```
//############################################################################
//Letter.h
//############################################################################
#pragma once
//***************************************************************************
// File name:   Letter.h
// Author:      Taylor Isaac
// Date:        5/2/2021
// Class:       CS 250
// Assignment:  046Polymorphism_Classes
// Purpose:     Demonstrate the Letter class
//***************************************************************************
#include <string>
#include <iostream>
#include "Parcels.h"

using namespace std;

class Letter : public Parcels {
  public:
    Letter ();
    virtual bool read (istream&);
    virtual void print (ostream&);

    virtual double getCost ();
    virtual int getDaysForDelivery () ;
    virtual double getInsuranceExpense (double) const;
    virtual double getRushExpense (double) const;

private:
  double mInsuranceFlatRate;
  double mRushCostMultiplier;
};


//############################################################################
//Overnight.h
//############################################################################
#pragma once
//***************************************************************************
// File name:   Overnight.h
// Author:      Taylor Isaac
// Date:        5/2/2021
// Class:       CS 250
// Assignment:  06Polymorphism_Classes
// Purpose:     Demonstrate the Overnight class
//***************************************************************************
#include <string>
#include <iostream>
```

```
#include "Parcels.h"
using namespace std;

class Overnight : public Parcels {
  public:
    Overnight ();
    int getVol () const { return mVol; }

    virtual bool read (istream&);
    virtual void print (ostream&);

    virtual double getCost ();
    virtual int getDaysForDelivery ();
    virtual double getInsuranceExpense (double) const;
    virtual double getRushExpense (double) const;

  private:
    int mVol;
    double mInsuranceCostMultiplier;
    double mRushCostMultiplier;
  };


//###########################################################################
//Parcels.h
//###########################################################################
#pragma once
//************************************************************************
// File name:   Parcels.h
// Author:      Taylor Isaac
// Date:        5/2/2021
// Class:       CS 250
// Assignment:  06Polymorphism_Classes
// Purpose:     Demonstrate the Parcels class
//************************************************************************

#include <string>
#include <iostream>

using namespace std;

class Parcels {
public:
  Parcels();
  virtual bool read (istream&);
  virtual void print (ostream&);
  static bool getValidUserTID (int index, Parcels* apcArrayOfParcels[],
                              int userChoice);
  bool getInsuranceTruth ();
  bool getRushTruth ();
  int getWeightOz () const { return mWeightOz; }
  int getDistance () const { return mDistanceToTravel; }

  virtual double getCost () = 0;
  virtual int getDaysForDelivery () = 0;
  virtual double getInsuranceExpense (double) const = 0;
  virtual double getRushExpense (double) const = 0;

  void addInsurance ();
  void addRush ();

private:
  int mTrackingNumb;
  string mToAddress;
  string mFromAddress;
  int mWeightOz;
  int mDistanceToTravel;
  int mMilesPerDayCanTravel;
  int mDayMinimumOfTravel;
  bool mbInsured;
  bool mbRush;
};


//###############################################################################
//Postcard.h
//###############################################################################
#pragma once
//*****************************************************************************
// File name:   Postcard.h
// Author:      Taylor Isaac
// Date:        5/2/2021
// Class:       CS 250
// Assignment:  06Polymorphism_Classes
// Purpose:     Demonstrate the Postcard class
//*****************************************************************************

#include <string>
#include <iostream>
#include "Parcels.h"

using namespace std;

class Postcard : public Parcels {
  public:
    Postcard ();
    virtual bool read (istream&);
    virtual void print (ostream&);
```

```cpp

    virtual double getCost ();
    virtual int getDaysForDelivery ();
    virtual double getInsuranceExpense (double) const;
    virtual double getRushExpense (double) const;

  private:
    string mMessage;
    double mInsuranceFlatRate;
    double mRushCost;
  };


//##############################################################################
//Letter.cpp
