EE273 CODING PROJECT

Project F: Student Workload (Life) Planner

Group 28

Arran Moffat – 201606898

Jiayi Zhang - 201720432

APRIL 25, 2018

UNIVERSITY OF STRATHCLYDE

1 Contents

2 Abstract

This report covers the implementation of the project F - “Student Workload (Life) Planner”. The report contains a detailed outline on how the project was planned and how this planning helped with the implementation. Extensive testing of the project is covered and the results are discussed as the testing occurs. Further potential developments of the code is also included to highlight how the project could be improved. A conclusion looking at the overall success of the project is also included.

3 Introduction

The aim of this project was to design and test a program which would generate a weekly timetable planner of the user’s life. An excel file would be used to allow the user to enter their weekly commitments and social activities and any information they wish to add. The program would then generate a timetable consisting of twelve hours, 09:00 to 21:00, for each day, Monday to Sunday. This would represent the user’s weekly schedule in an easy to read form. If the user had any alterations to make to their weekly schedule they could simply rerun the program to generate an updated program.

4 Project Description

The project title is “Student Workload (Life) Planner”, which was a project that required the creation of a weekly timetable for which the data such as activity name and the time it occurred at would be entered by a user. The user could enter either an activity which had a set start and end time such as a university lecture, or a personal activity which would be assigned to free time by the program. The finalised timetable for the user should be clear and understandable and should allow multiple users to generate a timetable. If any activities occur at the same time the user should be notified.

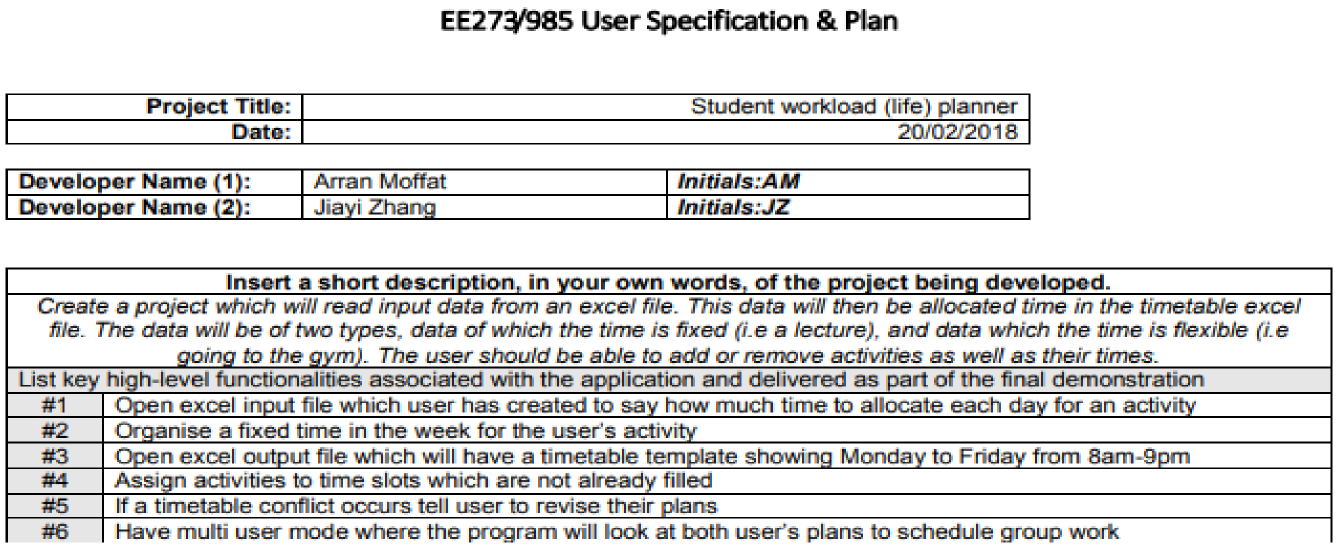
5 Planning

The project had several requirements which must be met in order to generate a functioning timetable. These include:

- The user should be allowed to enter their own activities and tasks.
- The user should be able to enter both time defined activities and time undefined activities.
- The user should also be able to alter their timetable at any time.
- The user should be able to enter both academic and non-academic tasks.
- A multi –user capability should be implemented to allow multiple users to create their own timetable.

Having looked at the task description and the above requirements, a user specification could be created which would highlight all the functionalities program would carry out for the user. The User Specification can be seen in Figure 1.

5.1 User Specification



EE273/985 User Specification & Plan

Project Title:	Student workload (life) planner
Date:	20/02/2018

Developer Name (1):	Arran Moffat	***Initials:AM***
Developer Name (2):	Jiayi Zhang	***Initials:JZ***

Insert a short description, in your own words, of the project being developed.

Create a project which will read input data from an excel file. This data will then be allocated time in the timetable excel file. The data will be of two types, data of which the time is fixed (i.e a lecture), and data which the time is flexible (i.e going to the gym). The user should be able to add or remove activities as well as their times.

List key high-level functionalities associated with the application and delivered as part of the final demonstration

#1	Open excel input file which user has created to say how much time to allocate each day for an activity
#2	Organise a fixed time in the week for the user's activity
#3	Open excel output file which will have a timetable template showing Monday to Friday from 8am-9pm
#4	Assign activities to time slots which are not already filled
#5	If a timetable conflict occurs tell user to revise their plans
#6	Have multi user mode where the program will look at both user's plans to schedule group work

Figure 1 - User Specification

Part one of the User Specification, indicated the program should be able to read in data from an excel file which would include an activity and a length of time it would last. It was initially decided an Excel file in the CSV (Comma-Separated Values) would be the most effective method of having the user enter their data. The alternative approach would be to have the user enter each activity into the command window. This would be an untidy and hard to read process where an incorrect entry would require the entire process to be stopped and restarted. Using the CSV format would allow the user to enter data neatly into an Excel spreadsheet which would in turn be converted into a text file with each Excel cell being separated by a comma. This would allow for editing of the users input data to be done quickly and neatly.

The second part of the specification documentation, states that the program must organise a fixed time for the activities the user enters into the Excel sheet. This means that program should read in the user defined data and assign it to free time slots throughout the course of a week.

Part three describes the use of an output file. The program would output the data the user had entered to an Excel file, also in the CSV format. This output file would clearly represent the data in the form of a timetable, showing the 7 days of the week and the times from 8am-9pm. The activities occurring would be represented at their scheduled time. A GUI (Graphical User Interface) was the alternative option to using an Excel file to output the data. It was decided however that the Excel file would be the best option. This was due to the user being able to alter their timetable by simply altering the cells of the Excel spreadsheet which would be generated by the code. It also meant that the user could easily print their timetable off using Excels printing functions. Using a GUI would offer a more personalized experience as it could be programmed to display a certain output offering the user a more visually pleasing timetable. GUIs such as “Qt” and “gtkmm” were possibilities for an interface. Excel was however, the best most easy to use option for outputting the data which therefor lead to it being the method chosen.

The fourth part stated the program should assign only time which is free to activities. The user is entering both time defined and time undefined activities. Examples of these are lectures which have a scheduled time, and personal activities such as going to the gym which the user indicates how many times a week that activity occurs. The program should assign personal activities only time which has not been taken up by a time defined activity. This meaning if the person is busy with scheduled events for the entire week, the personal activities will not be assigned any time.

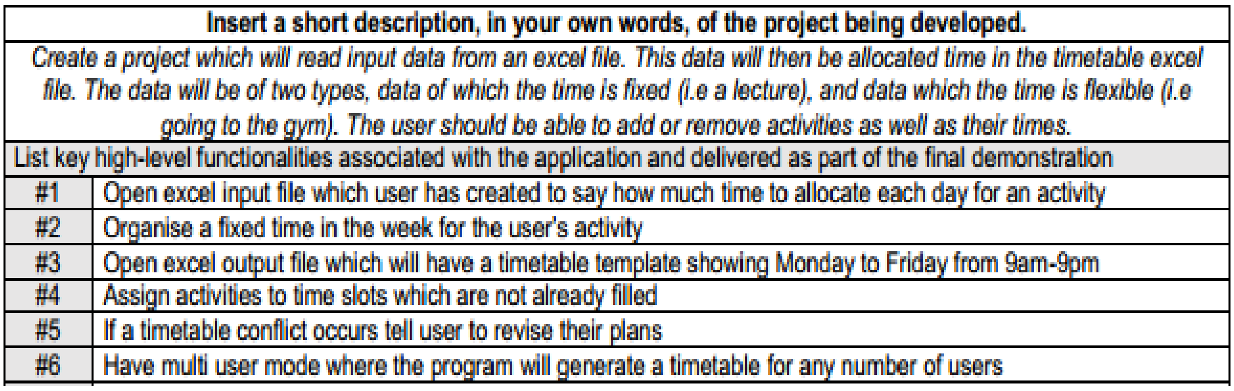
Part five continues from part four as it states that if there were any conflicts occurring in the timetable, the user would be asked to re-evaluate their input. The only possible conflict which could occur would be between two time defined activities. This would be due to the user saying two activities were occurring at the same time.

The final piece of functionality the program would perform would be that it would allow multiple users to create timetables. It would also allow any users doing joint projects it would allocate them the same time per week to work on them.

The User Specification from Figure 1 was adapted as steps three and six were revised during the implementation process. These adaptions were as follows:

- Step three – time changed from 8am-9pm to 9am-9pm as it was realised that on average a person's day starts at 9am and not 8am. This means the program will now cover a twelve hour period rather than a thirteen hour period.
- Step six – the multi user functionality remained however the joint project functionality was removed as this was only researched into and would be quite difficult to implement. It was also not an essential requirement of the project description.

The finalised User Specification can be seen in Figure 2.



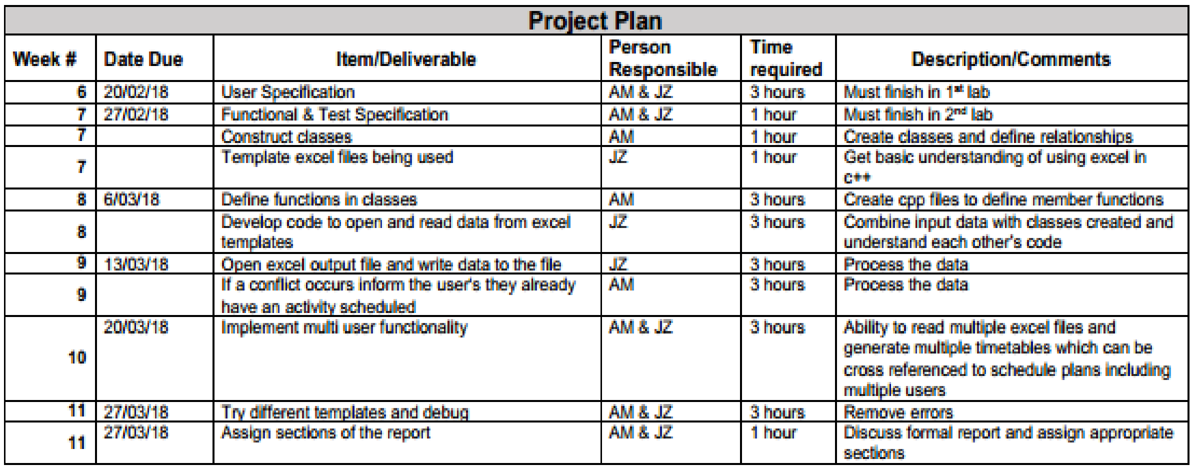
Insert a short description, in your own words, of the project being developed.	
Create a project which will read input data from an excel file. This data will then be allocated time in the timetable excel file. The data will be of two types, data of which the time is fixed (i.e a lecture), and data which the time is flexible (i.e going to the gym). The user should be able to add or remove activities as well as their times.	
List key high-level functionalities associated with the application and delivered as part of the final demonstration	
#1	Open excel input file which user has created to say how much time to allocate each day for an activity
#2	Organise a fixed time in the week for the user's activity
#3	Open excel output file which will have a timetable template showing Monday to Friday from 9am-9pm
#4	Assign activities to time slots which are not already filled
#5	If a timetable conflict occurs tell user to revise their plans
#6	Have multi user mode where the program will generate a timetable for any number of users

Figure 2 - Updated User Specification

With the User Specification complete, a plan could be created which would allow specific tasks to be completed each week in order to implement each piece of functionality.

5.2 Weekly Plan

The plan would allow for targets to be set for each week in order for the program to be progressing at a steady rate whilst also allowing for parallel development of the code. The plan can be seen in Figure 3.



