American University of Science and Technology

Department of Computer Science

CSI 250 – Computer Programming II

Section A-MWF 10:00-10:50

Sami Farhat ID #:12100013

Ahmad Bazzi ID #:12100563

Computer Programming II Final Project

Submitted to:

Mr. Elie Nasr

Date: Friday the 1st of June 2012

INDEX:

Exercise #	Page #
10.10	3
12.9	7
12.10	11
13.15	15
13.16	21
17.6	25
17.7	28
17.8	32
17.9	34
17.12	37

Exercise 10.10:

(Card Shuffling and Dealing) Create a program to shuffle and deal a deck of cards. The program should consist of class Card, class DeckOfCards and a driver program. Class Card should provide:

a) Data members face and suit of type int.

b) A constructor that receives two ints representing the face and suit and uses them to initialize the data members.

c) Two static arrays of strings representing the faces and suits.

d) A toString function that returns the Card as a string in the form “face of suit.” You can use the + operator to concatenate strings.

Class DeckOfCards should contain:

a) A vector of Cards named deck to store the Cards.

b) An integer currentCard representing the next card to deal.

c) A default constructor that initializes the Cards in the deck. The constructor should use vector function push_back to add each Card to the end of the vector after the Card is created and initialized. This should be done for each of the 52 Cards in the deck.

d) A shuffle function that shuffles the Cards in the deck. The shuffle algorithm should iterate through the vector of Cards. For each Card, randomly select another Card in the deck and swap the two Cards.

e) A dealCard function that returns the next Card object from the deck.

f) A moreCards function that returns a bool value indicating whether there are more Cards to deal.

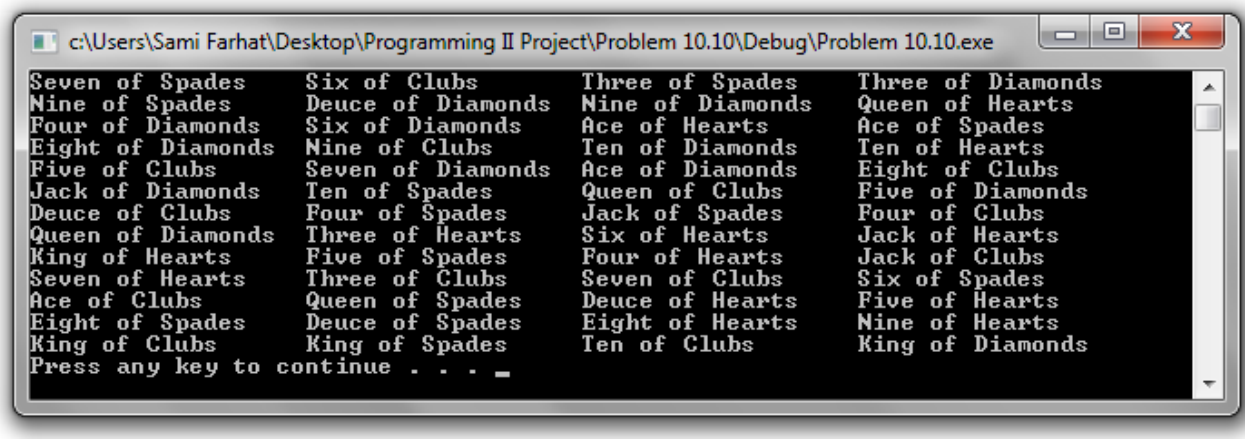
The driver program should create a DeckOfCards object, shuffle the cards, then deal the 52 cards.

```
#include <string>
#include <vector>
#include <ctime>
#include <cstdlib>
#include <iostream>
#include <iomanip>
using namespace std;
```

```
class Card
{
public:
   static const int totalFaces = 13; // total number of faces
   static const int totalSuits = 4; // total number of suits

   Card( int cardFace, int cardSuit ); // initialize face and suit
   string toString() const; // returns a string representation of a Card

   // get the card's face
   int getFace() const
   {
      return face;
   } // end function getFace

   // get the card's suit
   int getSuit() const
   {
      return suit;
   } // end function getSuit
private:
   int face;
   int suit;

   static const string faceNames[ totalFaces ];
   static const string suitNames[ totalSuits ];
}; // end class Card

Card::Card( int cardFace, int cardSuit )
{
   face = ( cardFace >= 0 && cardFace < totalFaces ) ? cardFace : 0;
   suit = ( cardSuit >= 0 && cardSuit < totalSuits ) ? cardSuit : 0;
} // end Card constructor

// returns a string representation of a Card
string Card::toString() const
{
   // strings can be concatenated with +
   return faceNames[ face ] + " of " + suitNames[ suit ];
} // end function toString

// contents of arrays to convert index into name
const string Card::faceNames[ totalFaces ] =
   { "Ace", "Deuce", "Three", "Four", "Five", "Six",
     "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
```

```
const string Card::suitNames[ totalSuits ] =
   { "Hearts", "Diamonds", "Clubs", "Spades" };

// DeckOfCards class definition
class DeckOfCards
{
public:
   DeckOfCards(); // constructor initializes deck
   void shuffle(); // shuffles cards in deck
   Card dealCard(); // deals cards in deck
   bool moreCards() const; // are there any more cards left
private:
   vector< Card > deck; // represents deck of cards
   unsigned currentCard; // index of next card to be dealt
}; // end class DeckOfCards
DeckOfCards::DeckOfCards()
{
   currentCard = 0; // set currentCard so first Card dealt is deck[ 0 ]

   // populate deck with Card objects
   for ( int i = 0; i < Card::totalFaces * Card::totalSuits; ++i )
   {
      Card card( i % Card::totalFaces, i / Card::totalFaces );
      deck.push_back( card ); // adds copy of card to the end of the deck
   } // end for

   srand( time( 0 ) ); // seed random number generator
} // end DeckOfCards default constructor

// shuffle cards in deck
void DeckOfCards::shuffle()
{
   // after shuffling, dealing should start at deck[ 0 ] again
   currentCard = 0; // reinitialize currentCard

   // for each Card, pick another random Card and swap them
   for ( unsigned first = 0; first < deck.size(); ++first )
   {
      // select a random number between 0 and 51
      unsigned second = rand() % deck.size();

      // swap current Card with randomly selected Card
      Card temp = deck[ first ];
      deck[ first ] = deck[ second ];
      deck[ second ] = temp;
   } // end for
```

```
} // end function shuffle

// deal cards in deck
Card DeckOfCards::dealCard()
{
  return deck[ currentCard++ ]; // return current Card in vector
} // end function dealCard

// Check if there are more cards in the deck
bool DeckOfCards::moreCards() const
{
  return currentCard < deck.size();
} // end function moreCards
int main()
{
  DeckOfCards myDeckOfCards;
  myDeckOfCards.shuffle(); // place Cards in random order

  // print all 52 Cards in the order in which they are dealt
  for ( int i = 1; myDeckOfCards.moreCards(); i++ )
  {
    // deal and display a Card
    cout << left << setw( 19 ) << myDeckOfCards.dealCard().toString();

    if ( i % 4 == 0 ) // output newline every 4 cards
      cout << endl;
  } // end for
  system("pause");
  return 0;
} // end main
```



Discussion:

In this class we did a game in order to shuffle cards and we demonstrated the use of two classes and how they are combined together to give a randomly generating card game.