//##############################################################################
//*****************************************************************************
// File name:  Letter.cpp
// Author:     Taylor Isaac
// Date:       5/2/2021
// Class:      CS 250
// Assignment: 06Polymorphism_Classes
// Purpose:    Demonstrate Letter and its pertinence to inheritance
//*****************************************************************************

#include "Parcels.h"
#include "Letter.h"
#include <iostream>
#include <string>
#include <iomanip>

//*****************************************************************************
// Constructor: Letter
//
// Description: Provides initialization for the appropiate data member
//              variables of a letter--this should have pertinence to
//              inheritance since this class is a derived class
//
// Parameters:  None
//
// Returned:    none
//*****************************************************************************

Letter::Letter() : Parcels(), mInsuranceFlatRate(0.45),
                   mRushCostMultiplier(1.1) {
}

//*****************************************************************************
// Function:    read
```

```
//
// Description: utilizes the properties of inheritance, (from its parent
//              function) reading in data
//
// Parameters:  rcIn - designated input option
//
// Returned:    determines whether or not data was read in
//***************************************************************************

bool Letter::read(istream& rcIn) {
  bool bTheTruth = false;
  if (Parcels::read(rcIn)) {
    bTheTruth = true;
  }
  return bTheTruth;
}

//***************************************************************************
// Function:    print
//
// Description: prints an object's correct associated information
//
// Parameters:  rcOut - one may specift whether to the console or to an
//                        output file
//
// Returned:    none
//***************************************************************************

void Letter::print(ostream& rcOut) {
  Parcels::print(rcOut);
  rcOut << "\n";
}

//***************************************************************************
// Function:    getCost
//
// Description: gets the total running costs if particular attributes are
//              valid
//
// Parameters:  none
//
// Returned:    the total running cost
//***************************************************************************

double Letter::getCost() {
  double runningCost = (getWeightOz() * mInsuranceFlatRate);
  if (Parcels::getInsuranceTruth()) {
    runningCost += mInsuranceFlatRate;
  }
```

```cpp

  if (Parcels::getRushTruth()) {
    runningCost *= mRushCostMultiplier;
  }
  cout << fixed << setprecision(2);
  return runningCost;
}

//***************************************************************************
// Function:    getDaysForDelivery
//
// Description: gets the total running number of days if particular
//              attributes are valid
//
// Parameters:  none
//
// Returned:    the total running number of days required for delivery
//***************************************************************************

int Letter::getDaysForDelivery() {
  const int MILES_PER_DAY_CAN_TRAVEL = 100;
  const int ZERO_THRESHOLD = 0;
  const int ONE_DAY_THRESHOLD = 1;
  int milesToTravel = Parcels::getDistance();
  int daysForDelivery = 0;

  if (MILES_PER_DAY_CAN_TRAVEL >= milesToTravel) {
    daysForDelivery = 1;
  }
  else {
    while (milesToTravel > ZERO_THRESHOLD) {
      (milesToTravel -= MILES_PER_DAY_CAN_TRAVEL);
      daysForDelivery++;
    }
  }
  if (Parcels::getRushTruth()) {
    if (daysForDelivery > ONE_DAY_THRESHOLD) {
      daysForDelivery -= 1;
    }
  }
  return daysForDelivery;
}

//***************************************************************************
// Function:    getInsuranceExpense
//
// Description: returns the calculated insurance expense for the unique
//              parcel
//
```

```cpp
// Parameters:  none
//
// Returned:    the insurance's flat rate for the unique parcel
//***************************************************************************

double Letter::getInsuranceExpense(double currCost) const {
  return mInsuranceFlatRate;
}

//***************************************************************************
// Function:    getRushExpense
//
// Description: returns the calculated rush expense for the unique
//              parcel
//
// Parameters:  currCost - the current cost of the unique type of parcel to
//
// Returned:    the rushing expense
//***************************************************************************

