CIS 200-75 Program 0

Address.cs:

```
//Yen Hsieh Hsu, 5398334
//CIS 200-75
//Program 0
//9/10/2022
//Application to test addresses/letters.

using System;
using System.Collections.Generic;
using System.Text;

namespace CIS_200_75_Program_0
{
   public class Address
   {
      //defining constant values
      public const int MIN_ZIP = 0;
      public const int MAX_ZIP = 99999;

      //defining backing fields
      private string _name;
      private string _addressline1;
      private string _addressline2;
      private string _city;
      private string _state;
      private int _zipcode;

      //constructor
      //Pre: defined objects above not null
      //Post: address is created
      public Address(string name, string addressline1, string addressline2, string city, string
state, int zipcode)
      {
         Name = name;
         Addressline1 = addressline1;
         Addressline2 = addressline2;
         City = city;
```

```
            State = state;
            Zipcode = zipcode;
        }

        //overloaded constructor where address 2 is null
        public Address(string name, string addressline1, string city, string state, int zipcode) :
this(name, addressline1, string.Empty, city, state, zipcode)
        {

        }

        //set property for each data field, if field blank throw in error message
        public string Name {
            get { return _name; }
            set { if (string.IsNullOrWhiteSpace(value))
                    throw new ArgumentOutOfRangeException($"{nameof(Name)}",value,
$"{nameof(Name)} must not be empty.");
                else _name = value.Trim(); }
        }

        public string Addressline1 {
            get { return _addressline1; }
            set {
                if (string.IsNullOrWhiteSpace(value))
                    throw new ArgumentOutOfRangeException($"{nameof(Addressline1)}", value,
$"{nameof(Addressline1)} must not be empty.");
                else _addressline1 = value.Trim(); }
        }
        public string Addressline2 {
            get { return _addressline2; }
            set {

                if (value == null)
                    value = string.Empty;

                _addressline2 = value.Trim();
            }
        }
        public string City {
            get { return _city; }
            set {
                if (string.IsNullOrWhiteSpace(value))
                    throw new ArgumentOutOfRangeException($"{nameof(City)}", value,
$"{nameof(City)} must not be empty.");
```

```
            else _city = value.Trim(); }
        }
        public string State {
            get { return _state; }
            set {
                if (string.IsNullOrWhiteSpace(value))
                    throw new ArgumentOutOfRangeException($"{nameof(State)}", value,
$"{nameof(State)} must not be empty.");
                else _state = value.Trim(); }
        }

        //zipcode must be value between 0-99999
        public int Zipcode {
            get
            {
                return _zipcode;
            }
            set
            {
                if (value >= MIN_ZIP && value <= MAX_ZIP)
                    _zipcode = value;
                else
                    throw new ArgumentOutOfRangeException($"{nameof(Zipcode)}", value,
$"{nameof(Zipcode)} must not be empty and must be a 5 digit US zip code.");
            }
        }

        //ToString method, returns data
        public override string ToString()
        {
            string NL = Environment.NewLine;
            string result;

            result = $"{Name}{NL}{Addressline1}{NL}";

            //if address 2 is not empty, display data
            if (!string.IsNullOrWhiteSpace(Addressline2))
            {
                result += $"{Addressline2}{NL}";
            }

            result += $"{City}{NL}{State}{NL}{Zipcode:D5}";

            return result;
```

```
        }

    }
}
```

Letter.cs:

```
//Yen Hsieh Hsu, 5398334
//CIS 200-75
//Program 0
//9/10/2022
//Application to test addresses/letters.

using System;
using System.Collections.Generic;
using System.Text;

namespace CIS_200_75_Program_0
{
    public class Letter : Parcel
    {
        //define backing field
        private decimal _fixedCost;

        //must have a cost value
        //fixed cost is set
        public Letter(Address originAddress, Address destAddress, decimal cost ) :
base(originAddress, destAddress)
        {
            FixedCost = cost;
        }

        //set fixed costs to defined value.
        public decimal FixedCost
        {
            get { return _fixedCost; }
            set
            {
                if (value >=0)
                {
                    _fixedCost = value;
                }
                else
                {
```

```
                throw new ArgumentOutOfRangeException($"{nameof(FixedCost)}", value,
$"{nameof(FixedCost)} must be a value larger than 0.");
            }
        }
    }

    //get the costs, returns fixed cost value
    public override decimal CalcCost()
    {
        return FixedCost;
    }

    //returns data
    public override string ToString()
    {
        string NL = Environment.NewLine;

        return $"Letter{NL}{base.ToString()}";
    }
  }
}
```

Parcel.cs:

```
//Yen Hsieh Hsu, 5398334
//CIS 200-75
//Program 0
//9/10/2022
//Application to test addresses/letters.

using System;
using System.Collections.Generic;
using System.Text;

namespace CIS_200_75_Program_0
{
  public abstract class Parcel
  {
    //defining backing fields
    private Address _originAddress;
    private Address _destAddress;

    //set origin/dest addresses
    public Parcel(Address originAddress, Address destAddress)
```

```
    {
        OriginAddress = originAddress;
        DestinationAddress = destAddress;
    }

    //set property for each data field, if field blank throw in error message
    public Address OriginAddress
    {
        get { return _originAddress; }
        set {
            if (value == null)
            {
                throw new ArgumentOutOfRangeException($"{nameof(OriginAddress)}", value,
$"{nameof(OriginAddress)} must not be null.");
            }
            else
                _originAddress = value;
        }
    }

    public Address DestinationAddress {
        get { return _destAddress; }
        set {
            if (value == null)
            {
                throw new ArgumentOutOfRangeException($"{nameof(DestinationAddress)}",
value, $"{nameof(DestinationAddress)} must not be null.");
            }
            else
                _destAddress = value;
        }
    }

    //output cost
    public abstract decimal CalcCost();

    //ToString method, returns data. Displays cost in currency.
    public override string ToString()
    {
        string NL = Environment.NewLine;

        return $"Origin Address:{NL}{OriginAddress}{NL}{NL}Destination Address:{NL}" +
$"{DestinationAddress}{NL}Cost:{CalcCost():C}{NL}";
    }
```

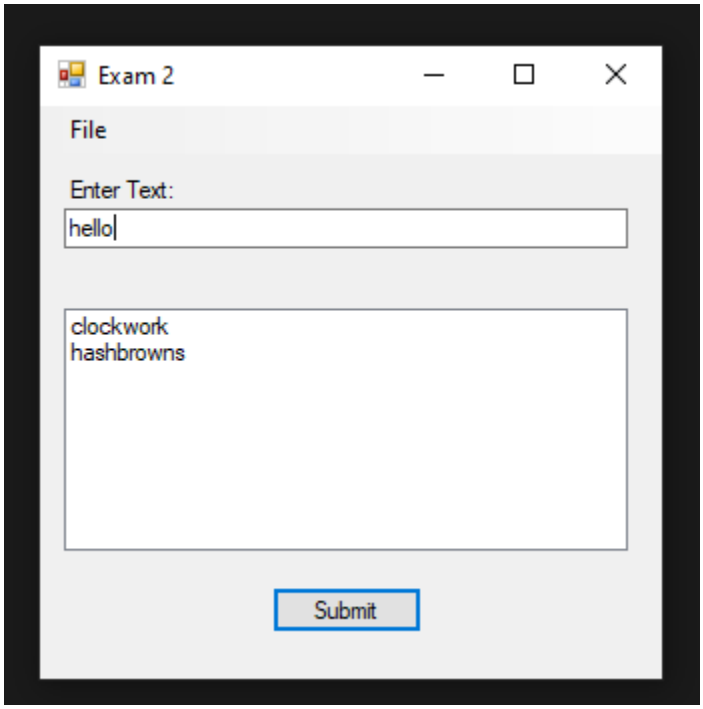
```
    }

}
```

Program.cs:

```
//Yen Hsieh Hsu, 5398334
//CIS 200-75
//Program 0
//9/10/2022
//Application to test addresses/letters.

using System;

namespace CIS_200_75_Program_0
{
    class Program
    {
        static void Main(string[] args)
        {

            //test data for addresses
            Address address1 = new Address("Lina Orwell", "500 South 4th Street", "Apt 3511",
"Louisville", "Kentucky", 40201);
            Address address2 = new Address("Gregory Flint", "1314 South 3rd Street", "Apt 106",
"Louisville", "Kentucky", 40228);
            Address address3 = new Address("John Smith", "11 Upper Cibolo Creek Rd", "",
"Boerne", "Texas", 78006);
            Address address4 = new Address("Jane Doe", "100 Palisade", "", "Austin", "Texas",
78737);


            //test data for letters
            Letter letter1 = new Letter(address1, address2, 0.50M);
            Letter letter2 = new Letter(address2, address3, 1.50M);
            Letter letter3 = new Letter(address3, address4, 2.70M);

            //store into array
            Parcel[] parcels = { letter1, letter2, letter3};

            //display data
```

```
            Console.WriteLine("Your parcels: ");
            Console.WriteLine();
            printData(parcels);

        }

        //print out data
        public static void printData(Parcel[] parcels)
        {
            for (int i = 0; i < parcels.Length; i++)
            {
                Console.WriteLine("Parcel #" + (i + 1));
                Console.WriteLine();
                Console.WriteLine(parcels[i]);
                Console.WriteLine("----------------------------");
            }
        }
    }
}
```