Exercise 12.9:

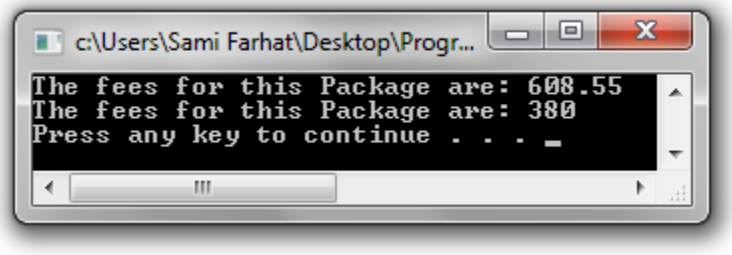
(Package *Inheritance Hierarchy*) Package-delivery services, such as FedEx®, DHL® and UPS®, offer a number of different shipping options, each with specific costs associated. Create an inheritance hierarchy to represent various types of packages. Use class Package as the base class of the hierarchy, then include classes TwoDayPackage and OvernightPackage that derive from Package. Base class Package should include data members representing the name, address, city, state and ZIP code for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package. Package's constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values. Package should provide a public member function calculateCost that returns a double indicating the cost associated with shipping the package. Package's calculateCost function should determine the cost by multiplying the weight by the cost per ounce. Derived class TwoDayPackage should inherit the functionality of base class Package, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service. TwoDayPackage's constructor should receive a value to initialize this data member. TwoDayPackage should redefine member function calculateCost so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base class Package's calculateCost function. Class OvernightPackage should inherit directly from class Package and contain an additional data member representing an additional fee per ounce charged for overnight-delivery service. OvernightPackage should redefine member function calculateCost so that it adds the additional fee per ounce to the standard cost per ounce before calculating the shipping cost. Write a test program that creates objects of each type of Package and tests member function calculateCost.

```
#ifndef PACKAGE_H
#define PACKAGE_H
#include<iostream>
#include<cstring>
using namespace std;
class Package {
private:
        char* name;
        char* address;
        char* city;
        char* state;
        char* rZip;
        char* sZip;
        double weight;
        double costPerOunce;
public:
        Package
(char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",double=0.0,double=0.0);
        double calculateCost ();
```

```
};
#endif

#include"Package.h"
Package :: Package (char* n, char* a, char* c, char* s, char* rz, char* sz, double w, double co)
{
	int k = strlen (n);
	name = new char [k+1];
	strncpy (name,n,k);
	name[k]='\0';
	int q = strlen (a);
	address = new char [q+1];
	strncpy (address,a,q);
	address[q]='\0';
	int ss = strlen (s);
	state = new char [ss+1];
	strncpy (state,s,ss);
	state[ss]='\0';
	int cc = strlen (c);
	city = new char [cc+1];
	strncpy (city,c,cc);
	city[cc]='\0';
	int kz = strlen (rz);
	rZip = new char [kz+1];
	strncpy (rZip,rz,kz);
	rZip[kz]='\0';
	int k1 = strlen (sz);
	sZip = new char [k1+1];
	strncpy (sZip,sz,k1);
	sZip[k1]='\0';
	if (w>0)
		weight=w;
	else weight=0;
	if (co>0)
		costPerOunce=co;
	else costPerOunce=0;
}
double Package :: calculateCost ()
{
	return weight*costPerOunce;
}

#ifndef OVERNIGHTPACKAGE_H
#define OVERNIGHTPACKAGE_H
#include"Package.h"
class OvernightPackage : public Package {
private:
```

```
	double feeNight;
public:
	OvernightPackage
(char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",double=0.0,double=0.0,double=0.0)
;
	double calculateCost ();
};
#endif

#include"OvernightPackage.h"
OvernightPackage :: OvernightPackage (char* nn, char* aa, char* cc, char* ss, char* rrz, char*
ssz, double ww, double coo, double addfee):Package(nn,aa,cc,ss,rrz,ssz,ww,coo)
{
	if (addfee>0)
		feeNight=addfee;
	else feeNight=0;
}
double OvernightPackage :: calculateCost ()
{
	return (double) (Package :: calculateCost () + feeNight);
}

#ifndef TWODAYPACKAGE_H
#define TWODAYPACKAGE_H
#include"Package.h"
class TwoDayPackage : public Package {
private:
	double flatFee;
public:
	TwoDayPackage
(char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",double=0.0,double=0.0,double=0.0)
;
	double calculateCost ();
};
#endif

#include"TwoDayPackage.h"
TwoDayPackage :: TwoDayPackage (char* nn, char* aa, char* cc, char* ss, char* rrz, char* ssz,
double ww, double coo,double flat):Package(nn,aa,cc,ss,rrz,ssz,ww,coo)
{
	if (flat>0)
		flatFee=flat;
	else flatFee=0;
}
double TwoDayPackage :: calculateCost ()
{
	return (double) (Package :: calculateCost () + flatFee);
```

```
}

#include"OvernightPackage.h"
#include"TwoDayPackage.h"
int main ()
{
	OvernightPackage over
("Late","See","London","Stanford","1234","3421",30.5,10.1,300.5);
	cout<<"The fees for this Package are: "<<over.calculateCost()<<endl;
	TwoDayPackage two ("Morning","Land","USA","California","5678","4321",12,15,200);
	cout<<"The fees for this Package are: "<<two.calculateCost()<<endl;
	system("pause");
	return 0;
}
```



Discussion:

This is an inheritance program where we demonstrated how inheritance is done in a simple way. We did the main to show you how we use the functions and how they are called.

Exercise 12.10:

(Account Inheritance Hierarchy) Create an inheritance hierarchy that a bank might use to represent customers' bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).

Create an inheritance hierarchy containing base class Account and derived classes Savings-Account and CheckingAccount that inherit from class Account. Base class Account should include one data member of type double to represent the account balance. The class should provide a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it's greater than or equal to 0.0. If not, the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid. The class should provide three member functions. Member function credit should add an amount to the current balance. Member function debit should withdraw money from the Account and ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged and the function should print the message "Debit amount exceeded account balance." Member function getBalance should return the current balance.

Derived class SavingsAccount should inherit the functionality of an Account, but also include a data member of type double indicating the interest rate (percentage) assigned to the Account. SavingsAccount's constructor should receive the initial balance, as well as an initial value for the SavingsAccount's interest rate. SavingsAccount should provide a public member function calculateInterest that returns a double indicating the amount of interest earned by an account. Member function calculateInterest should determine this amount by multiplying the interest rate by the account balance. [Note: SavingsAccount should inherit member functions credit and debit as is without redefining them.]

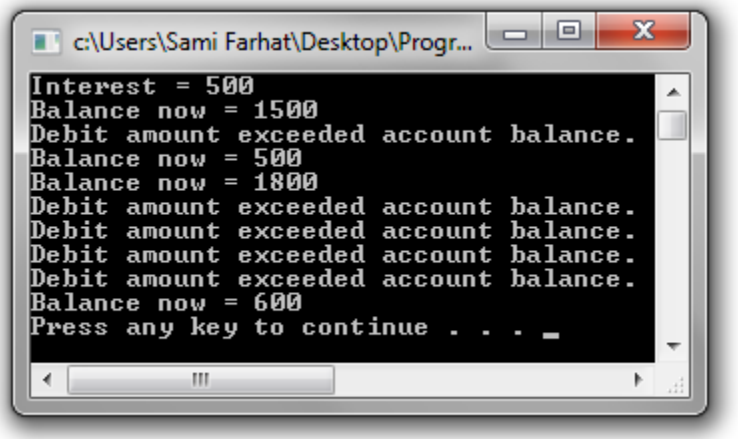
Derived class CheckingAccount should inherit from base class Account and include an additional data member of type double that represents the fee charged per transaction. Checking-Account's constructor should receive the initial balance, as well as a parameter indicating a fee amount. Class CheckingAccount should redefine member functions credit and debit so that they subtract the fee from the account balance whenever either transaction is performed successfully. CheckingAccount's versions of these functions should invoke the base-class Account version to perform the updates to an account balance. CheckingAccount's debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). [Hint: Define Account's debit function so that it returns a bool indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]