double Letter::getRushExpense(double currCost) const {
  double actualRushExpense = 0;
  actualRushExpense = (currCost * mRushCostMultiplier);
  return actualRushExpense;
}


//################################################################################
//Overnight.cpp
//################################################################################
//***************************************************************************
// File name:  Parcels.cpp
// Author:     Taylor Isaac
// Date:       5/2/2021
// Class:      CS 250
// Assignment: 06Polymorphism_Classes
// Purpose:    Demonstrate Inheritance
//***************************************************************************

#include "Parcels.h"
#include "Overnight.h"
#include <iostream>
#include <string>
#include <iomanip>

//***************************************************************************
// Constructor: Overnight
//
// Description: Provides initialization for the appropiate data member
//               variables of an overnight package. This should set up
//               inheritance for this class's derived classes
//
// Parameters:   None
//
// Returned:     none
//***************************************************************************

Overnight::Overnight() : Parcels(), mVol(-1), mInsuranceCostMultiplier(0.25),
                         mRushCostMultiplier(0.75) {

}

//***************************************************************************
// Function:     read
//
// Description:  using inherited functionality from its parent class to read
//               in more unique data tailored to its own derived class
//
// Parameters:   rcIn - designated input option
//
// Returned:     the truth of whether or not data was successful in reading in
//***************************************************************************

bool Overnight::read(istream& rcIn) {
  bool bTheTruth = false;
  Parcels::read(rcIn);
  if (rcIn >> mVol) {
    bTheTruth = true;
  }
  return bTheTruth;
}

//***************************************************************************
// Function:     print
//
// Description:  prints an object's correct associated information
//
// Parameters:   rcOut - one may specift whether to the console or to an
//                       output file
//
// Returned:     none
//***************************************************************************

void Overnight::print(ostream& rcOut) {
  Parcels::print(rcOut);
  rcOut << "\tOVERNIGHT!\n";
}

```

```cpp
//***************************************************************************
// Function:    getCost
//
// Description: gets the total running costs if particular attributes are
//              valid
//
// Parameters:  none
//
// Returned:    the total running cost
//***************************************************************************

double Overnight::getCost() {
  const double INSURANCE_COST_MULTIPLIER = 1.25;
  const double RUSH_MULTIPLIER = 1.75;
  const int UPPER_BOUND_COST = 100;
  double runningCost = 0;
  if (getVol() > UPPER_BOUND_COST) {
    runningCost = 20;
  }
  if (getVol() <= UPPER_BOUND_COST) {
    runningCost = 12;
  }
  if (Parcels::getInsuranceTruth()) {
    runningCost *= INSURANCE_COST_MULTIPLIER;
  }
  if (Parcels::getRushTruth()) {
    runningCost *= RUSH_MULTIPLIER;
  }
  cout << fixed << setprecision(2);
  return runningCost;
}

//***************************************************************************
// Function:    getDaysForDelivery
//
// Description: gets the total running number of days if particular
//              attributes are valid
//
// Parameters:  none
//
// Returned:    the total running number of days required for delivery
//***************************************************************************

int Overnight::getDaysForDelivery() {
  const int MAX_MILES_PER_DAY_CAN_TRAVEL = 1000;
  int milesToTravel = Parcels::getDistance();
  int daysForDelivery = 0;

  if (MAX_MILES_PER_DAY_CAN_TRAVEL >= milesToTravel) {
    daysForDelivery = 1;
  }
  else {
      daysForDelivery += 2;
   }
  if (Parcels::getRushTruth()) {
    daysForDelivery = 1;
  }
  return daysForDelivery;
}

//***************************************************************************
// Function:    getInsuranceExpense
//
// Description: returns the calculated insurance expense for the unique
//              parcel
//
// Parameters:  none
//
// Returned:    the insurance's overall rate for the unique parcel
//***************************************************************************

double Overnight::getInsuranceExpense(double currCost) const {
  double actualInsuranceExpense = 0;
