2016

An Investigation of Multi-Variable Optimization Applied to the Hospitality Industry

Peng Gao
Lehigh University

An Investigation of Multi-Variable Optimization Applied to the Hospitality Industry

by

Peng GAO

A Thesis

Presented to the Graduate and Research Committee of Lehigh University in Candidacy
for the Degree of Master of Science

in

Mechanical Engineering

Lehigh University

05/2016

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

Date

Thesis Advisor

Chairperson of Department

Table of Content

Abstract

This thesis focused on a complex multi-variable multi-objective optimization problem applied to the hospitality industry. Unlike running a normal hotel, running a condo hotel needs to make both the guests and the unit owners satisfied. To make guests happy, hotel operators try the best to fulfill requests of the guests. On the other hand, making owners happy requires that those same operators provide a transparent and balanced income summary for each owner. Opproprately balancing, these often conflicting objectives is crucial in condo hotel management. Unfortunately, even though current commercial packages claimed to contain equalization method, the result using them is negative. This thesis introduced an engineering optimization method to this problem. Several optimization methods were considered, and one that was selected as most appropriate was developed and tested with sample potential input information. The result demonstrated a marked increase in performance once optimization method was applied.

Key words: multi-variable multi-object opitimization, condo hotel management, equanlization method, greedy algorithm.

1 Introduction and objectives

1.1 Complex multi-variable multi-objective optimization problems

In mathematics, computer science and operations research, mathematical optimization (alternatively, optimization or mathematical programming) is the selection of a best element (with regard to some criteria) from some set of available alternatives. An optimization problem is the problem of finding the best solution from all feasible solutions. An optimization problem often includes multiple variables and adds more than one objective. This kind of optimization problem is called multi-variable multi-objective optimization.

Multi-objective optimization (also known as multi-objective programming, vector optimization, multicriteria optimization, multiattribute optimization or Pareto optimization) is an area of multiple criteria decision making, that is concerned with mathematical optimization problems involving more than one objective function to be optimized simultaneously. Multi-objective optimization has been applied in many fields of science, including engineering, economics and logistics where optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives. There are examples of multi-objective optimization applications in many fields of study. In finance, a common problem is to choose a portfolio when there are two conflicting objectives, the desire to have the expected value of portfolio returns be as high as possible, and the desire to have risk, often measured by the standard deviation of portfolio returns, be as low as possible. In engineering, many problems involve multiple objectives which are not describable as the-more-the-better or the-less-the-better; instead, there is an ideal target value for each objective, and the desire is to get as close as possible to the desired value of each objective. For example, energy systems typically have a trade-off between performance and cost or one might want to adjust a rocket's fuel usage and orientation so that it arrives both at a specified place and at a specified time.

One of the best examples of a complex multi-variable optimization problems lies in the hospitality industry.

1.2 Hospitality problem description

A condo hotel, also known as a hotel-condo or a Condotel, is a building, or multi-building facility that is legally a condominium but which is operated as a hotel, offering short term rentals, and which maintains a front desk.

Condo hotels are typically high-rise buildings developed and operated as luxury hotels, usually in major cities and resorts [1]. These hotels have condominium units which allow someone to own a full-service vacation home. When they are not using this home, they can leverage the marketing and management done by the hotel chain to rent and manage the condo unit as it would any other hotel room.

Starting in Europe in the 1970s, condo hotels have been a big success across the world. According to WTO research, over 150,000 families held condo hotel properties over 500 vacation resorts worldwide in 1980[2]. By the end of 1995, 350,000 families bought their unit in over 5000 condo hotels in over 81 countries. At that time, the total sale of condo hotels worldwide was 4 billion dollars. But in 2001, the number increased to 8 billion dollars and has kept a 15% annual growth rate. In 2004, the number hit 30 billion dollars worldwide [3].

The reason for this successful and dramatic growth is that a condo hotel, if operated correctly, benefits both the owners and the guests. Different with homestay, where the owners have to list their properties on a website for rent, a condo hotel is operated by a management team hired by the owner's leadership board. Running like a regular hotel, owners don't have to contact guests, offer and recover keys; do the cleaning before a guest comes and after the guest leaves. The management team will take care of those businesses. Like a regular hotel, a condo hotel is often equipped with pools, spas, fitness rooms and other facilities that regular hotels have. Owners are usually allowed to use those amenities for free during their stay at the hotel [4]. For guests, a condo hotel room is more like an apartment unit fully equipped with a kitchen and all modern conveniences. Guests can enjoy a more leisured stay in such a unit. In addition, a condo hotel unit is often more carefully decorated than a regular hotel room, which makes the stay more enjoyable.

Different from a regular hotel, a condo hotel is owned by different owners and operated by a management company hired by the owner's board [5]. The financial distribution is different too. In a condo hotel, after paying the management team, each owner's income is often determined by how many nights his/her room is rent. Two owners holding rooms of the same type can theoretically end up with quite different levels of annual income.

The goal of running a successful condo hotel is to make both the customers and unit owners satisfied. To make the guests happy, the management team should run the hotel without having the guests realize this is anything other than a normal hotel, which means providing high quality services and accommodations as a normal hotel to best fulfill customers' demands. To make unit owners happy, management team needs to assure a fair distribution of the collective income to each owner and try to balance every owner's income regarding how they use their own room.

The reservation requests presented by customers provide the input variables in this problem. Each reservation contains a reservation date, desired unit type, check-in date and check-out date. In addition, however, it might contain extra information such as, is this reservation made by the owner or does the guest have special room request. In most hotels, guests are often given the chance to specify preferences or to choose their favorite among the rooms available in the hotel. Rooms with following characteristics would most likely to be reserved more. Those located closer to elevators, or those with good sunlight coming in might get more. For a beach resort hotel, oceanfront rooms with a balcony usually get more reservations than the others. So, fulfilling guests' requests and balancing the income for owners might be conflicting. So managing reservations for a condo hotel is a perfect example of a complex multi-variable, multi-objective optimization problem.

The remainder of this thesis concentrates on this problem and tries to solve it using a secience-based engineering approach.

1.3 Specified application

The investigation started with an evaluation of 3 condo hotels as examples starting with the case of the Flanders Hotel. The hotel was built in 1922 in Ocean City New Jersey. It was originally designed to serve as one of the best hotels in America to provide an oceanfront resort for successful businessmen and their families. The hotel operated as a very successful luxury resort destination for decades. In 1996, the building was completely removed into a collection of individually owned condo units. In 2010, The Flanders Condominium Association acquired the 2nd floor banquet center, offices, conference rooms and adjacent parking lot. The acquisition marked an exciting return to the fundamental concept behind this historic grand hotel. The purchase reunited all of the Flanders' stunning facilities under common ownership, and restored seamless cohesion to the operation of the hotel's wide array of services. The hotel has undergone many modifications since it opened, and the millions of dollars invested in the building over the recent years have completely refreshed the hotel's striking interior and exterior, modernizing and beautifying the guest suites, pool and deck [6].

Figure 1 one bedroom deluxe suite of the Flanders Hotel

Picture taken from Flanders Hotel website, http://theflandershotel.com/

The Flanders Hotel contains approximately 100 one bedroom suites, that are held by different owners and collectively participate in a common rental program. The hotel also holds two bedroom suites, three bedroom suites and even five bedroom suites.

The second condo we studied is also located at a beach resort. Churchill Suites Monte Carlo is located in Miami Beach Florida, one of the major beach resorts in the southeast of the USA. This 13-floor building is a combination of a residential apartment facility and a condo hotel. The 36 suites running as a condo hotel are located on the floors 6, 7, 12 and 16. All of these 36 suites include one bedroom, a living room with an ocean view, a full kitchen and a modern bathroom. The deluxe suites also have a balcony with a good view.

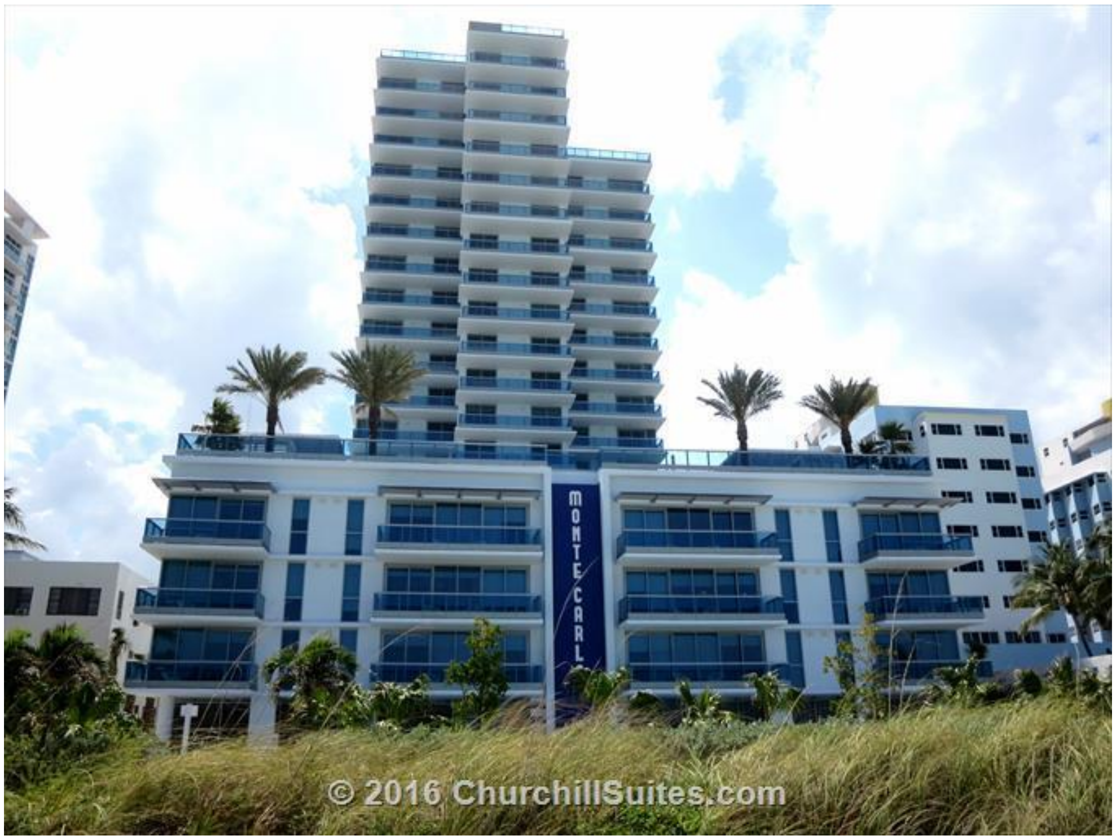


Figure 2 outside of the Churchill Suites Monte Carlo Miami Beach

Picture taken from the Churchill Suites Monte Carlo Miami Beach website, http://churchillsuites.com

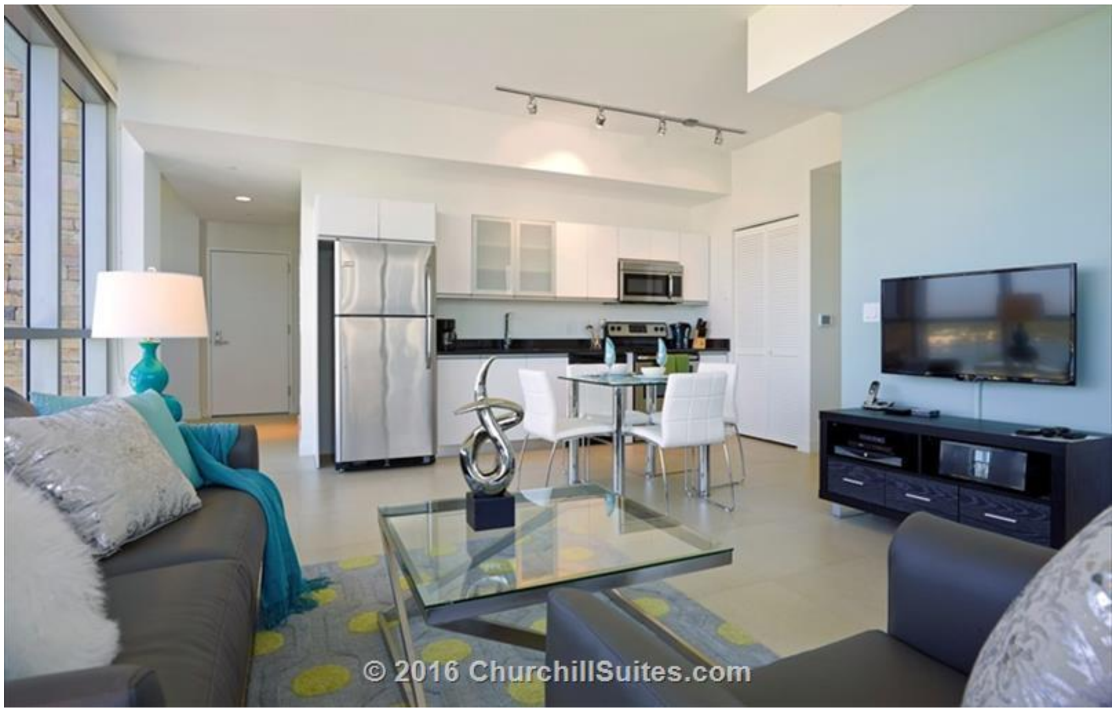


Figure 3 one bedroom suite of Churchill Suites Monte Carlo Miami Beach

Picture taken from the Churchill Suites Monte Carlo Miami Beach website, http://churchillsuites.com

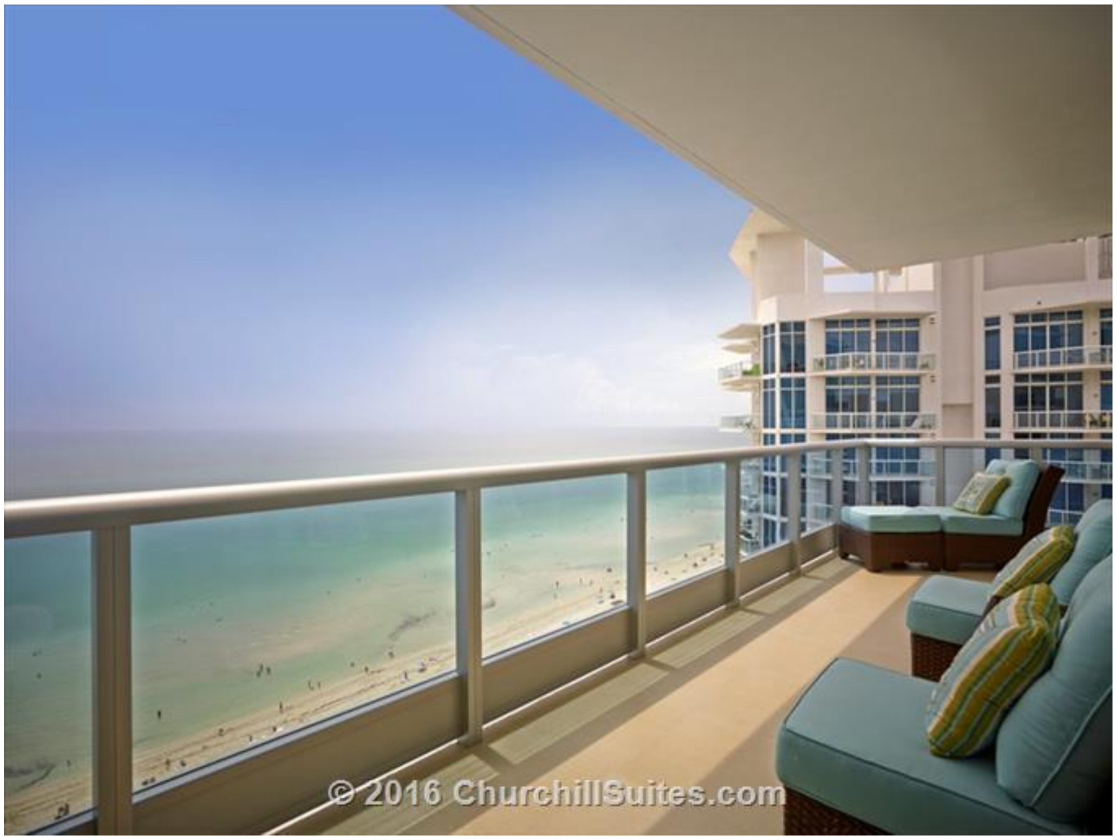


Figure 4 Balcony of one bedroom oceanfront suite of Churchill Suites Monte Carlo Miami beach

Picture taken from the Churchill Suites Monte Carlo Miami Beach website, http://churchillsuites.com

The last example is Trump International Hotel in Las Vegas Nevada. This modern condo hotel run by the Trump International Hotel company, is considered one of the finest hotels in the world. This condo hotel holds 1282 different suites, from one bedroom to three bedrooms, providing different views of the city of Las Vegas. Every room is equipped with a massage tub, a living room and a full kitchen. This hotel also features an on-site spa with 9 treatment rooms and an outdoor pool.

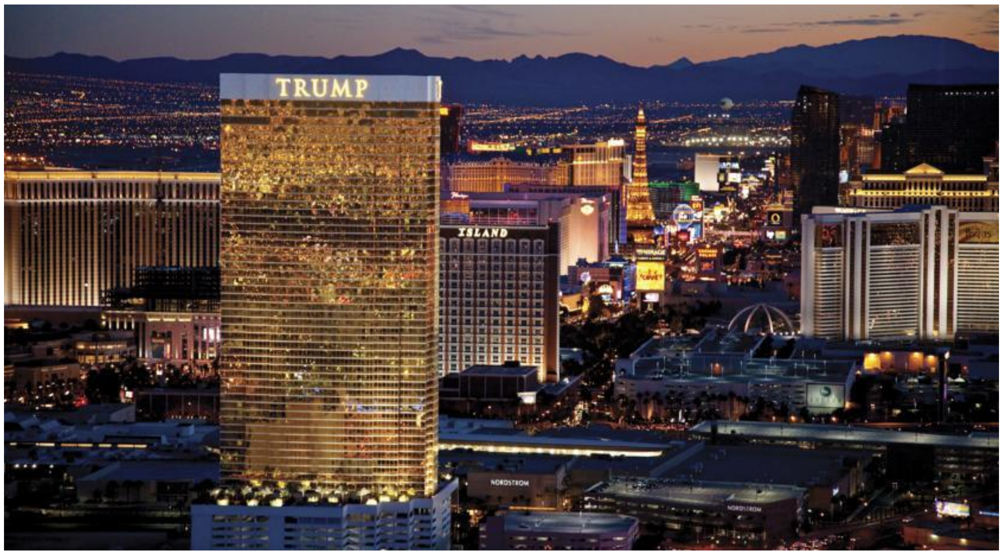


Figure 5 outside of Trump International Hotel in Las Vegas

Picture taken from Trump International Hotel Las Vegas website, https://www.trumphotelcollection.com/las-vegas/

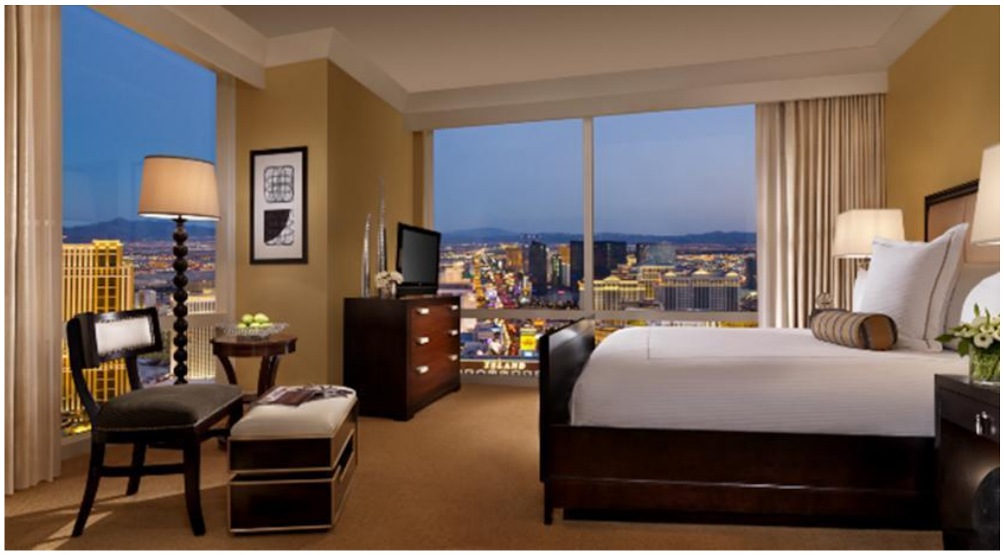


Figure 6 one bedroom city view suite of Trump International Hotel in Las Vegas

Picture taken from Trump International Hotel Las Vegas website, https://www.trumphotelcollection.com/las-vegas/

Figure 7 treatment room of Trump International Hotel in Las Vegas

Picture taken from Trump International Hotel Las Vegas website, https://www.trumphotelcollection.com/las-vegas/

All the three of these example hotels are operated by a central management unit, but the rooms are held by different owners. That means balancing the income for different owners while getting the maximum income for the whole hotel is very crucial.

