CIS023 Laboratory 1 Assignment Mockup

Exercise L1-1 Chapter 10 Classes & Data Abstraction

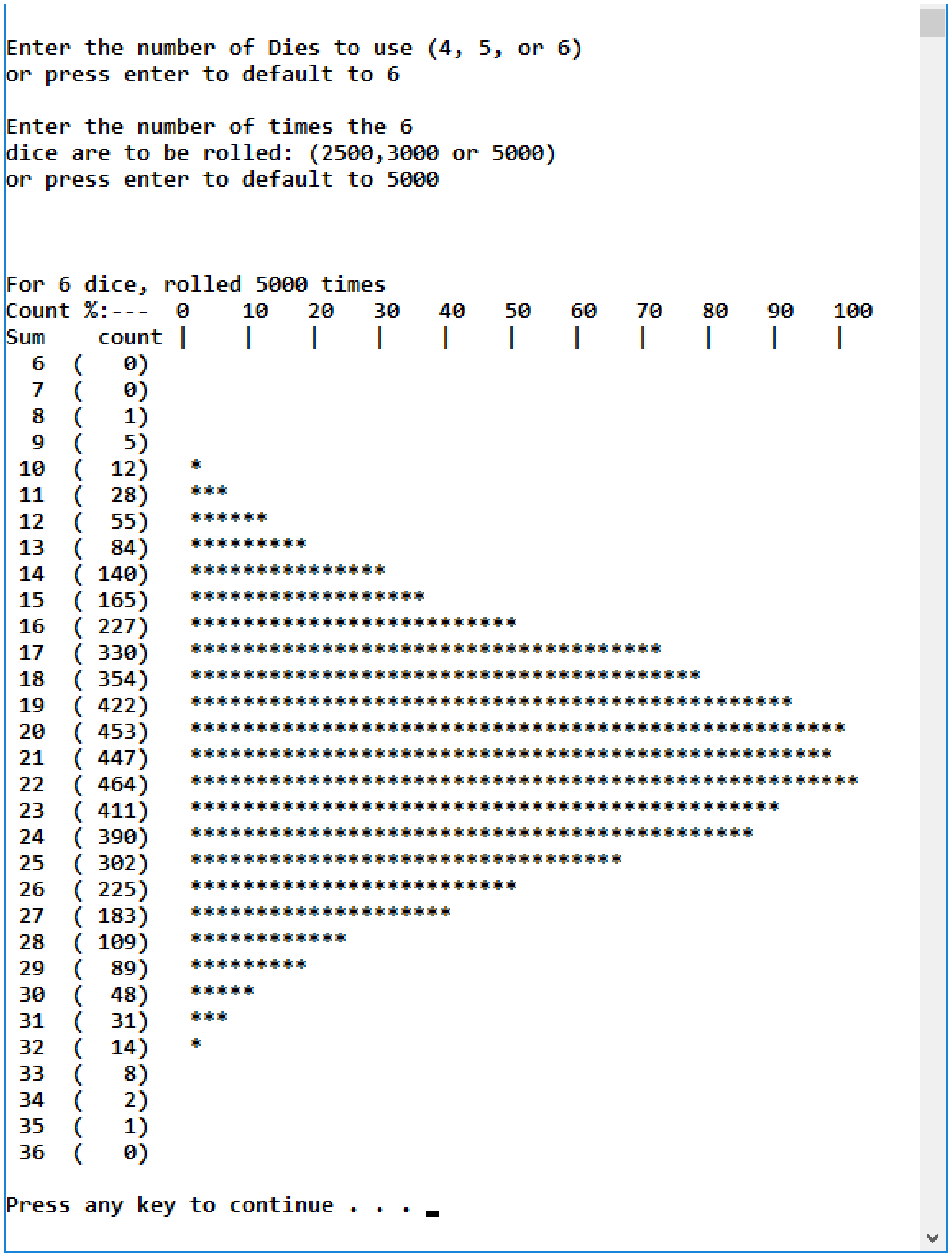
Write a program that uses the **die** class (**die.h** file is shown below), you must implement this six-sided die class in the **die.cpp** file. Your program will prompt the user to enter the number of dies in a set (4 through 6) that will be rolled together. The sum of the faces on the set will be the index to an array that holds the number of times this sum has occurred. You must also prompt your user for the number of times the set will be rolled. (2500, 3000, or 5000 times).

