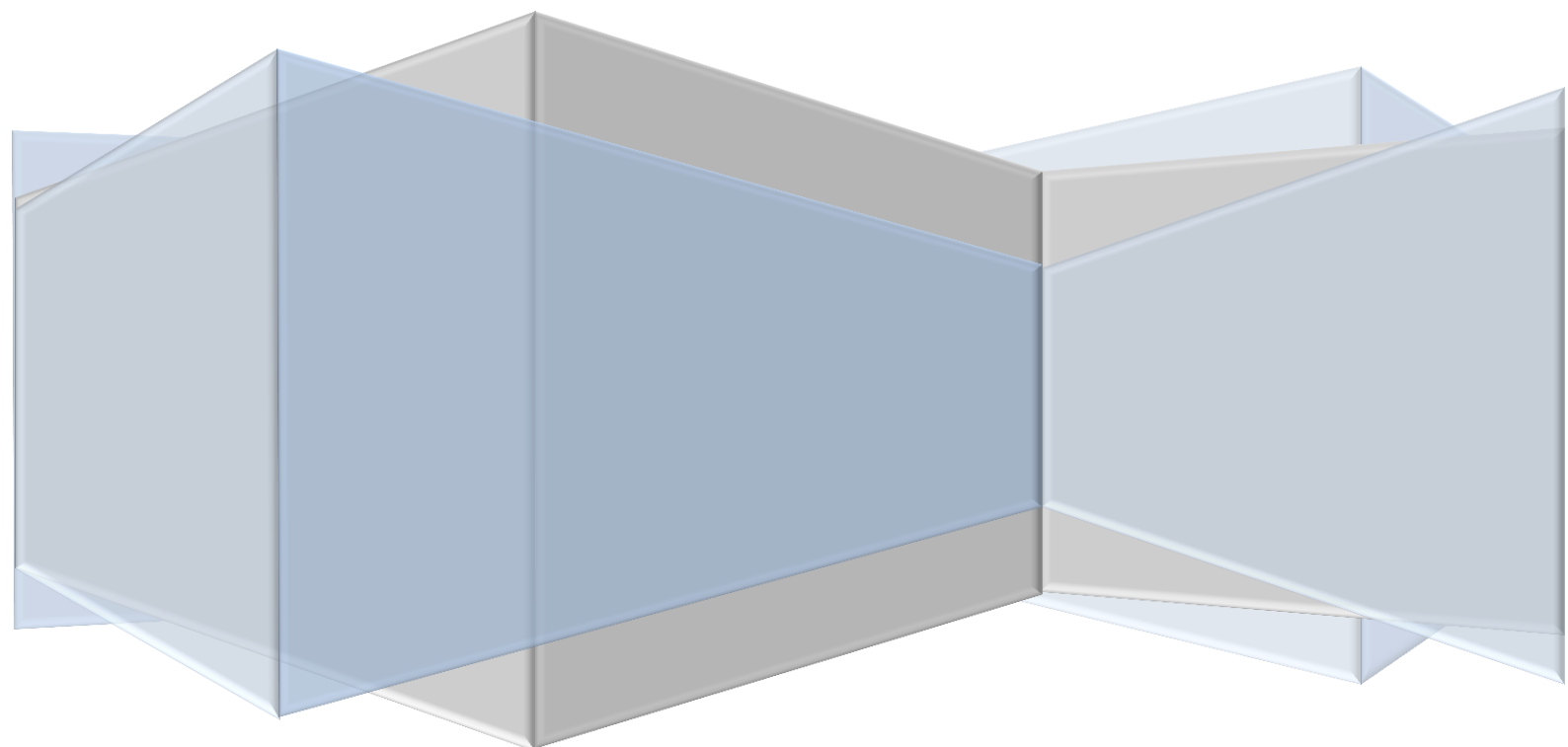


School-Based Assessment (SBA)

Module D: Software Development

Title: Phone book directory



Contents

Chapter 1 Introduction.....	2
1.1 Problem & Situation.....	2
1.2 Objectives.....	2
Chapter 2 Design of Solution	3
2.1 Brief Description	3
2.2 Intended users and their expectations	4
2.3 Refinement of Problem.....	6
2.4 User Identification.....	9
2.5 Data Protection	10
2.6 Structure of the Program	11
2.7 Design of Interface	13
Chapter 3 Implementation.....	19
3.1 Brief Description	19
3.2 Data Structure	20
3.3 Procedures in the Program	23
3.4 Main Body of the Program.....	37
Chapter 4 Testing & Evaluation	40
4.1 Brief Description	40
4.2 Testing and Evaluation Plan	40
4.3 Internal Tests.....	41
4.4 External Tests	51
Chapter 5 Evaluation.....	52
5.1 Pros and cons of my Program	52
5.2 Future Improvement.....	52
5.3 Self-Reflection	53
Chapter 6 Reference and Acknowledgement	54
Appendices:.....	54
Program Code	55

Chapter 1 Introduction

1.1 Problem & Situation

Nowadays, mobile phone becomes the compulsory part of our daily life. Due to the need of daily interaction, it is important for us to memory others' personal details. For this, we need a safe and convenient platform to save and view the data. In view of this need, I am going to develop a phone book system to save not only phone numbers but also the related data of that person.

1.2 Objectives

The phone book system was decided to include the following features:

- 1, *It should be easy to use.*
- 2, *It should be convenient in data management.*
- 3, *It should be able to protect the data.*

Chapter 2 Design of Solution

2.1 Brief Description

In this Chapter, I will design the program based on Chapter 1.

I assume that:

- 1, There are more than one user in the device.*
- 2, It is dangerous if data leakage occurs.*
- 3, There are some user expectations.*
- 4, The users may not be able to make complex operations.*

Base on the above, I will design:

- 1, The functions of the program*
- 2, The method of user identification*
- 3, The method of data protection*
- 4, The overall structure of the program*
- 5, The design of interface*

2.2 Intended users and their expectations

The program intended to face the users in different age group, for this, I launched a questionnaire to find out the data types they want to save in the program, in a bid to satisfy their expectations.

As a result, I got the following data types:

1. *habits*
2. *birthday*
3. *address*
4. *nickname*
5. *email address*
6. *identity*

As a common sense, name and phone number is the basic and compulsory data types in a phone book system, so it must exist in the program. However, included the name and phone number, there are 8 data types in total, and makes the program too cumbersome.

To refine the result, I further launched some interviews with the people in different age group, the result is as following:

Data Types	10 -19	20-29	30-39	40-49	50-59
Address	●	●	●	●	
Birthday	●	●	●	●	●
Email Address		●	●	●	●
Habits	●			●	
Identity		●	●		
Nickname	●	●	●	●	●

According to the result of interviews, I removed the data types of habits and identity from the program as they seem relatively less significant in general.

Finally, I got the list of data types to be included in the program through the user expectations. The result is as following:

Name

Phone Number

Birthday

Email Address

Address

Nickname

2.3 Refinement of Problem

The program aims at data management, so it must have several functions, included:

Add new data

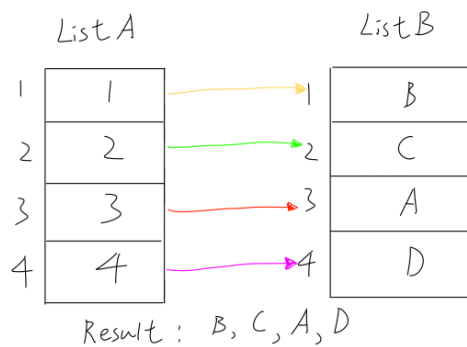
Remove current data

Change current data

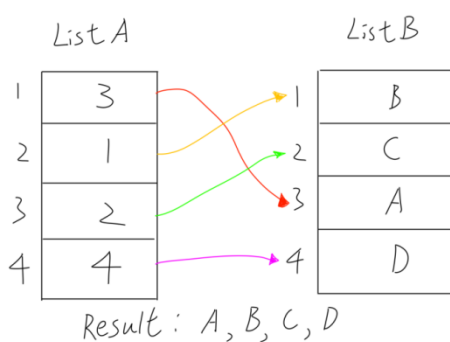
Show current data

Save the data

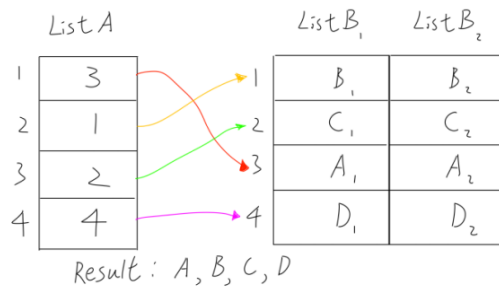
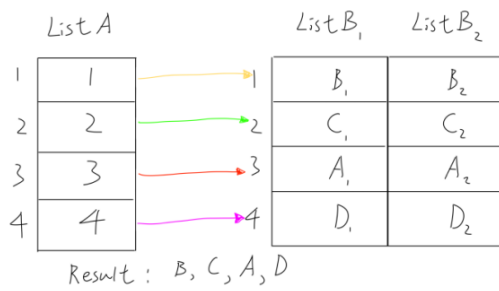
To handle the data, I decided to use an extra list of number to represent the order in the program. The method is as following:



To begin with, I assume that there is a list without order (List B) and a list which save the corresponding numbers of order (List A). To manage the data, we could simply change the number in List A, as following:



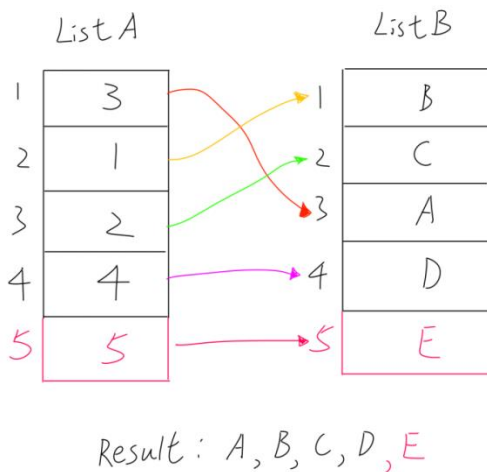
It seems quite meaningless with only one list of data, however, if there are several lists of data, it can improve the processing capacity significantly



In my case, the phone book system has to handle 6 list of data, as mentioned in [ch.2.2](#), named Name, Phone Number, Birthday, Email Address, Address, and Nickname. These data are belonged to the same person in one record, so I decided to use such method in order to improve the processing capacity.

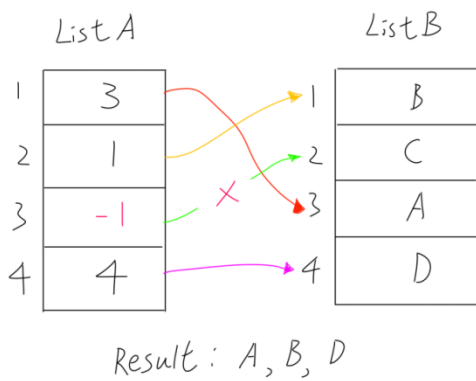
In the following, I would explain how it works in the different functions.

Add new data:



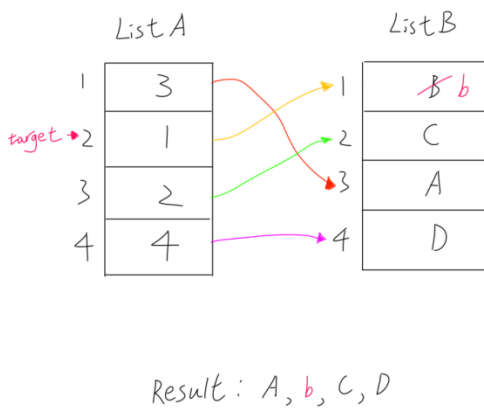
Add the data to the end of both list.

Remove current data:



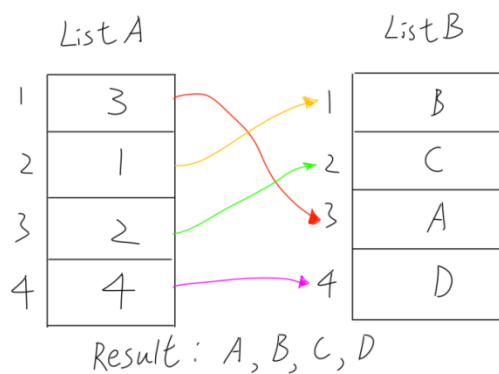
Change the corresponding number of order to -1, so that the program can indicate and not to show it in the result.

Change current data:



Indicate the target through the order of List A, and then change the corresponding result in List B.

Show current data:



Use the number of order in List A to show the record in List B in correct order.

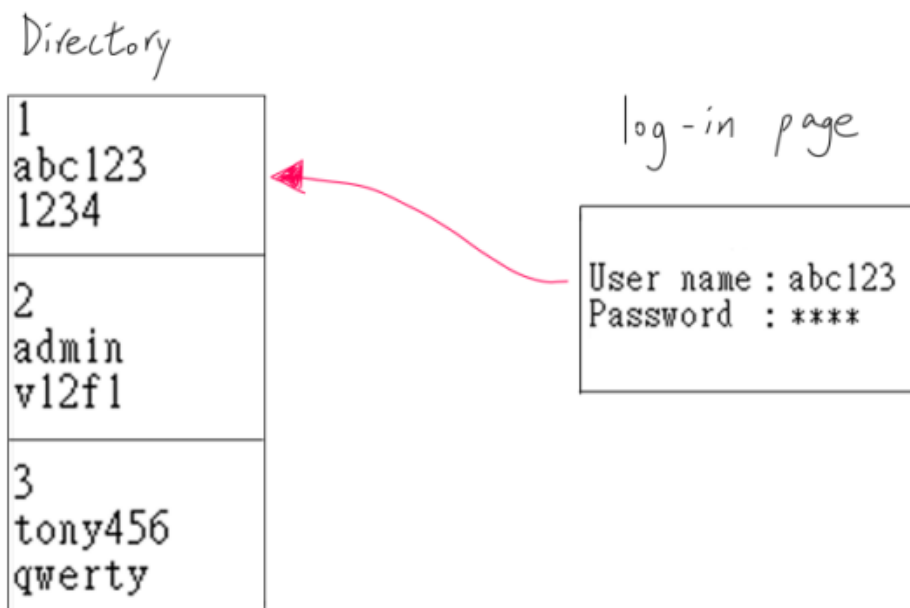
Save the data:

The program will save the data in correct order base on the number of record in List A. In the saving process, the data which marked as -1 in the number of record will not be saved to the file, in order to delete the data and not to show it next time. Through the process of saving data, all the changes will be confirmed.

This method can concentrate the saving process and reduce the extra meaningless saving of data, resulting in the improvement of processing capacity.

2.4 User Identification

As mentioned in [ch.2.1](#), I assume that there are more than one user in the same device. In view of this, the program should have some measures to indicate and separate each user. I decided to use a log-in system to achieve the goal.



The program will ask the user to enter the user name and password, and then check it in the directory. If there exist the corresponding user name and password, the program will continue and read the file which saved the data of that user. In the above example, the file name of user abc123 is "1".

Inspired by the login system in our daily life, I decided to turn the characters of password entered in log-in page into "*", for the sake of keeping the account's data in secret.

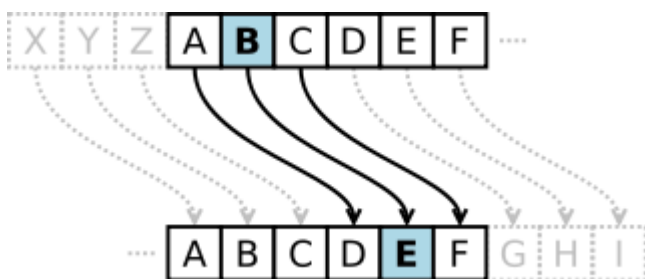
2.5 Data Protection

As mentioned in [ch.2.1](#), I assume that it is dangerous if data leakage occurs. However, for the current design, other people can simply view the data through checking the data file.