1.4 Research goals and objectives

Our primary goal is to build software using a method considering equalization between owners and test it with real data and analyze the result. But unfortunately, we were not able to get actual complete reservation data for any condo hotel. With that in mind, a random but very relevant set of data was created for utilization in this study.

The process of creating and applying a reservation assignment method is divided in to the following steps:

a) Develop a numerial program that creates a set of random reservation data including reservation date, check-in date, check-out date and etc.
b) Develop a program algorithm that arranges the reservations using a method with equalization.
c) Develop a program that arranges the reservations using a method without equalization as conventional result.
d) Analyze and compare the results.

1.5 Research approach and structure of the thesis

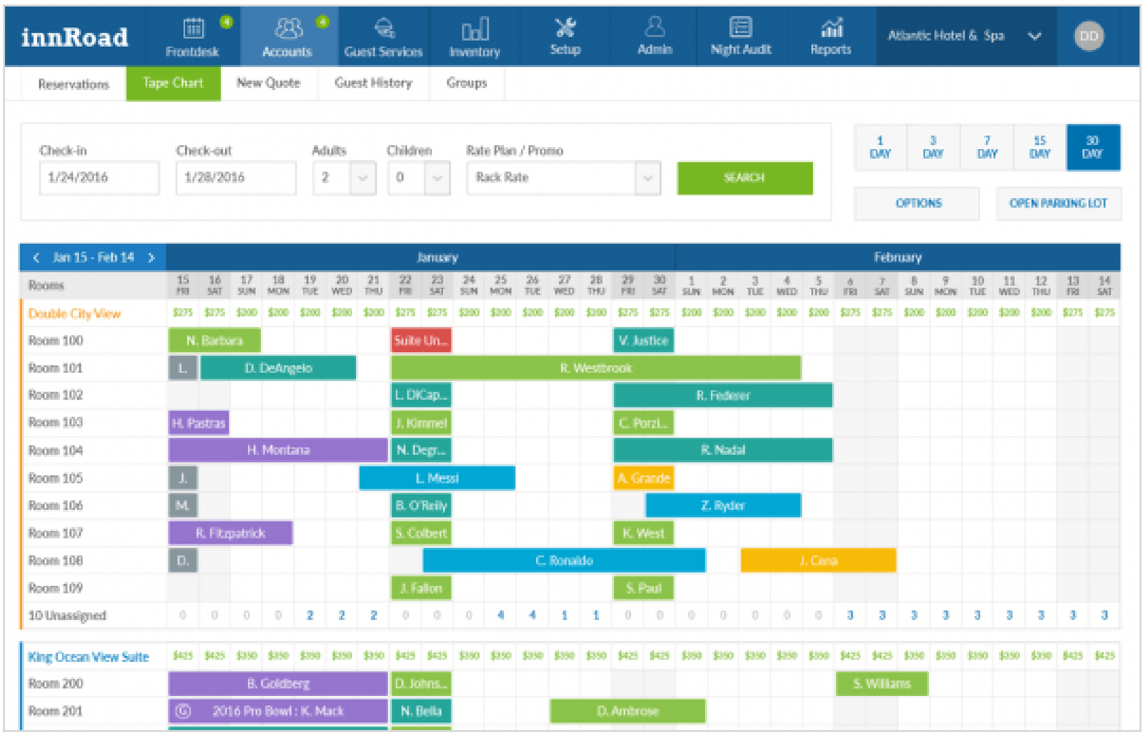
Before starting programming, in chapter 2, we dig deeper into the case of the Flanders Hotel example to see how the current software is working and how the management team is solving the balancing problem. Then we introduce our method with equalization in chapter 3 and introduce the variables and logic for programming.

For programming, we will first introduce a new kind of reservation and build a project that creates the random set of reservation data. This part will be demonstrated in chapter 4. We then build a project using a method with equalization to arrange the set of data, to prove the method is correct and the software is helping. This part is demonstrated in chapter 5. We then introduce another type of reservation, and rebuild the project for the final version of data, and rebuild the project for reservation arrangement including rate for each night. We can get the annual income report for the random data using the method with equalization. Then we build a conventional project using a method without equalization that current software is using to arrange the random data. This part is demonstrated in chapter 6. In chapters 7 and 8 we compare the results and analyze the two methods. In chapter 9 we introduce human involvement to this program to give the management team some freedom running the condo hotel with the equalized method. Finally, in chapter 10, we make conclusions and discuss the future improvement of the program.

2 Related scientific development

2.1 State of the art

Hotel management in recent days is mostly operated by commercial software. The software differs from each other but can all help run a traditional non-condo hotel perfectly. Naming a few, InnRoad is built for managing small size or medium size hotel. InnRoad contains a PC to mobile reservation system, and can be operated on Window, Apple and Linux systems. InnRoad is favored by small business users for its reasonable price.



Check-in: 1/24/2016
Check-out: 1/28/2016
Adults: 2
Children: 0
Rate Plan / Promo: Rack Rate

Jan 15 - Feb 14

Rooms	15 FRI	16 SAT	17 SUN	18 MON	19 TUE	20 WED	21 THU	22 FRI	23 SAT	24 SUN	25 MON	26 TUE	27 WED	28 THU	29 FRI
Double City View	$275	$275	$200	$200	$200	$200	$200	$275	$275	$200	$200	$200	$200	$200	$275
10 Unassigned	0	0	0	0	2	2	2	0	0	0	4	4	1	1	0
King Ocean View Suite	$425	$425	$350	$350	$350	$350	$350	$425	$425	$350	$350	$350	$350	$350	$425

Rooms	30 SAT	1 SUN	2 MON	3 TUE	4 WED	5 THU	6 FRI	7 SAT	8 SUN	9 MON	10 TUE	11 WED	12 THU	13 FRI	14 SAT
Double City View	$275	$200	$200	$200	$200	$200	$275	$275	$200	$200	$200	$200	$200	$275	$275
10 Unassigned	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
King Ocean View Suite	$425	$350	$350	$350	$350	$350	$425	$425	$350	$350	$350	$350	$350	$425	$425

Rooms	Reservations
Room 100	N. Barbara; Suite Un...; V. Justice
Room 101	L.; D. DeAngelo; R. Westbrook
Room 102	L. DiCap...; R. Federer
Room 103	H. Pastras; J. Kimmel; C. Porzi...
Room 104	H. Montana; N. Degr...; R. Nadal
Room 105	J.; L. Messi; A. Grande
Room 106	M.; B. O'Reily; Z. Ryder
Room 107	R. Fitzpatrick; S. Colbert; K. West
Room 108	D.; C. Ronaldo; J. Cena
Room 109	J. Fallon; S. Paul
Room 200	B. Goldberg; D. Johns...; S. Williams
Room 201	2016 Pro Bowl : K. Mack; N. Bella; D. Ambrose

` Figure 8 InnRoad software tape chart module. Picture taken from http://www.softwareadvice.com/hotel-management/innroad-profile/

One of the most widely used software package is the Infor Hospitality Suite. It is used to manage properties around the globe, specifically over 20,000 hotels in more than 100 countries. The software is suitable for many types of properties but is typically recommended to hotel chains, campgrounds, resorts, cruise lines, casinos, and government lodges. The system offers integrated hospitality management applications including front desk & property management, central reservations, housekeeping, CRM, financials, business intelligence, and maintenance management. Comparing to InnRoad, Infor Hospitality Suite is suitable for small to large size hotels, and contains a module of saving guests profile and comments. This module helps large hotel managers to contact guests more often to get feedback and provide discounts for such guests.

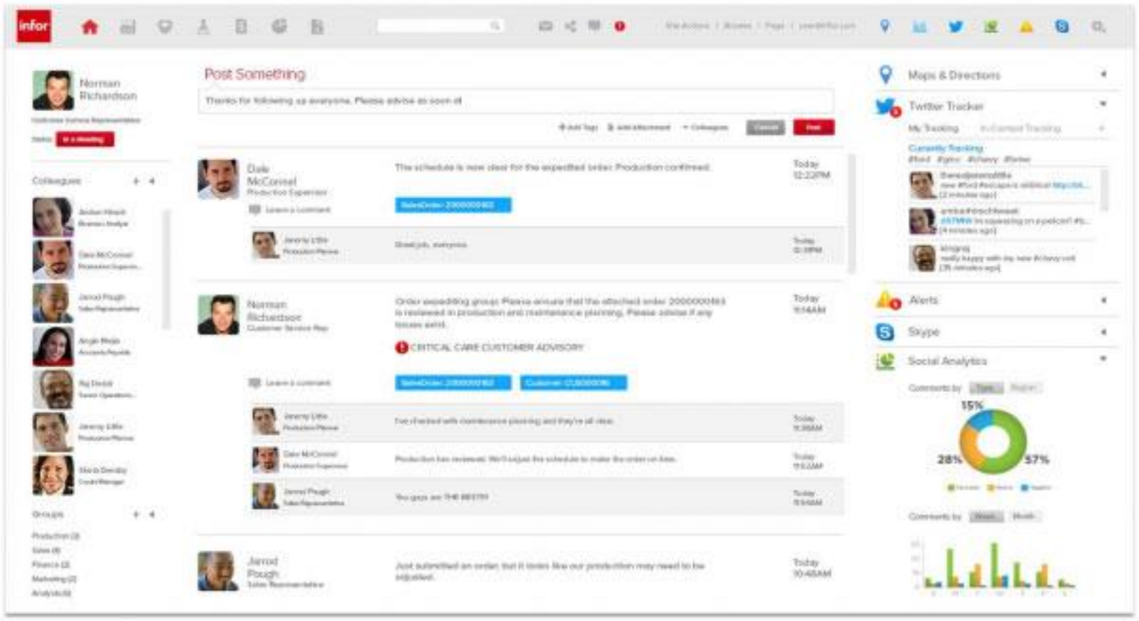


Figure 9 Infor Hospitality Suite software guest management module. Picture taken from

http://www.softwareadvice.com/hotel-management/inforhotel-profile/

But when it comes to condo hotel management software, there are just a few choices. There are some widely used software, like RDP (resort data processing), TimeShareWare and Guest Tracker. Guest Tracker has been used for over 15 years [7]. It is designed for regular hotels and doesn't consider the equalization at all. TimeShareWare allows manager to set a rotation rule to be fair [8]. But I didn't get a demo version of the software, so I don't know how it works. RDP provides two versions, one for regular hotels and one for condo hotels. The description for condo hotel version says "RDP software helps equalize rental revenue for all owners and takes into account reservation length, seasons, property type, property location, owner use, guest of owner, and reservation cancellations. Various reports are available to show revenue per unit each month and year-to-date. Available units are displayed in "turn to be rented order" but still allow reservationists to select any available unit based on guest preferences to keep guests happy." According to the aompany's website, the RDP rental equalization logic is time tested, as it has now been in use for 26 years. Long standing RDP customers have analyzed the results and determined rental revenues fall within a few percentage points for similar units, while still allowing flexibility to select units based on guest preferences." [9]

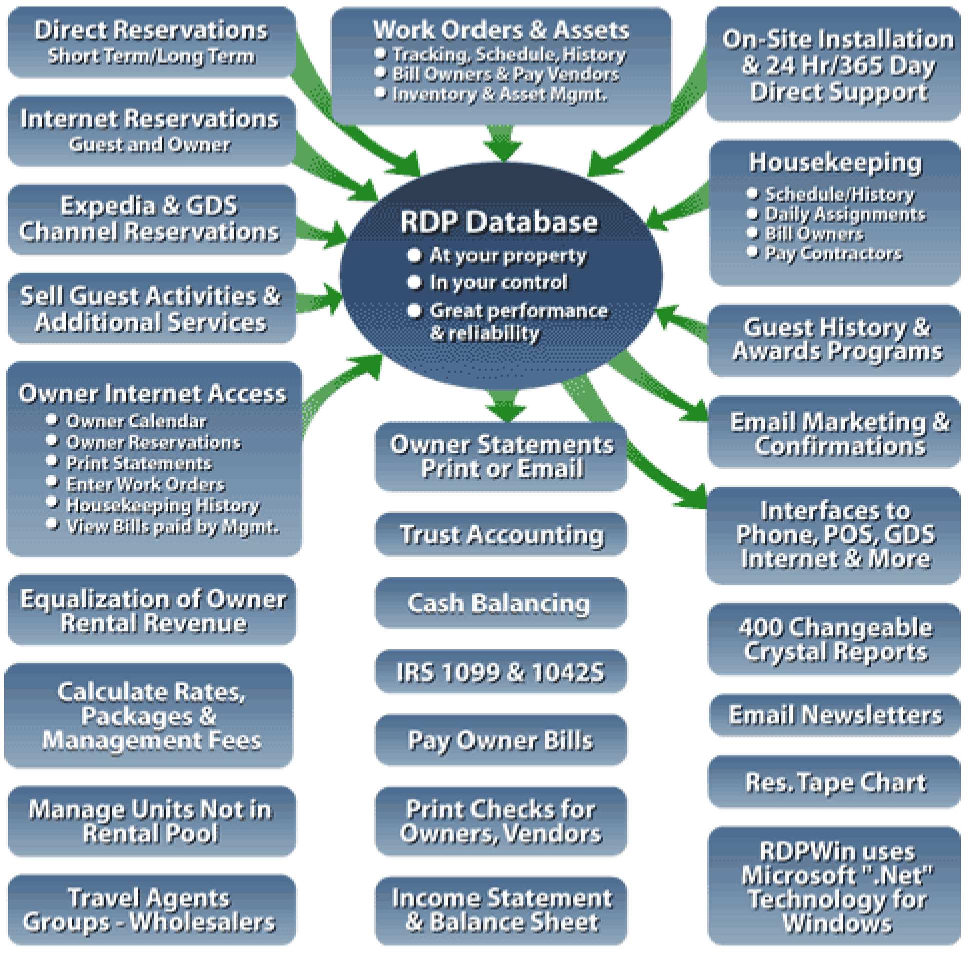

Direct Reservations
Short Term/Long Term
Work Orders & Assets
Tracking, Schedule, History
Bill Owners & Pay Vendors
Inventory & Asset Mgmt.
On-Site Installation & 24 Hr/365 Day Direct Support
Internet Reservations
Guest and Owner
Housekeeping
Schedule/History
Daily Assignments
Bill Owners
Pay Contractors
Expedia & GDS Channel Reservations
RDP Database
At your property
In your control
Great performance & reliability
Sell Guest Activities & Additional Services
Guest History & Awards Programs
Owner Internet Access
Owner Calendar
Owner Reservations
Print Statements
Enter Work Orders
Housekeeping History
View Bills paid by Mgmt.
Owner Statements Print or Email
Email Marketing & Confirmations
Trust Accounting
Interfaces to Phone, POS, GDS Internet & More
Equalization of Owner Rental Revenue
Cash Balancing
400 Changeable Crystal Reports
IRS 1099 & 1042S
Calculate Rates, Packages & Management Fees
Email Newsletters
Pay Owner Bills
Res. Tape Chart
Manage Units Not in Rental Pool
Print Checks for Owners, Vendors
RDPWin uses Microsoft ".Net" Technology for Windows
Travel Agents Groups - Wholesalers
Income Statement & Balance Sheet

Figure 10 Modules for ResortDataProcessing(RDP). Picture taken from RDP website, http://sales.resortdata.com/CondoHotel.htm

The Flanders Hotel is using RDP to arrange the reservations. After hearing from the management team for the Flanders Hotel, I learned that the software apparently builds a rotation list by the room numbers from 1 to 100. When a reservation comes, the software checks the availability for each room according to such order and puts the reservation into the first available room and puts the room to the bottom of the list. This methology contains no equalization.

In this case, a human decision maker (DM) is involved. It is very normal to introduce DM into a multi-object optimization problem, because the objects are conflicting to each other. But what the management team does to help improve the imbalance brought on by the RDP software is to check the total income for each room every two weeks. When they find that a particular unit is falling far behind, they manually put more reservations into that room until it catches up.

This method helps improve the situation to some extent. But there are two major problems in such a method. Firstly, as we all know, the room rate for a beach resort hotel varies during the year. Rates will be higher in summer while lower during the rest of the year. At the same time, guests tend to reserve more during the summer compared to other seasons. In this case, if a room gets no or little reservations during the summer session and falls behind, it requires many more reservations to be assigned in the rest of the year to catch up. And sometimes it will never catch up because the difference is too magnificent or there are simply not enough reservations in autumn and winter. Secondly, sometimes an owner falls behind because he or she is over using his or her own unit. In this case, the management team would also allow this owner to catch up during the rest of the year if they did not realize the root causes problem. So in the end of the year, an owner who uses his or her room more could end up with equal income to that of an owner that does not use the room at all during the year.

So we need to build a mathematical method to help decrease human involvement, and improve the situation.

2.2 Software basis

To improve this situation, I decided to build a project to arrange the reservation using a method with equalization. The project should be able to handle reservation data, and provide detailed information about how each reservation is handled, and provide annual income reports. Instead of providing a visual interface, I decided to use C++ to build the project and use Microsoft Excel as the final output.

3 Scientific approach

By communicating with the management team of the Flanders Hotel, we know that there are 4 types of reservations. They are regular reservations, owners' reservations, reservations requiring a certain room, and other reservations.