Once the rolls are completed the program will display a bar graph, like the one below, showing the sum value, the number of times it was rolled, and a bar of asterisks each one representing 2% of the highest count recorded in this run. In the example below, the highest value is 464, therefor, each asterisk is 2% of 464 which, rounded down, is 9. So, for the highest value of 464 there are 49 asterisks.

Except for the values for the rolls counts, because these should be random based on the dies used in the rolls, your output should look identical to the mockup below.

Executable versions of this exercise are in the projects folders under _Solutions, so you can see the results of various trials.

```cpp
#pragma once
class die
{
public:
    die();
    //Default constructor
    //Sets the default number rolled by a die to 1

    void roll();
    //Function to roll a die.
    //This function uses a random number generator to randomly
    //generate a number between 1 and 6, and stores the number
    //in the instance variable num.

    int getNum() const;
    //Function to return the number on the top face of the die.
    //Returns the value of the instance variable num.

private:
    int num; //The value of this instance ( 1 through 6variable num.
};

void die::roll()
{

}

int die::getNum() const
{

}
```



```
Enter the number of Dies to use (4, 5, or 6)
or press enter to default to 6

Enter the number of times the 6
dice are to be rolled: (2500,3000 or 5000)
or press enter to default to 5000



For 6 dice, rolled 5000 times
Count %:---  0    10   20   30   40   50   60   70   80   90   100
Sum    count |    |    |    |    |    |    |    |    |    |    |
  6  (   0)
  7  (   0)
  8  (   1)
  9  (   5)
 10  (  12)   *
 11  (  28)   ***
 12  (  55)   ******
 13  (  84)   *********
 14  ( 140)   ***************
 15  ( 165)   ******************
 16  ( 227)   *************************
 17  ( 330)   ************************************
 18  ( 354)   ***************************************
 19  ( 422)   **********************************************
 20  ( 453)   **************************************************
 21  ( 447)   *************************************************
 22  ( 464)   ***************************************************
 23  ( 411)   *********************************************
 24  ( 390)   *******************************************
 25  ( 302)   *********************************
 26  ( 225)   *************************
 27  ( 183)   ********************
 28  ( 109)   ************
 29  (  89)   *********
 30  (  48)   *****
 31  (  31)   ***
 32  (  14)   *
 33  (   8)
 34  (   2)
 35  (   1)
 36  (   0)

Press any key to continue . . .
```

You may use the example code below as a starting point for this graph.

```
cout << "For " << # of dice << " dice, rolled " << # of rolls << " times" << endl;
cout << "Count %:---    0    10   20   30   40   50   60   70   80   90   100" << endl;
cout << "Sum    count   |....|....|....|....|....|....|....|....|....|....|" << endl;

cout << " "   << setw(2) << sum of dice faces ;          // Formatting so Sum and count
columns
cout << "  (" << setw(4) << # of times rolled << ")   "; // are neat and tidy.
```

Exercise L1-2 Chapter 11 Inheritance & Composition

This exercise is designed so each patient can have multiple doctors, and multiple bills. The details are listed below:

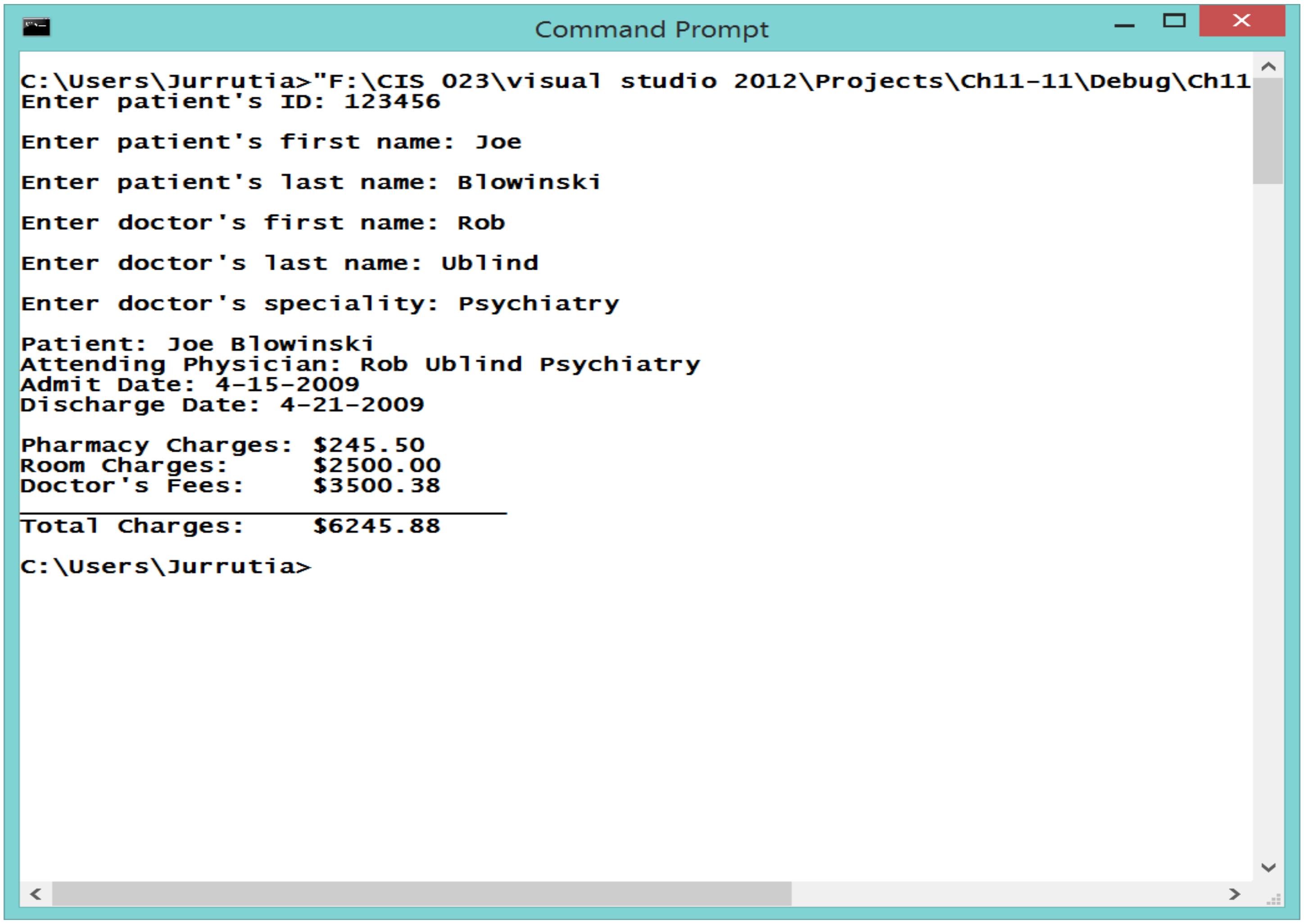
A. Design the class doctorType, inherited from the class personType, given below. Add a data member to store the doctors specialty. Also add appropriate constructors and member functions to initialize, access, and manipulate the data members.

```
#pragma once
#include <string>

using namespace std;

class personType
{
public:
    void print() const;
      //Function to output the first name and last name
      //in the form firstName lastName.

    void setName(string first, string last);
      //Function to set firstName and lastName according
      //to the parameters.
      //Postcondition: firstName = first; lastName = last

    string getFirstName() const;
      //Function to return the first name.
      //Postcondition: The value of the data member firstName
            //             is returned.

    string getLastName() const;
      //Function to return the last name.
      //Postcondition: The value of the data member lastName
            //             is returned.

    personType(string first = "", string last = "");
      //constructor
      //Sets firstName and lastName according to the parameters.
      //The default values of the parameters are empty strings.
      //Postcondition: firstName = first; lastName = last
 private:
    string firstName; //variable to store the first name
    string lastName;  //variable to store the last name
};

void personType::print() const
{
    cout << firstName << " " << lastName;
}

void personType::setName(string first, string last)
{
    firstName = first;
    lastName = last;
}

string personType::getFirstName() const
{
    return firstName;
}

string personType::getLastName() const
{
    return lastName;
}

//constructor
personType::personType(string first, string last)
{
   firstName = first;
   lastName = last;
}
```

B. Designed the class billType with data members to store a patients ID and a patient's hospital charges, such as pharmacy charges for medicine, Doctors fee, and room charges. Add appropriate constructors and member functions to initialize, access, and manipulate the data members.

C. Design the class patientType, inherited from the class personType, with additional data members to store the patients ID, the age, date of birth, attending Physicians name, the date when the patient was admitted in the hospital, and the date when the patient was discharged from the hospital. (use the class dateType to store the date of birth, admit date, discharge date, and the class doctorType to store to be attending Physicians name.) Add appropriate constructors and member functions to initialize, access, and manipulate the data members.