In order to protect the data, I decided to encrypt the data in the data file.

Inspired by the Caesar cipher, I designed an encryption system through shifting the position of characters in ASCII table

Code	Character	Code	Character	Code	Character	Code	Character
32	(space)	58	:	84	T	110	n
33	!	59	;	85	U	111	o
34	"	60	<	86	V	112	p
35	#	61	=	87	W	113	q
36	\$	62	>	88	X	114	r
37	%	63	?	89	Y	115	s
38	&	64	@	90	Z	116	t
39	'	65	A	91	[117	u
40	(66	B	92	\	118	v
41)	67	C	93]	119	w
42	*	68	D	94	^	120	x
43	+	69	E	95	_	121	y
44	,	70	F	96	`	122	z
45	-	71	G	97	a	123	{
46	.	72	H	98	b	124	
47	/	73	I	99	c	125	}
48	0	74	J	100	d	126	~
49	1	75	K	101	e		
50	2	76	L	102	f		
51	3	77	M	103	g		
52	4	78	N	104	h		
53	5	79	O	105	i		
54	6	80	P	106	j		
55	7	81	Q	107	k		
56	8	82	R	108	l		
57	9	83	S	109	m		



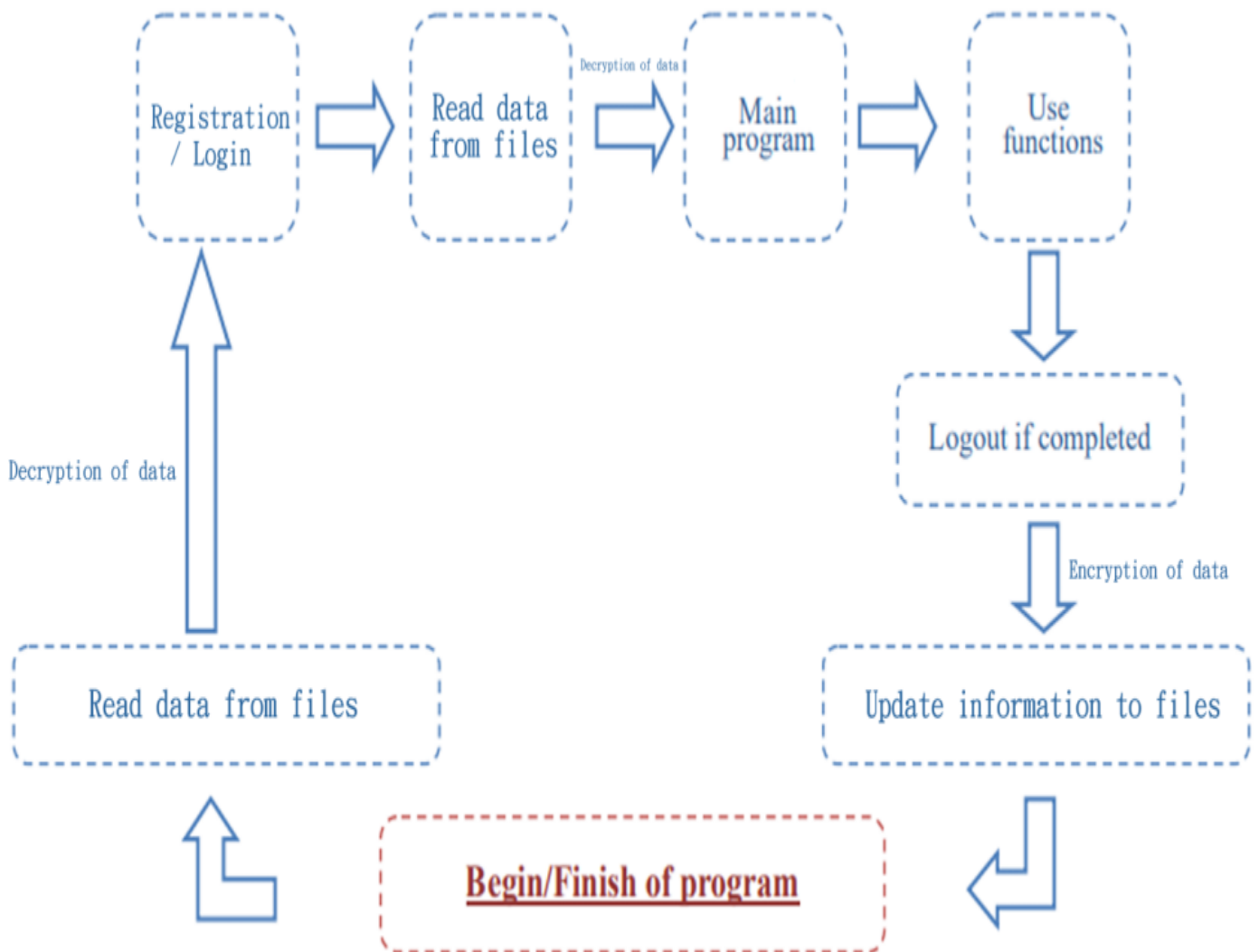
For example, I can encrypt 'abc'(97,98,99) into 'def'(100,101,102) = $[97+3, 98+3, 99+3]$ through shifting the position of 'abc' in ASCII table by 3.

Moreover, to make it more difficult in decryption, I will vary the value of shifting position in a fixed pattern. For example, if the value float from 1 to 3, then 'abc'(97,98,99) will be encrypted as 'bdf'(98,100,102) = $[97+1, 98+2, 99+3]$.

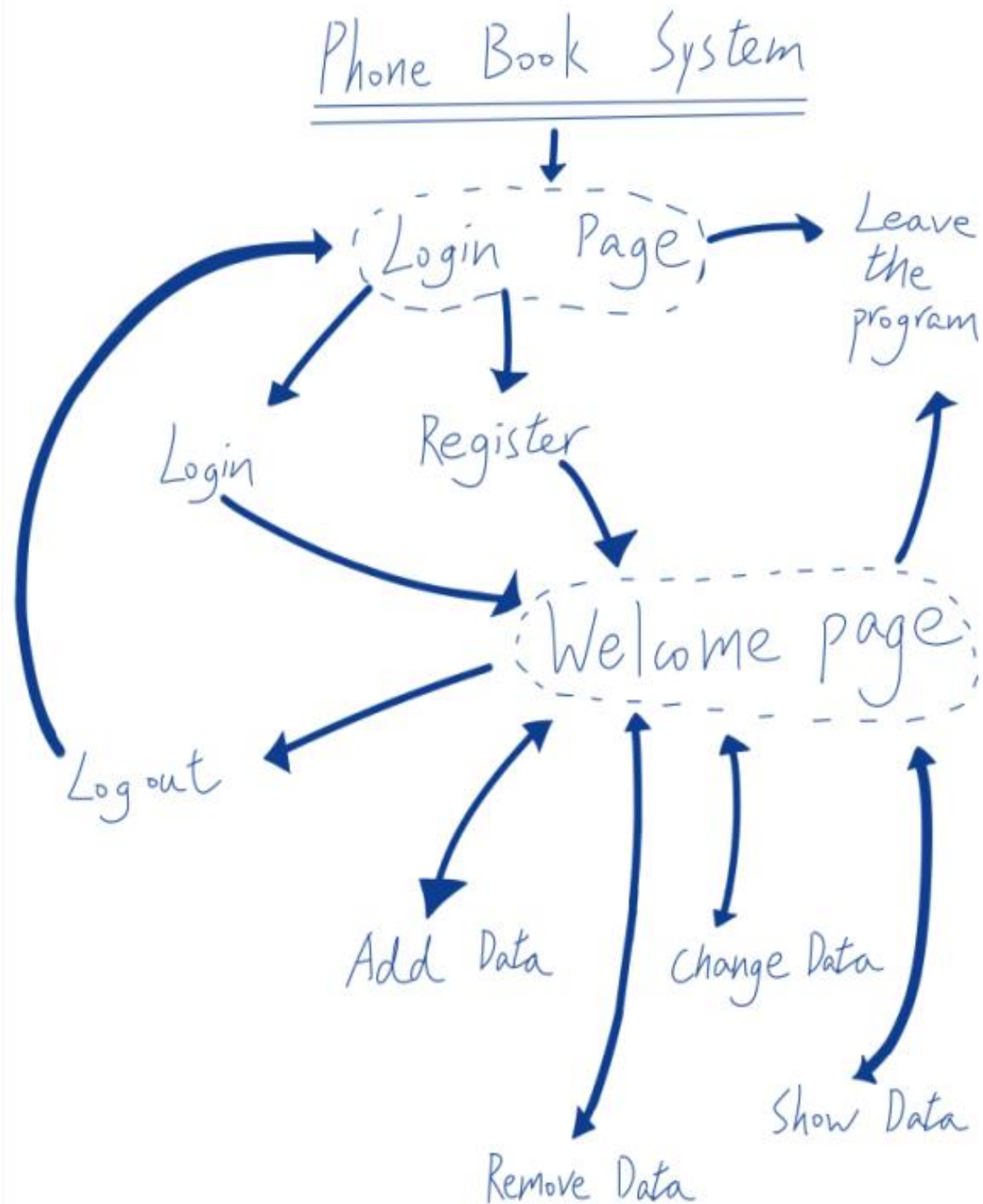
I surmise that this encryption system should be able to protect the data of the normal users.

2.6 Structure of the Program

The flow of the program



Draft design of the main program



2.7 Design of Interface

After a several considerations, I decided to use Dev-pascal for the coding of phone book system. The reason is as following:

- 1, It has approving compatibility in different devices.
- 2, Graphical user interface is not compulsory for Phone book system, and could operate in Text-based user interface, such as Dev-pascal.
- 3, Its code is relatively easy to understand, and I do not have high programming skills, thus Dev-pascal is suitable for my level.
- 4, Dev-pascal's function is satisfying and its UI is friendly for coding.

As Dev-pascal is Text-based user interface, the design of my program's interface must also be Text-based. The following is my design.

Login page

To commence with, a login page will be shown once the program opened.

```
Welcome! Please log in or create new account.  
Press 1 to log in,  
Press 2 to create new account,  
Press 3 to leave the program.
```

If the user pressed "1" , the program would ask user to enter user name and password.

```
Welcome! Please log in or create new account.  
Press 1 to log in,  
Press 2 to create new account,  
Press 3 to leave the program.  
  
-----log in-----  
User name:  
User password:
```

If the user pressed "2" , the program would ask user to register.

```
=====
Add New Account
=====
What is the user name?
What is the password?
Missing data! Please try again.
What is the user name?
```

If the user pressed "3" , the program would ready for shut down.

```
Welcome! Please log in or create new account.  
Press 1 to log in,  
Press 2 to create new account,  
Press 3 to leave the program.  
  
Finished saving. press Enter to leave the program.
```

Main page

If the user logged in or registered, the program would turn into main page.

```
Welcome! admin . What you want to do?
1, Add new data
2, Remove current data
3, Change current data
4, Show current data
5, Log out
6, save and shut down
Please enter the corresponding number.
```

The following is the corresponding function's design of interface.

Add New Data page

```
=====
Add Data
=====
How many data to be added(0 to skip this process)? 1
=====
Record 12
what is the Name?
what is the PhoneNum?
what is the Birthday?
what is the EmailAddress?
what is the LivingArea?
what is the Nickname?
```


Remove Current Data page

```
=====
Delect Data
=====
Record 1
Chen Shao Gu
31440791
1989
k1hwz20cjrj@krsltk.cn
Yau Tsim Mong
Max
=====
Record 2
Chen Xin Yi
31804674
1984
tjdndxvf3wag6@xiudnd.com
Wan Chai
Ryan
=====
Record 3
Lu Jing You
99613903
1987
lw1bw1qlykpna@kosc.net
Kwun Tong
Diego
=====
Record 4
Su Zheng Ru
67712988

Alicia
=====
Record 10
Wu Ven Yun
90826161
1990
gw1zndvwwdo@pbkxv.net
Tai Po
Susana
=====
Record 11
Xie Mao Yi
93039233
1989
x5y03hclms@hivu.com
Wong Tai Sin
Miranda
=====
Record 12
Yang Jian Liang
97839950
1990
cp5p@emkwh.com
Kwai Tsing
Nia
=====
How many data to be delected(0 to skip this process)? 1
=====
process 1
What is the number of the data to be delected?
```

Change Current Data page

```
=====
Change Data
=====
Record 1
Chen Shao Gu
31440791
1989
xlhwz20cjrj@krs1tk.cn
Yau Tsim Mong
Max
=====
Record 2
Chen Xin Yi
31804674
1984
tjdndxvf3wag6@xiudnd.com
Wan Chai
Ryan
=====
Record 3
Lu Jing You
99613903
1987
lw1bw1qlykprna@kosc.net
Kwun Tong
Diego
=====
Record 4
Su Zheng Ru
67712988
```

```
Wong Tai Sin
Miranda
=====
Record 11
Yang Jian Liang
97839950
1990
cp5p@emkwh.com
Kwai Tsing
Nia
=====
How many data to be changed?(0 to skip this process) 1
=====
process 1
What is the number of the data to be changed? 1
The data of the target is,
=====
Chen Shao Gu
31440791
1989
xlhwz20cjrj@krs1tk.cn
Yau Tsim Mong
Max
=====
What is the changed Name(press Enter directly to skip)?
What is the changed PhoneNum(press Enter directly to skip)?
What is the changed Birthday(press Enter directly to skip)?
What is the changed EmailAddress(press Enter directly to skip)?
What is the changed LivingArea(press Enter directly to skip)?
What is the changed Nickname(press Enter directly to skip)?
```

Show Current Data page

```
=====
List of the current Data
=====
Record 1
Chen Shao Gu
31440791
1989
klhwz20cjrj@krsltk.cn
Yau Tsim Mong
Max
=====
Record 2
Chen Xin Yi
31804674
1984
tjdndxvf3wag6@xiudnd.com
Wan Chai
Ryan
=====
Record 3
Lu Jing You
99613903
1987
lwxlbwqliykpna@kosc.net
Kwun Tong
Diego
=====
Record 4
Su Zheng Ru
67712988
dgyattn@dexr.net
Tsuen Wan
Alicia
=====
Record 9
Wu Wen Yun
90826161
1990
gwlzndwvwo@pbkxv.net
Tai Po
Susana
=====
Record 10
Xie Mao Yi
93039233
1989
x5y03hc1ms@hivu.com
Wong Tai Sin
Miranda
=====
Record 11
Yang Jian Liang
97839950
1990
cp5p@emkwh.com
Kwai Tsing
Nia
=====
Finished DisplayData, press Enter to continue
```

Chapter 3 Implementation

3.1 Brief Description

To implement the system, the Dev-Pascal has been chosen as mentioned in [ch.2.7](#). After a long period of hard work, the sample program has been produced. In the following, I would introduce the program according to my design in [ch.1](#) and [ch.2](#).

This chapter will include:

- 1, The data structure of the program.*
- 2, The procedures of the program.*
- 3, The overall structure of the program in view of program code.*

3.2 Data Structure

```
Program PhoneBookSystem;
Uses crt, Dos;
type
  AccountType = record
    Uid : string;
    Uname : string;
    Upw : string
  end;
  RecordType = record
    Name : string;
    PhoneNum : string;
    Birthday : string;
    EmailAddress : string;
    LivingArea : string;
    Nickname : string
  end;
  ArrayType = array[1..100] of integer;
var
  PersonalData : array[1..100] of RecordType;
  currentuser : AccountType;
  userlist : array[1..50] of AccountType;
  ListOrder : arraytype;
  Procedure_templist : arraytype;
  countmax, accountmaxnum, i : integer;
  selection : char;
  target_text : string;
  ifleave, iflogout, iftype, endprogram : boolean;

function FileExist (name : string) : boolean;
var f : file; a : word;
begin
  assign (f, name);
  GetFAttr (f, a);
  FileExist := false;
  if DosError = 0 then if ((a and Directory) = 0) and ((a and VolumeId) = 0) then FileExist := true;
end;
```

Records

I have made use of the characteristic of 'record' in pascal since it helps me hold a set of data which belongs to the same class. As mentioned in [ch.2.2](#), there are 6 types of data already, together with the data during the operation of program, it must be a huge amount of data in total. The characteristic of 'record' helps me classify the data and thus easier to find them.

