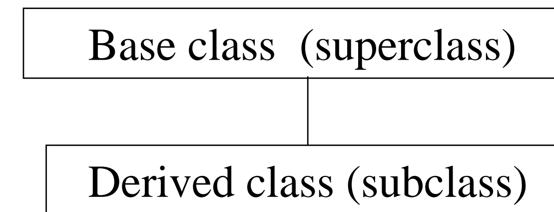
OOD and C++

Section 4: Inheritance

Class Hierarchies

Class: Object that encapsulates data and associated functions

Class Hierarchy: Build a set of classes which share common data and functionality



Inheritance: Derived class 'inherits' properties from its base class

Derived class has access to **public** and **protected** member functions and data of the base class

private data and functions are encapsulated in the base class

C++ Implementation of Inheritance

```
class BaseClass {
        public:
            fbase_public();
        protected:
            fbase_protected();
        private:
            fbase_private();
};
```

```
class DerivedClass: public BaseClass {
        public:

        protected:

        private:
};
```

Base class is inherited as public, protected or private

Derived class access public and protected member functions & data of base class as if they were member of DerivedClass:

```
DerivedClass.fbase_public();
DerivedClass.fbase_protected();
```

Constructors of BaseClass are never inherited

Example: FileNameC

StringC (base)

FileNameC (derived)

StringC declaration

```
class StringC {
     private:
            char *ps;
     public:
            StringC();        // default constructor
            StringC(char *string);
            int Length();
            void print();
};
```

FileNameC declaration

```
class FileNameC: public StringC {
     public:
            FileNameC(char *);
            bool IsValid();
            void print();
};
```

User program

```
void myprogram(){
     FileNameC  fn(“myfile.dat”);
     fn.Length();    // member function from StringC
     fn.print();      // member function from FileNameC
}
```

C++ Inheritance Rules

(1) Constructors, assignment and destructors
are never inherited

(2) If base class has a default constructor it is automatically
called.

(3) Derived class inherits public & protected members
access as:

```
derivedclass.basefunction();
derivedclass.basedata;
```

(4) If derived class member function of same name overrides
the base class function.
access base class function as:

```
baseclass::basefunction();
```

Inheritance of Constructors/Destructors

Constructors are never inherited

•If the base class has a default constructor its invoked implicitly on constructing derived class

•If all base class constructors require some arguments then derived class must explicitly call base class constructors

Constructor for DerivedClass

```
DerivedClass::DerivedClass(arg1, arg2,arg3)
  : baseclass(arg2),              // call base class constructor
    data1(arg1), data2(arg3)      // member data for derived
{
        // other initialisation
}
```

class objects are constructed from the bottom up

- base class
- members
- derived class

destroyed from top down (opposite order)

StringC Constructor

filename.hh

```
#include "string.hh"

class FileNameC : public StringC {
    private:
            int filetype;
    public:
            FileNameC();            // default constructor
            FileNameC(char *c, int t); // constructor
};
```

filename.cc

```
#include "filename.hh"

FileNameC::FileNameC()
: StringC()          // call default constructor for string
{}

FileNameC::FileNameC(char *c, int t)
: StringC(c),          // constructor for string from char *
  filetype(t)          // member data
{}
```

Copy Constructors/Assignment - Revisited

```
Copy member data for class
        myclass b;
        myclass a(b);  // copy constructor
        f(a);          // copy constructor
        a = b;         // assignment
        a = f();       // assignment
```

myclass.hh

```
class myclass {
private:
        int mydata;
  public:
        myclass();
        myclass(const myclass& c);          // copy constructor
        myclass& operator=(const myclass& c); // assignment
};
```

myclass.cc

```
myclass::myclass(const myclass& c)
: mydata(c.mydata)      // copy member data
{}

myclass& myclass::operator=(const myclass& c)
{
        mydata = c.mydata;
        return *this;
}
```

Copy Constructor & Assignment with Inheritance

Constructors/Assignment not inherited from base class

If derived class does not define copy/assignment compiler will generate one - bit wise copy.

filename.hh

```
class FileNameC: public StringC {
    private:
            int filetype;
    public:
            FileNameC(const FileNameC& fn);
            FileNameC& operator=(const FileNameC& fn);
};
```

filename.cc

```
FileNameC::FileNameC(const FileNameC& fn)
   : StringC(fn),          // construct from StringC copy  - cast fn
     filetype(fn.filetype)
{}

FileNameC& FileNameC::operator=(const FileNameC& fn)
{
      StringC::operator=(fn);   // StringC assignment -explicit
      filetype = fn.filetype;
      return *this;
}
```

Member Functions

Derived class inherits all public & protected member functions

Access member functions as if they are member of derived class

derivedclass.basefunction();

Member functions of the **same name** in the derived class override the base member function

Access overridden member functions explicitly

baseclass::basefunction();

Example - Member Function Inheritance

```
class StringC {
        public:
                void print();
};

class FileNameC: public StringC {
        public:
              void print();
};
```

filename.cc

```
void FileNameC::print() {
            cout << “file type:” << filetype << “\n”;
            StringC::print();      // explicit call overridden member
};
```

Casting

Slicing: copying a derived class to a base class copies only the member data for base.

```
FileNameC f;
StringC  s=f;    // copy base data/member funtion
s.print();         // StringC::print();
```

Derived class can be assigned to a base class without explicit casting.