CIS200E2P3

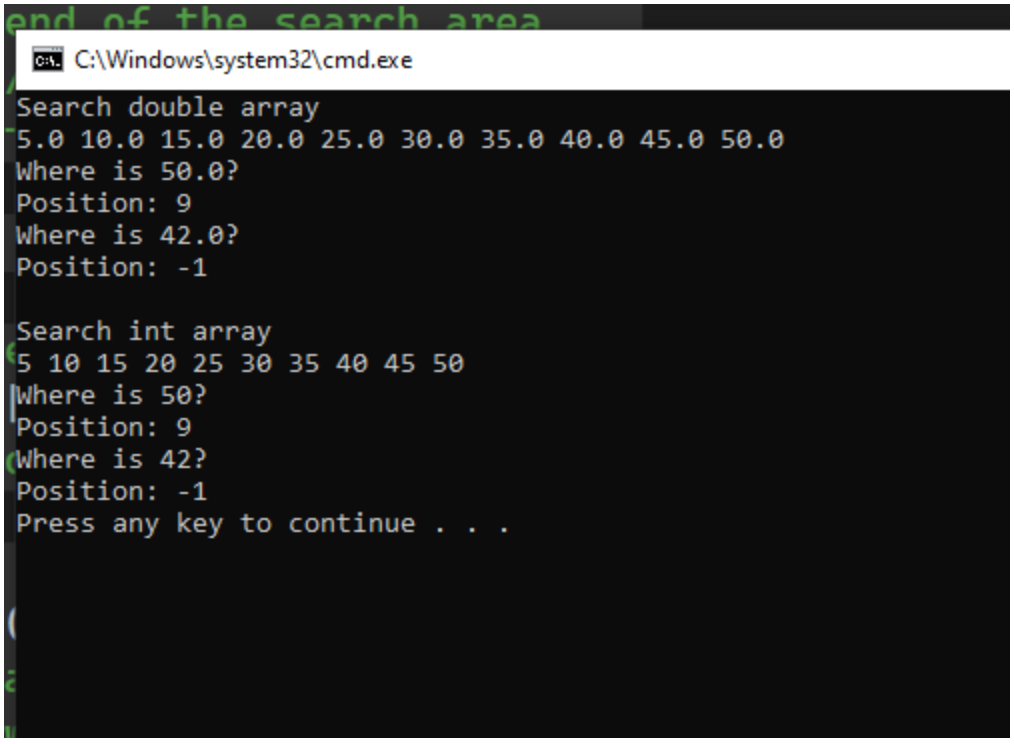


Form1.cs:

```
//Yen Hsieh Hsu
//Student ID: 5398334
//CIS200-75
//Due: 11/3/2022
//Simple application to add info to listbox


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CIS200E2P3
{
   public partial class Form1 : Form
   {

      public Form1()
      {
         InitializeComponent();

      }

      private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
      {
         String NL = Environment.NewLine;
         MessageBox.Show($"Exam 2 Part 3{NL}Yen Hsieh Hsu{NL}Student ID:
5398334{NL}CIS200-75{NL}Due: 11/3/2022{NL}Simple application to add info to listbox");
      }

      private void exitToolStripMenuItem_Click(object sender, EventArgs e)
      {
         Application.Exit();
      }
```

```
        internal string DataText
        {
            get { return dataText.Text; }
            set { dataText.Text = value; }
        }


        private void submitButton_Click(object sender, EventArgs e)
        {

            if (!string.IsNullOrWhiteSpace(dataText.Text))
            {
                dataList.Items.Add($"{DataText}");
                dataText.Clear();
            }
            else
            {
                MessageBox.Show("Please add valid data in the text box!");
            }
        }
    }
}
```

Final-200-F22



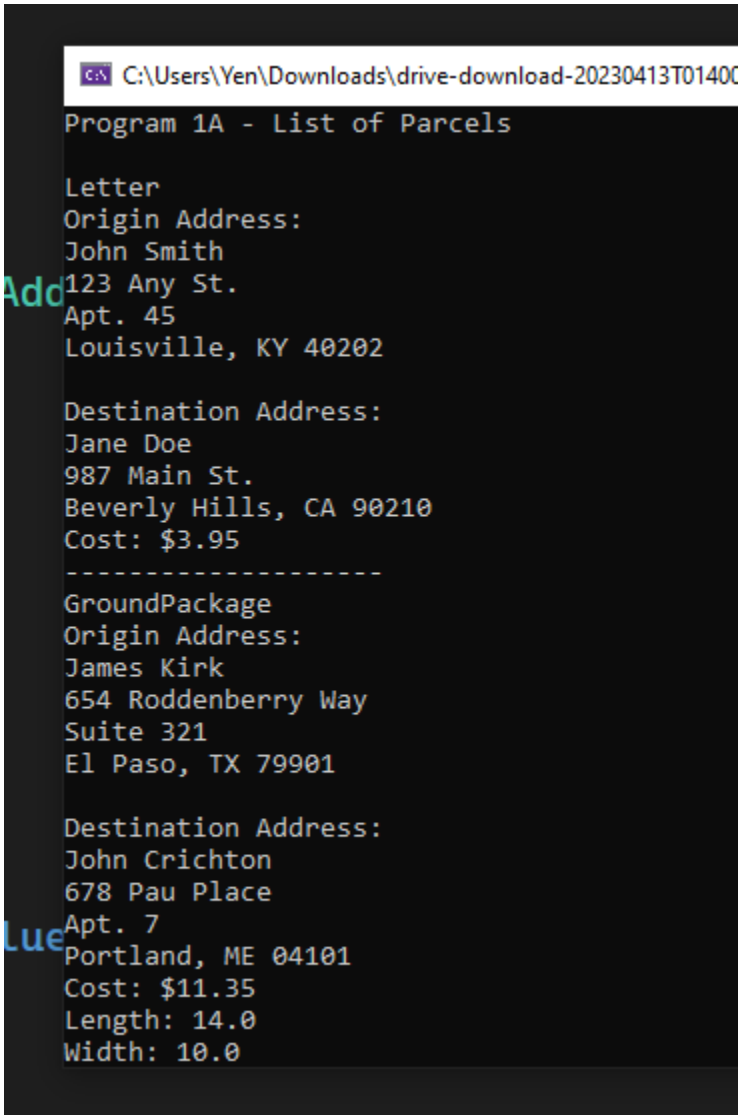
GenBinSearchTest.cs:

```
// Yen Hsieh Hsu, 5398334

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using static System.Console;


public class GenBinSearchTest
{
   public static void Main(string[] args)
   {
      int[] a1 = { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 };
      double[] a2 = { 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0, 50.0 };

      WriteLine("Search double array");
      foreach (double e in a2)
         Write($"{e:f1} ");
      WriteLine();
```

```
    WriteLine("Where is 50.0?");
    WriteLine($"Position: {BinarySearch(a2, 50.0)}");
    WriteLine("Where is 42.0?");
    WriteLine($"Position: {BinarySearch(a2, 42.0)}");

    WriteLine();

    WriteLine("Search int array");
    foreach (int e in a1)
        Write($"{e:d} ");
    WriteLine();

    WriteLine("Where is 50?");
    WriteLine($"Position: {BinarySearch(a1, 50)}");
    WriteLine("Where is 42?");
    WriteLine($"Position: {BinarySearch(a1, 42)}");
}

public static int BinarySearch<T>(T[] data, T searchElement) where T:IComparable<T>
{
    int low = 0; // low end of the search area
    int high = data.Length - 1; // high end of the search area
    int middle = (low + high + 1) / 2; // middle element
    int location = -1; // return value; -1 if not found

    do // loop to search for element
    {
        // if the element is found at the middle
        if (searchElement.CompareTo(data[middle]) == 0)
            location = middle; // location is the current middle

        // middle element is too high
        else if (searchElement.CompareTo(data[middle]) < 0)
            high = middle - 1; // eliminate the higher half
        else // middle element is too low
            low = middle + 1; // eliminate the lower half

        middle = (low + high + 1) / 2; // recalculate the middle
    } while ((low <= high) && (location == -1));

    return location; // return location of search key
} // end method BinarySearch
```

