**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.

#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()

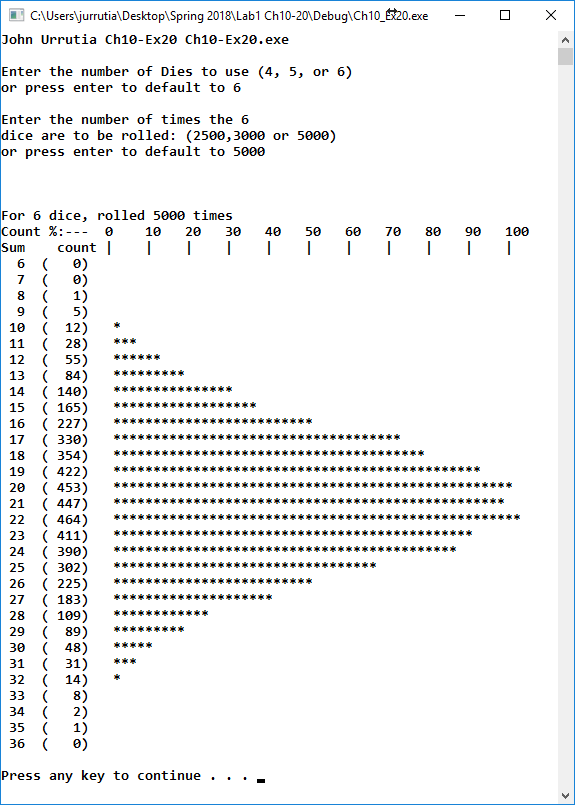
{

}

int die::getNum() const

{

}



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:

1. 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)

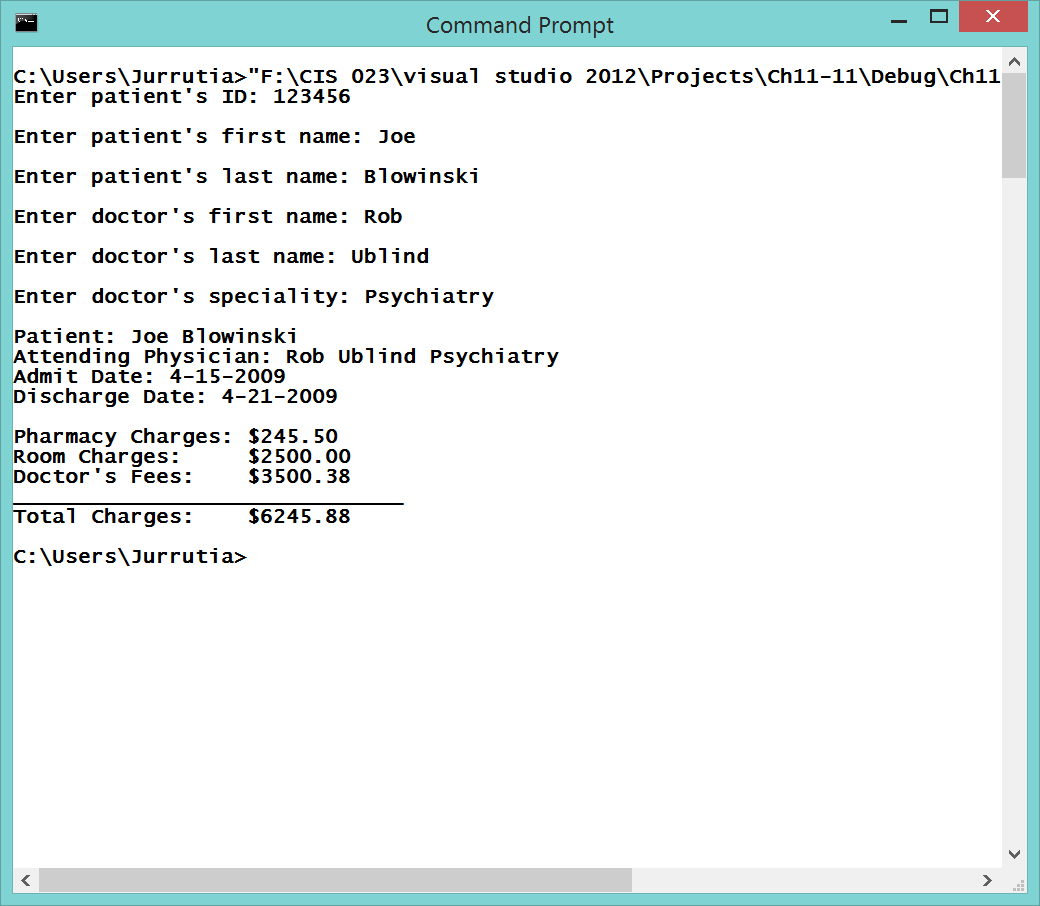
{

firstName = first;

lastName = last;

}

1. 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.
2. 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.
3. Write a program to test your classes. You can check your class definitions by using **L1-2CmplTest.cpp** found in the **L1-2** folder.



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

**patientType Class** *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);

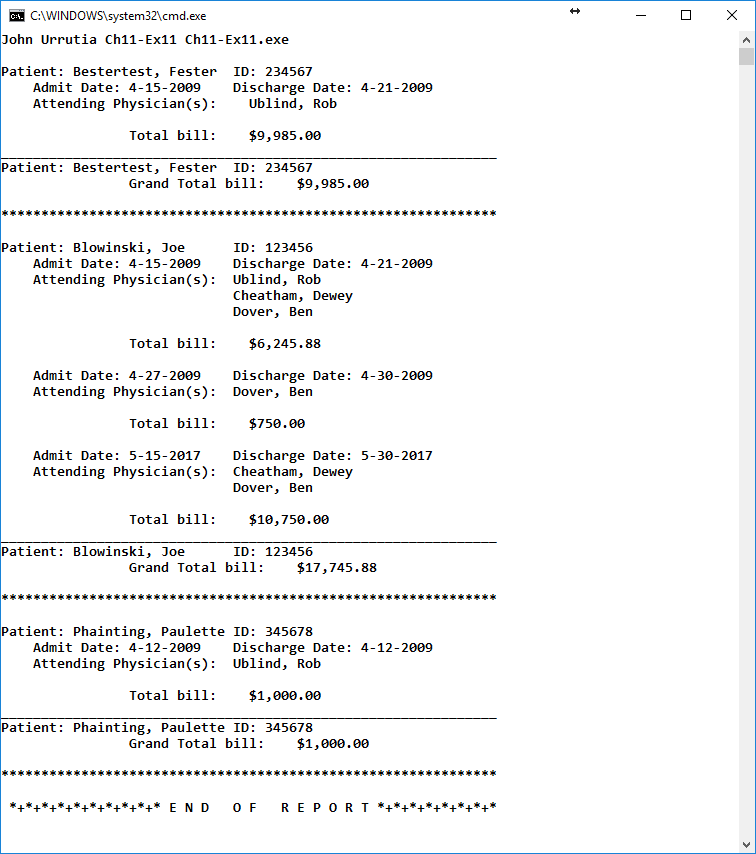
double getDoctorFee();

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.**



**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.

1. Sort the array with each of the simple sorting algorithms and record the elapsed time for each sorting algorithm and display the results.
2. Re-load the array with sequential numbers in ascending order and repeat step A.
3. 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;

}

