**INDEX**

1. ABOUT C++…….……………………………………………………...Pg-01
2. OOP……………………………………….................................….Pg-03
3. SYSTEM ANALYSIS……….…………………………………..Pg-06
4. WATERFALL MODEL…………………………………..……….Pg-07
5. SYSTEM FLOW………………………….............................Pg-08
6. SYSTEM DESCRIPTION……………………………………..Pg-09
7. ERROR HANDLING…………………………………………….…Pg-10
8. DATA DICTIONARY…………………………………………….Pg-12
9. SOURCE CODE……………………………………………………….Pg-13
10. SCREENSHOTS…………………………………….…..…..........Pg-79
11. ACKNOWLEDGEMENT……………………………….………..Pg-82
12. BIBLIOGRAPHY……………………………………………….…….Pg-83

**ABOUT C++**

**C++** (pronounced [*see plus plus*](http://en.wikipedia.org/wiki/Wikipedia:Pronunciation_respelling_key)) is a[statically typed](http://en.wikipedia.org/wiki/Statically_typed), [free-form](http://en.wikipedia.org/wiki/Free-form_language), [multi-paradigm](http://en.wikipedia.org/wiki/Multi-paradigm_programming_language), [compiled](http://en.wikipedia.org/wiki/Compiled_language), general-purpose [programming language](http://en.wikipedia.org/wiki/Programming_language). It is regarded as a "middle-level" language, as it comprises a combination of both [high-level](http://en.wikipedia.org/wiki/High-level_programming_language) and [low-level](http://en.wikipedia.org/wiki/Low-level_programming_language) language features. It was developed by [Bjarne Stroustrup](http://en.wikipedia.org/wiki/Bjarne_Stroustrup) starting in 1979 at [Bell Labs](http://en.wikipedia.org/wiki/Bell_Labs) as an enhancement to the [C language](http://en.wikipedia.org/wiki/C_(programming_language)) and originally named *C with Classes*. It was renamed *C++* in 1983.

As one of the most popular programming languages ever created, C++ is widely used in the software industry. Some of its application domains include systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as [video games](http://en.wikipedia.org/wiki/Video_games). Several groups provide both free and proprietary C++ [compiler](http://en.wikipedia.org/wiki/Compiler) software, including the [GNU Project](http://en.wikipedia.org/wiki/GNU_Compiler_Collection), [Microsoft](http://en.wikipedia.org/wiki/Microsoft_Visual_C%2B%2B), [Intel](http://en.wikipedia.org/wiki/Intel_C%2B%2B_Compiler) and [Borland](http://en.wikipedia.org/wiki/Borland_C%2B%2B_Builder). C++ has greatly influenced many other popular programming languages, most notably [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)) and [Java](http://en.wikipedia.org/wiki/Java_(programming_language)).

C++ is also used for [hardware design](http://en.wikipedia.org/wiki/Hardware_design), where design is initially described in C++, then analyzed, architecturally constrained, and scheduled to create a [register transfer level](http://en.wikipedia.org/wiki/Register_transfer_level) [hardware description language](http://en.wikipedia.org/wiki/Hardware_description_language) via [high-level synthesis](http://en.wikipedia.org/wiki/High-level_synthesis).

The language began as enhancements to [C](http://en.wikipedia.org/wiki/C_(programming_language)), first adding [classes](http://en.wikipedia.org/wiki/Class_(computer_science)), then [virtual functions](http://en.wikipedia.org/wiki/Virtual_functions), [operator overloading](http://en.wikipedia.org/wiki/Operator_overloading), [multiple inheritance](http://en.wikipedia.org/wiki/Multiple_inheritance), [templates](http://en.wikipedia.org/wiki/Template_(programming)), and [exception handling](http://en.wikipedia.org/wiki/Exception_handling) among other features. After years of development, the C++ programming language standard was ratified in 1998 as [*ISO/IEC 14882*](http://en.wikipedia.org/wiki/ISO/IEC_14882)*:1998*. That standard is still current, but is amended by the 2003 technical [corrigendum](http://en.wikipedia.org/wiki/Errata), *ISO/IEC 14882:2003*. The next standard version (known informally as [C++0x](http://en.wikipedia.org/wiki/C%2B%2B0x)) is in development.