A regular reservation is a reservation made by a paying guest, and can be assigned to any unit that is available. The unit taking such reservation gets the full payment.

An owner's reservation is a reservation made by owner to stay in his/her own property to host family and/or friends for vacation. In a condo hotel, the leadership boards often allow owners to use their unit to a certain extent with no income penalty. If an owner uses more than that number, he/she is defined as "over using", and his/her income should be influenced.

A reservation requiring or requsting a certain room is a reservation made by a paying guest who asks for a certain unit. This would most likely happen with returning customers. If a guest has stayed in the hotel before and enjoyed the stay in a certain room, the guest would more likely ask for the same room the next time. For this situation, the hotel will try to put the reservation into the room required. But if it is not available, any room taking such reservation gets the full payment.

There are still more kinds of reservations. For example, at times certain guests stay at the hotel for free for advertising purposes. These reservations can be assigned to any room available but the owner makes no income. Since this does not occur too frequently, such reservations are not included in this thesis.

4 Getting reservation data

For simplicity it was assumed that there are 100 one-bedroom units of the same size and condition in a subject property. Our goal is to equalize the income for those 100 owners. It is also assumed that the condo hotel is busier during May to Semptember, as would be the case for a U.S. based beach resort. August is always fully booked while in the other months, each room is booked for around 23 nights per month. From October to April, the hotel is much less booked for about 13 nights per month for each room. During the summer vacation for students (June, July and August), guests tend to stay longer in the hotel to enjoy the vacation at the beach, while for the other months, most guests just come to stay for a weekend. Accordingly, we assume a distribution of the reservations as followed.

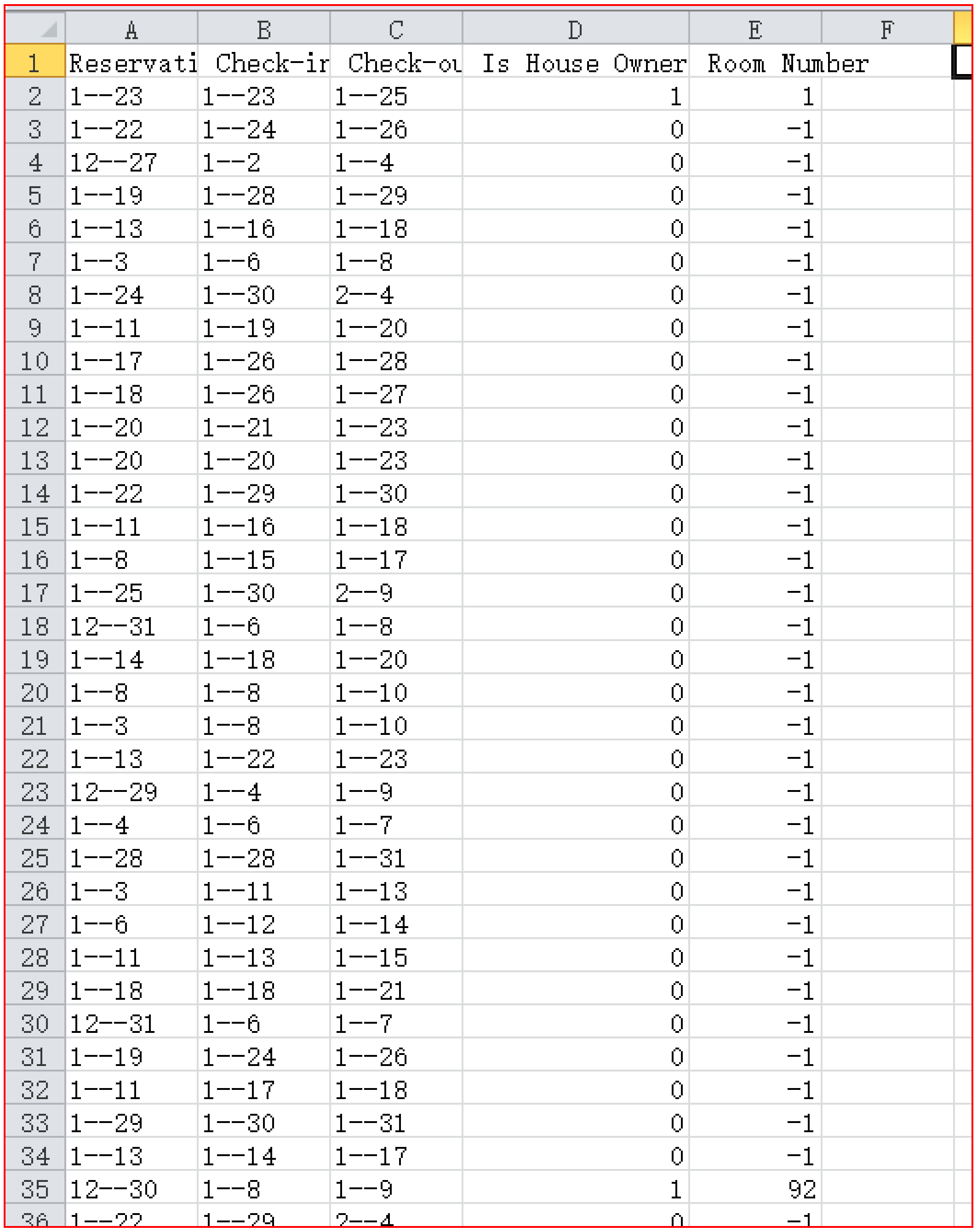
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Nights booked	13	13	13	13	23	23	23	28	23	13	13	13
1-night reservation	35%					5%			35%			
2-nights	35%					30%			35%			
3-nights	20%					30%			20%			
4-nights	7%					15%			7%			
5-nihgts	1%					10%			1%			
6-nights	1%					5%			1%			
7+nights	1%					5%			1%			

Table 1 distribution of the reservations

Owners' reservations take 5% of total reservations, and the owners can only book their own rooms.

Each reservation should contain the following information: reservation date, check-in date, check-out date, owner's reservation flag and room number. The output is a Microsoft Excel file containing 5 columns. In the "Is house owner" column, 1 means the reservation is made by owner, 0 means the reservation is made by a paying guest. And in the "Room number" column, -1 means the reservation can be put into any available room; all other numbers mean that the reservation can only be arranged in the room number shown.

For example:



	A	B	C	D	E	F
1	Reservati	Check-ir	Check-ou	Is House Owner	Room Number	
2	1--23	1--23	1--25	1	1	
3	1--22	1--24	1--26	0	-1	
4	12--27	1--2	1--4	0	-1	
5	1--19	1--28	1--29	0	-1	
6	1--13	1--16	1--18	0	-1	
7	1--3	1--6	1--8	0	-1	
8	1--24	1--30	2--4	0	-1	
9	1--11	1--19	1--20	0	-1	
10	1--17	1--26	1--28	0	-1	
11	1--18	1--26	1--27	0	-1	
12	1--20	1--21	1--23	0	-1	
13	1--20	1--20	1--23	0	-1	
14	1--22	1--29	1--30	0	-1	
15	1--11	1--16	1--18	0	-1	
16	1--8	1--15	1--17	0	-1	
17	1--25	1--30	2--9	0	-1	
18	12--31	1--6	1--8	0	-1	
19	1--14	1--18	1--20	0	-1	
20	1--8	1--8	1--10	0	-1	
21	1--3	1--8	1--10	0	-1	
22	1--13	1--22	1--23	0	-1	
23	12--29	1--4	1--9	0	-1	
24	1--4	1--6	1--7	0	-1	
25	1--28	1--28	1--31	0	-1	
26	1--3	1--11	1--13	0	-1	
27	1--6	1--12	1--14	0	-1	
28	1--11	1--13	1--15	0	-1	
29	1--18	1--18	1--21	0	-1	
30	12--31	1--6	1--7	0	-1	
31	1--19	1--24	1--26	0	-1	
32	1--11	1--17	1--18	0	-1	
33	1--29	1--30	1--31	0	-1	
34	1--13	1--14	1--17	0	-1	
35	12--30	1--8	1--9	1	92	
36	1--22	1--29	2--4	0	-1	

Table 2 an example of reservations

For this set of data, we can read out that the reservation on the first row is made by the unit owner, the unit number is 1. The reservation is made on January 23rd, and check-in is on the 23rd, check-out on the 25th. The second reservation is made by a paying guest on January 22nd. At this time, I didn’t include the reservations requiring a certain room for simplification.

A C++ program is develpoed to get a random set of data. The reservation period is a 365-days year. Check-in date starts on January 1st, and ends on December 31st. The code can be found in Appendix 1.

By running this project, we can get a set of reservation data. This set contains 8605 reservations including 443 owners’ reservations. We will use this set of data in the following chapter to check the reservation assignment project.

5 Reservation assignment project with equalization of nights stayed

The optimization method that was selected is fairly simple. We creating a unit list containing the following information: up to now, how many nights are reserved for this room and which dates are already booked. As each reservation comes, we update the room list to check availability. For all available rooms, we put the reservation in the room that is booked for the least nights. If there is more than one room booked for the same number of nights, we check the last check-out date, and arrange the reservation in the room booked least recently. We also keep record of owner's reservation and calculate the total occupancy time for each owner.

To accomplish the equalized result, we consult the greedy algorithm. A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum [10]. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time [11].

In this case, guests are checking-in and checking-out while reservations are being made, so we will never get a global optimum. A global optimum could only be found when we get a reservation data for the whole year and try to arrange the reservations by check-in date. This would lead to a situation where an earlier available reservation is abandoned to leave space for a later reservation. In real hotel management, this should not happen. So, we arrange the reservation by reservation date. The process is as followed.

i) Form a room list of 100 rooms and pair room number with total occupancy time and owner's occupancy time.

ii) Sort the reservation data by reservation date.

iii) Set the initial conditions: all room available for all date, occupancy time=0, last visit time=0, owner's occupancy time=0.

iv) When a reservation comes, check availability for all rooms from the start time until the end time for this reservation, and pick up all available rooms. If there is no room available, abandon the reservation and mark the reservation as not assigned.

v) Check if it is an owner's reservation.

   a) If it is, check the availability for the room required. If it is available, assign the reservation. Set availability to false for this room from the start date to end date, update the last visit time=end time, update occupancy time for this room and owner's occupancy time, set this reservation as arranged. If it not available, simply abandon this reservation and mark the reservation as not assigned.

b) If it is not an owner’s reservation, sort all the available rooms by occupancy time, and arrange this reservation in the room with the least occupancy time, then update the last visit time=end time, update occupancy time for this room, and mark this reservation as assigned. If there are more than one room available having the least occupancy time, we arrange the reservation in the room with the least last visit time (lease recently booked).

vi) Update the room list and consider the next reservation.

We repeat steps iv) through vi) for all reservations, we can get a locally optimized result using the greedy algorithm.

We build “reservation.h” to give definition to variables, use “main.cpp” to build the final room list and write files, use the file “reservation.cpp” to complete the assignment process shown above. The three files are shown in Appendix 2.

By running main.cpp we can get two files. One is the room list, it provides total occupancy time and owner’s occupancy time for each unit. The other file is a reservation list, it provides reservation date, check-in date, check-out date, owners reservation flag and if this reservation is assigned.

The room list file is shown below.

Room Number	Occupancy time	Owner Occupancy time
1	130	7
2	131	8
3	131	5
4	131	8
5	130	9
6	131	11
7	131	2
8	130	5
9	131	3
10	130	0
11	130	4
12	130	4
13	130	13
14	130	10
15	130	10
16	132	6
17	130	4
18	130	10
19	131	17
20	130	7
21	133	10
22	130	1
23	129	5
24	130	8
25	132	8
26	131	2
27	132	1
28	132	5
29	129	18
30	132	6
31	131	17
32	130	6
33	130	6
34	131	0
35	130	2

36	131	7
37	132	5
38	129	0
39	130	1
40	131	2
41	130	10
42	130	1
43	130	2
44	130	10
45	130	7
46	131	2
47	131	3
48	130	7
49	131	9
50	134	7
51	132	4
52	130	7
53	131	12
54	130	17
55	132	4
56	130	7
57	129	2
58	130	7
59	133	2
60	131	9
61	129	1
62	131	7
63	130	2
64	135	0
65	131	8
66	131	2
67	129	7
68	130	8
69	130	6
70	130	14
71	130	9
72	130	5
73	131	13
74	132	4
75	133	9

76	129	14
77	130	9
78	130	4
79	130	2
80	131	7
81	130	7
82	130	2
83	130	5
84	131	4
85	133	5
86	131	4
87	130	8
88	133	7
89	133	8
90	130	10
91	130	4
92	132	2
93	130	5
94	132	10
95	130	11
96	131	9
97	131	5
98	131	3
99	130	4
100	131	7

Table 3 reservations and owner's usage for each room

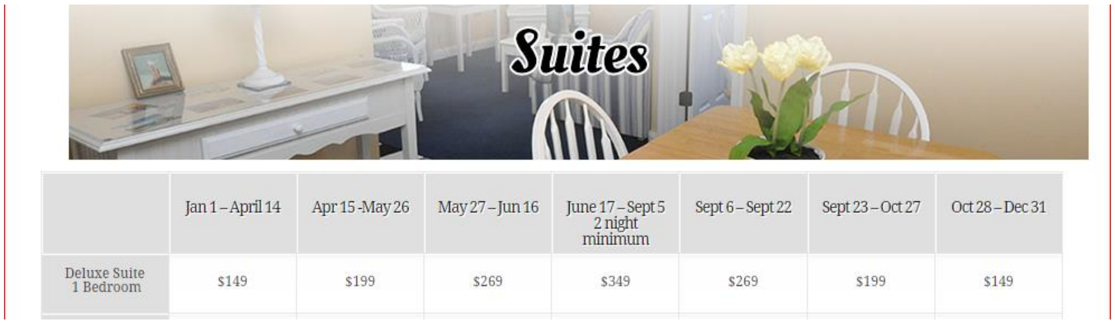
In this case, the average of total occupancy time turned out to be 130.7 and the standard deviation was 1.1237. The result is well equalized. The method is proved.

By this method and this program, we successfully balanced total nights reserved for each unit. We can apply the same method to balance the total income for each room after we introduce the rate. Details will be demonstrated in the next chapter.

6 Reservation arrangement with equalization of income and the conventional group

After comparing the three condo hotels mentioned in chapter 2, we found that the rate changes thoughout the year. Basically, rates during summer (June, July, and August) will be higher and special requirements such as a minimum of two nights is required for each stay. The rate for the rest of the year is cheaper, and there might be special events or discounts to attract guests. Based on this fact, it is acceptable to use the rate from one of the three hotels to build our program.

Firstly, we introduce the rate for the Flanders Hotel. The rates for 1-bedroom units for 2016 are shown below.



	Jan 1 – April 14	Apr 15 -May 26	May 27 – Jun 16	June 17 – Sept 5 2 night minimum	Sept 6 – Sept 22	Sept 23 – Oct 27	Oct 28 – Dec 31
Deluxe Suite 1 Bedroom	$149	$199	$269	$349	$269	$199	$149

Figure 11 rate for the Flanders Hotel of 2016

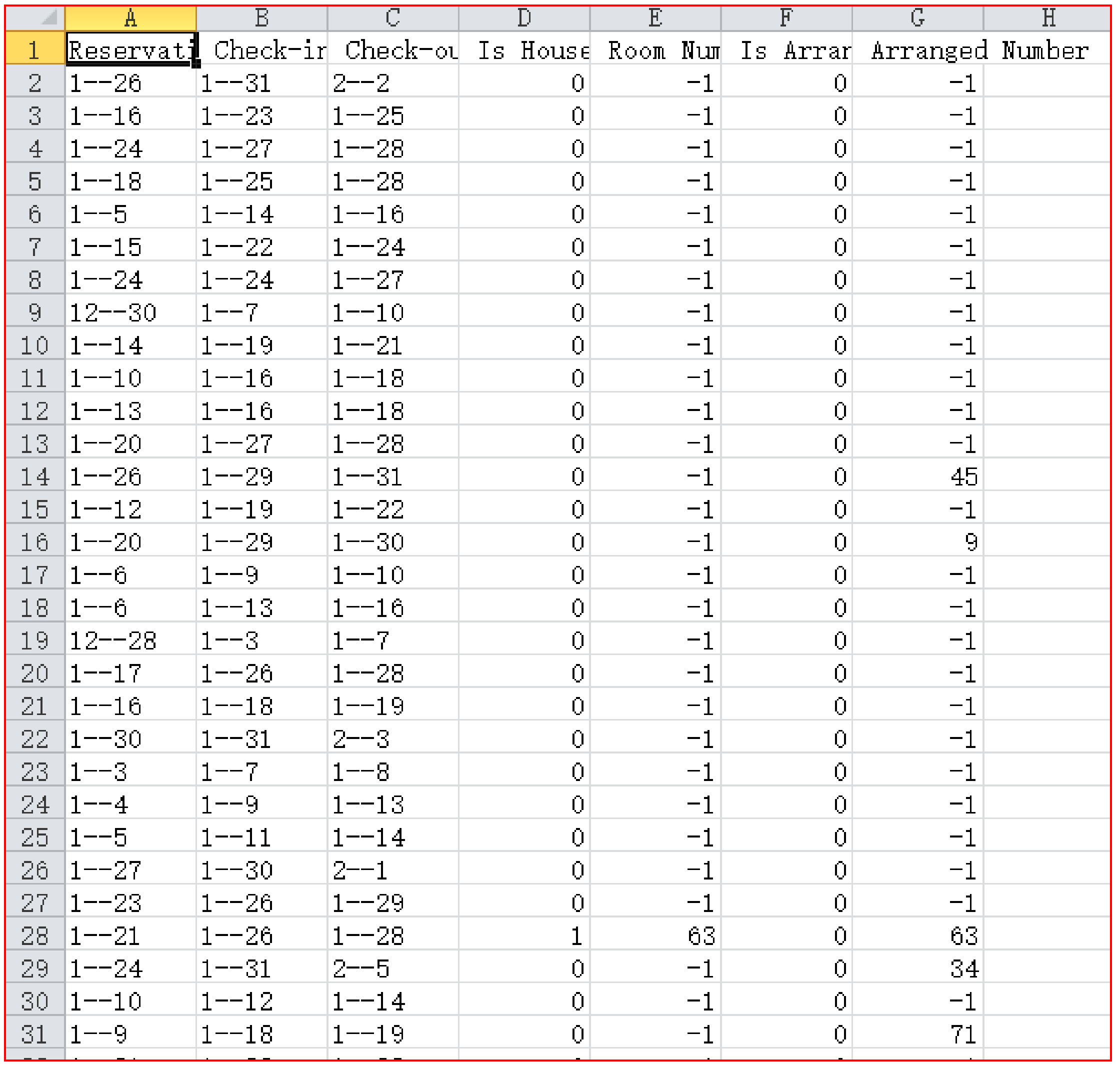
The rates are always changing during the year. To get a combined rate for a reservation, or total cost for a reservation, we introduce 7 moneysplits, which show the date the rate changes. These splitting points are counted from

January 1st, in a 365-day calendar year. The 7 splitting points end up at 103, 145,197, 247, 264, 299 and 364. If a reservation is across a money split date, the combined rate for the reservation should be calculate as $rate1 \times (moneysplittime - starttime) + rate2 \times (endtime - moneysplittime)$. By applying these conditions to our program, we can calculate the total income for each room after reading the reservation data.

