|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Text  Description automatically generated | | **ASSIGNMENT COVER PAGE** | | | C:\Users\hoching.tay\Desktop\Lincoln_UK_06092017-01.png |
| **Programme** | | | **Course Code and Title** | | |
| Bachelor of Computer Science (Hons)/  Bachelor of Computer Science (Hons) In Computer & Network Technology/  Bachelor of Software Engineering (Hons) | | | CPR3113/N Principles of Programming | | |
| **Student’s name / student’s id** | | | **Lecturer’s name** | | |
| CHAN SEOW FEN / 0207368 | | | Tan Phit Huan | | |
| **Date issued** | **Submission Deadline** | | | **Indicative Weighting** | |
| Week 3 - 26/09/2022 | Week 12 – 02/12/2022 | | | 30% | |
| **Assignment 2 title** | Functions, Arrays and Files | | | | |
| This assessment assesses the following course learning outcomes | | | | | |
| **# as in Course Guide** | **UOWM KDU Penang University College Learning Outcome** | | | | |
| LO3 | Implement function and arrays in problem solutions. | | | | |
| LO4 | Develop programs that create, read and write files. | | | | |
| **# as in Course Guide** | **University of Lincoln Learning Outcome** | | | | |
| LO1 | Identify, select, and apply appropriate data structures and operators in common programming solutions | | | | |
| LO3 | Apply object-oriented principles to the implementation of software programs | | | | |
| LO3 | Using appropriate knowledge of programming concepts, construct code segments and functions to perform input and output operations with error handling | | | | |
| LO1 | understand the time and space efficiency of algorithms and how to calculate/estimate/evaluate and improve them | | | | |
| **Student’s declaration** | | | | | |
| I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.  Student’s signature: Submission Date: | | | | | |

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **1.** | **Java source code** | 3 - 12 |
| **2.** | **Description**  **2.1 Demonstration using SampleInput.txt**  **2.2 Demonstration of validation system  in the program** | 13  14 – 20  20 - 34 |

1. **Java Source Code**

package assignment;

import java.io.\*; //for file input output

import java.util.Scanner; //scanner class use to scan user's input

import java.util.regex.Pattern; //for ic format pattern

public class assignment2 {

static Scanner sc = new Scanner (System.in);

static File file;

static Scanner input;

static String fileName = "SampleInput.txt"; //input file name

public static void main (String[]args)

{

try{

final int size = NoOfRecord(fileName); //get the number of records in the input file

String[] name = new String[size];

String[] identityCardNo = new String[size];

String[] bloodType = new String[size];

String[] weightStatus = new String[size];

String[] dob = new String[size];

String[] gender = new String[size];

double[] weight = new double[size];

double[] height = new double[size];

double[] bmi = new double[size];

readData(name,identityCardNo,bloodType); //read data from file

nameValidation(name); //name from file validation

ICValidation(identityCardNo); //IC from file validation

bloodTypeValidation(bloodType); //bloodtype from file validation

keyInHeight (height, name); //key in height and validation

keyInWeight (weight, name); //key in weight and validation

extractDOBfromIC(identityCardNo, dob); //extract dob from IC

extractGenderfromIC(identityCardNo, gender); //extract gender from IC

calculateBMI(height, weight, bmi); //calculate BMI

assignWeightStatus(bmi, size, weightStatus); //assign weight status according to BMI

double heightMean = calculateHeightMean(height); //calculate the height mean for the group of patients

double heightSD = calculateHeightSD(height,heightMean); //calculate the height standard deviation for the group of patients

double weightMean = calculateWeightMean(weight); //calculate the weight mean for the group of patients

double weightSD = calculateWeightSD(weight,weightMean); //calculate the weight standard deviation for the group of patients

String highest = findHighestPatient(height, name); //name for highest patients

String heaviest = findHeaviestPatient(weight, name); //name for heaviest patients

writeData(size, name, identityCardNo, bloodType, weightStatus, dob, gender,

height, weight, bmi, heightMean, heightSD, weightMean, weightSD, highest, heaviest); /\*write record of each patient,

the mean and standard deviation for the group of patients, and name of the highest and the heaviest patients into a text file\*/