**ABOUT OOP**

**Object-oriented programming** (**OOP**) is a [programming paradigm](http://en.wikipedia.org/wiki/Programming_paradigm) that uses "[objects](http://en.wikipedia.org/wiki/Object_(computer_science))" – [data structures](http://en.wikipedia.org/wiki/Data_structure) consisting of [data fields](http://en.wikipedia.org/wiki/Field_(computer_science)) and [methods](http://en.wikipedia.org/wiki/Method_(computer_science)) together with their interactions – to design applications and computer programs. Programming techniques may include features such as [data abstraction](http://en.wikipedia.org/wiki/Data_abstraction), [encapsulation](http://en.wikipedia.org/wiki/Encapsulation_(object-oriented_programming)), [modularity](http://en.wikipedia.org/wiki/Module_(programming)), [polymorphism](http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming), and [inheritance](http://en.wikipedia.org/wiki/Inheritance_(computer_science)). Many modern [programming languages](http://en.wikipedia.org/wiki/Programming_language) now support OOP.

Class

A class is a template for an object, a user-defined data-type that contains variables, properties, and methods. A class defines the abstract characteristics of a thing (object), including its characteristics (its **attributes**, [**fields**](http://en.wikipedia.org/wiki/Field_(computer_science)) or [**properties**](http://en.wikipedia.org/wiki/Property_(programming))) and the **things it can do** (behaviors, [**methods**](http://en.wikipedia.org/wiki/Method_(computer_science)), **operations** or **features**). One might say that a class is a *blueprint* or *factory* that describes the nature of something. For example, the class Dog would consist of traits shared by all dogs, such as breed and fur color (characteristics), and the ability to bark and sit (behaviors). Classes provide [modularity](http://en.wikipedia.org/wiki/Modularity_(programming)) and structure in an object-oriented computer program. A class should typically be recognizable to a non-programmer familiar with the problem domain, meaning that the characteristics of the class should make sense in context. Also, the code for a class should be relatively self-contained (generally using **encapsulation**). Collectively, the properties and methods defined by a class are called **members**.

Instance

One can have an instance of a class; the instance is the actual object created at run-time. In programmer vernacular, the Lassie object is an **instance** of the Dog class. The set of values of the attributes of a particular object is called its [state](http://en.wikipedia.org/wiki/State_(computer_science)). The object consists of state and the behavior that's defined in the object's classes.

Method

Method is a set of procedural statements for achieving the desired result. It performs different kinds of operations on different data types. In a programming language, methods (sometimes referred to as "functions") are verbs. Lassie, being a Dog, has the ability to bark. So bark( ) is one of Lassie's methods. She may have other methods as well, for example sit( ) or eat( ) or walk( ) or save(Timmy). Within the program, using a method usually affects only one particular object; all Dogs can bark, but you need only one particular dog to do the barking.

Inheritance

Inheritance is a process in which a class inherits all the state and behavior of another class. This type of relationship is called child-Parent or is-a relationship. "Subclasses" are more specialized versions of a class, which *inherit* attributes and behaviors from their parent classes, and can introduce their own.

For example, the class Dog might have sub-classes called Collie, Chihuahua, and GoldenRetriever. In this case, Lassie would be an instance of the Collie subclass. Suppose the Dog class defines a method called bark( ) and a property called furColor. Each of its sub-classes (Collie, Chihuahua, and GoldenRetriever) will inherit these members, meaning that the programmer only needs to write the code for them once.