  actualInsuranceExpense = (currCost * mInsuranceCostMultiplier);
  return actualInsuranceExpense;
}

//***************************************************************************
// Function:    getRushExpense
//
// Description: returns the calculated rush expense for the unique
//              parcel
//
// Parameters:  currCost - the current cost of the unique type of parcel to
//
// Returned:    the rushing expense
//***************************************************************************

double Overnight::getRushExpense(double currCost) const {
  double actualRushExpense = 0;
  actualRushExpense = (currCost * mRushCostMultiplier);
  return actualRushExpense;
}


//############################################################################
//Parcels.cpp
//############################################################################
```

```
//***************************************************************************
// File name:  Parcels.cpp
// Author:     Taylor Isaac
// Date:       5/2/2021
// Class:      CS 250
// Assignment: 06Polymorphism_Classes
// Purpose:    Demonstrate Inheritance using a Parcel
//***************************************************************************

#include "Parcels.h"
#include <iostream>
#include <string>

//***************************************************************************
// Constructor: Parcels
//
// Description: Provides initialization for the appropiate data member
//              variables of a parcel. This should set up inheritance for
//              this class's derived classes
//
// Parameters:  None
//
// Returned:    none
//***************************************************************************

Parcels::Parcels() {
  mTrackingNumb = -1;
  mToAddress = "";
  mFromAddress ="";
  mWeightOz = -1;
  mDistanceToTravel = -1;
  mMilesPerDayCanTravel = -1;
  mDayMinimumOfTravel = -1;
  mbInsured = false;
  mbRush = false;
}

//***************************************************************************
// Function:    read
//
// Description: reads in specified data (the derived classes will use this
//              exact function, but add more special things to read in
//
// Parameters:  rcIn - designated input option
//
// Returned:    determination of whether or not data was read in
//***************************************************************************

bool Parcels::read(istream& rcIn) {
```

```cpp
  bool bTheTruth = false;
  if (rcIn >> mTrackingNumb >> mToAddress >> mFromAddress >> mWeightOz
          >> mDistanceToTravel) {
    bTheTruth = true;
  }

  return bTheTruth;
}

//****************************************************************************
// Function:    print
//
// Description: prints an object's correct associated information
//
// Parameters:  ostream - one may specift whether to the console or to an
//                        output file
//
// Returned:    none
//****************************************************************************

void Parcels::print(ostream& rcOut) {
  rcOut << "TID: " << mTrackingNumb << "\tFrom: " << mFromAddress
        << "\tTo: " << mToAddress;
  if (mbInsured) {
    rcOut << "\tINSURED\t";
  }
  if (mbRush) {
    rcOut << "\tRUSH";
  }

}

//****************************************************************************
// Function:    getUserTID
//
// Description: gets the user input's desired TID. This function checks
//              to see if any of the TID's in the array of pointers matches
//              up with what the user enters in
//
// Parameters:  index             - actual size of array of pointers
//
//              apcArrayOfParcels - the array of pointers passed in
//
//              userTID           - passes in the user's inputted initial TID
//
// Returned:    returns a boolean variable if valid TID
//****************************************************************************

bool Parcels::getValidUserTID(int index, Parcels* apcArrayOfParcels[],
                                int userTID) {
  bool validUserTID = false;
  for (int start = 0; start < index; start++) {
    if (apcArrayOfParcels[start] != nullptr &&
        apcArrayOfParcels[start]->mTrackingNumb == userTID) {
      validUserTID = true;
    }
  }
  return validUserTID;
  }

//***************************************************************************
// Function:    getInsuranceTruth
//
// Description: gets the returned value of the current state of being insured
//              or not
//
// Parameters:  none
//
// Returned:    returns a boolean variable of insurance
//***************************************************************************