After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the SavingsAccount object by first invoking its calculateInterest function, then passing the returned interest amount to the object's credit function.

```
#include<iostream>
using namespace std;
class Account {
private:
	double balance;
public:
	Account (double=0.0);
	double getBalance ();
	void credit (double);
	bool debit (double);
};
Account :: Account (double a)
{
	if (a>=0.0)
		balance=a;
	else balance=0.0;
}
double Account :: getBalance () {return balance;}
void Account :: credit (double c)
{
	balance+=c;
}
bool Account :: debit (double d)
{
	if (balance-d>=0.0)
	{
		balance-=d;
		return true;
	}
	cout<<"Debit amount exceeded account balance."<<endl;
	return false;
}
class SavingsAccount : public Account {
private:
	double inrate;
public:
	SavingsAccount (double=0.0,double=0.0);
	double calculateInterest ();
};
SavingsAccount :: SavingsAccount (double a, double i):Account(a)
{
```

```
        if (i>=0)
                inrate=i;
        else inrate=0;
}
double SavingsAccount :: calculateInterest ()
{
        return (double) inrate*getBalance();
}
class CheckingAccount : public Account {
private:
        double feecha;
public:
        CheckingAccount (double=0.0,double=0.0);
        void credit (double);
        void debit (double);
};
CheckingAccount :: CheckingAccount (double a, double f):Account(a)
{
        if (f>=0)
                feecha=f;
        else feecha=0.0;
}
void CheckingAccount :: credit (double b)
{
        Account::credit(b);
        if (getBalance()-feecha>=0)
                Account::debit(feecha);
        else cout<<"Charged fee exceeded account balance.";
}
void CheckingAccount :: debit (double d)
{
        if (Account::debit(d)==true)
        {
                if (Account::debit(feecha)==true) {}
                else cout<<"Debit amount with the charged fee exceeded account
balance."<<endl;
        }
        else cout<<"Debit amount exceeded account balance."<<endl;
}
int main ()
{
        SavingsAccount s (1000,0.5);
        cout<<"Interest = "<<s.calculateInterest()<<endl;
        s.credit(s.calculateInterest());
        cout<<"Balance now = "<<s.getBalance()<<endl;
        s.debit(2000);
```

```
        s.debit(1000);
        cout<<"Balance now = "<<s.getBalance()<<endl;
        CheckingAccount c (1000,200);
        c.credit(1000);
        cout<<"Balance now = "<<c.getBalance()<<endl;
        c.debit(10000);
        c.debit(15000);
        c.debit(1000);
        cout<<"Balance now = "<<c.getBalance()<<endl;
        system("pause");
        return 0;
}
```



Discussion:

In this herarcy, we made a small bank program of two types of accounts. The base class has the private attributed and the derived classes are which we made objects from.

Exercise 13.15:

*(*Package *Inheritance Hierarchy)* Use the Package inheritance hierarchy created in Exercise 12.9 to create a program that displays the address information and calculates the shipping costs for several Packages. The program should contain a vector of Package pointers to objects of classes TwoDayPackage and OvernightPackage. Loop through the vector to process the Packages polymorphically. For each Package, invoke *get* functions to obtain the address information of the sender and the recipient, then print the two addresses as they would appear on mailing labels. Also, call each Package's calculateCost member function and print the result. Keep track of the total shipping cost for all Packages in the vector, and display this total when the loop terminates.