We then introduce our third type of reservation mentioned in chapter 3. A reservation requiring a certain room is a reservation made by paying guest that requests a certain unit to stay in. Guests might ask for rooms located near elevators, closer to a dining hall, or on a higher floor for a better view of the ocean. Dealing with this type of reservation, we first try to fulfill the guest's request, and arrange the reservation in the room demanded. If the room is unavailable, we arrange it as a regular reservation which can be assigned to any available room. We modify the project for getting random reservation data to add this type of reservation. The code is shown in Appendix 3.

By running this project, we get a random set of data containing 8653 reservations.

Here is a brief view of the set of data.



	A	B	C	D	E	F	G	H
1	Reservati	Check-i	Check-o	Is House	Room Nu	Is Arra	Arranged Number	
2	1--26	1--31	2--2	0	-1	0	-1	
3	1--16	1--23	1--25	0	-1	0	-1	
4	1--24	1--27	1--28	0	-1	0	-1	
5	1--18	1--25	1--28	0	-1	0	-1	
6	1--5	1--14	1--16	0	-1	0	-1	
7	1--15	1--22	1--24	0	-1	0	-1	
8	1--24	1--24	1--27	0	-1	0	-1	
9	12--30	1--7	1--10	0	-1	0	-1	
10	1--14	1--19	1--21	0	-1	0	-1	
11	1--10	1--16	1--18	0	-1	0	-1	
12	1--13	1--16	1--18	0	-1	0	-1	
13	1--20	1--27	1--28	0	-1	0	-1	
14	1--26	1--29	1--31	0	-1	0	45	
15	1--12	1--19	1--22	0	-1	0	-1	
16	1--20	1--29	1--30	0	-1	0	9	
17	1--6	1--9	1--10	0	-1	0	-1	
18	1--6	1--13	1--16	0	-1	0	-1	
19	12--28	1--3	1--7	0	-1	0	-1	
20	1--17	1--26	1--28	0	-1	0	-1	
21	1--16	1--18	1--19	0	-1	0	-1	
22	1--30	1--31	2--3	0	-1	0	-1	
23	1--3	1--7	1--8	0	-1	0	-1	
24	1--4	1--9	1--13	0	-1	0	-1	
25	1--5	1--11	1--14	0	-1	0	-1	
26	1--27	1--30	2--1	0	-1	0	-1	
27	1--23	1--26	1--29	0	-1	0	-1	
28	1--21	1--26	1--28	1	63	0	63	
29	1--24	1--31	2--5	0	-1	0	34	
30	1--10	1--12	1--14	0	-1	0	-1	
31	1--9	1--18	1--19	0	-1	0	71	

Table 4 brief view of random data

The "Arranged number" column determines if a reservation comes with requirements. If the number is not -1, that means the reservation requires a certain room number, and the number shown in that column is the number of the room requested. So, the three kinds of reservations are marked as followed in the data set.

a) Regular reservation’s column D is marked as 0 and column E,G marked as -1

b) For an owner’s reservation, column D is marked as 1 and column E, G show the owner’s room number.

c) For a reservation requesting a certain room, column D is marked as 0, column E is marked as -1, and column G shows the room number required.

Regarding to the data shown above, row 28 is an owner’s reservation by the owner of room number 63. Rows 14, 16, 29 and 31 are reservations requesting certain rooms. The room numbers requisted are 45, 9, 34 and 71. The rest are normal reservations.

Speaking of equalization of total income, owners’ over use of their rooms should be considered. We assume that an owner can use their room with no penalty for 10 nights per year. Any more owners’ reservations beyond that should influence the owners’ income. By solving that problem, we introduce two variables: “moneyearn” and “moneyvirtual”. The variable “moneyearn” shows the real income for each owner and it is shown on the final result form. On the other hand, “moneyvirtual” is introduced for the “owner’s over using”. If this owner has reserved 10 nights or more already, his next owner’s reservation taken would add to “moneyvirtual” but not to “moneyearn” as a result. If this owner has reserved less than 10 nights, his next owner’s reservation would not add to any of these variables which means he/she is reserving with no income penalty. Second, for a regular reservation, we can put that reservation into any room that is available. Any rooms that are free during the reservation daters are available. If there is no room available, we discard the reservation and marked it as “not taken”. A regular reservation taken would add to both of “moneyvirtual” and “moneyearn” if it is taken. We check “moneyvirtual” for all the available rooms, and pick up the smallest number to take this reservation. Finally, for a reservation with a unit number request, we first check the requested unit. If it is possible, we put the reservation into that room and add the rate to both “moneyvirtual” and “moneyearn” variables. If the room reqested is not available, we consider such reservation as a regular one.

The arranging process is as followed:

i) Form a room list of 100 rooms and pair room number with total occupancy time, owner’s occupancy time, moneyearn and moenyvirtual.

ii) Sort the reservation data by reservation date.

iii) Set the initial conditions: all rooms available for all dates, occupancy time=0, last visit time=0, owner’s occupancy time=0, moneyearn=0 and moneyvirtual=0.

iv) When a reservation comes, check availability for all rooms from the start time till end time for this reservation, and pick up all available rooms. If there is no room available, abandon the reservation and mark the reservation as not assigned.

v) Check if it is an owner's reservation.

   a) If it is, check the availability for the room required. If it is available, arrange the reservation. Set availability to false for this room from the start date to end date, update the last visit time=end time, update occupancy time for this room and owner's occupancy time, set this reservation as arranged. If taken, check if the owner has already booked over 10 nights.

      i. If not over 10 nights, moneyearn and moneyvirtual should be 0 for this reservation, update moneyearn and moneyvirtual.

      ii. If owner has booked over 10 nights, moneyearn should be 0 and moneyvirtual should be the combined rates for this reservation. Update moneyearn and moneyvirtual

      If it not available, simply abandon this reservation and mark the reservation as not arranged.

   b) If it is not an owner's reservation, check if it is a reservation requesting a certain room.

      i. If it is such a reservation, check the availability for the room requested. If it is available, assign the reservation to that room and update the occupancy time, last visit time, moneyearn and moneyvirtual with the combined rates for this reservation. Mark the reservation as arranged.

c) If the room required is not available or it is just a regular reservation, sort all the available room by moneyvirtual, and arrange this reservation in the room with least moneyvirtual, then update the last visit time=end time, update occupancy time, moneyearn and moneyvirtual with the combined rates for this reservation, and mark this reservation as arranged. If there are more than one room available having the least moneyvirtual, we arrange the reservation in the room with the least occupancy time.

vi) Update the room list and consider the next reservation.

We repeat steps iv) thoughout vi) for all reservations. We then can get a room list showing room number, occupancy time, owner's occupancy time, total income for each room. We can also get an update to the set of reservation data, including how each reservation is arranged.

For the conventional group, we use the assumed method traditional unit assignment in chapter 2. Simply arrange the reservations by room number. The process is as follows.

i) Form a room list of 100 rooms and pair room number with total occupancy time, owner’s occupancy time, moneyearn.

ii) Sort the reservation data by reservation date.

iii) Set the initial conditions: all room available for all date, occupancy time=0, last visit time=0, owner’s occupancy time=0, moneyearn=0.

iv) Form a room order sorted by room number.

v) When a reservation comes, check availability for all rooms from the start time till end time for this reservation, and pick up all available rooms. If there is no room available, abandon the reservation and mark the reservation as not arranged.

vi) Check if it is an owner’s reservation.

   a) If it is, check the availability for the room required. If it is available, arrange the reservation. Set availability to false for this room from the start date to end date, update the last visit time=end time, update occupancy time for this room, owner’s occupancy time and moneyearn, then set this reservation as arranged. Put this room number to the bottom of the room order.

   b) If it is not an owner’s reservation, check if it is a reservation requires a certain room.

   c) If it is such reservation, check the availability for the room required. If it is available, arrange the reservation to that room and update the occupancy time, last visit time, moneyearn with the combined rates for this reservation. Mark the reservation as arranged. Put this room number to the bottom of the room order.

d) If the room required is not available or it is just a regular reservation, sort all the available room by room order, and arrange this reservation in the first available room in the room order, then update the last visit time=end time, update occupancy time, moneyearn with the combined rates for this reservation, and mark this reservation as arranged. Put this room number to the bottom of the room order.

vii) Update the room list and consider the next reservation.

We can get our conventional results by repeating steps iv) to vi).

We build “reservation.h” to give definition to variables, use “main.cpp” to build the final room list and write files, use the file “reservation.cpp” to complete the arranging process shown above. The code can be found in Appendix 3.

We will demonstrate the results in the next chapter.

7 Analysis and discussion

By using the equalized method, we took 8618 reservations out of the 8653 original reservations. The average income was $29,338.73 while the standard deviation was $670.53. More detailed results are shown below.

Room Number	Occupancy time	Owner Occupancy time	Income	Difference
1	197	0	29403	64.627
2	199	12	29651	312.627
3	198	6	29502	163.627
4	199	1	29821	482.627
5	193	16	28757	-581.373
6	199	0	29971	632.627
7	192	13	28928	-410.373
8	193	15	28807	-531.373
9	201	9	30439	1100.627
10	199	9	29701	362.627
11	197	6	29723	384.627
12	202	11	30638	1299.627
13	198	7	29992	653.627
14	201	7	30069	730.627
15	194	6	28906	-432.373
16	197	7	29403	64.627
17	197	11	29403	64.627
18	184	23	27586	-1752.37
19	199	6	29651	312.627
20	197	12	29353	14.627
21	200	7	30290	951.627
22	195	13	29055	-283.373
23	198	12	29792	453.627
24	200	13	30340	1001.627
25	187	21	27913	-1425.37
26	201	9	30269	930.627
27	196	12	29254	-84.373
28	197	6	29523	184.627
29	199	3	30091	752.627

30	190	11	28480	-858.373
31	192	17	28978	-360.373
32	189	20	28281	-1057.37
33	194	17	28906	-432.373
34	196	1	29204	-134.373
35	187	21	27863	-1475.37
36	189	19	28211	-1127.37
37	198	9	29942	603.627
38	188	17	28332	-1006.37
39	187	20	28183	-1155.37
40	197	4	29523	184.627
41	199	11	30091	752.627
42	189	19	28161	-1177.37
43	198	13	29672	333.627
44	199	2	29651	312.627
45	192	16	28928	-410.373
46	198	10	29872	533.627
47	197	9	29403	64.627
48	197	9	29653	314.627
49	195	13	29055	-283.373
50	194	16	29396	57.627
51	197	11	29353	14.627
52	199	6	30141	802.627
53	199	8	29651	312.627
54	194	7	29396	57.627
55	197	10	29653	314.627
56	196	5	29374	35.627
57	196	4	29404	65.627
58	190	15	28800	-538.373
59	197	12	29353	14.627
60	185	23	27685	-1653.37
61	200	10	29970	631.627
62	197	4	29403	64.627
63	195	12	29105	-233.373
64	196	11	29254	-84.373
65	195	10	29475	136.627
66	198	5	29502	163.627
67	197	10	29523	184.627
68	196	9	29424	85.627
69	192	18	28778	-560.373

70	198	2	29872	533.627
71	200	5	30290	951.627
72	193	14	29197	-141.373
73	197	7	29523	184.627
74	197	10	29403	64.627
75	182	23	27458	-1880.37
76	195	14	29375	36.627
77	200	7	29800	461.627
78	186	20	28134	-1204.37
79	192	15	28728	-610.373
80	198	0	29822	483.627
81	199	6	30141	802.627
82	194	13	29446	107.627
83	198	6	29872	533.627
84	184	22	27786	-1552.37
85	194	14	28956	-382.373
86	199	3	30021	682.627
87	194	13	29076	-262.373
88	198	7	29552	213.627
89	198	8	29622	283.627
90	195	6	29395	56.627
91	195	1	29545	206.627
92	200	4	30290	951.627
93	201	10	29949	610.627
94	193	14	28877	-461.373
95	200	4	30170	831.627
96	194	12	29226	-112.373
97	198	7	29502	163.627
98	193	15	28757	-581.373
99	195	12	29425	86.627
100	197	9	29403	64.627
		Average	29338.73	
		Stdev.s	670.5314	

Table 5 income for each room, average income and standard deviation using the method with equalization

By using the method without equalization, we took 8607 reservations out of the 8653 original reservations. The average income turned out to be $29,250.82 while the standard deviation was 1,305.33. More detailed results are shown below:

Room Number	Occupancy time	Owner Occupancy time	Income	difference
1	214	0	31936	2685.18
2	213	12	32177	2926.18
3	171	6	25479	-3771.82
4	200	1	30170	919.18
5	193	16	28857	-393.82
6	195	0	29395	144.18
7	192	13	28728	-522.82
8	202	15	30468	1217.18
9	184	9	27856	-1394.82
10	204	9	30396	1145.18
11	197	6	29523	272.18
12	177	11	26743	-2507.82
13	196	7	29204	-46.82
14	202	7	30468	1217.18
15	198	6	29602	351.18
16	198	7	29992	741.18
17	202	11	30588	1337.18
18	189	23	28581	-669.82
19	185	6	27565	-1685.82
20	193	12	28927	-323.82
21	195	7	29225	-25.82
22	192	13	28728	-522.82
23	194	12	28956	-294.82
24	195	13	29155	-95.82
25	200	21	29970	719.18
26	193	9	29127	-123.82
27	189	12	28531	-719.82
28	198	6	29992	741.18

29	200	3	30050	799.18
30	199	11	30071	820.18
31	182	17	27118	−2132.82
32	186	20	28154	−1096.82
33	186	17	27714	−1536.82
34	201	1	30439	1188.18
35	189	21	28601	−649.82
36	195	19	29055	−195.82
37	183	9	27437	−1813.82
38	204	17	30936	1685.18
39	189	20	28481	−769.82
40	180	4	26990	−2260.82
41	194	11	29276	25.18
42	201	19	30069	818.18
43	194	13	29226	−24.82
44	198	2	30042	791.18
45	181	16	26969	−2281.82
46	210	10	31660	2409.18
47	214	9	32086	2835.18
48	210	9	31580	2329.18
49	200	13	29920	669.18
50	200	16	30290	1039.18
51	188	11	28062	−1188.82
52	197	6	29353	102.18
53	205	8	30545	1294.18
54	204	7	30766	1515.18
55	195	10	29105	−145.82
56	193	5	29197	−53.82
57	201	4	30119	868.18
58	188	15	28012	−1238.82
59	197	12	29553	302.18
60	194	23	29026	−224.82
61	203	10	30617	1366.18
62	198	4	29922	671.18
63	204	12	30516	1265.18
64	195	11	29055	−195.82
65	191	10	28579	−671.82
66	194	5	29146	−104.82
67	180	10	26870	−2380.82
68	194	9	29276	25.18

69	187	18	28183	-1067.82
70	188	2	28062	-1188.82
71	206	5	30744	1493.18
72	191	14	28459	-791.82
73	207	7	31213	1962.18
74	194	10	29226	-24.82
75	200	23	29920	669.18
76	188	14	28012	-1238.82
77	208	7	31162	1911.18
78	200	20	30120	869.18
79	193	15	28807	-443.82
80	190	0	28310	-940.82
81	187	6	28353	-897.82
82	193	13	28927	-323.82
83	195	6	29105	-145.82
84	187	22	28403	-847.82
85	176	14	26544	-2706.82
86	202	3	30148	897.18
87	199	13	29651	400.18
88	191	7	28459	-791.82
89	179	8	26671	-2579.82
90	208	6	31042	1791.18
91	203	1	30417	1166.18
92	180	4	26990	-2260.82
93	195	10	29105	-145.82
94	185	14	27935	-1315.82
95	189	4	28331	-919.82
96	204	12	30566	1315.18
97	198	7	29552	301.18
98	198	15	29822	571.18
99	193	12	28977	-273.82
100	196	9	29644	393.18
		Average:	**29250.82**	
		Stdev.s	**1305.329**	

Table 6 income for each room, average income and standard deviation using the method without equalization

From the results we can see that the difference in standard deviation between the two methods is huge. The equalization method worked very well.

Let’s take a closer look at the results using the equalization method. Room 75 earned the least for the owner has booked for him/herself for 22 nights. Remember there will be punishment for those who book their own rooms over 10 nights. It is pretty fair that this owner earned the least.

Reservation Date	Check-in Date	Check-out Date	Is House Owner	Whether Arranged	Room Number
1--3	1--5	1--7	0	1	75
1--21	1--24	1--25	0	1	75
1--31	2--4	2--11	1	1	75
2--4	2--12	2--13	0	1	75
2--7	2--11	2--12	0	1	75
2--9	2--13	2--14	0	1	75
2--12	2--20	2--21	0	1	75
2--12	2--21	2--24	1	1	75
2--14	2--14	2--18	0	1	75
2--23	2--28	3--3	0	1	75
3--1	3--3	3--6	0	1	75
3--7	3--15	3--16	0	1	75
3--9	3--18	3--20	0	1	75
3--14	3--21	3--23	0	1	75
3--15	3--17	3--18	1	1	75
3--20	3--24	3--26	0	1	75
3--24	3--27	3--29	0	1	75
3--29	4--6	4--7	0	1	75
3--31	4--7	4--8	0	1	75

4--3	4--11	4--13	0	1	75
4--7	4--14	4--15	0	1	75
4--11	4--16	4--18	0	1	75
4--15	4--24	4--26	0	1	75
4--19	4--19	4--20	0	1	75
4--21	4--21	4--22	0	1	75
4--24	4--27	4--28	0	1	75
4--26	4--28	4--30	0	1	75
4--29	5--1	5--4	0	1	75
5--4	5--6	5--9	0	1	75
5--8	5--10	5--12	0	1	75
5--10	5--15	5--16	1	1	75
5--11	5--13	5--14	0	1	75
5--13	5--17	5--20	0	1	75
5--17	5--25	5--26	0	1	75
5--18	5--21	5--23	0	1	75
5--22	5--26	5--28	0	1	75
5--25	5--28	6--1	0	1	75
6--2	6--7	6--11	0	1	75
6--14	6--17	6--20	0	1	75
6--17	6--23	6--26	0	1	75
6--17	6--21	6--23	0	1	75
6--20	6--26	6--29	0	1	75
6--24	6--29	7--3	0	1	75
6--29	7--8	7--12	0	1	75
7--5	7--5	7--7	0	1	75
7--8	7--13	7--15	0	1	75
7--11	7--18	7--22	0	1	75
7--17	7--17	7--18	0	1	75
7--18	7--25	7--28	0	1	75
7--22	7--28	8--1	1	1	75
7--23	7--23	7--24	0	1	75
7--23	8--1	8--8	0	1	75
8--2	8--10	8--12	0	1	75
8--5	8--8	8--10	0	1	75
8--8	8--15	8--19	0	1	75
8--13	8--19	8--21	0	1	75
8--15	8--22	8--25	0	1	75
8--22	8--31	9--3	0	1	75
8--23	8--25	8--27	0	1	75

8--25	8--27	8--31	1	1	75
8--26	9--3	9--4	0	1	75
8--27	9--5	9--10	0	1	75
9--2	9--4	9--5	0	1	75
9--4	9--11	9--12	0	1	75
9--4	9--12	9--14	0	1	75
9--7	9--16	9--17	0	1	75
9--8	9--10	9--11	0	1	75
9--9	9--14	9--16	1	1	75
9--10	9--18	9--21	0	1	75
9--14	9--17	9--18	0	1	75
9--16	9--25	9--27	0	1	75
9--17	9--21	9--22	1	1	75
9--18	9--22	9--24	0	1	75
9--21	9--29	10--1	0	1	75
9--24	9--24	9--25	0	1	75
9--25	9--27	9--29	0	1	75
9--30	10--1	10--3	0	1	75
10--6	10--11	10--13	0	1	75
10--13	10--15	10--16	0	1	75
10--15	10--21	10--24	0	1	75
10--23	10--30	11--1	0	1	75
10--29	11--3	11--10	0	1	75
11--14	11--23	11--24	0	1	75
11--17	11--19	11--21	0	1	75
11--21	11--24	11--26	0	1	75
11--25	11--29	11--30	0	1	75
11--28	11--28	11--29	0	1	75
11--30	12--3	12--5	0	1	75
12--5	12--12	12--15	0	1	75
12--28	1--2	1--4	0	1	75
12--28	12--28	12--30	0	1	75
12--28	12--31	1--2	0	1	75

Table 7 arrangement detail for room 75 using the method with equalization method

Let's take another look at room 12, who led with the income of $30,638.