AccountType = record

```
    Uid : string;  
    Uname : string;  
    Upw : string  
end;
```

AccountType is the type of data which save the data relevant to the login system, included user id, user name and user password.

The Uid(user id) not only identify the user, but also represent the file name of the .txt file which saved the users' data in hard disk .

The Uname(user name) and Upw(User password) is the data sustaining the login system, as well as the data which has to be memorized by user.

In the program, currentuser and userlist are the variables with AccountType.

RecordType = record

```
    Name : string;  
    PhoneNum : string;  
    Birthday : string;  
    EmailAddress : string;  
    LivingArea : string;  
    Nickname : string  
end;
```

RecordType is the type of data which save the user's data, included Name, Phone Number, Birthday, Email, Address, Address and Nickname. In other words, it is the "List B" in [ch.2.3](#).

All the data will be saved as string to make it easier in processing.

In the program, PersonalData is the only variable with RecordType.

ArrayType = array[1..100] of integer;

ArrayType is the type of data which save the number of order, in other words, it is the "List A" in [ch.2.3](#).

The number of order will be saved as integer.

In the program, ListOrder and Procedure_templist are the variables with ArrayType.

Global Variables

Here are the global variables used in the program so that the processes of calling in or out variables and procedures will not affect the vital information.

```
countmax, accountmaxnum, i : integer;  
selection : char;  
target_text : string;  
ifleave, iflogout, iftype, endprogram : boolean;
```

countmax, accountmaxnum and i are integer. They supporting lots of logical operations of the program. In general, countmax saved the number of records of the current user, accountmaxnum saved the maximum number of users' account and mainly supporting the login system, and i is a pan use variable for counting.

Selection is character. It saved the selection of function from the user. The program will provide different function in accordance to this variable.

target_text is string. It saved the target text messages to be handled.

ifleave, iflogout, iftype and endprogram are boolean. They are the flag of the program and help the program to deter which stage it is.

Function

The program has one function named FileExist. It was used to deter whether the .txt file exist, and avoid run-time error when launching the program without directory.

3.3 Procedures in the Program

Booleans, characters and other types of variables can be found in the procedures.

According to the functions and design illustrated in [Chapter 2](#), the following procedures will be constructed in the program:

**The instruction statements may only be the major parts of the procedures*

Reading data from .txt file

There are 2 procedures responsible for reading data, named ReadDirectory and Readfile.

ReadDirectory

```
Procedure ReadDirectory;
var
  i : integer;
  f : text;
begin
  i := 0;
  if FileExist('directory.txt')
  then begin
    assign(f, 'directory.txt');
    reset(f);
    while not eof(f) do
    begin
      i := i + 1 ;
      readln(f, userlist[i].Uid );
      readln(f, userlist[i].Uname);
      readln(f, userlist[i].Upw);
    end;
    close(f);
    accountmaxnum := i
  end
  else begin
    assign(f, 'directory.txt');
    rewrite(f);
    close(f);
    accountmaxnum := i
  end;
end;
```

This procedure is responsible for reading data from directory.txt, which saving the user id, user name and user password. The procedure will first make use of function FileExist to detect whether the file "directory.txt" exist. In general, the file "directory.txt" may not exist if it is the first time the program being launched, and the procedure will create a blank "directory.txt" file. Otherwise, the procedure will read the existing "directory.txt" file, and save the data to variable "userlist". The variable accountmaxnum will save the number of account read from "directory.txt" file.

Readfile

```
Procedure Readfile(filename : string);
var
  i, j : integer;
  f : text;
begin
  assign(f, filename + '.txt');
  reset(f);
  i := 0;
  while not eof(f) do
    begin
      i := i + 1 ;
      readln(f, PersonalData[i].Name);
      readln(f, PersonalData[i].PhoneNum);
      readln(f, PersonalData[i].Birthday);
      readln(f, PersonalData[i].EmailAddress);
      readln(f, PersonalData[i].LivingArea);
      readln(f, PersonalData[i].Nickname);
    end;
  close(f);
  countmax := i;
  for j := 1 to countmax do ListOrder[j] := j;
end;
```

This procedure is responsible for reading data from the .txt file with the specified file name when calling the procedure. All the data will be saved to variable PersonalData, and variable countmax will save the total number of records. After that, the procedure will initialise the number of order in variable ListOrder.

Writing data into .txt file

There are 4 procedures responsible for writing data, named Writefile, Uppercase, Savefile and SaveDirectory.

Writefile and Uppercase

```
Procedure Uppercase;
var i, maxnum : integer;
begin
    maxnum := length(target_text);
    if ord(target_text[1]) in [97..122] then target_text[1] := chr(ord(target_text[1])-32);
    for i := 2 to maxnum-1 do
        if target_text[i] = ' ' then
            if ord(target_text[i+1]) in [97..122] then target_text[i+1] := chr(ord(target_text[i+1])-32);
    end;
    {-----uppercase-----}

Procedure Writefile(filename : string);
var
    i : integer;
    f : text;
begin
    assign(f, filename + '.txt');
    rewrite(f);
    for i := 1 to countmax do if ListOrder[i] > -1 then
        with PersonalData[ListOrder[i]] do
            begin
                target_text := Name;
                Uppercase;
                Name := target_text;
                writeln(f, Name);

                writeln(f, PhoneNum);
                writeln(f, Birthday);
                writeln(f, EmailAddress);

                target_text := LivingArea;
                Uppercase;
                LivingArea := target_text;
                writeln(f, LivingArea);

                target_text := Nickname;
                Uppercase;
                Nickname := target_text;
                writeln(f, Nickname);
            end;
        end;
    close(f);
end;
```

These two procedures will run simultaneously. They are responsible for writing the user data saved in variable PersonalData into the .txt file in hard disk in accordance to the number of order saved in variable ListOrder. During the process, these procedures will formalise the data of name, living area and nickname, for example, change "phone book system" to "Phone Book System". Same as the design in [ch.2.3](#), the data with the number of order -1 will not be wrote into the .txt file for the sake of deleting the data.

Savefile

```
Procedure Savefile(filename : string);
var
  i : integer;
  f : text;
begin
  assign(f, filename + '.txt');
  rewrite(f);
  for i := 1 to countmax do if ListOrder[i] > -1 then
  with PersonalData[ListOrder[i]] do
  begin
    writeln(f, Name);
    writeln(f, PhoneNum);
    writeln(f, Birthday);
    writeln(f, EmailAddress);
    writeln(f, LivingArea);
    writeln(f, Nickname);
  end;
  close(f)
end;
```

This procedure is the same as Writefile. The only difference is that this procedure will not formalise the data.

SaveDirectory

```
Procedure SaveDirectory;
var i : integer; f : text;
begin
  assign(f, 'directory.txt');
  rewrite(f);
  for i := 1 to accountmaxnum do
  begin
    writeln(f, userlist[i].Uid );
    writeln(f, userlist[i].Uname);
    writeln(f, userlist[i].Upw)
  end;
  close(f);
end;
```

This procedure is responsible for writing data into directory.txt. It will save the user id, user name and user password in the variable userlist to the "directory.txt" file.

Encryption and Decryption

There are 4 procedures responsible for the Encryption and Decryption of the data. They mainly service two types of data, which are the account data and the users' data. These 4 procedures will launch after reading data from .txt file and before writing data to .txt file, in order to ensure the data in the .txt file is unreadable and meaningless for human.

Encryption_data

```
Procedure Encryption_data;
var i, j, k : integer;
begin
    k := 0;
    for i := 1 to countmax do with PersonalData[i] do
        begin
            k := k mod 5 + 1;
            for j := 1 to length(Name) do Name[j] := chr(ord(Name[j]) - k);
            for j := 1 to length(PhoneNum) do PhoneNum[j] := chr(ord(PhoneNum[j]) - k);
            for j := 1 to length(Birthday) do Birthday[j] := chr(ord(Birthday[j]) - k);
            for j := 1 to length(EmailAddress) do EmailAddress[j] := chr(ord(EmailAddress[j]) - k);
            for j := 1 to length(LivingArea) do LivingArea[j] := chr(ord(LivingArea[j]) - k);
            for j := 1 to length(Nickname) do Nickname[j] := chr(ord(Nickname[j]) - k);
        end
    end;
end;
```

As mentioned in [ch.2.5](#), I will use Caesar Cipher to encrypt the data through ASCII table. This procedure will shift each character in each user record by the variable k. However, the value will float from 1 to 5. This is achieved by using "mod". Through the command $k := k \bmod 5 + 1$, the value of k will change in each loop body, and float from 1 to 5. Thus, the data can be encrypted in a complicated way and protect the data from leakage, and achieve the goal in [chapter 1](#).

Encryption_directory

```
Procedure Encryption_directory;
var i, j, k : integer;
begin
    k := 0;
    for i := 1 to accountmaxnum do with Userlist[i] do
        begin
            k := k mod 5 + 1;
            for j := 1 to length(Uid) do Uid[j] := chr(ord(Uid[j]) - k);
            for j := 1 to length(Username) do Username[j] := chr(ord(Username[j]) - k);
            for j := 1 to length(Upw) do Upw[j] := chr(ord(Upw[j]) - k);
        end
    end;
end;
```

It is the same as procedure Encryption_data, however, its target is the data from the file "directory.txt".

Decryption_data

```
Procedure Decryption_data;
var i, j, k : integer;
begin
    k := 0;
    for i := 1 to countmax do with PersonalData[i] do
        begin
            k := k mod 5 + 1;
            for j := 1 to length(Name) do Name[j] := chr(ord(Name[j])+k);
            for j := 1 to length(PhoneNum) do PhoneNum[j] := chr(ord(PhoneNum[j])+k);
            for j := 1 to length(Birthday) do Birthday[j] := chr(ord(Birthday[j])+k);
            for j := 1 to length(EmailAddress) do EmailAddress[j] := chr(ord(EmailAddress[j])+k);
            for j := 1 to length(LivingArea) do LivingArea[j] := chr(ord(LivingArea[j])+k);
            for j := 1 to length(Nickname) do Nickname[j] := chr(ord(Nickname[j])+k);
        end
    end;
end;
```

The logic of decryption is the same as encryption. As the process of encryption subtracted the ASCII code of each character by the value of k, we could recover the data through adding the value of k to each character.

Decryption_directory

```
Procedure Decryption_directory;
var i, j, k : integer;
begin
    k := 0;
    for i := 1 to accountmaxnum do with Userlist[i] do
        begin
            k := k mod 5 + 1;
            for j := 1 to length(Uid) do Uid[j] := chr(ord(Uid[j])+k);
            for j := 1 to length(Username) do Username[j] := chr(ord(Username[j])+k);
            for j := 1 to length(Upw) do Upw[j] := chr(ord(Upw[j])+k);
        end
    end;
end;
```

It is the same as procedure Decryption_data, however, its target is the data from the file "directory.txt".

Initialize

The program will initialize the data to avoid error when changing the user account, procedure ClearRam was used.

```
Procedure ClearRam;
var i : integer;
begin
  for i := 1 to countmax do
    begin
      PersonalData[i].Name := '';
      PersonalData[i].PhoneNum := '';
      PersonalData[i].Birthday := '';
      PersonalData[i].EmailAddress := '';
      PersonalData[i].LivingArea := '';
      PersonalData[i].Nickname := '';
    end;
  countmax := 0
end;
```

This procedure will initialize the record in variable PersonalData into empty.

Sorting

I have chosen merge sort to tidy up the order of data because of its high efficiency. As I used an extra list to represent the order, the sorting need only change the number of order saved in variable ListOrder. Variable Procedure_templist was used to save the temporary data of the number of order. There are 3 procedures related to sorting, named Merge, Mergesort and Check_if_need_sort.

Check_if_need_sort

```
Procedure Check_if_need_sort;
var flag : boolean;
begin
  i := 0;
  flag := false;
  while not flag and (i < countmax) do
    begin
      i := i + 1;
      if PersonalData[i].Name > PersonalData[i+1].Name then flag := True;
    end;
  if flag then Mergesort(1, countmax)
end;
```

This procedure will check whether the data is in ascending order through the order of name of each record. If the current name of the record is larger than the value of the name in next record, it is not in ascending order and need sorting. Variable flag was used to indicate the result. If variable flag is true, then this procedure will call the procedure Mergesort to launch the sorting process.

Merge and Mergesort

```
Procedure Merge(left, middle, right : integer);
var
    pointLow, pointHigh, pointTemp, highlength, count : integer;
begin
    pointLow := left;
    pointHigh := middle + 1;
    pointTemp := left;
    repeat
        if PersonalData[ListOrder[pointLow]].Name <= PersonalData[ListOrder[pointHigh]].Name
        then begin
            Procedure_templist[pointTemp] := ListOrder[pointLow];
            pointLow := pointLow + 1;
        end
        else begin
            Procedure_templist[pointTemp] := ListOrder[pointHigh];
            pointHigh := pointHigh + 1;
        end;
        pointTemp := pointTemp + 1;
    until (pointLow > middle) or (pointHigh > right);
    highlength := right - middle;
    if pointLow > middle
    then for count := pointHigh to right do Procedure_templist[count] := ListOrder[count]
    else for count := pointLow to middle do Procedure_templist[count + highlength] := ListOrder[count];
    for count := left to right do ListOrder[count] := Procedure_templist[count]
end;

Procedure Mergesort(left, right : integer);
var
    middle : integer;
begin
    if left < right
    then begin
        middle := (left + right) div 2;
        MergeSort(left, middle);
        MergeSort(middle + 1, right);
        Merge(left, middle, right)
    end
end;
```

These two procedures were responsible for conducting merge sort.

Login and Register

There are 2 procedures responsible for Login and Register respectively.

CreateAccount

```
procedure CreateAccount;
var f : text; inputchar : char; check : boolean;
begin
  ClrScr;
  writeln('=====');
  writeln('Add New Account');
  writeln('=====');
  accountmaxnum := accountmaxnum + 1;
  repeat
    write('What is the user name? ');
    readln(Userlist[accountmaxnum].Uname);
    write('What is the password? ');
    Userlist[accountmaxnum].Upw := '';
    repeat
      inputchar := Readkey;
      case inputchar of
        #8 : if length(Userlist[accountmaxnum].Upw) > 0 then
            begin
              Userlist[accountmaxnum].Upw := copy(Userlist[accountmaxnum].Upw, 1, length(Userlist[accountmaxnum].Upw)-1);
              GotoXY(WhereX-1,WhereY);
              write(' ');
              GotoXY(WhereX-1,WhereY)
            end;
        #13 : ;
        else begin
              Userlist[accountmaxnum].Upw := Userlist[accountmaxnum].Upw + inputchar;
              write('*')
            end
      end
    until inputchar = #13;
    if (Userlist[accountmaxnum].Uname <> '') and (Userlist[accountmaxnum].Upw <> '') then check := true;
    if not check then begin writeln;writeln('Missing data! Please try again.') end
  until check;
  str(accountmaxnum, userlist[accountmaxnum].Uid);
  assign(f, userlist[accountmaxnum].Uid+'.txt');
  rewrite(f);
  close(f);
  writeln;
  currentuser.Uname := Userlist[accountmaxnum].Uname;
  currentuser.Upw := Userlist[accountmaxnum].Upw;
  currentuser.Uid := Userlist[accountmaxnum].Uid;
  writeln('Welcome! ',Userlist[accountmaxnum].Uname,' ,press Enter to continue. ');
  readln
end;
```

This procedure will ask the new user to enter user name and password for registration. As mentioned in [ch.2.4](#), the program has to change the character of password entered by user to "*" in order to ensure the security of user's account. To realise it, the procedure will detect the password entered by user character-by-character, and record then into variable Userlist.Upw. All the input from keyboard will be turned into "*", apart from Backspace and Enter key. If the user pressed Backspace, the procedure will delete one character from the end of the variable Userlist.Upw and show it on the screen immediately. Once the user pressed Enter, the input process will end and confirm the user name and password. After that, the procedure will create a new blank file for the new user, and directly enter the main page of the program.