  bool Parcels::getInsuranceTruth() {
  return mbInsured;
}

//***************************************************************************
// Function:    getRushTruth
//
// Description: gets the returned value of whether or not there is rushed
//              delivery
//
// Parameters:  none
//
// Returned:    returns a boolean variable of whether or not its rushed
//***************************************************************************

bool Parcels::getRushTruth() {
  return mbRush;
}

//***************************************************************************
// Function:    addInsurance
//
// Description: sets true the state of being insured for the unique type of
//              parcel
//
// Parameters:  none
//
```

```cpp
// Returned:    none
//***************************************************************************

void Parcels::addInsurance() {
  mbInsured = true;
}

//***************************************************************************
// Function:    addRush
//
// Description: sets true the unique parcel's state of being rushed
//
// Parameters:  none
//
// Returned:    none
//***************************************************************************

void Parcels::addRush() {
  mbRush = true;
}


//###########################################################################
//Postcard.cpp
//###########################################################################
//***************************************************************************
// File name:  Postcard.cpp
// Author:     Taylor Isaac
// Date:       5/2/2021
// Class:      CS 250
// Assignment: 06Polymorphism_Classes
// Purpose:    Demonstrate Inheritance
//***************************************************************************

#include "Parcels.h"
#include "Letter.h"
#include <iostream>
#include "Postcard.h"
#include <string>
#include <iomanip>

//***************************************************************************
// Constructor: Postcard
//
// Description: Provides initialization for the appropiate data member
//              variables of a postcard. This should utilize inheritance for
//              this class's derived classes
//
// Parameters:  None
```

```
//
// Returned:    none
//***************************************************************************

Postcard::Postcard() : Parcels(), mMessage(""), mInsuranceFlatRate(0.15),
                       mRushCost(0.25) {
}

//***************************************************************************
// Function:    read
//
// Description: using inherited functionality from its parent class to read
//              in more unique data tailored to its own derived class
//
// Parameters:  rcIn - designated input option
//
// Returned:    determination of whether or not data was read in
//***************************************************************************

bool Postcard::read(istream& rcIn) {
  bool bTheTruth = false;
  Parcels::read(rcIn);
  if (rcIn >> mMessage) {
    bTheTruth = true;
  }
  return bTheTruth;
}

//***************************************************************************
// Function:    print
//
// Description: prints an object's correct associated information
//
// Parameters:  rcOut - one may specift whether to the console or to an
//                      output file
//
// Returned:    none
//***************************************************************************

void Postcard::print(ostream& rcOut) {
  Parcels::print(rcOut);
  rcOut << "\t" << mMessage << "\n";
}

//***************************************************************************
// Function:    getCost
//
// Description: gets the total running costs if particular attributes are
//              valid
```

```cpp
//
// Parameters:  none
//
// Returned:    the total running cost for the postcard
//***************************************************************************

double Postcard::getCost() {
  double runningCost = 0.15;

  if (Parcels::getRushTruth()) {
    runningCost += mRushCost;
  }
  if (Parcels::getInsuranceTruth()) {
    runningCost += mInsuranceFlatRate;
  }
  cout << fixed << setprecision(2);
  return runningCost;
}

//***************************************************************************
// Function:    getDaysForDelivery
//
// Description: gets the total running number of days if particular
//              attributes are valid
//
// Parameters:  none
//
// Returned:    the total running number of days required for delivery of a
//              postcard
//***************************************************************************

int Postcard::getDaysForDelivery() {