Each subclass can alter its inherited traits. For example, the Collie subclass might specify that the default furColor for a collie is brown-and-white. The Chihuahua subclass might specify that thebark( ) method produces a high pitch by default. Subclasses can also add new members. The Chihuahua subclass could add a method called tremble( ). So an individual chihuahua instance would use a high-pitched bark( ) from the Chihuahua subclass, which in turn inherited the usualbark( ) from Dog. The Chihuahua object would also have the tremble( ) method, but Lassie would not, because she is a Collie, not a Chihuahua. In fact, inheritance is an "***a*… is a**" relationship between classes, while instantiation is an "**is a**" relationship between an object and a class: ***a*** Collie *is a* Dog ("a… is a"), but Lassie *is a* Collie ("is a"). Thus, the object named Lassie has the methods from both classes Collie and Dog.

[Multiple inheritance](http://en.wikipedia.org/wiki/Multiple_inheritance) is inheritance from more than one ancestor class, neither of these ancestors being an ancestor of the other. For example, independent classes could define Dogs and Cats, and a Chimera object could be created from these two that inherits all the (multiple) behavior of cats and dogs. This is not always supported, as it can be hard to implement.

Abstraction

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes.

Encapsulation

Encapsulation conceals the functional details of a class from objects that send messages to it.

For example, the Dog class has a bark( ) method variable, data. The code for the bark( ) method defines exactly how a bark happens (e.g., by inhale( ) and then exhale( ), at a particular pitch and volume). Timmy, Lassie's friend, however, does not need to know exactly how she barks. Encapsulation is achieved by specifying which classes may use the members of an object. The result is that each object exposes to any class a certain [*interface*](http://en.wikipedia.org/wiki/Interface_(computer_science)) — those members accessible to that class. The reason for encapsulation is to prevent clients of an interface from depending on those parts of the implementation that are likely to change in the future, thereby allowing those changes to be made more easily, that is, without changes to clients. For example, an interface can ensure that puppies can only be added to an object of the class Dog by code in that class. Members are often specified as **public**, **protected** or **private**, determining whether they are available to all classes, sub-classes or only the defining class. Some languages go further: [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) uses the **default** access modifier to restrict access also to classes in the same package, [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)) and [VB.NET](http://en.wikipedia.org/wiki/Visual_Basic.NET) reserve some members to classes in the same assembly using keywords **internal** (C#) or **Friend** (VB.NET). [Eiffel](http://en.wikipedia.org/wiki/Eiffel_(programming_language)) and [C++](http://en.wikipedia.org/wiki/C%2B%2B)allow one to specify which classes may access any member.

Polymorphism

Polymorphism allows the programmer to treat derived class members just like their parent class's members. More precisely, [Polymorphism in object-oriented programming](http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming) is the ability of [objects](http://en.wikipedia.org/wiki/Object_(computer_science)) belonging to different [data types](http://en.wikipedia.org/wiki/Data_type) to respond to calls of [methods](http://en.wikipedia.org/wiki/Method_(computer_science)) of the same name, each one according to an appropriate type-specific behavior. One method, or an operator such as +, -, or \*, can be abstractly applied in many different situations. If a Dog is commanded to speak( ), this may elicit a bark( ). However, if a Pig is commanded to speak( ), this may elicit an oink( ). Each subclass overrides the speak( ) method inherited from the parent class Animal.

**SYSTEM ANALYSIS**

**TOOLS, LANGUAGE AND PLATFORM USED**

* HARDWARE SPECIFICATION
* MEMORY : 4GB
* MICROPROCESSOR:5.6 Ghz
* HARD DISK: 1 TB
* PRINTER: HP Laserjet M1136 MFP printer
* SOFTWARE SPECIFICATION
* PLATFORM : C++ with graphics
* FRONT END: C++
* WINDOWS 10
* M.S WORD

**WATERFALL MODEL**

**SYSTEM FLOW**

Login

User Details

Buy

Main Menu

Share Details

External file

**SYSTEM DESCRIPTION**

* This project starts up with a logo screen and proceeds to a little briefing.
* Then a Screen Shows up which displays a table containing share opening and closing prices along with prices of other commodities such as petroleum, gold

and currencies.