Login

```
Procedure Login;
var i : integer; flag : boolean; inputchar : char;
begin
  repeat
    Clrscr;
    writeln('Welcome! Please log in or create new account. ');
    writeln('Press 1 to log in, ');
    writeln('Press 2 to create new account, ');
    writeln('Press 3 to leave the program. ');
    selection := Readkey;
    case selection of
      '1' : begin {log in}
              writeln;
              writeln('-----log in-----');
              write('User name: ');
              readln(currentuser.Uname);
              write('User password: ');
              currentuser.Upw := '';
              repeat
                inputchar := Readkey;
                case inputchar of
                  #8 : if length(currentuser.Upw) > 0 then
                        begin
                          currentuser.Upw := copy(currentuser.Upw, 1, length(currentuser.Upw)-1);
                          GotoXY(WhereX-1,WhereY);
                          write(' ');
                          GotoXY(WhereX-1,WhereY)
                        end;
                  #13 : ;
                  else begin
                          currentuser.Upw := currentuser.Upw + inputchar;
                          write('*')
                        end
                end
              until inputchar = #13;
            {search}

            {search}
            i := 0;
            flag := false;
            while (not flag) and (i <= accountmaxnum) do
              begin
                i := i + 1;
                if ( currentuser.Uname = userlist[i].Uname) and ( currentuser.Uname <> '')
                  and ( currentuser.Upw = userlist[i].Upw) and ( currentuser.Upw <> '') then
                  begin
                    currentuser.Uid := userlist[i].Uid;
                    flag := true
                  end
                end;
              if flag then begin writeln; write('Welcome back! ',currentuser.Uname,' . press Enter to enter the system.') end
              else begin writeln;write('Wrong user name or password, please try again.') end;
              readln
            end;
            '2' : begin CreateAccount;flag := true end;
            '3' : begin ifleave := true; iftype := true; endprogram := true end
            else writeln('Invalid value. Try again.')
            end
          until (flag) or (endprogram)
        end;
end;
```

This procedure will ask the user to select the function at first.

If the user pressed "1", then the program will ask the user to enter the user name and password. The input process of the password is the same as procedure CreateAccount in above. After that, the procedure will compare the login data from user with the account data loaded from "directory.txt". If it is valid, then the program will continue, else the user has to enter the data again.

If the user pressed "2", then the program will call the procedure CreateAccount for registration, and set the flag into true in order to let the program continue.

If the user pressed "3", then the program will set the flags into true and ready for shut down.

Main Function of the Program

There are 4 procedures responsible for the main functions of the program, named DisplayData, AddData, DelectData and ChangeData. There logic was explained in ch.2.3 and the following procedures are the same as those logic graphs.

DisplayData

```
procedure DisplayData;
var count : integer;
begin
  writeln('=====');
  for count := 1 to countmax do if ListOrder[count] <> -1 then
    begin
      writeln('Record ',count);
      with PersonalData[ListOrder[count]] do
        begin
          writeln(Name);
          writeln(PhoneNum);
          writeln(Birthday);
          writeln(EmailAddress);
          writeln(LivingArea);
          writeln(Nickname);
        end;
      writeln('=====')
    end
  end;
end;
```

This procedure could show the data base on the number of order saved in variable ListOrder.

AddData

```
Procedure AddData;
var numofdata, i : integer;
begin
  ClrScr;
  writeln('=====');
  writeln('Add Data');
  writeln('=====');
  write('How many data to be added(0 to skip this process)? ');
  readln(numofdata);
  if numofdata > 0 then for i := 1 to numofdata do
  begin
    countmax := countmax + 1;
    writeln('=====');
    writeln('Record ',countmax);
    write('what is the Name? ');
    readln(target_text);
    Uppercase;
    PersonalData[countmax].Name := target_text;
    write('what is the PhoneNum? ');
    readln(PersonalData[countmax].PhoneNum);
    write('what is the Birthday? ');
    readln(PersonalData[countmax].Birthday);
    write('what is the EmailAddress? ');
    readln(PersonalData[countmax].EmailAddress);
    write('what is the LivingArea? ');
    readln(target_text);
    Uppercase;
    PersonalData[countmax].LivingArea := target_text;
    write('what is the Nickname? ');
    readln(target_text);
    Uppercase;
    PersonalData[countmax].Nickname := target_text;
    ListOrder[countmax] := countmax;
  end;
  if numofdata > 0 then Mergesort(1, countmax);
  ClrScr;
  writeln('The current list ater AddData: ');
  DisplayData;
  write('Finished AddData, press Enter to continue');
  readln;
  ClrScr;
end;
```

This procedure will ask the user to enter the data and add it to the end of the variable PersonalData, and consequently formalise the Name, LivingArea and Nickname. Then the procedure will sort the data and show the result.

DelectData

```
Procedure DelectData;
var numofdata, i, target : integer;
begin
  ClrScr;
  writeln('=====');
  writeln('Delect Data');
  DisplayData;
  write('How many data to be delected(0 to skip this process)? ');
  readln(numofdata);
  if numofdata > 0 then for i := 1 to numofdata do
    begin
      writeln('=====');
      writeln('process ',i);
      write('What is the number of the data to be delected? ');
      readln(target);
      write('The data of ', PersonalData[ListOrder[target]].Name, ' has been delected, press Enter to the next process ');
      ListOrder[target] := -1 ;
      readln;
    end;
  WriteFile(currentuser.Uid);
  ReadFile(currentuser.Uid);
  Encryption_data;
  SaveFile(currentuser.Uid);
  Decryption_data;
  ClrScr;
  writeln('The current list after DelectData: ');
  DisplayData;
  write('Finished DelectData, press Enter to continue');
  readln;
  ClrScr;
end;
```

This procedure will ask the user to choose which data and how many data to be deleted, and then delete those data and update the current data to the .txt file to confirm the changes. After that, the program will show the result.

ChangeData

```

Procedure ChangeData;
var numofdata, i, target : integer; name_temp, input_temp : string;
begin
  ClrScr;
  writeln('=====');
  writeln('Change Data');
  DisplayData;
  write('How many data to be changed?(0 to skip this process) ');
  readln(numofdata);
  if numofdata > 0 then for i := 1 to numofdata do
  begin
    writeln('=====');
    writeln('process ',i);
    write('What is the number of the data to be changed? ');
    readln(target);
    writeln('The data of the target is, ');
    writeln('-----');
    with PersonalData[ListOrder[target]] do
      begin
        writeln(Name);
        writeln(PhoneNum);
        writeln(Birthday);
        writeln(EmailAddress);
        writeln(LivingArea);
        writeln(Nickname);
      end;

    writeln('-----');
    write('What is the changed Name(press Enter directly to skip)? ');
    name_temp := PersonalData[ListOrder[target]].Name;
    readln(input_temp);
    if input_temp <> '' then
      begin
        target_text := input_temp;
        Uppercase;
        input_temp := target_text;
        PersonalData[ListOrder[target]].Name := input_temp;
      end;

    write('What is the changed PhoneNum(press Enter directly to skip)? ');
    readln(input_temp);
    if input_temp <> '' then PersonalData[ListOrder[target]].PhoneNum := input_temp;
    write('What is the changed Birthday(press Enter directly to skip)? ');
    readln(input_temp);
    if input_temp <> '' then PersonalData[ListOrder[target]].Birthday := input_temp;
    write('What is the changed EmailAddress(press Enter directly to skip)? ');
    readln(input_temp);
    if input_temp <> '' then PersonalData[ListOrder[target]].EmailAddress := input_temp;

    write('What is the changed LivingArea(press Enter directly to skip)? ');
    readln(input_temp);
    if input_temp <> '' then
      begin
        target_text := input_temp;
        Uppercase;
        input_temp := target_text;
        PersonalData[ListOrder[target]].LivingArea := input_temp;
      end;

    write('What is the changed Nickname(press Enter directly to skip)? ');
    readln(input_temp);
    if input_temp <> '' then
      begin
        target_text := input_temp;
        Uppercase;
        input_temp := target_text;
        PersonalData[ListOrder[target]].Nickname := input_temp;
      end;
    ClrScr;
    writeln('The current list after ChangeData: ');
    DisplayData;
    write('Finished.The data of ', name_temp, ' has been changed to ',
    PersonalData[ListOrder[target]].Name, ' , press Enter to the next process ');
    readln;
  end;

  if numofdata > 0 then Mergesort(1, countmax);
  ClrScr;
  writeln('The current list after ChangeData: ');
  DisplayData;
  write('Finished ChangeData, press Enter to continue');
  readln;
  ClrScr;
end;

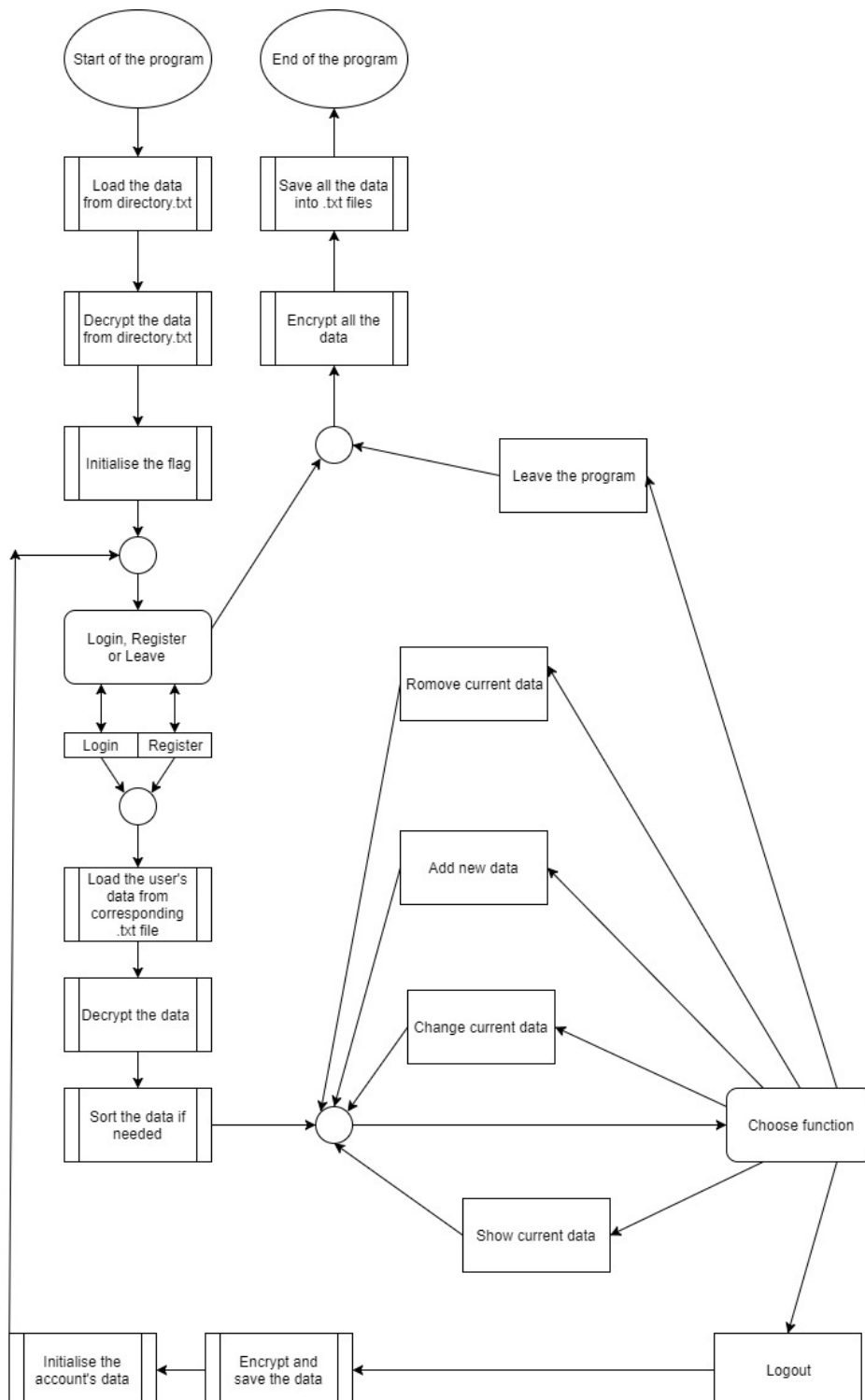
```

This procedure will ask the user to choose how many data to be changed, then ask which data to be changed and show the target's data. After that, the procedure will ask the user to enter the data and use it to cover the current data if the input message is not empty. Finally, the procedure will show the result.

3.4 Main Body of the Program

Base on the design in [ch.2.6](#), I finished the program's main body. In the following, I will introduce the main body of this program.

Flowcharts of the program



The code of program main body

```
begin
  ReadDirectory;
  Decryption_directory;
  ifleave := false;
  endprogram := false;
repeat
  Login;
  if not endprogram then
    begin
      ReadFile(currentuser.Uid);
      Decryption_data;
      Check_if_need_sort;
      ClrScr;
      iflogout := false;
      repeat
        writeln('Welcome! ', currentuser.Uname, ' . What you want to do? ');
        writeln('1, Add new data');
        writeln('2, Romove current data');
        writeln('3, Change current data');
        writeln('4, Show current data');
        writeln('5, Log out');
        writeln('6, save and shut down');
        write('Please enter the corresponding number. ');
        iftype := false;
        repeat
          selection := Readkey;
          case selection of
            '1' : begin AddData; iftype := true end;
            '2' : begin DelectData; iftype := true end;
            '3' : begin ChangeData; iftype := true end;
            '4' : begin
                  ClrScr;
                  writeln('=====');
                  writeln('List of the current Data');
                  DisplayData;
                  iftype := true;
                  write('Finished DisplayData, press Enter to continue');
```

```

        readln;
        ClrScr;
    end;
'5' : begin
    currentuser.Uname := "";
    currentuser.Upw := "";
    iftype := true;
    iflogout := true;
    writeln;
    Encryption_data;
    SaveFile(currentuser.Uid);
    ClearRam;
    write('press Enter to return the start page. See you next time!');
    readln
    end;
'6' : begin ifleave := true; iftype := true end
else writeln('Invalid input. Try again.')
end;

until iftype
until (iflogout) or (ifleave)
end
until ifleave;
writeln;
Encryption_directory;
SaveDirectory;
if not endprogram then
begin
    Encryption_data;
    SaveFile(currentuser.Uid)
end;
write('Finished saving. press Enter to leave the program. ');
readln
End.

```

In general, neglected some minor details, the main body's flow is the same as the flowchart above.

Chapter 4 Testing & Evaluation

4.1 Brief Description

In this chapter, a set of tests has been done to find out the bugs in the program, as well as checking whether the program can achieve its purposes in [Chapter 1](#). Debug and further improvements of the program will base on the testing results.

In the following, I will talk about:

- 1, The Testing and Evaluation Plan*
- 2, The Internal Tests and Corresponding Results*
- 3, The External Tests and Corresponding Results*
- 4, The Evaluation upon the Program*

4.2 Testing and Evaluation Plan

Generally speaking, there are three types of errors that may occur in the program, including syntax, run-time and logic errors.

Syntax error:

There exist lots of Syntax errors occurred during the implementation stage. After a long time of revise, all the syntax errors fixed and the program compiled successfully. In the current stage, I surmise that there should be no syntax error in the program as it is executable.

Run-time error:

Even though the program is executable, there might exist several logical errors and make the program cannot operate normally as same as the design. Some tests should be launched to detect those errors, together with further improvements of the program.

Logic error:

Similar to the run-time error, the logic of the program might go wrong, and makes the results of the program different from expectations. Some tests should be launched by comparing the results with the expectations, in order to ensure that there is no logic error.

