CEE432/CEE532/MAE541
Developing Software for Engineering Applications

Lecture 13: Objects 303
(Chapter 13)

Case Study: Steel Beam Selection Program

- You are required to design and operate a “AISC Steel Beam Cross-Section Selection” program. The beam cross-section has a symmetric I-section. The user of the system enters (a) the largest moment, (b) the largest tensile force, and (c) the largest compressive force, the beam is subjected to. The system finds the lightest cross-section from the available cross-sections database that will meet the strength (stress) requirements.

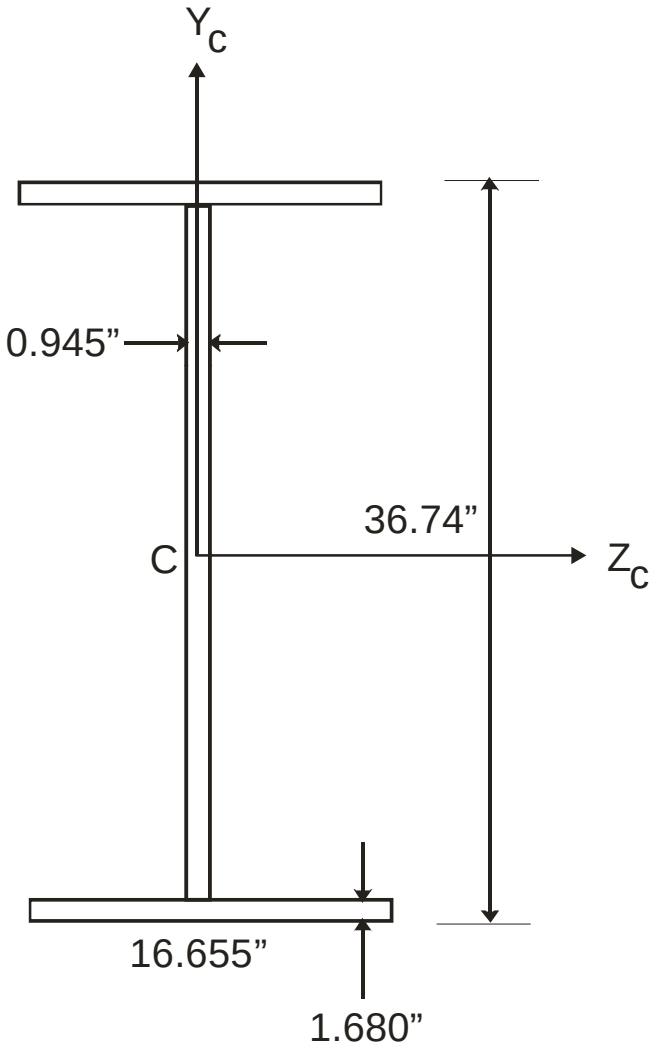
Steel Beam Selection Program

Selection Criteria

$$\sigma^{c}_{\max} = \frac{|N_c|}{A} + \frac{|M|}{S} \leq 20000\ psi$$

$$\sigma^{t}_{\max} = \frac{|N_t|}{A} + \frac{|M|}{S} \leq 20000\ psi$$

$$S = \frac{I}{y_{\max}}$$



W36 x 360

Analysis of the Problem Statement

Noun or noun clauses		
Beam	I-section	User Input
Largest moment	Tensile Force	Compressive Force
Lightest cross-section	X/S database	

Analysis of the Problem Statement

- I-section is a beam cross-section
- Cross-sections are stored in the X/S database
- User inputs the largest moment, tensile and compressive forces
- Need to locate the lightest section that meets the design criteria

Class Organization

- CISection
  - Stores I-section related data
- CXSDatabase
  - Database of I-section cross-sections
- CXSSelector
  - Used to select the lightest I-section that meets the selection criteria

Class Organization

Class: CISection

Responsibilities:

know properties

allow access to properties

Helpers:

