```cpp
#include <string>
#include <iostream>
#include <fstream>
using namespace std;

class patient_record
{

	public:
		int age;
		string name;
		double charge;
};


class Patient_Info
{
public:
	Patient_Info(); //default constructor; initialize DB
					//with the data stored in the file "Patient.txt"
	Patient_Info(const Patient_Info &); //copy constructor
	double total_charges(double threshold); //returns the sum of all the patient charges
	friend ostream & operator<<(ostream & out, const Patient_Info & HC);

private:
	int count;                  //number of elements stored in DB
	int capacity;               //number of memory cells (capacity) allocated to DB
	patient_record *DB;  //dynamic array
};

//QUESTION: Implement the default constructor "Patient_Info".Initialize DB with the file
"Patient.txt"; the
//fields in the file are read in the following order : age, name, charge.Start with an initial
capacity
//of 100.  If DB becomes full, print a message stating that "No more records can be added because
DB is full".
//Note that DB is a dynamic array.


Patient_Info::Patient_Info()
{
	count = 0;
	capacity = 100;
	DB = new patient_record[capacity];

	ifstream in;

	in.open("Patient.txt");

	while (!in.eof())
	{
		in >> DB[count].age >> DB[count].charge >> DB[count].name;
		count++;
	}

	in.close();
}


//QUESTION:: Implement the function "total_charges".The function will return the sum of all the
charges
//for patients with a charge(cost) greater than or equal to the given threshold.

double Patient_Info::total_charges(double threshold)
{
	double total = 0;

	for (int i = 0; i < count; i++)
```

```
        {
                if (DB[i].charge >= threshold)
                {
                        total+=DB[i].charge;
                }
        }
        return total;
}

//Question:"  Implement the copy constructor "Patient_Info".

Patient_Info::Patient_Info(const Patient_Info & Org)
{
        count = Org.count;
        capacity = Org.capacity;
        DB = new patient_record[capacity];
        for (int i = 0; i < count; i++)
        {
                DB[i] = Org.DB[i];
        }
}

//QUESTION:: Implement the overloaded "operator<<" with chaining. This function will print
//all the fields of every patient_record stored in DB to the screen.

ostream & operator<<(ostream & out, const Patient_Info & HC)
{
        for (int i = 0; i < HC.count; i++)
        {
                out << HC.DB[i].age << "\t"
                        << HC.DB[i].charge << "\t"
                        << HC.DB[i].name << endl;
        }
        return out;
}

//QUESTION::  Write a string function that accepts a string as an argument and returns its
reverse.
//Examples: if the string contains "abc" then the function returns "cba"; if  the string contains
"a"
//then the function returns "a"; and if the string contains "dmck", then the function returns
"kcmd".
//Remember, the function has one string formal parameter. You are expected to name and declare
the function,
//the formal parameter and any variables the function may use.  Implement this function below:

string & reverse(string & s)
{
        if (s.length() <= 1)
        {
                return s;
        }
        else
        {

                return s[s.length() - 1)] + reverse(s.substr(0, s.length() - 1)); //recusive call
        }
}



//QUESTION:: Define a class (give the class declaration) for a type called "CounterType".  An
object of this
//type is used to count things, so it records a count that is a nonnegative whole number. Include
a prototype
//for the default constructor that sets the counter to zero.  Also include prototypes for two
member functions
//to increase the count by 1 and to decrease the count by 1; you may call these whatever name you
wish.
```

```
class CounterType
{
public:
        CounterType() { count = 0; };
        void increase();
        void decrease();
private:
        int count;
};

//QUESTION:: Consider the following class declaration:
//class ABC
//{
//public:
//      QRS();
//      QRS(const QRS &);
//      void H();
//      int I(char & ch);
//      string & X(string & s);
//      QRS & operator+(char & ch);
//private:
//      int f, g, j;
//};
//
//Write a main function (driver) to test the class given above (give a statement to invoke every
function
//that has a prototype in the class).

int main()
{
        QRS A;  //QRS invoked -- default constructor
        char ch;
        string s;

        QRS B = A;  //copy constructor called
        A.H();  // member function H for object A invoked
        B.I(ch); // member function I for object B invoked
        cout << A.X(s) << endl;  //member function X for object A invoked returning a string
        B + 'a' + 'c' + ch;  //operator+ invoked by object B with chainings

        return 0;
}
```