EE3093 – C/C++ Programming
C/C++ Programming Assignment
(60% of the overall course grade)
Due 18/12/2021 before 6:00 pm

- This assignment is worth 60% of your overall course grade.
- Two parts (1) C Part & (2) C++ Part are in this assignment. Total 100 marks is equally divided as 50 marks for C Part and 50 marks for C++ Part.
- For the C Part submit the source code as the report. For the C++ Part follow the instructions in the C++ Part instructions.
- Submit the source code (.c file) as the report.
- Submit the signed Plagiarism Sheet.
- Submit typed report (*.c source file). Handwritten / scanned report will not be graded.
- Attach a signed Plagiarism Sheet with your submission. Failing to do so or adhere to plagiarism rules as set by the University will lead to appropriate penalties.
- Files must be uploaded on MyAberdeen via the relevant link.

C Part
Implementation of Algorithm to Compute the Peak-Power and Full-Width at Half-Maximum (FWHM) of a Pulse in C Programming Language

Temporal domain pulse data is available as a sequential file named `pulse.txt` with time vector (1D data) in the first column and pulse power vector in the second column. Plot the data in `pulse.txt` using MATLAB or Excel to get an idea of how the pulse looks in the temporal domain. Develop a C program for the implementation of algorithm to compute the peak-power and Full-Width at Half-Maximum (FWHM) of the pulse after reading the data from the `pulse.txt` file. Following steps can help to develop and implement the algorithm:

- Using `fopen` function, open the `pulse.txt` sequential file and assign its address to a `FILE` pointer with checking condition for successfully opening the file. If the sequential file is successfully opened for reading, `FILE` pointer will point to the beginning of the `pulse.txt` sequential file.
- Store the location (beginning of the `pulse.txt` sequential file) pointed by the `FILE` pointer, as the algorithm needs the `FILE` pointer to point to the beginning file once again, after counting the number of data points. This can be achieved by `ftell` function.
- Count the number of \n (new line character) while reading the characters in the `pulse.txt` sequential file using `getc` function until EOF (End-Of-File). This count will provide the number of data points in the `pulse.txt` sequential file.
- Using `fseek` function, make the `FILE` pointer to point to the beginning file once again.
- Using the number of data points in `malloc` function, dynamically allocate the required amount of memory to store the time and pulse power data.
- Using `fscanf` function, read and store the time and pulse power data from the `pulse.txt` file.
- Scan through the pulse power data to find the peak-power (maximum value of the pulse power).
- Scan through the left-side and right-side of the peak-power position in the pulse power data points to find the locations, where the pulse power goes just below the half of the peak-power value.
- Compute the left-side and right-side average time data values using the respective two-time data points (above and below), where the pulse power goes just below the half of the peak-power value.
- FWHM of the pulse can be calculated using either the sum (if both sides averaged time value has opposite sign) or difference (if both sides averaged time value has same sign) of the modulus (use `fabs` function) of the left-side and right-side average time data values.

Details of the percentage of the marks:
1. Declaration of necessary variables, pointers, arrays, structures, function prototypes [3 marks].
2. Opening sequential file with necessary checking condition for successful access [4 marks].
3. Correctly counting the number of data points in the sequential file [7 marks].
4. Allocation of necessary 1D arrays memory to store time and pulse power data [3 marks].
5. Computing the peak-power of the pulse [5 marks].
6. Locating the left and right sides data positions where pulse power goes below its peak [7 marks].
7. Calculation of left and right sides two-time data points average values using a general function written for the mean value calculation of any number of data [7 marks].
8. Correctly calculating the pulse FWHM [5 marks].
9. Display of the number of data points, pulse peak-power and FWHM [3 marks].
10. Release of the dynamically allocated memory for the arrays and closing file [3 marks].
11. Structured programming and adding comments about the algorithm implementation [3 marks].