D. Write a program to test your classes. You can check your class definitions by using **L1-2CmplTest.cpp** found in the **L1-2** folder.



```
C:\Users\Jurrutia>"F:\CIS 023\visual studio 2012\Projects\Ch11-11\Debug\Ch11
Enter patient's ID: 123456

Enter patient's first name: Joe

Enter patient's last name: Blowinski

Enter doctor's first name: Rob

Enter doctor's last name: Ublind

Enter doctor's speciality: Psychiatry

Patient: Joe Blowinski
Attending Physician: Rob Ublind Psychiatry
Admit Date: 4-15-2009
Discharge Date: 4-21-2009

Pharmacy Charges: $245.50
Room Charges:     $2500.00
Doctor's Fees:    $3500.38
______________________________
Total Charges:    $6245.88

C:\Users\Jurrutia>
```

Class members

doctorType Class *must contain at least these functions*

```
doctorType(string first, string last, string spl); //First Name, Last Name,
Specialty
```

```
void print() const; //Formatted Display First Name, Last Name, Specialty
void setSpeciality(string); //Set the doctor’s Specialty
string getSpeciality(); //Return the doctor’s Specialty
```

<u>patientType Class</u> *must contain at least these functions*

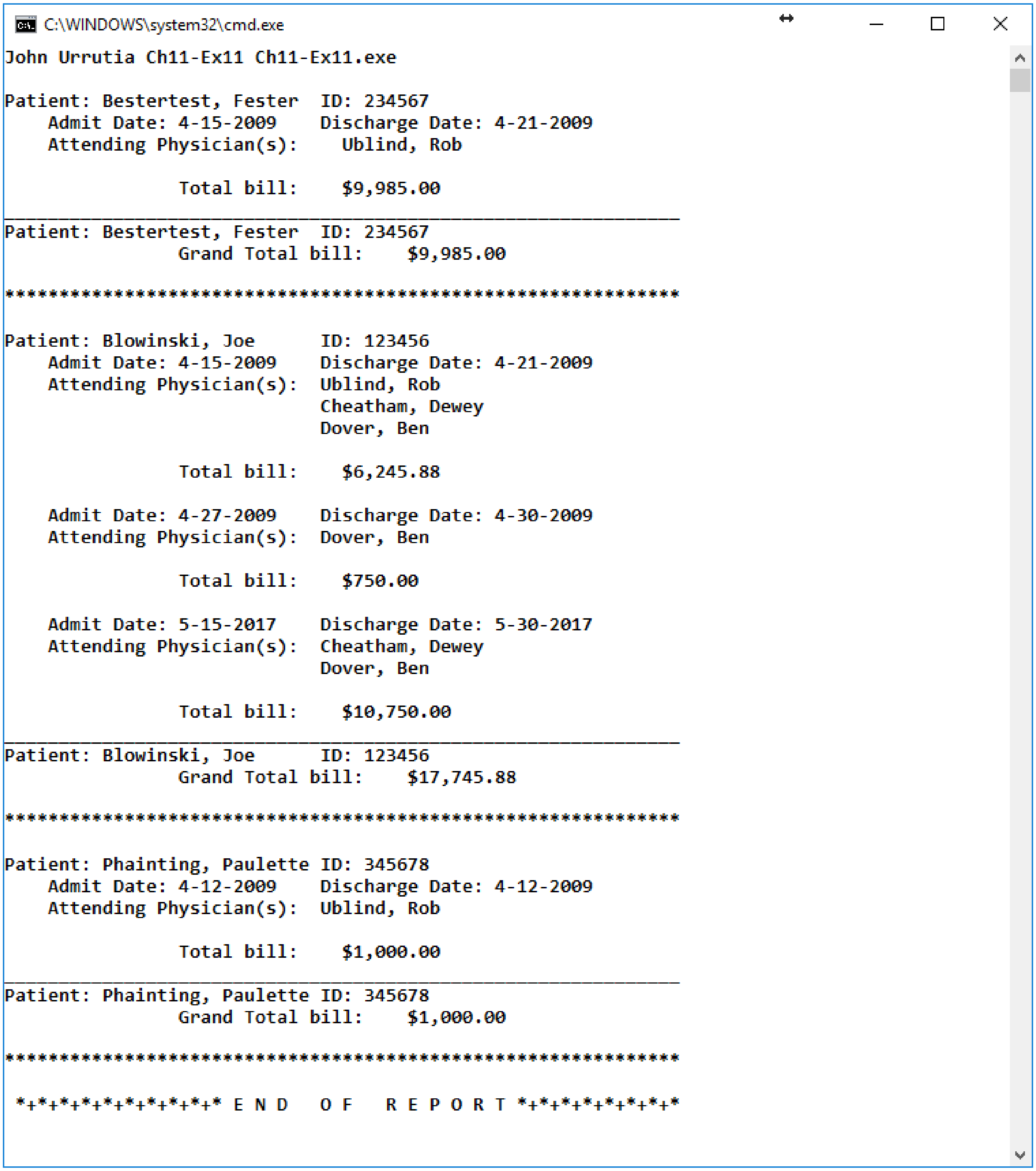
```
void   setInfo(
   string id, string fName, string lName,
   int bDay, int bMth, int bYear,
   string docFrName, string docLaName, string docSpl,
   int admDay, int admMth, int admYear,
   int disChDay, int disChMth, int disChYear);
void   setID(string);
string getID();
void   setBirthDate(int dy, int mo, int yr);
int    getBirthDay();
int    getBirthMonth();
int    getBirthYear();
void   setDoctorName(string fName, string lName);
void   setDoctorSpl(string);
string getDoctorFName();
string getDoctorLName();
string getDoctorSpl();
void   setAdmDate(int dy, int mo, int yr);
int    getAdmDay();
int    getAdmMonth();
int    getAdmYear();
void   setDisDate(int dy, int mo, int yr);
int    getDisDay();
int    getDisMonth();
int    getDisYear();
```