Project Plan

Week #	Date Due	Item/Deliverable	Person Responsible	Time required	Description/Comments
6	20/02/18	User Specification	AM & JZ	3 hours	Must finish in 1st lab
7	27/02/18	Functional & Test Specification	AM & JZ	1 hour	Must finish in 2nd lab
7		Construct classes	AM	1 hour	Create classes and define relationships
7		Template excel files being used	JZ	1 hour	Get basic understanding of using excel in c++
8	6/03/18	Define functions in classes	AM	3 hours	Create cpp files to define member functions
8		Develop code to open and read data from excel templates	JZ	3 hours	Combine input data with classes created and understand each other's code
9	13/03/18	Open excel output file and write data to the file	JZ	3 hours	Process the data
9		If a conflict occurs inform the user's they already have an activity scheduled	AM	3 hours	Process the data
10	20/03/18	Implement multi user functionality	AM & JZ	3 hours	Ability to read multiple excel files and generate multiple timetables which can be cross referenced to schedule plans including multiple users
11	27/03/18	Try different templates and debug	AM & JZ	3 hours	Remove errors
11	27/03/18	Assign sections of the report	AM & JZ	1 hour	Discuss formal report and assign appropriate sections

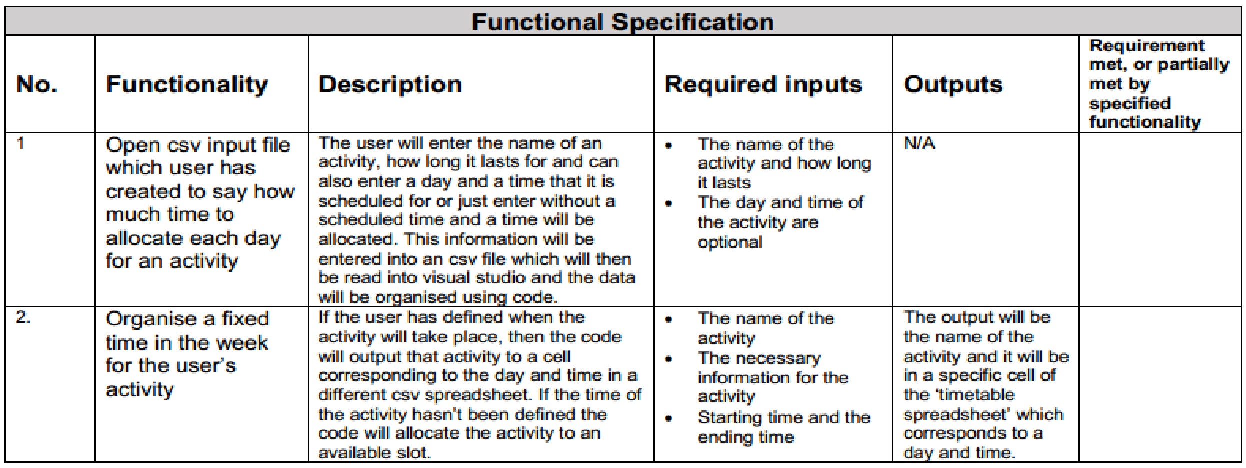
Figure 3 - Weekly Plan Highlighting Targets and Team Member Responsible.

The plan was reasonably adhered to however slight delays from week nine caused the project to be finished at a date later than 27/03/18.

From the User Specification, a Functional Specification could be determined which would describe the functionality while also outlining the different inputs required an expected outputs for each step of the User Specification.

5.3 Functional Specification

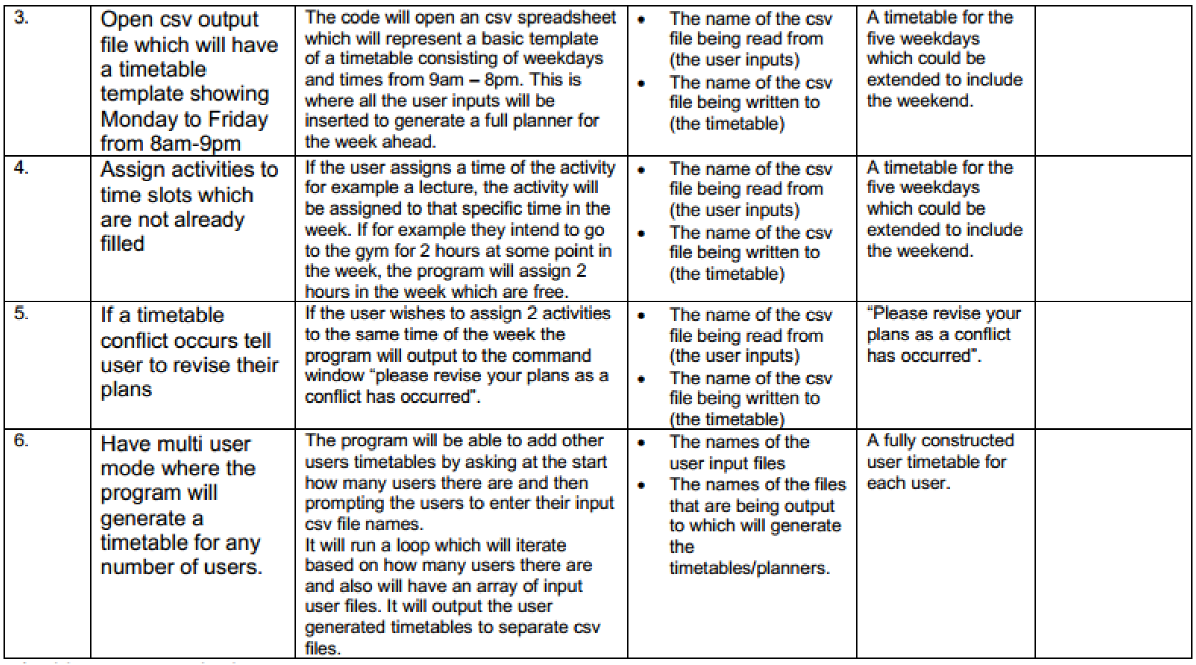
The Functional Specification was generated as seen in Figure 4 and Figure 5. It includes every step from the User Specification with a detailed description of each step along with the required inputs to perform the functionality as well as the outputs generated by the functionality.



Functional Specification

No.	Functionality	Description	Required inputs	Outputs	Requirement met, or partially met by specified functionality
1	Open csv input file which user has created to say how much time to allocate each day for an activity	The user will enter the name of an activity, how long it lasts for and can also enter a day and a time that it is scheduled for or just enter without a scheduled time and a time will be allocated. This information will be entered into an csv file which will then be read into visual studio and the data will be organised using code.	• The name of the activity and how long it lasts • The day and time of the activity are optional	N/A	
2.	Organise a fixed time in the week for the user's activity	If the user has defined when the activity will take place, then the code will output that activity to a cell corresponding to the day and time in a different csv spreadsheet. If the time of the activity hasn't been defined the code will allocate the activity to an available slot.	• The name of the activity • The necessary information for the activity • Starting time and the ending time	The output will be the name of the activity and it will be in a specific cell of the 'timetable spreadsheet' which corresponds to a day and time.	

Figure 4 - Functional Specification Part 1



3.	Open csv output file which will have a timetable template showing Monday to Friday from 8am-9pm	The code will open an csv spreadsheet which will represent a basic template of a timetable consisting of weekdays and times from 9am – 8pm. This is where all the user inputs will be inserted to generate a full planner for the week ahead.	• The name of the csv file being read from (the user inputs) • The name of the csv file being written to (the timetable)	A timetable for the five weekdays which could be extended to include the weekend.	
4.	Assign activities to time slots which are not already filled	If the user assigns a time of the activity for example a lecture, the activity will be assigned to that specific time in the week. If for example they intend to go to the gym for 2 hours at some point in the week, the program will assign 2 hours in the week which are free.	• The name of the csv file being read from (the user inputs) • The name of the csv file being written to (the timetable)	A timetable for the five weekdays which could be extended to include the weekend.	
5.	If a timetable conflict occurs tell user to revise their plans	If the user wishes to assign 2 activities to the same time of the week the program will output to the command window "please revise your plans as a conflict has occurred".	• The name of the csv file being read from (the user inputs) • The name of the csv file being written to (the timetable)	"Please revise your plans as a conflict has occurred".	
6.	Have multi user mode where the program will generate a timetable for any number of users.	The program will be able to add other users timetables by asking at the start how many users there are and then prompting the users to enter their input csv file names. It will run a loop which will iterate based on how many users there are and also will have an array of input user files. It will output the user generated timetables to separate csv files.	• The names of the user input files • The names of the files that are being output to which will generate the timetables/planners.	A fully constructed user timetable for each user.	

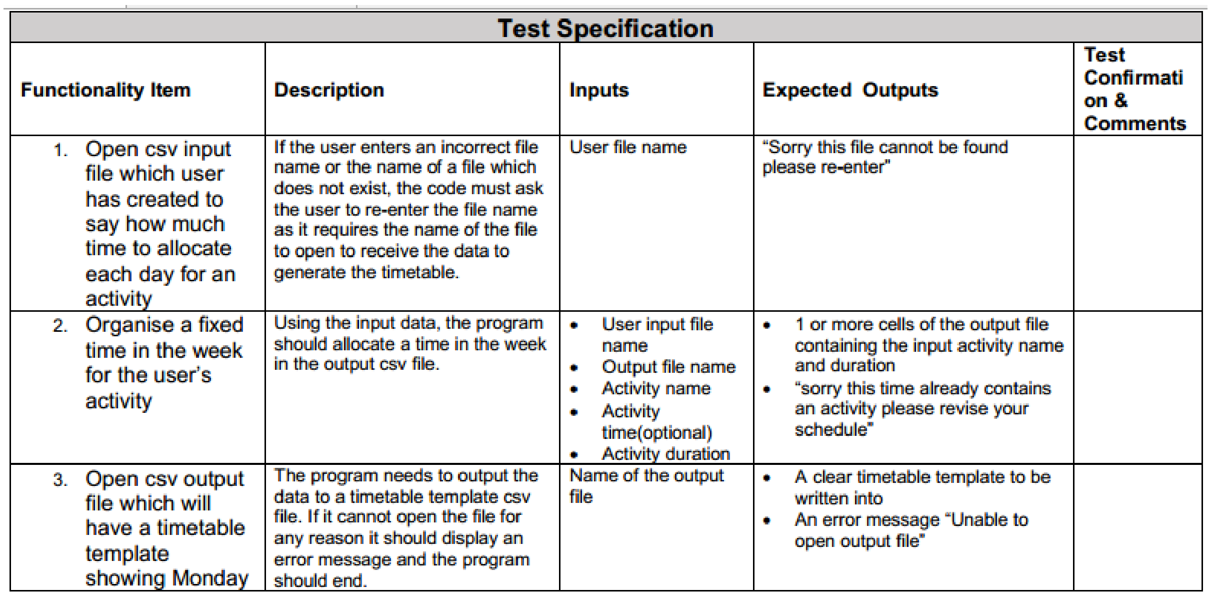
Figure 5 - Functional Specification Part 2

The Functional Specification would aid in the development of each functionality as it highlights any required inputs and the output expected from each piece of functionality. This would help to establish when each piece of functionality was operating correctly.

The final piece of planning carried out was a Test Specification.

5.4 Test Specification

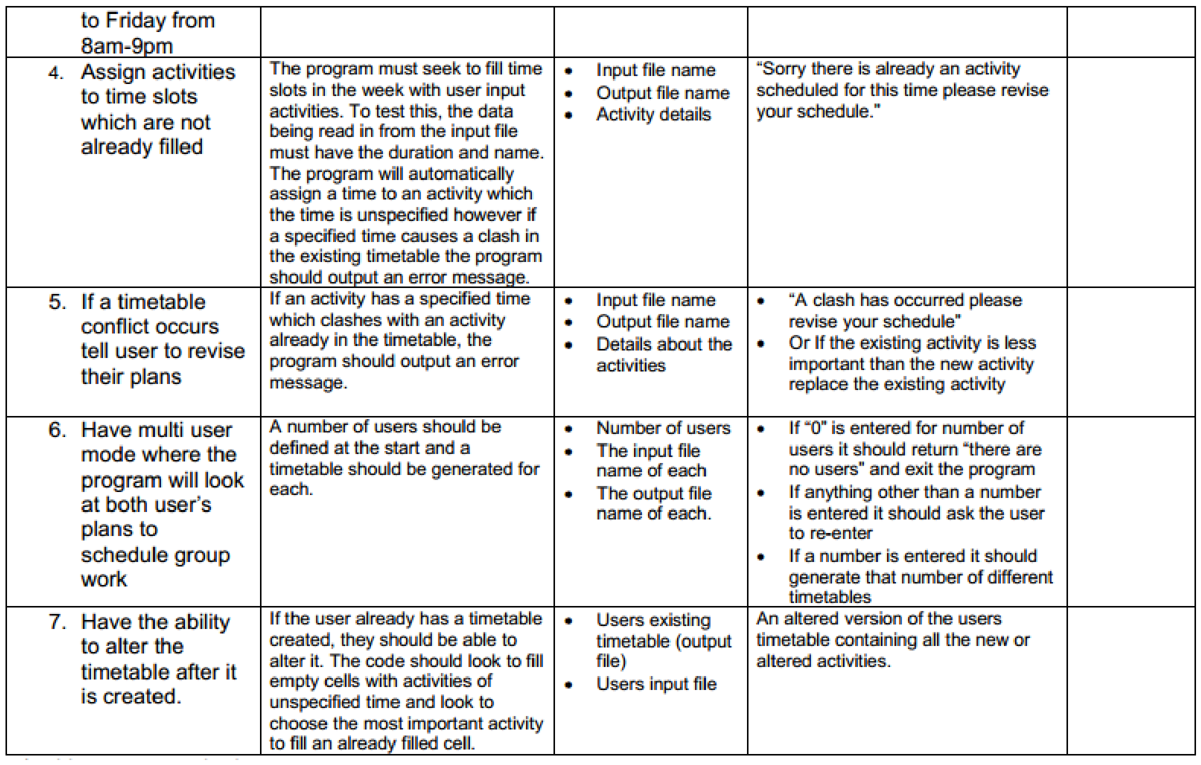
The Test Specification highlights how every functionality would be tested while also explaining what should happen when any problems occur. The Test Specification can be seen in Figure 6 and Figure 7 .



