

Assignment 3 Report

Samar Ahmed

Computer Science and Engineering Department, The American University in Cairo

Cs 1101

Dr. Howaida Ismaeel

3/25/2023

This report contains screenshots of the codes and the output of the program including its source files and header files, showing the 5 Classes that were created, with a detailed explanation as to why I used specific codes and an explanation of the output.

Person.h and Person.cpp:

```
1  #pragma once
2  #include "Appointment.h"
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  class Person { //The base class
8  private:
9      string Name;
10     string ID;
11     int Age;
12
13 public:
14     Person();
15
16     void setName(string n);
17     void setID(string n);
18     void setAge(int a);
19
20     string getName();
21     string getID();
22     int getAge();
23
24     void printInfo();
25 };
```

```
1  #include "Person.h"
2  //base class
3  Person::Person() { //default constructor
4      Name = "";
5      Age = 0;
6      ID = "";
7  }
8
9  void Person::setName(string n) { Name = n; } //setter for name
10 void Person::setID(string n) { ID = n; } //setter for ID
11 void Person::setAge(int a) { Age = a; } //setter for age
12
13 string Person::getName() { return Name; } //getter for name
14 string Person::getID() { return ID; } //getter for ID
15 int Person::getAge() { return Age; } //getter for Age
16
17 void Person::printInfo() { //print
18     cout << "Name : " << Name << endl;
19     cout << "ID : " << ID << endl;
20     cout << "Age : " << Age << endl;
21 }
```

The first class was Person. It contains a header source to list all its functions and attributes and a source file to define all the functions. Person is the base class that the rest of the class will be inheriting from. Person has name of type String, the ID of type string since the ID can be composed of letters and numbers, and the Age. All of these attributes will be required in the other classes therefore, it can be written only once and get inherited. At the public, there should be a constructor, setters and getters, and a display function. Their definitions are shown in the source file and person.h must be included.

Appointment.h:

```
1  #pragma once // to program once
2  struct Appointment {
3      int hour;
4      int mins;
5  };
```

Appointment is a struct that has the attributes: hours and minutes only. This is to get the time of the customers and mechanics and to compare them. Leading to the customer being assigned to a specific mechanic. This struct is a header file since there are no functions to be defined and it was included in Person.h. By doing so the other classes will inherit from Person.h will also have appointment.h.

Mechanic.h:

```
1  #include "Person.h"
2
3  class Mechanic : public Person { //inherited from the base class person
4  private:
5      int counter;
6      Appointment *apps;
7
8  public:
9      Mechanic();
10
11     void setCounter(int c);
12     void setApps(Appointment *a);
13     Appointment* getApps();
14     int getCounter();
15     bool isAvailable(Appointment ap);
16     void addAppointment(int h, int m);
17     void printInfo();
18 };
```

Mechanic.h inherits from the base class person since all mechanics are required to have a name, ID, and age. However, they will need extra attributes that are only specific to them such as the counter and the pointer of type appointment, to create a dynamic array later on. The functions

that are specific to mechanics would be its constructors, the setters and getters for the new attributes, the bool function to check if the mechanic is available, the function to add the new appointment, and finally its own display function.

Mechanic.cpp:

```
1  #include "Mechanic.h"
2
3  Mechanic::Mechanic() : Person() { //inherits from person class
4      counter = 0;
5      apps = new Appointment[30]; //array of appointment
6  }
7
8  void Mechanic::setCounter(int c) { counter = c; } //setter for counter
9  void Mechanic::setApps(Appointment *a) { apps = a; } //setter for apps object
10
11 int Mechanic::getCounter() { return counter; } //getter for counter
12 Appointment *Mechanic::getApps() { return apps; } //getter for apps
13
14 bool Mechanic::isAvailable(Appointment ap) { //function to check if mechanic is available
15     for (int i = 0; i < counter; i++) {
16         if (apps[i].hour == ap.hour && apps[i].mins == ap.mins) {
17             return false;
18         }
19     }
20     return true;
21 }
22
23 void Mechanic::addAppointment(int h, int m) {
24     apps[counter].hour = h;
25     apps[counter].mins = m;
26     counter++;
27 }
28
29 void Mechanic::printInfo() { //print by inheriting from class then adding extra outputs
30     Person::printInfo();
31     cout << "Number of appointments = " << counter << endl;
32     for (int i = 0; i < counter; i++) {
33         cout << apps[i].hour << " : " << apps[i].mins << endl;
34     }
35     cout << endl;
36 }
37 }
```

