CS 101: Computer Programming and Utilization

20-Inheritance

Instructor: Sridhar Iyer
IIT Bombay
Avoid redundancy in these class definitions

```cpp
class Rectangle{
    private:
        int numVertices;
        float *xCoord, *yCoord;
    public:
        void set(float *x, float *y, int nV);
        float area();
    }

class Polygon{
    private:
        int numVertices;
        float *xCoord, *yCoord;
    public:
        void set(float *x, float *y, int nV);
    }

class Triangle{
    private:
        int numVertices;
        float *xCoord, *yCoord;
    public:
        void set(float *x, float *y, int nV);
        float area();
    }
```
Inheritance Concept

class Polygon{
    protected:
        int numVertices;
        float *xCoord, float *yCoord;
    public:
        void set(float *x, float *y, int nV);
};

class Rectangle : public Polygon{
    public:
        float area();
};

class Rectangle{
    protected:
        int numVertices;
        float *xCoord, float *yCoord;
    public:
        void set(float *x, float *y, int nV);
        float area();
};
Inheritance Concept

```cpp
class Polygon{
    protected:
        int numVertices;
        float *xCoord, float *yCoord;
    public:
        void set(float *x, float *y, int nV);
    }

class Triangle : public Polygon{
    public:
        float area();
    }

class Triangle{
    protected:
        int numVertices;
        float *xCoord, float *yCoord;
    public:
        void set(float *x, float *y, int nV);
        float area();
    }
```
Inheritance: Another Example

```cpp
class Point{
    protected:
    int x, y;
    public:
    void set (int a, int b);
};

class Circle : public Point{
    private:
    double r;
};

class 3D-Point: public Point{
    private:
    int z;
};
```
Why Inheritance?

Inheritance is a mechanism for building class types from existing class types

- A way to establish **Is-a** relationship between objects
  - Polygon – base class.
  - Rectangle – Derived class.

- A way to reuse the existing code of base class.

http://www.cplusplus.com/doc/tutorial/inheritance/
Point is the base class of 3D-Point; 3D-Point is the base class of Sphere
Derived Class: Members

The derived class can define its own members, in addition to the members inherited from the base class.

```cpp
class Point{
    protected:
        int x, y;
    public:
        void set(int a, int b);
};

class Circle : public Point{
    private:
        double r;
    public:
        void set_r(double c);
    void set(int a, int b);
};
```
Derived Class: Function Overriding

- A derived class can override methods defined in its parent class. With overriding,
  - the method in the subclass has the identical signature to the method in the base class.
  - a subclass implements its own version of a base class method.

```cpp
class A {
   protected:
      int x, y;
   public:
      void print ()
         {cout<<“From A”<<endl;}
};

class B : public A {
   public:
      void print ()
         {cout<<“From B”<<endl;}
};
```
class Point{
    protected:
        int x, y;
    public:
        void set(int a, int b)
        {
            x=a; y=b;
        }
        void foo();
        void print();
};

class Circle : public Point{
    private: double r;
    public:
        void set (int a, int b, double c) {
            Point :: set(a, b); //same name function call
            r = c;
        }
        void print();
};

Circle C;
C.set(10,10,100);    // from class Circle
C.foo ();            // from base class Point
C.print();           // from class Circle

Point A;
A.set(30,50);        // from base class Point
A.print();           // from base class Point
Activity: Inheritance and Multiple files

- Code walk-through – demo19-point.h
- Build and run – demo19-point.cpp

- Code walk-through – demo19-circle.h
- Build and run – demo19-circle.cpp

- Code walk-through – demo19-cylinder.h
- Build and run - demo19-cylinder.cpp
Notes: Inheritance – Purpose (Optional Reading)

- Augmenting the original class

  - Polygon
    - Rectangle
    - Triangle

  - Point
    - Circle
    - 3D-Point

- Specializing the original class

  - ComplexNumber
    - RealNumber
    - ImaginaryNumber
      - real
      - imag
    - real
    - imag
• Time is the base class
• ExtTime is the derived class with public inheritance that has the notion of timezones.