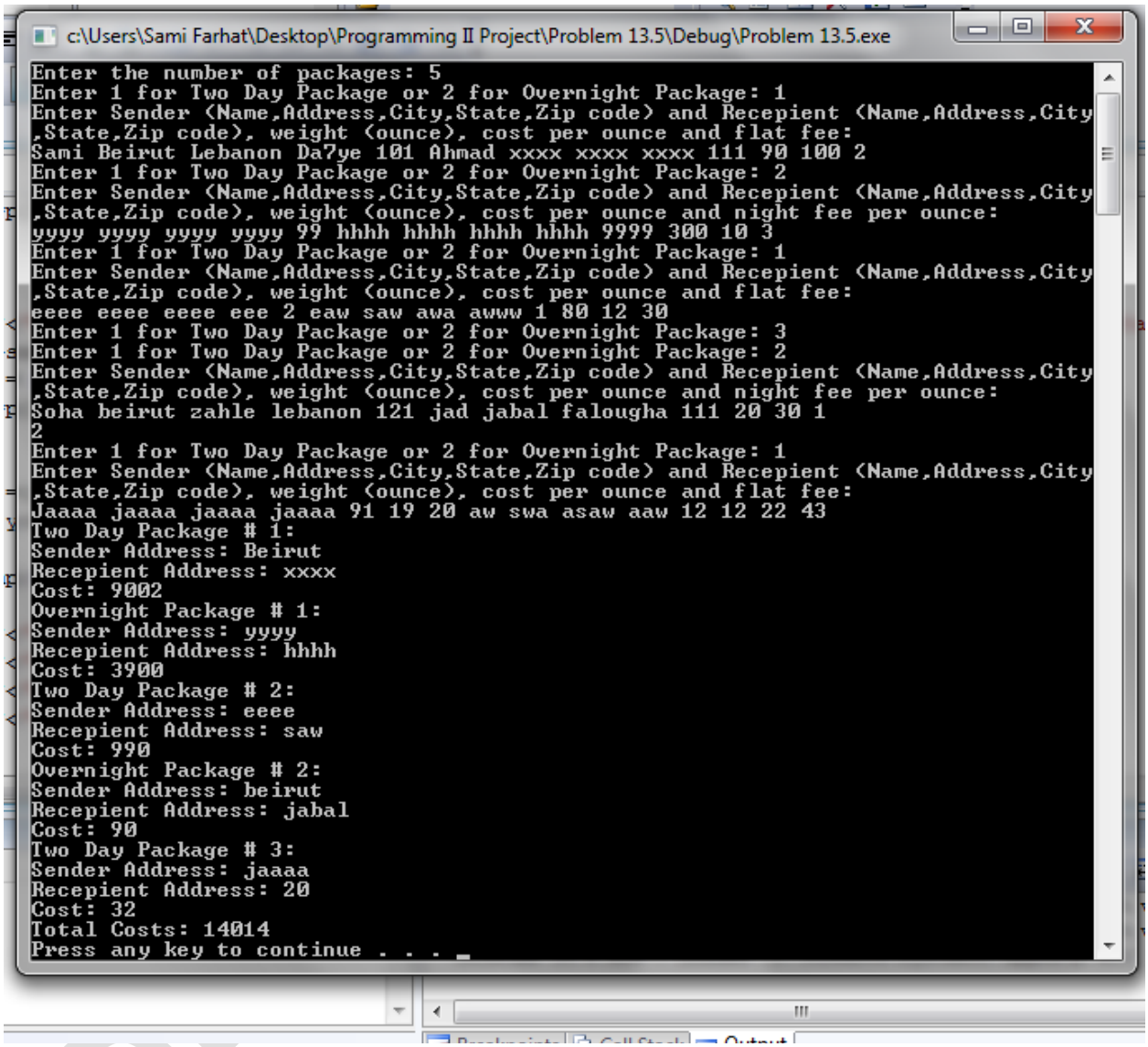
```
#include<iostream>
using namespace std;
class Package {
protected:
        char* Rname;
        char* Raddress;
        char* Rcity;
        char* Rstate;
        char* Rzip;
        char* Sname;
        char* Saddress;
        char* Scity;
        char* Sstate;
        char* Szip;
        double weight;
        double costPerOunce;
public:
        Package
(char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]
="",double=0.0,double=0.0);
        virtual double calculateCost ();
        virtual char* getRaddress () = 0;
        virtual char* getSaddress () = 0;
};
Package :: Package (char* sn, char* sa, char* sc, char* ss, char* sz, char* rn, char* ra, char* rc,
char* rs, char* rz, double w, double co)
{
        int k1 = strlen (sn);
        Sname = new char [k1+1];
        strncpy (Sname,sn,k1);
        Sname[k1]='\0';
        int k2 = strlen (sa);
        Saddress = new char [k2+1];
        strncpy (Saddress,sa,k2);
        Saddress [k2] = '\0';
```

```
        int k3 = strlen (sc);
        Scity = new char [k3+1];
        strncpy (Scity,sc,k3);
        Scity[k3] = '\0';
        int k4 = strlen (ss);
        Sstate = new char [k4];
        strncpy (Sstate,ss,k4);
        Sstate[k4]='\0';
        int k5 = strlen (sz);
        Szip = new char [k5];
        strncpy (Szip,sz,k5);
        Szip[k5]='\0';
        int k6 = strlen (rn);
        Rname = new char [k6+1];
        strncpy (Rname,rn,k6);
        Rname[k6]='\0';
        int k7 = strlen (ra);
        Raddress = new char [k7+1];
        strncpy (Raddress,ra,k7);
        Raddress [k7] = '\0';
        int k8 = strlen (rc);
        Rcity = new char [k8+1];
        strncpy (Rcity,rc,k8);
        Rcity[k8] = '\0';
        int k9 = strlen (rs);
        Rstate = new char [k9];
        strncpy (Rstate,rs,k9);
        Rstate[k9]='\0';
        int k0 = strlen (rz);
        Rzip = new char [k0];
        strncpy (Rzip,rz,k0);
        Rzip[k0]='\0';
        if (w>0)
                weight=w;
        else weight=0;
        if (co>0)
                costPerOunce=co;
        else costPerOunce=0;
}
double Package :: calculateCost ()
{
        return weight*costPerOunce;
}
class OvernightPackage : public Package {
private:
        double feeNight;
```

```
public:
	OvernightPackage
(char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]
="",double=0.0,double=0.0,double=0.0);
	double calculateCost ();
	char* getRaddress ();
	char* getSaddress ();
};
char* OvernightPackage :: getRaddress () {return Raddress;}
char* OvernightPackage :: getSaddress () {return Saddress;}
OvernightPackage :: OvernightPackage (char* sn, char* sa, char* sc, char* ss, char* sz, char* rn,
char* ra, char* rc, char* rs, char* rz, double ww, double coo, double
addfee):Package(sn,sa,sc,ss,sz,rn,ra,rc,rs,rz,ww,coo)
{
	if (addfee>0)
		feeNight=addfee;
	else feeNight=0;
}
double OvernightPackage :: calculateCost ()
{
	return (double) (Package :: calculateCost () + feeNight*weight);
}
class TwoDayPackage : public Package {
private:
	double flatFee;
public:
	TwoDayPackage
(char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]="",char[]
="",double=0.0,double=0.0,double=0.0);
	double calculateCost ();
	char* getRaddress ();
	char* getSaddress ();
};
char* TwoDayPackage :: getRaddress () {return Raddress;}
char* TwoDayPackage :: getSaddress () {return Saddress;}
TwoDayPackage :: TwoDayPackage (char* sn, char* sa, char* sc, char* ss, char* sz, char* rn,
char* ra, char* rc, char* rs, char* rz, double ww, double coo, double
flat):Package(sn,sa,sc,ss,sz,rn,ra,rc,rs,rz,ww,coo)
{
	if (flat>0)
		flatFee=flat;
	else flatFee=0;
}
double TwoDayPackage :: calculateCost ()
{
	return (double) (Package :: calculateCost () + flatFee);
```

```
}
int main ()
{
	int N,k;
	double sum=0;
	cout<<"Enter the number of packages: ";
	cin>>N;
	Package ** p = new Package* [N];
	char ssn [20];
	char ssa [20];
	char ssc [20];
	char sss [20];
	char ssz [20];
	char rrn [20];
	char rra [20];
	char rrc [20];
	char rrs [20];
	char rrz [20];
	double w,x,y;
	for (int i=0; i<N; i++)
	{
		do
		{
			cout<<"Enter 1 for Two Day Package or 2 for Overnight Package: ";
			cin>>k;
		} while (k!=1 && k!=2);
		if (k==1)
		{
			cout<<"Enter Sender (Name,Address,City,State,Zip code) and Recepient
(Name,Address,City,State,Zip code), weight (ounce), cost per ounce and flat fee: "<<endl;
			cin>>ssn>>ssa>>ssc>>sss>>ssz>>rrn>>rra>>rrc>>rrs>>rrz>>w>>x>>y;
			p[i] = new TwoDayPackage (ssn,ssa,ssc,sss,ssz,rrn,rra,rrc,rrs,rrz,w,x,y);
			sum+=p[i]->calculateCost();
		}
		else
		{
			cout<<"Enter Sender (Name,Address,City,State,Zip code) and Recepient
(Name,Address,City,State,Zip code), weight (ounce), cost per ounce and night fee per ounce:
"<<endl;
			cin>>ssn>>ssa>>ssc>>sss>>ssz>>rrn>>rra>>rrc>>rrs>>rrz>>w>>x>>y;
			p[i] = new OvernightPackage (ssn,ssa,ssc,sss,ssz,rrn,rra,rrc,rrs,rrz,w,x,y);
			sum+=p[i]->calculateCost();
		}
	}
	int ee=1,jjjj=1;
	for (int y=0;y<N;y++)
```

```
	{
		if (strcmp(typeid(*p[y]).name(),"class TwoDayPackage")==0)
		{
			cout<<"Two Day Package # "<<ee++<<": "<<endl;
			cout<<"Sender Address: "<<p[y]->getSaddress()<<endl;
			cout<<"Recepient Address: "<<p[y]->getRaddress()<<endl;
			cout<<"Cost: "<<p[y]->calculateCost()<<endl;
		}
		else
		{
			cout<<"Overnight Package # "<<jjjj++<<": "<<endl;
			cout<<"Sender Address: "<<p[y]->getSaddress()<<endl;
			cout<<"Recepient Address: "<<p[y]->getRaddress()<<endl;
			cout<<"Cost: "<<p[y]->calculateCost()<<endl;
		}
	}
	cout<<"Total Costs: "<<sum<<endl;
	system("pause");
	return 0;
}
```



Discussion:

Now, using polymorphism we made exercise 12.9 after modifying it to call functions polymorphically.