After including the header file in the source file all of the functions will be defined. The constructor will have inherited from the person constructor since defining name, age, and ID would be the same, and counter and app will be added to the constructor. An array would be created of size 30. The setter and getters would then be defined. For the function isAvailable. First of all, the function would allow input of type ap, there would be a for loop that would end depending on the size of counter, then an if statement that shows if the appointment that has been assigned is equal to one of the appointments that the mechanic already has then return false, the mechanic cannot take the job. However, if the hours and mins do not match then it shows he is free and the function would return true. For the addAppointment function, it would take the hour and mins imputed and add it to the list of appointments by the user and increase the counter by

one. Finally, the print info would inherit from person and add extra outputs such as the counter, showing the number of appointments the mechanic has, and a loop to display all of the times the mechanic has appointments during the day by outputting the array of type appointment.

Customer.h:

```
#include "Person.h"

class Customer : public Person { //inherited from the base class person
private:
    string mechanicID;
    Appointment appointment;

public:
    Customer();

    string getMechanicID();
    Appointment getAppointment();

    void setMechanicID(string n);
    void setAppointment(Appointment app);

    //operator overload
    bool operator==(Customer &c);
    bool operator>(Customer &c);
    bool operator<(Customer &c);

    void printInfo();
};
```

Customer.h inherits from base class person, the class customer will contain all of the attributes and functions that the class Person has. It will contain extra attributes which are the mechanicID and the appointment of type appointment. The functions will be the constructor, the setters and getters, the operating overload functions, and the display function.

Customer.cpp:

```
1  #include "Customer.h"
2  //inherits from person class
3  Customer::Customer() : Person() { //default constructor
4      mechanicID = "";
5      appointment.hour = 0;
6      appointment.mins = 0;
7  }
8
9  string Customer::getMechanicID() { return mechanicID; } //getter for mechanicID
10 Appointment Customer::getAppointment() { return appointment; } //setter for mechanicID
11
12 void Customer::setMechanicID(string n) { mechanicID = n; } //setter for mechanicID
13 void Customer::setAppointment(Appointment app) { appointment = app; } //setter for app object
14
15 bool Customer::operator==(Customer &c) { //operator overload ==
16     if (appointment.hour == c.appointment.hour &&
17         appointment.mins == c.appointment.mins)
18         return true;
19     else
20         return false;
21 }
22
23 bool Customer::operator>(Customer &c) { //operator overload form greater than
24     if (appointment.hour > c.appointment.hour)
25         return true;
26     else if (appointment.hour < c.appointment.hour)
27         return false;
28     else {
29         if (appointment.mins > c.appointment.mins)
30             return true;
31         else
32             return false;
33     }
34 }
35
36 bool Customer::operator<(Customer &c) { //operator overload for smaller than
37     if (appointment.hour < c.appointment.hour)
38         return true;
39     else if (appointment.hour > c.appointment.hour)
40         return false;
41     else {
42         if (appointment.mins < c.appointment.mins)
43             return true;
44         else
45             return false;
46     }
47 }
48
49 void Customer::printInfo() { //print
50     Person::printInfo();
51     cout << "You are appointed to the mechanic with this ID: " << mechanicID << " at " << appointment.hour << ":" << appointment.mins<<endl<<endl;
52 }
53 }
```

Similar to the mechanic.cpp, customer.cpp constructor will also be inheriting from the constructor of person however the appointment hour, mins and mechanicID will be initialized to blanks. The setters and getters will then be defined. The operating overload that has been defined is three different types. First is the equal operator. It will receive a type of customer and would already have this operator. This operator and input c will be compared depending on the hours and minutes. If they are equal then the operator will return true if not it will return false. The second operation is greater than. First, there will be two types of if statements one for the hour and the other for the minute they will both compare this operator with input c and if this is great than c then it will return true other than that it will return false. Finally, the smaller-than operator

is similar to the greater-than operator just with different signs. If this is smaller than input c then the function will return true otherwise it will return false. Furthermore, there is a display function that will contain the display function of Person since it will inherit from it using the scope resolution operator. Then there will be an extra output showing the MechanicID of the mechanic that has been assigned to the customer at a specific time. This display will not have the same display as the mechanic since they do not inherit from each other.