Sample output:
```
Number of sampled points is 1024
Pulse peak power is 0.017421
Pulse FWHM is 29.785156
```

EE3093: C++   Instructions to Students

UNIVERSITY OF ABERDEEN   SESSION 2021-22

EE3093: Cpp part (50 marks out of 100)

Degree Examination in EE3093 C/C++ PROGRAMMING
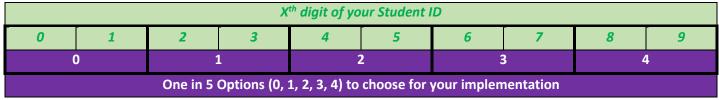
First Half Session: 2021-22

This is an open book online exam which is expected to take up to 6 hours to complete but submission will be accepted, without incurring any penalties, until the above submission deadline; students who have not submitted after this time will be required to present for reassessment in the resit diet.

Instructions to users

- Create a **new directory** and **create** there a **new C++ project (console application)**, as shown in labs/tutorials.
- When creating header/cpp files for your project, use **file names that clearly identify** this assignment, the section they refer to and the class they implement; for **example**: "cpp_part1_classname1.h", "cpp_part1_classname1.cpp" (if needed), "cpp_part1_classname2.h", "cpp_part1_test.h" etc.

In this document, specific requirements for your implementation are set depending on your student ID, based on the Table below. All details are provided in each section of this paper.

Table 1: Relation between **digit X** of your **student ID** and the **option for your implementation** <u>out of 5 choices</u>.

X^{th} digit of your Student ID									
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0		1		2		3		4	
One in 5 Options (0, 1, 2, 3, 4) to choose for your implementation									

Example assuming the student ID is: *51807834*:

For the 8th digit (*4*) the option to choose is **Option 2**. For the 7th digit (*3*) the option to choose is **Option 1**.
For the 6th digit (*8*) the option to choose is **Option 4**. For the 5th digit (*7*) the option to choose is **Option 3**.
For the 4th digit (*0*) the option to choose is **Option 0**.

Submission info

The assessment in this document is developed across sections that test increasing levels of knowledge and skills; make sure you have understood, implemented and tested one section before moving on to the next.

Your **submission** via myAberdeen **for the CPP part** should include:

- **The Student Selection Document** (provided with submission material), **reporting** your **selections**, based on your student ID (as instructed throughout this document), **and test outputs**.
- All relevant **source code,** in clearly-named **separate (.h/.cpp) files**. Please **indicate in** the relevant tables of the Student Selection Document the name of the file(s) containing the implementation and testing. Only files named in that document will be considered for marking. You can submit all files as a single zipped file.

Recommended completion, backup and submission strategy:
<u>Start working on the day the assignment is released</u>; if you can setup, compile and run an early version of your cpp project (even if it only implements one of the required functions), you can ensure your HW and SW are working properly. If a technical problem arises, you can then contact course staff readily.

If you work on your own PC (not via VDI): As you progress with your answers, <u>save all material</u> (cpp/h files and PDF), on an external repository. For example: at the end of each day of work, you can zip all material (source files and PDF: no executables) finalized up to that day and email it to yourself. If your PC fails on a given day, you will not lose multiple days of work.

You will submit your assignment via MyAberdeen. During the submission period, MyAberdeen accepts multiple submissions. When you are satisfied with your answer, submit all material (source files and PDF). If you then improve your answers before the deadline, you can **submit again (all source files and PDF);** this later submission will be marked.
Do not wait till the deadline to make your one and only submission, as unexpected technical problems may lead you to miss the submission deadline entirely.

Part 1: Flight Timetable basic components (28 marks out of 50)

Introduction

Implement SW objects (classes) to create an airport daily timetable. An entry in an airport timetable comprises the fields in Table 2. The <u>first class to implement</u> is **FlightTimetableEntry** and represents a (potential) <u>entry in airport timetables</u>. The **Basic Rules** that apply to a valid <u>airport timetable entry</u> are in Table 4 (based on your student ID).

Table 2: Fields of a *Flight Timetable Entry*.

Field	Description and/or Possible values
Origin	A city amongst: {*Aberdeen*, *London*, *Manchester*, *Copenhagen*, *Esbjerg*}
Destnation	A different city amongst: {*Aberdeen*, *London*, *Manchester*, *Copenhagen*, *Esbjerg*}
Airline_ID	One of: {BA, SK, KL, EZY, LM}
Flight_Code	A four-digit number: **XYZW** following the <u>rules in Table 4</u>
Scheduled_Departure	**Local time**, format **hh:mm**; within **00:00** and **23:59**
Scheduled_Arrival	**Local time**, format hh:mm, as per flight duration; **local time in DNK = local time in UK +1hr**
Expected Duration	Format hh:mm; as per <u>flight duration in Table 3</u>

Table 3: Flight duration hh:mm. *Scheduled Departure* and *Arrival* fields must be consistent with the given flight duration.