  const int MILES_PER_DAY_CAN_TRAVEL = 10;
  const int ZERO_THRESHOLD = 0;
  const int ONE_DAY_THRESHOLD = 1;
  int milesToTravel = Parcels::getDistance();
  int daysForDelivery = 0;

  if (MILES_PER_DAY_CAN_TRAVEL >= milesToTravel) {
    daysForDelivery = 1;
  }
  else {
    while (milesToTravel > ZERO_THRESHOLD) {
      (milesToTravel -= MILES_PER_DAY_CAN_TRAVEL);
      daysForDelivery++;
    }
  }
  if (Parcels::getRushTruth()) {
    if (daysForDelivery > ONE_DAY_THRESHOLD) {
      daysForDelivery -= 1;
    }
  }
  return daysForDelivery;
}

//***************************************************************************
// Function:    getInsuranceExpense
//
// Description: returns the calculated insurance expense for the unique
//              parcel
//
// Parameters:  none
//
// Returned:    the insurance's flat rate for the unique parcel
//***************************************************************************

double Postcard::getInsuranceExpense(double currCost) const {
  return mInsuranceFlatRate;
}

//***************************************************************************
// Function:    getRushExpense
//
// Description: returns the calculated rush expense for the unique
//              parcel
//
// Parameters:  currCost - the current cost of the unique type of parcel to
//
// Returned:    the rushing expense
//***************************************************************************

double Postcard::getRushExpense(double currCost) const {
  return mRushCost;
}


//############################################################################
//main.cpp
//############################################################################
//***************************************************************************
// File name:   main.cpp
// Author:      Taylor Isaac
// Date:        5/2/2021
// Class:       CS 250
// Assignment:  06Polymorphism
// Purpose:     Demonstrate the parcels class and its derived classes
//***************************************************************************

```

```
#include "Parcels.h"
#include "Letter.h"
#include "Overnight.h"
#include "Postcard.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <vld.h>
const int OPTION_ONE = 1;
const int OPTION_TWO = 2;
const int OPTION_THREE = 3;
const int OPTION_FOUR = 4;
const int OPTION_FIVE = 5;
const string INPUT_FILE = "parcels.txt";
int printOptionMenuAndGetChoice();
void openFileForRead(ifstream& rcInfile);
void closeFileForRead(ifstream& rcInfile);
void printAllParcels(int index, Parcels* apcParcelsObjs[]);
int obtainInitialTID();
using namespace std;

//***************************************************************************
// Function:    main
//
// Description: contains the necessary components, variables, and functions
//              to execute tasks and operate on parcel related activity
//
// Parameters:  None
//
// Returned:    EXIT_SUCCESS
//***************************************************************************

int main() {
  const int MAX_NUMB_ARRAY = 25;
  const string PROMPT_CHOICE = "Choice> ";
  const char POSTCARD_SYMBOL = 'P';
  const char LETTER_SYMBOL = 'L';
  const char OVERNIGHT_SYMBOL = 'O';
  const int NO_PARCELS = 0;
  char parcelEat = '.';
  int index = 0;
  int initialID = -1;
  int userChoice = -2;
  bool bIsValidTID = false;
  double eatCost = 0;
  Parcels* apcParcelsObjs[MAX_NUMB_ARRAY] = { nullptr };
  ifstream cInFile;

  openFileForRead(cInFile);
  cout << "Mail Simulator!\n";

  while (cInFile >> parcelEat) {

    switch (parcelEat) {
    case POSTCARD_SYMBOL: apcParcelsObjs[index] = new Postcard();
      apcParcelsObjs[index]->read(cInFile);
      ++index;
      break;
    case LETTER_SYMBOL:  apcParcelsObjs[index] = new Letter();
      apcParcelsObjs[index]->read(cInFile);
      ++index;
      break;
    case OVERNIGHT_SYMBOL: apcParcelsObjs[index] = new Overnight();
      apcParcelsObjs[index]->read(cInFile);
      ++index;
      break;
    default: break;
    }
  }
  if (NO_PARCELS == index) {
    cout << "File is empty.\n";
    exit(EXIT_FAILURE);
  }
  while (OPTION_FIVE != userChoice) {
    userChoice = printOptionMenuAndGetChoice();
    if (OPTION_ONE == userChoice) {
      printAllParcels(index, apcParcelsObjs);
    }

    else if (OPTION_FOUR == userChoice) {

      initialID = obtainInitialTID();
      if (apcParcelsObjs[initialID - 1] != nullptr) {
        bIsValidTID = Parcels::getValidUserTID(index, apcParcelsObjs,
          initialID);
        if (bIsValidTID) {
          cout << "Delivered!\n";
          cout << apcParcelsObjs[initialID - 1]->getDaysForDelivery();
          cout << " Day, " << "$"