* The Screen stays for a while showing % increase or fall in share prices.
* Then the login screen appears if you are a new user your data automatically gets inserted, if already a user then your data gets updated
* Then the main menu appears again displaying the share prices along with three options.

1. **ENTER SHARE NAME TO VIEW STATS:** Choosing this option displays a screen containing graph with current stats of the selected company.
2. **ENTER BUY TO BUY SHARES :-**Choosing this option displays a screen containing the share price table and asks you to enter the number of shares along with the share names and enter the track time.

:- Then it displays a multi-lined graph tracking your shares in real time and displaying gain or loss in the end.

1. **EXIT :** Choosing this will lead you out of the program.

**ERROR HANDLING**

Syntax Errors

These occur when the rules of a programming language are misused, i.e. when the grammatical rule of C++ is violated.

For instance,

int a,

cout<a;

In the first statement, the statement is not terminated by a semicolon but a comma. Also in the second statement, the cout statement has only one insertion operator whereas it needs two.

Semantic Errors

These errors occur when statements are not meaningful. Semantics refers to the set of rules which give the meaning of a statement.

For instance,

x \* y = z;

This statement will result into a semantic error as an expression cannot come on the left side of an assignment statement.

Type Errors

Data in C++ has an asociated data type. If a function is given wrong type of data, type eror is signaled by the compiler.

For instance, if the integer argument was expected but actually a string is given in its place, it is a type error.

Run-time Errors

A run time error occurs during the execution of a program. It is caused of some illegal operation taking place or non-availability of desired or required condition for the execution of the program.

For instance, if a program is trying to open a file which does not exist or does not open, it results into a run time error.

Logical Errors

A logical error is that error which causes a program to produce incorrect or undesired output.

For instance, if we are trying to print the table of 3 and we specify the condition of the loop - a>10 then the loop will not be executed even once.

Errors we encountered during the making of this project:

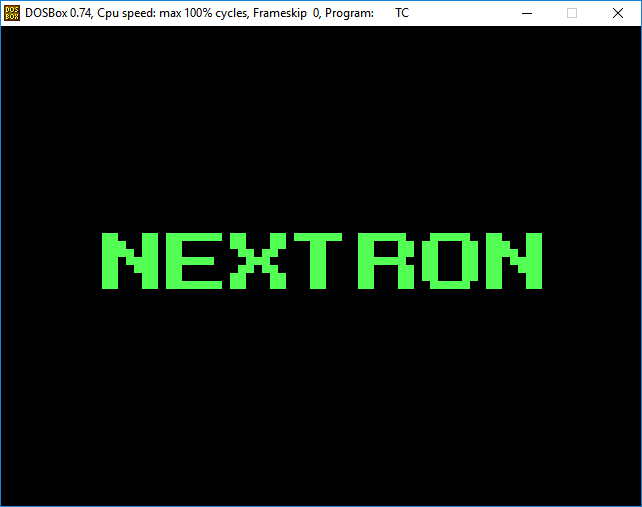
|  |  |  |
| --- | --- | --- |
| **Sr.**  **No.** | **Error message received** | **Reason due to which error was received** |
| 1. | Statement missing | A statement is not terminated by a semicolon (;). |
| 2. | Multiple declarations for i | The variable i is declared more than once in the same block. |
| 3. | Function exit( ) must have a prototype | A function is used without mentioning the prototype or including the necessary header files. |
| 4. | Too few/many arguments to call void strcpy(char\*,char\*) | A function is given lesser/more arguments than expected. |
| 5. | BGI error: graphics not initialized. | To use graphics, the function initgraph has to be used, which defines the graphics mode and location of BGI files. |