}

catch(Exception e){

//as error message already prompted by other function, so nothing here

}

}

public static int NoOfRecord(String fileName) //get the number of records in the input file

{

int lines = 0;

try (BufferedReader reader = new BufferedReader(new FileReader(fileName)))

{

while (reader.readLine() != null)

{

lines++;

}

}catch (Exception e)

{

//as error message already prompted by other function, so nothing here

}

return lines;

}

public static void readData(String[] name,String[] identityCardNo,String[] bloodType) //read data from file

{

file = new File(fileName);

int i = 0;

try{

//make link

if (!file.exists())

{

System.out.println(fileName+" does not exist.");

}

input = new Scanner(file); //load the file into the scanner (input)

//process data

while(input.hasNext())

{

String line = input.nextLine();

String[] parts = line.split("#");

name[i] = parts[0];

identityCardNo[i] = parts[1];

bloodType[i] = parts[2];

i++;

}

input.close(); //close link

}

//NullPointerException, InputMismatchException, FileNotFoundException

catch(Exception e){

System.out.println("Something wrong");

}

}

public static void nameValidation(String[] name) //name from file validation

{

boolean validName=false;

for(int i=0;i<name.length;i++)

{

do

{

//validation for name

if (name[i].isBlank()) //check if the name input is blank

{

System.out.print("Invalid name for Record "+(i+1)+", it should not be blank.\n");

validName=false;

}

else

{

for (int j=0;j<name[i].length();j++) //checking if the name only contain space and alphabet

{

char character = name[i].charAt(j);

if (!Character.isLetter(character) && !Character.isWhitespace(character) && character!='/')

{

System.out.print("Invalid name for Record "+(i+1)+", it should not contain special characters or numbers.\n");

validName=false;

break;

}

else

{

validName=true;

}

}

}

while(!validName) //if its not a valid name, allow the user to key in again

{

System.out.print("Please enter the name again for Record "+(i+1)+'.');

System.out.print("\nName: ");

name[i]= sc.nextLine();

break;

}

}while(!validName);

}

}

public static void ICValidation(String[] identityCardNo) //IC from file validation

{

boolean validIC;

Pattern p = Pattern.compile("\\d{6}-\\d{2}-\\d{4}"); //patern for ic, xxxxxx-xx-xxxx

for(int i=0;i<identityCardNo.length;i++)

{

do

{

//validation for identityCardNo

if (!p.matcher(identityCardNo[i]).matches())

{

System.out.print("Invalid IC for Record "+(i+1)+", please follow the format 'xxxxxx-xx-xxxx' and enter the IC again for Record "+(i+1)+'.');

validIC=false;

}

else //validate correct birth date

{

int date = Integer.parseInt(identityCardNo[i].substring(4,6));

int month= Integer.parseInt(identityCardNo[i].substring(2,4));

if ((date>31 || date<1)||(month<1 || month>12))

{

System.out.print("Invalid IC for Record "+(i+1)+", six leading digits is incorrect, please enter the IC again for Record "+(i+1)+'.');

validIC=false;

}

else

{

validIC=true;

}

}

while(!validIC) //if its not a valid IC, allow the user to key in again

{

System.out.print("\nIdentity card number: ");

identityCardNo[i]=sc.nextLine();

break;

}

}while(!validIC);

}

}

public static void bloodTypeValidation(String[] bloodType) //bloodtype from file validation

{

boolean validBloodType;

for(int i=0;i<bloodType.length;i++)

{

//validation of blood type

do

{

if(!bloodType[i].equals("A+") && !bloodType[i].equals("A-")&&

!bloodType[i].equals("B+")&& !bloodType[i].equals("B-")&&

!bloodType[i].equals("O+")&& !bloodType[i].equals("O-")&&

!bloodType[i].equals("AB+")&& !bloodType[i].equals("AB-"))

{

validBloodType=false;

System.out.print("Invalid blood type for Record "+(i+1)+"."+"\nPlease enter the blood type again for record "+(i+1)+'.');