Even though this owner is 1 night beyond the “10-night rule”, his room was requested for a whole week in August which gave him/her the huge lead.

Reservation Date	Check-in Date	Check-out Date	Is House Owner	Room Number	Is Arranged	Arranged Number
1--21	1--22	1--23	1	12	0	12
3--30	3--31	4--2	1	12	0	12
4--30	5--6	5--8	0	-1	0	12
6--15	6--15	6--20	1	12	0	12
7--3	7--10	7--12	0	-1	0	12
8--12	8--12	8--19	0	-1	0	12
8--27	8--30	9--1	1	12	0	12
9--18	9--18	9--19	1	12	0	12

Table 8 arrangement detail for room 12 using the method with equalization method

8 Reservation assignment with an added human decision maker

As it is mentioned in chapter 1 and chapter 2, human involvement is crucial in such circumstances. In this chapter, we will try to modify our program to give the management team some freedom to adjust the rules of running the hotel.

In chapter 6, we set the rules running the hotel as follows:

a) Owners can use their room for 10 nights per year without harming their income. Those who use over 10 nights will lose 100% of the rate of the date they stayed for each additional night.

b) When a guest requires a certain room, the management team will first try to fulfill the guest’s requirement and assign the reservation to the room asked for. If that room is not available, they then treat it as a normal reservation and assign it to any available room.

But in some cases, owners are welcome if the condo hotel is not as busy even if they went beyond the 10-nights limit. Or management team should charge less punishment for the nights owners over stayed to keep the owners happy and keep them in the leasing program. More commonly, a certain room might be requested a lot during summer session and makes it too difficult for others to catch up. In such circumstances, the management team might ignore the requirement from the guests and balance the income for different owners.

To achieve such function, we make some change to the program and introduce a configuration file to set the rules of running the hotel. The configuration file contains four basic variables.

Owner_Max_Occupancy_Days decides how many nights the management team would like the owner to use without harming their income. This variable can be given an integer from 0 to 365. If the number is given 0, that means every night that the owner stays would harm the income. If it is given 365, that means owner can use the room for 365 days per year without harming the income, which is basically, no punishment for over staying.

Owner_Occupancy_Penalty is a variable that decides how much it influence the income if an owner is over using his or her room. This variable can be given a number from 0 to 1.0. If the number is 0, it means that the owner is not punished for over using the room, while if this number is 1.0, it means that the owner will lose 100% of the rate of the date they stayed for each additional night as it is mentioned in chapter 6.

Owner_Panalty_Flag decides if the management team would like to punish owners for over staying. This variable can be given 0 or 1. 0 means the owner is not punished for over staying while 1 means the opposite.

The final variable is called *Considering_Custom_Room_Number_Flag*. This variable decides if the management team would consider guests' requests to assign the reservations to the rooms requested. This variable can be given 0 or 1. 0 means the management team do not consider guests' request and treat those reservations as normal reservations, while 1 means the managers would assign the reservations to the rooms requested if they are available.

By modifying this configuration file before running the project, the management team is given freedom to change the rules that they run the hotel by. And different rules would bring different optimization results. We will demonstrate 5 groups of results based on different configuration variables.

The code can be found in Appendix 4 and the configuration file can be found in Appendix 5.

9 Optimization results based on different configuration variables

Group 1

The results shown in chapter 7 were based on the configuration variables below.

Owner_Max_Occupancy_Days 10

Owner_Occupancy_Penalty 1.0

Owner_Penalty_Flag 1

Considering_Custom_Room_Number_Flag 1

Group 2

Owner_Max_Occupancy_Days 10

Owner_Occupancy_Penalty 1.0

Owner_Penalty_Flag 0

Considering_Custom_Room_Number_Flag 1

This set of variables means that the managers would not penalize owners for using their rooms. And the result is shown below.

Room Number	Occupancy time	Owner Occupancy time	Income	Difference
1	198	0	30042	709.23
2	195	12	29055	-277.77
3	198	6	29502	169.23
4	196	1	29424	91.23

5	196	16	29254	-78.77
6	197	0	29353	20.23
7	197	13	29843	510.23
8	196	15	29574	241.23
9	195	9	29425	92.23
10	197	9	29893	560.23
11	195	6	29055	-277.77
12	197	11	29573	240.23
13	195	7	29055	-277.77
14	196	7	29574	241.23
15	195	6	29055	-277.77
16	195	7	29375	42.23
17	195	11	29225	-107.77
18	194	23	29076	-256.77
19	199	6	29651	318.23
20	197	12	29793	460.23
21	195	7	29055	-277.77
22	195	13	29055	-277.77
23	195	12	29375	42.23
24	195	13	29055	-277.77
25	195	21	29225	-107.77
26	194	9	29146	-186.77
27	194	12	29026	-306.77
28	194	6	29076	-256.77
29	198	3	29502	169.23
30	195	11	29055	-277.77
31	196	17	29524	191.23
32	198	20	29872	539.23
33	198	17	29502	169.23
34	196	1	29204	-128.77
35	194	21	29196	-136.77
36	195	19	29175	-157.77
37	196	9	29254	-78.77
38	195	17	29055	-277.77
39	195	20	29105	-227.77
40	196	4	29204	-128.77
41	195	11	29225	-107.77
42	194	19	29206	-126.77
43	196	13	29204	-128.77
44	193	2	29127	-205.77

45	194	16	29026	-306.77
46	192	10	28978	-354.77
47	198	9	29502	169.23
48	197	9	29773	440.23
49	196	13	29254	-78.77
50	198	16	29502	169.23
51	201	11	30489	1156.23
52	193	6	29127	-205.77
53	196	8	29204	-128.77
54	196	7	29204	-128.77
55	193	10	29097	-235.77
56	194	5	29156	-176.77
57	195	4	29055	-277.77
58	195	15	29545	212.23
59	195	12	29055	-277.77
60	197	23	29353	20.23
61	195	10	29225	-107.77
62	192	4	28608	-724.77
63	201	12	30489	1156.23
64	195	11	29055	-277.77
65	196	10	29204	-128.77
66	194	5	29076	-256.77
67	195	10	29055	-277.77
68	193	9	29177	-155.77
69	195	18	29495	162.23
70	194	2	29346	13.23
71	196	5	29644	311.23
72	195	14	29175	-157.77
73	194	7	29326	-6.77
74	194	10	29106	-226.77
75	193	23	29247	-85.77
76	192	14	29028	-304.77
77	196	7	29694	361.23
78	197	20	29353	20.23
79	198	15	29992	659.23
80	196	0	29204	-128.77
81	196	6	29254	-78.77
82	195	13	29495	162.23
83	194	6	29076	-256.77
84	194	22	29076	-256.77

85	194	14	29446	113.23
86	195	3	29225	-107.77
87	197	13	29723	390.23
88	195	7	29105	-227.77
89	195	8	29475	142.23
90	196	6	29694	361.23
91	197	1	29723	390.23
92	195	4	29275	-57.77
93	197	10	29723	390.23
94	195	14	29545	212.23
95	194	4	29226	-106.77
96	196	12	29324	-8.77
97	196	7	29374	41.23
98	194	15	29276	-56.77
99	192	12	29148	-184.77
100	195	9	29055	-277.77
		Average	29332.77	
		Stdev.s	308.4995	

Table 9 Arrangement result with 2nd group configuration file

8616 out of the 8653 reservations are taken. And the average income of all owners was $29,332.77 while the standard deviation was $308.4995.

We found out the standard deviation decreased because the moneyvritual and moneyearn are the same for each owner under such rule. So the difference between each owner decreased. But under such rule one might still earn more even an owner uses the room very often. Take room number 62 as an example, even though the owner just used for 4 nights, that particular room did the worst.

Group 3

Owner_Max_Occupancy_Days 10

Owner_Occupancy_Penalty 1.0

Owner_Penalty_Flag 1

Considering_Custom_Room_Number_Flag 0

This set of configuration variables means the management team would not consider the guest' s request when arranging the reservations. The arrangement result is shown below.

Room Number	Occupancy time	Owner Occupancy time	Income	Difference
1	199	0	29701	393.56
2	195	12	29175	-132.44
3	200	6	30170	862.56
4	200	1	30290	982.56
5	192	16	28658	-649.44
6	199	0	29651	343.56
7	193	13	29077	-230.44
8	193	15	29077	-230.44
9	197	9	29353	45.56
10	199	9	29821	513.56
11	197	6	29723	415.56
12	197	11	29353	45.56
13	197	7	29573	265.56
14	197	7	29353	45.56
15	197	6	29593	285.56

16	198	7	29942	634.56
17	198	11	29792	484.56
18	184	23	27536	-1771.44
19	199	6	29651	343.56
20	196	12	29204	-103.44
21	198	7	29872	564.56
22	193	13	28927	-380.44
23	195	12	29055	-252.44
24	195	13	29055	-252.44
25	188	21	28382	-925.44
26	197	9	29353	45.56
27	195	12	29155	-152.44
28	197	6	29353	45.56
29	196	3	29404	96.56
30	195	11	29225	-82.44
31	192	17	28608	-699.44
32	187	20	28033	-1274.44
33	192	17	28608	-699.44
34	197	1	29673	365.56
35	187	21	28233	-1074.44
36	189	19	28481	-826.44
37	199	9	29651	343.56
38	187	17	28353	-954.44
39	189	20	28161	-1146.44
40	198	4	29502	194.56
41	200	11	29800	492.56
42	190	19	28510	-797.44
43	199	13	29651	343.56
44	199	2	30021	713.56
45	191	16	28629	-678.44
46	201	10	30439	1131.56
47	198	9	29602	294.56
48	195	9	29475	167.56
49	194	13	29226	-81.44
50	192	16	28978	-329.44
51	196	11	29204	-103.44
52	197	6	29523	215.56
53	197	8	29403	95.56
54	195	7	29425	117.56
55	197	10	29473	165.56

56	198	5	29622	314.56
57	198	4	29922	614.56
58	193	15	28807	-500.44
59	195	12	29055	-252.44
60	183	23	27437	-1870.44
61	198	10	29502	194.56
62	200	4	29800	492.56
63	196	12	29574	266.56
64	197	11	29353	45.56
65	200	10	30290	982.56
66	195	5	29595	287.56
67	198	10	29672	364.56
68	195	9	29475	167.56
69	190	18	28750	-557.44
70	198	2	29552	244.56
71	197	5	29793	485.56
72	194	14	28906	-401.44
73	197	7	29403	95.56
74	199	10	30071	763.56
75	183	23	27557	-1750.44
76	193	14	28927	-380.44
77	203	7	30787	1479.56
78	188	20	28382	-925.44
79	194	15	28956	-351.44
80	199	0	30141	833.56
81	196	6	29574	266.56
82	195	13	29345	37.56
83	197	6	29523	215.56
84	182	22	27608	-1699.44
85	196	14	29494	186.56
86	197	3	29353	45.56
87	195	13	29375	67.56
88	197	7	29523	215.56
89	194	8	29396	88.56
90	196	6	29694	386.56
91	195	1	29495	187.56
92	197	4	29403	95.56
93	196	10	29424	116.56
94	199	14	30191	883.56
95	198	4	29502	194.56

96	195	12	29305	−2.44
97	199	7	30141	833.56
98	193	15	28757	−550.44
99	195	12	29105	−202.44
100	199	9	30071	763.56
		Average	29307.44	
		Stdev.s	628.3798	

Table 10 Arrangement result with 3rd group configuration file

We then try to do some modification to the rule of penalizing owners for over using. Introducing the next configuration file:

Group 4

Owner_Max_Occupancy_Days 10

Owner_Occupancy_Penalty 0.85

Owner_Penalty_Flag 1

Considering_Custom_Room_Number_Flag 1

Comparing to the first group, this set of variable charges 85% of the rate for each night an owner reserved beyond 10 free nights. Decreasing this percentage would satisfy the owners and keep them in the condo program.

The result is shown below.

Room Number	Occupancy time	Owner Occupancy time	Income	Difference
1	197	0	29403	88.11
2	196	12	29204	-110.89
3	199	6	30141	826.11
4	197	1	29353	38.11
5	192	16	28608	-706.89
6	196	0	29574	259.11
7	195	13	29225	-89.89
8	195	15	29105	-209.89
9	197	9	29403	88.11
10	195	9	29495	180.11
11	199	6	29651	336.11
12	196	11	29374	59.11
13	195	7	29425	110.11
14	198	7	29502	187.11
15	192	6	28928	-386.89
16	198	7	29552	237.11
17	198	11	30042	727.11
18	186	23	27714	-1600.89
19	198	6	29502	187.11
20	195	12	29225	-89.89
21	197	7	29453	138.11
22	195	13	29175	-139.89
23	194	12	29276	-38.89
24	194	13	29026	-288.89
25	187	21	28233	-1081.89
26	199	9	30091	776.11
27	197	12	29553	238.11
28	197	6	29353	38.11
29	197	3	29693	378.11
30	196	11	29324	9.11
31	191	17	28509	-805.89
32	188	20	28382	-932.89
33	196	17	29204	-110.89
34	197	1	29353	38.11
35	187	21	28033	-1281.89
36	189	19	28211	-1103.89
37	197	9	29353	38.11

38	191	17	28629	-685.89
39	188	20	28332	-982.89
40	198	4	29942	627.11
41	196	11	29374	59.11
42	192	19	29098	-216.89
43	196	13	29204	-110.89
44	197	2	29353	38.11
45	192	16	28778	-536.89
46	196	10	29624	309.11
47	197	9	29673	358.11
48	197	9	29403	88.11
49	194	13	28956	-358.89
50	192	16	28608	-706.89
51	197	11	29353	38.11
52	198	6	29552	237.11
53	197	8	29473	158.11
54	194	7	29396	81.11
55	197	10	29723	408.11
56	197	5	29473	158.11
57	199	4	29651	336.11
58	195	15	29225	-89.89
59	195	12	29105	-209.89
60	188	23	28232	-1082.89
61	198	10	29822	507.11
62	199	4	30091	776.11
63	198	12	29992	677.11
64	196	11	29204	-110.89
65	195	10	29425	110.11
66	198	5	29502	187.11
67	197	10	29523	208.11
68	196	9	29524	209.11
69	192	18	28778	-536.89
70	197	2	29353	38.11
71	198	5	29552	237.11
72	194	14	29076	-238.89
73	197	7	29523	208.11
74	195	10	29495	180.11
75	182	23	27658	-1656.89
76	199	14	30191	876.11
77	198	7	29502	187.11

78	189	20	28161	-1153.89
79	196	15	29694	379.11
80	198	0	29502	187.11
81	198	6	29822	507.11
82	194	13	29126	-188.89
83	197	6	29673	358.11
84	189	22	28531	-783.89
85	194	14	29276	-38.89
86	196	3	29374	59.11
87	193	13	29097	-217.89
88	198	7	29502	187.11
89	197	8	29793	478.11
90	197	6	29523	208.11
91	198	1	29872	557.11
92	199	4	30071	756.11
93	199	10	30021	706.11
94	192	14	28808	-506.89
95	199	4	30021	706.11
96	201	12	30489	1174.11
97	193	7	29297	-17.89
98	196	15	29374	59.11
99	196	12	29574	259.11
100	198	9	29872	557.11
		Average	29314.89	
		Stdev.s	534.7979	

Table 11 Arrangement result with 4th group configuration file

Take a closer look at room number 75. This owner did the worst as well with the first group configuration variables. But he made $200 more comparing to group 1. So this set of variables would make this owner happier.

We can also find that the one who did the best is room number 96 comparing to room number 12 by group 1. This is caused by the change of moneyvirtual when decreasing the penalty percentage, and lead to the change to the rotation of the rooms. We pick out the reservation details for room number 96 and find out that this room is also greatly preferred by the guests. This room is requested by guests 7 times for 18 nights, and 7 of them are among May and July.