		Destination				
		Aberdeen	*London*	*Manchester*	*Copenhagen*	*Esbjerg*
Origin	*Aberdeen*	-	**1:35**	**1:20**	**1:35**	**1:25**
	London	**1:35**	-	**1:00**	**1:45**	**1:35**
	Manchester	**1:20**	**1:00**	-	**1:40**	**1:30**
	Copenhagen	**1:35**	**1:45**	**1:40**	-	**0:50**
	Esbjerg	**1:25**	**1:35**	**1:30**	**0:50**	-

Table 4: Rules for *Flight Code* and *Departure Time* fields. Your **Option number** is based on your **student ID 8th digit** as in **Table 1**.

Basic_Rules Option 0					
	Aberdeen (UK)	*London (UK)*	*Manchester (UK)*	*Copenhagen (DK)*	*Esbjerg (DK)*
Departing Flight (local) time rule	Departures within 5:30 and 21:00	Departures within 4:30 and 21:10	Departures within 5:00 and 21:05	Departures within 5:00 and 21:45	Departures within 6:00 and 21:00
THIS	First flight code digit (X): 0 for domestic flights (origin & destination in the same country); 5 for international flights.				
Flight code (XYZW) rules	Second flight code digit (Y): 1 for flights departing and arriving before noon (local time); 6 for flights departing and arriving after 18:00 (local time); 3 for all other flights.				
	Third and fourth flight code digits (ZW): random number in the following ranges for each airline: *BA*: [00-19], *SK*: [20-39], *KL*: [40-59], *EZY*: [60-79], *LM*: [80-99]				

Basic_Rules Option 1					
	Aberdeen (UK)	*London (UK)*	*Manchester (UK)*	*Copenhagen (DK)*	*Esbjerg (DK)*
Departing Flight (local) time rule	Departures within 5:00 and 20:50	Departures within 5:00 and 21:05	Departures within 5:15 and 20:55	Departures within 5:00 and 21:45	Departures within 5:45 and 21:30
Flight code (XYZW) rules	First flight code digit (X): 0 for domestic flights departing before noon; 2 for international flights departing before noon (local time); 7 for domestic flights departing after noon; 9 for international flights departing after noon (local time).				
	Second flight code digit (Y): 0 for flight duration up to (and including) 1:30; 5 for longer flights.				
	Third and fourth flight code digits (ZW): random number in the following ranges for each airline: *BA*: [80-99], *SK*: [00-19], *KL*: [20-39], *EZY*: [40-59], *LM*: [60-79]				

Basic_Rules Option 2					
	Aberdeen (UK)	*London (UK)*	*Manchester (UK)*	*Copenhagen (DK)*	*Esbjerg (DK)*
Departing Flight (local) time rule	Departures within 5:30 and 20:45	Departures within 5:00 and 21:00	Departures within 5:30 and 21:05	Departures within 5:00 and 22:00	Departures within 5:00 and 22:00
Flight code (XYZW) rules	First flight code digit (X): **0** for domestic flights (origin & destination in the same country) lasting less than 1:00; **2** for domestic flights lasting 1:00 to 1:30; **3** for domestic flights lasting more than 1:30. **4** for international flights lasting less than 1:40; **5** international flights lasting 1:40 or longer.				
	Second flight code digit (Y): *BA:* 0, *SK:* 2*, KL*: 4*, EZY*: 6*, LM*: 8.				
	Third and fourth flight code digits (ZW): random number in the range: [00-99]				

Basic_Rules Option 3					
Departing from	*Aberdeen (UK)*	*London (UK)*	*Manchester (UK)*	*Copenhagen (DK)*	*Esbjerg (DK)*
Departing Flight (local) time rule	Domestic flight (origin & destination in the same country) departures within 6:00 and 22:30. International flight departures within 5:00 and 21:00.				
Flight code (XYZW) rules	First flight code digit (X): 1 for domestic flights; 9 for international flights.				
	Second flight code digit (Y): 1	Second flight code digit (Y): 3	Second flight code digit (Y): 5	Second flight code digit (Y): 7	Second flight code digit (Y): 9
	Third and fourth flight code digits (ZW): even/odd random number in the following ranges for each airline: *BA/SK:* even/odd in [00-39], *KL/EZY*: even/odd in [40-79]*, LM*: even in [80-99]				