Test Specification

Functionality Item	Description	Inputs	Expected Outputs	Test Confirmation & Comments
1. Open csv input file which user has created to say how much time to allocate each day for an activity	If the user enters an incorrect file name or the name of a file which does not exist, the code must ask the user to re-enter the file name as it requires the name of the file to open to receive the data to generate the timetable.	User file name	"Sorry this file cannot be found please re-enter"	
2. Organise a fixed time in the week for the user's activity	Using the input data, the program should allocate a time in the week in the output csv file.	• User input file name • Output file name • Activity name • Activity time(optional) • Activity duration	• 1 or more cells of the output file containing the input activity name and duration • "sorry this time already contains an activity please revise your schedule"	
3. Open csv output file which will have a timetable template showing Monday	The program needs to output the data to a timetable template csv file. If it cannot open the file for any reason it should display an error message and the program should end.	Name of the output file	• A clear timetable template to be written into • An error message "Unable to open output file"	

Figure 6 - Test Specification Part 1



to Friday from 8am-9pm				
4. Assign activities to time slots which are not already filled	The program must seek to fill time slots in the week with user input activities. To test this, the data being read in from the input file must have the duration and name. The program will automatically assign a time to an activity which the time is unspecified however if a specified time causes a clash in the existing timetable the program should output an error message.	• Input file name • Output file name • Activity details	"Sorry there is already an activity scheduled for this time please revise your schedule."	
5. If a timetable conflict occurs tell user to revise their plans	If an activity has a specified time which clashes with an activity already in the timetable, the program should output an error message.	• Input file name • Output file name • Details about the activities	• "A clash has occurred please revise your schedule" • Or If the existing activity is less important than the new activity replace the existing activity	
6. Have multi user mode where the program will look at both user's plans to schedule group work	A number of users should be defined at the start and a timetable should be generated for each.	• Number of users • The input file name of each • The output file name of each.	• If "0" is entered for number of users it should return "there are no users" and exit the program • If anything other than a number is entered it should ask the user to re-enter • If a number is entered it should generate that number of different timetables	
7. Have the ability to alter the timetable after it is created.	If the user already has a timetable created, they should be able to alter it. The code should look to fill empty cells with activities of unspecified time and look to choose the most important activity to fill an already filled cell.	• Users existing timetable (output file) • Users input file	An altered version of the users timetable containing all the new or altered activities.	

Figure 7 - Test Specification Part 2

The Test Specification indicates how the different functionalities may be tested and what should happen when certain errors occur. This played a vital part in helping the development of the program as it allowed all functionalities to be tested and objectives to be set.

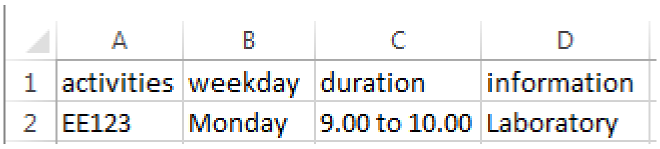
Having created all necessary documentation and planning materials, the programming section could begin in order to implement a solution to the User Specification.

6 Implementation

To implement the code, the User Specification from Figure 1 was used as a guide to implementing each step.

6.1 Taking in the Data

The first step which had to be implemented was the program had to read in the users excel file. To do this a set template excel file had to be created. This would allow an algorithm to be created to read in the data in a specific way. It was decided that the template should consist of activities defined by time at the top and personal activities and hobbies at the bottom. For each time defined activity, the user should enter every time defined activity in the first column of cells under the heading 'activities'. Each activity they enter must be accompanied by three attributes; which weekday it is on, the duration of time it lasts for, and any information to accompany it. An example of this can be seen in Figure 8.



	A	B	C	D
1	activities	weekday	duration	information
2	EE123	Monday	9.00 to 10.00	Laboratory

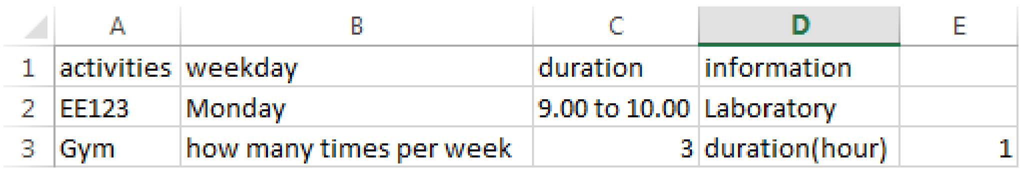
Figure 8 - Example of Input File Time Defined Activities Format

From Figure 8 it can be seen that the three required components for each activity must be entered in a horizontal manner. At this point, certain rules had to be made in order to set a standard for the information being read in. the rules for time defined activities were as follows:

- The activity name could contain any user defined value.
- The weekday on which it occurs can start with a capital or lower case letter however it must not be followed by a space. For example, "Monday", "monday" etc.
- The duration for which it occurs, must be in the format of a 24-hour clock. It must have a start time and an end time in the form "hour.minute" which cannot contain any spaces. These times were separated by the word "to" which contains a space on either side. For example, "9.00 to 15.00".
- The information accompanying the activity can contain any user defined value. For example, "Laboratory – RC446".

These rules must be adhered to when entering any time defined activities as the code was developed around them. If the user wishes to enter multiple instances of an activity such as a lecture and a laboratory for EE123 in this case, they simply copy the format from columns B, C and D and insert them in columns E, F and G. This process can be repeated for as many instances of the activity as the user requires.

Having created a set format for time defined activities, a format had to also be created for time undefined activities such as hobbies. This had to be done in a slightly different format as there would be no times to insert. The format used can be seen in Figure 9.



	A	B	C	D	E
1	activities	weekday	duration	information	
2	EE123	Monday	9.00 to 10.00	Laboratory	
3	Gym	how many times per week	3	duration(hour)	1

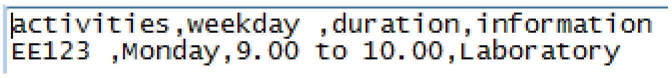
Figure 9 - Example of Input File Time Undefined Activities Format

From Figure 9, it can be seen that the format differs when using time undefined activities. The rules applied to time undefined activities were as follows:

- The activity name could be anything the user wishes.
- The weekday it occurs on was changed to "how many times per week". This was due to the undefined time activities not occurring on a weekday but rather the program would allocate it to weekdays with free time. The program would analyse the cell searching for the word "times" which would allow the program to realise the data about to be read in was for an time undefined activity.
- To enter the number of times per week, the cell directly to the right would be used. The limitations on the value entered were that it had to be a positive integer which could not contain any spaces. In this case "3" was used.
- The next column contained the words "duration(hour)". This would be used to enter how long each instance of the activity would last. The same limitations were required such that it was a positive integer which did not contain any spaces. In this case "1" was used.

If the user wished to add any more time defined activities these would simply be listed in the exact format used to create the first activity just below the EE123 activity in Figure 9. For time undefined activities, more entries could be added beneath "Gym" in Figure 9.

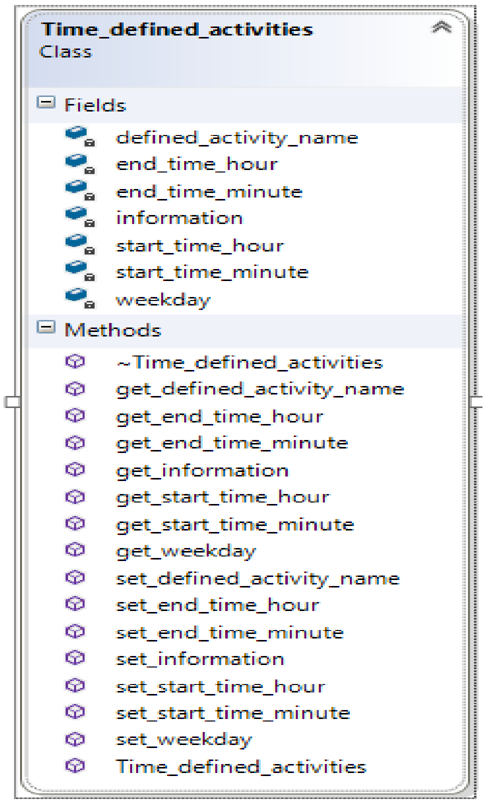
The input file from Figure 9 was saved in the form of a CSV document as this represented the contents of each cell in a .txt document with a comma delimiter between each. This can be seen in Figure 10.



```
activities,weekday ,duration,information
EE123 ,Monday,9.00 to 10.00,Laboratory
```

Figure 10 - Input File in CSV Format

Having identified a format for reading in the data, the algorithm could now be developed. Initially two classes had to be created. A class for time defined activities and a class for time undefined activities. Two classes were required due to the fact that a different format was used for each type of activity therefor they would need different data members. A main file was thus created followed by a header and source file for each class. Class diagrams for each can be seen in Figure 11 and Figure 12.

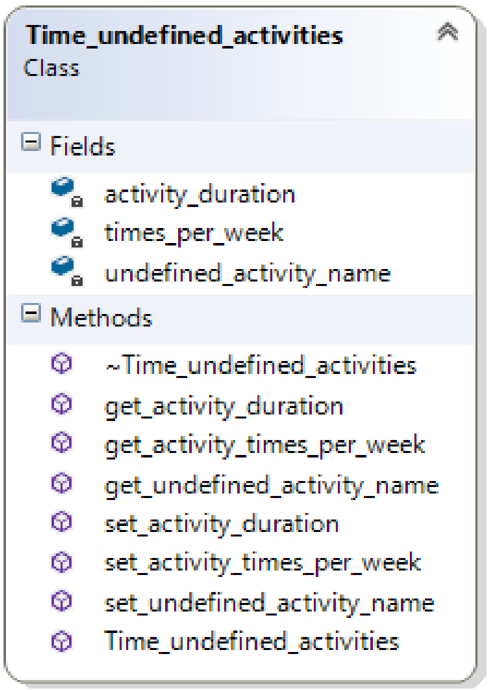


Time_defined_activities
Class

Fields
- defined_activity_name
- end_time_hour
- end_time_minute
- information
- start_time_hour
- start_time_minute
- weekday

Methods
- ~Time_defined_activities
- get_defined_activity_name
- get_end_time_hour
- get_end_time_minute
- get_information
- get_start_time_hour
- get_start_time_minute
- get_weekday
- set_defined_activity_name
- set_end_time_hour
- set_end_time_minute
- set_information
- set_start_time_hour
- set_start_time_minute
- set_weekday
- Time_defined_activities

Figure 11 - Class Diagram for Time Defined Activities



Time_undefined_activities
Class

Fields
- activity_duration
- times_per_week
- undefined_activity_name

Methods
- ~Time_undefined_activities
- get_activity_duration
- get_activity_times_per_week
- get_undefined_activity_name
- set_activity_duration
- set_activity_times_per_week
- set_undefined_activity_name
- Time_undefined_activities

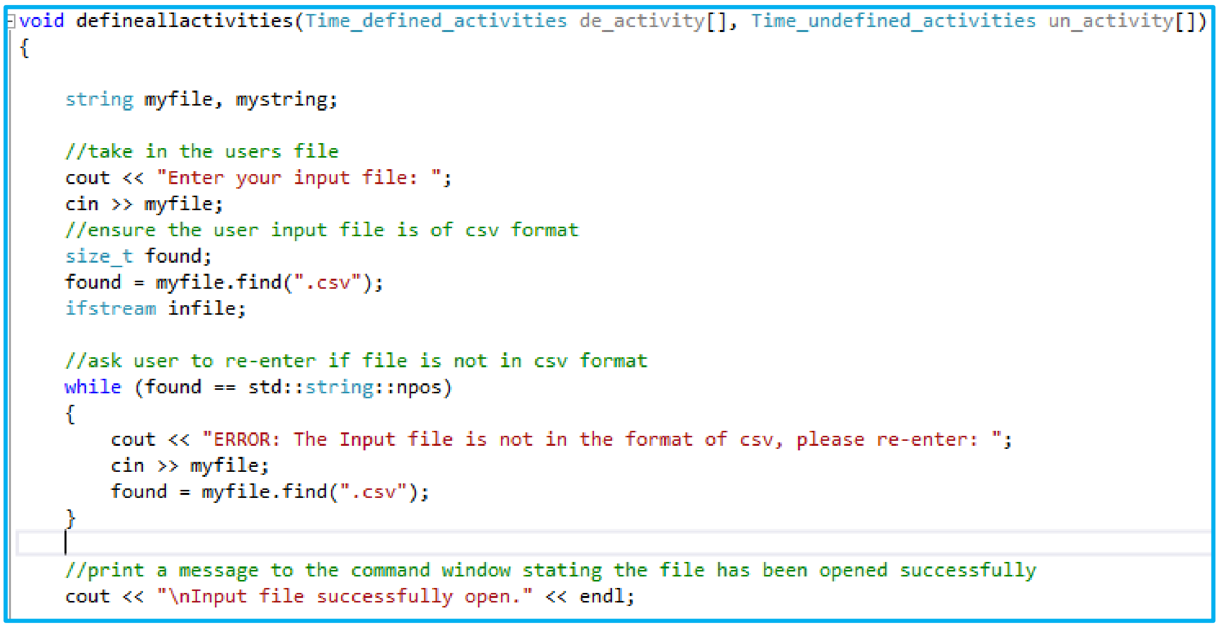
Figure 12 - Class Diagram for Time Undefined Activities