System.out.print("\nBlood type(A+,A-,B+,B-,O+,O-,AB+,AB-): ");

bloodType[i]=sc.nextLine(); //if its not a valid blood type, allow the user to key in again

}

else

{

validBloodType=true;

}

}while(!validBloodType);

}

}

public static void keyInHeight (double[] height, String[] name) //key in height and validation

{

boolean validHeight, heightNumeric=true;

for (int i=0;i<height.length;i++)

{

do

{

System.out.print("Please enter the height for Patient "+(i+1)+", "+name[i]+".");

System.out.print("\nHeight(in meter): ");

String s\_height = sc.nextLine();

try //check if the height input by administrator is number

{

height[i] = Double.parseDouble(s\_height); //convert string to numeric type

}

catch (NumberFormatException e) //indicate the string is not in numeric format

{

heightNumeric=false;

}

if(heightNumeric) //validate true height value which should not be negative or zero or higher than 3m.

{

if (height[i]>0 && height[i]<=3)

{

validHeight=true;

}

else

{

System.out.print("Invalid height, it should not be negative value or higher than 3m.\nPlease input again.\n");

validHeight=false;

}

}

else

{

System.out.print("Invalid height, please enter a valid height which is a number.\n");

validHeight=false;

heightNumeric=true; //reset heightNumeric to true for try catch

}

}while(!validHeight);

}

}

public static void keyInWeight (double[] weight, String[] name) //key in weight and validation

{

boolean validWeight, weightNumeric=true;

for (int i=0;i<weight.length;i++)

{

do

{

System.out.print("Please enter the weight for Patient "+(i+1)+", "+name[i]+".");

System.out.print("\nWeight(in kilogram): ");

String s\_weight = sc.nextLine();

try //check if the weight input by administrator is number

{

weight[i] = Double.parseDouble(s\_weight); //convert string to numeric type

}

catch (NumberFormatException e) //indicate the string is not in numeric format

{

weightNumeric=false;

}

if(weightNumeric)

{

if (weight[i]>0 && weight[i]<=640 ) //validate true weight value which should not be negative or zero or heavier than 640kg.

{

validWeight=true;

}

else

{

System.out.print("Invalid weight, it should not be 0 or lower than 0 or heavier than 640kg.\nPlease input again.\n");

validWeight=false;

}

}

else

{

System.out.print("Invalid weight, please enter a valid height which is a number.\n");

validWeight=false;

weightNumeric=true; //reset weightNumeric to true for try catch

}

}while(!validWeight);

}

}

public static void extractDOBfromIC(String[] identityCardNo, String[] dob) //extract dob from IC

{

for(int i=0;i<identityCardNo.length;i++)

{

dob[i] =identityCardNo[i].substring(4,6)+"/"

+identityCardNo[i].substring(2,4)+"/"

+identityCardNo[i].substring(0,2); //store dob in format of DD/MM/YY

}

}

public static void extractGenderfromIC(String[] identityCardNo, String[] gender) //extract gender from IC

{

for(int i=0;i<identityCardNo.length;i++)

{

if(identityCardNo[i].charAt(11)%2!=0) //last digit even number is female, odd number is male

{

gender[i] = "Male";

}

else

{

gender[i] = "Female";

}

}

}

public static void calculateBMI(double[] height, double[] weight,double[] bmi) //calculate BMI

{

for(int i=0;i<height.length;i++)

{

bmi[i]=weight[i]/(Math.pow(height[i],2)); //calculate bmi value

}

}

public static void assignWeightStatus(double[] bmi,int f, String[] weightStatus) //assign weight status according to BMI

{

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

{

if(bmi[i]>=30) //assigning weight status

{

weightStatus[i]="Obese";

}

else if(bmi[i]<30 && bmi[i]>=25)

{

weightStatus[i]="Overweight";

}

else if(bmi[i]<25 && bmi[i]>=18.5)

{

weightStatus[i]="Healthy Weight";

}

else

{

weightStatus[i]="Underweight";

