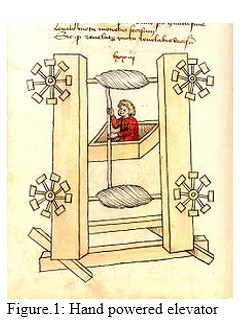
**CHAPTER 1:**

**INTRODUCTION**

**1.1. HISTORY**

The first reference to an elevator is in the works of the Roman architect Vitruvius, who reported that Archimedes built his first elevator probably in 236 B.C. In some literary sources of later historical periods, elevators were mentioned as cabs on a hemp rope and powered by hand or by animals. It is supposed that elevators of this type were installed in the Sinai monastery of Egypt.

In 1000, the Book of Secrets by the Arab Ibn Khalaf alMuradi in Islamic Spain described the use of an elevatorlike lifting device, in order to raise a large battering ram to destroy a fortress. In the 17th century the prototypes of elevators were located in the palace buildings of England and France.

In 1793 Ivan Kulibin created an elevator with the screw lifting mechanism for the Winter Palace of Saint Petersburg. In 1816 an elevator was established in the main building of sub Moscow village called Arkhangelskoye. In 1823, an "ascending room" made its debut in London.

In the middle 1800's, there were many types of crude elevators that carried freight. Most of them ran hydraulically. The first hydraulic elevators used a plunger below the car to raise or lower the elevator. A pump applied water pressure to a plunger, or steel column, inside a vertical cylinder. Increasing the pressure allowed the elevator to descend. The elevator also used a system of counter-balancing so that the plunger did not have to lift the entire weight of the elevator and its load. The plunger, however, was not practical for tall buildings, because it required a pit as deep below the building as the building was tall. Later a rope-geared elevator with multiple pulleys was developed.

Henry Waterman of New York is credited with inventing the "standing rope control" for an elevator in 1850.

In 1852, Elisha Otis introduced the safety elevator, which prevented the fall of the cab if the cable broke. The design of the Otis safety elevator is somewhat similar to one type still used today. A governor device engages knurled roller(s); locking the elevator to its guides should the elevator descend at excessive speed. He demonstrated it at the New York exposition in the Crystal Palace in 1854.

On March 23, 1857 the first Otis passenger elevator was installed at 488 Broadway in New York City. The first elevator shaft preceded the first elevator by four years. Construction for Peter Cooper's Cooper Union building in New York began in 1853. An elevator shaft was included in the design for Cooper Union, because Cooper was confident that a safe passenger elevator would soon be invented. The shaft was cylindrical because Cooper felt it was the most efficient design.

The first electric elevator was built by Werner von Siemens in 1880. The safety and speed of electric elevators were significantly enhanced by Frank Sprague. The development of elevators was led by the need for movement of raw materials including coal and lumber from hillsides. The technology developed by these industries and the introduction of steel beam construction worked together to provide the passenger and freight elevators in use today. In 1874, J.W. Meaker patented a method which permitted elevator doors to open and close safely.

In 1882, when hydraulic power was a well established technology, a company later named the London Hydraulic Power Company was formed. It constructed a network of high pressure mains on both sides of the Thames which, ultimately, extended to 184 miles and powered some 8,000 machines, predominantly lifts (elevators) and cranes. In 1929, Clarence Conrad Crispen, with Inclinator Company of America, created the first residential elevator.

**CHAPTER 2:**

**ANALYSIS AND DESIGN**

**2.1. OBJECTIVES OF THE PROJECT**

The company problem appears when it is intended to equip its building with an elevator and want to know according to its needs the way to settle the elevator to best satisfy this needs. Where this company my need from 1 to 4 elevators and the building will be of 100 floors (not including ground floor). The way where an Object-Oriented Software Simulator is to be installed, concerning the logic required to move the elevator between floors, giving the company a clear idea and leading it to choose the best design which satisfy this needs. An elevator (or lift) is a vertical transport vehicle that efficiently moves people or goods between floors of a building. They are generally powered by electric motors that either drive traction cables and counterweight systems, or pump hydraulic fluid to raise a cylindrical piston.