Having developed a class for each type of activity, instances of each class would have to be created to store the data being read in. Arrays were used to create instances. This was due to the limit of 18 potential time defined activities and 10 potential time undefined activities. It was established that the user would be very unlikely to have more than either of these limits. Each array was given the type of its associated class as seen in Figure 13.

```
//create 18 time defined activities which will fill up slots in a week
Time_defined_activities de_activity[18];

//create another 10 time undefined activities which will fill up free time in a week
Time_undefined_activities un_activity[10];
```

Figure 13 - Arrays of Each Activity Type

A new source file named “inputdata” was created as this would be used to read in the input file and assign the data read in to the arrays created for each class. The function inside the file was called “defineallactivities" and would be of the void type as nothing would be returned. The parameters for this function would simply the names of both the arrays. The initial step for the function would be to take in and open the user’s file which can be seen in Figure 14.

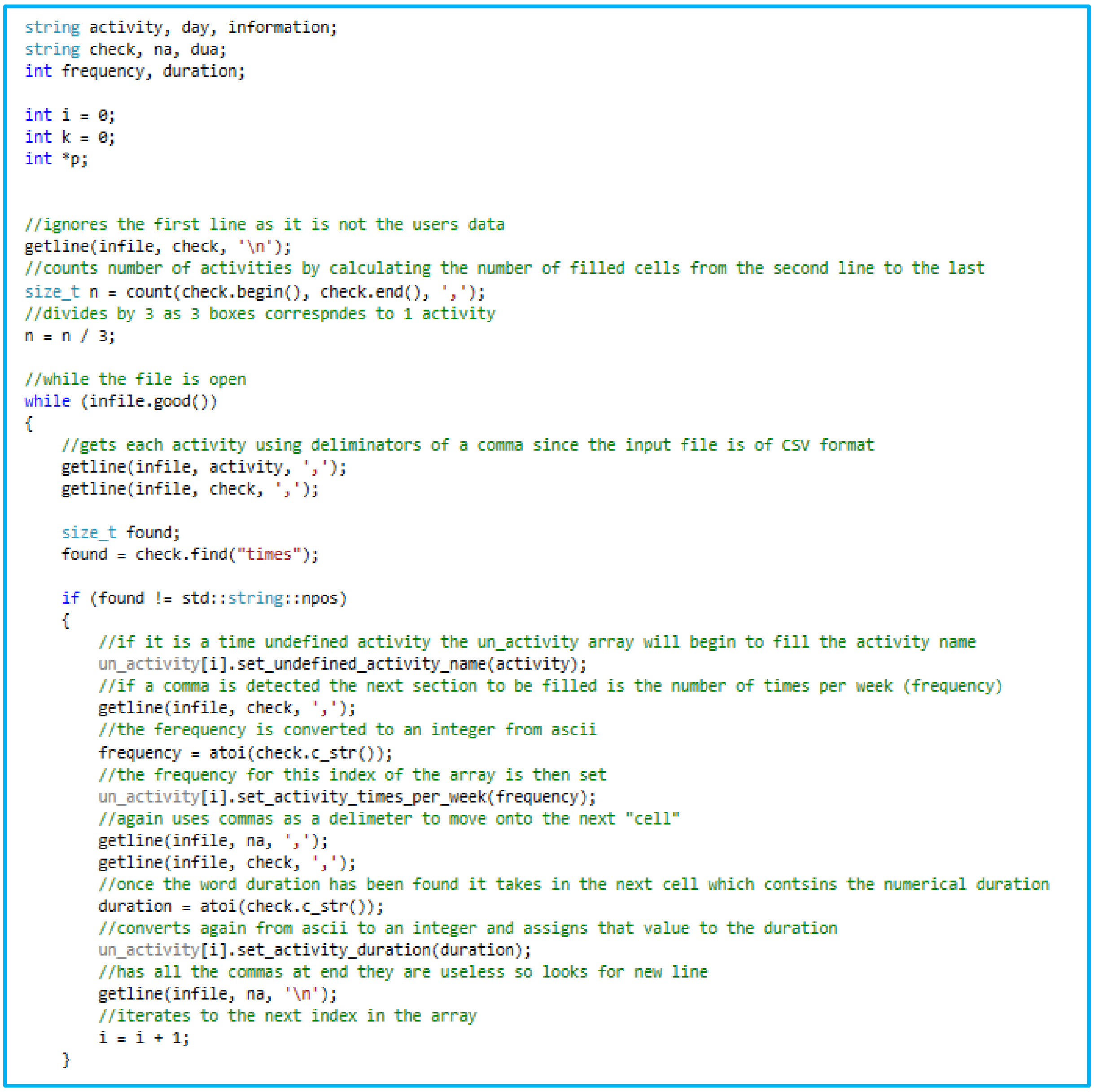


```
void defineallactivities(Time_defined_activities de_activity[], Time_undefined_activities un_activity[])
{

    string myfile, mystring;

    //take in the users file
    cout << "Enter your input file: ";
    cin >> myfile;
    //ensure the user input file is of csv format
    size_t found;
    found = myfile.find(".csv");
    ifstream infile;

    //ask user to re-enter if file is not in csv format
    while (found == std::string::npos)
    {
        cout << "ERROR: The Input file is not in the format of csv, please re-enter: ";
        cin >> myfile;
        found = myfile.find(".csv");
    }

    //print a message to the command window stating the file has been opened successfully
    cout << "\nInput file successfully open." << endl;
```

Figure 14 - Opening the User Input File

The operation uses the fstream library (filestream) to open files and output files using keywords “ifstream” and “ofstream”. It can be see that if the users file is not in the format of a CSV file, the program will prompt the user to re-enter their input file name. The simplest way for the user to insert their file name was to simply drag and drop the file into the command window. This would input the address of the file meaning the “.csv” requirement would always be present. Once the user’s input file had been opened, the command window would present a message to inform the user that this step had been a success.

Following this step the next step would be to being reading in the data from the file. The method used to read in the activities which are not time defined can be seen in Figure 15.

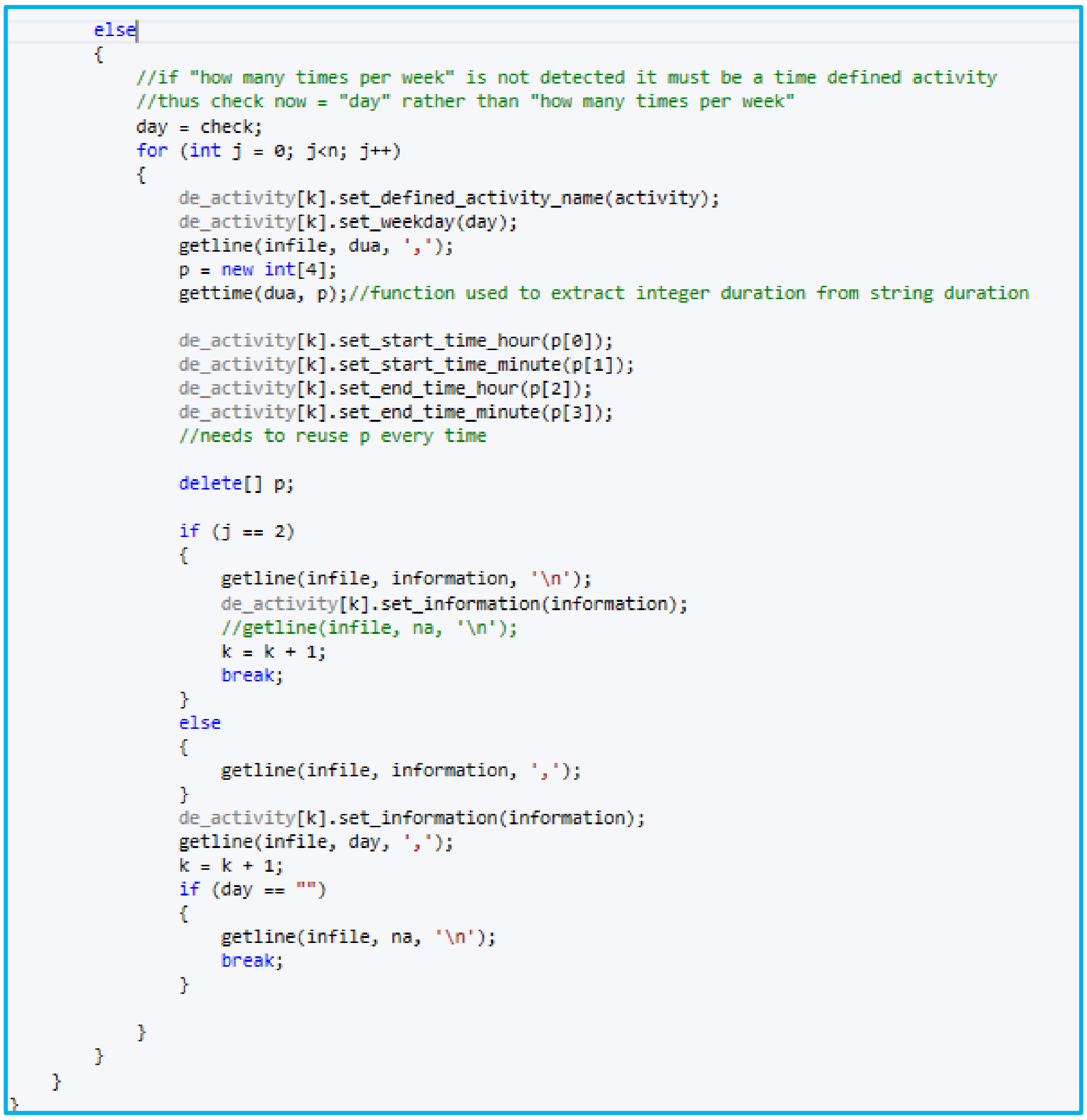


```
string activity, day, information;
string check, na, dua;
int frequency, duration;

int i = 0;
int k = 0;
int *p;


//ignores the first line as it is not the users data
getline(infile, check, '\n');
//counts number of activities by calculating the number of filled cells from the second line to the last
size_t n = count(check.begin(), check.end(), ',');
//divides by 3 as 3 boxes correspndes to 1 activity
n = n / 3;

//while the file is open
while (infile.good())
{
    //gets each activity using deliminators of a comma since the input file is of CSV format
    getline(infile, activity, ',');
    getline(infile, check, ',');

    size_t found;
    found = check.find("times");

    if (found != std::string::npos)
    {
        //if it is a time undefined activity the un_activity array will begin to fill the activity name
        un_activity[i].set_undefined_activity_name(activity);
        //if a comma is detected the next section to be filled is the number of times per week (frequency)
        getline(infile, check, ',');
        //the ferequency is converted to an integer from ascii
        frequency = atoi(check.c_str());
        //the frequency for this index of the array is then set
        un_activity[i].set_activity_times_per_week(frequency);
        //again uses commas as a delimeter to move onto the next "cell"
        getline(infile, na, ',');
        getline(infile, check, ',');
        //once the word duration has been found it takes in the next cell which contsins the numerical duration
        duration = atoi(check.c_str());
        //converts again from ascii to an integer and assigns that value to the duration
        un_activity[i].set_activity_duration(duration);
        //has all the commas at end they are useless so looks for new line
        getline(infile, na, '\n');
        //iterates to the next index in the array
        i = i + 1;
    }
```

Figure 15 - Reading in the Time Undefined Activities

Referencing back to Figure 10, an initial "getline" function is used to ignore the first row of cells as they are simply headers for the user to understand which cell to enter their data into.

