Object Oriented Programming

Project Report

Problem Statement:

Mobile Finder Application

Develop an object oriented Mobile Finder Application in C++. The application should have the following features:

Inventory Management:

- Add/Edit/Delete mobile information records
- Permanent storage of mobile phones' data

Customized Searching:

- Get the user preferences (e.g. brand, type, price, RAM, OS, screen size, camera etc.) and display all the phones which satisfy the user requirements.
- The search result should also be saved in a file (named by User ID).

Sales Management:

- Generate, display and save each sale's invoice.
- Maintain a log of all the sales date wise.
- Generate a report of sales data over a particular period of time specified by start and end date. The report should also compute the total amount of all these sales.

Requirements:

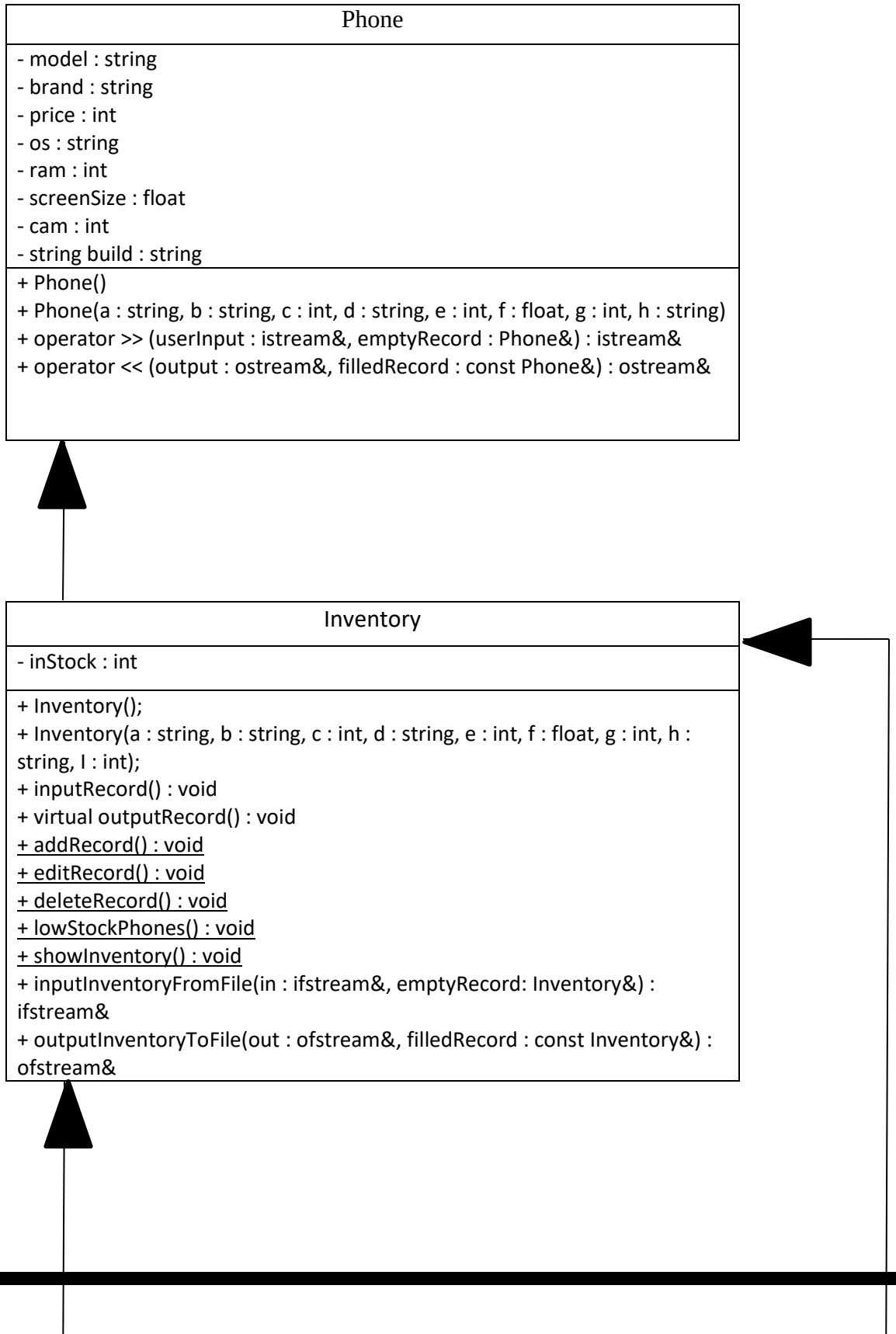
- Application should be well designed using object oriented concepts learned.
- Data should be displayed by overloading << and >> operators.
- You are allowed to design the classes as required.

Objectives:

- To build a user-friendly program that easily read, write large amounts of data and easily find required items.
- To show how to create a clean and well-organized object oriented program that uses most of the concepts studied in the course
- To show how to use separate compilation so that each class can be stored in a separate header file and all functions of all classes can be defined in just one cpp file.
- To show how different operations in a problem statement can be converted into class member functions.
- To use the four pillars of OOP (abstraction, encapsulation, inheritance, polymorphism) to solve a problem
- To identify which member functions of base class can be turned into virtual functions to allow static polymorphism
- To use filehandling to permanently store data and to update it as the program generates new data
- To read and write data to files in a uniform manner to allow fast and effortless data handling.
- To show how to generate data reports from existing data.
- To use functions to manipulate input and output functions
- To use functions to pick time directly from the system
- To account for exceptions and to write conditions to handle those exceptions.
- To show how to use multi-level class inheritance to meet specified requirements
- To show how to overload “<<” and “>>” operators to display class objects

(Note: MS word was acting strange due to the following UML diagram, you might receive the file with all the formatting messed up)

UML diagram:


Phone
- model : string
- brand : string
- price : int
- os : string
- ram : int
- screenSize : float
- cam : int
- string build : string
+ Phone()
+ Phone(a : string, b : string, c : int, d : string, e : int, f : float, g : int, h : string)
+ operator >> (userInput : istream&, emptyRecord : Phone&) : istream&
+ operator << (output : ostream&, filledRecord : const Phone&) : ostream&
Inventory
- inStock : int
+ Inventory();
+ Inventory(a : string, b : string, c : int, d : string, e : int, f : float, g : int, h : string, I : int);
+ inputRecord() : void
+ virtual outputRecord() : void
+ addRecord() : void
+ editRecord() : void
+ deleteRecord() : void
+ lowStockPhones() : void
+ showInventory() : void
+ inputInventoryFromFile(in : ifstream&, emptyRecord: Inventory&) : ifstream&
+ outputInventoryToFile(out : ofstream&, filledRecord : const Inventory&) : ofstream&

Search
- userID : string
+ Search(a : string) + Search(a : string, b : string, c : int, d : string, e : int, f float, g : int, h : string, i : int, j : string) + virtual outputRecord() : void + byPreference() : void + outputSearchToFile(out : ofstream&, filledRecord: const Search&): ofstream&

Sales
- date : int - customerName : string - contactNum : string - cardNum : string
+ Sales() + Sales(a : string, b : string, c : int, d : string, e : int, f : float, g : int, h : string, I : int, j : string, k : string, l : string) + getInput() : void + virtual outputRecord() : void + buyPhone() : void + findInvoices() : void + generateReport() : void + inputSalesFromFile(in : ifstream&, emptyRecord : Sales&) : ifstream& + outputSalesToFile(out : ofstream&, filledRecord : const Sales&) : ofstream&

Note:

I've added copies of the data files in the project folder. You can use those in case data gets messed up during testing.

Design:

In such a problem where all involved objects are so different from each other it becomes really diificult to implement the is-a relation in its true sense. With that being said, I've implemented the design that made the most sense.

There are 4 main entities in this problem; phone, inventory, search and sales.

In my program, I've created a class for phone and it acts as a base class to all other classes. That's because you can't have inventory, search and sales features if there isn't any phone data.

Secondly, I've created a class for inventory and it inherits from the phone class, because otherwise it can't perform its functions.

Lastly, I've created classes for search and sales. Both inherit from Inventory class because both of these aspects depend on what is available in the inventory.

Extra features added:

- Showing low-stock phones
- Showing entire inventory
- Searching phones based on a single aspect (instead of comparing entire object). This provides much better search results
- Finding invoices

Justification for disabling the warning and using localtime() function. The function is considered unsafe because it does not take a parameter for buffer size and might write in memory blocks that are holding important data. In my program, I use that function only once (when an object is created). As I'm using it only once, the buffer never gets full and hence, it's relatively safe to use.

Also, I tried using the safe versions of the function, localtime_s() and localtime_r() but I couldn't figure out how to make them work.