Exercise 13.16:

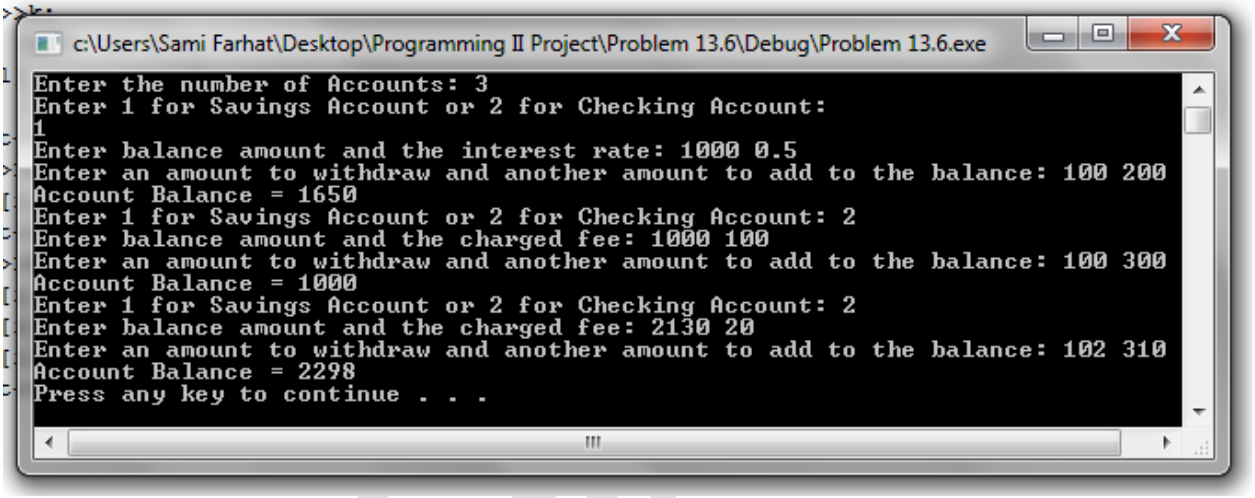
(Polymorphic Banking Program Using Account *Hierarchy)* Develop a polymorphic banking program using the Account hierarchy created in Exercise 12.10. Create a vector of Account pointers to SavingsAccount and CheckingAccount objects. For each Account in the vector, allow the user to specify an amount of money to withdraw from the Account using member function debit and an amount of money to deposit into the Account using member function credit. As you process each Account, determine its type. If an Account is a SavingsAccount, calculate the amount of interest owed to the Account using member function calculateInterest, then add the interest to the account balance using member function credit. After processing an Account, print the updated account balance obtained by invoking base-class member function getBalance.

```
#include<iostream>
using namespace std;
class Account {
private:
        double balance;
public:
        Account (double=0.0);
        virtual double getBalance ();
        virtual void credit (double);
        virtual void debit (double);
};
Account :: Account (double a)
{
        if (a>=0.0)
                balance=a;
        else balance=0.0;
}
double Account :: getBalance () {return balance;}
void Account :: credit (double c)
{
        balance+=c;
}
void Account :: debit (double d)
{
        if (balance-d>=0.0)
        {
                balance-=d;
        }
        else cout<<"Debit amount exceeded account balance."<<endl;
}
class SavingsAccount : public Account {
private:
        double inrate;
public:
```

```
        SavingsAccount (double=0.0,double=0.0);
        double calculateInterest ();
        double getBalance ();
        void credit (double);
        void debit (double);
};
SavingsAccount :: SavingsAccount (double a, double i):Account(a)
{
        if (i>=0)
                inrate=i;
        else inrate=0;
}
void SavingsAccount :: credit (double b)
{
        Account::credit(b);
}
void SavingsAccount :: debit (double d)
{
        Account::debit(d);
}
double SavingsAccount :: calculateInterest ()
{
        return (double) inrate*getBalance();
}
double SavingsAccount :: getBalance () {return Account::getBalance();}
class CheckingAccount : public Account {
private:
        double feecha;
public:
        CheckingAccount (double=0.0,double=0.0);
        double getBalance ();
        void credit (double);
        void debit (double);
};
CheckingAccount :: CheckingAccount (double a, double f):Account(a)
{
        if (f>=0)
                feecha=f;
        else feecha=0.0;
}
void CheckingAccount :: credit (double b)
{
        Account::credit(b);
        if (getBalance()-feecha>=0)
                Account::debit(feecha);
        else cout<<"Charged fee exceeded account balance.";
```

```
}
void CheckingAccount :: debit (double d)
{
	if (getBalance()-d-feecha>=0)
	{
		Account::debit(d);
		Account::debit(feecha);
	}
	else cout<<"Debit amount and charged fee exceeded account balance."<<endl;
}
double CheckingAccount :: getBalance() {return Account::getBalance();}
int main ()
{
	int N;
	cout<<"Enter the number of Accounts: ";
	cin>>N;
	Account ** aaa = new Account* [N];
	int k;
	double bal,infee;
	for (int i=0; i<N; i++)
	{
		do
		{
			cout<<"Enter 1 for Savings Account or 2 for Checking Account: ";
			cin>>k;
		} while (k!=1 && k!=2);
		if (k==1)
		{
			cout<<"Enter balance amount and the interest rate: ";
			cin>>bal>>infee;
			aaa[i] = new SavingsAccount (bal,infee);
			cout<<"Enter an amount to withdraw and another amount to add to the
balance: ";
			cin>>bal>>infee;
			aaa[i]->credit(infee);
			aaa[i]->debit(bal);
			aaa[i]->credit(((SavingsAccount*)aaa[i])->calculateInterest());
			cout<<"Account Balance = "<<aaa[i]->getBalance()<<endl;
		}
		else
		{
			cout<<"Enter balance amount and the charged fee: ";
			cin>>bal>>infee;
			aaa[i] = new CheckingAccount (bal,infee);
			cout<<"Enter an amount to withdraw and another amount to add to the
balance: ";
```

```
                    cin>>bal>>infee;
                    aaa[i]->credit(infee);
                    aaa[i]->debit(bal);
                    cout<<"Account Balance = "<<aaa[i]->getBalance()<<endl;
                }
        }
        system("pause");
        return 0;
}
```



Discussion:

Problem 12.10 modified where now functions are called polymorphically.

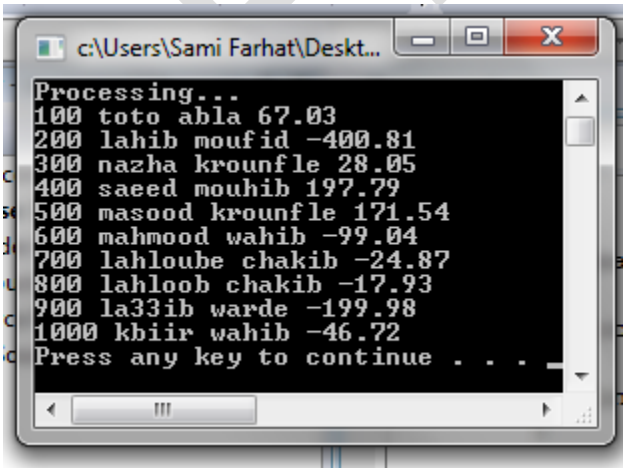
Exercise 17.6:

