[Example: 30] Basic Class Concept

CPP File {Example30.cpp}

```
//    Fraction class program
//    Create and use a simple fraction class

#include <iostream.h>
#include "FractionClass.h"

//    Prototype Declarations
void getData (int& numer, int& denom);

int main (void)
{
//    Local Declarations
      Fraction fr;

      int numer;
      int denom;

//    Statements
      cout << "This program creates a fraction\n\n";

      getData  (numer, denom);
      fr.store (numer, denom);

      cout << "\nYour fraction contains: ";
      fr.print ();

      cout << "\n\nThank you for using fractions\n";

      return 0 ;
} // main

/*    ================ getData ================
      Reads the numerator and denominator from the keyboard.
          Pre   numer and denom contain the numerator and
                denominator respectively.
          Post  data stored
*/
void getData (int& numer, int& denom)
{
//    Statements
      cout << "Please enter the numerator:   ";
      cin  >>  numer;

      cout << "Please enter the denominator: ";
      cin  >>  denom;

      return;
}     // getData

/*
Results
      This program creates a fraction

      Please enter the numerator:   19
      Please enter the denominator: 51

      Your fraction contains: 19/51

      Thank you for using fractions
*/
```

Header File {FractionClass.h}

```
//    Fraction class header file

//    Implemented in FractionClass.h
//    Header file for simple fraction class

//    Class Declarations
class Fraction
{
     private:
          int numerator;
          int denominator;
     public:
          void store     (int numer, int denom);
          void print     (void);
               Fraction  (void);     //Default constructor
};  // Fraction

/*   =============  Fraction :: Fraction ===============
     Default constructor for Fraction class
     It creates a valid zero valued fraction.
          Pre   Nothing
          Post  fraction object initialized
*/
Fraction :: Fraction (void)
{
//    Statements
     numerator   = 0;
     denominator = 1;

}    // Default constructor

/*   ================ Fraction :: store ===============
     Store the numerator and denominator in the fraction
     object.
          Pre   numer and denom contain the numerator and
                denominator respectively.
          Post  data stored
*/
void Fraction :: store (int numer, int denom)
{
//    Statements
     numerator   = numer;
     denominator = denom;

     return;
}    // Fraction store

/*   ================ Fraction :: print ===============
     Prints the numerator and denominator as a fraction
          Pre   fraction object must contain data
          Post  data printed
*/
void Fraction :: print (void)
{
//    Statements
     cout << numerator << "/" << denominator;

     return;
}    // Fraction print
     // ============ End Fraction Functions ============
```

[Example: 31] Constructor

CPP File {Example31.cpp}

```
//    Using initialization constructor
//    Demonstrate use of Fraction initializer constructor

#include <iostream.h>
#include "FractionClass.h"

int main (void)
{
//    Local Declarations
      Fraction fr1;
      Fraction fr2 (5, 8);

//    Statements
      cout << "fr1 contains: ";
      fr1.print ();
      cout << endl;

      cout << "\nfr2 contains: ";
      fr2.print ();
      cout << endl;
      return 0 ;
} // main

/*
Results
      fr1 contains: 0/1

      fr2 contains: 5/8
*/
```

Header File {FractionClass.h}

```
//    Fraction class header file
//    Implemented in FractionClass.h
//    Header file for simple fraction class

//    Class Declarations
class Fraction
{
      private:
          int numerator;
          int denominator;
      public:
          void store (int numer, int denom);
          void print     (void);
               Fraction  (void);                //Default
               Fraction  (int numer, int denom); //Initial
};  // Class Fraction

/*    =============  Fraction :: Fraction ===============
      Default constructor for Fraction class
      It creates a valid zero valued fraction.
          Pre   Nothing
          Post  fraction object initialized
*/
Fraction :: Fraction (void)
{
//    Statements
      numerator   = 0;
      denominator = 1;

}     // Default constructor
```

```
/*   =============  Fraction :: Fraction ===============
     Initialization constructor for Fraction class
     Initializes fraction to values in parameter list.
         Pre   numen and denom contain fraction values
         Post  fraction object initialized  */
Fraction :: Fraction (int numen, int denom)
{
//   Statements
     numerator   = numen;
     denominator = denom;
}    // Initialization constructor

/*   ================= Fraction :: store ===============
     Store the numerator and denominator in the fraction
     object.
         Pre   numer and denom contain the numerator and
               denominator respectively.
         Post  data stored  */
void Fraction :: store (int numer, int denom)
{
//   Statements
     numerator   = numer;
     denominator = denom;

     return;
}    // Fraction store

/*   ================ Fraction :: print ===============
     Prints the numerator and denominator as a fraction
         Pre   fraction object must contain data
         Post  data printed  */
void Fraction :: print (void)
{
//   Statements
     cout << numerator << "/" << denominator;
     return;
}    // Fraction print
     // ============ End Fraction Functions ============
```

