COMP 356
Programming Language Structures
Notes for class 7 (and for Chapter 11 of *Concepts of Programming Languages*)
Multiple inheritance and polymorphism

1 Polymorphism

In the theory of programming languages, *polymorphism* refers to several different, related ideas. The fundamental concept is that a subprogram is *polymorphic* if it can be executed with different data types as parameters. The different kinds of polymorphism include:

- *ad hoc polymorphism*: This occurs when multiple versions of a subprogram are written by the programmer, with each different version having a different signature (i.e. the number and/or data type of the parameters). This is usually called *overloading* the subprogram.
- *parametric polymorphism*: This occurs when a subprogram has a type parameter, as with generics in Java or templates in C++.
- *subtype polymorphism*: This occurs when a subclass *overrides* a method in its superclass, using the same signature—if the signature were different, the method would be overloaded, not overridden.

(See Section 9.9 of the textbook for additional discussion.)

In the context of object-oriented programming, the word "polymorphism" almost always refers to the subtype polymorphism defined above. From this point on, we discuss only subtype polymorphism. In Java, methods are polymorphic by default—only `static` and `final` methods are not polymorphic. In C++, methods are *not* polymorphic by default. In C++, polymorphic methods are called *virtual functions*, and they must be declared `virtual` in the superclass. See the source file `polymorphism.cpp` for an example. Experiment with removing the `virtual` keyword from one or both of the places where it occurs in this example. Note that the destructors can also be declared virtual, and often should be.

The key point about subtype polymorphism is that the program must determine which version of an overridden method to call at *runtime*. When a polymorphic method is called on a pointer or reference to an object, it is the actual type of the object itself (and not the type of the pointer or reference) which is used to determine which method to call. For example, in C++, suppose `print()` is a virtual method of the `Animal` class, and it is overridden in the `Tiger` subclass. Then a variable `animal` of type `Animal*` might actually point to an object of type `Tiger`. When `animal->print()` is invoked, the runtime system must somehow determine the real type of `*animal`. If it is actually `Tiger`, then `Tiger::print()` will be invoked.

1.1 Rules of thumb for polymorphism in C++

There is a slight performance penalty for using virtual functions, although the extent of this cost is debatable. If you search online, you will see many debates about when functions and destructors should be declared virtual. Here are some simple rules of thumb that will work most of the time:

- Any member function that might be overridden by a subclass should be virtual.

- Any class that might have a subclass should have a virtual destructor.

Failing to declare functions or destructors virtual is a common source of programming errors.

1.2 Implementation of subtype polymorphism by vtables

Polymorphism is usually implemented using a *vtable*, also called a *virtual method table* (Java) or *virtual function table* (C++). You need to understand the use of vtables for single inheritance (pages 566–567 of the textbook, Section 12.11.2, especially figure 12.7). You do not need to understand vtables for multiple inheritance (the example on page 568, and figure 12.8).

2 Multiple inheritance

Recall that in Java, an object can implement many interfaces, but can have only one superclass. In C++, an object can have more than one superclass; this feature of the language is known as *multiple inheritance*. See the file `multiple-inheritance.cpp` for an example. Multiple inheritance has advantages (power and flexibility) and disadvantages (complex to use and maintain). Many people believe the disadvantages outweigh the advantages; that's why some modern languages, like Java, don't permit multiple inheritance.

In this course, you aren't required to use multiple inheritance, but you need to be aware that it is a somewhat controversial feature of C++.

3 Abstract classes and Java-style interfaces in C++

Recall that in Java, a class can be declared `abstract`, meaning that it can't be instantiated, but it can be extended. Individual methods can also be declared `abstract`, meaning that no implementation is provided for the method, and subclasses must provide an implementation.

In C++, abstract methods are called *pure virtual functions*. You can specify that a virtual function is pure by adding `= 0` after its declaration (admittedly, this syntax is obscure, but you'll get used to it). For example,

```
virtual int getNumWombats() = 0;
```

declares a pure virtual function.

In C++, there is no special syntax for declaring an abstract class. A class is abstract if and only if it contains at least one pure virtual function.

Java-style interfaces are not built into C++, but they can be simulated using multiple inheritance of abstract classes that have no implemented methods. See the file `java-style-interface.cpp` for an example.