Example 13.1.1
Testing the *CISection* class

CISection

```
class CISection
{
    public:
        CISection ();
        CISection (const std::string&, float, float,
                   float, float);
        ~CISection ();
        // helper functions
        void Display () const;
        // accessor functions
        void Get (std::string&, float&, float&, float&, float&) const;
        // modifier functions
        void Set (const std::string&, const float, const float,
                  const float, const float);
        void Set (const CISection&);

    private:
        std::string m_szID;   // identification tag
        float       m_fArea;  // x/s area
        float       m_fSyy;   // section modulus y-axis
        float       m_fSzz;   // section modulus z-axis
        float       m_fWeight;// weight per unit length
};
```

Class Organization

Class: CXSDatabase

Responsibilities:
know all I-sections
allow access to individual I-section
add new sections
remove existing sections

Helpers:
CISection

Example 13.1.2
Testing the *CXSDatabase* class

CXSDatabase

```
class CXSDatabase
{
    public:
        CXSDatabase ();
        ~CXSDatabase ();
        // helper functions
        CISection GetOne (int) const;
        // accessor functions
        int GetSize () const;
        // modifier functions
        void Add (const CISection& ISection);
        int  Remove (const CISection& ISection);

    private:
        std::vector<CISection> m_ListofISections;
                               // list of available sections
        int m_nSize;           // # of sections
        std::ifstream m_IFile; // (file) source of database
};
```

Class Organization

Class: CXSSelector

Responsibilities:
get an I-section

Helpers:
CISection
CXSDatabase

Class Organization

Class: CWizard

Responsibilities:
know user input
process user input
display the results

Helpers:
CISection
CXSDatabase
CXSSelector

Example 13.1.3
Testing the *CWizard* and *CXSSelector* classes

CXSSelector

```
class CXSSelector
{
    public:
        CXSSelector ();
        ~CXSSelector ();
        // accessor functions
        int GetXSection (const CXSDatabase&,
                         CISection&, float, float, float);
};
```

CWizard

```
class CWizard
{
        public:
                CWizard ();
                ~CWizard ();
                // helper functions
                int GetUserInput ();
                void ProcessUserInput ();
                void DisplayResults ();

        private:
                CXSDatabase m_DBISection;
                CISection m_ISection;
                CXSSelector m_SelectLightest;
                float m_fBMMax, m_fTFMax, m_fCFMax;
                int m_nFound;
};
```