            << apcParcelsObjs[initialID - 1]->getCost();
          cout << "\n";
          apcParcelsObjs[initialID - 1]->print(cout);
          delete apcParcelsObjs[initialID - 1];
          apcParcelsObjs[initialID - 1] = { nullptr };
        }
      }
    }
    else if (OPTION_TWO == userChoice) {
      initialID = obtainInitialTID();
      if (apcParcelsObjs[initialID - 1] != nullptr) {
        bIsValidTID = Parcels::getValidUserTID(index, apcParcelsObjs,
          initialID);
        if (bIsValidTID) {
          cout << "Added Insurance for $";
          eatCost = (apcParcelsObjs[initialID - 1]->getCost());

          cout << apcParcelsObjs[initialID - 1]->
            getInsuranceExpense(eatCost);

          cout << "\n";
          apcParcelsObjs[initialID - 1]->addInsurance();
          apcParcelsObjs[initialID - 1]->print(cout);
        }
      }
    }
    else if (OPTION_THREE == userChoice) {
      initialID = obtainInitialTID();
      if (apcParcelsObjs[initialID - 1] != nullptr) {
        bIsValidTID = Parcels::getValidUserTID(index, apcParcelsObjs,
          initialID);
        if (bIsValidTID) {
          cout << "Added Rush for $";
          eatCost = (apcParcelsObjs[initialID - 1]->getCost());
          cout << apcParcelsObjs[initialID - 1]->getRushExpense(eatCost);
          cout << "\n";
          apcParcelsObjs[initialID - 1]->addRush();
          apcParcelsObjs[initialID - 1]->print(cout);
        }
      }
    }
  }
  for (int start = 0; start < index; start++) {
    delete (apcParcelsObjs[start]);
  }
  closeFileForRead(cInFile);
  return EXIT_SUCCESS;
}

//***************************************************************************
// Function:    printOptionMenuAndGetChoice
//
// Description: Outputs desired description or criterium with the option of
//              allowing the user to input a choice
//
// Parameters:  None
//
```

```
// Returned:    the returned valid choice as a number displayed from menu
//***************************************************************************

int printOptionMenuAndGetChoice() {
  int userChoice = -1;
  cout << "\n1. Print All\n2. Add Insurance\n3. Add Rush\n4. Deliver\n5. " ;
  cout << "Quit\n\n";
  do {
    cout << "Choice> ";
    cin >> userChoice;
  } while (!(OPTION_ONE == userChoice || OPTION_TWO == userChoice ||
    OPTION_THREE == userChoice || OPTION_FOUR == userChoice ||
    OPTION_FIVE == userChoice));
  return userChoice;
}

//***************************************************************************
// Function:    openFileForRead
//
// Description: opens file, checking for proper opening
//
// Parameters:  rcInfile - specified input file
//
// Returned:    None
//***************************************************************************

void openFileForRead(ifstream& rcInfile) {
  rcInfile.open(INPUT_FILE);
  if (!rcInfile.is_open()) {
    cout << "Error opening file.\n";
    exit(EXIT_FAILURE);
  }
}

//***************************************************************************
// Function:    closeFileForRead
//
// Description: closes a file opened for reading
//
// Parameters:  rInfile - the object used to close the file to be read
//
// Returned:    None
//***************************************************************************

void closeFileForRead(ifstream& rcInfile) {
  rcInfile.close();
}

//***************************************************************************
```

```
// Function:    printAllParcels
//
// Description: when user chooses option 1, it prints all associated parcels
//
// Parameters:  index - the correct amount of positions associated with the
//                      array of pointers
//
//              *apcArrayOfParcels[] - allows an array of pointers of class
//                                     Parcels to be passed in
//
// Returned:    None
//***************************************************************************

void printAllParcels(int index, Parcels* apcArrayOfParcels[]) {
  cout << "\n";
  for (int start = 0; start < index; start++) {
    if (apcArrayOfParcels[start] != nullptr) {
      apcArrayOfParcels[start]->print(cout);
    }
  }
}

//***************************************************************************
// Function:    obtainInitialTID
//
// Description: gets initial TID from user. We do not know if it's valid yet
//
// Parameters:  None
//
// Returned:    the user's inputted initial tracking ID number
//***************************************************************************

int obtainInitialTID() {
  int initialTID = -1;
  cout << "\nTID> ";
  cin >> initialTID;
  return initialTID;
}
```