}

}

}

public static double calculateHeightMean(double[] height) //calculate the height mean for the group of patients

{

double sum=0.0;

for (double h : height)

{

sum+=h;

}

return sum/height.length;

}

public static double calculateHeightSD(double[] height, double heightMean) //calculate the height standard deviation for the group of patients

{

double sum=0.0;

for (double h : height)

{

sum+=Math.pow(h - heightMean, 2);

}

return Math.sqrt(sum/height.length);

}

public static double calculateWeightMean(double[] weight) //calculate the weight mean for the group of patients

{

double sum=0.0;

for (double w : weight)

{

sum+=w;

}

return sum/weight.length;

}

public static double calculateWeightSD(double[] weight, double weightMean) //calculate the weight standard deviation for the group of patients

{

double sum=0.0;

for (double w : weight)

{

sum+=Math.pow(w - weightMean, 2);

}

return Math.sqrt(sum/weight.length);

}

public static String findHighestPatient (double[] height, String[] name) //name for highest patients

{

double highest = height[0];

String patients="";

for (int i=0;i<height.length;i++) //comparing highest height

{

if(height[i]>highest)

{

highest=height[i];

}

}

for (int j=0;j<height.length;j++)

{

if (height[j]==highest)

{

patients+=name[j]+", "; //add together all highest patients

}

}

return patients;

}

public static String findHeaviestPatient (double[] weight, String[] name) //name for heaviest patients

{

double heaviest = weight[0];

String patients="";

for (int i=0;i<weight.length;i++) //comparing heaviest weight

{

if(weight[i]>heaviest)

{

heaviest=weight[i];

}

}

for (int j=0;j<weight.length;j++)

{

if (weight[j]==heaviest)

{

patients+=name[j]+", "; //add together all heaviest patients

}

}

return patients;

}

public static void writeData(int size,String[] name, String[] identityCardNo, String[] bloodType,

String[] weightStatus, String[] dob, String[] gender, double[] height, double[] weight, double[] bmi,

double heightMean, double heightSD, double weightMean, double weightSD, String highestP, String heaviestP)

/\*write record of each patient,

the mean and standard deviation for the group of patients, and name of the highest and the heaviest patients into a text file\*/

{

try

{

PrintWriter output;

output = new PrintWriter("patientRecord.txt"); //output file name

for(int j=0;j<size;j++)

{

//write patient's information

output.println("\t\t\tRecord "+(j+1));

output.println("=======================================================");

output.println("Name: \t\t\t "+name[j]);

output.println("Identity card number: "+identityCardNo[j]);

output.printf("Height: \t\t %.2f m",height[j]);

output.printf("\nWeight: \t\t %.1f kg",weight[j]);

output.println("\nBlood type: \t\t "+bloodType[j]);

output.println("Date of birth[DD/MM/YY]: "+dob[j]);

output.println("Gender: \t\t "+gender[j]);

output.printf("BMI: \t\t\t %.1f",bmi[j]);

output.println("\nWeight status: \t\t "+weightStatus[j]);

output.println("=======================================================\n");

}

//write rhe mean and standard deviation for the group of patients, name of the highest and the heaviest patients

output.printf("\nMean of height: \t\t%.4f",heightMean);

output.printf("\nStandard deviation of height: \t%.4f",heightSD);

output.printf("\nMean of weight: \t\t%.4f",weightMean);

output.printf("\nStandard deviation of weight: \t%.4f",weightSD);

output.println("\nName of the highest patients: \t"+highestP.substring(0,(highestP.length()-2))); //-2 to delete ", "

output.println("Name of the heaviest patients: "+heaviestP.substring(0,(heaviestP.length()-2))); // -2 to delete ", "

output.close();//close link

}

catch (Exception e)

{

//NullPointerException, InputMismatchException, FileNotFoundException

System.out.println("Something wrong");

}

}

}

1. **Description**

This program is built to enable user to load patient’s name, identification number and blood type from a text file and key in each patient’s height and weight. Eventually, the program will write the record of each patient (name, identity card number, height, weight, blood type, date of birth, gender, BMI, and weight status), the mean and standard deviation of height and weight for the group of patients, and the name of the highest and the heaviest patients into a text file.