Source Code:

phone.h header file:

```
#pragma once
#include <iostream>
#include <string>

using namespace std;


class Phone
{
protected:
	string model;
	string brand;
	int price;
	string os;
	int ram;	//mentioning RAM in GBs is the norm these days. Also, when taking input, user is explicitly told to enter the value in GBs
	float screenSize;	//screenSizes usually are measured in inches and to the precision of 0.1 hence, I've made this variable a float
	int cam;	//mentioning camera resolution in Megapixels is the norm these days. Also, when taking input, user is explicitly told to enter the value in MPs
	string build;

public:
	Phone();
	Phone(string a, string b, int c, string d, int e, float f, int g, string h);

	friend istream& operator >> (istream& userInput, Phone& emptyRecord);
	//operator overloaded functions for taking input and displaying object values
	friend ostream& operator << (ostream& output, const Phone& filledRecord);
	//const objects can't be editted

};
```

inventory.h header file:

```
#pragma once
#include <iostream>
#include "phone.h"

using namespace std;

class Inventory : public Phone
{
protected:
	int inStock;	//number of articles in stock of each phone

public:
	Inventory();
	Inventory(string a, string b, int c, string d, int e, float f, int g, string h, int i);
	void inputRecord();
	virtual void outputRecord();
	static void addRecord();
	static void editRecord();
	static void deleteRecord();
	static void lowStockPhones();	//This is a special feature made just for the vendor so that they can quickly find the models with low stock
	static void showInventory();	//vendor can use this feature to quickly see the entire inventory

	friend ifstream& inputInventoryFromFile(ifstream& in, Inventory& emptyRecord);
	friend ofstream& outputInventoryToFile(ofstream& out, const Inventory& filledRecord);
		//functions can't modify const objects

};
```

search.h header file:

```
#pragma once
#include <iostream>
#include <string>
#include "inventory.h"

using namespace std;

class Search : public Inventory
{
private:
	string userId;

public:
	Search(string a);
	                        //Can't make an object of search without giving UserID
	Search(string a, string b, int c, string d, int e, float f, int g, string h, int i,
string j);
	virtual void outputRecord();
	                //function overriding
	void byPreference();
	                //Making just one object of Phone class and taking user input in that to
search for phones would have actually been a lot easier for me to implement but it won't be very
useful for the user in a lot of cases (for example if the user wants to see all phones within a
specific price range)


	friend ofstream& outputSearchToFile(ofstream& out, const Search& filledRecord);
	//functions can't modify const objects

};
```

sales.h header file:

```
#pragma once
#include <iostream>
#include "inventory.h"

using namespace std;

class Sales : public Inventory
{
private:
	int date;
	string customerName;
	string contactNum;
	string cardNum;

public:
	Sales();
	Sales(string a, string b, int c, string d, int e, float f, int g, string h, int i, string
j, string k, string l);
	void getInput();
	virtual void outputRecord();
		//function overriding
	void buyPhone();
	void findInvoices();
		//special function to help vendor find invoices of a particular customer
	void generateReport();

	friend ifstream& inputSalesFromFile(ifstream& in, Sales& emptyRecord);
	friend ofstream& outputSalesToFile(ofstream& out, const Sales& filledRecord);
	//functions can't modify const objects


	//////////////////////////////////////mode 1 for customer mode 2 for management
};
```

implementation.cpp file:

```
#pragma warning (disable:4996)                    //So that compiler allows us to use
localtime() function in time.h library

#include <iostream>

#include <string>

#include <fstream>

#include <time.h>                                 //for date and time manipulation

#include <iomanip>

#include <stdlib.h>                               //to use system function

#include <windows.h>                              //to use sleep function

#include "phone.h"

#include "inventory.h"

#include "search.h"

#include "sales.h"


using namespace std;


char inventoryFile[] = { "InventoryData.txt" };   //Name of file that holds all
inventory info. Would have been a waste of memory if all Inventory objects had a copy of this

char invoiceFile[] = { "SalesInvoice.txt" };

char reportFile[] = { "SalesReport.txt" };



//Defining '>>' and '<<' friend operator overloaded functions declared in Phone class


istream& operator >> (istream& userInput, Phone& emptyRecord)
```

```
{
        cin.ignore();
        cout << "\n\nEnter model: ";
        getline(userInput, emptyRecord.model);
        cin.ignore();
        cout << "Enter brand: ";
        getline(userInput, emptyRecord.brand);
        cin.ignore();
        cout << "Enter price: ";
        userInput >> emptyRecord.price;
        cin.ignore();
        cout << "\nEnter OS: ";
        getline(userInput, emptyRecord.os);
        cin.ignore();
        cout << "Enter RAM size (in GBs): ";
        userInput >> emptyRecord.ram;
        cin.ignore();
        cout << "Enter screen size (in Inches): ";
        userInput >> emptyRecord.screenSize;
        cin.ignore();
        cout << "Enter camera magnitude (in MPs): ";
        userInput >> emptyRecord.cam;
        cin.ignore();
        cout << "Enter build: ";
        getline(userInput, emptyRecord.build);
```

```
	cin.ignore();
	return userInput;
}

ostream& operator << (ostream& output, const Phone& filledRecord)
{
	output << "\nModel: " << filledRecord.model;
	output << "\nBrand: " << filledRecord.brand;
	output << "\nPrice: Rs. " << filledRecord.price;
	output << "\nOS: " << filledRecord.os;
	output << "\nRAM: " << filledRecord.ram << " GB";
	output << "\nScreen size: " << filledRecord.screenSize << " Inches";
	output << "\nCamera: " << filledRecord.cam << " MP";
	output << "\nBuild: " << filledRecord.build;
	return output;
}



//Phone class functions defined below

Phone::Phone() : model(""), brand(""), price(0), os(""), ram(0), screenSize(0.0), cam(0), build("")
{}
```

```
Phone::Phone(string a, string b, int c, string d, int e, float f, int g, string h) : model(a), brand(b), price(c), os(d), ram(e), screenSize(f), cam(g), build(h)
{}



//Defining friend functions declared in Inventory class


ifstream& inputInventoryFromFile(ifstream& in, Inventory& emptyRecord)
{
	string temp;
	getline(in, emptyRecord.model);
	getline(in, emptyRecord.brand);
	getline(in, temp);
	emptyRecord.price = stoi(temp);
	getline(in, emptyRecord.os);
	getline(in, temp);
	emptyRecord.ram = stoi(temp);
	getline(in, temp);
	emptyRecord.screenSize = stof(temp);
	getline(in, temp);
	emptyRecord.cam = stoi(temp);
	getline(in, emptyRecord.build);
	getline(in, temp);
```

```
	emptyRecord.inStock = stoi(temp);

	return in;

}


ofstream& outputInventoryToFile(ofstream& out, const Inventory& filledRecord)
	//functions can't modify const objects

{

	out << endl << filledRecord.model

		<< endl << filledRecord.brand

		<< endl << filledRecord.price

		<< endl << filledRecord.os

		<< endl << filledRecord.ram

		<< endl << filledRecord.screenSize

		<< endl << filledRecord.cam

		<< endl << filledRecord.build

		<< endl << filledRecord.inStock;

	return out;

}





//Inventory class functions defined below


Inventory::Inventory() : Phone::Phone()
```

```
{
	inStock = 0;
}

Inventory::Inventory(std::string a, std::string b, int c, std::string d, int e, float f, int g, string h, int i) :
Phone::Phone(a, b, c, d, e, f, g, h)
{
	inStock = i;
}

void Inventory::inputRecord()
{
	cin >> *this;
	//calling operator overloaded function using "this" pointer
	cout << "Enter number of units in stock: ";
	cin >> inStock;
	cin.ignore();
}

void Inventory::outputRecord()
{
	cout << *this;
	//calling operator overloaded function
	cout << "\nUnits in stock: " << inStock;
}
```

```
void Inventory::addRecord()

{

	Inventory newRecord;
	//newRecord stores the info of the new phone that will be added to the file


	ofstream output(::inventoryFile, ios::app);	//Ensures new record
gets written in the end

	if (!output)

	{

		cout << "\n\nError opening file.\n";

	}

	else

	{

		newRecord.inputRecord();
	//Taking user input

		outputInventoryToFile(output, newRecord);	//Outputting new
record to inventory file


		output.close();

		cout << "\n\nRecord added successfully.";

	}

}


void Inventory::editRecord()

{
```

```
	string phoneModel, line;
	//PhoneModel stores the model user is searching (for editting), line stores line read from file, temp is just a temporary data holder

	Inventory oldRecord, newRecord;
	//oldRecord holds values of record to be editted. newRecord hold values that will replace oldRecord

	int check = 0, confirmation = 0;
	//check is used to check whether specified model is in file. confirmation is to confirm whether user wants to edit the record



	ifstream input(::inventoryFile, ios::in);

	if (!input)

	{

		cout << "\n\nError opening file.\n";

	}

	else

	{

		cout << "\n\nEnter the model you want to edit: ";

		cin >> phoneModel;

		getline(input, line);
	//Ignoring the empty line in start of file

		while (!input.eof())

		{

			inputInventoryFromFile(input, oldRecord);
	//reading records from file



			if (oldRecord.model == phoneModel)

			{
```

```
                check = 1;
                        //if specified model found, break loop

                break;

        }


    }

    input.close();


    if (check == 0)
                //operation cancelled if model not found

    {

        cout << "\n\nSpecified model does not exit in database.\n";

    }

    else

    {

        Inventory* old = &oldRecord;

        old->outputRecord();

        cout << "\n\nEnter 1 to edit this record or 0 to cancel.\n\n";          //taking user
confirmation

        cin >> confirmation;


        if (confirmation == 1)

        {

                cin.ignore();

                cout << "\nEnter new values.\n";
    //taking in new values

                newRecord.inputRecord();
```

```
ifstream in(::inventoryFile, ios::in);

ofstream out("NewInventoryFile.txt", ios::out);


if (!in)

{

        cout << "\n\nError opening file.\n";

}

else

{

        getline(in, line);
//Ignoring the empty line in start of file

        while (!in.eof())

        {

                getline(in, line);

                if (line == oldRecord.model)
//if line matches data "model" in oldRecord, store newRecord in file

                {

                        outputInventoryToFile(out, newRecord);
//outputting new record to file


                        for (int i = 0; i < 8; ++i)
//loop to skip oldRecord values in original inventory file

                        {

                                getline(in, line);

                        }

                }
```

```
                        else
                        {
                                out << endl << line;
//Storing line as-is
                        }
                }
                in.close();
                out.close();

                remove(::inventoryFile);
                int result = rename("NewInventoryFile.txt", ::inventoryFile);
//Deleting old file and renaming new file to old one
                if (result == 0)
                {
                        cout << "\n\nRecord editted successfully.";
                }
                else
                {
                        cout << "\n\nOperation failed.\n";
                }
            }
        }
        else
        {
            cout << "\n\nOperation cancelled.\n";
        }
```

```
                }

        }

}


void Inventory::deleteRecord()

{

        string phoneModel, line;
        //PhoneModel stores the model user is searching (for deletion), line stores line read from file,
temp is just a temporary data holder

        Inventory record;
                //record holds values of record to be deleted

        int check = 0, confirmation = 0;
        //check is used to check whether specified model is in file. confirmation is to confirm whether
user wants to delete the record


        ifstream input(::inventoryFile, ios::in);

        if (!input)

        {

                cout << "\n\nError opening file.\n";

        }

        else

        {

                cout << "\n\nEnter the model you want to delete: ";

                cin >> phoneModel;


                getline(input, line);
        //Ignoring the empty line in start of file
```

```
while (!input.eof())
{
	inputInventoryFromFile(input, record);


	if (record.model == phoneModel)
//setting check to 1 and breaking loop
	{
		check = 1;
		break;
	}
}
input.close();


if (check == 0)
		//operation cancelled if model not found
{
	cout << "\n\nSpecified model does not exit in database.\n";
}
else
{
	Inventory* data = &record;
	data->outputRecord();
	cout << "\n\nEnter 1 to delete this record or 0 to cancel.\n\n";
//taking user confirmation
	cin >> confirmation;
```

```
if (confirmation == 1)
{

        ifstream in(::inventoryFile, ios::in);
        ofstream out("NewInventoryFile.txt", ios::out);

        if (!in)
        {
                cout << "\n\nError opening file.\n";
        }
        else
        {
                getline(in, line);
//Ignoring the empty line in start of file
                while (!in.eof())
                {
                        getline(in, line);
                        if (line == record.model)
//if line matches data "model" in record, skip that line and the next 8 lines as well
                        {
                                for (int i = 0; i < 8; ++i)
//loop to skip record values in file
                                {
                                        getline(in, line);
                                }
                        }
```

```
                    else
                    {
                        out << endl << line;
//Storing line as-is
                    }
                }
                in.close();
                out.close();

                remove(::inventoryFile);
                int result = rename("NewInventoryFile.txt", ::inventoryFile);
//Deleting old file and renaming new file to old one
                if (result == 0)
                {
                    cout << "\n\nRecord deleted successfully.";
                }
                else
                {
                    cout << "\n\nOperation failed.\n";
                }
            }
        }
        else
        {
            cout << "\n\nOperation cancelled.\n";
        }
```

```
                }
        }
}


void Inventory::lowStockPhones()
{
        string line;
        Inventory record;
        Inventory* recordPointer;
        int check = 0;
        //used to check whether there are any low-stock phones or not


        ifstream in(::inventoryFile, ios::in);
        if (!in)
        {
                cout << "\nError opening file.\n";
        }
        else
        {
                getline(in, line);
        //Ignoring the empty line in start of file
                while (!in.eof())
                {
                        inputInventoryFromFile(in, record);


                        if (record.inStock <= 10)
```

```
                {
                        ++check;
                        //Helps ensure following if condition is executed only once
                        if (check == 1)
                        {
                                cout << "\n\nLow stock phone/s:\n\n";
                        }

                        recordPointer = &record;
                        recordPointer->outputRecord();
                        cout << endl;
                }
        }
        in.close();
        if (check == 0)
        {
                cout << "\nNo low-stock phones in inventory.\n";
        }
    }
}

void Inventory::showInventory()
{
    string line;
    Inventory record;
```

```
	Inventory* recordPointer;


	ifstream in(::inventoryFile, ios::in);
	if (!in)
	{
		cout << "\nError opening file.\n";
	}
	else
	{
		getline(in, line);
	//Ignoring the empty line in start of file
		while (!in.eof())
		{
			inputInventoryFromFile(in, record);
			recordPointer = &record;
			recordPointer->outputRecord();
			cout << endl;
		}
		in.close();
		Sleep(15000);
	//Enough time to see inventory


	}
}
```

```
//defining Search class function below


Search::Search(string a) : Inventory()

{

	userId = a + ".txt";

}


Search::Search(string a, string b, int c, string d, int e, float f, int g, string h, int i, string j) : Inventory(a, b, c, d, e, f, g, h, i)

{

	userId = j + ".txt";

}


void Search::outputRecord()	//function overriding (the reason behind overriding this function is that we don't want to output number of units in stock because that info is irrelevant to customer)

{

	cout << *this;

	if (inStock == 0)

	{

		cout << "\n\nSold out.\n\n";	//While we don't want to show inStock value to the customer, it still affects the customer and that's why Search class inherits from Inventory class

	}

}
```