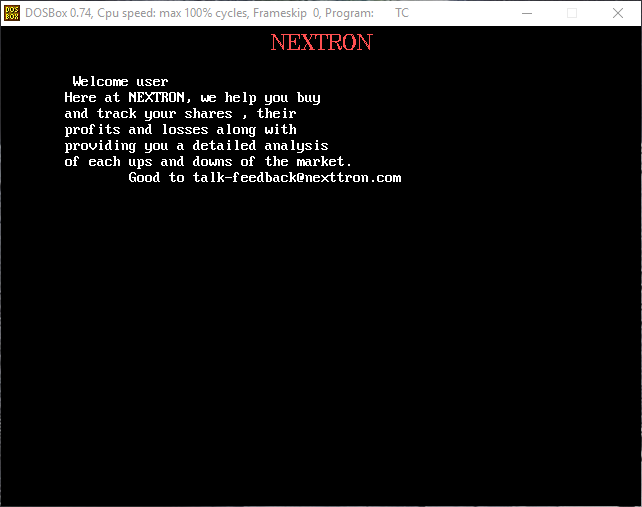
**DATA DICTIONARY**

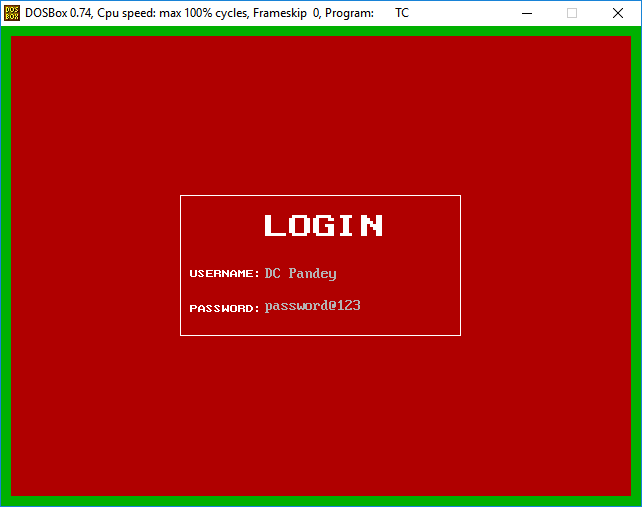
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.**  **No.** | **Field**  **Name** | **Type** | **Description** |
| 1 | User | Class | The instances of this class store the information about the users of the program like name, current stats, etc. during runtime. |
| 2 | userdata | .txt | It stores the records of all the users that have signed up to the program before and allows them to login and continue from where they left |
| 3 | share | structure | It stores the attributes of a single share and is used in corresponding member functions |
| 4 | ord | 2-d array | It stores the abscissa and the ordinate of points to be plotted in the graphs |
| 5 | login | function | It performs all necessary steps for registering the user and establishing a buffer to the external file for user data storage |
| 6 | logo | function | It displays the company logo for exponentially increasing intervals of time after each iteration |
| 7 | marquee | function | It displays the current stats of all commodities and member companies |