Graphical user interface, text, application, email

Description automatically generated

*Figure 1.0* Modify input file name

Text

Description automatically generated

*Figure 1.1* Modify output file name

The name for the input file to load patient’s information and the name for the output file to write patient’s records can be modify by the user as shown in *Figure 1.0* and *Figure 1.1*.

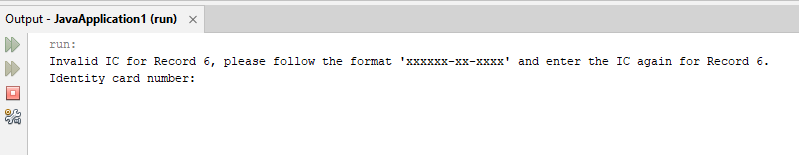
**2.1 Demonstration using SampleInput.txt**

Text

Description automatically generated

*Figure 2.0* SampleInput.txt

SampleInput.txt provided as shown in *Figure 2.0* will be used to demonstrate the program.



*Figure 2.1* Prompting invalid IC message for Record 6

After running the program, it first prompted that the IC for Record 6 is invalid. This is due to the reason that, the identity card number for record 6 is 06-05-5412 which does not follow the correct IC format, XXXXXX-XX-XXXX. Therefore, the program ask the user to key in the correct IC for Record 6 as shown in *Figure 2.1*.

Application

Description automatically generated with medium confidence

Text

Description automatically generated with low confidenceText

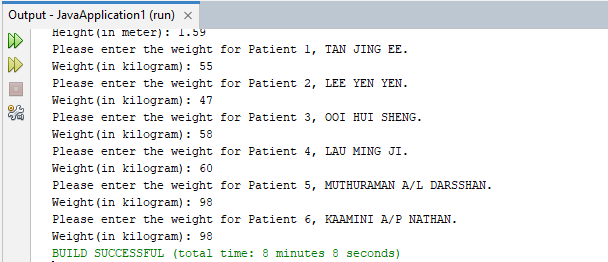
Description automatically generated

*Figure 2.2*, *Figure 2.3*, *Figure 2.4* Prompting for patient’s height

After the user key in the correct format of IC for Record 6, the program continues to prompt user to key in the height for each patient as shown in *Figure 2.2*, *Figure 2.3*, *Figure 2.4*.

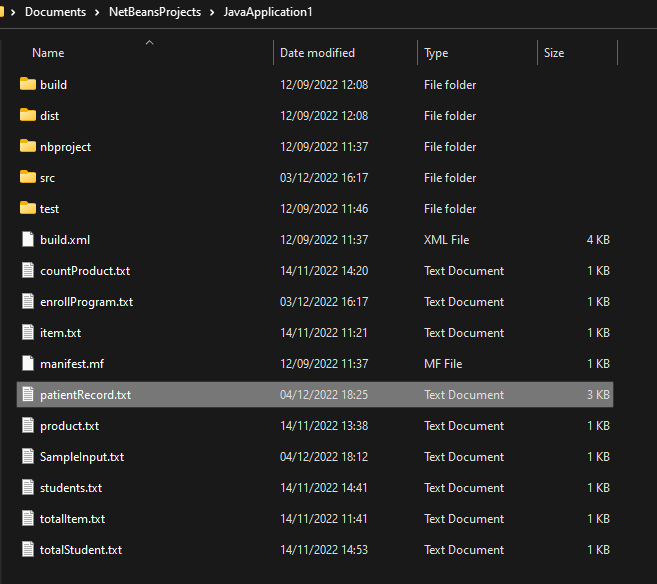
Text

Description automatically generated



*Figure 2.5*, *Figure 2.6* Prompting for patient’s weight

After the user key in valid height for each patient, the program continues to prompt user to key in the weight for each patient as shown in *Figure 2.5*, *Figure 2.6*. Then, the program will eventually end after the user key in valid weight for each patient.