[Example: 32] Destructor

CPP File {Example32.cpp}

```
//    Hypothetical Destructor
//    Hypothetical class destructor driver to test concept

//    Class Declarations
class Dynamic
{
      private:
           int *pData1;
           int *pData2;
      public:
           // Default constructor
           Dynamic (void);
           // Destructor
           ~Dynamic (void);
};  // Dynamic

/*    ============= Dynamic :: Dynamic =============
      Dynamic destructor. Releases dynamic memory allocated
      to memory variables.
         Pre     Class object is being destroyed
         Post    Memory released
*/
Dynamic :: Dynamic (void)
{
//    Statements
      pData1 = new (int);
      pData2 = new (int);
} // Dynamic constructor

/*    ============= Dynamic :: ~Dynamic =============
      Dynamic destructor. Releases dynamic memory allocated
      to memory variables.
         Pre     Class object is being destroyed
         Post    Memory released
*/
Dynamic :: ~Dynamic (void)
{
//    Statements
      delete pData1;
      delete pData2;
} // Dynamic destructor

#include <iostream.h>

int main (void)
{
//    Local Declarations
Dynamic item;

//    Statements
      cout << "This program does nothing\n";
      return 0;
}     // main
```

[Example: 33] Inheritance

CPP File {Example 33.cpp}

```
//    A polygon--triangle implementation
//    Demonstrate use of inheritance

#include <iostream.h>
#include <math.h>
#include "Polygons.h"
#include "Triangle.h"

int main (void)
{
//    Local Declarations
      Triangle tri (3, 4, 5);

//    Statements
      cout << "Start Polygon Demonstration\n\n";

      tri.printArea();
      tri.printPeri();

      cout << "\nEnd Polygon Demonstration\n";
      return 0 ;
} // main
// ================= End of Program ===================

/* Results
      Start Polygon Demonstration

      The area of your polygon is 6
      The perimeter of your polygon is 12

      End Polygon Demonstration
*/
```

Header File {Polygons.h}

```
class Polygons
{
      protected:
          double area;
          double perimeter;
      public:
          void printArea (void);
          void printPeri (void);
};    // Class Polygons

/*    ============== Polygons :: printArea =============
      Prints the area of a polygon
          Pre   area calculated & stored in area
          Post  area printed
*/
void Polygons :: printArea (void)
{
//    Statements
      cout << "The area of your polygon is "
           << area << endl;
      return;
}     // Polygons printArea

/*    ============== Polygons :: printPeri =============
      Prints the perimeter of a polygon
          Pre   polygon perimeter calculated and stored
          Post  perimeter printed
*/
void Polygons :: printPeri (void)
{
//    Statements
      cout << "The perimeter of your polygon is "
           << perimeter << endl;
      return;
}     // Polygons printPeri
```

Header File {Triangle.h}

```
//    Triangle class definition
class Triangle : public Polygons
{
     private:
          double sideA;
          double sideB;
          double sideC;
          void calcArea (void);
          void calcPeri (void);
     public:
          // initialization constructor
          Triangle (double sideAIn, double sideBIn, double sideCIn);
};  // Class Triangle

/*   ============= Triangle :: Triangle ==============
/*   Initialization constructor for Triangle class
     Stores sides. Calculates area and perimeter.
          Pre   Given sideA, sideB, and sideC
          Post  data stored; area & perimeter calculated
*/
Triangle :: Triangle (double sideAIn, double sideBIn, double sideCIn)
{
//   Statements
     sideA = sideAIn;
     sideB = sideBIn;
     sideC = sideCIn;
     calcPeri();
     calcArea();
     return ;
}    // Triangle initialization constructor
```

```
/*   =============== Triangle :: calcArea ==============
     Calculates triangle area & stores in base class area
          Pre   sideA, sideB, sideC, & perimeter available
          Post  area calculated and stored
*/
void Triangle :: calcArea (void)
{
//   Local Declarations
     double calcPeri;

//   Statements
     calcPeri = perimeter / 2;
     area = (  calcPeri * (calcPeri - sideA) * (calcPeri - sideB) * (calcPeri - sideC) );
     area = sqrt(area);
     return;
}    // Triangle calcArea


/*   =============== Triangle :: calcPeri ==============
     Calculates perimeter & stores in base class area
          Pre   sideA, sideB, & sideC available
          Post  perimeter calculated and stored
*/
void Triangle :: calcPeri (void)
{
//   Statements
     perimeter = sideA + sideB + sideC;
     return;
}    // Triangle :: calcPeri
```