Basic_Rules Option 4					
	Aberdeen (UK)	*London (UK)*	*Manchester (UK)*	*Copenhagen (DK)*	*Esbjerg (DK)*
Departing Flight (local) time rule	Domestic flight (origin & destination in the same country) departures within 6:00 and 23:00. International flight departures within 5:30 and 21:10.			Domestic flight within 6:30 and 22:30. International flight departures within 6:00 and 22:00.	
Flight code (XYZW) rules	First flight code digit (X): 2 for domestic flights; 8 for international flights.			First flight code digit (X): 3 for domestic flights; 8 for international flights.	
	Second flight code digit (Y): <u>Departing before 11:00</u>: 2 for flight duration under 1:30; 3 for longer. <u>Departing at 11:00</u>: 7 for flight duration under 1:30; 8 for longer				
	Third and fourth flight code digits (ZW): even/odd random number in the following ranges for each airline: *EZY/LM:* even/odd in [00-39], *BA/KL*: even/odd in [40-79]*, SK*: even [80-99]				

Details of the required implementation

Your class **FlightTimetableEntry** should support **the public member functions** in Table 5, along with any other functions/variables required in your implementation.

A class supporting the functionality required for **FlightTimetableEntry** constitutes the essential building block for the implementation in part 2.

The functions in Table 5 are assumed in the test routine in Table 6, which form the basis of the tests required to verify and demonstrate the correct behaviour of your implementation. Include your test results (sample in Figure 1) in the **Student Selection Document** (follow the instructions therein).

Table 5: Required public member functions for class **FlightTimetableEntry**. **Examples** of inputs in Table 6 and output in Figure 1. **Marks breakdown** in Table 7.

Function declaration	Description
Setters (input functions)	

`// type of inputs "origin", and "destination" is a specifically defined enum;`
`// similarly for "airlineID";`
`// departure time is given by dept_hour:dept_min`
`bool checkAndSetFTE(AirportEnum orig, AirportEnum dest, AirlinesEnum airID, int flCode, unsigned char dept_hour, unsigned char dept_min`
	If the entry is not already set: <u>if all input values</u> (origin, destin., airline ID, flight code, dept hour, dept minute) **comply with** the **Basic Rules,** then **set all the fields** of the entry; **return true**. **If any rule is violated**, **return false** and print an **error message**.
	<u>If the entry is already set</u>: print an error message and return **false** (must use the reset() function before setting)

`bool setRandomFTE()`	<u>If the entry is not already set</u>: set **all fields** with **random values** that **comply with** the **Basic Rules**. <u>If the entry is already set</u>: print an error message and return **false** (must use the reset() function before setting)
`void reset()`	Resets **all the fields** of the Timetable entry.
Getters (output functions)	
`// type of input parameters as in setFTE(…);` `bool getFTE(AirportEnum& orig, AirportEnum& dest, AirlinesEnum& airID, int& flCode, unsigned char& dept_hr, unsigned char& dept_min, unsigned char& arriv_hr, unsigned char&arriv_min, unsigned char& durat_hr, unsigned char& durat_min) const`	<u>If the entry is set</u>: **input variables** (passed by reference) are set with **entry field values** as in **Table 2**; **return true**. <u>If the entry is not set</u>: return **false.**
`void printEntry() const`	<u>If the entry is set</u>: print to scree; Otherwise error message.
Utilities (functions that help implement other functions; implemented as *static member function* because an underlying class instance is not needed)	
`static bool getFlightDurationTime(AirportEnum orig, AirportEnum dest, unsigned char& out_hrs, unsigned char& out_min)`	<u>If origin and destination are valid</u>: **input variables** (hours and minutes) are **set with flight duration** as in Table 3; **return true**. <u>Otherwise</u>: print message; return **false.**
`static bool getFlightArrivalTime(AirportEnum orig, AirportEnum dest, unsigned char dpt_hrs_loc, unsigned char dpt_min_loc, unsigned char& arr_hrs_loc, unsigned char& arr_min_loc)`	<u>If origin and dest. are valid</u>: **input variables** (pass by reference) are set with **local time** arrival (hours and min.) based on flight duration (Table 3); **return true**. <u>Otherwise</u>: print message; return **false.**
`static bool checkEntryVals(AirportEnum orig, AirportEnum dest, AirlinesEnum airID, int flCode, unsigned char dp_hr, unsigned char dp_min)`	<u>If all input values comply with</u> the **Basic Rules,** then **return true**. **If any rule is violated**, **return false** and print an **error message**.