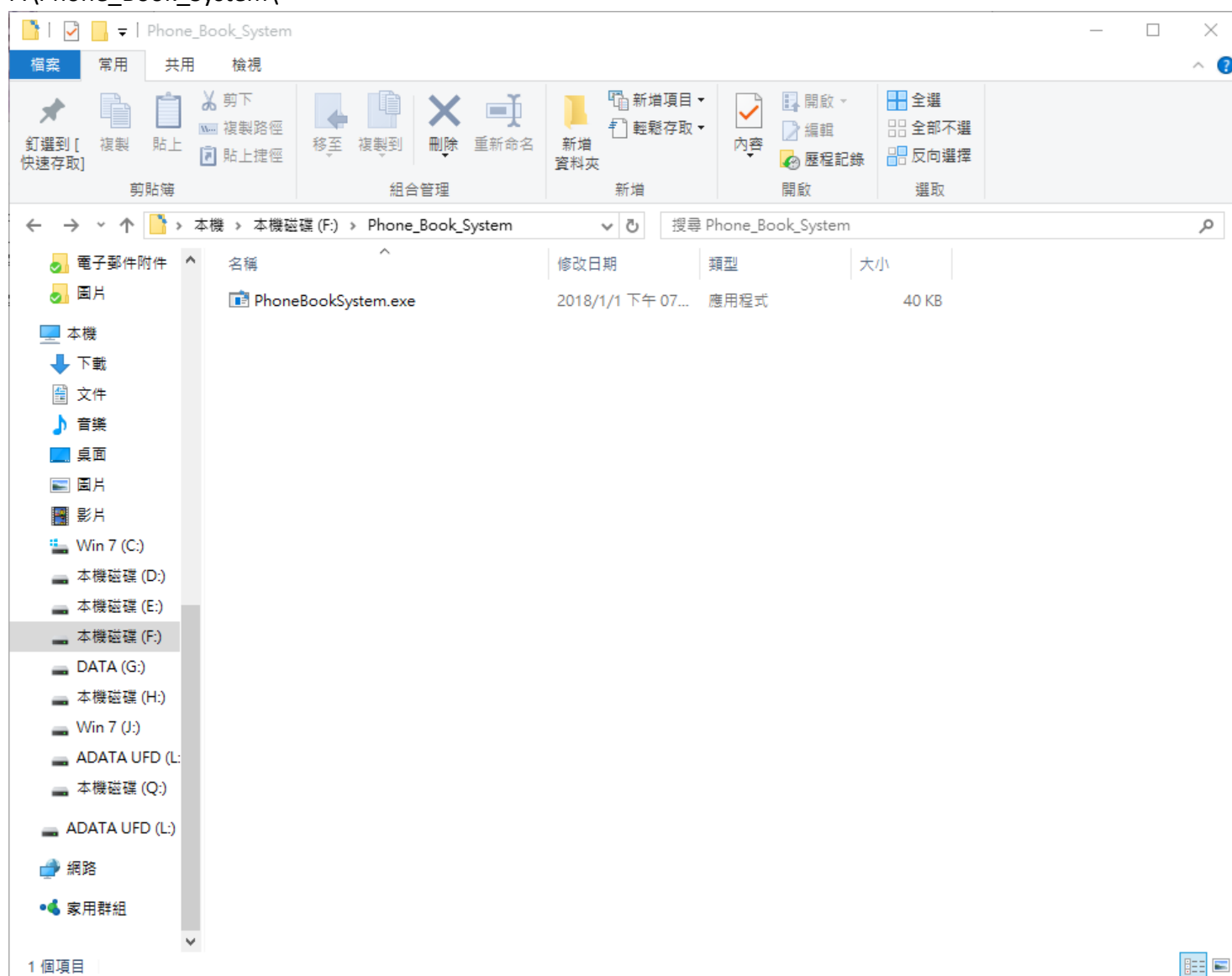
Base on the above, I will design a set of internal tests to find out the Run-time error and Logic error, and fix them if exist. Afterward, to make sure the program can satisfy users' expectations, I will invite several people to try the program and ask for their opinion.

4.3 Internal Tests

Functionality test

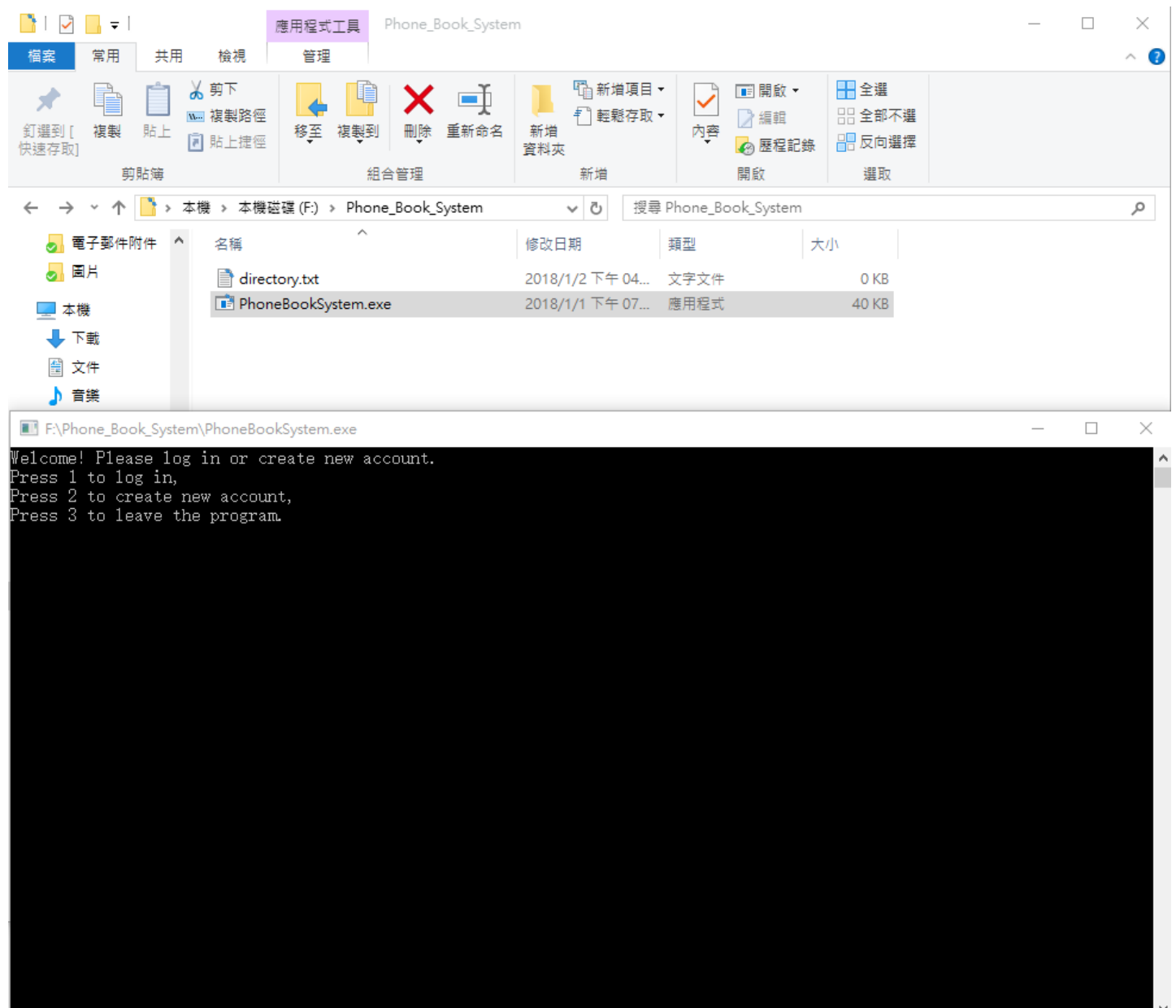
To ensure the program can run normally, I will input some correct data, which is satisfying the design, to ensure the functions of the program is all right. In the following, I will test all the functions by the user's view. I will use Windows 10 as the Operating System, and the program was saved in

F:\Phone_Book_System\



Launch the program

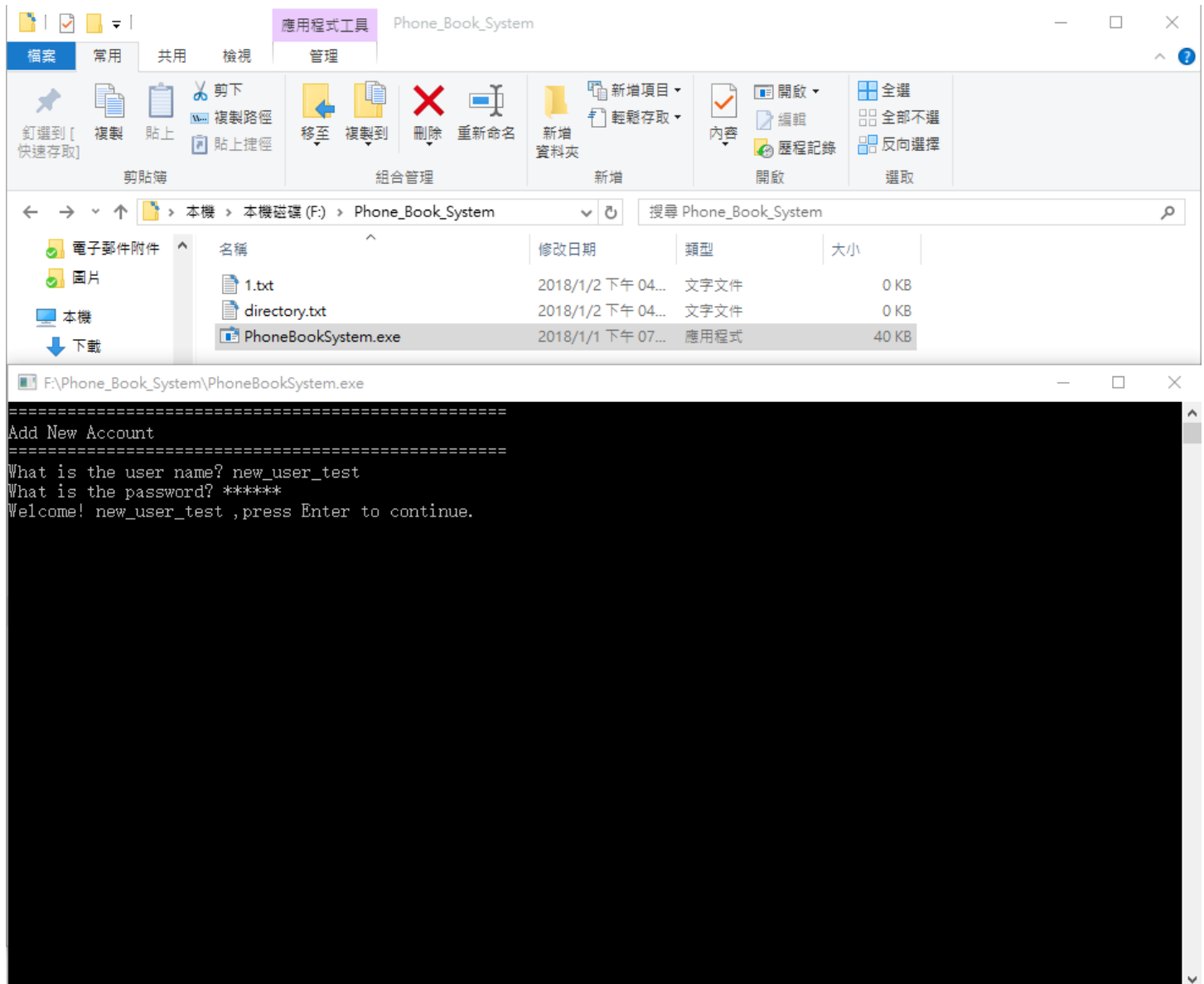
In **Chapter 3**, the program was designed to be independent, which means that the program can run even though there are not any .txt file together with the program. It was expected that the program can detect the condition and create an empty .txt file by itself.



The result is the same as design, it is functional.

Register

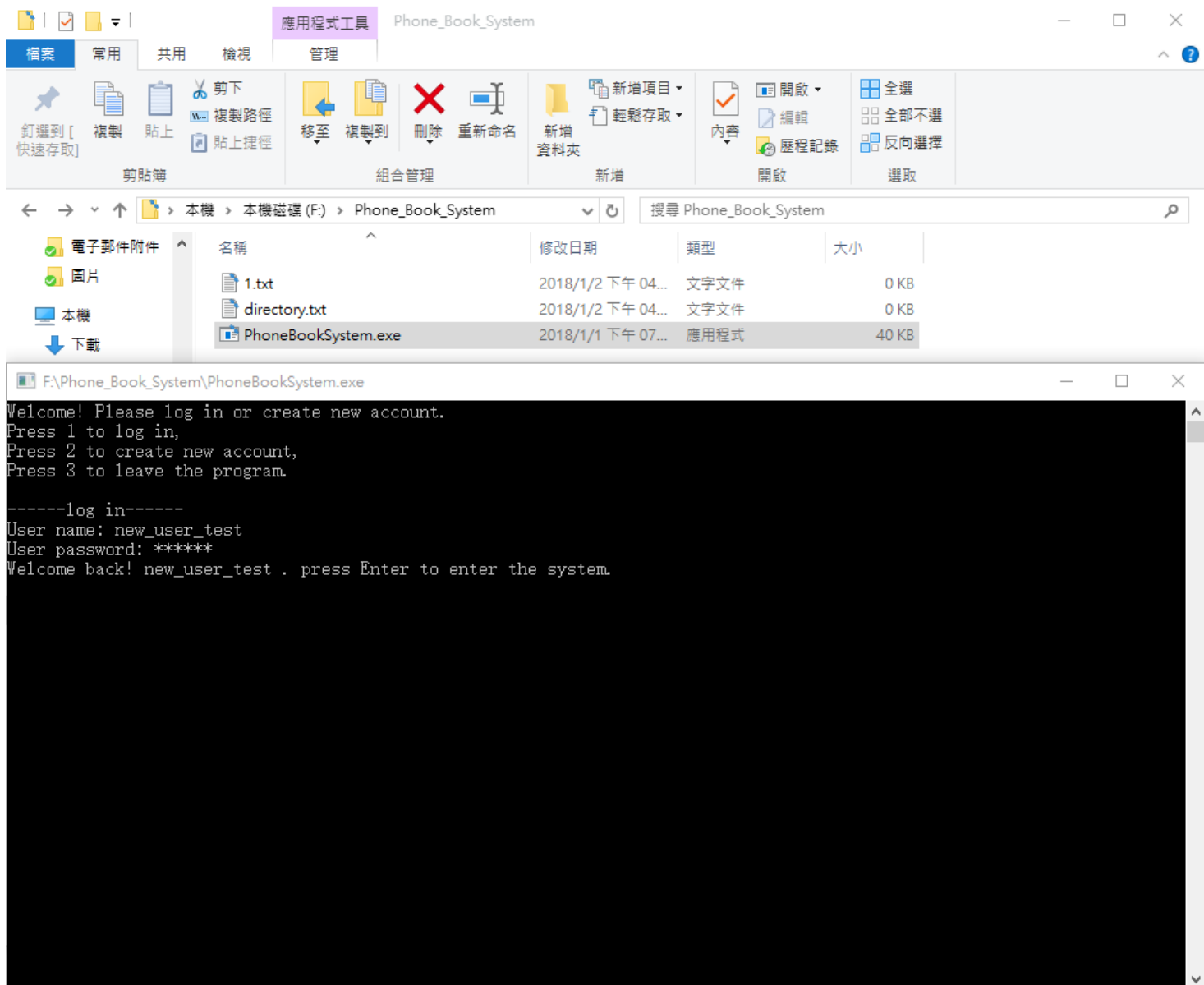
The program should be able to create new account, together with an empty .txt file with the user id. As this is the first user, the file name supposed to be "1.txt". The user name will be new_user_test, and the password will be a1b2c3.



The result is the same as design, it is functional.