Reservation Date	Check-in Date	Check-out Date	Is House Owner	Room Number	Is Arranged	Arranged Number
1--11	1--13	1--15	0	-1	0	96
2--14	2--19	2--22	1	96	0	96
3--27	3--29	4--1	1	96	0	96
3--30	4--7	4--10	0	-1	0	96
5--18	5--20	5--23	1	96	0	96
4--29	5--5	5--7	0	-1	0	96
5--28	5--28	5--29	0	-1	0	96
5--1	5--1	5--5	0	-1	0	96
7--11	7--11	7--13	0	-1	0	96
9--24	10--3	10--4	1	96	0	96
12--22	12--22	12--24	1	96	0	96
11--28	12--3	12--7	0	-1	0	96

Table 12 Reservation detail for room 96 with 4th group configuration file

Reservation Date	Check-in Date	Check-out Date	Is House Owner	Whether Arranged	Room Number
1--3	1--9	1--11	0	1	96
1--7	1--13	1--15	0	1	96
1--11	1--17	1--20	0	1	96
1--18	1--27	1--28	0	1	96
1--20	1--23	1--24	0	1	96
1--22	1--24	1--25	0	1	96
1--25	1--31	2--4	0	1	96
2--2	2--8	2--10	0	1	96
2--6	2--11	2--14	0	1	96

2--13	2--14	2--16	0	1	96
2--14	2--19	2--22	1	1	96
2--17	2--23	2--24	0	1	96
2--19	2--27	3--2	0	1	96
2--25	3--4	3--6	0	1	96
3--2	3--8	3--11	0	1	96
3--22	3--26	3--28	0	1	96
3--22	3--22	3--24	0	1	96
3--22	3--25	3--26	0	1	96
3--23	3--24	3--25	0	1	96
3--23	4--1	4--2	0	1	96
3--26	4--2	4--4	0	1	96
3--27	3--29	4--1	1	1	96
3--30	4--7	4--10	0	1	96
3--30	4--5	4--7	0	1	96
4--12	4--19	4--22	0	1	96
4--19	4--23	4--24	0	1	96
4--21	4--26	4--29	0	1	96
4--27	5--6	5--8	0	1	96
5--1	5--1	5--3	0	1	96
5--4	5--8	5--12	0	1	96
5--9	5--12	5--15	0	1	96
5--14	5--16	5--19	0	1	96
5--18	5--20	5--23	1	1	96
5--19	5--27	5--28	0	1	96
5--20	5--23	5--26	0	1	96
5--25	5--30	5--31	0	1	96
5--26	6--1	6--5	0	1	96
5--28	5--28	5--29	0	1	96
6--3	6--5	6--11	0	1	96
6--13	6--16	6--19	0	1	96
6--18	6--25	6--27	0	1	96
6--20	6--20	6--24	0	1	96
6--26	7--4	7--7	0	1	96
6--29	6--29	7--3	0	1	96
7--6	7--8	7--10	0	1	96
7--9	7--18	7--23	0	1	96
7--11	7--11	7--13	0	1	96
7--16	7--24	7--27	0	1	96
7--23	7--23	7--24	0	1	96

7--24	7--28	7--31	0	1	96
7--29	7--31	8--2	0	1	96
7--31	8--3	8--6	0	1	96
8--4	8--10	8--12	0	1	96
8--8	8--8	8--10	0	1	96
8--9	8--16	8--19	0	1	96
8--13	8--19	8--26	0	1	96
8--23	8--26	8--29	0	1	96
8--27	9--2	9--4	0	1	96
8--29	8--29	9--2	0	1	96
9--2	9--5	9--6	0	1	96
9--4	9--7	9--10	0	1	96
9--8	9--11	9--14	0	1	96
9--13	9--20	9--22	0	1	96
9--15	9--15	9--17	0	1	96
9--18	9--26	9--27	0	1	96
9--19	9--22	9--25	0	1	96
9--23	9--28	10--1	0	1	96
9--24	10--3	10--4	1	1	96
9--30	10--4	10--6	0	1	96
10--6	10--11	10--13	0	1	96
10--12	10--19	10--23	0	1	96
10--23	10--30	10--31	0	1	96
10--26	10--31	11--3	0	1	96
11--2	11--10	11--11	0	1	96
11--4	11--7	11--9	0	1	96
11--9	11--18	11--22	0	1	96
11--19	11--22	11--23	0	1	96
11--21	11--23	11--25	0	1	96
11--25	11--25	12--2	0	1	96
11--28	12--3	12--7	0	1	96
12--12	12--15	12--18	0	1	96
12--22	12--22	12--24	1	1	96
12--31	1--1	1--7	0	1	96

Table 13 Arrangement summary of room 96 with 4th group configuration file

Group 5

Owner_Max_Occupancy_Days 15

Owner_Occupancy_Penalty 1.0

Owner_Penalty_Flag 1

Considering_Custom_Room_Number_Flag 1

In this case, the total nights an owner can use without harming the total income increased to 15 comparing to group 1. This would also make the owners happy.

The result is shown below.

Room Number	Occupancy time	Owner Occupancy time	Income	Difference
1	197	0	29723	373.84
2	196	12	29204	-145.16
3	196	6	29574	224.84
4	195	1	29255	-94.16
5	199	16	30021	671.84
6	187	0	27863	-1486.16
7	196	13	29374	24.84
8	193	15	29247	-102.16
9	196	9	29574	224.84
10	198	9	30042	692.84
11	199	6	29651	301.84
12	195	11	29155	-194.16
13	196	7	29204	-145.16
14	199	7	29651	301.84
15	193	6	28757	-592.16
16	196	7	29204	-145.16
17	195	11	29175	-174.16
18	187	23	28033	-1316.16
19	198	6	29502	152.84

20	196	12	29204	−145.16
21	196	7	29424	74.84
22	197	13	29403	53.84
23	195	12	29225	−124.16
24	198	13	29872	522.84
25	192	21	28978	−371.16
26	199	9	29971	621.84
27	196	12	29404	54.84
28	196	6	29204	−145.16
29	198	3	30042	692.84
30	196	11	29694	344.84
31	191	17	28949	−400.16
32	191	20	28899	−450.16
33	197	17	29353	3.84
34	197	1	29353	3.84
35	189	21	28331	−1018.16
36	196	19	29204	−145.16
37	197	9	29403	53.84
38	193	17	29177	−172.16
39	191	20	28459	−890.16
40	196	4	29524	174.84
41	197	11	29403	53.84
42	192	19	28608	−741.16
43	195	13	29225	−124.16
44	197	2	29843	493.84
45	195	16	29055	−294.16
46	195	10	29475	125.84
47	196	9	29374	24.84
48	196	9	29624	274.84
49	195	13	29375	25.84
50	195	16	29055	−294.16
51	199	11	30021	671.84
52	194	6	29346	−3.16
53	196	8	29374	24.84
54	197	7	29403	53.84
55	195	10	29225	−124.16
56	196	5	29524	174.84
57	197	4	29353	3.84
58	194	15	29276	−73.16
59	195	12	29255	−94.16

60	189	23	28211	-1138.16
61	196	10	29254	-95.16
62	197	4	29403	53.84
63	198	12	29942	592.84
64	196	11	29204	-145.16
65	196	10	29204	-145.16
66	194	5	29126	-223.16
67	198	10	29872	522.84
68	196	9	29254	-95.16
69	194	18	28906	-443.16
70	202	2	30638	1288.84
71	197	5	29403	53.84
72	196	14	29204	-145.16
73	198	7	29992	642.84
74	197	10	29523	173.84
75	187	23	28033	-1316.16
76	195	14	29225	-124.16
77	197	7	29723	373.84
78	189	20	28401	-948.16
79	199	15	29651	301.84
80	197	0	29693	343.84
81	193	6	29297	-52.16
82	196	13	29374	24.84
83	198	6	29672	322.84
84	187	22	28233	-1116.16
85	197	14	29793	443.84
86	196	3	29374	24.84
87	195	13	29175	-174.16
88	202	7	30638	1288.84
89	196	8	29324	-25.16
90	195	6	29345	-4.16
91	197	1	29723	373.84
92	196	4	29524	174.84
93	198	10	29672	322.84
94	199	14	29821	471.84
95	196	4	29524	174.84
96	195	12	29225	-124.16
97	194	7	29226	-123.16
98	197	15	29673	323.84
99	196	12	29204	-145.16

100	199	9	30141	791.84
		Average	29349.16	
		Stdev.s	482.6593	

Table 12 Arrangement result with 4th group configuration file

By introducing the configuration file, we give the management team some freedom while running this program and applying the equalization method. Adjusting the configuration variables cleverly would help satisfy both the owners and guests.

10 Conclusion and future recommendation

In conclusion, both the method and the program worked well for this case. By the set of data provided in chapter 6, even though the two groups showed similar average number, which means they both took almost all reservations provided; the difference between standard deviation is significant. The equalization method helped a lot in balancing the total income for all owners. We can foresee that if condo hotels use software using this kind of method, the management should be much easier. Owners will not be complaining about getting unfair income.

And in chapter 9, we introduced human involvement. Unlike the human involvement of manually putting more reservations into those who fell behind, we provided variables to give the management team freedom to rum the condo hotel with different rules but still stick to the method with equalization. The result is positive. Comparing to the result using the method without equalization, the standard deviation decreased for all 5 groups, which means a more balanced arrangement result is formed by using different rules with the equalized method.

This technique can also be applied into many other industrial fields. For example, a manufacturing company holds several machines that can process the same material into a final product in the same amount of time. There are orders for such product coming in all the time. We can use this method and this program to arrange the work flow to balance the wear and tear for all machines and try to maximize the income.

For further improvement of this program, I am looking forward to a real set of reservation data. With a real set of data, we can study how the condo is handling with the reservations currently and work out the average income and standard deviation. Then we apply this set of data into the program and analyze the result. A real set of data is more realistic and convincible. Secondly, as we mentioned in chapter 3, we can introduce more reservation categories based on how the condo is running.

Besides, in chapter 9, we give the management team some freedom running the program. We can also provide a real-time modifier and allow the management team changes the variables and run the condo with different rules in different seasons. For example, during summer, owners are given less nights without penalty because the summer session is much busier. In the winter, owners would get more freedom to come and stay in their room for there is less guests coming and the condo hotel is almost empty.

In the future, this program should provide a GUI (graphical user interface) that allows manager to modify the variables in a much easier way instead of reading and changing the configuration file.

References

[1] Michael Corkery, Condo Hotels: The Latest Twist In Buying a Vacation Residence[N] The Wall Street Journal Online. February 28, 2006

[2] Chin, Calvin. "Guide to Condo Hotels - Part 2: Financial Analysis"[J] . Halogen Guides. Retrieved 2007-11-13.

[3] John E. Czarnecki, Building boom in hotel-condo combinations [J]. Architectural record. 2005, 193(11)

[4] Mary Scoviak, Condo-hotels: boom or bust? [J] Hotels. 2006, 40(9)

[5] Peterson, Tara (2007-05-23). "Condo Hotels: A Conceptual Overview".[J] Premium Condo Hotels. Retrieved 2008-03-20

[6] http://theflandershotel.com/history/

[7] http://www.guesttracker.com/products.htm

[8] http://www.ssctech.com/timeshareware/Solutions/OwnerMemberServices.aspx

[9] http://sales.resortdata.com/Vacationrental.htm#Equalize

[10] Cormen, Leiserson, Rivest. Introduction to Algorithms1990 [M], Chapter 17 "Greedy Algorithms" p. 329.

[11] G. Bendall and F. Margot, Greedy Type Resistance of Combinatorial Problems,[J] Discrete Optimization 3 (2006), 288–298.

Appendix 1

```
//  main.cpp
//  Reservation
//
//  Created by Video on 7/7/15.
//  Copyright (c) 2015 Video. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include "time.h"

using namespace std;
//percentage of rental period
static int summerRate[] = {5,30,30,15,10,5,5};
static int winterRate[] = {35,35,20,7,1,1,1};
//Is summer or not
static const bool summerFlag[] =
{false,false,false,false,false,true,true,
                             true,false,false,false,false};
//average rental period for each month
static int rentDays[] = {13,13,13,13,23,23,23,28,23,13,13,13};
//static int rentDays[] = {13,13,13,13,13,13,13,13,13,13,13,13};
static int daysInMonth[] =
{31,28,31,30,31,30,31,31,30,31,30,31};
//number of rooms
static const int roomNumber = 100;
static const int monthNumber = 12;
//percentage of owner's reservations.
static const int owner_percent = 5;

int getPeriod(int *rate, int randNumber);

struct reservationData {
    string reservationDate;
    string startDate;
    string endDate;
    bool ownerFlag;
    int ownerRoom;
};

bool writeList(string file_name, vector<reservationData> data);

int main(int argc, const char * argv[])
```

```
{
    vector<reservationData> reservationList;

    srand((unsigned int)time(NULL));
    for(int i = 0;i<monthNumber; i++){
        int offset = rand()%5-3;
        rentDays[i] = rentDays[i] + offset;
    }

    for(int i = 1;i<7;i++) {
        summerRate[i] = summerRate[i] + summerRate[i-1];
        winterRate[i] = winterRate[i] + winterRate[i-1];
    }

    for(int i = 0; i<monthNumber; i++) {
        int *rate;
        if(summerFlag[i]){
            rate = summerRate;
        }
        else{
            rate = winterRate;
        }

        for(int day = 0; day< rentDays[i]*roomNumber;){
            //get period by random number
            int period = getPeriod(rate,rand()%100);
            day = day+period;
            stringstream reservation_str;
            stringstream start_str;
            stringstream end_str;

            if(day>rentDays[i]*roomNumber){
                period = day-rentDays[i]*roomNumber;
                if(period==0){
                    break;
                }
            }

            int start_date = rand()%daysInMonth[i];

            //calculate reservation date.
            if(reserve_date<0){
                if(i==0){
                    temp_month = 11;
                }
                else{
                    temp_month = (i-1)%12;
                }
                temp_day = daysInMonth[temp_month]+reserve_date+1;
            }
            else{
                temp_month = i;
                temp_day = reserve_date+1;
            }
            temp_month = temp_month+1;
            reservation_str<<temp_month<<"--"<<temp_day;
            reservation_str>>tempReservation.reservationDate;

            //check-in date
            temp_month = i;
            temp_day = start_date+1;
            temp_month = temp_month+1;
            start_str<<temp_month<<"--"<<temp_day;
            start_str>>tempReservation.startDate;

            //check-out date
            if(end_date>=daysInMonth[i]){
                temp_month = (i+1)%12;
                temp_day = end_date-daysInMonth[temp_month]+1;
            }
            else{
                temp_month = i;
                temp_day = end_date+1;
            }
            temp_month = temp_month+1;
            end_str<<temp_month<<"--"<<temp_day;
            end_str>>tempReservation.endDate;
            //whether is an owner's reservation
            tempReservation.ownerFlag = 
(rand()%100<owner_percent);
            if(tempReservation.ownerFlag){
                tempReservation.ownerRoom = rand()%roomNumber+1;
            }
            else{
                tempReservation.ownerRoom = -1;
            }
            reservationList.push_back(tempReservation);
        }
    }
    writeList("reservation_short_new.csv", reservationList);
    return 0;
}
```

```
bool writeList(string file_name, vector<reservationData> data) {
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Reservation Date, Check-in Date, Check-out Date, Is
House Owner, Room Number"<<endl;

    for(int i = 0;i<data.size();i++){

fout<<data[i].reservationDate<<","<<data[i].startDate<<","<<data
[i].endDate<<","<<data[i].ownerFlag<<","<<data[i].ownerRoom<<end
l;
    }
    return true;
}

int getPeriod(int *rate, int randNumber) {
    int days = 0;
    for(int i = 0;i<7;i++)
    {
        if(randNumber<rate[i]){
            days = i+1;
            break;
        }
    }
    return days;
}
```

Appendix 2
Reservation.h

```
#ifndef RESERVATION_H
#define RESERVATION_H

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include "time.h"

using namespace std;
static int daysInMonth[] =
{31,28,31,30,31,30,31,31,30,31,30,31};

int getPeriod(int *rate, int randNumber);

int dateToTime(string date);

class reservationData {
public:

reservationData():ownerFlag(false),arrangeFlag(false),canInsertF
lag(false),arrangeRoom(rand()%100){}
    string reservationDate;
    string startDate;
    string endDate;
    int reservationTime;
    int startTime;
    int endTime;
    bool ownerFlag;
    int ownerRoom;
    bool arrangeFlag;
    bool canInsertFlag;
    int arrangeRoom;
};

class Room{
public:
    Room();
    bool checkAvailable(int startTime, int endTime);
    bool insertReservation(int startTime, int endTime);
    bool insertOwnReservation(int startTime, int endTime);
    vector<bool> available;
    int lastVisitTime;
    int livingTime;
    int ownerLivingTime;
    int index;
};

class Reservation{
public:
    Reservation(int roomNumber);
    void insertReservation(reservationData &r);
    bool writeRoomList(string file_name);
private:
    vector<Room> roomList;
    vector<pair<int,int> > sortedList;
};


bool loadList(string file_name, vector<reservationData> &data);

bool writeList(string file_name, vector<reservationData> &data);

bool writeArrangedList(string file_name, vector<reservationData>
&data);

bool compareReservationDataByStart(reservationData r1,
reservationData r2);

bool compareReservationDataByReservation(reservationData r1,
reservationData r2);

bool compareRoomByLiving(Room r1, Room r2);

bool compareRoomByLastVisit(Room r1, Room r2);

#endif
```

Main.cpp

```
//  main.cpp
//  Reservation
//
//  Created by Video on 7/7/15.
//  Copyright (c) 2015 Video. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include "time.h"
#include "reservation.h"

using namespace std;
//percentage of rental period

static const int roomNumber = 100;

int main(int argc, const char * argv[])
{

    Reservation myRes(roomNumber);
    vector<reservationData> reservationList;

    loadList("../data/reservation_short_new.csv",
reservationList);


sort(reservationList.begin(),reservationList.end(),&compareReser
vationDataByStart);

    for(auto &s:reservationList){
        if(s.ownerFlag){
            myRes.insertReservation(s);
        }
    }

    for(auto &s:reservationList){
        if(!s.arrangeFlag){
            myRes.insertReservation(s);
        }
    }
    writeArrangedList("../data/Arranged_Reservation_new.csv",
reservationList);
    myRes.writeRoomList("../data/room_List_new.csv");
    return 0;
}
```

