functions in C++**

- C++ introduces **virtual functions** to implement late binding.
- They allow derived classes to override methods and ensure the correct function is called dynamically.

```
virtual void Y() {...}
```

# Virtual Functions

```cpp
#include <iostream>
using namespace std;

// Base class
class BaseClass {
public:
    // virtual keyword
    virtual void display() {
        cout << "This .. method of Base class." << endl;
    }
};

// Derived class inheriting from Base class
class DerivedClass : public BaseClass {
public:
    // Derived class redefines the display() method
    void display() {
        cout << "This ... method of Derived class." << endl;
    }
};
```

# Practical Examples

Program 15.10, 15.11: Virtual function

- `GradedActivity.h`
- `GradedActivity.cpp`
- `PassFailActivity.h`
- `PassFailActivity.cpp`

# Practical Examples

Program 15.12: Polymorphism

- `GradedActivity.h`
- `GradedActivity.cpp`
- `PassFailActivity.h`
- `PassFailActivity.cpp`
- `PassFailExam.h`
- `PassFailExam.cpp`

# Practical Examples

```
#include <iostream>
#include <iomanip>
#include "PassFailExam.h"
using namespace std;

// Function prototype
void displayGrade(const GradedActivity &);

int main()
{
    // Create a GradedActivity object. The score is 88.
    GradedActivity test1(88.0);

    // Create a PassFailExam object. There are 100 questions,
    // the student missed 25 of them, and the minimum passing
    // score is 70.
    PassFailExam test2(100, 25, 70.0);

}
```

# Practical Examples

```
    // Display the grade data for both objects.
    cout << "Test 1:\n";
    displayGrade(test1);       // GradedActivity object
    cout << "\nTest 2:\n";

    displayGrade(test2);     // PassFailExam object

    return 0;
}
```

# Practical Examples

Program 15.13: Pointer + Polymorphism

- `GradedActivity.h`
- `GradedActivity.cpp`
- `PassFailActivity.h`
- `PassFailActivity.cpp`
- `PassFailExam.h`
- `PassFailExam.cpp`

# Practical Examples

```
#include <iostream>
#include <iomanip>
#include "PassFailExam.h"
using namespace std;

// Function prototype
void displayGrade(const GradedActivity *);

int main()
{
    // Create a GradedActivity object. The score is 88.
    GradedActivity test1(88.0);

    // Create a PassFailExam object. There are 100 questions,
    // the student missed 25 of them, and the minimum passing
    // score is 70.
    PassFailExam test2(100, 25, 70.0);

    // Display the grade data for both objects.
    cout << "Test 1:\n";
```

# Practical Examples

```
    displayGrade(&test1);       // Address of the GradedActivity
    cout << "\nTest 2:\n";
    displayGrade(&test2);       // Address of the PassFailExam
    return 0;
}


void displayGrade(const GradedActivity *activity)
{
    cout << setprecision(1) << fixed;
    cout << "The activity's numeric score is "
         << activity->getScore() << endl;
    cout << "The activity's letter grade is "
         << activity->getLetterGrade() << endl;
}
```

# Multiple Inheritance

- A derived class can have more than one base class
- Each base class can have its own access specification in derived class's definition