**2.2. REQUIREMENT SPECIFICATION**

1. The first and the foremost important hardware required is a laptop of a personal computer with a minimum of 4GB RAM.
2. We require an IDE where we can type our code and where it can be compiled and executed successfully.
3. Next is the right knowledge of Object Oriented Programming. Under it we are required to study concepts such has classes and objects, function overloading, constructors and destructors, inheritance concepts and types of inheritance concept, data abstraction , data encapsulation, visibility modes and their importance, friend functions and their rules of application, static variables and functions, function overriding, early binding, late binding, virtual functions concept and pure virtual functions.

With the availability of all the above resources, it is simple to design the project.

**2.3. ALGORITHM/PSEUDO CODE**

**2.3.1. PROBLEM FEATURES**

The problem concerns the logic required to move elevators between floors according to the following constraints:

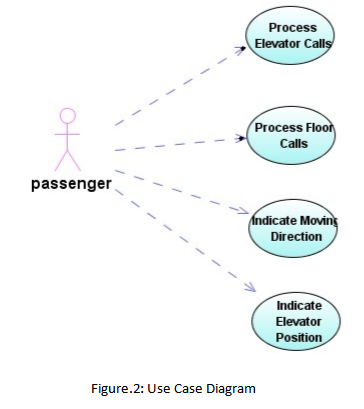
1. Having a unique control system, which control the elevators and pick up the requests.
2. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the elevator visits the corresponding floor.
3. Each floor, except the first floor and top floor has two buttons, one to request an up elevator and one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor and then moves in the desired direction.
4. Continue traveling in the same direction while there are remaining requests in that same direction.
5. When an elevator has no requests, it remains at its current floor with its doors closed.

The elevator algorithm is implemented in computer operating systems as an algorithm for scheduling hard disk requests.

**2.3.2 SYSTEM EVENTS**

The system events can be summarized as follows:

1. Process Elevator Calls: These scenarios includes that the elevator receives calls from the passengers, which is stored in a file format, updates the record of elevator calls stored in system controlling parts, etc.
2. Process Floor Calls: Similar to Elevator Call processing, this use case includes that the elevator receives Floor calls from the passengers, updates the record of floor calls in system controlling parts, etc.
3. Indicating Moving Direction: The elevator should have this mechanism to let the passengers know the current moving direction of the elevator such that the passenger might decide whether to enter the elevator or not.
4. Indicating Elevator Position: Similarly, the elevator should let the passengers know whether his/her destination floor is reached so that the passenger may decide to leave the elevator.
5. Status of source: The elevator should let the passengers know where the current position of the lift is, so that the passenger is well aware to know which lift he/she has to call.
6. Status of the destination: Similar to Status of source, the elevator should also let the passengers know where the elevator is taking him/her.
7. Status of the passengers: The maximum number of people the elevator can process is 8. So this should be made aware to the passengers well in before.



**2.3.3 SOURCE CODE**

#include<iostream>

#include<fstream>

using namespace std;

class LiftRequest {

public:

int dest;

int src;

int ppl;

void toString() {

cout<<"LIFT REQUEST -> SOURCE: "<<src<<" DESTINATION: "<<dest<<" PPL: "<<ppl<<endl;

cout<<endl;

}

};

class Lift {

public:

LiftRequest lr;

Lift() {

lr.src=0;

lr.dest=0;

lr.ppl=0;

}

void process(int id) {

cout<<endl<<" ############## Moving Lift "<<(id+1)<<" FROM "<<lr.src<<" TO "<<lr.dest<<" ############## "<<endl;

lr.src=lr.dest;

lr.ppl=0;

}

friend void toString(int, Lift);

};

void toString(int id, Lift l) {

cout<<"LIFT "<<(id+1)<<(l.lr.src==l.lr.dest?" @ REST":"LIFT @ MOVING")<<" SOURCE:"<<l.lr.src<<" DESTINATION:"<<l.lr.dest<<" PPL:"<<l.lr.ppl<<endl;