Login

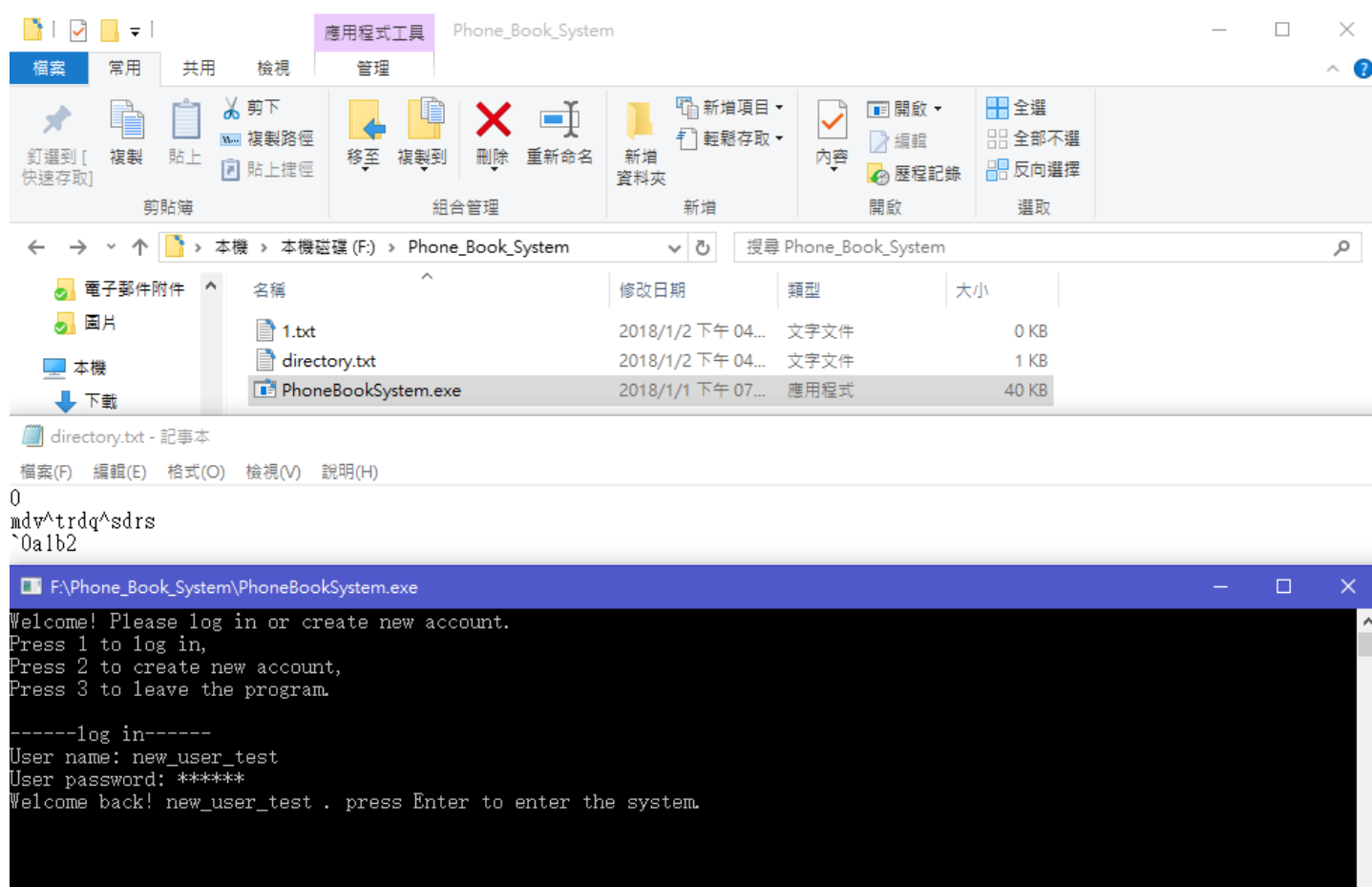
The program should be able to login in accordance to the above account.



The result is the same as design, it is functional.

Encryption and Decryption

According to the design, the data saved in directory.txt should be encrypted and became unreadable.



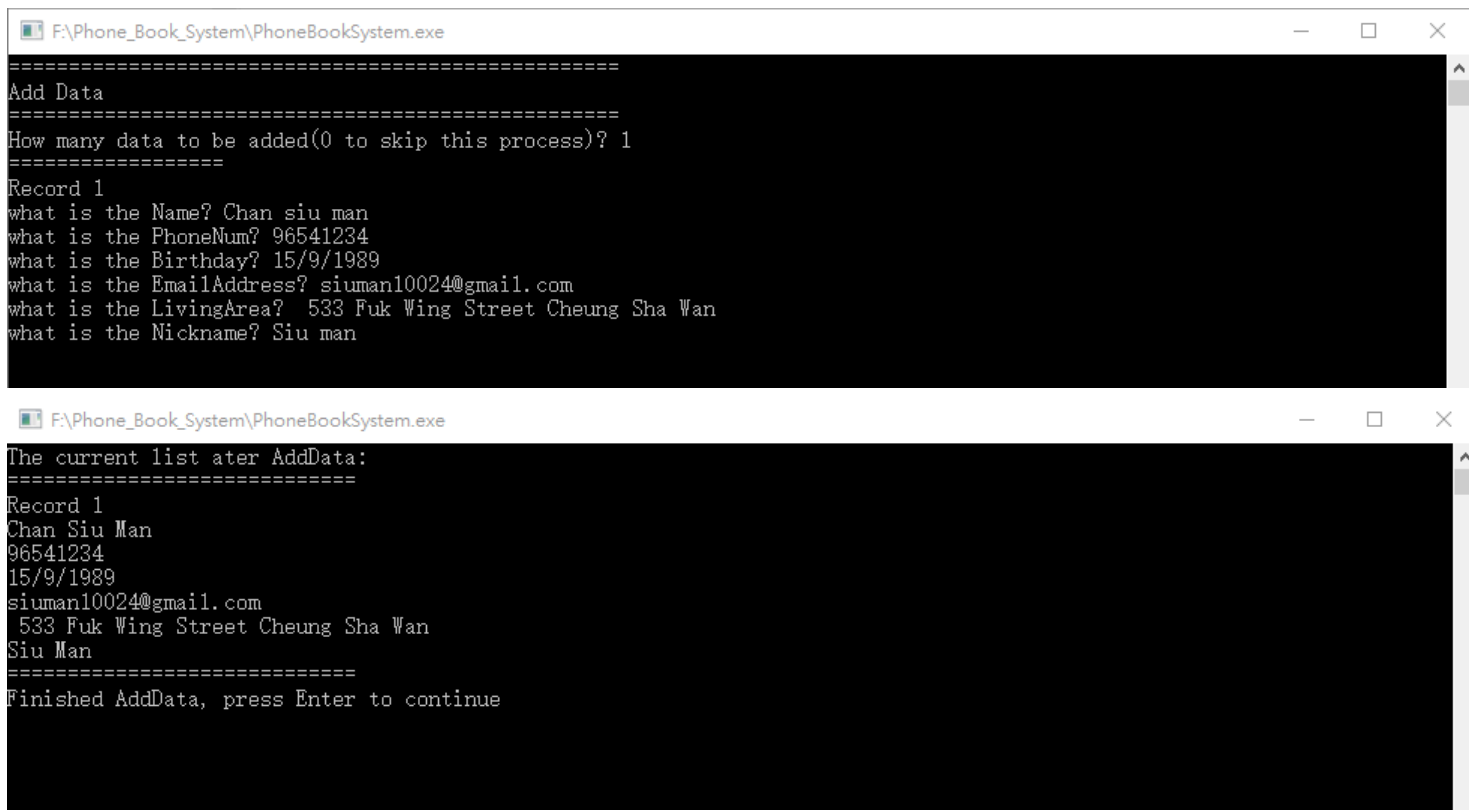
The result is the same as design, the program became unreadable for human but still can be read by the program. It is functional.

Main function



```
F:\Phone_Book_System\PhoneBookSystem.exe
Welcome! new_user_test . What you want to do?
1, Add new data
2, Remove current data
3, Change current data
4, Show current data
5, Log out
6, save and shut down
Please enter the corresponding number.
```

Adding record

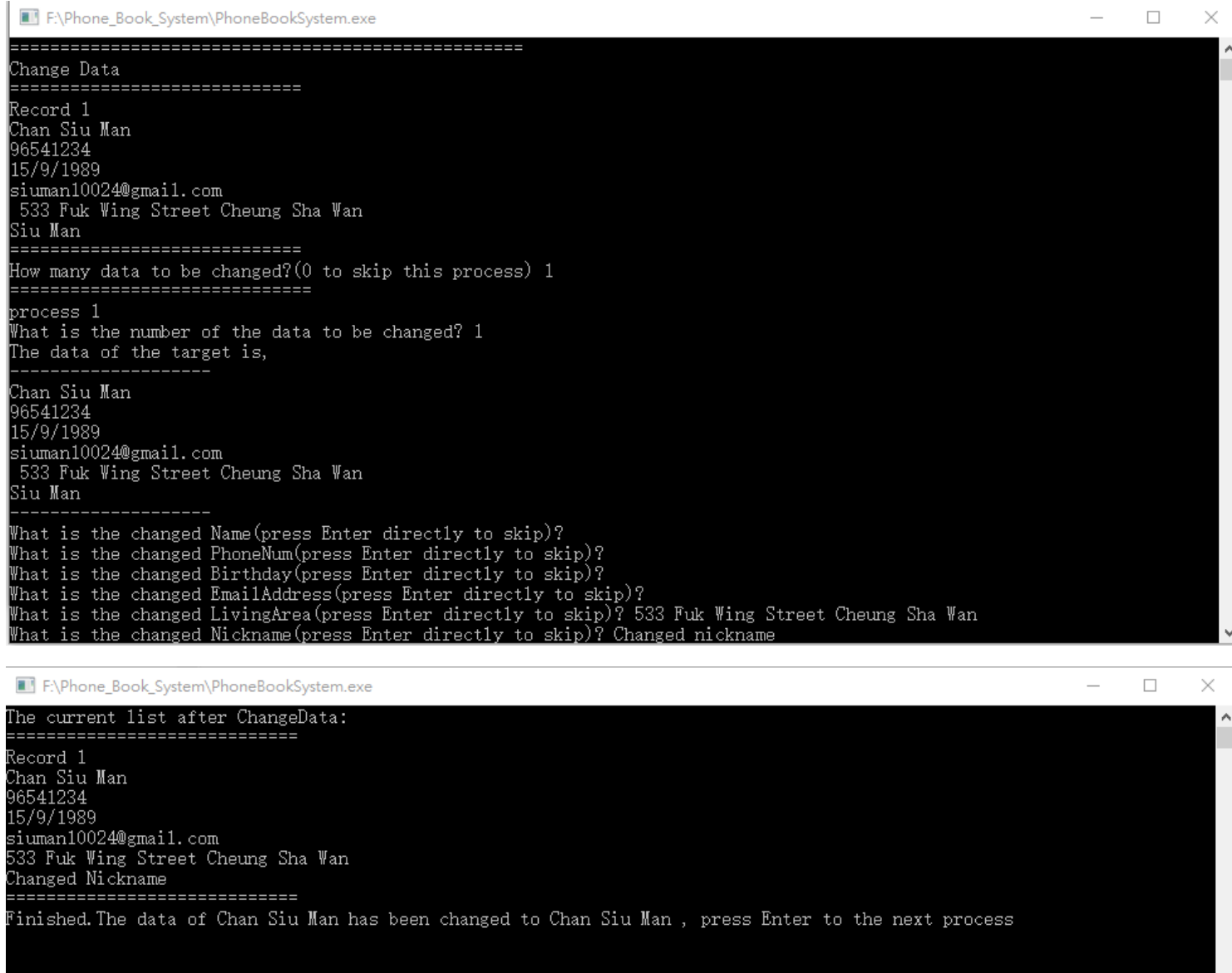


```
F:\Phone_Book_System\PhoneBookSystem.exe
=====
Add Data
=====
How many data to be added(0 to skip this process)? 1
=====
Record 1
what is the Name? Chan siu man
what is the PhoneNum? 96541234
what is the Birthday? 15/9/1989
what is the EmailAddress? siuman10024@gmail.com
what is the LivingArea? 533 Fuk Wing Street Cheung Sha Wan
what is the Nickname? Siu man

F:\Phone_Book_System\PhoneBookSystem.exe
The current list after AddData:
=====
Record 1
Chan Siu Man
96541234
15/9/1989
siuman10024@gmail.com
533 Fuk Wing Street Cheung Sha Wan
Siu Man
=====
Finished AddData, press Enter to continue
```

The result is the same as design, it is functional.

Changing record

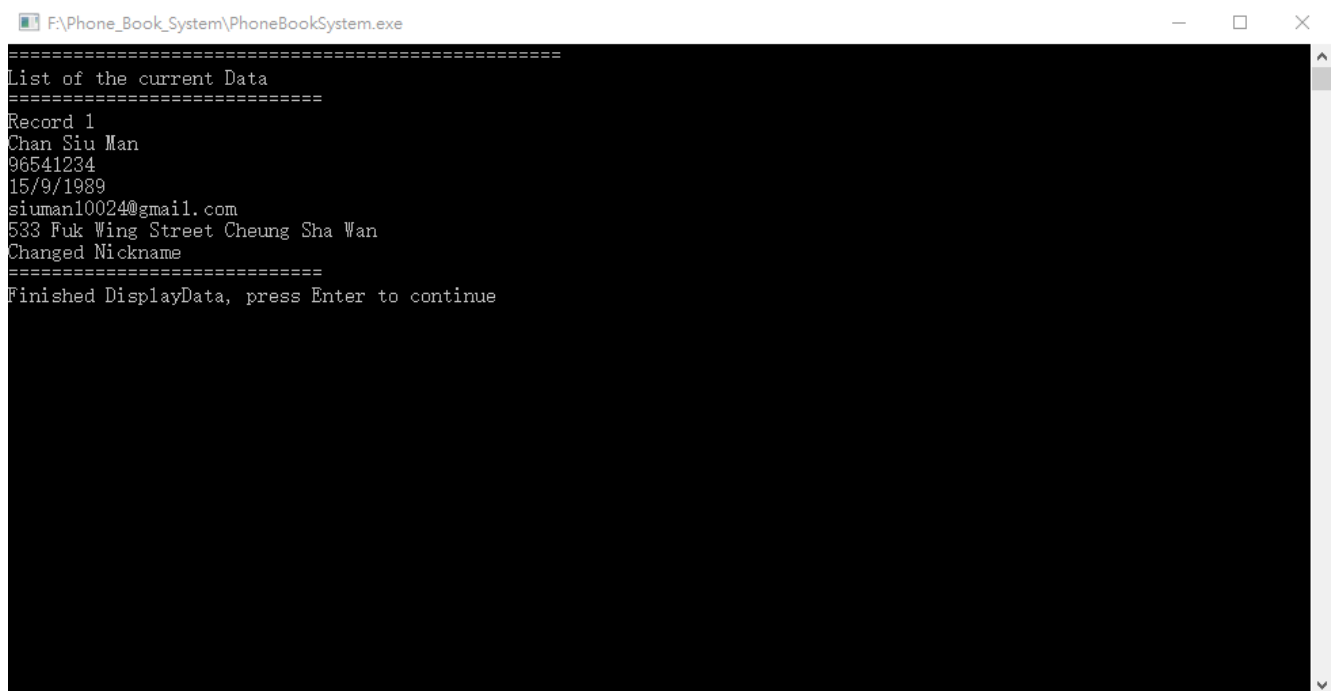


```
F:\Phone_Book_System\PhoneBookSystem.exe
=====
Change Data
=====
Record 1
Chan Siu Man
96541234
15/9/1989
siuman10024@gmail.com
533 Fuk Wing Street Cheung Sha Wan
Siu Man
=====
How many data to be changed?(0 to skip this process) 1
=====
process 1
What is the number of the data to be changed? 1
The data of the target is,
-----
Chan Siu Man
96541234
15/9/1989
siuman10024@gmail.com
533 Fuk Wing Street Cheung Sha Wan
Siu Man
-----
What is the changed Name(press Enter directly to skip)?
What is the changed PhoneNum(press Enter directly to skip)?
What is the changed Birthday(press Enter directly to skip)?
What is the changed EmailAddress(press Enter directly to skip)?
What is the changed LivingArea(press Enter directly to skip)? 533 Fuk Wing Street Cheung Sha Wan
What is the changed Nickname(press Enter directly to skip)? Changed nickname

F:\Phone_Book_System\PhoneBookSystem.exe
The current list after ChangeData:
=====
Record 1
Chan Siu Man
96541234
15/9/1989
siuman10024@gmail.com
533 Fuk Wing Street Cheung Sha Wan
Changed Nickname
=====
Finished. The data of Chan Siu Man has been changed to Chan Siu Man , press Enter to the next process
```

The result is the same as design, it is functional.