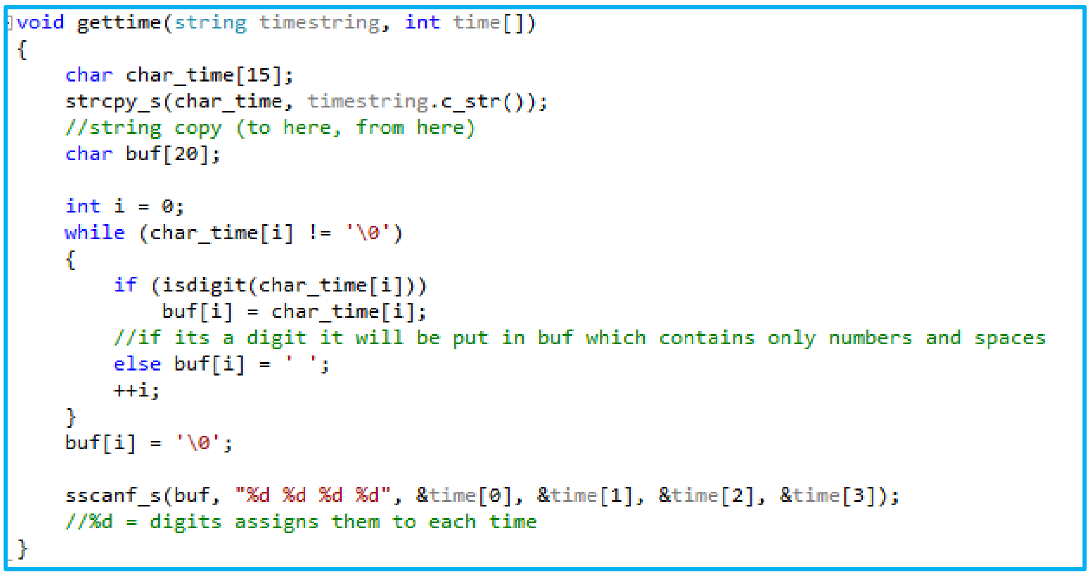
A while loop is used to allow the process to repeat until the end of the file has been reached, i.e. there are no more pieces of data to be read in. The "if" function is then used which detects whether the cell being analysed contains a weekday or the phrase "how many times per week". If the word "times" is detected, it must be a time undefined activity. The program then cycles through adjacent cells gathering the user's input data through the use of "getline" functions. When an undefined function is found, the program will assign each data member of the class and once all data members have been assigned a value the program will iterate to the next row of cells establishing whether it is a time defined or undefined activity. Through this process the array of time undefined activities could be filled. To fill the array of time defined activities an else statement was used as seen in Figure 16.



```
        else
        {
            //if "how many times per week" is not detected it must be a time defined activity
            //thus check now = "day" rather than "how many times per week"
            day = check;
            for (int j = 0; j<n; j++)
            {
                de_activity[k].set_defined_activity_name(activity);
                de_activity[k].set_weekday(day);
                getline(infile, dua, ',');
                p = new int[4];
                gettime(dua, p);//function used to extract integer duration from string duration

                de_activity[k].set_start_time_hour(p[0]);
                de_activity[k].set_start_time_minute(p[1]);
                de_activity[k].set_end_time_hour(p[2]);
                de_activity[k].set_end_time_minute(p[3]);
                //needs to reuse p every time

                delete[] p;

                if (j == 2)
                {
                    getline(infile, information, '\n');
                    de_activity[k].set_information(information);
                    //getline(infile, na, '\n');
                    k = k + 1;
                    break;
                }
                else
                {
                    getline(infile, information, ',');
                }
                de_activity[k].set_information(information);
                getline(infile, day, ',');
                k = k + 1;
                if (day == "")
                {
                    getline(infile, na, '\n');
                    break;
                }

            }
        }
    }
}
```

Figure 16 - Else Statement Used to Assign Time Defined Activities

Due to the data members being different for the time defined activity class a different method was required to take in the data. Due to the format of the input files being pre-set to a certain format, the data could simply be read in as the activity name, the day of the week it occurs and also the duration it occurs for as these three cells were adjacent to each other. The duration was however a string so the “gettime” function was created to extract the hour that the activity started and finished. The “gettime” function can be seen in Figure 17.



```
void gettime(string timestring, int time[])
{
    char char_time[15];
    strcpy_s(char_time, timestring.c_str());
    //string copy (to here, from here)
    char buf[20];

    int i = 0;
    while (char_time[i] != '\0')
    {
        if (isdigit(char_time[i]))
            buf[i] = char_time[i];
        //if its a digit it will be put in buf which contains only numbers and spaces
        else buf[i] = ' ';
        ++i;
    }
    buf[i] = '\0';

    sscanf_s(buf, "%d %d %d %d", &time[0], &time[1], &time[2], &time[3]);
    //%d = digits assigns them to each time
}
```

Figure 17 - Function Used to Calculate the Duration of a Time Defined Activity [1]

The input file had now been read in and all data had been assigned to either an index of the time defined or time undefined array. The first step of the user specification was now complete.

6.2 Organising a Time in the Week for Each Activity

The second step in the User Specification from Figure 2, was that the program would have to assign a time in the week to assign the users activities. A new class was created which was called "Day_of_week" which contained attributes such as activity name and the information about it. Both of these data members would be taken from the data which had been taken in from the users input file. The "Day_of_week" class diagram can be seen in Figure 18.

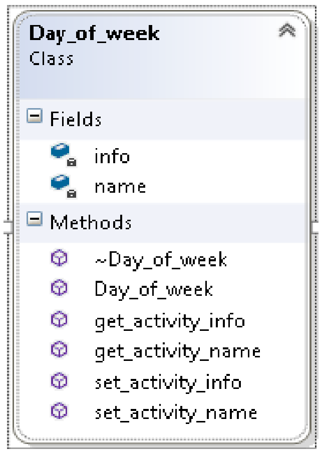
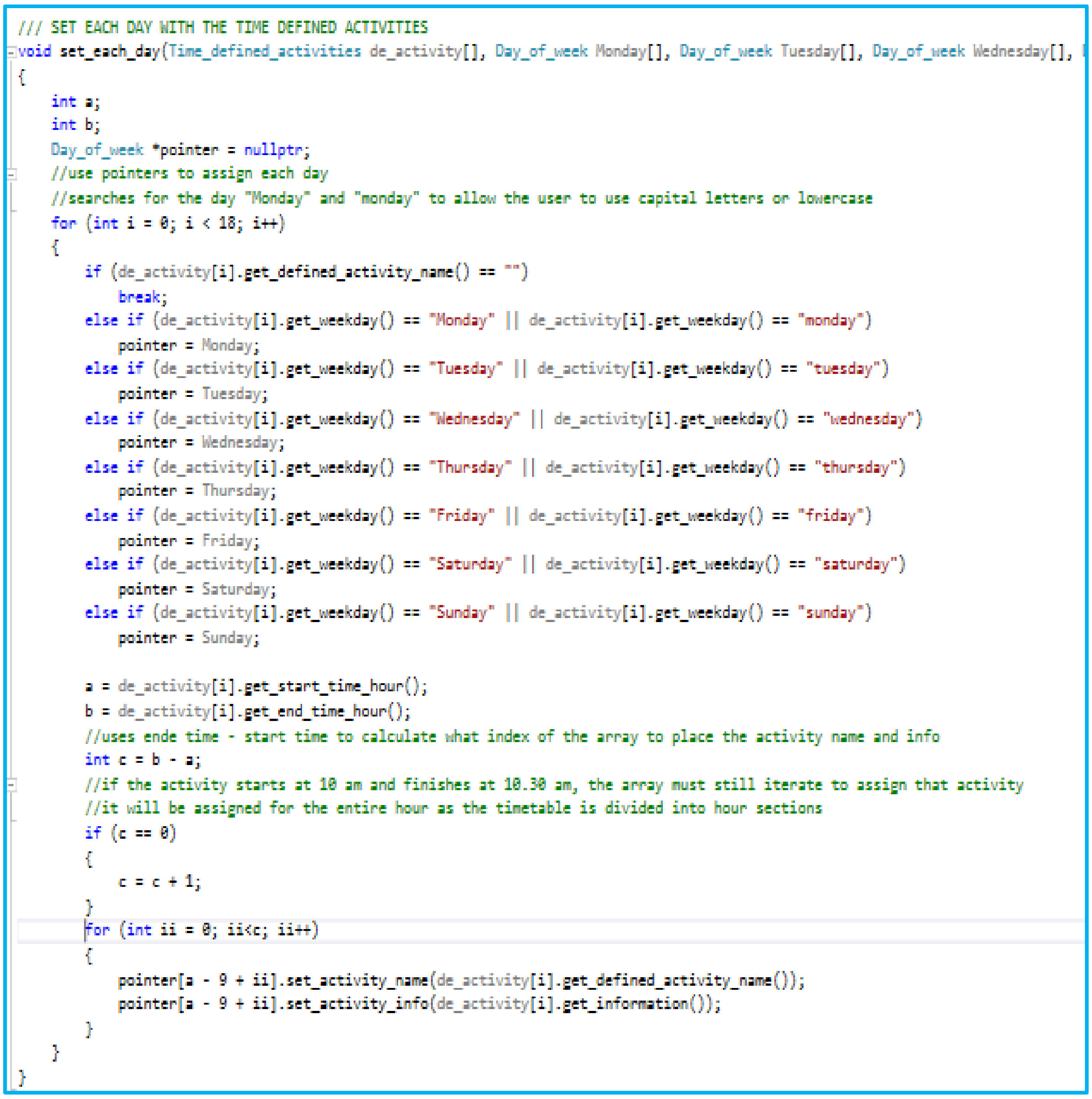


Figure 18 - Class Diagram for Class Defining Day of the Week

Seven instances of the class were created in the form of arrays. Each array was assigned to the size of twelve as the timetable would cover 12 hours, (9am – 9pm), as seen in the updated User Specification in Figure 2. Arrays were used as the size would always be twelve and to print out even the empty indexes to represent a free hour. This could be more easily carried out with arrays rather than vectors. Each array was named after a day of the week.

New source and header files were created called “slot_in_functions” which would contain functions which were used to assign the input data to days of the week. A function was defined in this file called “set_each_day”. The purpose of this function was to set the time defined activities to the day and time on which they occur. The function can be seen in Figure 19.

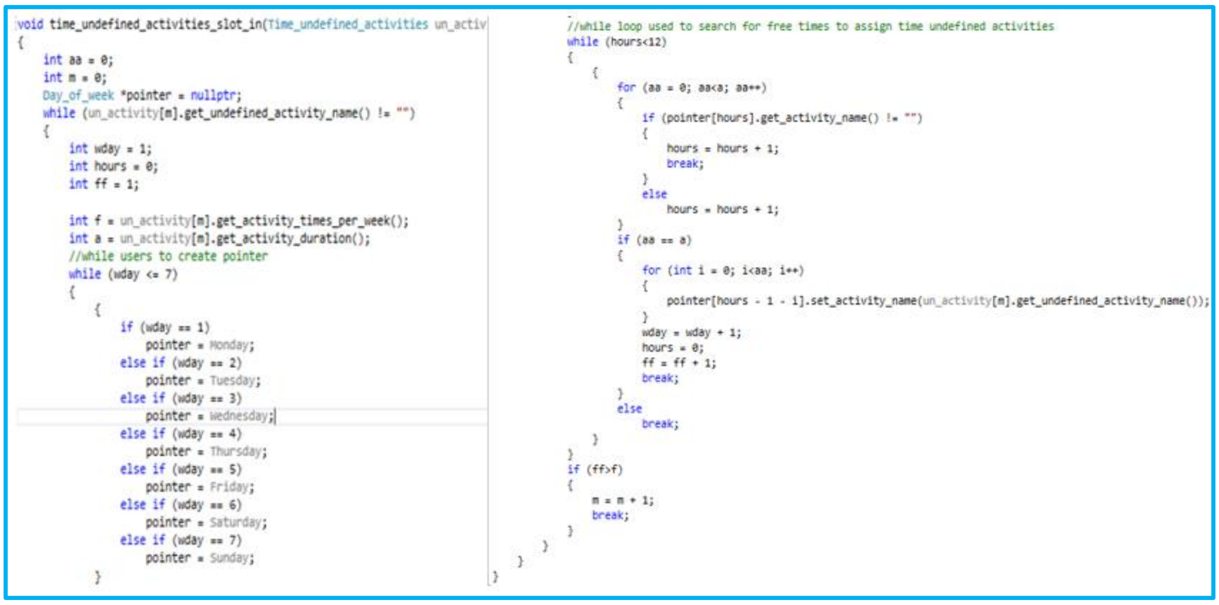


```
/// SET EACH DAY WITH THE TIME DEFINED ACTIVITIES
void set_each_day(Time_defined_activities de_activity[], Day_of_week Monday[], Day_of_week Tuesday[], Day_of_week Wednesday[],
{
    int a;
    int b;
    Day_of_week *pointer = nullptr;
    //use pointers to assign each day
    //searches for the day "Monday" and "monday" to allow the user to use capital letters or lowercase
    for (int i = 0; i < 18; i++)
    {
        if (de_activity[i].get_defined_activity_name() == "")
            break;
        else if (de_activity[i].get_weekday() == "Monday" || de_activity[i].get_weekday() == "monday")
            pointer = Monday;
        else if (de_activity[i].get_weekday() == "Tuesday" || de_activity[i].get_weekday() == "tuesday")
            pointer = Tuesday;
        else if (de_activity[i].get_weekday() == "Wednesday" || de_activity[i].get_weekday() == "wednesday")
            pointer = Wednesday;
        else if (de_activity[i].get_weekday() == "Thursday" || de_activity[i].get_weekday() == "thursday")
            pointer = Thursday;
        else if (de_activity[i].get_weekday() == "Friday" || de_activity[i].get_weekday() == "friday")
            pointer = Friday;
        else if (de_activity[i].get_weekday() == "Saturday" || de_activity[i].get_weekday() == "saturday")
            pointer = Saturday;
        else if (de_activity[i].get_weekday() == "Sunday" || de_activity[i].get_weekday() == "sunday")
            pointer = Sunday;

        a = de_activity[i].get_start_time_hour();
        b = de_activity[i].get_end_time_hour();
        //uses ende time - start time to calculate what index of the array to place the activity name and info
        int c = b - a;
        //if the activity starts at 10 am and finishes at 10.30 am, the array must still iterate to assign that activity
        //it will be assigned for the entire hour as the timetable is divided into hour sections
        if (c == 0)
        {
            c = c + 1;
        }
        for (int ii = 0; ii<c; ii++)
        {
            pointer[a - 9 + ii].set_activity_name(de_activity[i].get_defined_activity_name());
            pointer[a - 9 + ii].set_activity_info(de_activity[i].get_information());
        }
    }
}
```