}

class LiftHandler {

public:

Lift l[4];

void showLiftStatus() {

cout<<"------------------------------------------------"<<endl;

cout<<"ALL LIFT STATUS"<<endl;

cout<<"------------------------------------------------"<<endl;

for(int j=0;j<4;j++) {

toString(j, l[j]);

}

cout<<"------------------------------------------------"<<endl;

cout<<endl;

}

void process(LiftRequest lr) {

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"BEFORE"<<endl;

showLiftStatus();

cout<<"PROCESSING LIFT REQUEST........."<<endl;

lr.toString();

int found = 0;

do {

for(int i=0;i<4;i++) {

// IF IN REST and DESTINATION

if(lr.src==l[i].lr.dest) {

found = 1;

l[i].lr = lr;

l[i].process(i);

break;

}

}

if(!found) {

cout<<"NO LIFT FOUND"<<endl<<"GETTING LIFT TO "<<lr.src<<endl;

l[0].lr.src = lr.src;

l[0].lr.dest = l[0].lr.src;

l[0].lr.ppl=0;

toString(0, l[0]);

}

} while(!found);

cout<<endl<<"AFTER"<<endl;

showLiftStatus();

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

};

// File Handler Protocol

class FileHandler {

public:

virtual void readFromFile()=0;

virtual void writeToFile(LiftRequest)=0;

};

class MyFileHandler:public FileHandler {

private:

LiftHandler lh;

public:

void readFromFile();

void writeToFile(LiftRequest);

};

void MyFileHandler::writeToFile(LiftRequest lf) {

ofstream outfile;

outfile.open("LftRe.txt", ios::app);

outfile<<lf.dest<<" "<<lf.src<<" "<<lf.ppl<<endl;

outfile.close();

}

void MyFileHandler::readFromFile() {

char wait;

ifstream infile;

infile.open("LftRe.txt");

while(!infile.eof()) {

LiftRequest lf;

infile>>lf.src;

infile>>lf.dest;

infile>>lf.ppl;

cout<<"New Lift Request"<<endl;

lf.toString();

if(lf.ppl<8) {

lh.process(lf);

} else {

cout<<"NO OF PPL LIMIT EXCEEDED"<<endl;

}

cout<<"Press any key to continue..."<<endl;

cin>>wait;

}

}

int main() {

MyFileHandler fh;

fh.readFromFile();

//test();

int ij;

cin>>ij;

return 0;

}

**2.4. CLASS DIAGRAM**

Class diagram, one of the most commonly used diagrams in object-oriented system, models the static design view for a system. The static view mainly supports the functional requirements of a system, the services of the system should provide to the end users. We will see from our practical experience that lots of fun comes out when modeling out system with class diagrams. The discussion on different views of class diagrams for the system will be put into emphasis later in this paper.

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams involve global system description, such as the system architecture, and detail aspects such as the attributes and operations within a class as well. The most common contents of a class diagram are:

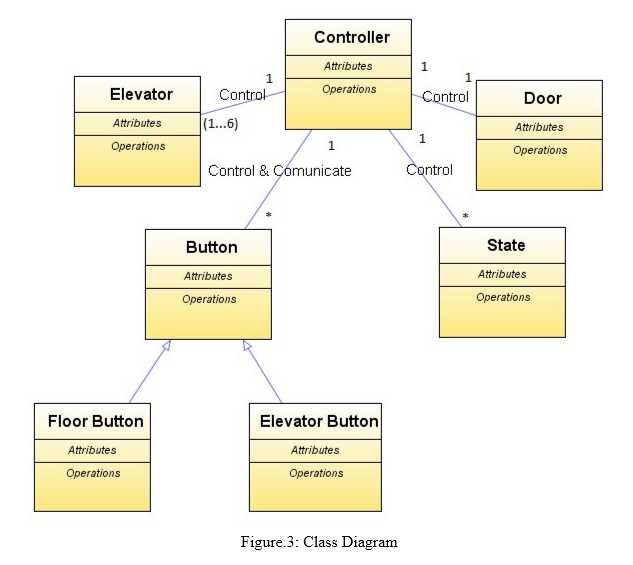
1. Classes
2. Interfaces
3. Collaborations
4. Dependency, generalization, and association relationships
5. Notes and constraints

Controller: The central controlling object in the elevator system. Controller communicates and controls all other objects in the system.