//Defiding friend function declared in Search class

```
ofstream& outputSearchToFile(ofstream& out, const Search& filledRecord)
{
	out << endl << filledRecord.model
		<< endl << filledRecord.brand
		<< endl << filledRecord.price
		<< endl << filledRecord.os
		<< endl << filledRecord.ram
		<< endl << filledRecord.screenSize
		<< endl << filledRecord.cam
		<< endl << filledRecord.build << endl;		//not storing number of units in storage in the user's search results file because that info is not shown to the customer (because its irrelevant to customer)
	return out;
}
```

//Defining Search class functions

```
void Search::byPreference()
{
	Search record(userId);
	Inventory* recordPointer;
	int check = 0;
	int option = 0;
	string line;

	while (1)
	{

		cout << "\n\t\t\t\t\t\t      Enter 1 to search by model,\n\t\t\t\t\t\t\t  2 to search by brand,"
			<< "\n\t\t\t\t\t\t       3 to search by price range,\n\t\t\t\t\t\t\t    4 to search by OS,"
			<< "\n\t\t\t\t\t\t\t5 to search by RAM size,\n\t\t\t\t\t\t       6 to search by screen size,"
			<< "\n\t\t\t\t\t\t    7 to search by camera resultion,\n\t\t\t\t\t\t      8 to search by build
material,"
			<< "\n\t\t\t\t\t\t      or 0 to return to main menu.\n\n\t\t\t\t\t\t\t\t    ";
		cin >> option;

		if (option == 1)
		{
			string phoneModel;

			cout << "Enter phone model: ";
			cin >> phoneModel;
```

```
ifstream in(::inventoryFile, ios::in);
if (!in)
{
        cout << "\nError opening file.\n";
}
else
{
        getline(in, line);
//Ignoring the empty line in start of file
        while (!in.eof())
        {
                inputInventoryFromFile(in, record);
        //Although friend functions are not inherited, an object of Search class is
still able to work with a friend function of its parent class because compiler implicitly typecasts it to
Inventory

                if (record.model == phoneModel)
                {
                        check = 1;
                        recordPointer = &record;
                        recordPointer->outputRecord();
                        cout << endl;

                        ofstream out(userId, ios::app);
                        if (!out)
                        {
```

```
                                        cout << "\nError opening file.\n";
                                }
                                else
                                {
                                        outputSearchToFile(out, record);
                        //Storing users search results in file named by user's ID
                                }
                                out.close();
                                break;
                //model names are unique hence there will be only one phone to output and hence we can break the loop
                                }
                        }
                        in.close();
                        if (check == 0)
                        {
                                cout << "Specified model does not exist in database";
                        }
                }
        }

        else if (option == 2)
        {
                string phoneBrand;

                cout << "Enter phone brand: ";
```

```
cin >> phoneBrand;


ifstream in(::inventoryFile, ios::in);
if (!in)
{
    cout << "\nError opening file.\n";
}
else
{
    getline(in, line);
//Ignoring the empty line in start of file
    while (!in.eof())
    {
        inputInventoryFromFile(in, record);
    //Although friend functions are not inherited, an object of Search class is
still able to work with a friend function of its parent class because compiler implicitly typecasts it to
Inventory



        if (record.brand == phoneBrand)
        {
            check = 1;
            recordPointer = &record;
            recordPointer->outputRecord();
            cout << endl;


            ofstream out(userId, ios::app);
```

```
                    if (!out)
                    {
                        cout << "\nError opening file.\n";
                    }
                    else
                    {
                        outputSearchToFile(out, record);
            //Storing users search results in file named by user's ID
                    }
                    out.close();
                }
            }
            in.close();

            if (check == 0)
            {
                cout << "\nPhones of specified brand do not exist in database.\n";
            }
        }
    }

    else if (option == 3)
    {
        int  lowerEnd, higherEnd;                                   //lowerEnd and
higherEnd help define the price range. Instead of using just one variable to set the maximum limit I'm
also using a variable to set the minimum limit. This is important because without both limits, user will
```