Queue.h:

```
1  template <class Type> //template class <T> Provide sample tem
2  class Queue {
3  private:
4      Type *arr; // To create a dynamic array of type Type
5      int rear;
6      int front;
7      int numberOfItems;
8  public:
9      Queue() {
10         arr = new Type[50]; //initialize array
11         rear = -1;
12         front = 0;
13         numberOfItems = 0;
14     }
15     void push(Type x) {
16         if (rear == 50)
17             rear = 0;
18         else
19             rear++;
20         arr[rear] = x;
21         numberOfItems++;
22     }
23     Type pop() {
24         Type x = arr[front];
25         if (front == 50)
26             front = 0;
27         else
28             front++;
29         numberOfItems--;
30         return x;
31     }
32     bool isEmpty() {
33         if (numberOfItems == 0)
34             return true;
35         else
36             return false;
37     }
38     int length() { return numberOfItems; }
39 };
```

Queue.h uses a template class, the declaration and the definition of the attributes and the functions are all written in one file since it uses a template class and an error is given if they are separated. The private attributes are a pointer of Type called arr, the rear, front, and the numberOfItems. There is a constructor, that will create a dynamic array of size 50, the rear with -1 since it needs to be before the start of the array index to represent the back of the array, the

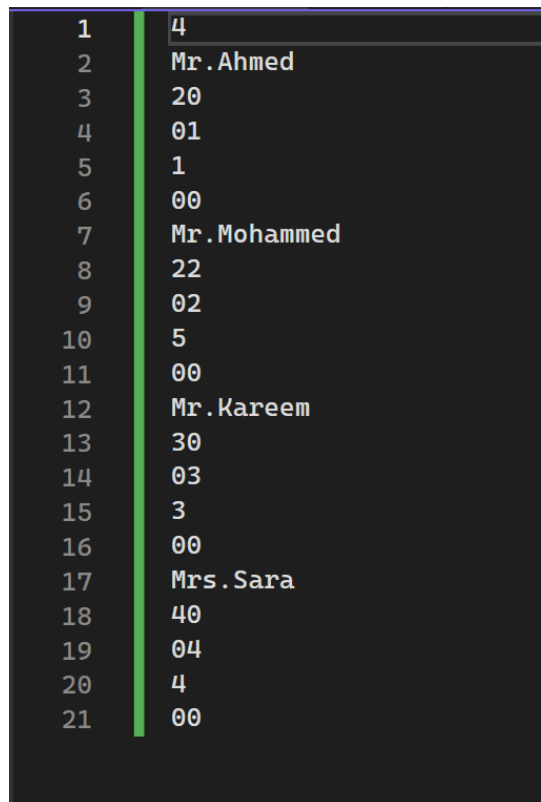
front to be on the start of the array index and a counter which is the numberOfItems. The function push will receive the value that is of type Type. it will check if the rear is at its maximum size if it is then it will go back to zero and start adding from the first, allowing the queue to function as a loop. If it is not at its maximum size then the rear will increase by one and x will be added to the array arr depending on the rear, which is the index number. The counter will increase since a new item was added. A pop function has also been declared and defined. A new value is created to allow the array value at a specific index to be added to it. If the front is at the maximum size it will also go back to zero, this shows that the front will move back before the rear, therefore, the rear will always be behind the front. Otherwise, if it is not at maximum size then the front will be incremented. And the counter will reduce by one since the items have been removed. This pop function will return x. Furthermore, the function created was isEmpty(), it will check the numberOfItems. If it is zero then it is true that there are no items and the array is empty otherwise return false. The final function is length, to return the numberOfItems which is the size of the array.

mechanics.txt:

```
1 3
2 Mai
3 40
4 1
5 3
6 1
7 00
8 5
9 30
10 3
11 15
12 Ayman
13 45
14 2
15 3
16 2
17 05
18 9
19 45
20 3
21 30
22 Khaled
23 55
24 3
25 3
26 1
27 00
28 6
29 45
30 2
31 30
32
```


Mechanic.txt is a text file that will be opened by the program, it first contains the number of mechanics. The name, age, and ID are shown correspondingly, the number of appointments they already have scheduled, and finally, the timings in which these appointments are at. Ayman and Khaled have the same thing in the exact same order. This order needs to be kept in the exact same format for the program to work properly.