}

Prog0-for-1B



Address.cs

// Program 0
// CIS 200-76
// Fall 2020
// Due: 9/7/2020
// By: Andrew L. Wright (students use Grading ID)

```
// File: Address.cs
// This classes stores a typical US address consisting of name,
// two address lines, city, state, and 5 digit zip code.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Address
{
   public const int MIN_ZIP = 0;     // Minimum ZipCode value
   public const int MAX_ZIP = 99999; // Maximum ZipCode value

   private string _name;     // Address' name
   private string _address1; // First address line
   private string _address2; // Second address line, optional
   private string _city;     // Address' city
   private string _state;    // Address' state
   private int _zip;         // Address' zip code

   // Precondition:  name, address1, city, state may not be null, empty nor all whitespace
   //                MIN_ZIP <= zipcode <= MAX_ZIP
   // Postcondition: The address is created with the specified values for
   //                name, address1, address2, city, state, and zipcode
   public Address(string name, string address1, string address2,
      string city, string state, int zipcode)
   {
      Name = name;
      Address1 = address1;
      Address2 = address2;
      City = city;
      State = state;
      Zip = zipcode;
   }

   // Precondition:  name, address1, city, state may not be null, empty nor all whitespace
   //                MIN_ZIP <= zipcode <= MAX_ZIP
   // Postcondition: The address is created with the specified values for
   //                name, address1, city, state, and zipcode
   public Address(string name, string address1, string city,
      string state, int zipcode) :
      this(name, address1, string.Empty, city, state, zipcode)
```

```
{
   // No body needed
   // Calls previous constructor sending empty string for Address2
}

public string Name
{
   // Precondition:  None
   // Postcondition: The address' name has been returned
   get
   {
      return _name;
   }

   // Precondition:  value must not be null, empty nor all whitespace
   // Postcondition: The address' name has been set to the
   //                specified value
   set
   {
      if (string.IsNullOrWhiteSpace(value))
         throw new ArgumentOutOfRangeException($"{nameof(Name)}",
            value, $"{nameof(Name)} must not be empty");
      else
         _name = value.Trim();
   }
}

public string Address1
{
   // Precondition:  None
   // Postcondition: The address' first address line has been returned
   get
   {
      return _address1;
   }

   // Precondition:  value must not be null, empty nor all whitespace
   // Postcondition: The address' first address line has been set to
   //                the specified value
   set
   {
      if (string.IsNullOrWhiteSpace(value))
         throw new ArgumentOutOfRangeException($"{nameof(Address1)}",
            value, $"{nameof(Address1)} must not be empty");
```

```
        else
            _address1 = value.Trim();
    }
}

public string Address2
{
    // Precondition:  None
    // Postcondition: The address' second address line has been returned
    get
    {
        return _address2;
    }

    // Precondition:  None
    // Postcondition: The address' second address line has been set to
    //                the specified value
    set
    {
        if (value == null) // Just in case
            value = string.Empty;

        _address2 = value.Trim();
    }
}

public string City
{
    // Precondition:  None
    // Postcondition: The address' city has been returned
    get
    {
        return _city;
    }

    // Precondition:  value must not be null, empty nor all whitespace
    // Postcondition: The address' city has been set to the
    //                specified value
    set
    {
        if (string.IsNullOrWhiteSpace(value))
            throw new ArgumentOutOfRangeException($"{nameof(City)}",
                value, $"{nameof(City)} must not be empty");
        else
```

```
            _city = value.Trim();
        }
    }

    public string State
    {
        // Precondition:  None
        // Postcondition: The address' state has been returned
        get
        {
            return _state;
        }

        // Precondition:  value must not be null, empty nor all whitespace
        // Postcondition: The address' state has been set to the
        //                specified value
        set
        {
            if (string.IsNullOrWhiteSpace(value))
                throw new ArgumentOutOfRangeException($"{nameof(State)}",
                    value, $"{nameof(State)} must not be empty");
            else
                _state = value.Trim();
        }
    }

    public int Zip
    {
        // Precondition:  None
        // Postcondition: The address' zip code has been returned
        get
        {
            return _zip;
        }

        // Precondition:  MIN_ZIP <= value <= MAX_ZIP
        // Postcondition: The address' zip code has been set to the
        //                specified value
        set
        {
            if ((value >= MIN_ZIP) && (value <= MAX_ZIP))
                _zip = value;
            else
                throw new ArgumentOutOfRangeException($"{nameof(Zip)}", value,
```

```
                $"{nameof(Zip)} must be U.S. 5 digit zip code");
        }
    }

    // Precondition:  None
    // Postcondition: A String with the address' data has been returned
    public override string ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut
        string result;                   // Builds formatted string

        result = $"{Name}{NL}{Address1}{NL}";

        if (!string.IsNullOrWhiteSpace(Address2)) // Is Address2 not empty?
            result += $"{Address2}{NL}";

        result += $"{City}, {State} {Zip:D5}";

        // -- OR --
        // Compact Way
        //result = $"{Name}{NL}{Address1}{NL}{Address2}" +
        //    $"{(String.IsNullOrWhiteSpace(Address2) ? string.Empty : NL)}" +
        //    $"{City}, {State} {Zip:D5}";

        return result;
    }
}
```

AirPackage.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


    public abstract class AirPackage : Package
    {
        public const double HEAVY_THRESHOLD = 75; //min weight for package to be considered
heavy
        public const double LARGE_THRESHOLD = 100; //min dimension for package to be
considered large
```

```
        public AirPackage(Address originAddress, Address destAddress, double length, double
width, double height, double weight) : base(originAddress, destAddress, length, width, height,
weight)
        {
            //left empty
        }

        //checking whether or not the package is heavy or large. Should return a T/F.
        public bool IsHeavy()
        {
            return (Weight >= HEAVY_THRESHOLD);
        }

        public bool IsLarge()
        {
            return (TotalDimension >= LARGE_THRESHOLD);
        }

        //display result
        public override string ToString()
        {
            string NL = Environment.NewLine; // NewLine shortcut

            return $"Air{base.ToString()}{NL}Heavy: {IsHeavy()}{NL}Large: {IsLarge()}";
        }
    }
```

GroundPackage.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


    internal class GroundPackage : Package
    {
        public GroundPackage(Address originAddress, Address destAddress, double length,
double width, double height, double weight) : base(originAddress, destAddress, length, width,
height, weight)
```

```
        {
            //left empty
        }

        //read only property
        public int ZoneDistance
        {
            get
            {
                const int FIRST_DIGIT_FACTOR = 10000; //to find first digit of orig/dest zip code
                int dist;

                dist = Math.Abs((OriginAddress.Zip / FIRST_DIGIT_FACTOR) -
(DestinationAddress.Zip / FIRST_DIGIT_FACTOR));

                return dist;
            }
        }

        public override decimal CalcCost()
        {
            const double DIM_FACTOR = 0.15;
            const double WEIGHT_FACTOR = 0.07;

            return (decimal)(DIM_FACTOR * TotalDimension + WEIGHT_FACTOR * (ZoneDistance
+ 1) * Weight);
        }

        public override string ToString()
        {
            string NL = Environment.NewLine; // NewLine shortcut

            return $"Ground{base.ToString()}{NL}Zone Distance: {ZoneDistance}";
        }
    }
```

Letter.cs

```
// File: Letter.cs
// The Letter class is a concrete derived class of Parcel. Letters
// have a fixed cost.
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Letter : Parcel
{
   private decimal _fixedCost; // Cost to send letter

   // Precondition:  cost >= 0
   // Postcondition: The letter is created with the specified values for
   //                origin address, destination address, and cost
   public Letter(Address originAddress, Address destAddress, decimal cost)
      : base(originAddress, destAddress)
   {
      FixedCost = cost;
   }

   private decimal FixedCost // Helper property
   {
      // Precondition:  None
      // Postcondition: The letter's fixed cost has been returned
      get
      {
         return _fixedCost;
      }

      // Precondition:  value >= 0
      // Postcondition: The letter's fixed cost has been set to the
      //                specified value
      set
      {
         if (value >= 0)
            _fixedCost = value;
         else
            throw new ArgumentOutOfRangeException($"{nameof(FixedCost)}", value,
               $"{nameof(FixedCost)} must be >= 0");
      }
   }

   // Precondition:  None
   // Postcondition: The letter's cost has been returned
   public override decimal CalcCost()
   {
```

```
        return FixedCost;
    }

    // Precondition:  None
    // Postcondition: A String with the letter's data has been returned
    public override string ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut

        return $"Letter{NL}{base.ToString()}"; // Let base class help
    }
}
```

NextDayAirPackage.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


    public class NextDayAirPackage : AirPackage
    {
        private decimal _expFee;

        //constructor with given values.
        public NextDayAirPackage(Address originAddress, Address destAddress, double length,
double width, double height, double weight, decimal expFee) : base(originAddress,
destAddress, length, width, height, weight)
        {
            ExpressFee = expFee;
        }

        //setting property
        public decimal ExpressFee
        {
            get { return _expFee; }
            set
            {
                if (value > 0)
                {
                    _expFee = value;
```

```
            }
            else
                throw new ArgumentOutOfRangeException(nameof(ExpressFee), value,
$"{nameof(ExpressFee)} must be > 0");
        }
    }

    //calculate the cost of a heavy/large package
    public override decimal CalcCost()
    {
        //given in prompt
        const double DIM_FACTOR = 0.35;
        const double WEIGHT_FACTOR = 0.25;
        const double HEAVY_FACTOR = 0.2;
        const double LARGE_FACTOR = 0.22;

        decimal cost;

        cost = (decimal)(DIM_FACTOR * TotalDimension + WEIGHT_FACTOR * Height) +
ExpressFee;

        if (IsHeavy())
        {
            cost += (decimal)(HEAVY_FACTOR * Weight);
        }

        if (IsLarge())
        {
            cost += (decimal)(LARGE_FACTOR * TotalDimension);
        }

        return cost;
    }

    //display data
    public override string ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut

        return $"Next Day {base.ToString()}{NL}Express Fee: {ExpressFee:C}";
    }
  }
```

Package.cs