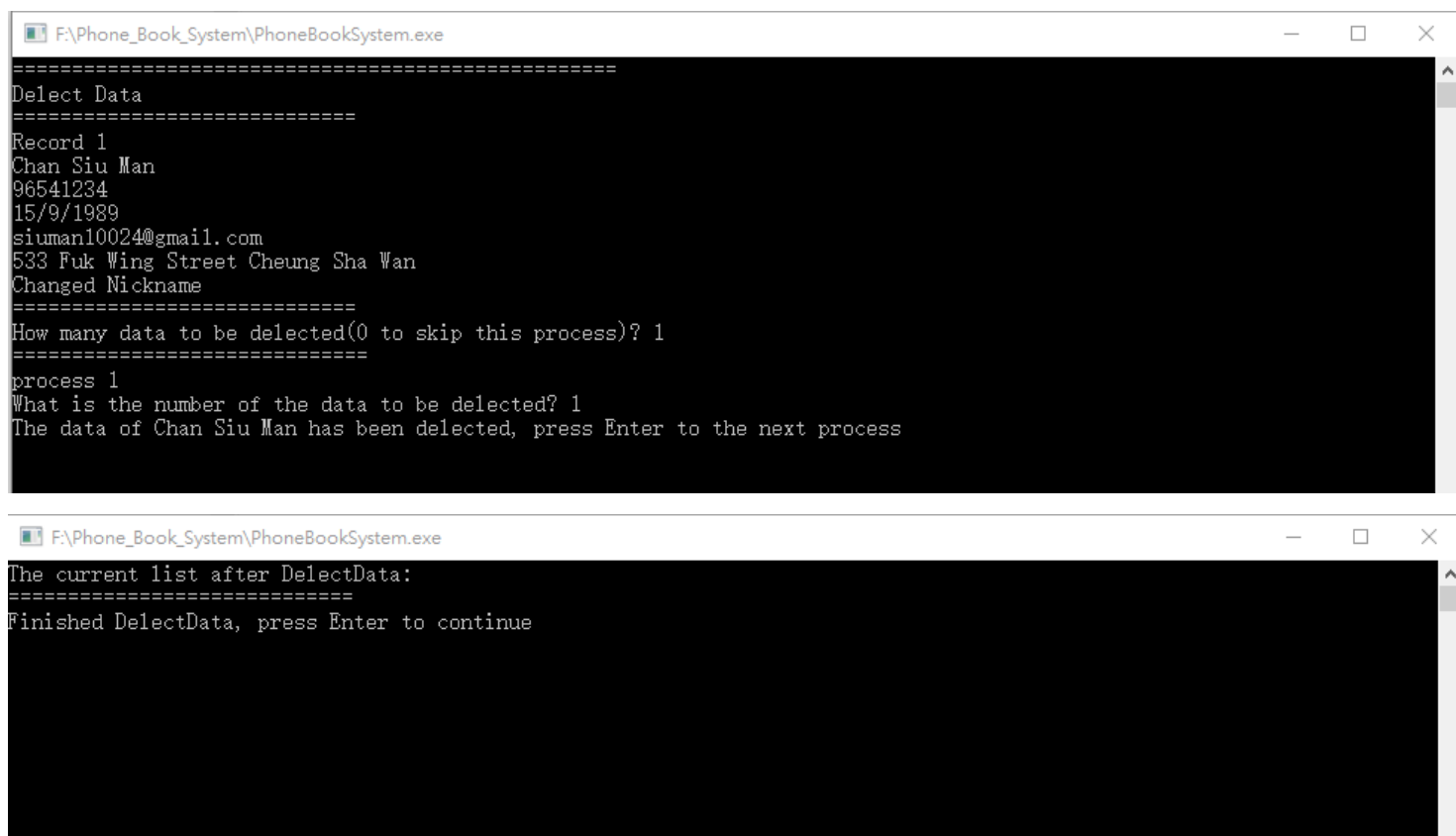
Showing record



```
F:\Phone_Book_System\PhoneBookSystem.exe
=====
List of the current Data
=====
Record 1
Chan Siu Man
96541234
15/9/1989
siuman10024@gmail.com
533 Fuk Wing Street Cheung Sha Wan
Changed Nickname
=====
Finished DisplayData, press Enter to continue
```

The result is the same as design, it is functional.

Delete record



```
F:\Phone_Book_System\PhoneBookSystem.exe
=====
Delect Data
=====
Record 1
Chan Siu Man
96541234
15/9/1989
siuman10024@gmail.com
533 Fuk Wing Street Cheung Sha Wan
Changed Nickname
=====
How many data to be delected(0 to skip this process)? 1
=====
process 1
What is the number of the data to be delected? 1
The data of Chan Siu Man has been delected, press Enter to the next process

F:\Phone_Book_System\PhoneBookSystem.exe
The current list after DelectData:
=====
Finished DelectData, press Enter to continue
```

The result is the same as design, it is functional.

Logout and shut down

```
F:\Phone_Book_System\PhoneBookSystem.exe
Welcome! new_user_test . What you want to do?
1, Add new data
2, Romove current data
3, Change current data
4, Show current data
5, Log out
6, save and shut down
Please enter the corresponding number.
press Enter to return the start page. See you next time!

F:\Phone_Book_System\PhoneBookSystem.exe
Welcome! new_user_test . What you want to do?
1, Add new data
2, Romove current data
3, Change current data
4, Show current data
5, Log out
6, save and shut down
Please enter the corresponding number.
Finished saving. press Enter to leave the program.
```

The results are the same as design, they are functional.

Conclusion of Functionality test

All the functions went the same as design and had no Run-time error and logic error.

Extreme Cases test

In the following, I will test the program with some extreme cases, which are not expected to occur during the design stage. The operating is difficult to show by picture as some of them are quite complicated. Thus, I will use text to represent.

Condition	Expected Result	Actual Result	Corrective measures
To edit some data with extreme long word length	The program is able to manage it.	All the fuchtions went as expected, but the messages became difficult to read.	Add some words in front of the data to indicate which type of data it is.
Close the program suddenly	The edited data is unable to save into the .txt files.	The same as expected. In addition, all the changes of data became invalid, and they returned into the status of last valid saving.	Save the data Frequently without user requirements.
Delete the .txt files from hard disk	All the data will disappear, but the program could still operate.	The same as expected. The status of the program is the same as the first time it launched.	Write instructions for use to warn the users not to edit the .txt files directly by themselves.
Input some meaningless data	The program is able to manage it	There aren't any error, but as a result the records becomes meaningless.	To detect whether the data is meaningful.

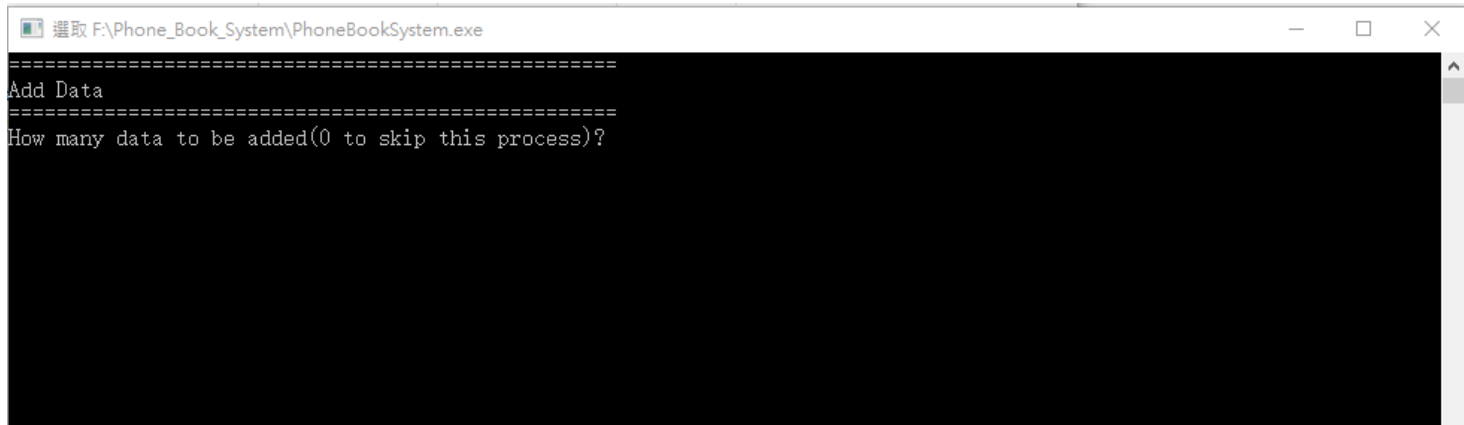
During the test, there aren't any Run-time error or Logic error even though I input incorrect data (condition 1 and 4) and use the program incorrectly (condition 2 and 3). Therefore, I surmise that there are roughly no serious problems in the program.

4.4 External Tests

I have invited several people to try my program, the result is as following.

Bugs of the program:

One bug was found by volunteers.



```
=====  
Add Data  
=====  
How many data to be added(0 to skip this process)?
```

When the program asks user to enter the amount of data to be edited, if the user press enter directly, or enter non-numeric data, Run-time error will occur.

The opinion from volunteers:

Positives

- 1, *It is useful for phone book data management.*
- 2, *The program is able to protect user's data.*
- 3, *Its file size is tiny and available for everyone.*
- 4, *The program operates smoothly*

Negatives

- 1, *The user interface is not friendly enough. It is quite difficult to read the data.*
- 2, *The process of adding and changing data is too cumbersome.*

Chapter 5 Evaluation

5.1 Pros and cons of my Program

After the above test, I found that my program has some Advantages and Disadvantages.

Pros:

- 1, Smooth operation*
- 2, High security level*
- 3, Tiny file size*
- 4, High availability*
- 5, Easy data management*

Cons:

- 1, The user interface is simple and crude*
- 2, Some process is cumbersome*
- 3, Insufficient functions of the program*
- 4, Existence of bugs*

5.2 Future Improvement

Base on the disadvantages of my program, I would make the following improvements:

- 1, Beautify the interface*
- 2, Simplify the process of adding and changing data*
- 3, Add some indicative words*
- 4, Add more functions to the program, for example, searching the record*
- 5, Write instructions for use to guide the users*
- 6, Fix the bugs if I can*

5.3 Self-Reflection

In this project, I acquired a precious experience on creating a large scale computer system. I never did something this big before because of my limited programming knowledge and time. However, computer systems to be large scale and comprehensive are the cornerstones to success nowadays so this is really a valuable experience.

Besides, my programming skills and logic skills were refined in doing this project as it is very complicated and huge in scale, involving a lot of calculations and complex logical thinking. Now, I can write more precise algorithms and solve problems more effectively. I am truly grateful to have this opportunity, hoping that the skills acquired can be made good use of in my future career.

Chapter 6 Reference and Acknowledgement

Internet

1. <http://borlpasc.narod.ru/english/faqs/test.htm>
(File Checking function)
2. <https://www.freepascal.org/docs-html/rtl/crt/index-5.html>
(Reference for unit 'Crt')

Books

1. NSS ICT Elective D1 Software Development

Acknowledgement

1. ICT teacher Mr. Chu – For his programming advice and lessons

Appendices:

Working schedule

Date	Task to be done
May-2017	Choice of Topic, Background research + Define the objectives + Propose Functions
July-2017	Design of Solution
Oct-2017	Implementation
Nov-2017	Testing & Evaluation
Dec-2017	Final Report

Program Code

```
Program PhoneBookSystem;
Uses crt, Dos;
type
  AccountType = record
    Uid : string;
    Uname : string;
    Upw : string
  end;
  RecordType = record
    Name : string;
    PhoneNum : string;
    Birthday : string;
    EmailAddress : string;
    LivingArea : string;
    Nickname : string
  end;
  ArrayType = array[1..100] of integer;
var
  PersonalData : array[1..100] of RecordType;
  currentuser : AccountType;
  userlist : array[1..50] of AccountType;
  ListOrder : arraytype;
  Procedure_templist : arraytype;
  countmax, accountmaxnum, i : integer;
  selection : char;
  target_text : string;
  ifleave, iflogout, iftype, endprogram : boolean;

function FileExist (name : string) : boolean;
var f : file; a : word;
begin
  assign (f, name);
  GetFAttr (f, a);
  FileExist := false;
  if DosError = 0 then if ((a and Directory) = 0) and ((a and Volumeld) = 0) then FileExist := true;
end;
```


Procedure Encryption_data;

var i, j, k : integer;

begin

 k := 0;

 for i := 1 to countmax do with PersonalData[i] do

 begin

 k := k mod 5 + 1;

 for j := 1 to length(Name) do Name[j] := chr(ord(Name[j])-k);

 for j := 1 to length(PhoneNum) do PhoneNum[j] := chr(ord(PhoneNum[j])-k);

 for j := 1 to length(Birthday) do Birthday[j] := chr(ord(Birthday[j])-k);

 for j := 1 to length(EmailAddress) do EmailAddress[j] := chr(ord(EmailAddress[j])-k);

 for j := 1 to length(LivingArea) do LivingArea[j] := chr(ord(LivingArea[j])-k);

 for j := 1 to length(Nickname) do Nickname[j] := chr(ord(Nickname[j])-k);

 end

end;

Procedure Encryption_directory;

var i, j, k : integer;

begin

 k := 0;

 for i := 1 to accountmaxnum do with Userlist[i] do

 begin

 k := k mod 5 + 1;

 for j := 1 to length(Uid) do Uid[j] := chr(ord(Uid[j])-k);

 for j := 1 to length(Username) do Username[j] := chr(ord(Username[j])-k);

 for j := 1 to length(Upw) do Upw[j] := chr(ord(Upw[j])-k);

 end

end;

Procedure Decryption_data;

var i, j, k : integer;

begin

 k := 0;

 for i := 1 to countmax do with PersonalData[i] do

 begin

 k := k mod 5 + 1;

 for j := 1 to length(Name) do Name[j] := chr(ord(Name[j])+k);

 for j := 1 to length(PhoneNum) do PhoneNum[j] := chr(ord(PhoneNum[j])+k);

 for j := 1 to length(Birthday) do Birthday[j] := chr(ord(Birthday[j])+k);

```

    for j := 1 to length(EmailAddress) do EmailAddress[j] := chr(ord(EmailAddress[j])+k);
    for j := 1 to length(LivingArea) do LivingArea[j] := chr(ord(LivingArea[j])+k);
    for j := 1 to length(Nickname) do Nickname[j] := chr(ord(Nickname[j])+k);
  end
end;

```

```

Procedure Decryption_directory;
var i, j, k : integer;
begin
  k := 0;
  for i := 1 to accountmaxnum do with Userlist[i] do
    begin
      k := k mod 5 + 1;
      for j := 1 to length(Uid) do Uid[j] := chr(ord(Uid[j])+k);
      for j := 1 to length(Username) do Username[j] := chr(ord(Username[j])+k);
      for j := 1 to length(Upw) do Upw[j] := chr(ord(Upw[j])+k);
    end
  end;
end;