get lots of irrelevant result. For exmple a user looking for a Rs. 60,000 phone will also be shown Rs. 5,000 phones

```
cout << "\nEnter upper price limit: ";
cin >> higherEnd;
cout << "\nEnter lower price limit: ";
cin >> lowerEnd;

ifstream in(::inventoryFile, ios::in);
if (!in)
{
    cout << "\nError opening file.\n";
}
else
{
    getline(in, line);
//Ignoring the empty line in start of file
    while (!in.eof())
    {
        inputInventoryFromFile(in, record);
    //Although friend functions are not inherited, an object of Search class is still able to work with a friend function of its parent class because compiler implicitly typecasts it to Inventory

        if (record.price < higherEnd && record.price > lowerEnd)
        {
            check = 1;
```

```
                    recordPointer = &record;
                    recordPointer->outputRecord();
                    cout << endl;

                    ofstream out(userId, ios::app);
                    if (!out)
                    {
                        cout << "\nError opening file.\n";
                    }
                    else
                    {
                        outputSearchToFile(out, record);
                //Storing users search results in file named by user's ID
                    }
                    out.close();
                }
            }
            in.close();

            if (check == 0)
            {
                cout << "\nPhones in specified price range do not exist in
database.\n";
            }
        }
    }
```

```
else if (option == 4)
{
        string phoneOS;

        cout << "Enter operating system: ";
        cin >> phoneOS;

        ifstream in(::inventoryFile, ios::in);
        if (!in)
        {
                cout << "\nError opening file.\n";
        }
        else
        {
                getline(in, line);
//Ignoring the empty line in start of file
                while (!in.eof())
                {
                        inputInventoryFromFile(in, record);
                //Although friend functions are not inherited, an object of Search class is
still able to work with a friend function of its parent class because compiler implicitly typecasts it to
Inventory

                        if (record.os == phoneOS)
                        {
                                check = 1;
```

```
                    recordPointer = &record;
                    recordPointer->outputRecord();
                    cout << endl;

                    ofstream out(userId, ios::app);
                    if (!out)
                    {
                        cout << "\nError opening file.\n";
                    }
                    else
                    {
                        outputSearchToFile(out, record);
            //Storing users search results in file named by user's ID
                    }
                    out.close();
                }
            }
            in.close();

            if (check == 0)
            {
                cout << "\nPhones with specified OS do not exist in database.\n";
            }
        }
```

```
}

else if (option == 5)
{
    int phoneRam;

    cin.ignore();
    cout << "Enter RAM (in GBs): ";
//Prices vary greatly according to RAM size hence we're using a limit so that results are relevant to the user. Also, the available RAM options are not that diverse so there's no need to use a range
    cin >> phoneRam;

    ifstream in(::inventoryFile, ios::in);
    if (!in)
    {
        cout << "\nError opening file.\n";
    }
    else
    {
        getline(in, line);
//Ignoring the empty line in start of file
        while (!in.eof())
        {
            inputInventoryFromFile(in, record);
        //Although friend functions are not inherited, an object of Search class is still able to work with a friend function of its parent class because compiler implicitly typecasts it to Inventory
```

```
        if (record.ram <= phoneRam)
        {
                check = 1;
                recordPointer = &record;
                recordPointer->outputRecord();
                cout << endl;

                ofstream out(userId, ios::app);
                if (!out)
                {
                        cout << "\nError opening file.\n";
                }
                else
                {
                        outputSearchToFile(out, record);
//Storing users search results in file named by user's ID
                }
                out.close();
        }
}
in.close();

if (check == 0)
{
```

```
                                        cout << "\nPhones with specified RAM limit do not exist in
database.\n";
                                }
                        }
                }

                else if (option == 6)
                {
                        float phoneScreenSize;

                        cin.ignore();
                        cout << "Enter screen size (in Inches): ";
                        cin >> phoneScreenSize;

                        ifstream in(::inventoryFile, ios::in);
                        if (!in)
                        {
                                cout << "\nError opening file.\n";
                        }
                        else
                        {
                                getline(in, line);
                //Ignoring the empty line in start of file
                                while (!in.eof())
                                {
```

```
inputInventoryFromFile(in, record);
//Although friend functions are not inherited, an object of Search class is still able to work with a friend function of its parent class because compiler implicitly typecasts it to Inventory


if (record.screenSize <= phoneScreenSize)
//Screen sizes don't vary that much hence we're using only one limit
{
    check = 1;
    recordPointer = &record;
    recordPointer->outputRecord();
    cout << endl;


    ofstream out(userId, ios::app);
    if (!out)
    {
        cout << "\nError opening file.\n";
    }
    else
    {
        outputSearchToFile(out, record);
//Storing users search results in file named by user's ID
    }
    out.close();
}
}
in.close();
```

```
                if (check == 0)
                {
                        cout << "\nPhones with specified screen size do not exist in
database.\n";
                }
        }
}

else if (option == 7)
{
        int phoneCam;

        cin.ignore();
        cout << "Enter camera resolution (in MPs): ";
        cin >> phoneCam;

        ifstream in(::inventoryFile, ios::in);
        if (!in)
        {
                cout << "\nError opening file.\n";
        }
        else
        {
                getline(in, line);
//Ignoring the empty line in start of file
```

```
while (!in.eof())
{
        inputInventoryFromFile(in, record);
//Although friend functions are not inherited, an object of Search class is still able to work with a friend function of its parent class because compiler implicitly typecasts it to Inventory


        if (record.cam <= phoneCam)
//Prices vary greatly according to camera resolution hence we're using a limit so that search results are relevant to the user
        {
                check = 1;
                recordPointer = &record;
                recordPointer->outputRecord();
                cout << endl;


                ofstream out(userId, ios::app);
                if (!out)
                {
                        cout << "\nError opening file.\n";
                }
                else
                {
                        outputSearchToFile(out, record);
//Storing users search results in file named by user's ID
                }
                out.close();
```

```
                }
            }
            in.close();

            if (check == 0)
            {
                cout << "\nPhones with specified camera resolution do not exist in
database.\n";
            }
        }
    }

    else if (option == 8)
    {
        string phoneBuild;

        cin.ignore();
        cout << "Enter build material: ";
        getline(cin, phoneBuild);

        ifstream in(::inventoryFile, ios::in);
        if (!in)
        {
            cout << "\nError opening file.\n";
        }
```

```
else
{
        getline(in, line);
//Ignoring the empty line in start of file
        while (!in.eof())
        {
                inputInventoryFromFile(in, record);
        //Although friend functions are not inherited, an object of Search class is
still able to work with a friend function of its parent class because compiler implicitly typecasts it to
Inventory

                if (record.build == phoneBuild)
                {
                        check = 1;
                        recordPointer = &record;
                        recordPointer->outputRecord();
                        cout << endl;

                        ofstream out(userId, ios::app);
                        if (!out)
                        {
                                cout << "\nError opening file.\n";
                        }
                        else
                        {
                                outputSearchToFile(out, record);
        //Storing users search results in file named by user's ID
```

```
                        }
                        out.close();
                    }
                }
                in.close();

                if (check == 0)
                {
                    cout << "\nPhones with specified camera resolution do not exist in
database.\n";
                }
            }
        }

        else if (option == 0)
        {
            break;
        }

        else
        {
            cout << "Please select a valid option";
        }
    }
}
```

```
//Defining friend functions declared in Sales class


ifstream& inputSalesFromFile(ifstream& in, Sales& emptyRecord)
{
	string temp;
	getline(in, emptyRecord.model);
	getline(in, emptyRecord.brand);
	getline(in, temp);
	emptyRecord.price = stoi(temp);
	getline(in, emptyRecord.os);
	getline(in, temp);
	emptyRecord.ram = stoi(temp);
	getline(in, temp);
	emptyRecord.screenSize = stof(temp);
	getline(in, temp);
	emptyRecord.cam = stoi(temp);
	getline(in, emptyRecord.build);
		//inStock value is not stored in invoice file hence we're not reading that from the file
	getline(in, temp);
	emptyRecord.date = stoi(temp);
	getline(in, emptyRecord.customerName);
```

```
	getline(in, emptyRecord.contactNum);
	getline(in, emptyRecord.cardNum);
	return in;
}

ofstream& outputSalesToFile(ofstream& out, const Sales& filledRecord)		//functions can't modify const objects
{
	out << endl << filledRecord.model
		<< endl << filledRecord.brand
		<< endl << filledRecord.price
		<< endl << filledRecord.os
		<< endl << filledRecord.ram
		<< endl << filledRecord.screenSize
		<< endl << filledRecord.cam
		<< endl << filledRecord.build
		<< endl << filledRecord.date
		<< endl << filledRecord.customerName
		<< endl << filledRecord.contactNum
		<< endl << filledRecord.cardNum;
	return out;
}
```