(File Matching) Exercise 17.3 asked you to write a series of single statements. Actually, these statements form the core of an important type of file-processing program, namely, a filematching program. In commercial data processing, it's common to have several files in each application system. In an accounts receivable system, for example, there is generally a master file containing detailed information about each customer, such as the customer's name, address, telephone number, outstanding balance, credit limit, discount terms, contract arrangements and, possibly, a condensed history of recent purchases and cash payments.
As transactions occur (e.g., sales are made and cash payments arrive), they're entered into a file. At the end of each business period (a month for some companies, a week for others and a day in some cases), the file of transactions (called trans.dat in Exercise 17.3) is applied to the master file (called oldmast.dat in Exercise 17.3), thus updating each account's record of purchases and payments. During an updating run, the master file is rewritten as a new file (newmast.dat), which is then used at the end of the next business period to begin the updating process again. File-matching programs must deal with certain problems that do not exist in single-file programs.
For example, a match does not always occur. A customer on the master file might not have made any purchases or cash payments in the current business period, and therefore no record for this customer will appear on the transaction file. Similarly, a customer who did make some purchases or cash payments may have just moved to this community, and the company may not have had a chance to create a master record for this customer. Use the statements from Exercise 17.3 as a basis for writing a complete file-matching accounts receivable program. Use the account number on each file as the record key for matching purposes.
Assume that each file is a sequential file with records stored in increasing order by account number. When a match occurs (i.e., records with the same account number appear on both the master and transaction files), add the dollar amount on the transaction file to the current balance on the master file, and write the newmast.dat record. (Assume purchases are indicated by positive amounts on the transaction file and payments are indicated by negative amounts.) When there is a master record for a particular account but no corresponding transaction record, merely write the master record to newmast.dat. When there is a transaction record but no corresponding master record, print the error message "Unmatched transaction record for account number ..." (fill in the account number from the transaction record).

```
//Excercise 17.6
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>
using namespace std;
void printOutput(ofstream&,int,const char *,const char *,double );//print function prototype

void main()
{
        int masterAccount,transactionAccount;
        double masterBalance,transactionBalance;
```

```
	char masterFirstName[20],masterLastName[20];

	ifstream inOldMaster("oldmast.dat",ios::in | ios::binary);//opening oldmast.dat
	ifstream inTransaction("trans.dat",ios::in | ios::binary);//openeing trans.dat
	ofstream outNewMaster("newmast.dat",ios::out | ios::binary);//creating newmast.dat

	if (!inOldMaster)
	{
		cerr << "Unable to open oldmast.dat\n";
		exit(1);
	}
	if (!inTransaction)
	{
		cerr << "Unable to open trans.dat\n";
		exit(1);
	}

	if (!outNewMaster)
	{
		cerr << "Unable to open newmast.dat\n";
		exit(1);
	}

	cout << "Processing...\n";
	inTransaction>>transactionAccount>>transactionBalance;

	while (!inTransaction.eof())
	{

	inOldMaster>>masterAccount>>masterFirstName>>masterLastName>>masterBalance;
		while (masterAccount<transactionAccount && !inOldMaster.eof())
		{

	printOutput(outNewMaster,masterAccount,masterFirstName,masterLastName,masterBal
ance);

	inOldMaster>>masterAccount>>masterFirstName>>masterLastName>>masterBalance;
		}

		if (masterAccount>transactionAccount)
		{
			cout<<"Unmatched transaction record for account:
"<<transactionAccount<<'\n';

			inTransaction>>transactionAccount>>transactionBalance;
		}
```

```
                if (masterAccount == transactionAccount)
                {
                        masterBalance += transactionBalance;

        printOutput(outNewMaster,masterAccount,masterFirstName,masterLastName,masterBal
ance );//function to print the newmast.dat
                }

                inTransaction>>transactionAccount>>transactionBalance;
        }

        inTransaction.close();
        outNewMaster.close();
        inOldMaster.close();
        system ("pause");
}

void printOutput(ofstream &oRef,int mAccount,const char *mfName,const char
*mlName,double mBalance )//print function definition
{
        cout.setf( ios::fixed | ios::showpoint );
        oRef.setf( ios::fixed | ios::showpoint );

        oRef<<mAccount<<' '<<mfName<<' '<<mlName<<' '<<
setprecision(2)<<mBalance<<'\n';
        cout<<mAccount<<' '<<mfName<<' '<<mlName<<'
'<<setprecision(2)<<mBalance<<'\n';
}
```



Discussion:

We showed the structure of binary files and we can open them to read values and print values.

Exercise 17.7:

(File Matching Test Data) After writing the program of Exercise 17.6, write a simple program to create some test data for checking out the program. Use the following sample account data:

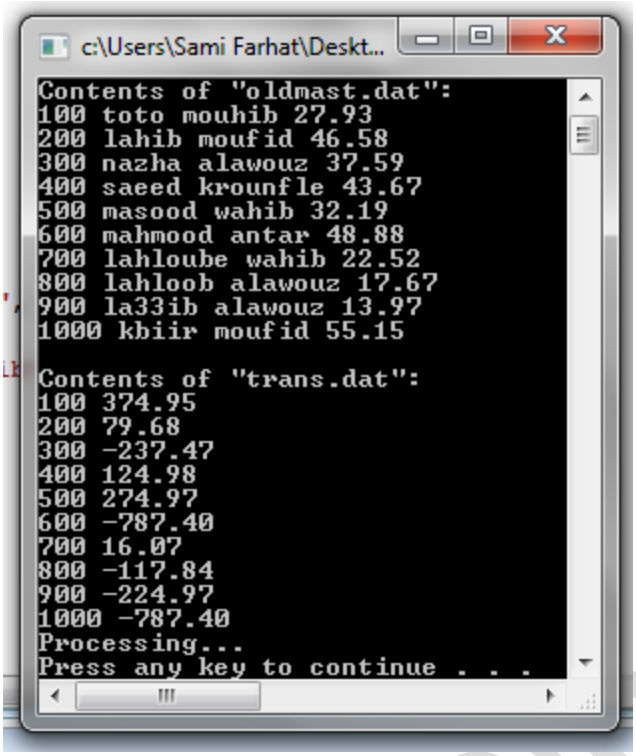
Master filenumber	Name	Balance
100	Alan Jones	348.17
300	Mary Smith	27.19
500	Sam Sharp	0.00
700	Suzy Green	–14.22

Transaction file Account number	Transaction amount
100	27.14
300	62.11
400	100.56
900	82.17

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;

void main()
{
	const char *firstNames[] = {"toto", "lahib", "nazha", "saeed", "masood",
		"mahmood", "lahloube", "lahloob", "la33ib", "kbiir" };
	const char *lastNames[] = { "warde", "afif", "alawouz", "wahib", "mouhib",
		"chakib", "abla", "antar", "krounfle", "moufid" };
	ofstream outOldMaster("oldMast.dat",ios::out|ios::binary);
	ofstream outTransaction("trans.dat",ios::out|ios::binary);
	int z;
	srand(time(0));
	if (!outOldMaster)
	{
		cerr << "Unable to open oldmast.dat\n";
		exit(1);
	}

	if ( !outTransaction )
	{
		cerr << "Unable to open trans.dat\n";
```

```
            exit(1);
        }

        // write data to "oldmast.dat"
        cout << setiosflags( ios::fixed | ios::showpoint ) //sets ios flags fixed to showpoint
            << "Contents of \"oldmast.dat\":\n";

        outOldMaster.setf( ios::fixed | ios::showpoint ); //setting specific flag fixed to showpoint
        for ( z = 1; z < 11; ++z )
        {
            int value = rand() % 10, value2 = rand() % 50;
            outOldMaster << z * 100 << ' ' << firstNames[ z - 1 ] <<' '<< lastNames[ value ]
<< ' ' << setprecision( 2 )<< ( value * 100 ) / ( value2 / 3 + 4.32 ) << '\n';
            cout << z * 100 << ' ' << firstNames[ z - 1 ] <<' '<< lastNames[ value ] << ' ' <<
setprecision( 2 )<< ( value * 100 ) / ( value2 / 3 + 4.32 ) << '\n';
        }