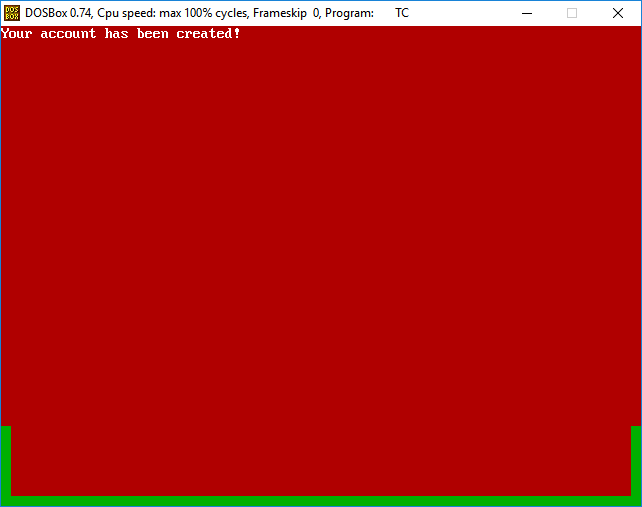
**SOURCE CODE**

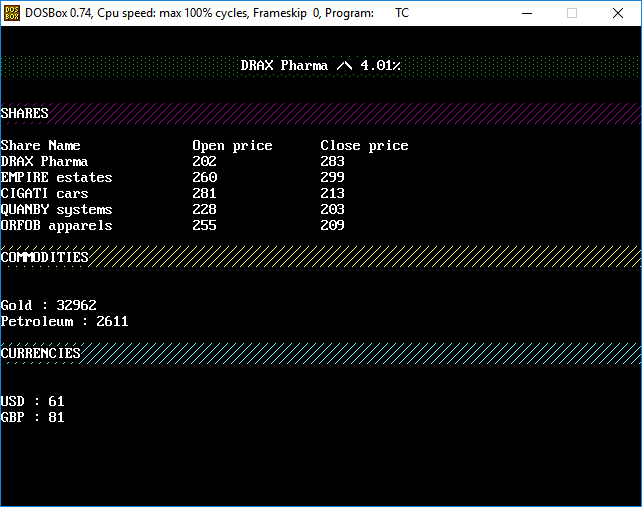
**OUTPUT**

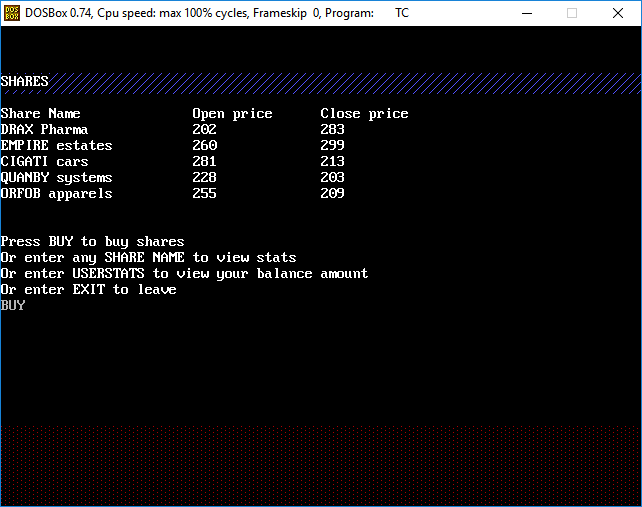
**1) **

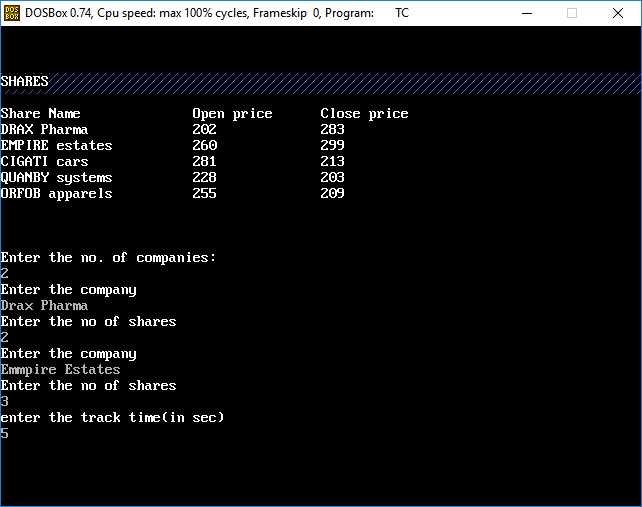
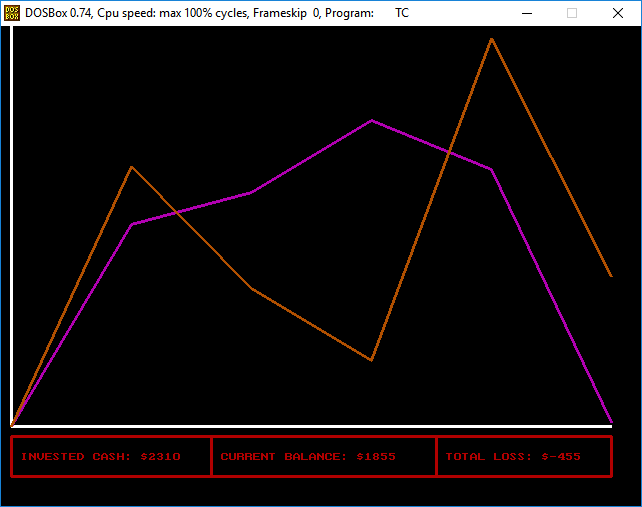
**2) **