customer.txt:



1	4
2	Mr . Ahmed
3	20
4	01
5	1
6	00
7	Mr . Mohammed
8	22
9	02
10	5
11	00
12	Mr . Kareem
13	30
14	03
15	3
16	00
17	Mrs . Sara
18	40
19	04
20	4
21	00

Customer.txt shows the number of customers that would like to be assigned to a mechanic, it starts with the name of the customer, the age, ID, and the time they want. This is the same for all the customers with exactly the same format.

main.cpp:

```
1 #include "Customer.h"
2 #include "Mechanic.h"
3 #include "Queue.h"
4 #include <fstream>
5
6 int main() {
7     ifstream instream("mechanics.txt"); //opens and reads the file mechanic.txt
8     if (instream.fail()) { //check if file fails to open
9         cout << "failed to open the file" << endl;
10        return 0;
11    }
12
13    int size; //first it will read the number of mechanics
14    instream >> size; //input the size
15    Mechanic *mech = new Mechanic[size]; //create an array of mechanic object
16    string name, ID;
17    int age;
18    int M, hour, mins;
19    // loop on the mechanics one by one
20    for (int i = 0; i < size; i++) {
21        instream >> name; //input into name
22        mech[i].setName(name);
23        instream >> age; //input into age
24        mech[i].setAge(age);
25        instream >> ID; //input into ID
26        mech[i].setID(ID);
27        instream >> M; //Read number of appointments
28        // loop M iterations to read all appointments
29        for (int j = 0; j < M; j++) {
30            instream >> hour; //input into hour
31            instream >> mins; //input into mins
32            mech[i].addAppointment(hour, mins); //add hour and mins into the mechanic array
33        }
34    }
35    instream.close(); //close the file
36    cout << "Available Mechanic List : " << endl;
37    for (int i = 0; i < size; i++) {
38        mech[i].printInfo(); //print the mechanic information
39    }
40    cout << endl;
41    instream.open("customers.txt"); //open customer file
42    if (instream.fail()) { //check if file fails to open
43        cout << "failed to open customer file" << endl;
44        return 0;
45    }
46    // first read number of customers
47    int NCustomer;
48    instream >> NCustomer; //input number of customers
49    Customer *customers = new Customer[NCustomer]; //array of customers
50    for (int i = 0; i < NCustomer; i++) { //input in every index depending on size the name, age and ID
51        instream >> name;
52        customers[i].setName(name);
53        instream >> age;
54        customers[i].setAge(age);
55        instream >> ID;
56        customers[i].setID(ID);
57        instream >> hour; // read data for hour and minutes in appointment
58        instream >> mins;
59        Appointment app; //assign appointment to customer
60        app.hour = hour;
61        app.mins = mins;
62        customers[i].setAppointment(app); //add appointments into the array of customers
63    }
64    instream.close(); //close the file
65
66    //bubble sorting depending on the appointment
67    for (int i = 0; i < NCustomer - 1; i++) { //using the < operating overload
68        for (int j = 0; j < NCustomer - 1; j++) {
69            if (customers[j] > customers[j + 1]) {
70                swap(customers[j], customers[j + 1]);
71            }
72        }
73    }
74
75    Queue<Customer> QCustomers;
76    for (int i = 0; i < NCustomer; i++) {
77        QCustomers.push(customers[i]); //loop on array and add it to the queue called QCustomers
78    }
79
80    Queue<Mechanic> QMechanic;
81    for (int i = 0; i < size; i++) {
82        QMechanic.push(mech[i]); //loop on the mechanic and add it to the que called QMechanic
83    }
84
85    cout << "The Customers are: " << endl;
86    while (!QCustomers.isEmpty()) { //Loop on Queue
87        Customer t = QCustomers.pop(); //read the first customer and remove from the queue
88        for (int i = 0; i < QMechanic.length(); i++) {
89            Mechanic c = QMechanic.pop(); //loop on mechanic to find the first available
90            if (c.isAvailable(t.getAppointment())) { //to check if mechanic is available by checking the appointment of customer
91                t.setMechanicID(c.getID()); //if its true then send the mechanics ID to the customers Mechanic ID
92                t.printInfo(); //print the info
93                QMechanic.push(c); //return the ID to the queue
94                break;
95            }
96            QMechanic.push(c); //to not make it repetitive
97        }
98    }
99 }
100 }
```