Figure 19 - Function Used to Assign Activities for Each Day at Specific Times

Pointers were used to allow a loop to be created which could condense the code. The code simply checked which day the activity occurred on and assigned that value to the pointer. The code then calculates the start time and duration in order to know which index of the array to insert the time. Activities which lasted less than one hour required one if condition in order to allow the loop to iterate through as end time minus the start time would result in zero meaning the loop would not iterate.

Another function was created which would assign time undefined activities into empty indexes of the arrays, (free times in the week). This function can be seen in Figure 20.



```cpp
void time_undefined_activities_slot_in(Time_undefined_activities un_activ
{
    int aa = 0;
    int m = 0;
    Day_of_week *pointer = nullptr;
    while (un_activity[m].get_undefined_activity_name() != "")
    {
        int wday = 1;
        int hours = 0;
        int ff = 1;

        int f = un_activity[m].get_activity_times_per_week();
        int a = un_activity[m].get_activity_duration();
        //while users to create pointer
        while (wday <= 7)
        {
            {
                if (wday == 1)
                    pointer = Monday;
                else if (wday == 2)
                    pointer = Tuesday;
                else if (wday == 3)
                    pointer = Wednesday;
                else if (wday == 4)
                    pointer = Thursday;
                else if (wday == 5)
                    pointer = Friday;
                else if (wday == 6)
                    pointer = Saturday;
                else if (wday == 7)
                    pointer = Sunday;
            }
```

```cpp
            //while loop used to search for free times to assign time undefined activities
            while (hours<12)
            {
                {
                    for (aa = 0; aa<a; aa++)
                    {
                        if (pointer[hours].get_activity_name() != "")
                        {
                            hours = hours + 1;
                            break;
                        }
                        else
                            hours = hours + 1;
                    }
                    if (aa == a)
                    {
                        for (int i = 0; i<aa; i++)
                        {
                            pointer[hours - 1 - i].set_activity_name(un_activity[m].get_undefined_activity_name());
                        }
                        wday = wday + 1;
                        hours = 0;
                        ff = ff + 1;
                        break;
                    }
                    else
                        break;
                }
            }
            if (ff>f)
            {
                m = m + 1;
                break;
            }
        }
    }
}
```

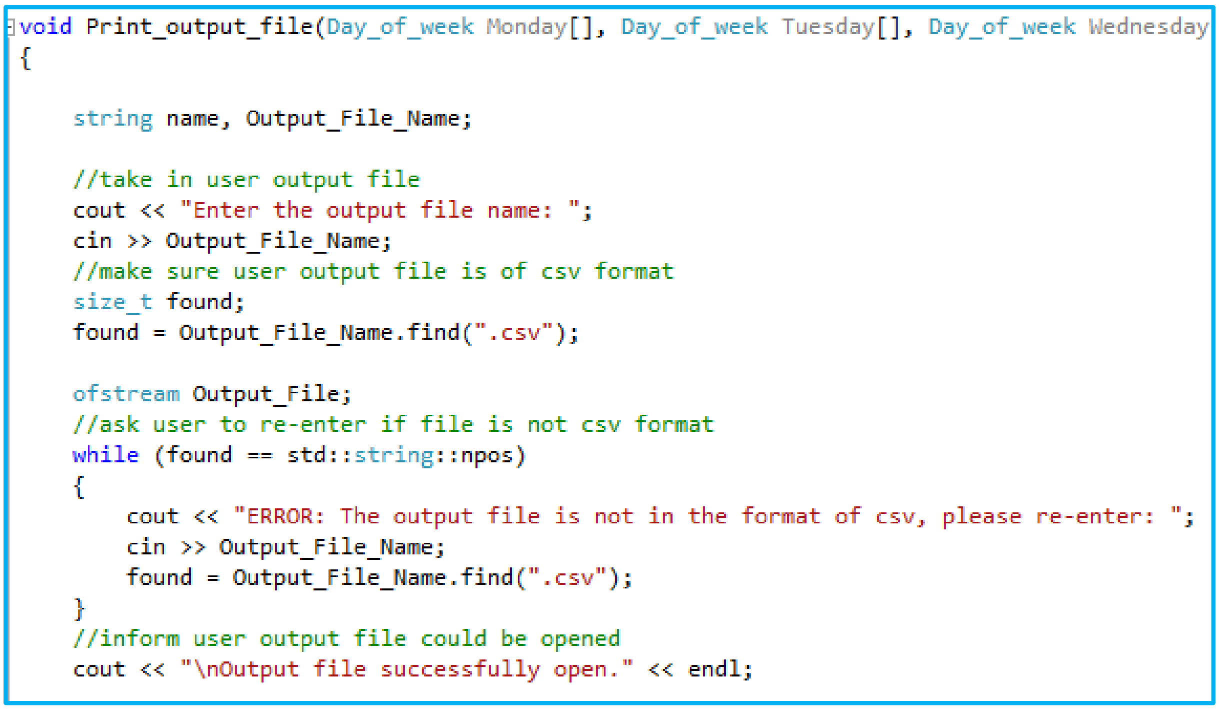
Figure 20 - Function Used to Assign Time Undefined Activities to Free Times throughout the Week

The pointer approach was again used to condense the code. The second while loop was used to search for free times throughout the week (empty indexes of each array). This would be an effective method of assigning the time undefined activities. If all slots were full, the activities would remain unassigned.

This now resulted in all the data being ready to output to a file which meant step two was complete and step three could now begin.

6.3 Create a Template Output Timetable

To create a timetable, a new source file called "otherfunctions" which would contain functions which were not relevant to the taking in data or the data assignment functionalities. The function "Print_output_file" was created in which a template containing each day in the first column and each hour from 9am to 9pm along the top row, would be created using code. Initially the program would take in the address of the user output file which is the file the timetable would be printed to. It would ensure the user file was of the CSV format as seen in Figure 21.



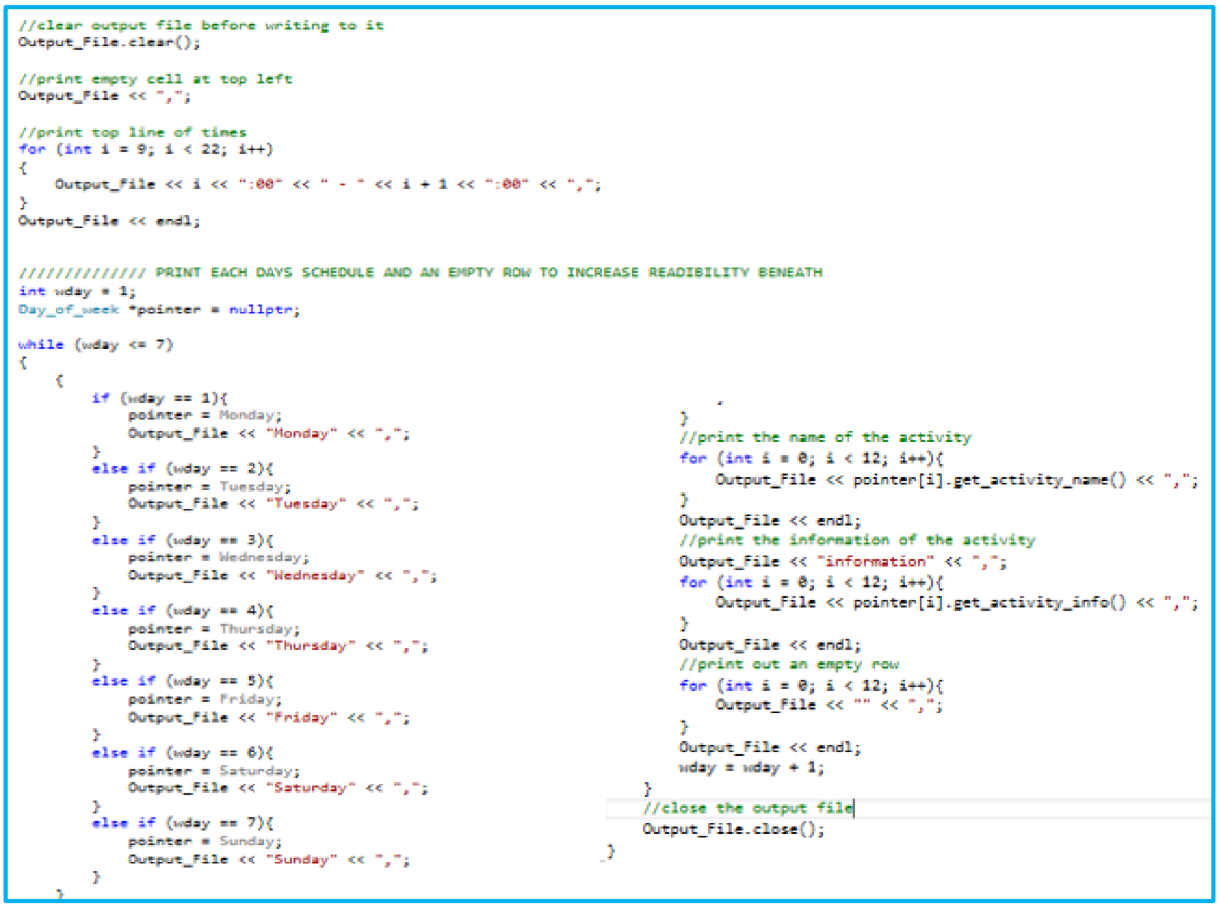
```
void Print_output_file(Day_of_week Monday[], Day_of_week Tuesday[], Day_of_week Wednesday
{

    string name, Output_File_Name;

    //take in user output file
    cout << "Enter the output file name: ";
    cin >> Output_File_Name;
    //make sure user output file is of csv format
    size_t found;
    found = Output_File_Name.find(".csv");

    ofstream Output_File;
    //ask user to re-enter if file is not csv format
    while (found == std::string::npos)
    {
        cout << "ERROR: The output file is not in the format of csv, please re-enter: ";
        cin >> Output_File_Name;
        found = Output_File_Name.find(".csv");
    }
    //inform user output file could be opened
    cout << "\nOutput file successfully open." << endl;
```

Figure 21 - Opening User Output File

The code ensures that the user enters a CSV file as it searches for “.csv” in the user input. This is the same method used to check the input file is of the CSV format. When a CSV file has been entered the user would be informed the file had been opened successfully. When the CSV file is opened it is cleared to ensure there would be no data in the file which could interfere with the final timetable.

The code then generates a timetable template which can be seen in Figure 22. The code prints an empty cell at the top left of the sheet. It then prints a row of times from 9am to 9pm. It then returns to the next row and prints “Monday” followed by two empty rows. The row “Monday” is on will contain the names of the activities whilst the row directly beneath will contain the information about the activity. The second empty row improves the readability of the timetable by spacing out each day.