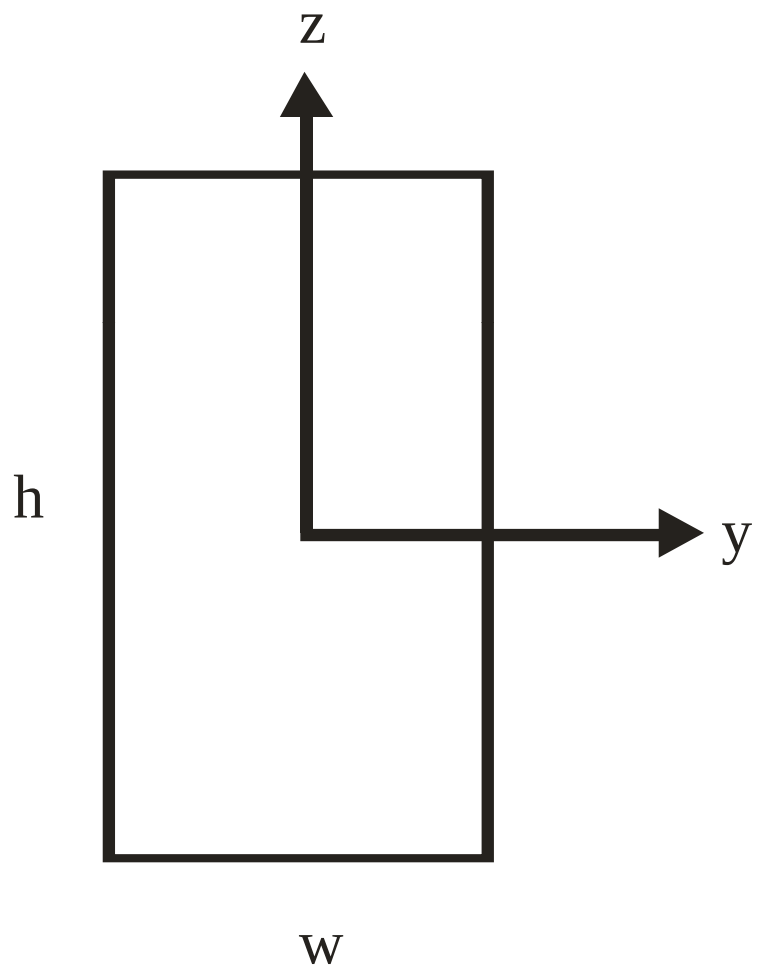
Laying the Foundation for Inheritance

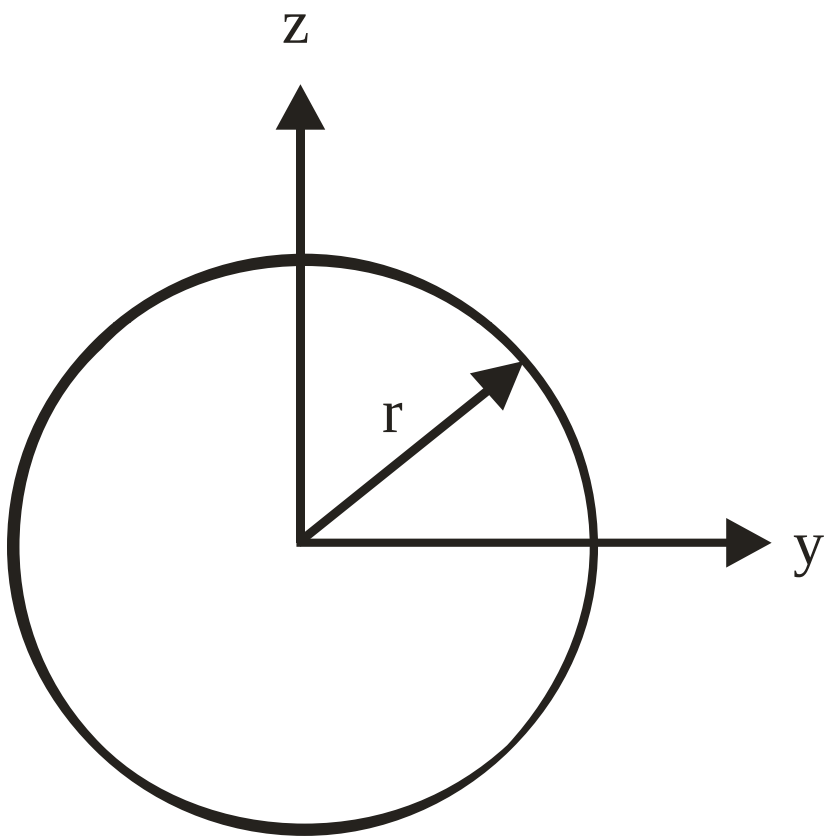
We looked at benefits of **encapsulation** in Chapters 7 and 9.
Now we will look at why **inheritance** is necessary.