The main starts with opening mechanic.txt. To do this a library called fstream must be included, to allow input from the file, ifstream will be used. The if statement is used to check if the file will fail to open if not then the program will run normally. The first thing that will be imputed from the file will be the size of mechanic. This will be used to create a dynamic array called mech with the number of mechanics. After doing so then the name, ID, and age will be input using a for loop to enter the name, age, and ID, and the number of appointments already scheduled for each mechanic in one index of the array mech. After all of the names, ID's, age and number of appointments have been imputed, there will be another for loop to use the number of appointments as the size and input all of the hours and minutes into the index using the addAppointment function. The file will then be closed. This means that the array index has all been organized each containing a mechanic and its information. To display this information a for loop will be used to pass through all of the indexes of the array and use the function printInfo(), which was created in the mechanic class to display the mechanic's information. The customer.txt will then be opened and if it fails to open a statement will be given. Otherwise, the number of customers will be read and imputed into the variable NCustomer. A new dynamic array will be created by doing this. A for loop will be made using the NCustomer and the name, age, and ID, with the hour and minutes that they want an appointment. The hour and minute will then be assigned to an object called app of type Appointment. This will be used with the function setAppointment to be added to the customer's array. The file will then be closed. To allow the display to be shown in order of appointments it will need to be sorted. A bubble sort is used and the smaller-than-operating overload is used. This is to sort specifically the appointment. It will then be added to a Queue that was created by the Queue.h The type for a new variable that will be used is customers. This is because inputting the customer will require all of the attributes and

functions of the class customer. The for loop is used to input all of the customers of the array into the Queue called QCustomers. The same thing will be done with a queue of type mechanic called QMechanic. This way at the end there will be two different Queues one for the mechanic and the other for customer containing all of their information from the array. The while loop will be used to make sure that the queue is not empty, if it is not then the customers will be removed one by one from the queue into the customers called t. The for loop will be made depending on the length of mechanic and then it will also be removed using the pop function into c of type mechanic. Both of these will be used to be compared. An if function will be used to check if the mechanic is available using the function isAvailable. Depending on the appointment that the customer wants which will be the parameter for is Available by using the getAppointment function. If the mechanic isAvailable then the ID of the mechanic is going to be set into the setMechanicID of the customer. All of this information will then be printed using the printInfo() function from the customer.cpp file. The mechanic ID will then be returned to the queue using the push function. This function is written twice since the ID will be returned either way even if the mechanic is available or not.

The output:

```
> sh -c make -s
> ./main
Available Mechanic List :
Name : Mai
ID : 1
Age : 40
Number of appointments = 3
1 : 0
5 : 30
3 : 15

Name : Ayman
ID : 2
Age : 45
Number of appointments = 3
2 : 5
9 : 45
3 : 30

Name : Khaled
ID : 3
Age : 55
Number of appointments = 3
1 : 0
6 : 45
2 : 30

The Customers are:
Name : Mr.Ahmed
ID : 01
Age : 20
You are appointed to the mechanic with this ID: 2 at 1:0

Name : Mr.Kareem
ID : 03
Age : 30
You are appointed to the mechanic with this ID: 3 at 3:0

Name : Mrs.Sara
ID : 04
Age : 40
You are appointed to the mechanic with this ID: 1 at 4:0

Name : Mr.Mohammed
ID : 02
Age : 22
You are appointed to the mechanic with this ID: 2 at 5:0
> █
```

This is the output of the program. It first shows the list of Mechanics, their name, ID, age, number of appointments, and the appointments they already have scheduled. Then it will show who the customers are. They will have the name, ID, Age, and who they are appointed to depending on the ID of the mechanic. Both the Mechanic and Customer classes have the same first 3 outputs since they are inherited from the class Person. An example of how the mechanic ID is used is that Ayman has ID 2, therefore, Mr. Ahmed was appointed to 2 at 1 since Mai was not free at that time slot and Ayman is free at that time. The rest of the Customers have the

same format and they all have been organized by the time they want their appointment showing 1:0, 3:0, 4:0, and 5:0 when their names and times were not ordered in this way in the text file. The output of the “Available Mechanic List :” is from the printInfo function that was made during the class mechanic and it printed everything that is found in the mech array. While from the start of “The customers are: “ is from the printInfo

function that is found in the customer class which shows who the customer is appointed to.