`static string value2string(AirlinesEnum airline)`
`static string value2string(AirportEnum airp)`
`static string value2string(unsigned char hour, unsigned char minute)`
`static string value2string(int flCode)`
	Output a string for the corresponding input value. Examples:
	Input AirlinesEnum is BA: out string is "BA" or "British Airways";
	Input AirportEnum is London: out string is "London";
	Inputs are hour=9, min=5: out string is "09:05";
	Input flightcode is 502: string is "0502"

`static int RandomValInBounds(int min_val, int max_val)`	Output is a random integer within: [min_val, max_val]

Required Testing

Test your implementation **with the test routines below**. Each routine requires minimal modification to be **adapted** to your implementation and **Basic Rules Option**; comment out any function calls that your implementation does not support yet. Call from **main()**; include all necessary header files. Report your test results as indicated in

the The functions in Table 8 are assumed in the test routine in Table 6, which form the basis of the tests required to verify and demonstrate the correct behaviour of your implementation. Include your test results (sample in Figure 1) in the **Student Selection Document** (follow the instructions therein).

Table 6: Test routines for class **FlightTimetableEntry**. Sample Outputs in **Figure 1**.

```cpp
void test_helpers()
{
    FlightTimetableEntry::AirportEnum orig, dest;
    unsigned char dur_hrs, dur_min, dpt_hrs, dpt_min, arr_hrs, arr_min;
    FlightTimetableEntry::AirlinesEnum airID;
    bool result; int flightCode;
    /////////////////// Input values for Test /////////////////////////////////
    orig = FlightTimetableEntry::Aberdeen; dest = FlightTimetableEntry::Esbjerg;
    dpt_hrs = 9; dpt_min = 5; airID = FlightTimetableEntry::BA;
    flightCode = 5100 + FlightTimetableEntry::RandomValInBounds(0,19);
    //  -----------------------------------------------------------------------//
    cout << "Test values: " << endl; cout << "Flight origin: " << FlightTimetableEntry::value2string(orig);
    cout << ".  Flight destination: " << FlightTimetableEntry::value2string(dest) << endl;
    cout << "Test getFlightDurationTime()" << endl;
    result = FlightTimetableEntry::getFlightDurationTime(orig, dest, dur_hrs, dur_min);
    if (result)
        cout << "Flight duration (hh:mm) is " << FlightTimetableEntry::value2string(dur_hrs, dur_min) << endl;
    else
        cout << "No flight duration found" << endl;
    /////////////////////////////////////////////////////////////
    cout << endl << "Test getFlightArrivalTime()" << endl;
    cout << "Flight departure time (hh:mm) is " << FlightTimetableEntry::value2string(dpt_hrs, dpt_min) << endl;
    result = FlightTimetableEntry::getFlightArrivalTime(orig,dest,dpt_hrs,dpt_min,arr_hrs,arr_min);
    if (result)
        cout << "Arrival (local) time (hh:mm) is " << FlightTimetableEntry::value2string(arr_hrs, arr_min) << endl;
    else
        cout << "No Arrival time found" << endl;
    /////////////////////////////////////////////////////////////
    cout << endl << "Test checkEntryVals()" << endl;
    cout << "Airline ID: " << FlightTimetableEntry::value2string(airID);
    cout << ". Flight code: " << FlightTimetableEntry::value2string(flightCode) << endl;
    result = FlightTimetableEntry::checkEntryVals(orig, dest, airID, flightCode, dpt_hrs, dpt_min);
    if (result)
        cout << "Flight entry is valid." << endl;
    else
        cout << "Flight entry NOT valid." << endl;
}
```