Weaknesses of the Previous Solution

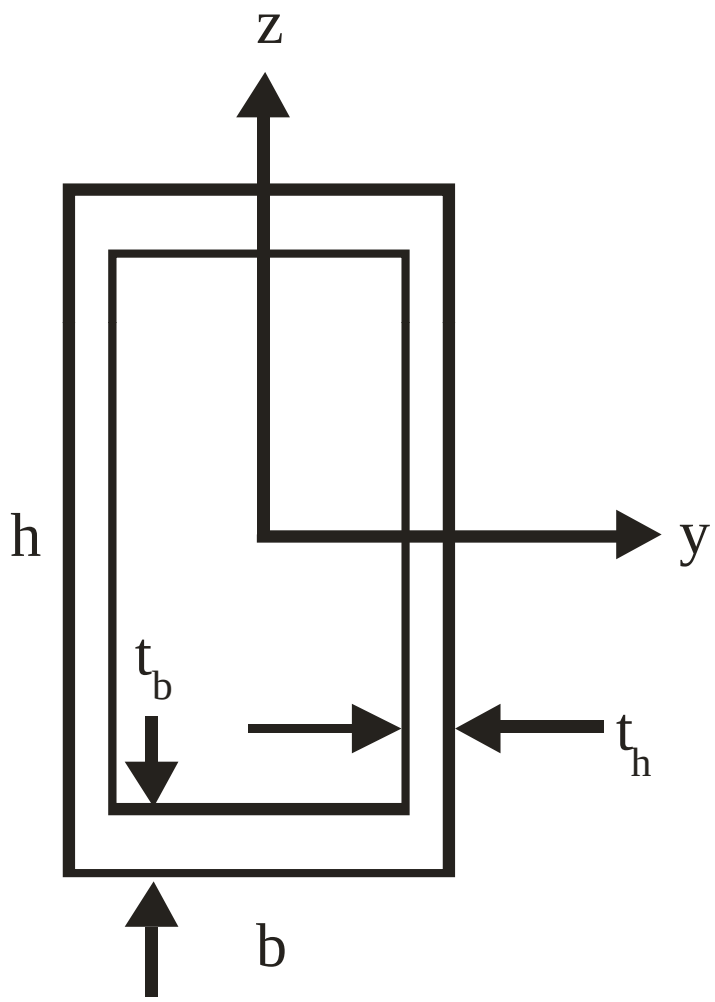
- There exist a variety of cross-section types.
- Cross-sections are described by dimensions rather than x/s properties.
- However, all x/s have common x/s properties – area, moment of inertia etc.