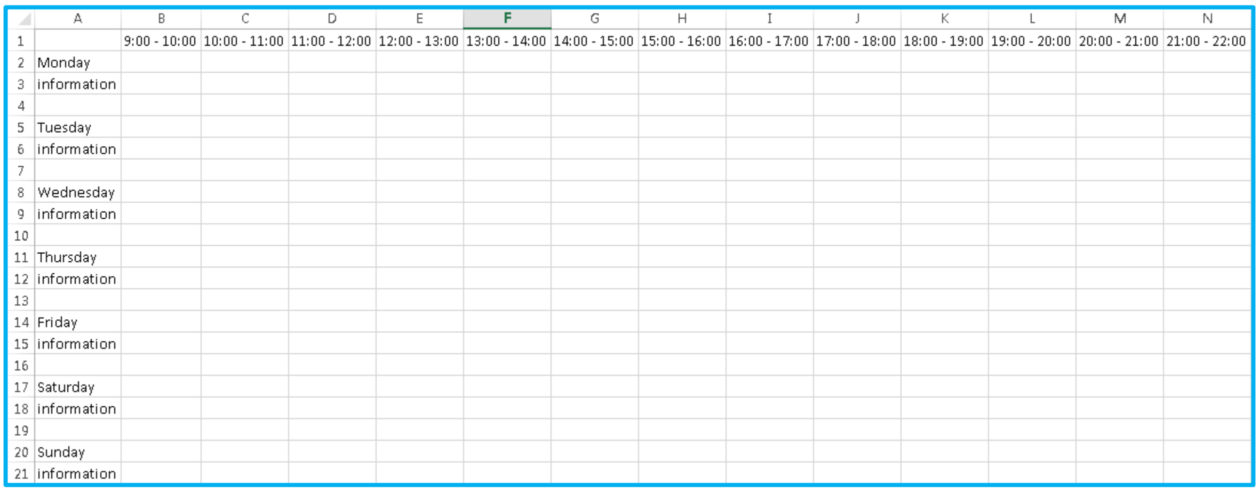
This code would create the file seen in Figure 23. The pointer used would allow step four of the User Specification to be completed as it would output the weekday arrays created earlier.



```
//clear output file before writing to it
Output_File.clear();

//print empty cell at top left
Output_File << ",";

//print top line of times
for (int i = 9; i < 22; i++)
{
    Output_File << i << ":00" << " - " << i + 1 << ":00" << ",";
}
Output_File << endl;


////////////// PRINT EACH DAYS SCHEDULE AND AN EMPTY ROW TO INCREASE READIBILITY BENEATH
int wday = 1;
Day_of_week *pointer = nullptr;

while (wday <= 7)
{
    {
        if (wday == 1){
            pointer = Monday;
            Output_File << "Monday" << ",";
        }
        else if (wday == 2){
            pointer = Tuesday;
            Output_File << "Tuesday" << ",";
        }
        else if (wday == 3){
            pointer = Wednesday;
            Output_File << "Wednesday" << ",";
        }
        else if (wday == 4){
            pointer = Thursday;
            Output_File << "Thursday" << ",";
        }
        else if (wday == 5){
            pointer = Friday;
            Output_File << "Friday" << ",";
        }
        else if (wday == 6){
            pointer = Saturday;
            Output_File << "Saturday" << ",";
        }
        else if (wday == 7){
            pointer = Sunday;
            Output_File << "Sunday" << ",";
        }
    }
```

```
        }
        //print the name of the activity
        for (int i = 0; i < 12; i++){
            Output_File << pointer[i].get_activity_name() << ",";
        }
        Output_File << endl;
        //print the information of the activity
        Output_File << "information" << ",";
        for (int i = 0; i < 12; i++){
            Output_File << pointer[i].get_activity_info() << ",";
        }
        Output_File << endl;
        //print out an empty row
        for (int i = 0; i < 12; i++){
            Output_File << "" << ",";
        }
        Output_File << endl;
        wday = wday + 1;
    }
    //close the output file
    Output_File.close();
}
```

Figure 22 - Code Which Creates the Timetable

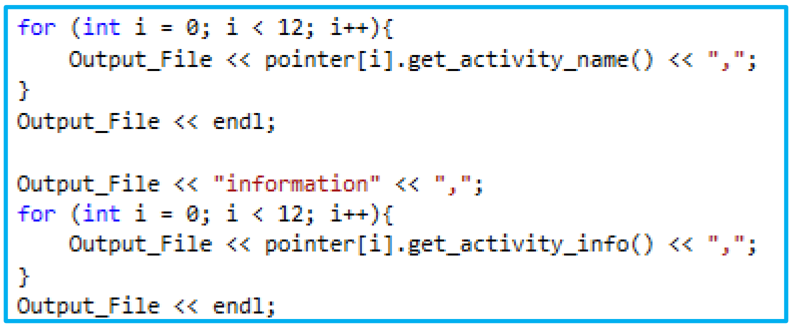


	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday													
3	information													
4														
5	Tuesday													
6	information													
7														
8	Wednesday													
9	information													
10														
11	Thursday													
12	information													
13														
14	Friday													
15	information													
16														
17	Saturday													
18	information													
19														
20	Sunday													
21	information													

Figure 23 - Timetable Template

6.4 Output Each Weekday Array Containing Activity Names and Information

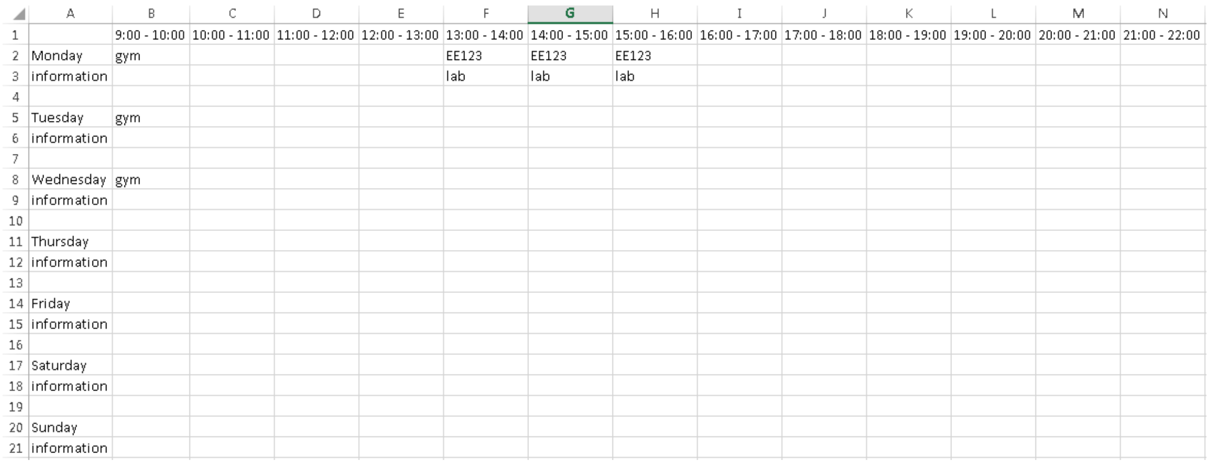
The step was carried out simultaneously as step three due to the similar nature of the operation. Loops in Figure 22, were implemented to print out each day. This was done through the use of pointers. The loops can be seen in Figure 24.



```cpp
for (int i = 0; i < 12; i++){
    Output_File << pointer[i].get_activity_name() << ",";
}
Output_File << endl;

Output_File << "information" << ",";
for (int i = 0; i < 12; i++){
    Output_File << pointer[i].get_activity_info() << ",";
}
Output_File << endl;
```

Figure 24 - Loops Used to Output Each Weekday

This would print each weekday iterating through the hours of the day, (indexes of the array). If an index of the array was empty it would print an empty cell which would represent free time. A simple example can be found when using Figure 9 as the input file generates the output file seen in Figure 25.



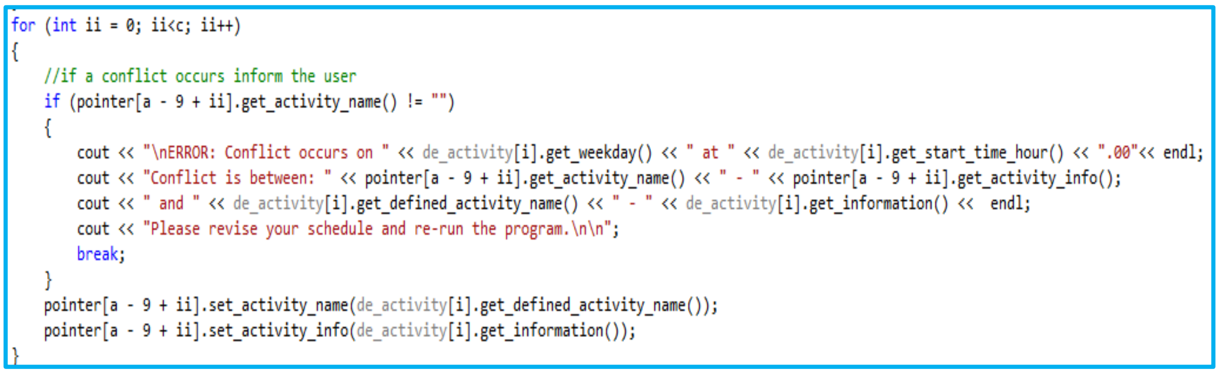
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday	gym				EE123	EE123	EE123						
3	information					lab	lab	lab						
4														
5	Tuesday	gym												
6	information													
7														
8	Wednesday	gym												
9	information													
10														
11	Thursday													
12	information													
13														
14	Friday													
15	information													
16														
17	Saturday													
18	information													
19														
20	Sunday													
21	information													

Figure 25 - Output from Figure 9 input

This meant step four was now complete.

6.5 If a Conflict Occurs the User Must be Notified

To identify if a conflict had occurred between activities in the user input file, a small if function was inserted into the function which takes in the data. This if function can be seen in Figure 26.



```cpp
for (int ii = 0; ii<c; ii++)
{
    //if a conflict occurs inform the user
    if (pointer[a - 9 + ii].get_activity_name() != "")
    {
        cout << "\nERROR: Conflict occurs on " << de_activity[i].get_weekday() << " at " << de_activity[i].get_start_time_hour() << ".00"<< endl;
        cout << "Conflict is between: " << pointer[a - 9 + ii].get_activity_name() << " - " << pointer[a - 9 + ii].get_activity_info();
        cout << " and " << de_activity[i].get_defined_activity_name() << " - " << de_activity[i].get_information() <<  endl;
        cout << "Please revise your schedule and re-run the program.\n\n";
        break;
    }
    pointer[a - 9 + ii].set_activity_name(de_activity[i].get_defined_activity_name());
    pointer[a - 9 + ii].set_activity_info(de_activity[i].get_information());
}
```

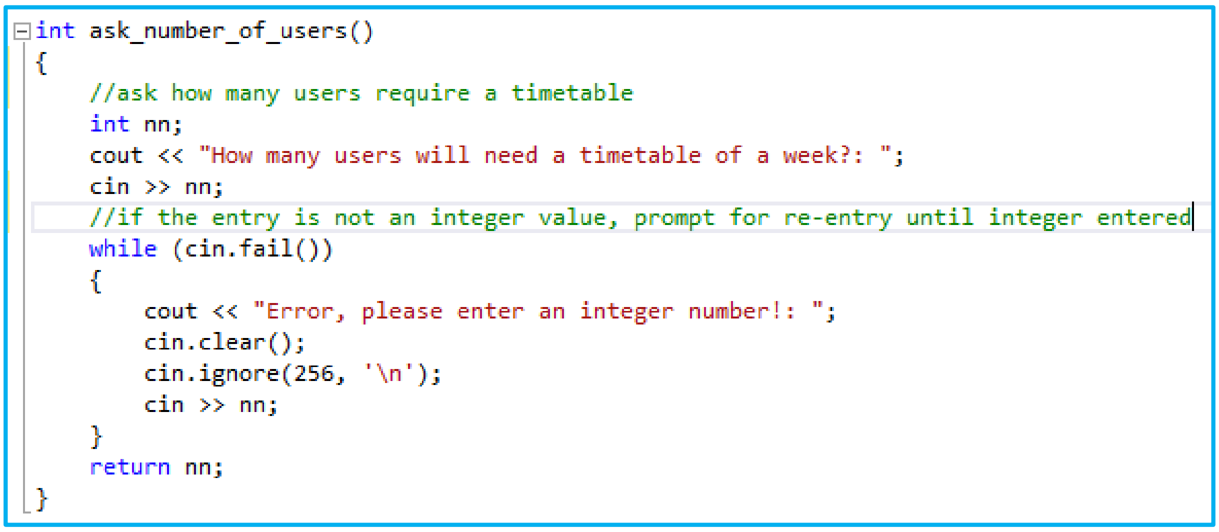
Figure 26 - Error Message to Highlight a Conflict Has Occurred

If this if statement was not present the program would simply overwrite the initial activity without informing the user. The “if” statement therefore informs the user so they would not be receiving an incorrect timetable unknowingly. This meant the fifth step of the User Specification was complete,

6.6 Implement Multi-user Functionality

The final step of the User Specification was that multiple users should be able to generate a timetable. This was a very simple piece of functionality to implement as the entire program could be placed into a loop which would iterate for a user defined number of times. The program would ask the user how many people would like a timetable and loop for that number of times.

A function called “ask_number_of_users” was created again in the “otherfunctions” source file, which would allow the user to enter a number of users requiring a timetable. The function was created to keep the amount of code in the main source file to a minimum thus improving readability. The function can be seen in Figure 27.