Table Continued overleaf

```cpp
void test_SettersAndGetters()
{
    FlightTimetableEntry testEntry; FlightTimetableEntry::AirportEnum orig, dest;
    unsigned char dur_hrs, dur_min, dpt_hrs, dpt_min, arr_hrs, arr_min;
    FlightTimetableEntry::AirlinesEnum airID; bool result; int flCode;
    ///////////////////// Input values for Test /////////////////////////////////////
    orig = FlightTimetableEntry::Aberdeen; dest = FlightTimetableEntry::Esbjerg;
    dpt_hrs = 9; dpt_min = 5; airID = FlightTimetableEntry::BA;  flCode = 5100;
    //  -------------------------------------------------------------------------//
    cout << "Test values: " << endl << "Flight origin: " << FlightTimetableEntry::value2string(orig);
    cout << ".  Flight destination: " << FlightTimetableEntry::value2string(dest) << endl;
    cout << "Flight departure time (hh:mm) is " << FlightTimetableEntry::value2string(dpt_hrs, dpt_min) << endl;
    cout << "Airline ID: " << FlightTimetableEntry::value2string(airID);
    cout << ". Flight code: " << FlightTimetableEntry::value2string(flCode) << endl;
    cout << "First, Test checkAndSetFTE()" << endl;
    result = testEntry.checkAndSetFTE(orig, dest, airID, flCode, dpt_hrs, dpt_min);
    if (!result)
        cout << "FTE could not be set." << endl;
    else
    {
        cout << "Done: FTE set." << endl << endl <<"Now Test getFTE() :" << endl;
        orig = FlightTimetableEntry::Aberdeen; dest = FlightTimetableEntry::Aberdeen;// if all works, these values will not matter
        dpt_hrs = 0; dpt_min = 0; airID = FlightTimetableEntry::KL;  flCode = 9999; // if all works, these values will not matter
        result = testEntry.getFTE(orig, dest, airID, flCode, dpt_hrs, dpt_min, arr_hrs, arr_min, dur_hrs, dur_min);
        if (result)
        {
            cout << "Airline ID: " << FlightTimetableEntry::value2string(airID);
            cout << ". Flight code: " << FlightTimetableEntry::value2string(flCode) << endl;
            cout << "Flight origin: " << FlightTimetableEntry::value2string(orig);
            cout << ".  Flight destination: " << FlightTimetableEntry::value2string(dest) << endl;
            cout << "Flight departure time (hh:mm) is " << FlightTimetableEntry::value2string(dpt_hrs, dpt_min) << endl;
            cout << "Flight arrival (local) time (hh:mm) is " << FlightTimetableEntry::value2string(arr_hrs, arr_min) << endl;
            cout << "Flight duration (hh:mm) is " << FlightTimetableEntry::value2string(dur_hrs, dur_min) << endl;
            cout << "Done." << endl << endl << "Next check that the same result is obtained from printEntry():" << endl;
            testEntry.printEntry();
            cout << "Done." << endl << endl << "Finally reset:" << endl;
            testEntry.reset();
            testEntry.printEntry();
        }
        else
            cout << "getFTE() did not work" << endl;
    }
}
```

Figure 1: Sample output for **test_helpers()** on the left and **test_SettersAndGetters()** on the right (assuming **Basic_Rules Option** 0).

Marking:

Note: no mark awarded for implementing Basic_Rules different from the Option required (based on your ID).

Table 7: Marks allocation for Part 1 (out of 50 marks for the entire C++ assignment)**:**

Part 1	Marks (up to)
Class Implementation: well structured, tidy and clearly commented.	**3**
checkAndSetFTE(…): implementation and test (via test routine)	**3**
setRandomFTE(…): implementation and test (via test routine)	**3**
reset(): implementation and test (via test routine)	**1**
getFTE(…): implementation and test (via test routine)	**2**
getFlightDurationTime(…): implementation and test (via test routine)	**2**
getFlightArrivalTime(): implementation and test (via test routine)	**2**
checkEntryVals(…): implementation and test (via test routine)	**3**
printEntry(): implementation and test (via test routine)	**2**
value2string(…): implementation and test (via test routine)	**4**
getRandomVal(…): implementation and test (via test routine)	**1**
Reporting of Test results (as instructed in the Student Selection Document)	**2**
	Tot Part 1 = 28 Marks