Reservation.cpp

```
//  main.cpp
//  Reservation
//
//  Created by Video on 7/7/15.
//  Copyright (c) 2015 Video. All rights reserved.
//
#include "reservation.h"
#include <algorithm>
//initial conditions and build room list
Room::Room(){
    available = vector<bool> (365+30,true);
    lastVisitTime = 0;
    livingTime = 0;
    ownerLivingTime = 0;
}
//check availability
bool Room::checkAvailable(int startTime, int endTime){
    for(int i = startTime;i<endTime;i++){
        if(!available[i]){
            return false;
        }
    }
    return true;
}
//arrange reservation and update room list
bool Room::insertReservation(int startTime, int endTime){
    if(!checkAvailable(startTime,endTime)){
        return false;
    }

    for(int i = startTime;i<endTime;i++){
        available[i] = false;
    }

    lastVisitTime = max(lastVisitTime, endTime);
    livingTime = livingTime + endTime - startTime;
    return true;
}
//pair room number with living time and owner living time
Reservation::Reservation(int roomNumber){
    roomList = vector<Room>(roomNumber,Room());
    int index = 0;
    for(auto &s:roomList){
        s.index = index+1;
        pair<int,int> indexPair(index,s.livingTime);
        sortedList.push_back(indexPair);
        index++;
    }
}

bool comparePair(pair<int,int> a, pair<int,int> b){
    return a.second<b.second;
}
//owner's reservation?
void Reservation::insertReservation(reservationData &r){
    if(r.arrangeFlag){
        return;
    }
    bool flag = false;
    if(r.ownerFlag){
        if(roomList[r.ownerRoom-
1].insertOwnReservation(r.startTime, r.endTime)){
            r.canInsertFlag = true;
        }
        else{
            r.canInsertFlag = false;
        }
        r.arrangeFlag = true;
    }
    else{
        sort(sortedList.begin(),sortedList.end(),&comparePair);
        for(int i = 0;i<sortedList.size();i++){

if(roomList[sortedList[i].first].insertReservation(r.startTime,
r.endTime)){
                r.arrangeRoom =
roomList[sortedList[i].first].index;
                flag = true;
                r.canInsertFlag = true;
                sortedList[i].second =
roomList[sortedList[i].first].livingTime;
                break;
            }
        }
        r.arrangeFlag = true;
        if(!flag){
            r.canInsertFlag = false;
            r.arrangeRoom = -1;
        }
    }
}
```

```
//read reservation data
bool Room::insertOwnReservation(int startTime, int endTime){
    if(!checkAvailable(startTime,endTime)){
        return false;
    }

    for(int i = startTime;i<endTime;i++){
        available[i] = false;
    }
    ownerLivingTime = ownerLivingTime + endTime - startTime;
    return true;
}

bool loadList(string file_name, vector<reservationData> &data) {
    data.clear();
    ifstream fin(file_name);
    if(!fin.is_open()){
        cout<<"Can't load file"<<endl;
        return false;
    }

    string tempStr;
    getline(fin,tempStr);
    while(getline(fin,tempStr)){
        reservationData tempData;
        size_t prePos = -1;
        size_t pos = tempStr.find(",",prePos+1);

tempData.reservationDate.assign(tempStr.begin()+prePos+1,tempStr
.begin()+pos);
        tempData.reservationTime =
dateToTime(tempData.reservationDate);

        prePos = pos;
        pos = tempStr.find(",",prePos+1);

tempData.startDate.assign(tempStr.begin()+prePos+1,tempStr.begin
()+pos);
        tempData.startTime = dateToTime(tempData.startDate);

        prePos = pos;
        pos = tempStr.find(",",prePos+1);

tempData.endDate.assign(tempStr.begin()+prePos+1,tempStr.begin()
+pos);
        tempData.endTime = dateToTime(tempData.endDate);
        //cross year
        if(tempData.endTime<tempData.startTime){
            tempData.endTime = tempData.endTime+365;
        }

        string str;
        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        str.assign(tempStr.begin()+prePos+1,tempStr.end());
        tempData.ownerFlag = atoi(str.c_str());

        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        if(pos==string::npos){
            str.assign(tempStr.begin()+prePos+1,tempStr.end());
            tempData.ownerRoom = atoi(str.c_str());
        }
        else{

str.assign(tempStr.begin()+prePos+1,tempStr.begin()+pos);
            tempData.ownerRoom = atoi(str.c_str());

            prePos = pos;
            pos = tempStr.find(",",prePos+1);

str.assign(tempStr.begin()+prePos+1,tempStr.begin()+pos);
            tempData.arrangeFlag = atoi(str.c_str());

            prePos = pos;
            str.assign(tempStr.begin()+prePos+1,tempStr.end()-1);
            tempData.arrangeRoom = atoi(str.c_str());
        }
        data.push_back(tempData);
    }
    return true;
}

bool compareReservationDataByStart(reservationData r1,
reservationData r2){
    return r1.startTime<r2.startTime;
}

bool compareReservationDataByReservation(reservationData r1,
reservationData r2){
    return r1.reservationTime<r2.reservationTime;
}
```

```
bool compareRoomByLiving(Room r1, Room r2){
    return r1.livingTime<r2.livingTime;
}

bool compareRoomByLastVisit(Room r1, Room r2){
    return r1.lastVisitTime<r2.lastVisitTime;
}

bool writeList(string file_name, vector<reservationData> data) {
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Reservation Date, Check-in Date, Check-out Date, Is
House Owner"<<endl;

    for(int i = 0;i<data.size();i++){

fout<<data[i].reservationDate<<","<<data[i].startDate<<","<<data
[i].endDate<<","<<data[i].ownerFlag<<endl;
    }

    return true;
}

bool Reservation::writeRoomList(string file_name){
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Room Number, Living Time, Owner Living Time"<<endl;
    for(auto &r:roomList){

fout<<r.index<<","<<r.livingTime<<","<<r.ownerLivingTime<<endl;
    }

    return true;
}

bool writeArrangedList(string file_name, vector<reservationData>
&data){
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }


    fout<<"Reservation Date, Check-in Date, Check-out Date, Is
House Owner, Whether Arranged, Room Number"<<endl;
    for(int i = 0;i<data.size();i++){

fout<<data[i].reservationDate<<","<<data[i].startDate<<","<<data
[i].endDate<<","<<data[i].ownerFlag<<","<<data[i].canInsertFlag<
<","<<data[i].arrangeRoom<<endl;
    }

    return true;
}
//tranform date to time
int dateToTime(string date){
    int month;
    int day;
    int time = 0;
    sscanf(date.c_str(), "%d--%d", &month, &day);

    for(int i = 0;i<month-1;i++){
        time = time+daysInMonth[i];
    }

    time = time+day-1;
    return time;
}

int getPeriod(int *rate, int randNumber) {
    int days = 0;
    for(int i = 0;i<7;i++)
    {
        if(randNumber<rate[i]){
            days = i+1;
            break;
        }
    }
    return days;
}
```

Appendix 3

```
/
//  main.cpp
//  Reservation
//  Created by Video on 7/7/15.
//  Copyright (c) 2015 Video. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include "time.h"

using namespace std;
//percentage of rental period
static int summerRate[] = {5,30,30,15,10,5,5};
static int winterRate[] = {35,35,20,7,1,1,1};
//Is summer or not
static const bool summerFlag[] =
{false,false,false,false,false,true,true,
                             true,false,false,false,false};
//average rental period for each month
static int rentDays[] = {13,13,13,13,23,23,23,28,23,13,13,13};
//static int rentDays[] = {13,13,13,13,13,13,13,13,13,13,13,13};
static int daysInMonth[] =
{31,28,31,30,31,30,31,31,30,31,30,31};
//number of rooms
static const int roomNumber = 100;
static const int monthNumber = 12;
//percentage of owner's reservations.
static const int owner_percent = 5;
static const int number_percent = 5;

int getPeriod(int *rate, int randNumber);

/*struct reservationData {
    string reservationDate;
    string startDate;
    string endDate;
    bool ownerFlag;
    int ownerRoom;
};*/

class reservationData {
public:
    string reservationDate;
    string startDate;
    string endDate;
    int reservationTime;
    int startTime;
    int endTime;
    bool ownerFlag;
    int ownerRoom;
    bool arrangeFlag;
    int arrangeRoom;
};


bool writeList(string file_name, vector<reservationData> data);

int main(int argc, const char * argv[])
{
    vector<reservationData> reservationList;

    srand((unsigned int)time(NULL));
    for(int i = 0;i<monthNumber; i++){
        int offset = rand()%5-3;
        rentDays[i] = rentDays[i] + offset;
    }

    for(int i = 1;i<7;i++) {
        summerRate[i] = summerRate[i] + summerRate[i-1];
        winterRate[i] = winterRate[i] + winterRate[i-1];
    }

    for(int i = 0; i<monthNumber; i++) {
        int *rate;
        if(summerFlag[i]){
            rate = summerRate;
        }
        else{
            rate = winterRate;
        }

        for(int day = 0; day< rentDays[i]*roomNumber;){
            //get period by random number
            int period = getPeriod(rate,rand()%100);
            day = day+period;
            stringstream reservation_str;
            stringstream start_str;
            stringstream end_str;
```

```
if(day>rentDays[i]*roomNumber){
    period = day-rentDays[i]*roomNumber;
    if(period==0){
        break;
    }
}

int start_date = rand()%daysInMonth[i];

//calculate reservation date.
if(reserve_date<0){
    if(i==0){
        temp_month = 11;
    }
    else{
        temp_month = (i-1)%12;
    }
    temp_day = daysInMonth[temp_month]+reserve_date+1;
}
else{
    temp_month = i;
    temp_day = reserve_date+1;
}
temp_month = temp_month+1;
reservation_str<<temp_month<<"--"<<temp_day;
reservation_str>>tempReservation.reservationDate;

//check-in date
temp_month = i;
temp_day = start_date+1;
temp_month = temp_month+1;
start_str<<temp_month<<"--"<<temp_day;
start_str>>tempReservation.startDate;

//check-out date
if(end_date>=daysInMonth[i]){
    temp_month = (i+1)%12;
    temp_day = end_date-daysInMonth[i]+1;
}
else{
    temp_month = i;
    temp_day = end_date+1;
}
temp_month = temp_month+1;
end_str<<temp_month<<"--"<<temp_day;
```

```
            end_str>>tempReservation.endDate;
            //whether is an owner's reservation
            tempReservation.ownerFlag =
(rand()%100<owner_percent);
            tempReservation.arrangeFlag = false;
            tempReservation.arrangeRoom = -1;
            if(tempReservation.ownerFlag){
                tempReservation.ownerRoom = rand()%roomNumber+1;
                tempReservation.arrangeRoom =
tempReservation.ownerRoom;
            }
            else{
                tempReservation.ownerRoom = -1;
                if(rand()%100<number_percent){
                    tempReservation.arrangeRoom =
rand()%roomNumber+1;
                }
                else{
                    tempReservation.arrangeRoom = -1;
                }
            }
            reservationList.push_back(tempReservation);
        }
    }
    writeList("../../data/reservation.csv", reservationList);
    return 0;
}

bool writeList(string file_name, vector<reservationData> data) {
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Reservation Date, Check-in Date, Check-out Date, Is
House Owner, Room Number, Is Arranged, Arranged Number"<<endl;

    for(int i = 0;i<data.size();i++){

fout<<data[i].reservationDate<<","<<data[i].startDate<<","

<<data[i].endDate<<","<<data[i].ownerFlag<<","<<data[i].ownerRoo
m<<","
        <<data[i].arrangeFlag<<","<<data[i].arrangeRoom<<endl;
    }
    return true;
}

int getPeriod(int *rate, int randNumber) {
    int days = 0;
    for(int i = 0;i<7;i++)
    {
        if(randNumber<rate[i]){
            days = i+1;
            break;
        }
    }
    return days;
}
```

Appendix 4
Reservation.h
```
#ifndef RESERVATION_H
#define RESERVATION_H

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include "time.h"

using namespace std;
static int daysInMonth[] =
{31,28,31,30,31,30,31,31,30,31,30,31};

static int dateSpilt[] = {103,145,197,247,264,299,364};
static int moneySplit[] = {149,199,269,349,269,199,149};

int getPeriod(int *rate, int randNumber);

int dateToTime(string date);

class reservationData {
public:

reservationData():ownerFlag(false),arrangeFlag(false),canInsertF
lag(false),arrangeRoom(-1){}//arrangeRoom(rand()%100)
    string reservationDate;
    string startDate;
    string endDate;
    int reservationTime;
    int startTime;
    int endTime;
    bool ownerFlag;
    int ownerRoom;
    bool arrangeFlag;
    bool canInsertFlag;
    int arrangeRoom;
};

class Room{
public:
    Room();
    bool checkAvailable(int startTime, int endTime);
    bool insertReservation(int startTime, int endTime);
    bool insertReservation(int startTime, int endTime, int
roomIndex);
    bool insertOwnReservation(int startTime, int endTime);

    vector<bool> available;
    int lastVisitTime;
    int livingTime;
    int moneyEarn;
    int moneyVirtual;
    int ownerLivingTime;
    int index;
};

class Reservation{
public:
    Reservation(int roomNumber);
    void insertReservation(reservationData &r);
    void insertReservation_Conventional(reservationData &r);
    bool writeRoomList(string file_name);
private:
    vector<Room> roomList;
    vector<pair<int,int> > sortedList;
};


bool loadList(string file_name, vector<reservationData> &data);

bool writeList(string file_name, vector<reservationData> &data);

bool writeArrangedList(string file_name, vector<reservationData>
&data);

bool compareReservationDataByStart(reservationData r1,
reservationData r2);

bool compareReservationDataByReservation(reservationData r1,
reservationData r2);

bool compareRoomByLiving(Room r1, Room r2);

bool compareRoomByLastVisit(Room r1, Room r2);

int calculateMoney(int startTime, int endTime);

int calculateMoneyOwn(int startTime, int endTime, int
livedDays);
#endif
```

```
//  main.cpp
//  Reservation
//
//  Created by Video on 7/7/15.
//  Copyright (c) 2015 Video. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include "time.h"

#include "reservation.h"

using namespace std;
//percentage of rental period

static const int roomNumber = 100;

int main(int argc, const char * argv[])
{

    Reservation myRes(roomNumber);
    vector<reservationData> reservationList;

    loadList("../../data/reservation.csv", reservationList);


sort(reservationList.begin(),reservationList.end(),&compareReser
vationDataByReservation);

    for(auto &s:reservationList){
        if(s.ownerFlag){
            //myRes.insertReservation(s);
            myRes.insertReservation_Conventional(s);
        }
    }

    for(auto &s:reservationList){
        if(!s.arrangeFlag){
            //myRes.insertReservation(s);
            myRes.insertReservation_Conventional(s);
        }
    }
writeArrangedList("../../data/Arranged_Reservation_Conventional.
csv", reservationList);
    myRes.writeRoomList("../../data/room_List_Conventional.csv");
    /*string dateList[] = {"4--14","5--26","7--17","9--5","9--
22","10--27","12--31"};
    for (int i = 0;i<7;i++){
        cout<<dateToTime(dateList[i])<<endl;
    }*/

    return 0;
}
```

```
//  reservation.cpp
//  Reservation
//
//  Created by Video on 7/7/15.
//  Copyright (c) 2015 Video. All rights reserved.
//
#include "reservation.h"

Room::Room(){
    available = vector<bool> (365+30,true);
    lastVisitTime = 0;
    livingTime = 0;
    ownerLivingTime = 0;
    moneyEarn = 0;
    moneyVirtual = 0;
}

bool Room::checkAvailable(int startTime, int endTime){
    for(int i = startTime;i<endTime;i++){
        if(!available[i]){
            return false;
        }
    }
    return true;
}

bool Room::insertReservation(int startTime, int endTime){
    if(!checkAvailable(startTime,endTime)){
        return false;
    }

    for(int i = startTime;i<endTime;i++){
        available[i] = false;
    }

    lastVisitTime = max(lastVisitTime, endTime);
    livingTime = livingTime + endTime - startTime;
    moneyEarn = moneyEarn + calculateMoney(startTime, endTime);
    moneyVirtual = moneyVirtual + calculateMoney(startTime,
endTime);
    return true;
}

bool Room::insertOwnReservation(int startTime, int endTime){
    if(!checkAvailable(startTime,endTime)){
        return false;
    }

    for(int i = startTime;i<endTime;i++){
        available[i] = false;
    }

    moneyVirtual = moneyVirtual + calculateMoneyOwn(startTime, 
endTime, ownerLivingTime);
    ownerLivingTime = ownerLivingTime + endTime - startTime;

    return true;
}


Reservation::Reservation(int roomNumber){
    roomList = vector<Room>(roomNumber,Room());
    int index = 0;
    for(auto &s:roomList){
        s.index = index+1;
        pair<int,int> indexPair(index,s.moneyVirtual);
        sortedList.push_back(indexPair);
        index++;
    }
}

bool comparePair(pair<int,int> a, pair<int,int> b){
    return a.second<b.second;
}

void Reservation::insertReservation_Conventional(reservationData 
&r){
    if(r.arrangeFlag){
        return;
    }
    bool flag = false;
    if(r.ownerFlag){
        if(roomList[r.ownerRoom-
1].insertOwnReservation(r.startTime, r.endTime)){
            r.canInsertFlag = true;
            flag = true;
            r.arrangeRoom = r.ownerRoom;
        }
        else{
            r.canInsertFlag = false;
            r.arrangeRoom = -1;
        }
```

```
            r.arrangeFlag = true;
        }
        else{
            if(r.arrangeRoom!=-1){
                if(roomList[r.arrangeRoom-
1].insertReservation(r.startTime, r.endTime)){
                    r.arrangeRoom = roomList[r.arrangeRoom-1].index;
                    flag = true;
                    r.canInsertFlag = true;
                    for(auto &s:sortedList){
                        if(s.first==r.arrangeRoom){
                            s.second = roomList[r.arrangeRoom-
1].moneyVirtual;
                            break;
                        }
                    }
                }
            }

            if(!flag){
                for(int i = 0;i<sortedList.size();i++){

if(roomList[sortedList[i].first].insertReservation(r.startTime,
r.endTime)){
                        r.arrangeRoom =
roomList[sortedList[i].first].index;
                        flag = true;
                        r.canInsertFlag = true;
                        pair<int,int> indexTemp = sortedList[i];
                        sortedList.erase(sortedList.begin()+i);
                        sortedList.push_back(indexTemp);
                        break;
                    }
                }
            }

            r.arrangeFlag = true;
            if(!flag){
                r.canInsertFlag = false;
                r.arrangeRoom = -1;
            }
        }
}


void Reservation::insertReservation(reservationData &r){
        if(r.arrangeFlag){
            return;
        }
        bool flag = false;
        if(r.ownerFlag){
            if(roomList[r.ownerRoom-
1].insertOwnReservation(r.startTime, r.endTime)){
                r.canInsertFlag = true;
                r.arrangeRoom = r.ownerRoom;
                flag = true;
                for(auto &s:sortedList){
                    if(s.first==r.arrangeRoom){
                        s.second = roomList[r.arrangeRoom-
1].moneyVirtual;
                        break;
                    }
                }
            }
            else{
                r.canInsertFlag = false;
                r.arrangeRoom = -1;
            }
            r.arrangeFlag = true;
        }
        else{
            if(r.arrangeRoom!=-1){
                if(roomList[r.arrangeRoom-
1].insertReservation(r.startTime, r.endTime)){
                    r.arrangeRoom = roomList[r.arrangeRoom-1].index;
                    flag = true;
                    r.canInsertFlag = true;
                    for(auto &s:sortedList){
                        if(s.first==r.arrangeRoom){
                            s.second = roomList[r.arrangeRoom-
1].moneyVirtual;
                            break;
                        }
                    }
                }
            }

            if(!flag){

sort(sortedList.begin(),sortedList.end(),&comparePair);
                for(int i = 0;i<sortedList.size();i++){
```

```
if(roomList[sortedList[i].first].insertReservation(r.startTime,
r.endTime)){
                    r.arrangeRoom =
roomList[sortedList[i].first].index;
                    flag = true;
                    r.canInsertFlag = true;
                    sortedList[i].second =
roomList[sortedList[i].first].moneyVirtual;
                    break;
                }
            }
        }

        r.arrangeFlag = true;
        if(!flag){
            r.canInsertFlag = false;
            r.arrangeRoom = -1;
        }
    }
}

bool loadList(string file_name, vector<reservationData> &data) {
    data.clear();
    ifstream fin(file_name);
    if(!fin.is_open()){
        cout<<"Can't load file"<<endl;
        return false;
    }

    string tempStr;
    getline(fin,tempStr);
    while(getline(fin,tempStr)){
        reservationData tempData;
        int prePos = -1;
        size_t pos = tempStr.find(",",prePos+1);

tempData.reservationDate.assign(tempStr.begin()+prePos+1,tempStr
.begin()+pos);
        tempData.reservationTime =
dateToTime(tempData.reservationDate);

        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        tempData.startDate.assign(tempStr,prePos+1,pos-
(prePos+1));
        tempData.startTime = dateToTime(tempData.startDate);
        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        tempData.endDate.assign(tempStr,prePos+1,pos-(prePos+1));
        tempData.endTime = dateToTime(tempData.endDate);
        //cross year
        if(tempData.endTime<tempData.startTime){
            tempData.endTime = tempData.endTime+365;
        }

        string str;
        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        str.assign(tempStr,prePos+1,pos-(prePos+1));
        tempData.ownerFlag = atoi(str.c_str());

        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        if(pos==string::npos){
            str.assign(tempStr,prePos+1,pos-(prePos+1));
            tempData.ownerRoom = atoi(str.c_str());
        }
        else{
            str.assign(tempStr,prePos+1,pos-(prePos+1));
            tempData.ownerRoom = atoi(str.c_str());

            prePos = pos;
            pos = tempStr.find(",",prePos+1);
            str.assign(tempStr,prePos+1,pos-(prePos+1));
            tempData.arrangeFlag = atoi(str.c_str());

            prePos = pos;
            str.assign(tempStr,prePos+1,pos-(prePos+1));
            tempData.arrangeRoom = atoi(str.c_str());
        }
        data.push_back(tempData);
    }
    return true;
}

bool compareReservationDataByStart(reservationData r1,
reservationData r2){
    return r1.startTime<r2.startTime;
}

bool compareReservationDataByReservation(reservationData r1,
reservationData r2){
    return r1.reservationTime<r2.reservationTime;
}

bool compareRoomByLiving(Room r1, Room r2){
    return r1.livingTime<r2.livingTime;
}

bool compareRoomByLastVisit(Room r1, Room r2){
    return r1.lastVisitTime<r2.lastVisitTime;
}

bool writeList(string file_name, vector<reservationData> data) {
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Reservation Date, Check-in Date, Check-out Date, Is
House Owner"<<endl;

    for(int i = 0;i<data.size();i++){

fout<<data[i].reservationDate<<","<<data[i].startDate<<","<<data
[i].endDate<<","<<data[i].ownerFlag<<endl;
    }

    return true;
}

bool Reservation::writeRoomList(string file_name){
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Room Number, Living Time, Owner Living Time,
Income"<<endl;
    for(auto &r:roomList){

fout<<r.index<<","<<r.livingTime<<","<<r.ownerLivingTime<<","<<r
.moneyEarn<<endl;
    }
    return true;
}

bool writeArrangedList(string file_name, vector<reservationData>
&data){
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Reservation Date, Check-in Date, Check-out Date, Is
House Owner, Whether Arranged, Room Number"<<endl;
    for(int i = 0;i<data.size();i++){

fout<<data[i].reservationDate<<","<<data[i].startDate<<","<<data
[i].endDate<<","

<<data[i].ownerFlag<<","<<data[i].canInsertFlag<<","<<data[i].ar
rangeRoom<<endl;
    }

    return true;
}


int dateToTime(string date){
    int month;
    int day;
    int time = 0;
    sscanf(date.c_str(), "%d--%d", &month, &day);

    for(int i = 0;i<month-1;i++){
        time = time+daysInMonth[i];
    }

    time = time+day-1;
    return time;
}

int getPeriod(int *rate, int randNumber) {
    int days = 0;
    for(int i = 0;i<7;i++)
    {
        if(randNumber<rate[i]){
            days = i+1;
            break;
        }
    }
    return days;
}

int calculateMoney(int startTime, int endTime){
    int sum = 0;
    for(int i = startTime; i<endTime; i++){
        int cur = moneySplit[0];
        for(int r = 0;r<7;r++){
            if(i<=r){
                cur  = moneySplit[r];
                break;
            }
        }
        sum = sum+cur;
    }
    return sum;

}

int calculateMoneyOwn(int startTime, int endTime, int
livedDays){
    int sum = 0;
    for(int i = startTime; i<endTime; i++){
        if(livedDays<10){
            livedDays++;
            continue;
        }
        else{
            int cur = moneySplit[0];
            for(int r = 0;r<7;r++){
                if(i<=r){
                    cur  = moneySplit[r];
                    break;
                }
            }
            sum = sum+cur;
            livedDays++;
        }
    }
    return sum;
}
```