```
//program should be same or similar to the guided one on panopto since I followed that one to complete this.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


    //class inherited from parcel
    public abstract class Package : Parcel
    {
        //dimensions of package
        private double _length;
        private double _width;
        private double _height;
        private double _weight;

        //constructor with given values
        public Package(Address destAddress, Address originAddress, double length, double width, double height, double weight) : base(destAddress, originAddress)
        {
            _length = length;
            _width = width;
            _height = height;
            _weight = weight;
        }

        //setting the properties
        public double Length
        {   get
            {
                return _length;
            }

            set
            {
                if (value > 0)
                {
                    _length = value;
                }
                else
```

```
            {
                throw new ArgumentOutOfRangeException(nameof(Length), value,
$"{nameof(Length)} must be > 0");
            }
        }
    }

    public double Width
    { get
        {
            return _width;
        }
      set
        {
            if (value > 0)
            {
                _width = value;
            }
            else
                throw new ArgumentOutOfRangeException(nameof(Width), value,
$"{nameof(Width)} must be > 0");
        }
    }

    public double Height
    {
        get
        {
            return _height;
        }
        set
        {
            if (value > 0)
            {
                _height = value;
            }
            else
                throw new ArgumentOutOfRangeException(nameof(Height), value,
$"{nameof(Height)} must be > 0");
        }
    }

    public double Weight
    {
```

```
            get
            {
                return _weight;
            }
            set
            {
                if (value > 0)
                {
                    _weight = value;
                }
                else
                    throw new ArgumentOutOfRangeException(nameof(Weight), value,
$"{nameof(Weight)} must be > 0");
            }
        }


        protected double TotalDimension
        {
            get
            {
                return (Length + Width + Height);
            }
        }

        //returns data
        public override String ToString()
        {
            string NL = Environment.NewLine; // NewLine shortcut

            return $"Package{NL}{base.ToString()}{NL}Length: {Length:N1}{NL}Width:
{Width:N1}{NL}" +
                $"Weight: {Height:N1}{NL}Weight: {Weight:N1}";
        }

    }
```

Parcel.cs

```
// Program 0
// CIS 200-76
// Fall 2020
// Due: 9/7/2020
// By: Andrew L. Wright (students use Grading ID)
```

```
// File: Parcel.cs
// Parcel serves as the abstract base class of the Parcel hierachy.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public abstract class Parcel
{
   private Address _originAddress; // The origin address for the parcel
   private Address _destAddress;   // The destination address for the parcel

   // Precondition:  None
   // Postcondition: The parcel is created with the specified values for
   //                origin address and destination address
   public Parcel(Address originAddress, Address destAddress)
   {
      OriginAddress = originAddress;
      DestinationAddress = destAddress;
   }

   public Address OriginAddress
   {
      // Precondition:  None
      // Postcondition: The parcel's origin address has been returned
      get
      {
         return _originAddress;
      }

      // Precondition:  value must not be null
      // Postcondition: The parcel's origin address has been set to the
      //                specified value
      set
      {
         if (value == null)
         {
            throw new ArgumentOutOfRangeException($"{nameof(OriginAddress)}",
               value, $"{nameof(OriginAddress)} must not be null");
         }
         else
            _originAddress = value;
```

```
        }
    }

    public Address DestinationAddress
    {
        // Precondition:  None
        // Postcondition: The parcel's destination address has been returned
        get
        {
            return _destAddress;
        }

        // Precondition:  value must not be null
        // Postcondition: The parcel's destination address has been set to the
        //                specified value
        set
        {
            if (value == null)
            {
                throw new ArgumentOutOfRangeException($"{nameof(DestinationAddress)}",
                    value, $"{nameof(DestinationAddress)} must not be null");
            }
            else
                _destAddress = value;
        }
    }

    // Precondition:  None
    // Postcondition: The parcel's cost has been returned
    public abstract decimal CalcCost();

    // Precondition:  None
    // Postcondition: A String with the parcel's data has been returned
    public override String ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut

        return $"Origin Address:{NL}{OriginAddress}{NL}{NL}Destination Address:{NL}" +
            $"{DestinationAddress}{NL}Cost: {CalcCost():C}";
    }
}
```

Program.cs

```
// Yen Hsieh Hsu, 5398334
```

```
// Program 1B
// CIS 200-75
// Due: 10/12/2022
// Program to calc cost of packages, and sort by zip, type, cost, etc.


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Console;

namespace Prog0
{
   class Program
   {

      // Precondition:  None
      // Postcondition: Small list of Parcels is created and displayed
      static void Main(string[] args)
      {
         bool VERBOSE = false;

         Address a1 = new Address("John Smith", "123 Any St.", "Apt. 45",
            "Louisville", "KY", 40202); // Test Address 1
         Address a2 = new Address("Jane Doe", "987 Main St.",
            "Beverly Hills", "CA", 90210); // Test Address 2
         Address a3 = new Address("James Kirk", "654 Roddenberry Way", "Suite 321",
            "El Paso", "TX", 79901); // Test Address 3
         Address a4 = new Address("John Crichton", "678 Pau Place", "Apt. 7",
            "Portland", "ME", 04101); // Test Address 4
         Address a5 = new Address("Lynn Smith", "122 Main St.",
            "Louisville", "KY", 40208); // Test Address 5
         Address a6 = new Address("Lina Doe", "800 Main St.",
            "Beverly Hills", "CA", 90212); // Test Address 6
         Address a7 = new Address("Robert Kirk", "654 Roddenberry Way", "Suite 200",
            "El Paso", "TX", 79902); // Test Address 7
         Address a8 = new Address("Joe Nelson", "678 Pau Place", "Apt. 17",
            "Portland", "ME", 04100); // Test Address 8


         //Test value/data
         Letter letter1 = new Letter(a1, a2, 3.95M);
```

```
        GroundPackage ground1 = new GroundPackage(a3, a4, 14, 10, 5, 12.5);
        NextDayAirPackage air1 = new NextDayAirPackage(a1, a4, 25, 15, 15, 85, 7.50M);
        TwoDayAirPackage air2 = new TwoDayAirPackage(a4, a1, 46.5, 39.5, 28.0, 80.5,
TwoDayAirPackage.Delivery.Saver);
        GroundPackage ground2 = new GroundPackage(a5, a8, 15, 15, 25, 35);
        NextDayAirPackage air3 = new NextDayAirPackage(a1, a7, 15, 15, 15, 45, 7.50M);
        TwoDayAirPackage air4 = new TwoDayAirPackage(a4, a6, 30, 30, 29, 73,
TwoDayAirPackage.Delivery.Early);
        NextDayAirPackage air5 = new NextDayAirPackage(a1, a3, 30, 35, 35, 105, 9.50M);

        List<Parcel> parcels = new List<Parcel> {letter1, ground1, ground2, air1, air2, air3, air4,
air5}; // Test list of parcels

        // Display data
        WriteLine("Program 1A - List of Parcels\n");

        foreach (Parcel p in parcels)
        {
            WriteLine(p);
            WriteLine("--------------------");
        }
        Pause();

        //Linq section for 1B
        //used guided recording provided by instructor, so a lot will be the same


        //select parcel by destination zip and ordered in a descending order
        var parcelsByDestZip =
            from p in parcels
            orderby p.DestinationAddress.Zip descending
            select p;

        WriteLine("Parcels by Destination Zip (descending order):");
        WriteLine("--------------------");

        foreach (Parcel p in parcelsByDestZip)
        {
            if (VERBOSE)
            {
                WriteLine(p);
                WriteLine("--------------------");
            }
            else
```

```
            WriteLine($"{p.DestinationAddress.Zip:D5}");
        }
        Pause();

        //select parcels, order by cost
        var parcelsByCost =
            from p in parcels
            orderby p.CalcCost()
            select p;

        WriteLine("Parcels by Cost:");
        WriteLine("--------------------");

        foreach (Parcel p in parcelsByCost)
        {
            if(VERBOSE)
            {
                WriteLine(p);
                WriteLine("--------------------");
            }
            else
                WriteLine($"{p.CalcCost(), 8:C}");
        }
        Pause();

        //select parcels in descending order, by type and cost
        //had to remove namespace Prog0 on the other ones so that it would not display
Prog0.ClassName when program is run.
        var parcelsByTypeCost =
            from p in parcels
            orderby p.GetType().ToString(), p.CalcCost() descending
            select p;

        WriteLine("Parcels by Type and Cost:");
        WriteLine("--------------------");

        foreach (Parcel p in parcelsByTypeCost)
        {
            if (VERBOSE)
            {
                WriteLine(p);
                WriteLine("--------------------");
            }
            else
```

```
            WriteLine("{0, -17} {1, 8:C}", p.GetType().ToString(), p.CalcCost());
        }
        Pause();

        //select airpackage, in descending order, by weight
        var heavyAirpackagesByWeight =
            from p in parcels
            let ap = p as AirPackage
            where (ap != null) && ap.IsHeavy()
            orderby ap.Weight descending
            select ap;