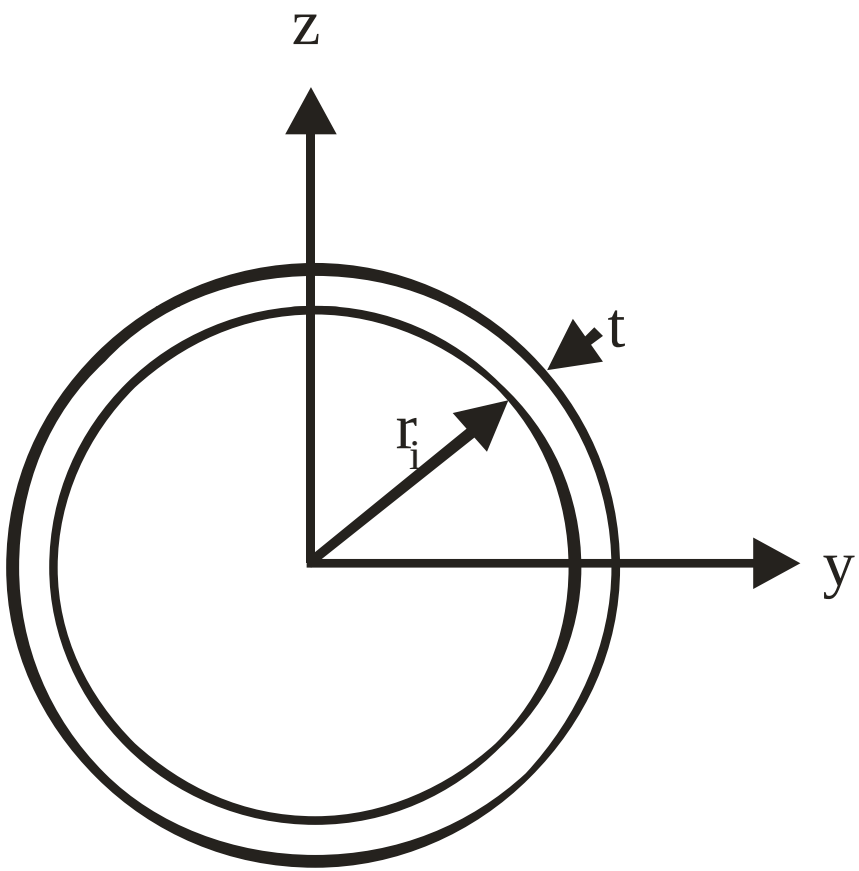
Types of Cross-Section



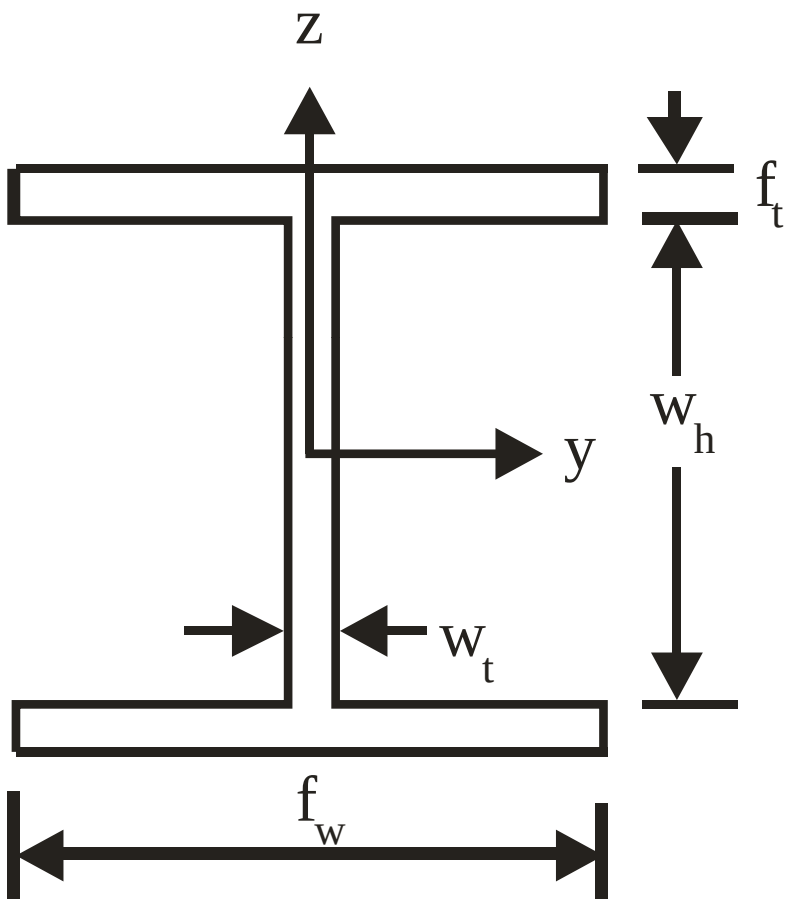


Types of Cross-Section

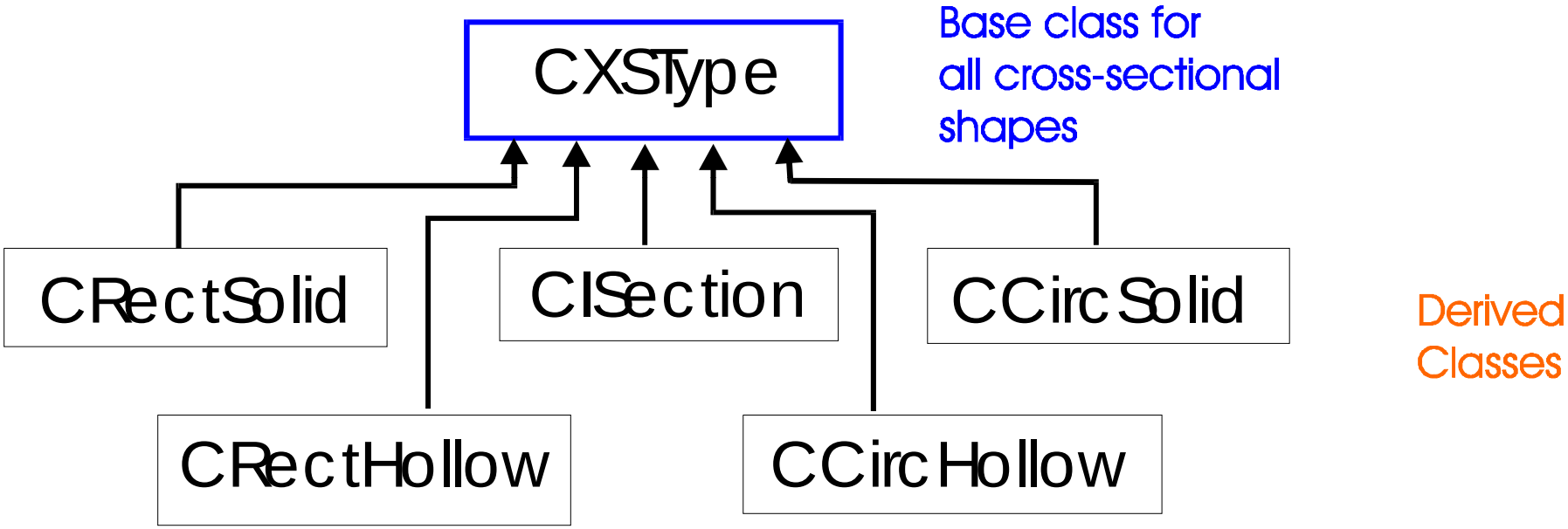




Types of Cross-Section



Inheritance Diagram



What is Base Class?

- Base class
  - This class contains member functions and variables that are “common” or generic to all the derived classes.
- Abstract Base Class
  - A base class that is not associated with any object. It is used merely to help define derived classes.

CXSType: Base Class

```
class CXSType
{
    public:
        CXSType ();
        CXSType (int);
        ~CXSType ();
        // helper function
        void DisplayProperties () const;
       void DisplayDimensions () const;
        // accessor functions
        void GetProperties (std::string&, float&, float&, float&);
        void GetDimensions (std::string&, CVector<float>&);
    private: // derived classes cannot access the following
        void Initialize ();
    protected: // derived classes can access the following
        std::string m_szID;              // identification tag
        float m_fArea;                   // x/s area
        float m_fSyy;                    // section modulus y-axis
        float m_fSzz;                    // section modulus z-axis
        int   m_numDimensions;           // number of dimensions
        CVector<float> m_fVDimensions; // the dimensions
};
```

What is a Derived Class?

- Inherited from a base class
- Has direct access to all `public` and `protected` member functions and variables
- Does **not** inherit specialized member functions such as ctor, dtor and copy ctor and = operator
- Base class member functions can be redefined in the derived class.

CISection: Derived Class

```
#include "xstype.h"
const int numISDimensions = 4;
#include <string>