//Defining Sales class functions below

Sales::Sales()

{

time_t t = time(0); //time_t is a datatype used to hold values returned by functions like time(). time(0) is a function that returns the current calendar time as a value of type time_t.

tm* now = localtime(&t); //tm is a structure made for holding time and date info. The time_t datatype is hard to read and isn't portable, the function localtime() helps convert time_t values into a tm object so that values can be easily manipulated.

string year = to_string((now->tm_year + 1900)); //converting year value to a string

string month, day, currentDate;

if ((now->tm_mon + 1) >= 10)
//this if-else condition helps ensure we end up with a two letter string for month

{

month = to_string((now->tm_mon + 1));

}

else

{

month = "0" + to_string((now->tm_mon + 1));

}

if ((now->tm_mday) >= 10) //this if-else condition helps ensure we end up with a two letter string for day

```
	{
		day = to_string((now->tm_mday));
	}
	else
	{
		day = "0" + to_string((now->tm_mday));
	}


	currentDate = year + month + day;
	//storing year, month and day as one combined string. the string is stored in this particular order (YYYYMMDD) after being converted to an int, the year will be the most significant number and the day will be the last significant. This will help with the comparisons in report generation

	date = stoi(currentDate);
	//converting currentDate string to integer so that comparison can be carried out

}


Sales::Sales(string a, string b, int c, string d, int e, float f, int g, string h, int i, string j, string k, string l) : Inventory(a,b,c,d,e,f,g,h,i)

{

	time_t t = time(0);

	tm* now = localtime(&t);


	string year = to_string((now->tm_year + 1900));	//converting year value to a string

	string month, day, currentDate;


	if ((now->tm_mon + 1) >= 10)
	//this if-else condition helps ensure we end up with a two letter string for month
```

```
{
        month = to_string((now->tm_mon + 1));
}
else
{
        month = "0" + to_string((now->tm_mon + 1));
}

if ((now->tm_mday) >= 10)                                                    //this
if-else condition helps ensure we end up with a two letter string for day
{
        day = to_string((now->tm_mday));
}
else
{
        day = "0" + to_string((now->tm_mday));
}

currentDate = year + month + day;
//storing year, month and day as one combined string
date = stoi(currentDate);
//converting currentDate string to integer so that comparison can be carried out

customerName = j;
contactNum = k;
cardNum = l;
```

```
}

void Sales::getInput()
{
	cin.ignore();
	cout << "\n\nEnter full name: ";
	getline(cin, customerName);
	cin.ignore();
	cout << "Enter contact number: ";
	getline(cin, contactNum);
	cin.ignore();
	cout << "Enter credit/debit card number to confirm purchase: ";
	getline(cin, cardNum);
	cin.ignore();
}

void Sales::outputRecord()
	//function overriding
{
	cout << "\nCustomer Invoice\n\n";
	cout << *this;
	cout << "\nPurchased by: " << customerName;
	cout << "\nContact number: " << contactNum;
	//invoice doesn't carry customer's credit/debit card number
}
```

```
void Sales::buyPhone()
{
	int check = 0;
	//checks whether specified model is in file or not
	string phoneModel, line;
	Inventory* record;
	//pointer used to access output function


	cout << "\nEnter phone model: ";
	cin >> phoneModel;
	//no spaces


	ifstream in(::inventoryFile, ios::in);
	if (!in)
	{
		cout << "\nError opening file";
	}
	else
	{
		getline(in, line);
	//Ignoring the empty line in start of file
		while (!in.eof())
		{
			inputInventoryFromFile(in, *this);
	//implicit typecasting to Inventory (upcasting)
			if (model == phoneModel)
```

```
        {
                check = 1;
                break;
        }
}
in.close();


if (check == 0)
{
        cout << "\nSpecified model doesn't exist in database.\b";
}
else
{
        if (this->inStock == 0)
        {
                cout << "\n\nSorry, the phone has sold out.\n";
        }
        else
        {
                --(this->inStock);
//decrementing inStock number
                this->getInput();
//getting name, contact info and card number from user;
                record = this;
                record->outputRecord();
//accessing output function of Sales class
```

```
ofstream out(::invoiceFile, ios::app);

outputSalesToFile(out, *this);
//Storing invoice data

out.close();


ifstream input(::inventoryFile, ios::in);
//updating inStock number in inventory file

ofstream output("NewInventoryFile.txt", ios::out);


getline(input, line);
//Ignoring the empty line in start of file

while (!input.eof())

{

    getline(input, line);

    if (line == model)

    {

        output << endl << line;
    //outputting model and the next 7 data values as-is

        for (int i = 0; i < 7; ++i)

        {

            getline(input, line);

            output << endl << line;

        }

        getline(input, line);
//ignoring old inStock value
```

```
                    output << endl << inStock;
            //storing updated value
                }
                else
                {
                    output << endl << line;
                }
            }

            input.close();
            output.close();

            remove(::inventoryFile);
            int temp = rename("NewInventoryFile.txt", ::inventoryFile);
            if (temp == 0)
            {
                cout << "\n\nPurchase completed successfully.";
            }
            else
            {
                cout << "\n\nOperation failed.";
            }
        }
    }
}
```

```
}

void Sales::findInvoices()
{
	Inventory* record;
	Sales emptyRecord;
	string customerContactNumber, line;
	int check = 0;
	cout << "\nEnter customer's contact number: ";
	cin >> customerContactNumber;	//no
spaces
	cin.ignore();

	ifstream in(::invoiceFile, ios::in);

	if (!in)
	{
		cout << "\nError opening file.\n";
	}
	else
	{
		getline(in, line);
	//Ignoring the empty line in start of file
		while (!in.eof())
		{
			inputSalesFromFile(in, emptyRecord);
```

```
                if (emptyRecord.contactNum == customerContactNumber)
                {
                        check = 1;
                        record = &emptyRecord;
                        record->outputRecord();
                        cout << endl;
                }
        }
        in.close();

        if (check == 0)
        {
                cout << "\nNo records with specified contact number found.\n";
        }
        else
        {
                Sleep(7000);
    //Enough time to look at invoice
        }
    }
}

void Sales::generateReport()
{
    string temp;
```

```
        string startYear, endYear;                                  //used for taking date inputs
        string startMonth, endMonth;
        string startDay, endDay;
        int startingDate, endingDate;
        int totalRevenue = 0;                                       //stores amount of total sales
made in specified period
        Sales emptyRecord;                                          //used for reading data
from invoice file, and for writing data to report file

        cout << "\nEnter starting date of report (in DD/MM/YYYY format): ";
        cin >> setw(2) >> startDay;
        cin.ignore();                                   //ignore '/'
        cin >> setw(2) >> startMonth;
        cin.ignore();                                   //ignore '/'
        cin >> setw(4) >> startYear;
        cin.ignore();                                   //ignore '/'

        temp = startYear + startMonth + startDay;
        startingDate = stoi(temp);


        cout << "\nEnter ending date of report (in DD/MM/YYYY format): ";
        cin >> setw(2) >> endDay;
        cin.ignore();                                   //ignore '/'
        cin >> setw(2) >> endMonth;
        cin.ignore();                                   //ignore '/'
```

```
	cin >> setw(4) >> endYear;
	cin.ignore();					//ignore '/'

	temp = endYear + endMonth + endDay;
	endingDate = stoi(temp);

	ifstream in(::invoiceFile, ios::in);
	ofstream out(::reportFile, ios::out);		//truncating file in case there was any other data in it

	if (!in)
	{
		cout << "\nError opening file.\n";
	}
	else
	{
		out << "\nSales Report\n\nPeriod: " << startDay << "/" << startMonth << "/" << startYear << " to " << endDay << "/" << endMonth << "/" << endYear << "\n\n";
		getline(in, temp);
				//ignoring the empty first line
		while (!in.eof())
		{
			inputSalesFromFile(in, emptyRecord);
			if (emptyRecord.date >= startingDate && emptyRecord.date <= endingDate)
			{
				outputSalesToFile(out, emptyRecord);
```

```
                        out << endl;
                        totalRevenue += emptyRecord.price;
                    }
                }
                out << endl << "\nTotal revenue generated: " << totalRevenue << endl;

                in.close();
                out.close();

                cout << "\nReport generated.\n";
        }
}



void programInterface()
{
        int option = 0, optionTwo = 0, optionThree = 0;
        string userIdentity;
        cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t Welcome to Phone Finder Application\n\n\t\t\t\t\t\t\t
";
        Sleep(2000);
        system("CLS");
```

```
while (1)
{
	cout << "\n\n\n\n\n\n\n\n\n\n\t\t\t\t   Enter 1 to access inventory management features,"
		<< "\n\t\t\t\t      2 to search phones on basis of preferences,"
		<< "\n\t\t\t\t\t\t  3 to buy a phone,"
		<< "\n\t\t\t\t\t4 to access sales management features,"
		<< "\n\t\t\t\t\t\tor 0 to exit the program.\n\n\t\t\t\t\t\t\t\t  ";
	cin >> option;

	if (option == 1)
	{
		system("CLS");
		cout << "\n\n\n\n\n\n\n\n\n\t\t\t\t\t     Enter 1 to add a new phone,"
			<< "\n\t\t\t\t\t\t  2 to edit a phone,\n\t\t\t\t\t\t 3 to delete a phone,"
			<< "\n\t\t\t\t\t     4 to view low-stock phones,\n\t\t\t\t\t\t5 to see all
inventory,"
			<< "\n\t\t\t\t\t    or 0 to go back to main menu\n\n\t\t\t\t\t\t\t  ";
		cin >> optionTwo;
		cout << endl << endl;
		if (optionTwo == 1)
		{
			Inventory::addRecord();
		}
		else if (optionTwo == 2)
		{
```

```
                Inventory::editRecord();
        }
        else if (optionTwo == 3)
        {
                Inventory::deleteRecord();
        }
        else if (optionTwo == 4)
        {
                Inventory::lowStockPhones();
        }
        else if (optionTwo == 5)
        {
                Inventory::showInventory();
        }
        else
        {
                continue;
        }
}
else if (option == 2)
{
        system("CLS");
        cin.ignore();
        cout << "\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\tEnter userID: ";
        getline(cin, userIdentity);
```

```
        Search s(userIdentity);
        system("CLS");
        cout << "\n\n";
        s.byPreference();
    }
    else if (option == 3)
    {
        system("CLS");
        cout << "\n\n";
        Sales customer;
        customer.buyPhone();
    }
    else if (option == 4)
    {
        system("CLS");
        cout << "\n\n";
        Sales salesman;
        cout << "\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tEnter 1 to find invoice/s of a customer,"
            << "\n\t\t\t\t\t     or 2 to generate sales report.\n\n\t\t\t\t\t\t\t  ";
        cin >> optionThree;

        if (optionThree == 1)
        {
            system("CLS");
            cout << "\n\n";
```

```
                salesman.findInvoices();
        }
        else if (optionThree == 2)
        {
                system("CLS");
                cout << "\n\n";
                salesman.generateReport();
        }
        else
        {
                cout << "\nInvalid value. Going back to main menu.";
                Sleep(3000);
                system("CLS");
                continue;
        }
}
else if (option == 0)
{
        system("CLS");
        cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t\tgoodbye!\n\n\n\n\n\n\n\n\n\n\n\n\n";
        exit(0);
}
else
{
        cout << "\n\nInvalid value.";
```

```
                }
                Sleep(3000);
                system("CLS");
        }
}
```