        WriteLine("Heavy AirPackages by Weight (descending order):");
        WriteLine("----------");

        foreach (AirPackage ap in heavyAirpackagesByWeight)
        {
            if (VERBOSE)
            {
                WriteLine(ap);
                WriteLine("----------");
            }
            else
                WriteLine("{0, -17} {1, 4:F1}", ap.GetType().ToString(), ap.Weight);
        }
        Pause();


    }

    //reads user pressing enter to have the program run the next step, and clear content of
previous step.
    public static void Pause()
    {
        WriteLine("Press Enter to continue...");
        ReadLine();

        Clear();
    }
  }
}
```

TwoDayAirPackage.cs

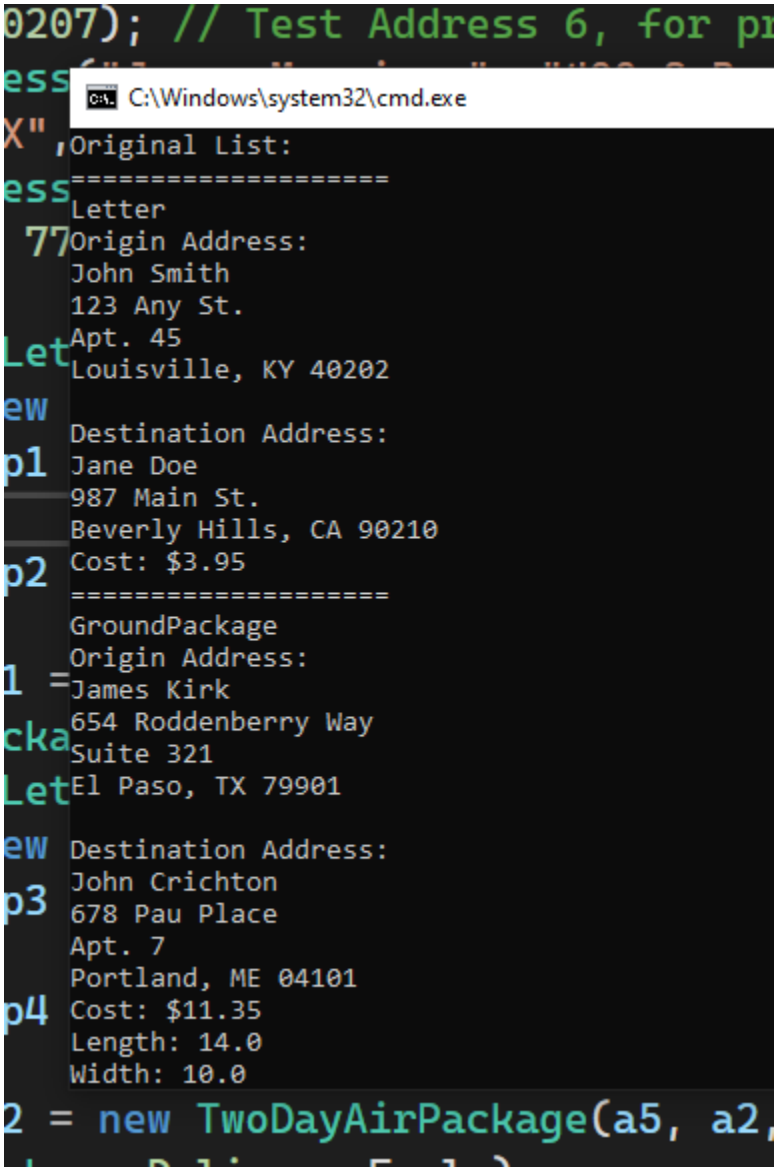
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


    public class TwoDayAirPackage : AirPackage
    {
        //enum used, Early and Saver assigned
        public enum Delivery { Early, Saver }

        private Delivery _deliveryType;

        //constructor
        public TwoDayAirPackage(Address originAddress, Address destAddress, double length,
double width, double height, double weight, Delivery delType) : base(originAddress,
destAddress, length, width, height, weight)
        {
            DeliveryType = delType;
        }

        //setting property
        public Delivery DeliveryType
        {
            get { return _deliveryType; }
            set
            {
                if (Enum.IsDefined(typeof(Delivery), value))
                {
                    _deliveryType = value;
                }
                else
                    throw new ArgumentOutOfRangeException(nameof(DeliveryType), value,
$"{nameof(DeliveryType)} must be {nameof(Delivery.Early)}" + $"or {nameof(Delivery.Saver)}");
            }
        }

        //calculate the cost of package depending on the delivery type
        public override decimal CalcCost()
        {
            const double DIM_FACTOR = 0.18;
            const double WEIGHT_FACTOR = 0.2;
            const decimal DISCOUNT_FACTOR = 0.15M;
```

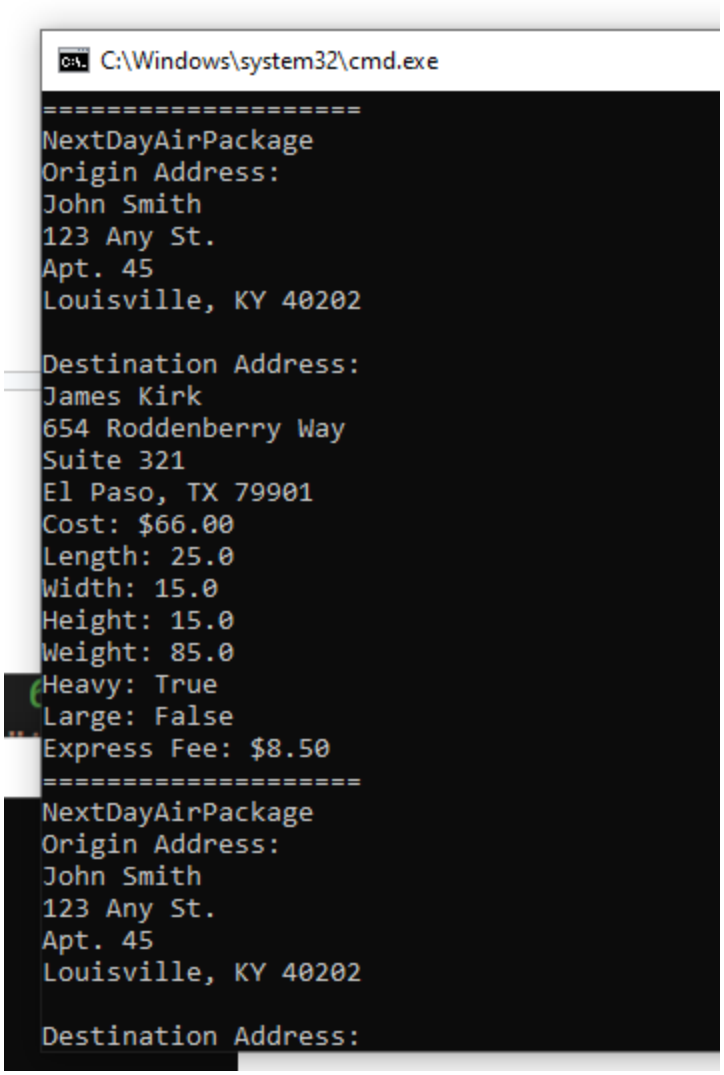
```
        decimal cost;

        cost = (decimal)(DIM_FACTOR * TotalDimension + WEIGHT_FACTOR * Weight);

        if (DeliveryType == Delivery.Saver)
        {
            cost *= (1-DISCOUNT_FACTOR);
        }

        return cost;
    }

    //returns data
    public override string ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut

        return $"Two Day {base.ToString()}{NL}Delivery Type: {DeliveryType}";
    }

}
```

Prog1A - Prog3

The above program files are continuation of Prog0, final result at Prog4.

Prog4


0207); // Test Address 6, for pr
C:\Windows\system32\cmd.exe
Original List:
==========
Letter
Origin Address:
John Smith
123 Any St.
Apt. 45
Louisville, KY 40202
Destination Address:
Jane Doe
987 Main St.
Beverly Hills, CA 90210
Cost: $3.95
==========
GroundPackage
Origin Address:
James Kirk
654 Roddenberry Way
Suite 321
El Paso, TX 79901
Destination Address:
John Crichton
678 Pau Place
Apt. 7
Portland, ME 04101
Cost: $11.35
Length: 14.0
Width: 10.0
2 = new TwoDayAirPackage(a5, a2,