class CISection: public CXSType
{
    public:
        CISection (const std::string&, const CVector<float>& fV);
        CISection (const CISection&);
        ~CISection ();

        // helper functions
        void DisplayDimensions () const;

    private:
        void ComputeProperties ();
};
```

What is a Derived Class?

- When a derived class is instantiated, one should explicitly instantiate the base class. Otherwise the default base class ctor is invoked.
- If the default base class ctor is not provided, compilation is not possible.

Derived Class

```
CISection::CISection (const std::string& szID,
                const CVector<float>& fV) : CXSType (numISDimensions)
{
    assert (fV.GetSize() >= numISDimensions);
    m_szID = szID;
    for (int i=1; i <= numISDimensions; i++)
        m_fVDimensions(i) = fV(i);
    ComputeProperties ();
}
```

Note the Following

- The base class overloaded constructor is called first followed by the derived class constructor.
- One should remember that if class B is derived from class A, then instantiating a class B object would involve class A constructor being invoked first followed by class B constructor. In the same vein, if object B goes out of scope, the destructor for class B is called first followed by the destructor for (base) class A.

More on Inheritance

- `Private` and `protected` inheritances are possible. Not very common.
- Multiple inheritances

```
classname::classname (…) : base_class1, base_class2
```

Example 13.2.1
Testing the *CISection* and *CXSType* classes

Summary

- Inheritance should be used with care especially with regards to using `protected` variables and functions.
- If used wisely, inheritance can make adding program capabilities much easier to implement.
- Remember the rules governing base class and derived classes.