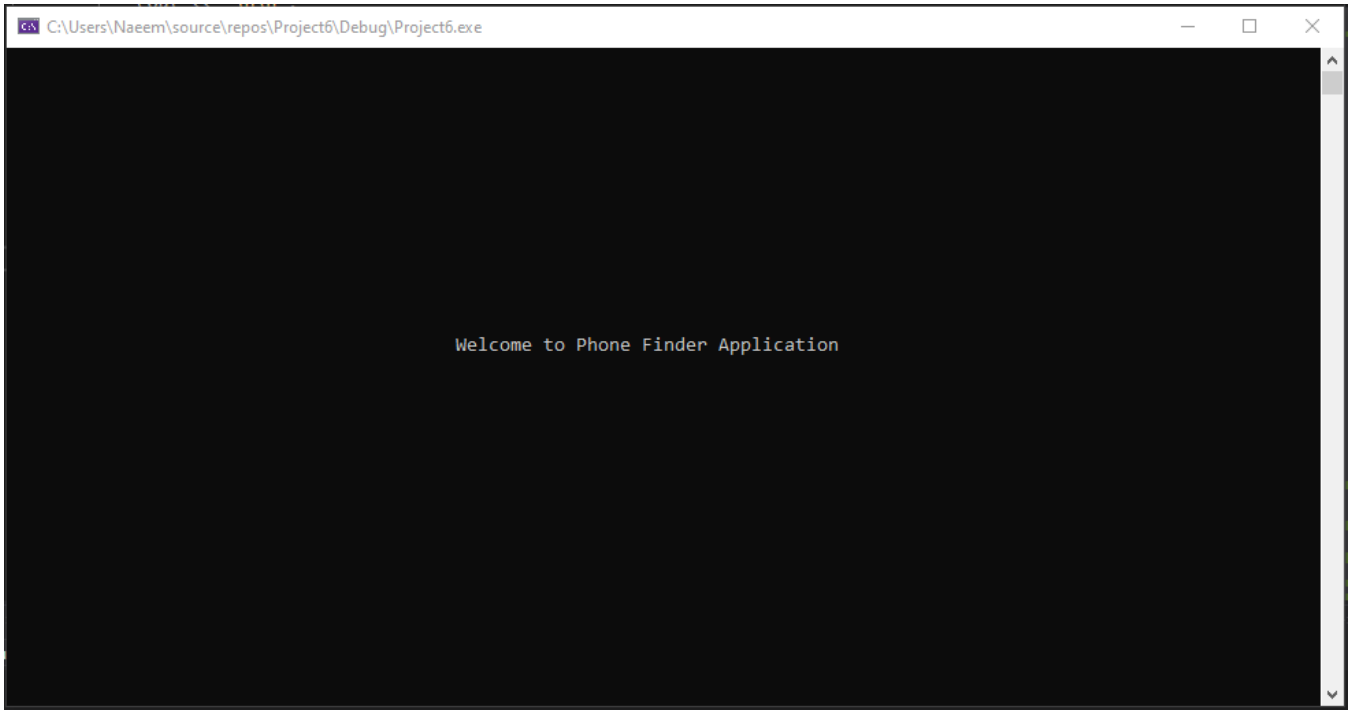
main.cpp file:

```
#include <iostream>
#include "search.h"
#include "sales.h"
using namespace std;

void programInterface();

int main()
{
	programInterface();
}
```

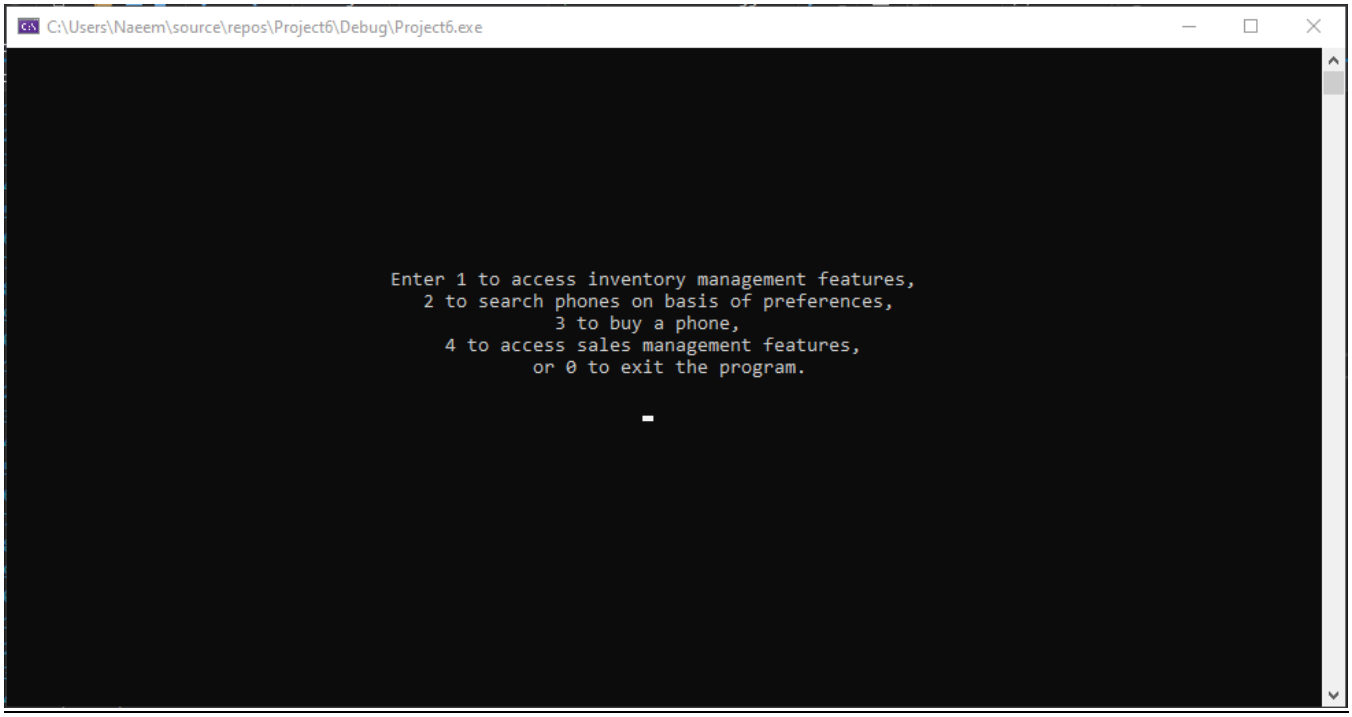
Tools Used: Visual Studio 2019, Microsoft Word