        // write data to "trans.dat"
        cout << "\nContents of \"trans.dat\":\n";
        outTransaction.setf( ios::fixed | ios::showpoint ); //setting specific flag fixed to showpoint

        for ( z = 1; z < 11; ++z ) {
            int value = 25 - rand() % 50;
            outTransaction << z * 100 << ' ' << setprecision( 2 )<< ( value * 100 ) / ( 2.667 *
( 1 + rand() % 10 ) )<< '\n';
            cout << z * 100 << ' ' << setprecision( 2 )<< ( value * 100 ) / ( 2.667 * ( 1 +
rand() % 10 ) ) << '\n';
        }

        outTransaction.close();
        outOldMaster.close();

        ifstream inMaster( "oldmast.dat" ), inTrans( "trans.dat" );
        ofstream newMaster( "newMast.dat" );

        if ( !inMaster ) {
            cerr << "Unable to open oldmast.dat\n";
            exit(1);
        }

        if ( !inTrans ) {
            cerr << "Unable to open trans.dat\n";
            exit(1);
        }

        if ( !newMaster ) {
```

```
            cerr << "Unable to open newmast.dat\n";
            exit(1);
        }

        int account, mAccount;
        double balance, mBalance;
        char mFirst[ 20 ], mLast[ 20 ];

        cout << "Processing...\n";
        inTrans >> account >> balance;

        while ( !inTrans.eof() ) {
            inMaster >> mAccount >> mFirst >> mLast >> mBalance;
            while ( mAccount < account && !inMaster.eof() )
                inMaster >> mAccount >> mFirst >> mLast >> mBalance;

            if ( mAccount > account ) {
                cout << "Unmatched transaction record: account " << account << '\n';

                inTrans >> account >> balance;
            }

            if ( mAccount == account ) {
                mBalance += balance;
                newMaster << mAccount << ' ' << mFirst << ' ' << mLast
                    << ' ' << mBalance << '\n';
            }

            inTrans >> account >> balance;
        }

        newMaster.close();
        inTrans.close();
        inMaster.close();
        system("pause");
}
```



Discussion:

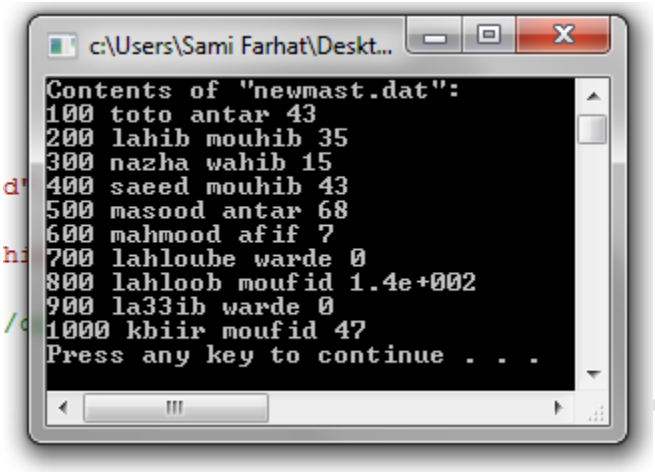
Testing data

Exercise 17.8:

(File Matching Test) Run the program of Exercise 17.6, using the files of test data created in Exercise 17.7. Print the new master file. Check that the accounts have been updated correctly.

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;

void main()
{
	const char *firstNames[] = {"toto", "lahib", "nazha", "saeed",
"masood",
		"mahmood", "lahloube", "lahloob", "la33ib", "kbiir" };
	const char *lastNames[] = { "warde", "afif", "alawouz", "wahib",
"mouhib",
		"chakib", "abla", "antar", "krounfle", "moufid" };
	ofstream outNewMaster("newMast.dat",ios::out|ios::binary);//opening
file for input output
	int z;
	srand(time(0));
	if (!outNewMaster)
	{
		cerr << "Unable to open newmast.dat\n";
		exit(1);
	}

	cout<< "Contents of \"newmast.dat\":\n";

	outNewMaster.setf( ios::fixed | ios::showpoint ); //setting specific flag fixed
to showpoint
	for ( z = 1; z < 11; ++z )
	{
		int value = rand() % 10, value2 = rand() % 50;
		outNewMaster << z * 100 << ' ' << firstNames[ z - 1 ] <<' '<<
lastNames[ value ] << ' ' << setprecision( 2 )<< ( value * 100 ) / ( value2 /
3 + 4.32 ) << '\n';
		cout << z * 100 << ' ' << firstNames[ z - 1 ] <<' '<< lastNames[
value ] << ' ' << setprecision( 2 )<< ( value * 100 ) / ( value2 / 3 + 4.32 )
<< '\n';
	}//fucntion to output contents of newmast.dat
/*ios::fixed is for When floatfield is set to fixed,
float values are written using fixed-point notation,
which means the value is represented with exactly as many digits in the
fraction part
as specified by the precision field and with no exponent part.*/
/* ios::showpoint When the showpoint format flag is set,
the decimal point is always written for floating point values insterted into
the stream,
even for whole numbers. Following the decimal point,
as many digits as necessary are written to match the
precision internal setting for thestream (if any).*/
```

```
	outNewMaster.close();//close file newmast.dat
	system("pause");
}
```



Discussion:

In this exercises we applied file matching to check if same files.

Exercise 17.9:

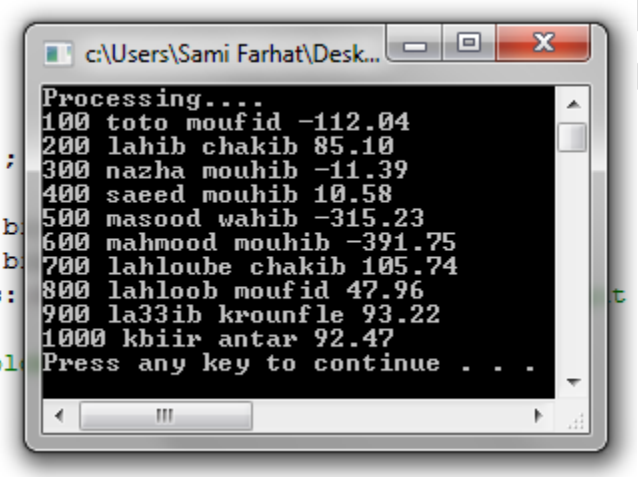
(File Matching Enhancement) It's common to have several transaction records with the same record key, because a particular customer might make several purchases and cash payments during a business period. Rewrite your accounts receivable file-matching program of Exercise 17.6 to provide for the possibility of handling several transaction records with the same record key. Modify the test data of Exercise 17.7 to include the following additional transaction records:

Account number	Dollar amount
300	83.89
700	80.78
700	1.53

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>
using namespace std;
void printOutput( ofstream &, int, const char *, const char *, double );//function prototype for print
void main()
{
	int masterAccount, transactionAccount;
	double masterBalance, transactionBalance;
	char masterFirstName[ 20 ], masterLastName[ 20 ];

	ifstream inOldmaster("oldmast.dat",ios::in|ios::binary);//open oldmas.dat
	ifstream inTransaction("trans.dat",ios::in|ios::binary);//open trans.dat
	ofstream outNewmaster("newmast.dat",ios::out|ios::binary);//create newmast.dat to input data on it

	if (!inOldmaster) //error code if couldnt open oldmast.dat
	{
		cerr <<"Unable to open oldmast.dat\n";
		exit(1);
	}

	if (!inTransaction) //error if couldnt open trans.dat
	{
		cerr <<"Unable to open trans.dat\n";
		exit(1);
	}
	if (!outNewmaster) // error if couldnt open newmast.dat
	{
		cerr <<"Unable to open newmast.dat\n";
		exit(1);
```