Address.cs:

```
// Program 0
// CIS 200-76
// Fall 2020
// Due: 9/7/2020
// By: Andrew L. Wright (students use Grading ID)

// File: Address.cs
// This classes stores a typical US address consisting of name,
// two address lines, city, state, and 5 digit zip code.

using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;

public class Address
{
   public const int MIN_ZIP = 0;     // Minimum ZipCode value
   public const int MAX_ZIP = 99999; // Maximum ZipCode value

   private string _name;     // Address' name
   private string _address1; // First address line
   private string _address2; // Second address line, optional
   private string _city;     // Address' city
   private string _state;    // Address' state
   private int _zip;         // Address' zip code

   // Precondition:  name, address1, city, state may not be null, empty nor all whitespace
   //              MIN_ZIP <= zipcode <= MAX_ZIP
   // Postcondition: The address is created with the specified values for
   //              name, address1, address2, city, state, and zipcode
   public Address(string name, string address1, string address2,
      string city, string state, int zipcode)
   {
      Name = name;
      Address1 = address1;
      Address2 = address2;
      City = city;
      State = state;
      Zip = zipcode;
   }

   // Precondition:  name, address1, city, state may not be null, empty nor all whitespace
   //              MIN_ZIP <= zipcode <= MAX_ZIP
   // Postcondition: The address is created with the specified values for
   //              name, address1, city, state, and zipcode
   public Address(string name, string address1, string city,
      string state, int zipcode) :
      this(name, address1, string.Empty, city, state, zipcode)
   {
      // No body needed
      // Calls previous constructor sending empty string for Address2
   }

   public string Name
   {
      // Precondition:  None
```

```
    // Postcondition: The address' name has been returned
    get
    {
        return _name;
    }

    // Precondition:  value must not be null, empty nor all whitespace
    // Postcondition: The address' name has been set to the
    //                specified value
    set
    {
        if (string.IsNullOrWhiteSpace(value))
            throw new ArgumentOutOfRangeException($"{nameof(Name)}",
                value, $"{nameof(Name)} must not be empty");
        else
            _name = value.Trim();
    }
}

public string Address1
{
    // Precondition:  None
    // Postcondition: The address' first address line has been returned
    get
    {
        return _address1;
    }

    // Precondition:  value must not be null, empty nor all whitespace
    // Postcondition: The address' first address line has been set to
    //                the specified value
    set
    {
        if (string.IsNullOrWhiteSpace(value))
            throw new ArgumentOutOfRangeException($"{nameof(Address1)}",
                value, $"{nameof(Address1)} must not be empty");
        else
            _address1 = value.Trim();
    }
}

public string Address2
{
    // Precondition:  None
```

```
    // Postcondition: The address' second address line has been returned
    get
    {
        return _address2;
    }

    // Precondition:  None
    // Postcondition: The address' second address line has been set to
    //                the specified value
    set
    {
        if (value == null) // Just in case
            value = string.Empty;

        _address2 = value.Trim();
    }
}

public string City
{
    // Precondition:  None
    // Postcondition: The address' city has been returned
    get
    {
        return _city;
    }

    // Precondition:  value must not be null, empty nor all whitespace
    // Postcondition: The address' city has been set to the
    //                specified value
    set
    {
        if (string.IsNullOrWhiteSpace(value))
            throw new ArgumentOutOfRangeException($"{nameof(City)}",
                value, $"{nameof(City)} must not be empty");
        else
            _city = value.Trim();
    }
}

public string State
{
    // Precondition:  None
    // Postcondition: The address' state has been returned
```

```
    get
    {
        return _state;
    }

    // Precondition:  value must not be null, empty nor all whitespace
    // Postcondition: The address' state has been set to the
    //                specified value
    set
    {
        if (string.IsNullOrWhiteSpace(value))
            throw new ArgumentOutOfRangeException($"{nameof(State)}",
                value, $"{nameof(State)} must not be empty");
        else
            _state = value.Trim();
    }
}

public int Zip
{
    // Precondition:  None
    // Postcondition: The address' zip code has been returned
    get
    {
        return _zip;
    }

    // Precondition:  MIN_ZIP <= value <= MAX_ZIP
    // Postcondition: The address' zip code has been set to the
    //                specified value
    set
    {
        if ((value >= MIN_ZIP) && (value <= MAX_ZIP))
            _zip = value;
        else
            throw new ArgumentOutOfRangeException($"{nameof(Zip)}", value,
                $"{nameof(Zip)} must be U.S. 5 digit zip code");
    }
}

// Precondition:  None
// Postcondition: A String with the address' data has been returned
public override string ToString()
{
```

```
        string NL = Environment.NewLine; // NewLine shortcut
        string result;                   // Builds formatted string

        result = $"{Name}{NL}{Address1}{NL}";

        if (!string.IsNullOrWhiteSpace(Address2)) // Is Address2 not empty?
            result += $"{Address2}{NL}";

        result += $"{City}, {State} {Zip:D5}";

        // -- OR --
        // Compact Way
        //result = $"{Name}{NL}{Address1}{NL}{Address2}" +
        //    $"{(String.IsNullOrWhiteSpace(Address2) ? string.Empty : NL)}" +
        //    $"{City}, {State} {Zip:D5}";

        return result;
    }
}
```

AirPackage.cs:

```
// Program 1A
// CIS 200-01
// Fall 2020
// Due: 9/21/2020
// By: Andrew L. Wright (students use Grading ID)

// File: AirPackage.cs
// The AirPackage class is an abstract derived class from Package. It is able
// to determine if the package is heavy or large.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public abstract class AirPackage : Package
{
    public const double HEAVY_THRESHOLD = 75; // Min weight of heavy package
    public const double LARGE_THRESHOLD = 100; // Min dimensions of large package

    // Precondition:  pLength > 0, pWidth > 0, pHeight > 0,
    //                pWeight > 0
```

```
    // Postcondition: The air package is created with the specified values for
    //                origin address, destination address, length, width,
    //                height, and weight
    public AirPackage(Address originAddress, Address destAddress,
        double pLength, double pWidth, double pHeight, double pWeight)
        : base(originAddress, destAddress, pLength, pWidth, pHeight, pWeight)
    {
        // All work done in base class constructor
    }

    // Precondition:  None
    // Postcondition: Returns true if air package is considered heavy
    //                else returns false
    public bool IsHeavy()
    {
        return (Weight >= HEAVY_THRESHOLD);
    }

    // Precondition:  None
    // Postcondition: Returns true if air package is considered large
    //                else returns false
    public bool IsLarge()
    {
        return (TotalDimension >= LARGE_THRESHOLD);
    }

    // Precondition:  None
    // Postcondition: A String with the air package's data has been returned
    public override string ToString()
    {
        string NL = Environment.NewLine; // Newline shorthand

        return $"Air{base.ToString()}{NL}Heavy: {IsHeavy()}{NL}Large: {IsLarge()}";
    }
}
```

DescendingByDestZipComparer.cs:

```
//Yen Hsieh Hsu, 5398334
//CIS 200-75
//Due 12/3/22
//Application to list out parcels and sort them

using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Prog1
{
   internal class DescendingByDestZipComparer : Comparer<Parcel>
   {
      //pre: none
      //post: returns 0 if p1 zip = p2 zip, returns -1 if p1 zip is less than p2 zip, returns 1 if p1 zip
is greater than p2 zip
      public override int Compare(Parcel p1, Parcel p2)
      {
         int zip1; //p1 zip
         int zip2; //p2 zip

         //handling null values
         if (p1 == null && p2 == null) //both values null?
            return 0;
         if (p1 == null) //only p1 null?
            return -1;
         if (p2 == null) //only p2 null?
            return 1;

         zip1 = p1.DestinationAddress.Zip;
         zip2 = p2.DestinationAddress.Zip;

         return (-1) * zip1.CompareTo(zip2); //returns values in descending order
      }
   }
}
```

GroundPackage.cs:

```
// Program 1A
// CIS 200-01
// Fall 2020
// Due: 9/21/2020
// By: Andrew L. Wright (students use Grading ID)

// File: GroundPackage.cs
// The Package class is a concrete derived class from Package. It adds
// a Zone Distance.
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


public class GroundPackage : Package
{
   // Precondition:  pLength > 0, pWidth > 0, pHeight > 0,
   //              pWeight > 0
   // Postcondition: The ground package is created with the specified values for
   //              origin address, destination address, length, width,
   //              height, and weight
   public GroundPackage(Address originAddress, Address destAddress,
      double pLength, double pWidth, double pHeight, double pWeight)
      : base(originAddress, destAddress, pLength, pWidth, pHeight, pWeight)
   {
      // All work done in base class constructor
   }

   public int ZoneDistance
   {
      // Precondition:  None
      // Postcondition: The ground package's zone distance is returned.
      //              The zone distance is the positive difference between the
      //              first digit of the origin address' zip code and the first
      //              digit of the destination address' zip code.
      get
      {
         const int FIRST_DIGIT_FACTOR = 10000; // Denominator to extract 1st digit
         int dist;                        // Calculated zone distance

         dist = Math.Abs((OriginAddress.Zip / FIRST_DIGIT_FACTOR) - (DestinationAddress.Zip
/ FIRST_DIGIT_FACTOR));

         return dist;
      }
   }

   // Precondition:  None
   // Postcondition: The ground package's cost has been returned
   public override decimal CalcCost()
   {
```

```
        const double DIM_FACTOR = .15;    // Dimension coefficient in cost equation
        const double WEIGHT_FACTOR = .07; // Weight coefficient in cost equation

        return (decimal)(DIM_FACTOR * TotalDimension + WEIGHT_FACTOR * (ZoneDistance +
1) * Weight);
    }

    // Precondition:  None
    // Postcondition: A String with the ground package's data has been returned
    public override string ToString()
    {
        string NL = Environment.NewLine; // Newline shorthand

        return $"Ground{base.ToString()}{NL}Zone Distance: {ZoneDistance}";
    }
}
```