*Figure 2.7* patientRecord.txt file in JavaApplication1 folder

Next, proceed to the JavaApplication1 folder, patientRecord.txt created can be found as shown in *Figure 2.7*.

Text

Description automatically generated Text

Description automatically generated

*Figure 2.8*, *Figure 2.9* patientRecord.txt file

After open the file, the record of each patient (name, identity card number, height, weight, blood type, date of birth, gender, BMI, and weight status), the mean and standard deviation of height and weight for the group of patients, and the name of the highest and the heaviest patients are written in the file as shown in *Figure 2.8*, *Figure 2.9*.

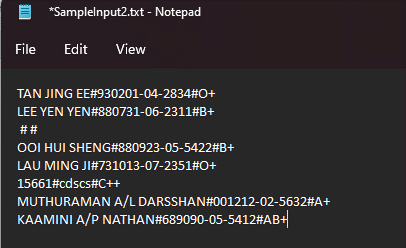
**2.2 Demonstration of validation system in the program**

Graphical user interface, text, application

Description automatically generated

*Figure 3.0* Input file does not exist

If the input file does not exist, the program will prompt error message as shown in *Figure 3.0*.



*Figure 3.1* SampleInput2.txt

In order to demonstrate the name validation, identity card number validation and blood type validation, SampleInput2.txt as shown in *Figure 3.1* is used.

Graphical user interface, text, application

Description automatically generated

*Figure 3.2* Prompt invalid name for Record 3

After running the program, due to the reason that the name for Record 3 is blank, the program prompted the user to key in a valid name for Record 3 as shown in *Figure 3.2*.

Text

Description automatically generated

*Figure 3.3* Invalid name – contain numbers

Text, letter

Description automatically generated

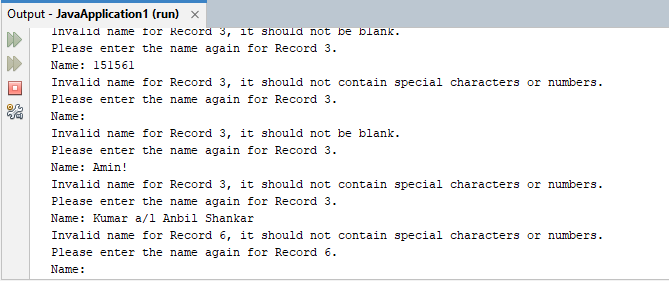
*Figure 3.4* Invalid name – blank

Graphical user interface, text, application

Description automatically generated

*Figure 3.5* Invalid name – contain special characters (‘/’ is excluded)

The program will validate name input by the user, if the name input is blank or contains numbers or special characters, the program will prompt error message and allow the user to repeat the input for patient’s name until the input is a valid name as shown in *Figure 3.3*, *Figure 3.4*, *Figure 3.5*.



*Figure 3.6* Prompt invalid name for Record 6

After entered a valid name for Record 3, the program will continue to prompt for the next invalid name, which is in Record 6, as the patient’s name in Record 6 is “15661”, which is not a valid name. (*Figure 3.6*)

Text

Description automatically generated

*Figure 3.7* Prompt invalid identity card number for Record 3

After entered a valid name for Record 6, due to the reason that the identity card number for Record 3 is blank, the program prompted the user to key in a valid identity card number for Record 3 as shown in *Figure 3.7*.

Graphical user interface, text, application

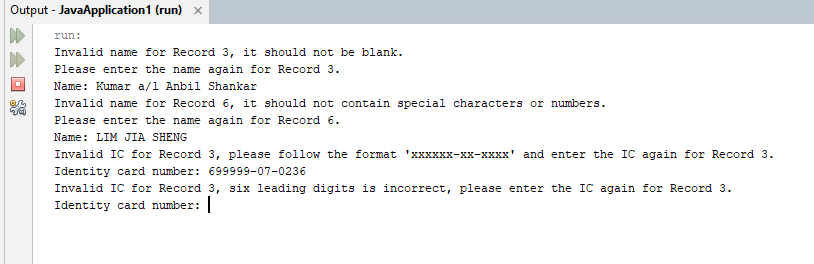
Description automatically generated

*Figure 4.0* Invalid identity card number – blank

Text, table

Description automatically generated with medium confidence

*Figure 4.1* Invalid identity card number – does not follow format “XXXXXX-XX-XXXX”