Sample Program Outputs:

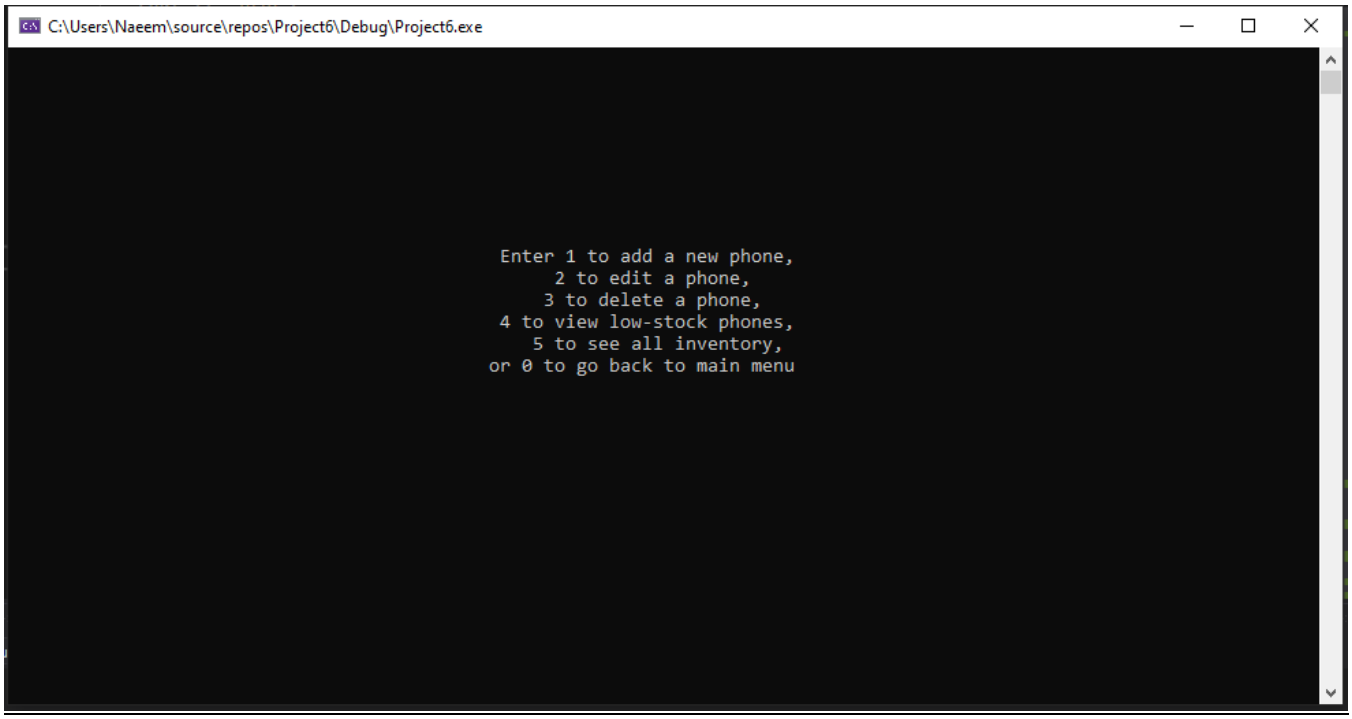
Welcome screen:



Main menu :



Inventory management features:



add new phone record feature:

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

                    Enter 1 to add a new phone,
                         2 to edit a phone,
                        3 to delete a phone,
                    4 to view low-stock phones,
                       5 to see all inventory,
                   or 0 to go back to main menu

                                1




Enter model: A11

Enter brand: Samsung

Enter price: 37000

Enter OS: Android

Enter RAM size (in GBs): 8
Enter screen size (in Inches): 6.4
Enter camera magnitude (in MPs): 12
Enter build: Plastic

Enter number of units in stock: 90


Record added successfully.
```

```
*InventoryData - Notepad
File  Edit  Format  View  Help
A11
Samsung
37000
Android
8
6.4
12
Plastic
90
```

edit phone record feature:

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe
                                                or 0 to go back to main menu

                                                              2




Enter the model you want to edit: A11

Model: A11
Brand: Samsung
Price: Rs. 37000
OS: Android
RAM: 8 GB
Screen size: 6.4 Inches
Camera: 12 MP
Build: Plastic
Units in stock: 90

Enter 1 to edit this record or 0 to cancel.

1

Enter new values.



Enter model: A12

Enter brand: LG

Enter price: 56000

Enter OS: Android

Enter RAM size (in GBs): 8
Enter screen size (in Inches): 6.5
Enter camera magnitude (in MPs): 14
Enter build: Metal

Enter number of units in stock: 80


Record editted successfully.
```

*InventoryData - Notepad

File Edit Format View Help

```
A11
Samsung
37000
Android
8
6.4
12
Plastic
90
```

InventoryData - Notepad

File Edit Format View Help

```
A12
LG
56000
Android
8
6.5
14
Metal
80
```

delete phone record:

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

                                        Enter 1 to add a new phone,
                                             2 to edit a phone,
                                            3 to delete a phone,
                                        4 to view low-stock phones,
                                           5 to see all inventory,
                                       or 0 to go back to main menu

                                                     3



Enter the model you want to delete: A12

Model: A12
Brand: LG
Price: Rs. 56000
OS: Android
RAM: 8 GB
Screen size: 6.5 Inches
Camera: 14 MP
Build: Metal
Units in stock: 80

Enter 1 to delete this record or 0 to cancel.

1


Record deleted successfully.
```

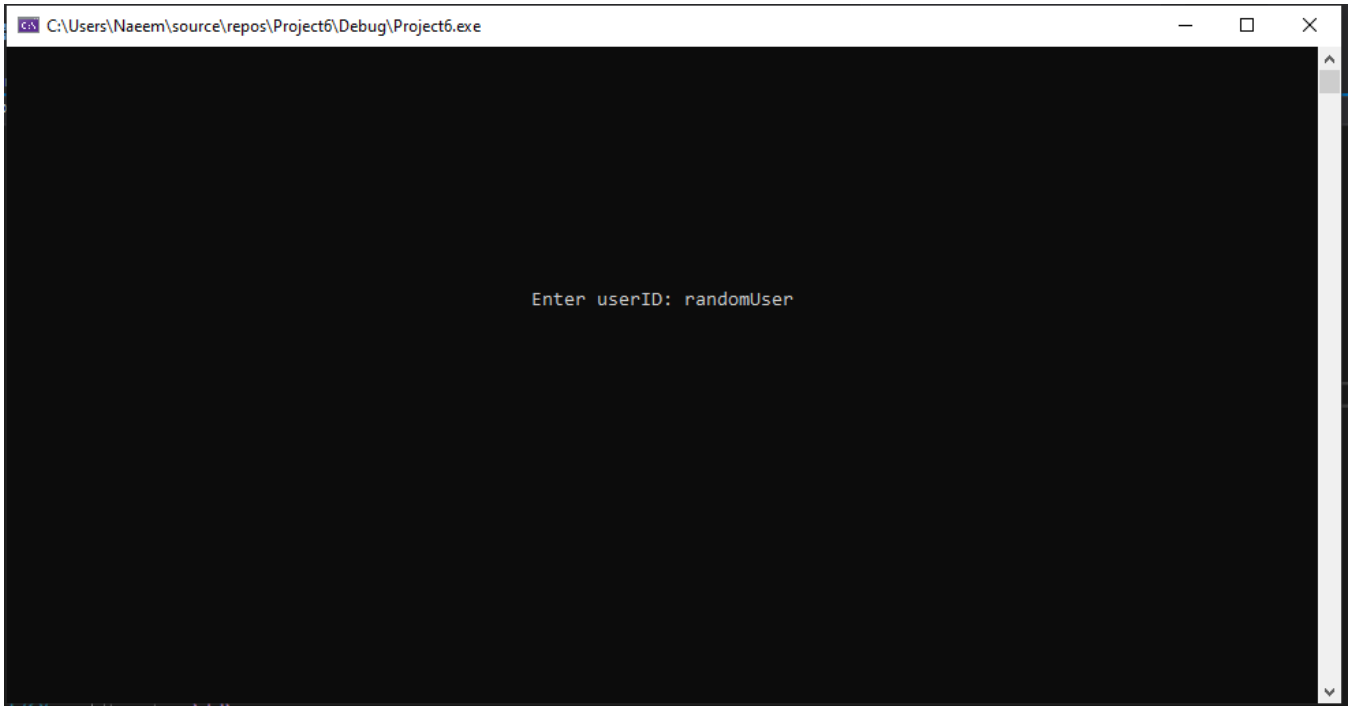
Low-stock phones:

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

                                        Enter 1 to add a new phone,
                                             2 to edit a phone,
                                            3 to delete a phone,
                                        4 to view low-stock phones,
                                           5 to see all inventory,
                                       or 0 to go back to main menu

                                                     4




Low stock phone/s:


Model: A70
Brand: Samsung
Price: Rs. 40000
OS: Android
RAM: 8 GB
Screen size: 6.5 Inches
Camera: 48 MP
Build: Plastic
Units in stock: 7

Model: Note10
Brand: Xiaomi
Price: Rs. 80000
OS: Android
RAM: 12 GB
Screen size: 6.6 Inches
Camera: 108 MP
Build: Metal
Units in stock: 4
```