Appendix 4
Main.cpp
```
//
//  main.cpp
//  Reservation
//
//  Created by Video on 7/7/15.
//  Copyright (c) 2015 Video. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include "time.h"

#include "reservation.h"

using namespace std;
//percentage of rental period

static const int roomNumber = 100;

int main(int argc, const char * argv[])
{

    Reservation myRes(roomNumber);
    vector<reservationData> reservationList;
    readParameter("../../data/config.txt");
    loadList("../../data/reservation.csv", reservationList);


sort(reservationList.begin(),reservationList.end(),&compareReser
vationDataByReservation);

    for(auto &s:reservationList){
        if(s.ownerFlag){
            myRes.insertReservation(s);
            //myRes.insertReservation_Conventional(s);
        }
    }

    for(auto &s:reservationList){
        if(!s.arrangeFlag){
            myRes.insertReservation(s);
            //myRes.insertReservation_Conventional(s);
        }
    }

    writeArrangedList("../../data/Arranged_Reservation_1.csv",
reservationList);
    myRes.writeRoomList("../../data/room_List_1.csv");


    return 0;
}
```

Reservation.cpp

```
//
//  main.cpp
//  Reservation
//
//  Created by Video on 7/7/15.
//  Copyright (c) 2015 Video. All rights reserved.
//
#include "reservation.h"

Room::Room(){
    available = vector<bool> (365+30,true);
    lastVisitTime = 0;
    livingTime = 0;
    ownerLivingTime = 0;
    moneyEarn = 0;
    moneyVirtual = 0;
}

bool Room::checkAvailable(int startTime, int endTime){
    for(int i = startTime;i<endTime;i++){
        if(!available[i]){
            return false;
        }
    }
    return true;
}

bool Room::insertReservation(int startTime, int endTime){
    if(!checkAvailable(startTime,endTime)){
        return false;
    }

    for(int i = startTime;i<endTime;i++){
        available[i] = false;
    }

    lastVisitTime = max(lastVisitTime, endTime);
    livingTime = livingTime + endTime - startTime;
    moneyEarn = moneyEarn + calculateMoney(startTime, endTime);
    moneyVirtual = moneyVirtual + calculateMoney(startTime, 
endTime);
    return true;
}

bool Room::insertOwnReservation(int startTime, int endTime){
    if(!checkAvailable(startTime,endTime)){
        return false;
    }

    for(int i = startTime;i<endTime;i++){
        available[i] = false;
    }

    moneyVirtual = moneyVirtual + calculateMoneyOwn(startTime,
endTime, ownerLivingTime);
    ownerLivingTime = ownerLivingTime + endTime - startTime;

    return true;
}


Reservation::Reservation(int roomNumber){
    roomList = vector<Room>(roomNumber,Room());
    int index = 0;
    for(auto &s:roomList){
        s.index = index+1;
        pair<int,int> indexPair(index,s.moneyVirtual);
        sortedList.push_back(indexPair);
        index++;
    }
}

bool comparePair(pair<int,int> a, pair<int,int> b){
    return a.second<b.second;
}

void Reservation::insertReservation_Conventional(reservationData
&r){
    if(r.arrangeFlag){
        return;
    }
    bool flag = false;
    if(r.ownerFlag){
        if(roomList[r.ownerRoom-
1].insertOwnReservation(r.startTime, r.endTime)){
            r.canInsertFlag = true;
            flag = true;
            r.arrangeRoom = r.ownerRoom;
        }
        else{
            r.canInsertFlag = false;
            r.arrangeRoom = -1;
        }
        r.arrangeFlag = true;
    }
    else{
        if(consideringCustomRoomNumber){
            if(r.arrangeRoom!=-1){
                if(roomList[r.arrangeRoom-
1].insertReservation(r.startTime, r.endTime)){
                    r.arrangeRoom = roomList[r.arrangeRoom-
1].index;
                    flag = true;
                    r.canInsertFlag = true;
                    for(auto &s:sortedList){
                        if(s.first==r.arrangeRoom){
                            s.second = roomList[r.arrangeRoom-
1].moneyVirtual;
                            break;
                        }
                    }
                }
            }
        }

        if(!flag){
            for(int i = 0;i<sortedList.size();i++){

if(roomList[sortedList[i].first].insertReservation(r.startTime,
r.endTime)){
                    r.arrangeRoom =
roomList[sortedList[i].first].index;
                    flag = true;
                    r.canInsertFlag = true;
                    pair<int,int> indexTemp = sortedList[i];
                    sortedList.erase(sortedList.begin()+i);
                    sortedList.push_back(indexTemp);
                    break;
                }
            }
        }

        r.arrangeFlag = true;
        if(!flag){
            r.canInsertFlag = false;
            r.arrangeRoom = -1;
        }
    }
}


void Reservation::insertReservation(reservationData &r){
    if(r.arrangeFlag){
        return;
    }
    bool flag = false;
    if(r.ownerFlag){
        if(roomList[r.ownerRoom-
1].insertOwnReservation(r.startTime, r.endTime)){
            r.canInsertFlag = true;
            r.arrangeRoom = r.ownerRoom;
            flag = true;
            for(auto &s:sortedList){
                if(s.first==r.arrangeRoom){
                    s.second = roomList[r.arrangeRoom-
1].moneyVirtual;
                    break;
                }
            }
        }
        else{
            r.canInsertFlag = false;
            r.arrangeRoom = -1;
        }
        r.arrangeFlag = true;
    }
    else{
        //add such flag
        if(consideringCustomRoomNumber){
            if(r.arrangeRoom!=-1){
                if(roomList[r.arrangeRoom-
1].insertReservation(r.startTime, r.endTime)){
                    r.arrangeRoom = roomList[r.arrangeRoom-
1].index;
                    flag = true;
                    r.canInsertFlag = true;
                    for(auto &s:sortedList){
                        if(s.first==r.arrangeRoom){
                            s.second = roomList[r.arrangeRoom-
1].moneyVirtual;
                            break;
                        }
                    }
                }
            }
        }

        if(!flag){

sort(sortedList.begin(),sortedList.end(),&comparePair);
            for(int i = 0;i<sortedList.size();i++){

if(roomList[sortedList[i].first].insertReservation(r.startTime,
r.endTime)){
                    r.arrangeRoom =
roomList[sortedList[i].first].index;
                    flag = true;
                    r.canInsertFlag = true;
                    sortedList[i].second =
roomList[sortedList[i].first].moneyVirtual;
                    break;
                }
            }
        }

        r.arrangeFlag = true;
        if(!flag){
            r.canInsertFlag = false;
            r.arrangeRoom = -1;
        }
    }
}

bool loadList(string file_name, vector<reservationData> &data) {
    data.clear();
    ifstream fin(file_name);
    if(!fin.is_open()){
        cout<<"Can't load file"<<endl;
        return false;
    }

    string tempStr;
    getline(fin,tempStr);
    while(getline(fin,tempStr)){
        reservationData tempData;
        int prePos = -1;
        size_t pos = tempStr.find(",",prePos+1);

tempData.reservationDate.assign(tempStr.begin()+prePos+1,tempStr
.begin()+pos);
        tempData.reservationTime =
dateToTime(tempData.reservationDate);

        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        tempData.startDate.assign(tempStr,prePos+1,pos-
(prePos+1));
        tempData.startTime = dateToTime(tempData.startDate);

        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        tempData.endDate.assign(tempStr,prePos+1,pos-(prePos+1));
        tempData.endTime = dateToTime(tempData.endDate);
        //cross year
        if(tempData.endTime<tempData.startTime){
            tempData.endTime = tempData.endTime+365;
        }

        string str;
        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        str.assign(tempStr,prePos+1,pos-(prePos+1));
        tempData.ownerFlag = atoi(str.c_str());

        prePos = pos;
        pos = tempStr.find(",",prePos+1);
        if(pos==string::npos){
            str.assign(tempStr,prePos+1,pos-(prePos+1));
            tempData.ownerRoom = atoi(str.c_str());
        }
        else{
            str.assign(tempStr,prePos+1,pos-(prePos+1));
            tempData.ownerRoom = atoi(str.c_str());

            prePos = pos;
            pos = tempStr.find(",",prePos+1);
            str.assign(tempStr,prePos+1,pos-(prePos+1));
            tempData.arrangeFlag = atoi(str.c_str());

            prePos = pos;
            str.assign(tempStr,prePos+1,pos-(prePos+1));
            tempData.arrangeRoom = atoi(str.c_str());
        }
        data.push_back(tempData);
    }
    return true;
}

bool compareReservationDataByStart(reservationData r1,
reservationData r2){
    return r1.startTime<r2.startTime;
}

bool compareReservationDataByReservation(reservationData r1,
reservationData r2){
    return r1.reservationTime<r2.reservationTime;
}

bool compareRoomByLiving(Room r1, Room r2){
    return r1.livingTime<r2.livingTime;
}

bool compareRoomByLastVisit(Room r1, Room r2){
    return r1.lastVisitTime<r2.lastVisitTime;
}

bool writeList(string file_name, vector<reservationData> data) {
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Reservation Date, Check-in Date, Check-out Date, Is
House Owner"<<endl;

    for(int i = 0;i<data.size();i++){

fout<<data[i].reservationDate<<","<<data[i].startDate<<","<<data
[i].endDate<<","<<data[i].ownerFlag<<endl;
    }

    return true;
}

bool Reservation::writeRoomList(string file_name){
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }
    fout<<"Room Number, Living Time, Owner Living Time, 
Income"<<endl;
    for(auto &r:roomList){
        
fout<<r.index<<","<<r.livingTime<<","<<r.ownerLivingTime<<","<<r
.moneyEarn<<endl;
    }

    return true;
}

bool writeArrangedList(string file_name, vector<reservationData> 
&data){
    ofstream fout(file_name);
    if(!fout.is_open()){
        cout<<"Can't write file"<<endl;
        return false;
    }

    fout<<"Reservation Date, Check-in Date, Check-out Date, Is 
House Owner, Whether Arranged, Room Number"<<endl;
    for(int i = 0;i<data.size();i++){
        
fout<<data[i].reservationDate<<","<<data[i].startDate<<","<<data
[i].endDate<<","
            
<<data[i].ownerFlag<<","<<data[i].canInsertFlag<<","<<data[i].ar
rangeRoom<<endl;
    }

    return true;
}


int dateToTime(string date){
    int month;
    int day;
    int time = 0;
    sscanf(date.c_str(), "%d--%d", &month, &day);

    for(int i = 0;i<month-1;i++){
        time = time+daysInMonth[i];
    }

    time = time+day-1;
    return time;
}

int getPeriod(int *rate, int randNumber) {
    int days = 0;
    for(int i = 0;i<7;i++)
    {
        if(randNumber<rate[i]){
            days = i+1;
            break;
        }
    }
    return days;
}

int calculateMoney(int startTime, int endTime){
    int sum = 0;
    for(int i = startTime; i<endTime; i++){
        int cur = moneySplit[0];
        for(int r = 0;r<7;r++){
            if(i<=r){
                cur  = moneySplit[r];
                break;
            }
        }
        sum = sum+cur;
    }
    return sum;

}

int calculateMoneyOwn(int startTime, int endTime, int
livedDays){
    int sum = 0;
    for(int i = startTime; i<endTime; i++){
        if(!ownerPunishmentFlag||livedDays<ownerMaxLivingDays){
            livedDays++;
            continue;
        }
        else{
            int cur = moneySplit[0];
            for(int r = 0;r<7;r++){
                if(i<=r){
                    cur = moneySplit[r];
                    break;
                }
            }
            sum = sum+cur*ownerLivingPunishment;
            livedDays++;
        }
    }
    return sum;
}

void readParameter(string file_name){
    ifstream fin(file_name);
    string s;
    char buf[200];
    if(!fin.is_open()){
        cout<<"Can't load configuration file! Press any key to
use default value."<<endl;
        cin>>s;
        return;
    }
    int val;
    getline(fin,s);
    sscanf(s.c_str(),"%s %d", buf,&ownerMaxLivingDays);
    getline(fin,s);
    sscanf(s.c_str(),"%s %lf", buf,&ownerLivingPunishment);
    getline(fin,s);
    sscanf(s.c_str(),"%s %d", buf,&val);
    ownerPunishmentFlag = (val==1);
    getline(fin,s);
    sscanf(s.c_str(),"%s %d", buf,&val);
    consideringCustomRoomNumber = (val==1);

}
```

```
Reservation.h
#ifndef RESERVATION_H
#define RESERVATION_H

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include "time.h"

using namespace std;
static int daysInMonth[] = {31,28,31,30,31,30,31,31,30,31,30,31};

static int dateSpilt[] = {103,145,197,247,264,299,364};
static int moneySplit[] = {149,199,269,349,269,199,149};

int getPeriod(int *rate, int randNumber);

int dateToTime(string date);

static int ownerMaxLivingDays = 10;
static double ownerLivingPunishment = 1.0;
static bool ownerPunishmentFlag = true;
static bool consideringCustomRoomNumber = true;

class reservationData {
public:

reservationData():ownerFlag(false),arrangeFlag(false),canInsertFlag(false),arrangeRoom
(-1){}//arrangeRoom(rand()%100)
	string reservationDate;
	string startDate;
	string endDate;
	int reservationTime;
	int startTime;
	int endTime;
	bool ownerFlag;
	int ownerRoom;
	bool arrangeFlag;
	bool canInsertFlag;
	int arrangeRoom;
};

class Room{
```

```
public:
        Room();
        bool checkAvailable(int startTime, int endTime);
        bool insertReservation(int startTime, int endTime);
        bool insertReservation(int startTime, int endTime, int roomIndex);
        bool insertOwnReservation(int startTime, int endTime);

        vector<bool> available;
        int lastVisitTime;
        int livingTime;
        int moneyEarn;
        int moneyVirtual;
        int ownerLivingTime;
        int index;
};

class Reservation{
public:
        Reservation(int roomNumber);
        void insertReservation(reservationData &r);
        void insertReservation_Conventional(reservationData &r);
        bool writeRoomList(string file_name);
private:
        vector<Room> roomList;
        vector<pair<int,int> > sortedList;
};


bool loadList(string file_name, vector<reservationData> &data);

void readParameter(string file_name);

bool writeList(string file_name, vector<reservationData> &data);

bool writeArrangedList(string file_name, vector<reservationData> &data);

bool compareReservationDataByStart(reservationData r1, reservationData r2);

bool compareReservationDataByReservation(reservationData r1, reservationData r2);

bool compareRoomByLiving(Room r1, Room r2);

bool compareRoomByLastVisit(Room r1, Room r2);
```

```
int calculateMoney(int startTime, int endTime);

int calculateMoneyOwn(int startTime, int endTime, int livedDays);

#endif
```

Appendix 5
Group 1
Owner_Max_Occupancy_Days 10
Owner_Occupancy_Penalty 1.0
Owner_Penalty_Flag 1
Considering_Custom_Room_Number_Flag 1

Vita

Peng GAO

27 Memorial Dr W
Department of Mechanical Engineering
Lehigh University
Bethlehem, PA, 18015
peg213@lehigh.edu

223 Summit St.
Bethlehem, PA, 18015

(260)5640258

Personal Information

Born: 1990/04/27, in Beijing China
Citizenship: Chinese
Mother: Yukun QI
Citizenship: Chinese
Father: Xinglong GAO
Citizenship: Chinese

Education

B.S Tsinghua University, Department of Thermal Engineering, Beijing, China, 2008-2012
R.A Gas Turbine Research Institute of Department of Thermal Engineering, Tsinghua University, 2012-2013

Research Experience

2011 Turbine blades through-flow partial Aerothermodynamic experiment and analysis
2011 Performance measurement experiments of large diameter hydraulic coupler
2010-2011 Research on compressor complex flow mechanism, modeling and control
2009 Gas turbine blade film cooling experiments, Student Research Training

.

Publications and Papers

Turbine blades through-flow partial Aerothermodynamic experiment and analysis, B.S thesis
2010 Numerical simulation of film cooling plane cascade under different blow ratio, Journal of Engineering Thermophysics