*Figure 4.2* Invalid identity card number – incorrect birth date and month (six leading digits)

The program will validate identity card number input by the user, if the identity card number input is blank or does not follow format “XXXXXX-XX-XXXX” or contains incorrect birth date and month (six leading digits) the program will prompt error message and allow the user to repeat the input for patient’s identity card number until the input is a valid identity card number as shown in *Figure 4.0*, *Figure 4.1*, *Figure 4.2*.

Graphical user interface, text, application

Description automatically generated

*Figure 4.3* Prompt invalid identity card number for Record 6

After entered a valid identity card number for Record 3, due to the reason that the identity card number for Record 6 is “cdscs”, which is not a valid identity card number, the program prompted the user to key in a valid identity card number for Record 6 as shown in *Figure 4.3*.

Text

Description automatically generated

*Figure 4.4* Prompt invalid identity card number for Record 8

After entered a valid identity card number for Record 6, due to the reason that the identity card number for Record 8 is “689090-05-5412”, which is not a valid identity card number, the program prompted the user to key in a valid identity card number for Record 8 as shown in *Figure 4.4*.

Graphical user interface, text, application

Description automatically generated

*Figure 4.5* Prompt invalid blood type for Record 3

After entered a valid identity card number for Record 8, due to the reason that the blood type for Record 3 is blank, the program prompted the user to key in a valid blood type for Record 3 as shown in *Figure 4.5*.

Text

Description automatically generated

*Figure 4.6* Invalid blood type – not in given list

The program will validate the patient’s blood type input by the user which should be in the given list (A+,A-,B+,B-,O+,O-,AB+,AB-). The program will prompt error message and allow the user to repeat the input for patient’s blood type for every invalid input. (*Figure 4.6)*

Graphical user interface, text, application

Description automatically generated

*Figure 4.7* Prompt invalid blood type for Record 6

After entered a valid blood type for Record 3, due to the reason that the blood type for Record 6 is “C++”, which is not a valid blood type, the program prompted the user to key in a valid blood type for Record 6 as shown in *Figure 4.7*.

Text

Description automatically generated

Text

Description automatically generated

*Figure 4.8*, *Figure 4.9* Prompting for patient’s height

After entered a valid blood type for Record 6, the program will prompt for patient’s height for each patient as shown in *Figure 4.8*, *Figure 4.9*.

Text

Description automatically generated

*Figure 5.0* Invalid height – contain non-numeric character

Text

Description automatically generated

*Figure 5.1* Invalid height – not within normal range (0m<x<= 3m)

The program will validate the patient’s height input by the user which should be numeric and within normal height range (0m<x<= 3m). The program will prompt error message and allow the user to repeat the input for patient’s height for every invalid input. (*Figure 5.0, Figure 5.1*)

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

*Figure 5.2*, *Figure 5.3* Prompting for patient’s weight

After entered a valid height for each patient, the program will prompt for patient’s weight for each patient as shown in *Figure 5.2*, *Figure 5.3*.

Text

Description automatically generated

*Figure 5.4* Invalid weight – contain non-numeric character

Text

Description automatically generated

*Figure 5.5* Invalid weight – not within normal range (0kg<x<= 640kg)

The program will validate the patient’s weight input by the user which should be numeric and within normal weight range (0kg<x<=640kg). The program will prompt error message and allow the user to repeat the input for patient’s weight for every invalid input. (*Figure 5.4, Figure 5.5*)

Text

Description automatically generated

*Figure 5.6* End of the program

After the user key in valid weight for each patient, the program will eventually end as shown in *Figure 5.6*.

Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated

*Figure 5.7*, *Figure 5.8*, *Figure 5.9* patientRecord.txt file

*Figure 5.7*, *Figure 5.8*, *Figure 5.9* shows the patientRecord.txt file produced by the data key in as shown above.