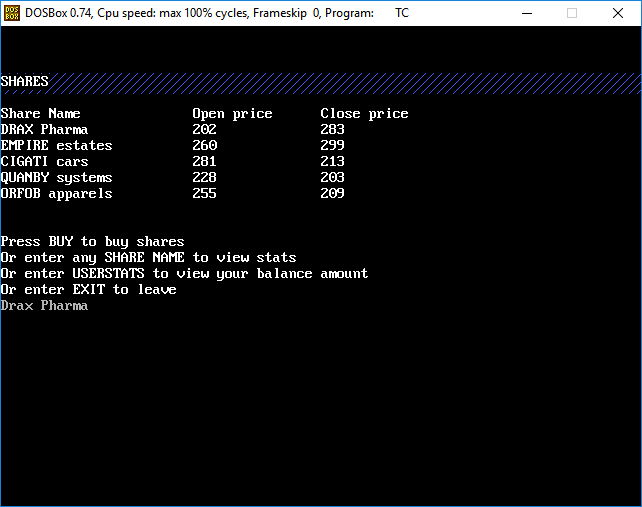
**3) **

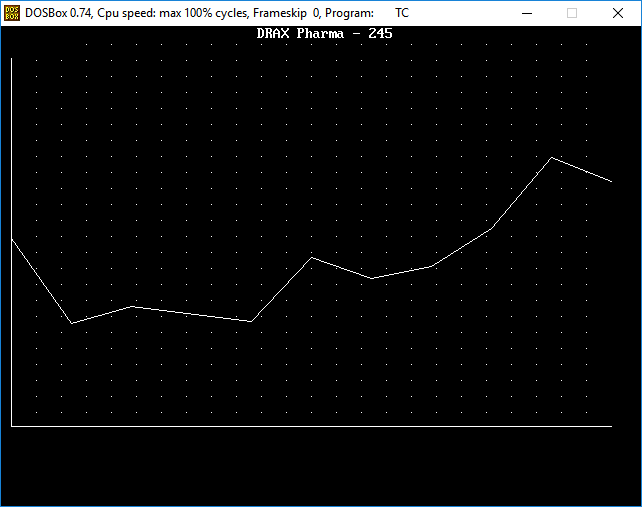
**4) **

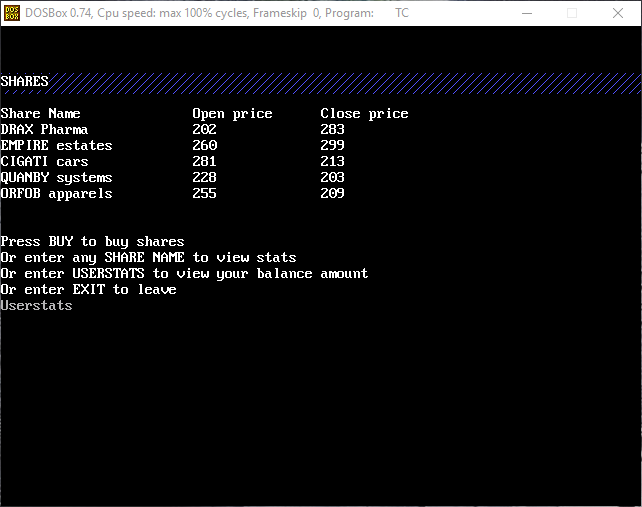
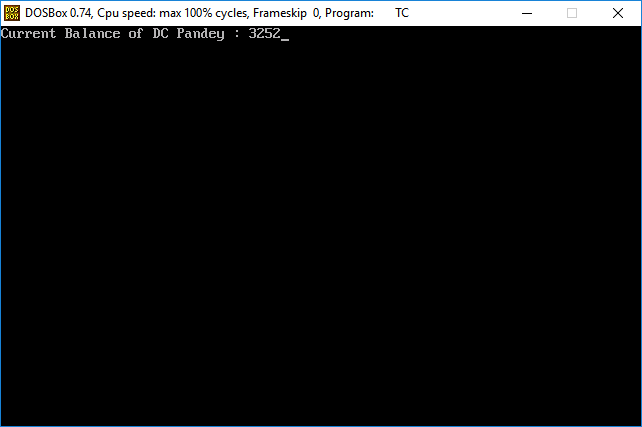
**5) **

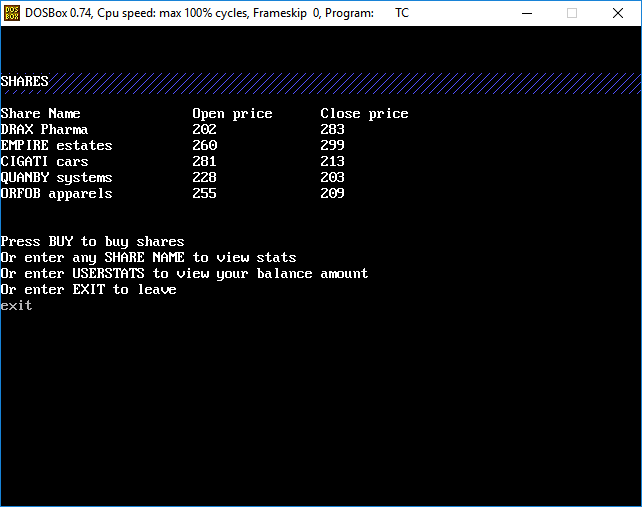
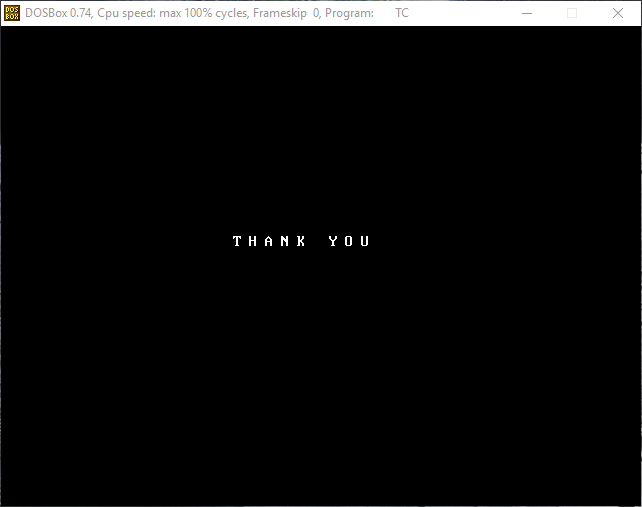
**6) **

**7)  8) **

**9) **

**10) **

**11)  12)**

**13) 14) **

**ACKNOWLEDGEMENT**

We would like to thank Jatin sir ,our computer science teacher for helping out on various aspects of our project. We would also like to thank our school and Mrugen sir ,our principal for providing us with the opportunity to work on this project. We would also like to thank our parents and friends and all others who have helped us in the making of this project.

**BIBLIOGRAPHY**

* SUMITA ARORA (CBSE class XII text book as per NCERT syllabus)
* SUMITA ARORA QUESTION BANK
* MONEYCONTROL APP
* INTERNET