Reference & Pointers: to avoid slicing pass references and pointers to objects of a class hierarchy & base member must be virtual

```
FileNameC f;
StringC& rs = f;      // reference to derived class
StringC* ps = &f;   // pointer to derived class
rs.print();               // FileNameC::print()
ps->print();            //  FileNameC::print()
```

Polymorphic behaviour: correct behaviour of base class independent of the derived class

In C++: (1) Objects must be passed by reference or pointer
(2) Member functions must be virtual

Virtual Member Functions

Allow declaration of a function in a base class which can be redefined in each derived class

```
class baseclass {
        virtual void function();
};
```

Specifies a user interface

Enable polymorphic behaviour of base object independent of derived class

Derived class can optionally override virtual function
- must specify virtual function with same name and arguments

Compiler stores a pointer in the base class to the definition of the virtual function in either the base or derived class
- ensures correct polymorphic behaviour of base class
- access to the virtual function is a single pointer dereference (as for other functions).
- base functions can be accessed directly: baseclass::function();

Constructors/Assignment are never virtual

Example - Virtual Member Functions

```
class StringC {
    public:
        virtual void print();
};

class FileNameC: public StringC {
    public:
        void print();
};

class TownNameC: public StringC {
    public:
        void print();
};
```

```
void myprogram(){
        FileNameC myfile("file.tex");
        TownNameC mytown("Brighton");
        StringC& rf(myfile);
        StringC& rt(mytown);

        rf.print();    // FileNameC::print()
        rt.print();    // TownNameC::print()
}
```

Pure Virtual Member Functions

No definition of one or more virtual function in class
=> **abstract class**

- No object of an abstract class can be created
- virtual destructor should be defined to ensure proper cleanup

INTERFACE DEFINITION

- Define an abstract representation of an object which is used to derive specific instances
- forces standard interface

```
class abstractbaseclass {
        public:
            virtual void function() = 0; // pure virtual function
            virtual ~abstractbaseclass(); // virtual destructor
};
```

Derived class must implement pure virtual functions

```
class derivedclass: abstractbaseclass {
        public:
            void function();
            ~derivedclass();
};
```

Example - Abstract Class

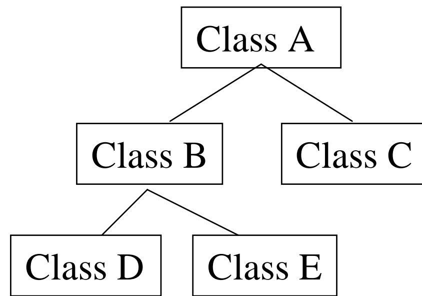
```
class ShapeC {
    public:
            virtual void rotate(double a) =0;
            virtual void draw() =0;
            virtual ~ShapeC;
};
```

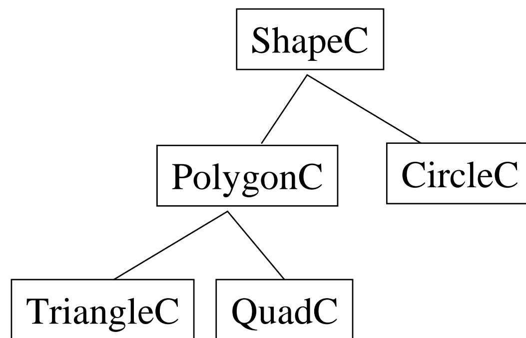
```
class Circle: public ShapeC {
    public:
            Circle(Point p, double r);
             ~Circle();
            void rotate(double a);
            void draw();
    private:
            Point p;
            double r;
};
```

```
class TriangleC: public ShapeC {
    public:
            Triangle(Point p1, Point p2, Point p3);
            ~Triangle();
            void  rotate(double a);
            void  draw();
};
```

Class Hierarchy

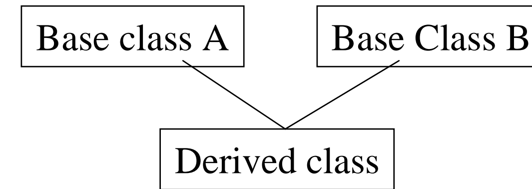
Tree of classes (each parent-child is a base and derived class)





Multiple Inheritance

Derived class inherits from more than one base class



Graph class hierarchy

```
class derivedclass: public baseclassA, public baseclassB {

};
```

Derived class inherits member data/functions from both base classes independently if base class is not virtual

If multiple base member functions have the same name resolved by explicit call or deriving member function of the same name.

```
baseclassA::function();
baseclassB::function();
```

Virtual base classes of the same name replaced by a single object.

Designing Class Hierarchy

Design so that user does not need to know about the details of the hierarchy

Design hierarchy to:

(1) define interfaces for common functions (abstract classes/virtual functions)
(2) Encapsulate common data
(3) Encapsulate common member functions

Example - Class Hierarchy for VideoPlayer