Door: The Command the doors to open and close, according to the situation stated in the use case. Elevator: The Elevator is being controlled to move up and down, to make stops at floors when necessary.

Button: The Controller class also controls the button class, which further generalizes two subclasses Floor Button and Elevator Button. The control object communicates with the Button objects, get the information whether a button is pressed and in turn controls the illumination of Button lights.

State: The indicators are controlled to show the information about the current position and moving direction of the elevator.



The classes captured in this diagram can cover all the functional aspects of the system: for moving or stopping the elevator, we have the class Elevator, and the control class Controller; for opening or closing the doors, we have the class Door; for the passenger to know the position and direction of the elevator, we have State class, for the passenger to make elevator calls or floor calls, we have the Button classes. All the classes have interfaces with the central controller, whose job is in charge of the actions of other objects. From the point of view of object division and system functioning, this class diagram helps understanding the basic design idea of the system.

**CHAPTER 3:**

**IMPLEMENTATION**

**3.1. TECHNICAL PLATFORM**

**3.1.1. DEV C++**

Dev-C++ is a free full-featured integrated development environment (IDE) distributed under the GNU General Public License for programming in C and C++. It is written in Delphi.

Dev-C++ is generally considered a Windows-only program, but there are attempts to create a Linux version: header files and path delimiters are switchable between platforms.

An additional aspect of Dev-C++ is its use of DevPaks: packaged extensions on the programming environment with additional libraries, templates, and utilities. DevPaks often contain, but are not limited to, [GUI](https://en.wikipedia.org/wiki/Graphical_user_interface) utilities, including popular toolkits such as [GTK+](https://en.wikipedia.org/wiki/GTK%2B), [wxWidgets](https://en.wikipedia.org/wiki/WxWidgets" \o "WxWidgets), and [FLTK](https://en.wikipedia.org/wiki/FLTK). Other DevPaks include libraries for more advanced function use. Users of Dev-C++ can download additional libraries, or packages of code that increase the scope and functionality of Dev-C++, such as graphics, compression, animation, sound support and many more. Users can create Devpaks and host them for free on the site. Also, they are not limited to use with Dev-C++ - the site says "A typical devpak will work with any MinGW distribution (with any IDE for MinGW)".

**3.1.2. DESIGN LANGUAGE**

The origin of C++ dates back to 1979 when Bjarne Stroustrup, also an employee of Bell AT &T, started working on language C with classes. He borrowed desirable features from many other languages like Simula, Ada, ML, CLU and ALGOL 68. Thus, in addition to features of C language, C++ also included classes, strong type checking, default function argument and basic inheritance. Till 1983, it was called C with classes, and in 1983 it was named C++. During 1998, a joint ANSI-ISO committee released the specification for C++ language standards.

There were 5 primary goals in the creation of C++ language:

1. C++ is a highly portable language and is often the language of choice for multi-device, multi-platform app development.
2. C++ is an object-oriented programming language and includes classes, inheritance, polymorphism, data abstraction and encapsulation.
3. C++ has a rich function library.
4. C++ allows exception handling, and function overloading which are not possible in C.
5. C++ is a powerful, efficient and fast language. It finds a wide range of applications – from GUI applications to 3D graphics for games to real-time mathematical simulations.

**3.2. OOPS CONCEPTS IMPLEMENTED**

**3.2.1. DATA ABSTRACTION**

Data abstraction is the method of hiding certain details from the user and showing only the details which are required for the outside world to be seen. This can be implemented using “private” modes.