billType Class *must contain at least these functions*

```
billType(string id = "",          //Patient ID
         double phCharges = 0,   //Pharmacy Charges
         double rRent = 0,       //Room rental
         double docFee = 0);           //Doctor Fees

void   printBill() const;

void   setInfo(string,            //Patient ID
         double,           //Pharmacy Charges
         double,           //Room rental
         double);          //Doctor Fees

void   setID(string);

string getID();

void   setPharmacyCharges(double);

double getPharmacyCharges();

void   updatePharmacyCharges(double);

void   setRoomRent(double);

double getRoomRent();

void   updateRoomRent(double);

void   setDoctorFee(double);

double getDoctorFee();

void   updateDoctorFee(double);

double billingAmount();
```

The instructors test program will construct arrays of 10 **`doctorType`**, 10 **`patientType`** and 20 **`billType`** instances and from this data will produce the report shown on the next page. All patients will have 1 or more billings. Make sure you validate your three classes against all the functions shown above. The instructor output will be billings by patient and will provide summary data similar to the screen shot on the next page. If you need help with using this for your testing contact the instructor.

20pts.(all or nothing) Extra credit if you can duplicate the report below.



```
John Urrutia Ch11-Ex11 Ch11-Ex11.exe

Patient: Bestertest, Fester  ID: 234567
    Admit Date: 4-15-2009    Discharge Date: 4-21-2009
    Attending Physician(s):    Ublind, Rob

                Total bill:    $9,985.00

______________________________________________________________
Patient: Bestertest, Fester  ID: 234567
                Grand Total bill:    $9,985.00

**************************************************************

Patient: Blowinski, Joe      ID: 123456
    Admit Date: 4-15-2009    Discharge Date: 4-21-2009
    Attending Physician(s):  Ublind, Rob
                             Cheatham, Dewey
                             Dover, Ben

                Total bill:    $6,245.88

    Admit Date: 4-27-2009    Discharge Date: 4-30-2009
    Attending Physician(s):  Dover, Ben

                Total bill:    $750.00

    Admit Date: 5-15-2017    Discharge Date: 5-30-2017
    Attending Physician(s):  Cheatham, Dewey
                             Dover, Ben

                Total bill:    $10,750.00

______________________________________________________________
Patient: Blowinski, Joe      ID: 123456
                Grand Total bill:    $17,745.88

**************************************************************

Patient: Phainting, Paulette ID: 345678
    Admit Date: 4-12-2009    Discharge Date: 4-12-2009
    Attending Physician(s):  Ublind, Rob

                Total bill:    $1,000.00

______________________________________________________________
Patient: Phainting, Paulette ID: 345678
                Grand Total bill:    $1,000.00

**************************************************************

 *+*+*+*+*+*+*+*+*+* E N D   O F   R E P O R T *+*+*+*+*+*+*+*
```