Part 2: Airport Timetable (22 marks out of 50)

Use FlightTimetableEntry as a building block for your next class, **AirportFlightTimetable,** which represents all daily departures and arrivals in the timetable of **your assigned airport** in **Table 8**. Any timetable entry follows **the Basic Rules specified in the previous section**; in addition, the following **Airport Rules apply:**

- **Common Airport Rules** (apply to all airports): A timetable cannot contain **two entries** (flights) that are operated by the same Airline and have either of the following features:
    - have the same origin & destination **AND** have the same departure/arrival time.
    - have the same flight-code (regardless of the origin/destination and departure/arrival time)
- **Specific Airport Rules:** These constrain: (*i*) the number of (international/local) departures/arrivals in a day and in an hour; (*ii*) the minimum temporal separation between flights with given characteristics. All details are given in Table 9 and are based on your student ID.

Table 8: Airport assignment. Your **Option number** is based on your **student ID 7th digit** as in **Table 1**

Aberdeen	London	Manchester	Copenhagen	Esbjerg
Option 0	**Option 1**	**Option 2**	**Option 3**	**Option 4**

Table 9: Rules for your **assigned airport (flight origin/destination). Option number** based on **student ID 6th digit** as in **Table 1**

Airport_Rules Option 0	
Max number of Departing Flights in a day	30
Max number of Arriving Flights in a day	30
Max number of Departing Flights in any one hour (from **xy:00** to **xy:59**)	5
Minimum time separation between any two departing flights with the same destination	15 min
Max number of Departing **International** Flights in any one hour (from **xy:00** to **xy:59**)	3

Airport_Rules Option 1	
Max number of Departing Flights in a day	40
Max number of Arriving Flights in a day	40
Max number of Arriving Flights in any one hour (from **xy:00** to **xy:59**)	6
Minimum time separation between any two flights arriving from the same origin	10 min
Max number of Arriving **International** Flights in any one hour (from **xy:00** to **xy:59**)	4

Airport_Rules Option 2	
Max number of Departing Flights in a day	35
Max number of Arriving Flights in a day	35
Max number of Departing Flights in any one hour (from **xy:00** to **xy:59**)	6
Max number of Arriving Flights in any one hour (from **xy:00** to **xy:59**)	5
Minimum time separation between any two Departing **International** flights	10 min

Airport_Rules Option 3	
Max number of Departing Flights in a day	32
Max number of Arriving Flights in a day	36
Minimum time separation between any two departing flights with the same destination	10 min
Minimum time separation between any two flights arriving from the same origin	15 min
Minimum time separation between any two Arriving **International** flights	20 min

Airport_Rules Option 4	
Max number of Departing Flights in a day	38
Max number of Arriving Flights in a day	34
Max number of Departing Flights in any one hour (from **xy:00** to **xy:59**)	4
Minimum time separation between any two flights arriving from the same origin	12 min
Max number of Departing **International** Flights in any one hour (from **xy:00** to **xy:59**)	2
Max number of Arriving **International** Flights in any one hour (from **xy:00** to **xy:59**)	2

Details of the required implementation

Your class **AirportFlightTimetable** should support **the public member functions** in Table 10, **along with any other functions/variables required** in your implementation.

Table 10: Required public functions for **AirportFlightTimetable**. **Sample** input/output in Table 11 & Figure 2. **Marks** in **Table 11**.

Function declaration	Description
`bool checkEntryIsValid(const FlightTimetableEntry& test_entry, bool verbose) const`	Verify that an entry complies with both **Common** and **Specific** Airport Rules. If it does: return *true*; *false* otherwise. If *vebose*: print to screen info on which rule is broken.
`bool checkAndAddEntry(const FlightTimetableEntry& test_entry)`	If the entry is valid, add to the timetable and return *true*; otherwise: *false*.
`void printTimetable(tmtblCaseEnum ArrOrDepartOrBoth) const`	Print the Arrival/Departure timetable to screen (sorted by Arrival/Departure time); input selects which timetable to print (Arrival/Departures/Both)
`bool isTimetableFull() const`	True if the timetable is full; false otherwise.