**Example:**

class MyFileHandler:public FileHandler {

private:

LiftHandler lh;

public:

void readFromFile();

void writeToFile(LiftRequest);

};

**3.2.2. DATA ENCAPSULATION**

Data encapsulation means combining of the data and the class member functions into a single unit. All the member functions in the project uses encapsulation concept as all of them uses class data member.

**Example:**

class LiftRequest {

public:

int dest;

int src;

int ppl;

void toString() {

cout<<"LIFT REQUEST -> SOURCE: "<<src<<" DESTINATION: "<<dest<<" PPL: "<<ppl<<endl;

cout<<endl;

}

};

**3.2.3. CLASSES AND OBJECTS**

Objects are the real time entities such as table, chair, building etc. Objects are the blueprints of the classes.

The project has used 5 classes:

1. class LiftRequest
2. class Lift
3. class LiftHandler
4. class FileHandler
5. class MyFileHandler : public FileHandler

**3.2.4. INHERITANCE**

[Inheritance](http://www.programtopia.net/cplusplus/docs/inheritance) is the process of inheriting properties of objects of one class by objects of another class. The class which inherits the properties of another class is called Derived or Child or Sub class and the class whose properties are inherited is called Base or Parent or Super class. When a single class is derived from a single parent class, it is called Single inheritance. It is the simplest of all inheritance.

**Example:**

class MyFileHandler : public FileHandler {

private:

LiftHandler lh;

public:

void readFromFile();

void writeToFile(LiftRequest);

};

**3.2.5. RUN TIME POLYMORPHISM**

Run time polymorphism is achieving which object method shall invoke during Runtime instead of compile time.  For instance, you have 2 classes B and C derived from a base Class A. And you create a pointer to the class A and make it point to any of the derived class object. And when you call a method using that pointer, the corresponding derived class method is called instead of base class method.

**3.2.6. VIRTUAL FUNCTIONS**

A virtual function is a member function which is declared within base class and is re-defined (Overridden) by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

**Example:**

class FileHandler {

public:

virtual void readFromFile()=0;

virtual void writeToFile(LiftRequest)=0;

};

**3.2.7. FRIEND FUNCTIONS**

Friend functions are the functions which are used to access the data members of a class and are not the member functions.

Friend functions are only declared inside a class and are defined outside a class.

**Example:**

class Lift {

public:

LiftRequest lr;

Lift() {

lr.src=0;

lr.dest=0;

lr.ppl=0;

}

friend void toString(int, Lift);

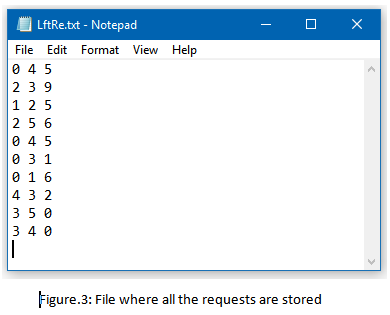
};

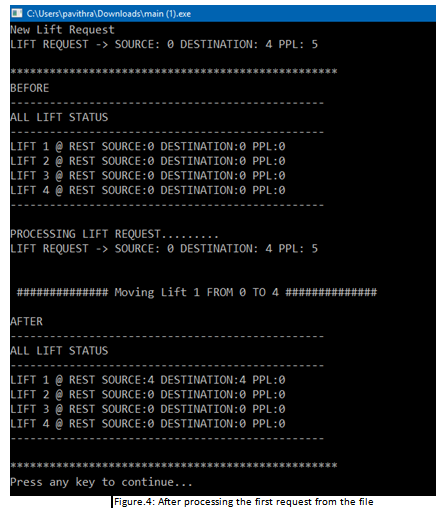
void toString(int id, Lift l) {

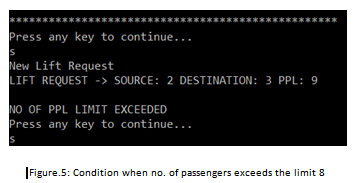
cout<<"LIFT "<<(id+1)<<(l.lr.src==l.lr.dest?" @ REST":"LIFT @ MOVING")<<" SOURCE:"<<l.lr.src<<" DESTINATION:"<<l.lr.dest<<" PPL:"<<l.lr.ppl<<endl;