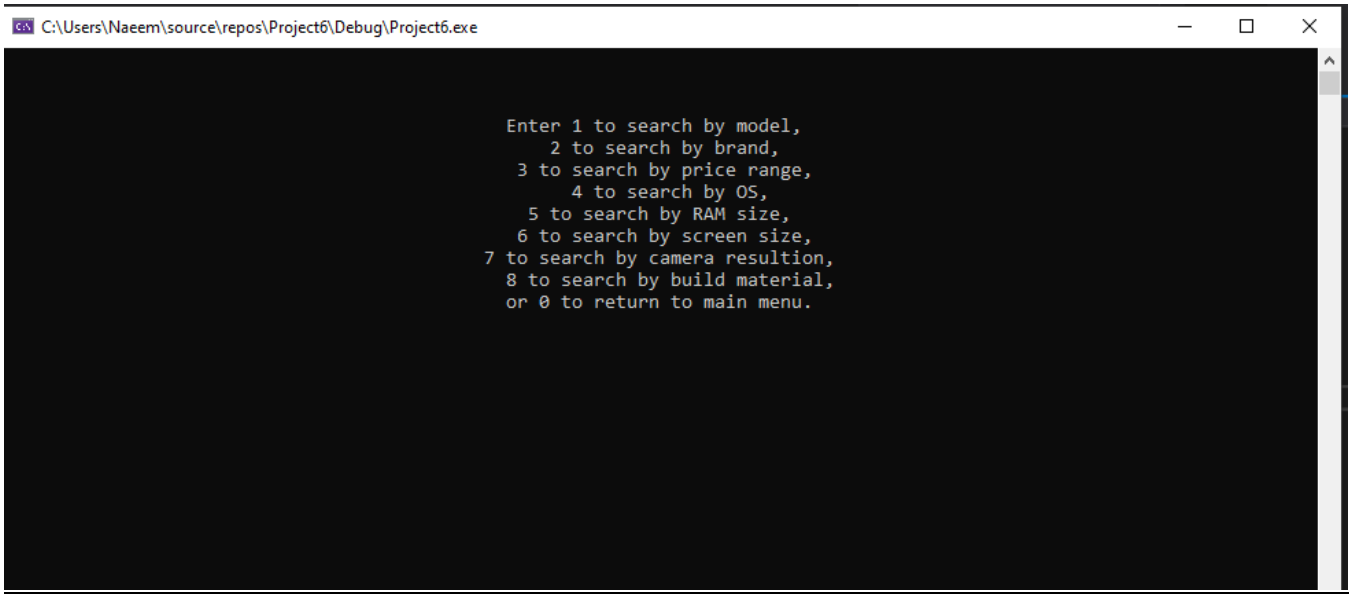
See all inventory records:

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

                                        Enter 1 to add a new phone,
                                             2 to edit a phone,
                                            3 to delete a phone,
                                        4 to view low-stock phones,
                                           5 to see all inventory,
                                       or 0 to go back to main menu

                                                     5



Model: A50
Brand: Samsung
Price: Rs. 30000
OS: Android
RAM: 4 GB
Screen size: 6.2 Inches
Camera: 10 MP
Build: Plastic
Units in stock: 100

Model: A60
Brand: Samsung
Price: Rs. 36000
OS: Android
RAM: 4 GB
Screen size: 6.4 Inches
Camera: 20 MP
```

user ID before search (to save search results):



Search by preference (price, brand, ram, etc.) (menu + samples):



C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

```
                                        Enter 1 to search by model,
                                            2 to search by brand,
                                         3 to search by price range,
                                              4 to search by OS,
                                           5 to search by RAM size,
                                          6 to search by screen size,
                                       7 to search by camera resultion,
                                         8 to search by build material,
                                         or 0 to return to main menu.

                                                       2
Enter phone brand: Samsung

Model: A50
Brand: Samsung
Price: Rs. 30000
OS: Android
RAM: 4 GB
Screen size: 6.2 Inches
Camera: 10 MP
Build: Plastic

Model: A60
Brand: Samsung
Price: Rs. 36000
OS: Android
RAM: 4 GB
Screen size: 6.4 Inches
Camera: 20 MP
Build: Plastic

Model: A70
Brand: Samsung
Price: Rs. 40000
OS: Android
RAM: 8 GB
```

C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

```
                                        Enter 1 to search by model,
                                            2 to search by brand,
                                         3 to search by price range,
                                              4 to search by OS,
                                           5 to search by RAM size,
                                          6 to search by screen size,
                                       7 to search by camera resultion,
                                         8 to search by build material,
                                         or 0 to return to main menu.

                                                      8
Enter build material: Metal

Model: Note10
Brand: Xiaomi
Price: Rs. 80000
OS: Android
RAM: 12 GB
Screen size: 6.6 Inches
Camera: 108 MP
Build: Metal

Model: iPhone11
Brand: Apple
Price: Rs. 130000
OS: iOS
RAM: 6 GB
Screen size: 6.2 Inches
Camera: 20 MP
Build: Metal
```

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe
                                        Enter 1 to search by model,
                                            2 to search by brand,
                                         3 to search by price range,
                                               4 to search by OS,
                                          5 to search by RAM size,
                                         6 to search by screen size,
                                       7 to search by camera resultion,
                                         8 to search by build material,
                                         or 0 to return to main menu.

                                                      3

Enter upper price limit: 50000

Enter lower price limit: 20000

Model: A50
Brand: Samsung
Price: Rs. 30000
OS: Android
RAM: 4 GB
Screen size: 6.2 Inches
Camera: 10 MP
Build: Plastic

Model: A60
Brand: Samsung
Price: Rs. 36000
OS: Android
RAM: 4 GB
Screen size: 6.4 Inches
Camera: 20 MP
Build: Plastic

Model: A70
Brand: Samsung
Price: Rs. 40000
OS: Android
RAM: 8 GB
```

C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

```
                                        Enter 1 to search by model,
                                            2 to search by brand,
                                         3 to search by price range,
                                              4 to search by OS,
                                           5 to search by RAM size,
                                          6 to search by screen size,
                                     7 to search by camera resultion,
                                          8 to search by build material,
                                          or 0 to return to main menu.

                                                        5
Enter RAM (in GBs): 8

Model: A50
Brand: Samsung
Price: Rs. 30000
OS: Android
RAM: 4 GB
Screen size: 6.2 Inches
Camera: 10 MP
Build: Plastic

Model: A60
Brand: Samsung
Price: Rs. 36000
OS: Android
RAM: 4 GB
Screen size: 6.4 Inches
Camera: 20 MP
Build: Plastic
```

randomUser - Notepad

File Edit Format View Help

```
A50
Samsung
30000
Android
4
6.2
10
Plastic

A60
Samsung
36000
Android
4
6.4
20
Plastic

A70
Samsung
40000
Android
8
6.5
48
Plastic
```

Figure 1: saved search results

buy phone feature (+ saving invoice):

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

Enter phone model: C11

Enter full name: Shahid Faisal

Enter contact number: 342-5234-233

Enter credit/debit card number to confirm purchase: 567456

Customer Invoice

Model: C11
Brand: Realme
Price: Rs. 25000
OS: Android
RAM: 3 GB
Screen size: 6.3 Inches
Camera: 12 MP
Build: Plastic
Purchased by: Shahid Faisal
Contact number: 342-5234-233

Purchase completed successfully.
```

SalesInvoice - Notepad

File Edit Format View Help

```
C11
Realme
25000
Android
3
6.3
12
Plastic
20200624
Shahid Faisal
342-5234-233
567456
```

find customer invoice:

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe


Enter customer's contact number: 315-1111-111

Customer Invoice


Model: C11
Brand: Realme
Price: Rs. 25000
OS: Android
RAM: 3 GB
Screen size: 6.3 Inches
Camera: 12 MP
Build: Plastic
Purchased by: Jahanzeb Naeem
Contact number: 315-1111-111
```

generate report:

```
C:\Users\Naeem\source\repos\Project6\Debug\Project6.exe

Enter starting date of report (in DD/MM/YYYY format): 23/02/2019

Enter ending date of report (in DD/MM/YYYY format): 20/04/2020

Report generated.
```

SalesReport - Notepad

File Edit Format View Help

```
Sales Report

Period: 23/02/2019 to 20/04/2020


C11
Realme
25000
Android
3
6.3
12
Plastic
20190624
Jahanzeb Naeem
315-1111-111
1234234

X
OnePlus
65000
Android
8
```

SalesReport - Notepad

File Edit Format View Help

```
6.6
108
Metal
20200215
Kamran Ali
324-1234-121
54233453

iPhone11
Apple
130000
iOS
6
6.2
20
Metal
20191216
Faiza Ahmad
361-2131-354
45675764


Total revenue generated: 665000
```

Conclusion:

It can be seen that it is indeed possible to create such an information system using UML diagrams, classes, text files, operator overloading, loops and selection statements, multilevel inheritances, virtual functions and static polymorphism.