• The derived class can
  • inherit all members from the base class, except the constructor
  • access all public and protected members of the base class
  • define its private data member
  • provide its own constructor
  • define its public member functions
  • override functions inherited from the base class
class Time Specification

// SPECIFICATION FILE (time.h)

class Time{

public:
    void Set (int h, int m, int s);
    void Increment()
    void Write() const;

    Time (int initH, int initM, int initS); // constructor
    Time(); // default constructor

protected:
    int hrs;
    int mins;
    int secs;

};
Class Interface Diagram

Time class

Protected data:
hrs
mins
secs

Set
Increment
Write
Time
Time
Derived Class ExtTime

// SPECIFICATION FILE ( exttime.h)

#include "time.h"

enum ZoneType {EST, CST, MST, PST, EDT, CDT, MDT, PDT} ;

class ExtTime : public Time
    // Time is the base class and use public inheritance
{
    public :
        void Set ( int h, int m, int s, ZoneType timeZone ) ;
        void Write ( ) const;    // overridden
        ExtTime (int initH, int initM, int initS, ZoneType initZone ) ;
        ExtTime ();    // default constructor

    private :
        ZoneType zone ;    // added data member

} ;
Class Interface Diagram

ExtTime class

Protected data:
- hrs
- mins
- secs

Private data:
- zone

Methods:
- Set
- Increment
- Write
- ExtTime
Implementation of \texttt{ExtTime}

Default Constructor

\begin{verbatim}
ExtTime :: ExtTime ( )
{
    zone = EST ;
}
\end{verbatim}

The default constructor of base class, \texttt{Time()}, is automatically called, when an \texttt{ExtTime} object is created.
Implementation of ExtTime

Another Constructor

ExtTime :: ExtTime (int initH, int initM, int initS, ZoneType initZone) 
  : Time (initH, initM, initS) 
  // constructor initializer 
{
    zone = initZone ; 
}

ExtTime *et2 = 
  new ExtTime(8,30,0,EST);

hrs = 8
mins = 30
secs = 0
zone = EST
Implementation of **ExtTime**

```cpp
void ExtTime :: Set (int h, int m, int s, ZoneType timeZone)
{
    Time :: Set (hours, minutes, seconds);  // same name function call
    zone = timeZone;
}
```

```cpp
void ExtTime :: Write ( ) const  // function overriding
{
    string zoneString[8] =
        {“EST”, “CST”, “MST”, “PST”, “EDT”, “CDT”, “MDT”, “PDT”};

    Time :: Write ( );
    cout << ‘ ‘ << zoneString[zone] << endl;
}
```
# Working with ExtTime

```c
#include "exttime.h"
... ...
int main()
{
    ExtTime thisTime ( 8, 35, 0, PST ) ;
    ExtTime thatTime ;                      // default constructor called
    thatTime.Write( ) ;                    // outputs 00:00:00 EST
    thatTime.Set (16, 49, 23, CDT) ;
    thatTime.Write( ) ;                    // outputs 16:49:23 CDT
    thisTime.Increment ( ) ;
    thisTime.Increment ( ) ;
    thisTime.Write ( ) ;                    // outputs 08:35:02 PST
}
```
Constructor rules for Derived Classes

The default constructor and the destructor of the base class are always called when a new object of a derived class is created or destroyed.

class A {
public:
    A( )
    {cout<< "A:default"<<endl;}
    A(int a)
    {cout<<"A:parameter"<<endl;}
};

class B : public A {
    public:
        B(int a)
        {cout<<"B"<<endl;}
    
};

B test(1);

output:

A:default

B
Constructor rules for Derived Classes

You can also specify a constructor of the base class other than the default constructor

```cpp
class A {
public:
    A () {
        cout << "A:default" << endl;
    }
    A (int a) {
        cout << "A:parameter" << endl;
    }
};
```

```cpp
class C : public A {
public:
    C (int a) : A(a) {
        cout << "C" << endl;
    }
};
```

class C : public A {
    public:
        C (int a) : A(a) {
            cout << "C" << endl;
        }
};
```

```cpp
C test(1);
```

output:

```
A:parameter
C
```