Exercise L1-3 Chapter 12 Pntrs, ADT Classes, Virt. Func.,and lists

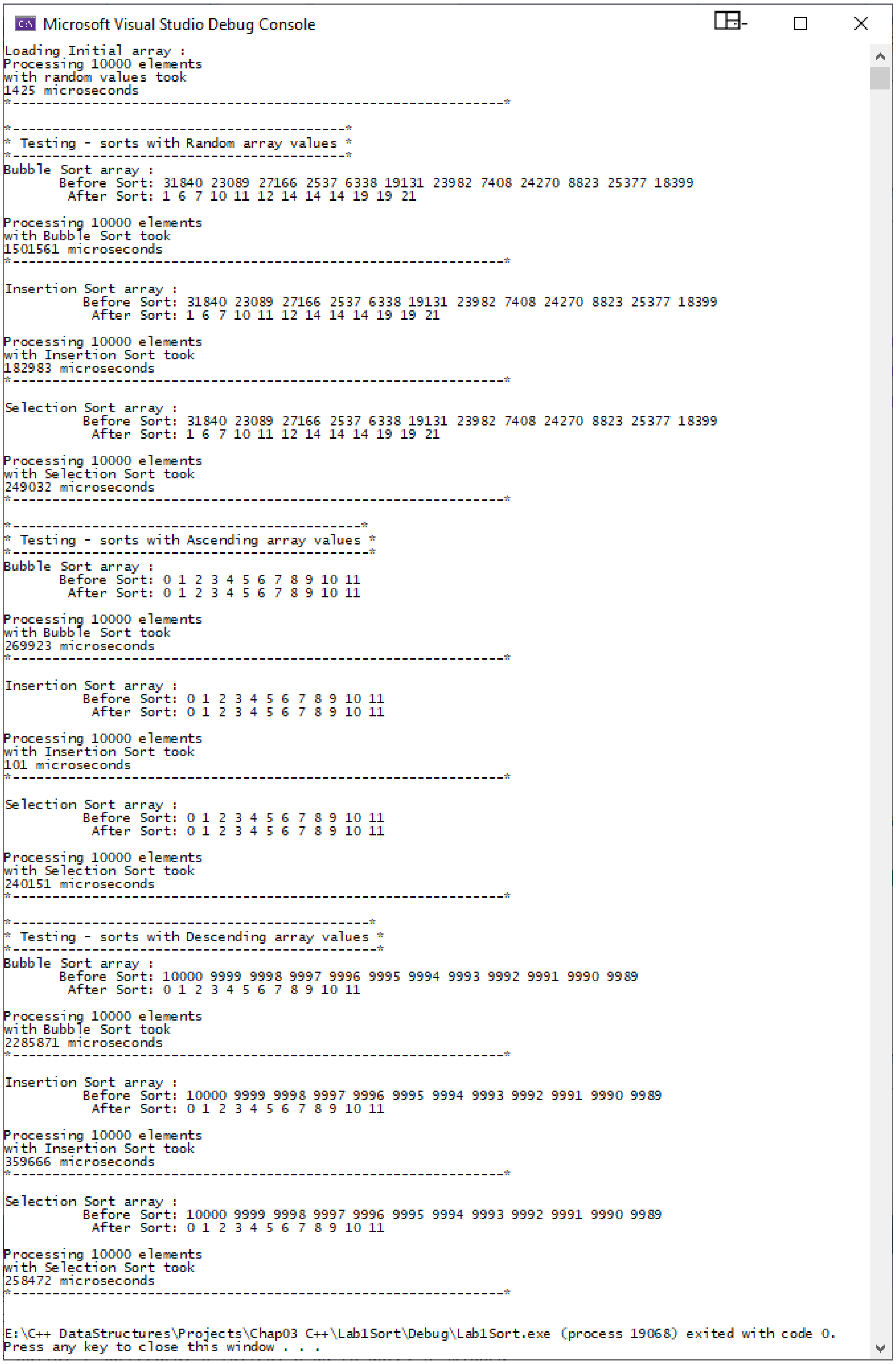
In this exercise use the program below (ClockTime.cpp) as a starting point for the timing function. Create a program (ch3SortTest.cpp) that will create an array of 10,000 random integers.

A. Sort the array with each of the simple sorting algorithms and record the elapsed time for each sorting algorithm and display the results.
B. Re-load the array with sequential numbers in ascending order and repeat step A.
C. Re-load the array with sequential numbers in descending order and repeat step A.