Letter.cs:

```
// Program 0
// CIS 200-76
// Fall 2020
// Due: 9/7/2020
// By: Andrew L. Wright (students use Grading ID)

// File: Letter.cs
// The Letter class is a concrete derived class of Parcel. Letters
// have a fixed cost.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Letter : Parcel
{
    private decimal _fixedCost; // Cost to send letter

    // Precondition:  cost >= 0
    // Postcondition: The letter is created with the specified values for
    //                origin address, destination address, and cost
    public Letter(Address originAddress, Address destAddress, decimal cost)
        : base(originAddress, destAddress)
    {
```

```
        FixedCost = cost;
    }

    private decimal FixedCost // Helper property
    {
        // Precondition:  None
        // Postcondition: The letter's fixed cost has been returned
        get
        {
            return _fixedCost;
        }

        // Precondition:  value >= 0
        // Postcondition: The letter's fixed cost has been set to the
        //                specified value
        set
        {
            if (value >= 0)
                _fixedCost = value;
            else
                throw new ArgumentOutOfRangeException($"{nameof(FixedCost)}", value,
                    $"{nameof(FixedCost)} must be >= 0");
        }
    }

    // Precondition:  None
    // Postcondition: The letter's cost has been returned
    public override decimal CalcCost()
    {
        return FixedCost;
    }

    // Precondition:  None
    // Postcondition: A String with the letter's data has been returned
    public override string ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut

        return $"Letter{NL}{base.ToString()}"; // Let base class help
    }
}
```

NextDayAirPackage.cs:

```
// Program 1A
// CIS 200-01
// Fall 2020
// Due: 9/21/2020
// By: Andrew L. Wright (students use Grading ID)

// File: NextDayAirPackage.cs
// The NextDayAirPackage class is a concrete derived class from AirPackage. It adds
// an express fee.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class NextDayAirPackage : AirPackage
{
   private decimal _expressFee; // Next day air package's express fee

   // Precondition:  pLength > 0, pWidth > 0, pHeight > 0,
   //                pWeight > 0, expFee >= 0
   // Postcondition: The next day air package is created with the specified values for
   //                origin address, destination address, length, width,
   //                height, weight, and express fee
   public NextDayAirPackage(Address originAddress, Address destAddress,
      double pLength, double pWidth, double pHeight, double pWeight, decimal expFee)
      : base(originAddress, destAddress, pLength, pWidth, pHeight, pWeight)
   {
      ExpressFee = expFee;
   }

   public decimal ExpressFee
   {
      // Precondition:  None
      // Postcondition: The next day air package's express fee has been returned
      get
      {
         return _expressFee;
      }

      // Precondition:  value >= 0
      // Postcondition: The next day air package's express fee has been set to the
      //                specified value
```

```
        private set // Helper set property
        {
            if (value >= 0)
                _expressFee = value;
            else
                throw new ArgumentOutOfRangeException(nameof(ExpressFee), value,
                    $"{nameof(ExpressFee)} must be >= 0");
        }
    }

    // Precondition:  None
    // Postcondition: The next day air package's cost has been returned
    public override decimal CalcCost()
    {
        const double DIM_FACTOR = .35;    // Dimension coefficient in cost equation
        const double WEIGHT_FACTOR = .25; // Weight coefficient in cost equation
        const double HEAVY_FACTOR = .2;   // Heavy coefficient in cost equation
        const double LARGE_FACTOR = .22;  // Large coefficient in cost equation

        decimal cost; // Running total of cost of package

        cost = (decimal)(DIM_FACTOR * TotalDimension + WEIGHT_FACTOR * Weight) +
ExpressFee;

        if (IsHeavy())
            cost += (decimal)(HEAVY_FACTOR * Weight);
        if (IsLarge())
            cost += (decimal)(LARGE_FACTOR * TotalDimension);

        return cost;
    }

    // Precondition:  None
    // Postcondition: A String with the next day air package's data has been returned
    public override string ToString()
    {
        string NL = Environment.NewLine; // Newline shorthand

        return $"NextDay{base.ToString()}{NL}Express Fee: {ExpressFee:C}";
    }
}
```

Package.cs:

```
// Program 1A
// CIS 200-01
// Fall 2020
// Due: 9/21/2020
// By: Andrew L. Wright (students use Grading ID)

// File: Package.cs
// The Package class is an abstract derived class from Parcel. It adds
// dimensions and weight.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public abstract class Package : Parcel
{
   private double _length; // Length of package in inches
   private double _width;  // Width of package in inches
   private double _height; // Height of package in inches
   private double _weight; // Weight of package in pounds

   // Precondition:  pLength > 0, pWidth > 0, pHeight > 0,
   //                pWeight > 0
   // Postcondition: The package is created with the specified values for
   //                origin address, destination address, length, width,
   //                height, and weight
   public Package(Address originAddress, Address destAddress,
      double pLength, double pWidth, double pHeight, double pWeight)
      : base(originAddress, destAddress)
   {
      Length = pLength;
      Width = pWidth;
      Height = pHeight;
      Weight = pWeight;
   }

   public double Length
   {
      // Precondition:  None
      // Postcondition: The package's length has been returned
      get
      {
```

```
        return _length;
    }

    // Precondition:  value > 0
    // Postcondition: The package's length has been set to the
    //                specified value
    set
    {
        if (value > 0)
            _length = value;
        else
            throw new ArgumentOutOfRangeException(nameof(Length), value,
                $"{nameof(Length)} must be > 0");
    }
}

public double Width
{
    // Precondition:  None
    // Postcondition: The package's width has been returned
    get
    {
        return _width;
    }

    // Precondition:  value > 0
    // Postcondition: The package's width has been set to the
    //                specified value
    set
    {
        if (value > 0)
            _width = value;
        else
            throw new ArgumentOutOfRangeException(nameof(Width), value,
                $"{nameof(Width)} must be > 0");
    }
}

public double Height
{
    // Precondition:  None
    // Postcondition: The package's height has been returned
    get
    {
```

```
        return _height;
    }

    // Precondition:  value > 0
    // Postcondition: The package's height has been set to the
    //                specified value
    set
    {
        if (value > 0)
            _height = value;
        else
            throw new ArgumentOutOfRangeException(nameof(Height), value,
                $"{nameof(Height)} must be > 0");
    }
}

public double Weight
{
    // Precondition:  None
    // Postcondition: The package's weight has been returned
    get
    {
        return _weight;
    }

    // Precondition:  value > 0
    // Postcondition: The package's weight has been set to the
    //                specified value
    set
    {
        if (value > 0)
            _weight = value;
        else
            throw new ArgumentOutOfRangeException(nameof(Weight), value,
                $"{nameof(Weight)} must be > 0");
    }
}

// Helper Property
protected double TotalDimension
{
    // Precondition:  None
    // Postcondition: The package's (Length + Width + Height) is returned
    get
```

```
        {
            return (Length + Width + Height);
        }
    }

    // Precondition:  None
    // Postcondition: A String with the package's data has been returned
    public override string ToString()
    {
        string NL = Environment.NewLine; // Newline shorthand

        return $"Package{NL}{base.ToString()}{NL}Length: {Length:N1}{NL}Width: {Width:N1}{NL}"
+
            $"Height: {Height:N1}{NL}Weight: {Weight:N1}";
    }
}
```

Parcel.cs:

```
//Yen Hsieh Hsu, 5398334
//CIS 200-75
//Due 12/3/22
//Application to list out parcels and sort them

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public abstract class Parcel : IComparable<Parcel>
{
    private Address _originAddress; // The origin address for the parcel
    private Address _destAddress;   // The destination address for the parcel

    // Precondition:  None
    // Postcondition: The parcel is created with the specified values for
    //                origin address and destination address
    public Parcel(Address originAddress, Address destAddress)
    {
        OriginAddress = originAddress;
        DestinationAddress = destAddress;
    }
```

```
public Address OriginAddress
{
   // Precondition:  None
   // Postcondition: The parcel's origin address has been returned
   get
   {
      return _originAddress;
   }

   // Precondition:  value must not be null
   // Postcondition: The parcel's origin address has been set to the
   //                specified value
   set
   {
      if (value == null)
      {
         throw new ArgumentOutOfRangeException($"{nameof(OriginAddress)}",
            value, $"{nameof(OriginAddress)} must not be null");
      }
      else
         _originAddress = value;
   }
}

public Address DestinationAddress
{
   // Precondition:  None
   // Postcondition: The parcel's destination address has been returned
   get
   {
      return _destAddress;
   }

   // Precondition:  value must not be null
   // Postcondition: The parcel's destination address has been set to the
   //                specified value
   set
   {
      if (value == null)
      {
         throw new ArgumentOutOfRangeException($"{nameof(DestinationAddress)}",
            value, $"{nameof(DestinationAddress)} must not be null");
      }
      else
```

```
            _destAddress = value;
        }
    }

    // Precondition:  None
    // Postcondition: The parcel's cost has been returned
    public abstract decimal CalcCost();

    // Precondition:  None
    // Postcondition: A String with the parcel's data has been returned
    public override String ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut

        return $"Origin Address:{NL}{OriginAddress}{NL}{NL}Destination Address:{NL}" +
            $"{DestinationAddress}{NL}Cost: {CalcCost():C}";
    }

    // Precondition:  None
    // Postcondition: returns 0 when both values are equal, negative when p1 < p2, positive when
p1 > p2
    public int CompareTo(Parcel p2)
    {
        decimal cost1; //cost of p1
        decimal cost2; //cost of p2

        if (p2 == null) //checking if p2 is null
            return 1;

        cost1 = this.CalcCost();
        cost2 = p2.CalcCost();

        return cost1.CompareTo(cost2);
    }
}
```