```
	}

	cout<<"Processing....\n";
	inTransaction>>transactionAccount>>transactionBalance;//read from trans.dat file

	while (!inTransaction.eof())
	{

	inOldmaster>>masterAccount>>masterFirstName>>masterLastName>>masterBalance;//
read from oldmast.dat

		while (masterAccount < transactionAccount && !inOldmaster.eof()) //condition
to print
		{
			printOutput(outNewmaster, masterAccount,
masterFirstName,masterLastName, masterBalance);
			inOldmaster>>masterAccount>>masterFirstName>>masterLastName>>
masterBalance;
		}

		if (masterAccount > transactionAccount) //condition if master account is greater
OR smaller than transaction account
		{
			cout <<"Unmatched transaction record for account:
"<<transactionAccount << '\n';
			inTransaction>>transactionAccount>>transactionBalance;
		}
		else if (masterAccount < transactionAccount)
		{
			cout <<"Unmatched transaction record for account:
"<<transactionAccount<< '\n';
			inTransaction>>transactionAccount>>transactionBalance;
		}

		while (masterAccount == transactionAccount && !inTransaction.eof()) //while
loop to check if master account equal to transaction account
		{
				masterBalance += transactionBalance;//the new master balance is
equal to old master balance + the transaction balance
				inTransaction>>transactionAccount>>transactionBalance; // read
from trans.dat
		}

		printOutput(outNewmaster, masterAccount, masterFirstName, masterLastName,
masterBalance );//print function
	}
```

```
        inTransaction.close();//close file trans.dat
        outNewmaster.close();//close file newmast.dat
        inOldmaster.close();//close file oldmast.dat

        system("pause");

}//end of main

void printOutput( ofstream &oRef, int mAccount, const char *mfName, const char *mlName,
double mBalance )//function definition
{
        cout.setf( ios::showpoint | ios::fixed );
        oRef.setf( ios::showpoint | ios::fixed );
        oRef << mAccount << ' ' << mfName << ' ' << mlName << ' '
                << setprecision( 2 ) << mBalance << '\n';
        cout << mAccount << ' ' << mfName << ' ' << mlName << ' '
                << setprecision( 2 ) << mBalance << '\n';
        cout.unsetf( ios::showpoint | ios::fixed );
}
```



Exercise 17.12:

(Telephone Number Word Generator) Standard telephone keypads contain the digits 0 through 9. The numbers 2 through 9 each have three letters associated with them, as is indicated by the following table:

Digit	Letter	Digit	Letter
2	A B C	6	M N O
3	D E F	7	P Q R S
4	G H I	8	T U V
5	J K L	9	W X Y Z

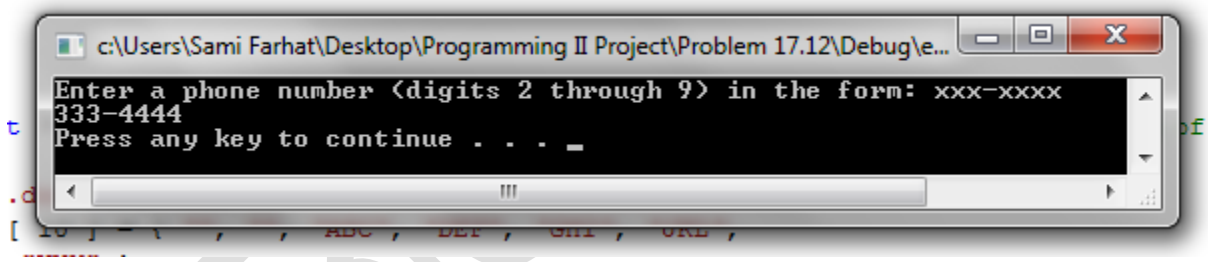
Many people find it difficult to memorize phone numbers, so they use the correspondence between digits and letters to develop seven-letter words that correspond to their phone numbers. For example, a person whose telephone number is 686-2377 might use the correspondence indicated in the above table to develop the seven-letter word "NUMBERS." Businesses frequently attempt to get telephone numbers that are easy for their clients to remember. If a business can advertise a simple word for its customers to dial, then no doubt the business will receive a few more calls. Each seven-letter word corresponds to exactly one seven-digit telephone number. The restaurant wishing to increase its take-home business could surely do so with the number 825-3688 (i.e., "TAKEOUT"). Each seven-digit phone number corresponds to many separate seven-letter words. Unfortunately, most of these represent unrecognizable juxtapositions of letters. It's possible, however, that the owner of a barber shop would be pleased to know that the shop's telephone number, 424-7288, corresponds to "HAIRCUT." A veterinarian with the phone number 738-2273 would be pleased to know that the number corresponds to "PETCARE."

Write a program that, given a seven-digit number, writes to a file every possible seven-letter word corresponding to that number. There are 2187 (3 to the seventh power) such words. Avoid phone numbers with the digits 0 and 1.

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cctype>
using namespace std;
void wordGenerator( const int * const );//fucntion prototype to calculate all possible combination of letters
void main()
{
        int phoneNumber[ 7 ] = { 0 };
```

```
	cout << "Enter a phone number (digits 2 through 9) "
		<< "in the form: xxx-xxxx\n";

	// loop 8 times: 7 digits plus hyphen
	// hyphen is is not placed in phoneNumber
	for ( int u = 0, v = 0; u < 8; ++u ) {
		int i = cin.get();


		if ( isdigit( i ) ) // cctype library to check if i is a digit or not
			phoneNumber[ v++ ] = i - '0';
	}

	wordGenerator( phoneNumber );

	system("pause");
}

void wordGenerator( const int * const n )//fucntion definition to calculate all possible
combination of letters
{
	ofstream outFile( "phone.dat",ios::out );
	const char *phoneLetters[ 10 ] = { "", "", "ABC", "DEF", "GHI", "JKL",
		"MNO", "PRS", "TUV", "WXY" };

	if ( !outFile )
	{
		cerr << "\"phone.dat\" could not be opened.\n";
		exit(1);
	}

	int count = 0;

	// output all possible combinations
	for ( int i1 = 0; i1 <= 2; ++i1 )
		for ( int i2 = 0; i2 <= 2; ++i2 )
			for ( int i3 = 0; i3 <= 2; ++i3 )
				for ( int i4 = 0; i4 <= 2; ++i4 )
					for ( int i5 = 0; i5 <= 2; ++i5 )
						for ( int i6 = 0; i6 <= 2; ++i6 )
							for ( int i7 = 0; i7 <= 2; ++i7 ) {
								outFile << phoneLetters[ n[ 0 ] ][ i1 ]
								<< phoneLetters[ n[ 1 ] ][ i2 ]
								<< phoneLetters[ n[ 2 ] ][ i3 ]
								<< phoneLetters[ n[ 3 ] ][ i4 ]
```

```
                << phoneLetters[ n[ 4 ] ][ i5 ]
                << phoneLetters[ n[ 5 ] ][ i6 ]
                << phoneLetters[ n[ 6 ] ][ i7 ] << ' ';

                if ( ++count % 9 == 0 )
                        outFile << '\n';
        }

        outFile << "\nPhone number is ";

        for ( int i = 0; i < 7; ++i ) {
                if ( i == 3 )
                        outFile << '-';

                outFile << n[ i ];
        }

        outFile.close();
}
```



Discussion:

This class provides advantages for stores if they want to choose a customer number that is easy to remember depending on letters.

TABLE OF CONTENTS		
Program #	**Percentage Done**	**Grade**
8.19	**0%**	
10.10	**100%**	
10.11	**0%**	
12.9	**90%**	
12.10	**100%**	
13.15	**100%**	
13.16	**100%**	
13.18	**0%**	
17.6	**100%**	
17.7	**95%**	
17.8	**95%**	
17.9	**95%**	
17.12	**100%**	
17.14	**0%**	

Ahmad Bazzi

Sami Farhat

Spring 2012