Your output should look like the screenshot below. Your timing values will vary based on the type of system you use for this test.

```
#include <iostream>
#include <ctime>
#include <cmath>
using namespace std;

int main ()
{
	float x,y;
	clock_t time_req;

	// Using pow function
	time_req = clock();		//Get the Starting Time in Microseconds
	for(int i=0; i<100000; i++)
	{
		y = log(pow(i,5));
	}
	time_req = clock() - time_req;		// Get the Ending Time in Microseconds
	cout << "Using pow function, it took " << (float)time_req/CLOCKS_PER_SEC << "
seconds" << endl;

	// Without pow function
	time_req = clock();
	for(int i=0; i<100000; i++)
	{
		y = log(i*i*i*i*i);
	}
	time_req = clock()- time_req;
```

```
        cout << "Without using pow function, it took " <<
(float)time_req/CLOCKS_PER_SEC << " seconds" << endl;

        return 0;
}
```



```
Loading Initial array :
Processing 10000 elements
with random values took
1425 microseconds
*--------------------------------------------------------------*

*------------------------------------------*
* Testing - sorts with Random array values *
*------------------------------------------*
Bubble Sort array :
       Before Sort: 31840 23089 27166 2537 6338 19131 23982 7408 24270 8823 25377 18399
        After Sort: 1 6 7 10 11 12 14 14 14 19 19 21

Processing 10000 elements
with Bubble Sort took
1501561 microseconds
*--------------------------------------------------------------*

Insertion Sort array :
          Before Sort: 31840 23089 27166 2537 6338 19131 23982 7408 24270 8823 25377 18399
           After Sort: 1 6 7 10 11 12 14 14 14 19 19 21

Processing 10000 elements
with Insertion Sort took
182983 microseconds
*--------------------------------------------------------------*

Selection Sort array :
          Before Sort: 31840 23089 27166 2537 6338 19131 23982 7408 24270 8823 25377 18399
           After Sort: 1 6 7 10 11 12 14 14 14 19 19 21

Processing 10000 elements
with Selection Sort took
249032 microseconds
*--------------------------------------------------------------*

*---------------------------------------------*
* Testing - sorts with Ascending array values *
*---------------------------------------------*
Bubble Sort array :
       Before Sort: 0 1 2 3 4 5 6 7 8 9 10 11
        After Sort: 0 1 2 3 4 5 6 7 8 9 10 11

Processing 10000 elements
with Bubble Sort took
269923 microseconds
*--------------------------------------------------------------*

Insertion Sort array :
          Before Sort: 0 1 2 3 4 5 6 7 8 9 10 11
           After Sort: 0 1 2 3 4 5 6 7 8 9 10 11

Processing 10000 elements
with Insertion Sort took
101 microseconds
*--------------------------------------------------------------*

Selection Sort array :
          Before Sort: 0 1 2 3 4 5 6 7 8 9 10 11
           After Sort: 0 1 2 3 4 5 6 7 8 9 10 11

Processing 10000 elements
with Selection Sort took
240151 microseconds
*--------------------------------------------------------------*

*----------------------------------------------*
* Testing - sorts with Descending array values *
*----------------------------------------------*
Bubble Sort array :
       Before Sort: 10000 9999 9998 9997 9996 9995 9994 9993 9992 9991 9990 9989
        After Sort: 0 1 2 3 4 5 6 7 8 9 10 11

Processing 10000 elements
with Bubble Sort took
2285871 microseconds
*--------------------------------------------------------------*

Insertion Sort array :
          Before Sort: 10000 9999 9998 9997 9996 9995 9994 9993 9992 9991 9990 9989
           After Sort: 0 1 2 3 4 5 6 7 8 9 10 11

Processing 10000 elements
with Insertion Sort took
359666 microseconds
*--------------------------------------------------------------*

Selection Sort array :
          Before Sort: 10000 9999 9998 9997 9996 9995 9994 9993 9992 9991 9990 9989
           After Sort: 0 1 2 3 4 5 6 7 8 9 10 11

Processing 10000 elements
with Selection Sort took
258472 microseconds
*--------------------------------------------------------------*


E:\C++ DataStructures\Projects\Chap03 C++\Lab1Sort\Debug\Lab1Sort.exe (process 19068) exited with code 0.
Press any key to close this window . . .
```