```

```

Procedure Readfile(filename : string);
var
  i, j : integer;
  f : text;
begin
  assign(f, filename + '.txt');
  reset(f);
  i := 0;
  while not eof(f) do
    begin
      i := i + 1 ;
      readln(f, PersonalData[i].Name);
      readln(f, PersonalData[i].PhoneNum);
      readln(f, PersonalData[i].Birthday);
      readln(f, PersonalData[i].EmailAddress);
      readln(f, PersonalData[i].LivingArea);
      readln(f, PersonalData[i].Nickname);
    end;
  close(f);
  countmax := i;
  for j := 1 to countmax do ListOrder[j] := j;

```

end;

Procedure ReadDirectory;

var

i : integer;

f : text;

begin

i := 0;

if FileExist('directory.txt')

then begin

 assign(f, 'directory.txt');

 reset(f);

 while not eof(f) do

 begin

 i := i + 1 ;

 readln(f, userlist[i].Uid);

 readln(f, userlist[i].Uname);

 readln(f, userlist[i].Upw);

 end;

 close(f);

 accountmaxnum := i

end

else begin

 assign(f, 'directory.txt');

 rewrite(f);

 close(f);

 accountmaxnum := i

end;

end;

Procedure Uppercase;

var i, maxnum : integer;

begin

 maxnum := length(target_text);

 if ord(target_text[1]) in [97..122] then target_text[1] := chr(ord(target_text[1])-32);

 for i := 2 to maxnum-1 do

 if target_text[i] = ' ' then

 if ord(target_text[i+1]) in [97..122] then target_text[i+1] := chr(ord(target_text[i+1])-32);

end;

Procedure Writefile(filename : string);

var

i : integer;

f : text;

begin

assign(f, filename + '.txt');

rewrite(f);

for i := 1 to countmax do if ListOrder[i] > -1 then

with PersonalData[ListOrder[i]] do

begin

target_text := Name;

Uppercase;

Name := target_text;

writeln(f, Name);

writeln(f, PhoneNum);

writeln(f, Birthday);

writeln(f, EmailAddress);

target_text := LivingArea;

Uppercase;

LivingArea := target_text;

writeln(f, LivingArea);

target_text := Nickname;

Uppercase;

Nickname := target_text;

writeln(f, Nickname);

end;

close(f)

end;

Procedure Savefile(filename : string);

var

i : integer;

f : text;

begin

assign(f, filename + '.txt');

```

rewrite(f);
for i := 1 to countmax do if ListOrder[i] > -1 then
with PersonalData[ListOrder[i]] do
begin
    writeln(f, Name);
    writeln(f, PhoneNum);
    writeln(f, Birthday);
    writeln(f, EmailAddress);
    writeln(f, LivingArea);
    writeln(f, Nickname);
end;
close(f)
end;

```

```

Procedure SaveDirectory;
var i : integer; f : text;
begin
    assign(f, 'directory.txt');
    rewrite(f);
    for i := 1 to accountmaxnum do
    begin
        writeln(f, userlist[i].Uid );
        writeln(f, userlist[i].Uname);
        writeln(f, userlist[i].Upw)
    end;
    close(f);
end;

```

```

Procedure ClearRam;
var i : integer;
begin
    for i := 1 to countmax do
    begin
        PersonalData[i].Name := "";
        PersonalData[i].PhoneNum := "";
        PersonalData[i].Birthday := "";
        PersonalData[i].EmailAddress := "";
        PersonalData[i].LivingArea := "";
        PersonalData[i].Nickname := "";
    end;
end;

```

```

    countmax := 0
end;

Procedure Merge(left, middle, right : integer);
var
    pointLow, pointHigh, pointTemp, highlength, count : integer;
begin
    pointLow := left;
    pointHigh := middle + 1;
    pointTemp := left;
    repeat
        if PersonalData[ListOrder[pointLow]].Name <= PersonalData[ListOrder[pointHigh]].Name
            then begin
                Procedure_templist[pointTemp] := ListOrder[pointLow];
                pointLow := pointLow + 1
            end
            else begin
                Procedure_templist[pointTemp] := ListOrder[pointHigh];
                pointHigh := pointHigh + 1
            end;
        pointTemp := pointTemp + 1
    until (pointLow > middle) or (pointHigh > right);
    highlength := right - middle;
    if pointLow > middle
        then for count := pointHigh to right do Procedure_templist[count] := ListOrder[count]
        else for count := pointLow to middle do Procedure_templist[count + highlength] :=
ListOrder[count];
        for count := left to right do ListOrder[count] := Procedure_templist[count]
    end;

Procedure Mergesort(left, right : integer);
var
    middle : integer;
begin
    if left < right
        then begin
            middle := (left + right) div 2;
            MergeSort(left, middle);
            MergeSort(middle+1, right);
            Merge(left, middle, right)

```

```

        end
end;

Procedure Check_if_need_sort;
var flag : boolean;
begin
    i := 0;
    flag := false;
    while not flag and (i < countmax) do
        begin
            i := i + 1;
            if PersonalData[i].Name > PersonalData[i+1].Name then flag := True;
        end;
    if flag then Mergesort(1, countmax)
end;

procedure CreateAccount;
var f : text; inputchar : char; check : boolean;
begin
    ClrScr;
    writeln('=====');
    writeln('Add New Account');
    writeln('=====');
    accountmaxnum := accountmaxnum + 1;
    repeat
        write('What is the user name? ');
        readln(Userlist[accountmaxnum].Uname);
        write('What is the password? ');
        Userlist[accountmaxnum].Upw := '';
        repeat
            inputchar := Readkey;
            case inputchar of
                #8 : if length(Userlist[accountmaxnum].Upw) > 0 then
                    begin
                        Userlist[accountmaxnum].Upw := copy(Userlist[accountmaxnum].Upw, 1,
length(Userlist[accountmaxnum].Upw)-1);
                        GotoXY(WhereX-1,WhereY);
                        write(' ');
                        GotoXY(WhereX-1,WhereY)
                    end;
            end;

```

```

#13 : ;
else begin
    Userlist[accountmaxnum].Upw := Userlist[accountmaxnum].Upw + inputchar;
    write('*')

    end
end
until inputchar = #13;
if (Userlist[accountmaxnum].Uname <> '') and (Userlist[accountmaxnum].Upw <> '') then check := true;
if not check then begin writeln;writeln('Missing data! Please try again.') end
until check;
str(accountmaxnum, userlist[accountmaxnum].Uid);
assign(f, userlist[accountmaxnum].Uid+'.txt');
rewrite(f);
close(f);
writeln;
currentuser.Uname := Userlist[accountmaxnum].Uname;
currentuser.Upw := Userlist[accountmaxnum].Upw;
currentuser.Uid := Userlist[accountmaxnum].Uid;
writeln('Welcome! ',Userlist[accountmaxnum].Uname,' ,press Enter to continue. ');
readln
end;

```

Procedure Login;

var i : integer; flag : boolean; inputchar : char;

begin

repeat

Clrscr;

writeln('Welcome! Please log in or create new account.');

writeln('Press 1 to log in, ');

writeln('Press 2 to create new account, ');

writeln('Press 3 to leave the program. ');

selection := Readkey;

case selection of

'1' : begin

writeln;

writeln('-----log in-----');

write('User name: ');

readln(currentuser.Uname);

write('User password: ');


```

currentuser.Upw := "";
repeat
    inputchar := Readkey;
    case inputchar of
        #8 : if length(currentuser.Upw) > 0 then
            begin
                currentuser.Upw := copy(currentuser.Upw, 1, length(currentuser.Upw)-1);
                GotoXY(WhereX-1,WhereY);
                write(' ');
                GotoXY(WhereX-1,WhereY)
            end;
        #13 : ;
        else begin
            currentuser.Upw := currentuser.Upw + inputchar;
            write('*')

            end
        end
    until inputchar = #13;

i := 0;
flag := false;
while (not flag) and (i <= accountmaxnum) do
    begin
        i := i + 1;
        if ( currentuser.Uname = userlist[i].Uname) and ( currentuser.Uname <> "")
            and ( currentuser.Upw = userlist[i].Upw) and ( currentuser.Upw <> "") then
            begin
                currentuser.Uid := userlist[i].Uid;
                flag := true
            end
        end;
        if flag then begin writeln; write('Welcome back! ',currentuser.Uname,' . press Enter to enter
the system.') end
        else begin writeln;write('Wrong user name or password, please try again.') end;
        readln
    end;
'2' : begin CreateAccount;flag := true end;
'3' : begin ifleave := true; iftype := true; endprogram := true end
else writeln('Invalid value. Try again.')

```

```

    end
    until (flag) or (endprogram)
end;

procedure DisplayData;
var count : integer;
begin
    writeln('=====');
    for count := 1 to countmax do if ListOrder[count] <> -1 then
        begin
            writeln('Record ',count);
            with PersonalData[ListOrder[count]] do
                begin
                    writeln(Name);
                    writeln(PhoneNum);
                    writeln(Birthday);
                    writeln(EmailAddress);
                    writeln(LivingArea);
                    writeln(Nickname);
                end;
            writeln('=====');
        end
    end;
end;

```

```

{
Procedure DelectFile(target : integer);
var i : integer;
begin
    ListOrder[target] := -1;
end;

```

```

Procedure AddFile;
begin
    ListOrder[countmax] := countmax
end;

```

```

Procedure ChangeFile(target : integer);
begin

```

```

    ListOrder[target] := countmax
end;
}
{-----End of data process handout-----}

```

```

Procedure AddData;
var numofdata, i : integer;
begin
    ClrScr;
    writeln('=====');
    writeln('Add Data');
    writeln('=====');
    write('How many data to be added(0 to skip this process)? ');
    readln(numofdata);
    if numofdata > 0 then for i := 1 to numofdata do
    begin
        countmax := countmax + 1;
        writeln('=====');
        writeln('Record ',countmax);
        write('what is the Name? ');
        readln(target_text);
        Uppercase;
        PersonalData[countmax].Name := target_text;
        write('what is the PhoneNum? ');
        readln(PersonalData[countmax].PhoneNum);
        write('what is the Birthday? ');
        readln(PersonalData[countmax].Birthday);
        write('what is the EmailAddress? ');
        readln(PersonalData[countmax].EmailAddress);
        write('what is the LivingArea? ');
        readln(target_text);
        Uppercase;
        PersonalData[countmax].LivingArea := target_text;
        write('what is the Nickname? ');
        readln(target_text);
        Uppercase;
        PersonalData[countmax].Nickname := target_text;
        ListOrder[countmax] := countmax;
    end;
    if numofdata > 0 then Mergesort(1, countmax);

```

```

ClrScr;
writeln('The current list ater AddData: ');
DisplayData;
write('Finished AddData, press Enter to continue');
readln;
ClrScr;
end;

```

```

Procedure DelectData;
var numofdata, i, target : integer;
begin
  ClrScr;
  writeln('=====');
  writeln('Delect Data');
  DisplayData;
  write('How many data to be delected(0 to skip this process)? ');
  readln(numofdata);
  if numofdata > 0 then for i := 1 to numofdata do
    begin
      writeln('=====');
      writeln('process ',i);
      write('What is the number of the data to be delected? ');
      readln(target);
      write('The data of ', PersonalData[ListOrder[target]].Name, ' has been delected, press Enter to the
next process ');
      ListOrder[target] := -1 ;
      readln;
    end;
  WriteFile(currentuser.Uid);
  ReadFile(currentuser.Uid);
  Encryption_data;
  SaveFile(currentuser.Uid);
  Decryption_data;
  ClrScr;
  writeln('The current list after DelectData: ');
  DisplayData;
  write('Finished DelectData, press Enter to continue');
  readln;
  ClrScr;
end;

```

```

Procedure ChangeData;
var numofdata, i, target : integer; name_temp, input_temp : string;
begin
  ClrScr;
  writeln('=====');
  writeln('Change Data');
  DisplayData;
  write('How many data to be changed?(0 to skip this process) ');
  readln(numofdata);
  if numofdata > 0 then for i := 1 to numofdata do
  begin
    writeln('=====');
    writeln('process ',i);
    write('What is the number of the data to be changed? ');
    readln(target);
    writeln('The data of the target is, ');
    writeln('-----');
    with PersonalData[ListOrder[target]] do
      begin
        writeln(Name);
        writeln(PhoneNum);
        writeln(Birthday);
        writeln(EmailAddress);
        writeln(LivingArea);
        writeln(Nickname);
      end;
    writeln('-----');
    write('What is the changed Name(press Enter directly to skip)? ');
    name_temp := PersonalData[ListOrder[target]].Name;
    readln(input_temp);
    if input_temp <> '' then
      begin
        target_text := input_temp;
        Uppercase;
        input_temp := target_text;
        PersonalData[ListOrder[target]].Name := input_temp
      end;

    write('What is the changed PhoneNum(press Enter directly to skip)? ');

```

```

readln(input_temp);
if input_temp <> '' then PersonalData[ListOrder[target]].PhoneNum := input_temp;
write('What is the changed Birthday(press Enter directly to skip)? ');
readln(input_temp);
if input_temp <> '' then PersonalData[ListOrder[target]].Birthday := input_temp;
write('What is the changed EmailAddress(press Enter directly to skip)? ');
readln(input_temp);
if input_temp <> '' then PersonalData[ListOrder[target]].EmailAddress := input_temp;

write('What is the changed LivingArea(press Enter directly to skip)? ');
readln(input_temp);
if input_temp <> '' then
begin
target_text := input_temp;
Uppercase;
input_temp := target_text;
PersonalData[ListOrder[target]].LivingArea := input_temp
end;

write('What is the changed Nickname(press Enter directly to skip)? ');
readln(input_temp);
if input_temp <> '' then
begin
target_text := input_temp;
Uppercase;
input_temp := target_text;
PersonalData[ListOrder[target]].Nickname := input_temp
end;
ClrScr;
writeln('The current list after ChangeData: ');
DisplayData;
write('Finished.The data of ', name_temp, ' has been changed to ',
PersonalData[ListOrder[target]].Name, ' , press Enter to the next process ');
readln;
end;
if numofdata > 0 then Mergesort(1, countmax);
ClrScr;
writeln('The current list after ChangeData: ');
DisplayData;
write('Finished ChangeData, press Enter to continue');

```

```

    readln;
    ClrScr;
end;

begin
    ReadDirectory;
    Decryption_directory;
    ifleave := false;
    endprogram := false;
repeat
    Login;
    if not endprogram then
        begin
            ReadFile(currentuser.Uid);
            Decryption_data;
            Check_if_need_sort;
            ClrScr;
            iflogout := false;
            repeat
                writeln('Welcome! ', currentuser.Uname, ' . What you want to do? ');
                writeln('1, Add new data');
                writeln('2, Romove current data');
                writeln('3, Change current data');
                writeln('4, Show current data');
                writeln('5, Log out');
                writeln('6, save and shut down');
                write('Please enter the corresponding number. ');
                iftype := false;
                repeat
                    selection := Readkey;
                    case selection of
                        '1' : begin AddData; iftype := true end;
                        '2' : begin DelectData; iftype := true end;
                        '3' : begin ChangeData; iftype := true end;
                        '4' : begin
                            ClrScr;
                            writeln('=====');
                            writeln('List of the current Data');
                            DisplayData;
                            iftype := true;

```

```

        write('Finished DisplayData, press Enter to continue');
        readln;
        ClrScr;
    end;
'5' : begin
        currentuser.Uname := "";
        currentuser.Upw := "";
        iftype := true;
        iflogout := true;
        writeln;
        Encryption_data;
        SaveFile(currentuser.Uid);
        ClearRam;
        write('press Enter to return the start page. See you next time!');
        readln
    end;
'6' : begin ifleave := true; iftype := true end
else writeln('Invalid input. Try again.')
end;

until iftype
until (iflogout) or (ifleave)
end
until ifleave;
writeln;
Encryption_directory;
SaveDirectory;
if not endprogram then
begin
    Encryption_data;
    SaveFile(currentuser.Uid)
end;
write('Finished saving. press Enter to leave the program. ');
readln
end.

```