Required Testing
Test your implementation **with the test routine given below** to ensure it is correct. The routine requires minimal modification to be adapted to your implementation.

Table 11: Test routines for class **AirportFlightTimetable**. Sample Outputs in **Figure 2**

```cpp
void testInsertion()
{
    FlightTimetableEntry testEntry; FlightTimetableEntry::AirportEnum orig, dest, test_airport=FlightTimetableEntry::Aberdeen;
    unsigned char dur_hrs, dur_min, dpt_hrs, dpt_min, arr_hrs, arr_min; FlightTimetableEntry::AirlinesEnum airID; bool result;
    int flCode; AirportFlightTimetable ABD_Timetable; bool check; int flCode, tot_tests = 50;
    for (int test_i = 0; test_i < tot_tests; test_i++)
    {
        if (ABD_Timetable.isTimetableFull())
            break;
        cout << "Random entry " << test_i << ":" << endl;
        while (true)
        {
            testEntry.reset(); testEntry.setRandomFTE();
            if (testEntry.isEntrySet())
            {
                testEntry.getFTE(orig, dest, airID, flCode, dpt_hrs, dpt_min, arr_hrs, arr_min, dur_hrs, dur_min);
                if (orig == test_airport || dest == test_airport)
                {
                    cout << "Checking entry:" << endl; testEntry.printEntry();
                    check = ABD_Timetable.checkEntryIsValid(testEntry, true);// print to screen the (first) rule that is violated
                    if (check)
                    {   // insert entry and print
                        cout << endl << "Entry is valid: Inserting Entry" << endl; check = ABD_Timetable.checkAndAddEntry(testEntry);
                        if (check)
                        { cout << "Entry Added" << endl; ABD_Timetable.printTimetable(); }
                        else
                        { cout << "Inserting Entry FAILED" << endl; return;}
                        break;
                    }
                    else
                        cout << "Check Failed: Entry can't be added to current timetable; try a different one" << endl << endl;
                }
            }
            else
                cout << "Entry Not set" << endl;
        }
        cout << "----------------" << endl;
    }
}
```
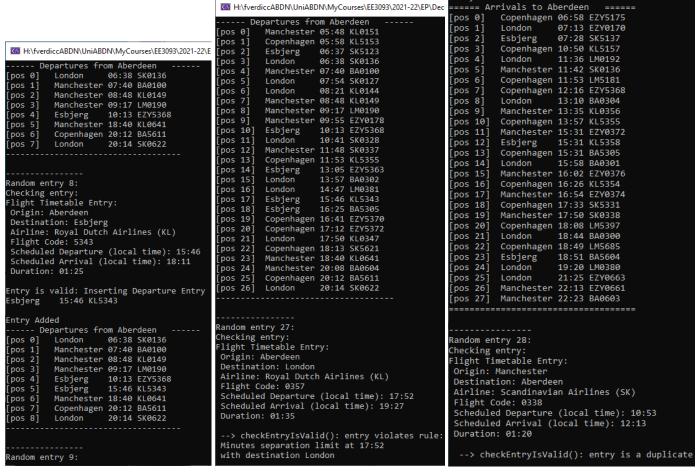
Figure 2: Output of random entry generation and insertion. **Left:** successful insertion of an entry; **Middle:** attempt to insert an entry that violates one Airpot_Rule (assuming Airport_Rules Option 0). **Right:** attempt to insert an entry that violates a Common_Airport rule (same airline and flight code of an entry already in the timetable: position 19).

Marking:

Note: no mark awarded for implementing Airport_Rules different from the Option required (based on your ID).

Table 12: Marks allocation for Part2 (out of 50 marks for the entire C++ assignment)**:**

Part 2	Marks (up to)
Class Implementation: well structured, tidy and clearly commented.	3
checkEntryIsValid (…): implementation and test (via test routine)	8 3 if Common Airport rule correct 5 if Specific Airport rule correct
checkAndAddEntry (…): implementation and test (via test routine)	4
printTimetable (…): implementation and test (via test routine)	3
isTimetableFull (…): implementation and test (via test routine)	2
Reporting of Test results (as instructed in the Student Selection Document)	3
	Tot Part 2 = 22 Marks