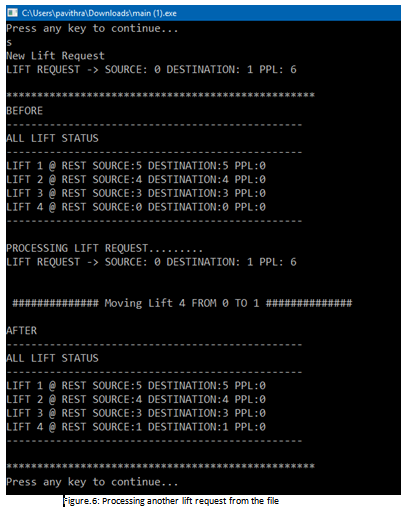
}

**CHAPTER 4:**

**SAMPLE OUTPUT**







**CHAPTER 5:**

**CONCLUSION**

**5.1. SUMMARY**

Simulation is a way of developing application or design simulating real world, the way which help in clarifying the real idea. The reasons upon using simulations may be for literature, cost, time, or safety reasons. Company problem appears when it is intended to equip its building with an elevator and want to know according to its needs the way to settle the elevator to best satisfy this needs. The way where we settle an Object-Oriented Software Simulator, concerning the logic and algorithm required to move the elevator between floors, giving the company a clear idea and leading it to choose the best design which satisfies this needs.

**5.2. FUTURE WORK PERSPECTIVES**

Some of future work which could be implemented:

1. Speed control may be available to the user of the simulator to see different simulations with different speeds.
2. The elevator needs the ability to change directions if the user decides to go down instead of up (and vice versa) when there are no other requests for that elevator.
3. Elevator will be supplied by inventor, in the case of electricity failure elevator will go down to the nearest floor and open its doors.
4. An emergency alarm switch will sound an alarm when activated by a passenger.
5. Elevator will be supplied by Priority system, which is used by specific users like managers, where they can press certain code making the elevator moving straight forward to the wanted floor without stopping in any floor.

**CHAPTER 6:**

**BIBLIOGRAPHY**

**6.1. REFERENCES**

* [http://en.wikipedia.org/wiki/Elevator#History](http://en.wikipedia.org/wiki/Elevator%23History%20)
* [http://www.us.schindler.com/sec\_index/sec\_kg/sec\_kg\_safety/sec\_kg\_profile\_safety\_el evatorsafetyfeatures.htm](http://www.us.schindler.com/sec_index/sec_kg/sec_kg_safety/sec_kg_profile_safety_el%20evatorsafetyfeatures.htm%20)
* <http://www.elevatorchallenge.com>
* <http://www.angelfire.com/trek/software/elevator.html>
* <http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/>
* <http://www.cnet.com/Content/Features/Dlife/Bugs/?dd>
* Hermann Kopetz. Real-Time Systems, Design Principles for Distributed Embedded Applications.

**6.2. APPENDICES**

[1] The Book of Secrets - Kitab al Asrar of al-Muradi - part 1 of 2

[2] <http://www.popularmechanics.com/science/extreme_machines/1280851.html>

[3] The Elevator Museum, timeline

[4] <http://www.cooper.edu/facilities/library/archive/symbol/symbol5.html>

[5] The History of the Elevator - Elisha Otis

[6] Ralph Turvey, London Lifts and Hydraulic Power, Transactions of the Newcomen Society, Vol. 65, 1993-94, PP.147-164

[7] Hetzel, William C. The Complete Guide to Software Testing, 2nd ed. Publication info: Wellesley, Mass.: QED Information Sciences, 1988.