TestParcels.cs:

```
//Yen Hsieh Hsu, 5398334
//CIS 200-75
//Due 12/3/22
//Application to list out parcels and sort them

using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using static System.Console;

namespace Prog1
{
   class TestParcels
   {
      // Precondition:  None
      // Postcondition: Parcels have been created and displayed
      static void Main(string[] args)
      {
         bool VERBOSE = false;


         // Test Data - Magic Numbers OK
         Address a1 = new Address("  John Smith  ", "   123 Any St.   ", "  Apt. 45 ",
            "  Louisville   ", "  KY   ", 40202); // Test Address 1
         Address a2 = new Address("Jane Doe", "987 Main St.",
            "Beverly Hills", "CA", 90210); // Test Address 2
         Address a3 = new Address("James Kirk", "654 Roddenberry Way", "Suite 321",
            "El Paso", "TX", 79901); // Test Address 3
         Address a4 = new Address("John Crichton", "678 Pau Place", "Apt. 7",
            "Portland", "ME", 04101); // Test Address 4
         Address a5 = new Address(" Sally Smith ", "123 Transylvania St.", "Building 45 ",
            "Horror Town", "CA", 90666); // Test Address 5, for prog4
         Address a6 = new Address("Louie Watson", "987 E Market St.",
            "Conrad", "IL", 90207); // Test Address 6, for prog4
         Address a7 = new Address("James Morrison", "400 S Broadway St.", "Suite 231",
            "San Antonio", "TX", 78801); // Test Address 7, for prog4
         Address a8 = new Address("Luke Gernan", "55 Ronan Palace", "Apt. 17",
            "Portland", "OR", 77901); // Test Address 8, for prog4

         Letter letter1 = new Letter(a1, a2, 3.95M);                        // Letter test object
         GroundPackage gp1 = new GroundPackage(a3, a4, 14, 10, 5, 12.5);        // Ground test
object
         NextDayAirPackage ndap1 = new NextDayAirPackage(a1, a3, 25, 15, 15,    // Next Day
test object
            85, 8.50M);
         NextDayAirPackage ndap2 = new NextDayAirPackage(a1, a7, 20, 15, 20,    // Next Day
test object, added to test prog4
            80, 7.50M);
```

```
        TwoDayAirPackage tdap1 = new TwoDayAirPackage(a4, a1, 46.5, 39.5, 28.0, // Two
Day test object
            80.5, TwoDayAirPackage.Delivery.Saver);
        Letter letter2 = new Letter(a4, a5, 3.95M);                    // Letter test object, added to
test prog4
        GroundPackage gp2 = new GroundPackage(a2, a8, 15, 15, 7, 10.5);      // Ground test
object, added to test prog4
        NextDayAirPackage ndap3 = new NextDayAirPackage(a1, a6, 23, 23, 15,    // Next Day
test object, added to test prog4
            85, 9.50M);
        NextDayAirPackage ndap4 = new NextDayAirPackage(a1, a4, 27, 25, 20,    // Next Day
test object, added to test prog4
            68, 6.50M);
        TwoDayAirPackage tdap2 = new TwoDayAirPackage(a5, a2, 48, 37, 28, // Two Day test
object, added to test prog4
            95.5, TwoDayAirPackage.Delivery.Early);

        List<Parcel> parcels;      // List of test parcels

        parcels = new List<Parcel>();

        parcels.Add(letter1); // Populate list
        parcels.Add(gp1);
        parcels.Add(ndap1);
        parcels.Add(ndap2); //added to test prog4
        parcels.Add(tdap1);
        parcels.Add(letter2); // added to test prog4
        parcels.Add(gp2); //added to test prog4
        parcels.Add(ndap3); //added to test prog4
        parcels.Add(ndap4); //added to test prog4
        parcels.Add(tdap2); //added to test prog4

        WriteLine("Original List:");
        WriteLine("====================");
        foreach (Parcel p in parcels)
        {
            WriteLine(p);
            WriteLine("====================");
        }
        Pause();

        parcels.Sort();
        WriteLine("Sorted in Natural Order (Ascending by Cost):");
        WriteLine("====================");
```

```
    foreach (Parcel p in parcels)
    {
        if (VERBOSE)
            WriteLine(p);
        else
            WriteLine($"{p.CalcCost(),8:C}");
    }
    Pause();

    parcels.Sort(new DescendingByDestZipComparer());
    WriteLine("Sorted using first Comparer (Descending by Destination Zip):");
    WriteLine("====================");
    foreach (Parcel p in parcels)
    {
        if (VERBOSE)
            WriteLine(p);
        else
            WriteLine($"{p.DestinationAddress.Zip:D5}");
    }
    Pause();

    parcels.Sort(new TypeAndCostComparer());
    WriteLine("Sorted using EC Comparer (Ascending by Type then Descensing by Cost):");
    WriteLine("====================");
    foreach (Parcel p in parcels)
    {
        if (VERBOSE)
            WriteLine(p);
        else
            WriteLine($"{p.GetType().ToString(),-17} {p.CalcCost(),8:C}");
    }
    Pause();

}

// Precondition:  None
// Postcondition: Pauses program execution until user presses Enter and
//                then clears the screen
public static void Pause()
{
    WriteLine("Press Enter to Continue...");
    ReadLine();

    Console.Clear(); // Clear screen
```

```
      }
   }
}
```

TwoDayAirPackage.cs

```
// Program 1A
// CIS 200-01
// Fall 2020
// Due: 9/21/2020
// By: Andrew L. Wright (students use Grading ID)

// File: TwoDayAirPackage.cs
// The TwoDayAirPackage class is a concrete derived class from AirPackage. It adds
// a delivery type.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class TwoDayAirPackage : AirPackage
{
   public enum Delivery { Early, Saver } // Delivery types

   private Delivery _deliveryType; // TDAP's delivery type (Early or Saver)

   // Precondition:  pLength > 0, pWidth > 0, pHeight > 0,
   //                pWeight > 0
   // Postcondition: The two day air package is created with the specified values for
   //                origin address, destination address, length, width,
   //                height, weight, and delivery type
   public TwoDayAirPackage(Address originAddress, Address destAddress,
      double pLength, double pWidth, double pHeight, double pWeight, Delivery delType)
      : base(originAddress, destAddress, pLength, pWidth, pHeight, pWeight)
   {
      DeliveryType = delType;
   }

   public Delivery DeliveryType
   {
      // Precondition:  None
      // Postcondition: The two day air package's delivery type has been returned
```

```
    get
    {
        return _deliveryType;
    }

    // Precondition:  value must be Early or Saver
    // Postcondition: The two day air package's delivery type has been set to the
    //                specified value
    set
    {
        if (Enum.IsDefined(typeof(Delivery), value))
            _deliveryType = value;
        else
            throw new ArgumentOutOfRangeException(nameof(DeliveryType), value,
                $"{nameof(DeliveryType)} must be {nameof(Delivery.Early)} " +
                $"or {nameof(Delivery.Saver)}");
    }
}

// Precondition:  None
// Postcondition: The two day air package's cost has been returned
public override decimal CalcCost()
{
    const double DIM_FACTOR = .18;        // Dimension coefficient in cost equation
    const double WEIGHT_FACTOR = .2;      // Weight coefficient in cost equation
    const decimal DISCOUNT_FACTOR = .15M; // Discount factor in cost equation

    decimal cost; // Running total of cost of package

    cost = (decimal)(DIM_FACTOR * TotalDimension + WEIGHT_FACTOR * Weight);

    if (DeliveryType == Delivery.Saver)
        cost *= (1-DISCOUNT_FACTOR);

    return cost;
}

// Precondition:  None
// Postcondition: A String with the two day air package's data has been returned
public override string ToString()
{
    string NL = Environment.NewLine; // Newline shorthand

    return $"TwoDay{base.ToString()}{NL}Delivery Type: {DeliveryType}";
```

```
    }
}
```

TypeAndCostComparer.cs:

```
//Yen Hsieh Hsu, 5398334
//CIS 200-75
//Due 12/3/22
//Application to list out parcels and sort them

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

public class TypeAndCostComparer : Comparer<Parcel>
{
    public override int Compare(Parcel p1, Parcel p2)
    {
        string type1;
        string type2;

        if (p1 == null && p2 == null) // checking whether both are null
            return 0;
        if (p1 == null) // checking whether p1 is null
            return -1;
        if (p2 == null) //checking whether p2 is null
            return 1;

        type1 = p1.GetType().ToString();
        type2 = p2.GetType().ToString();

        //for different types
        if (type1 != type2)
            return type1.CompareTo(type2);

        //for same types
        else
            return (-1)*p1.CompareTo(p2); //descending order
    }
}
```