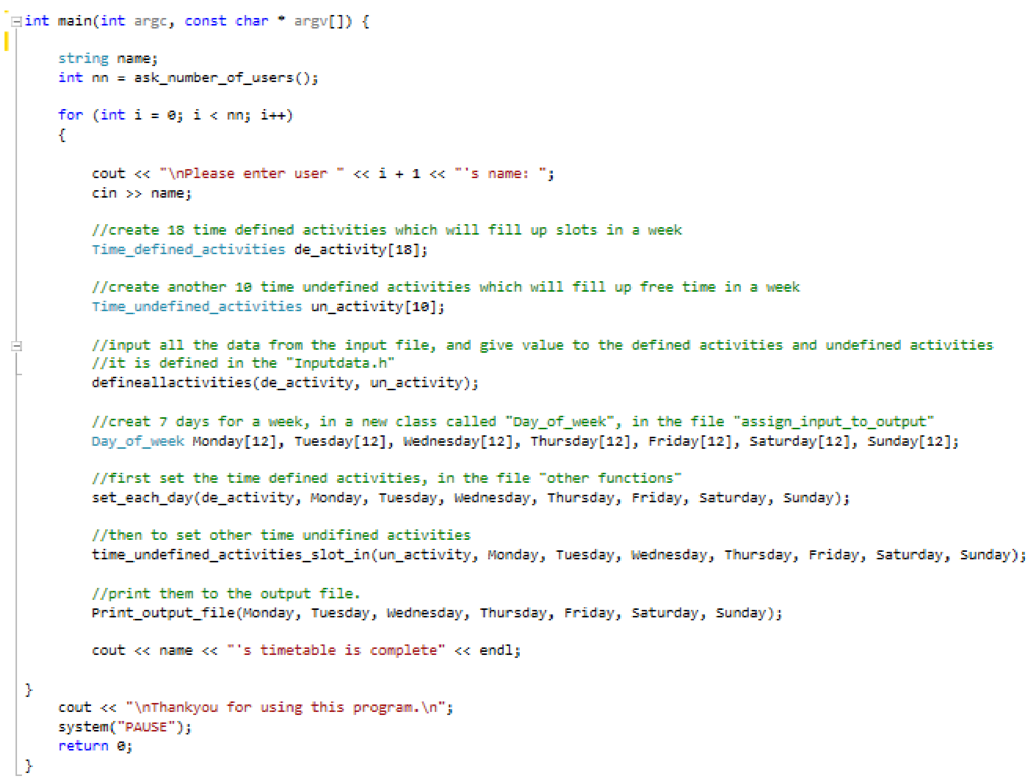


```
int ask_number_of_users()
{
    //ask how many users require a timetable
    int nn;
    cout << "How many users will need a timetable of a week?: ";
    cin >> nn;
    //if the entry is not an integer value, prompt for re-entry until integer entered
    while (cin.fail())
    {
        cout << "Error, please enter an integer number!: ";
        cin.clear();
        cin.ignore(256, '\n');
        cin >> nn;
    }
    return nn;
}
```

Figure 27 - Function to ask Number of Users Requiring a Timetable

The function can be seen to force the user to enter an integer. If the user entered anything other than an integer the program would instantly crash. This step made the program more robust.

The loop which then implemented the multiuser functionality can be seen in Figure 28.



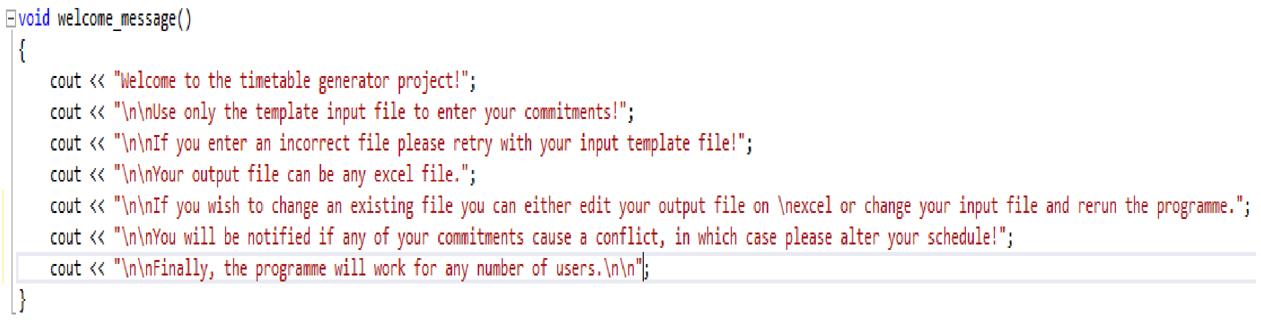
```
int main(int argc, const char * argv[]) {

    string name;
    int nn = ask_number_of_users();

    for (int i = 0; i < nn; i++)
    {

        cout << "\nPlease enter user " << i + 1 << "'s name: ";
        cin >> name;

        //create 18 time defined activities which will fill up slots in a week
        Time_defined_activities de_activity[18];

        //create another 10 time undefined activities which will fill up free time in a week
        Time_undefined_activities un_activity[10];

        //input all the data from the input file, and give value to the defined activities and undefined activities
        //it is defined in the "Inputdata.h"
        defineallactivities(de_activity, un_activity);

        //creat 7 days for a week, in a new class called "Day_of_week", in the file "assign_input_to_output"
        Day_of_week Monday[12], Tuesday[12], Wednesday[12], Thursday[12], Friday[12], Saturday[12], Sunday[12];

        //first set the time defined activities, in the file "other functions"
        set_each_day(de_activity, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);

        //then to set other time undifined activities
        time_undefined_activities_slot_in(un_activity, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);

        //print them to the output file.
        Print_output_file(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);

        cout << name << "'s timetable is complete" << endl;

}
    cout << "\nThankyou for using this program.\n";
    system("PAUSE");
    return 0;
}
```

Figure 28 - Main Source File Showing Multi-User Functionality

From Figure 28, it is evident that the code has been improved to add the user's name. When the user's timetable has been created, it will inform the specific user that their timetable is ready. All steps of the User Specification had now been complete however an extra function was added to create a welcome message which would assist the user with using the program.

6.7 Welcome Message

The welcome message contains brief instructions to assist the user with operating the program. This can be seen in Figure 29.



```
void welcome_message()
{
    cout << "Welcome to the timetable generator project!";
    cout << "\n\nUse only the template input file to enter your commitments!";
    cout << "\n\nIf you enter an incorrect file please retry with your input template file!";
    cout << "\n\nYour output file can be any excel file.";
    cout << "\n\nIf you wish to change an existing file you can either edit your output file on \nexcel or change your input file and rerun the programme.";
    cout << "\n\nYou will be notified if any of your commitments cause a conflict, in which case please alter your schedule!";
    cout << "\n\nFinally, the programme will work for any number of users.\n\n";
}
```

Figure 29 - Welcome Message Function

The code was now fully developed and ready to be tested.

7 Results and Discussion

To test the program was operating correctly the Test Specification from Figure 6 and Figure 7 was used as a guideline on how the software would be tested. For each piece of functionality, an extensive variety of inputs will be used and the output for each will be provided. For all testing purposes, the welcome message would be removed to decrease the size of the command window. For all tests other than the multi-user functionality, the number of users will be one and the user name will be "user". This will provide consistency.

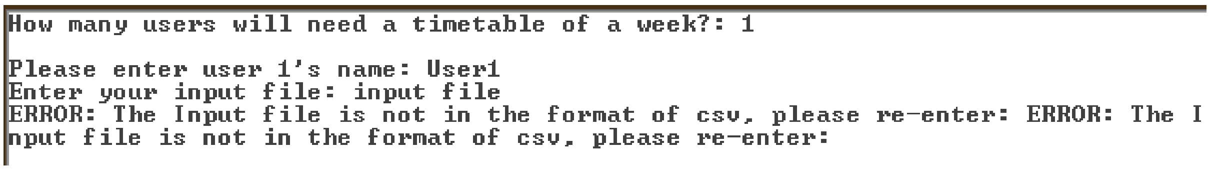
7.1 Open the User Input CSV File

To test if the program could open the user input file, there was only one condition which would have to be met. The condition was that the file type entered would have to be of CSV format as this was the format in which the whole development of the code was based upon. The implementation of this requirement can be seen in Figure 10. There would be several inputs which could test this functionality. When prompted for the input file the user could input:

- A string such as "a" or "input file".
- A numerical character such as "1".
- The phrase ".csv".
- A file not of CSV format such as a .txt document.
- A CSV file which contains no data.
- A correct format CSV file.

7.1.1 Input of a String

The program was run and "input file" was entered as the input file input which can be seen in Figure 30.



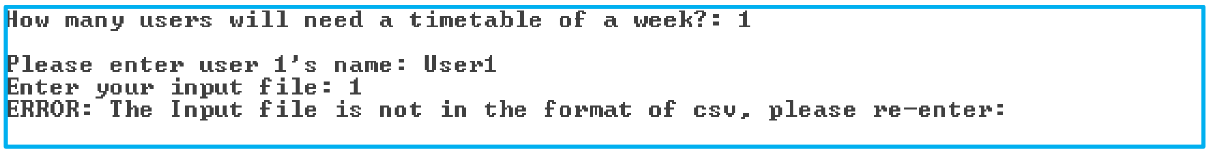
```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: input file
ERROR: The Input file is not in the format of csv, please re-enter: ERROR: The I
nput file is not in the format of csv, please re-enter:
```

Figure 30 - String Input for Input File

Entering "input file" causes the error message to be printed twice. This is due to the program treating "input" and "file" as two different words. For each word it searches for ".csv" which it cannot find thus returning the error message twice. This is what would be expected and allows the user to re-enter their CSV file. The process is slightly untidy as the error message is printed out twice however there is no feasible way to correct this.

7.1.2 Numerical Input

The next input to be tested was that of a number. The number used was “1” as seen in Figure 31.



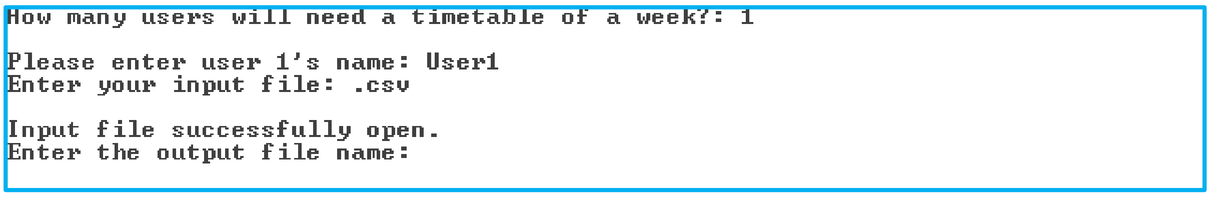
```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: 1
ERROR: The Input file is not in the format of csv, please re-enter:
```

Figure 31 - Numerical Entry for Input File

Entering a number caused the expected output of the program asking the user to re-enter.

7.1.3 Entering the Phrase “.csv”

The program looks for the term “.csv” which would indicate the file being entered was of the CSV format. Entering only “.csv” would not cause an error and the program would operate as normal. It would simply read in no data as the file being read in did not exist. This can be seen in Figure 32.

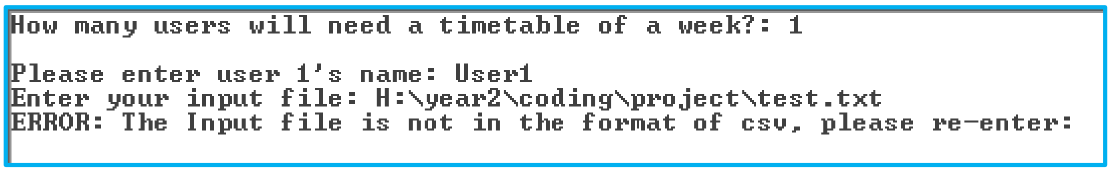


```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: .csv

Input file successfully open.
Enter the output file name:
```

Figure 32 - ".csv" Entry for Input File

7.1.4 Using a Different Format of File

Entering a different file format would cause the program to prompt the user to re-enter their input. This is evident from Figure 33, where the file “test.txt” was used.

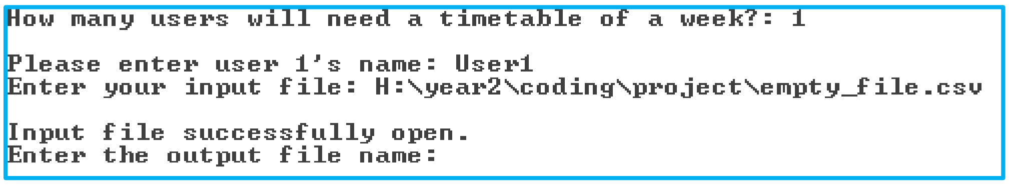


```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: H:\year2\coding\project\test.txt
ERROR: The Input file is not in the format of csv, please re-enter:
```

Figure 33 - Alternate File Format Used for Input File

7.1.5 Using a CSV File Which Contains No Data

For a CSV file with no data, the program will run as normal. This can be seen in Figure 34.

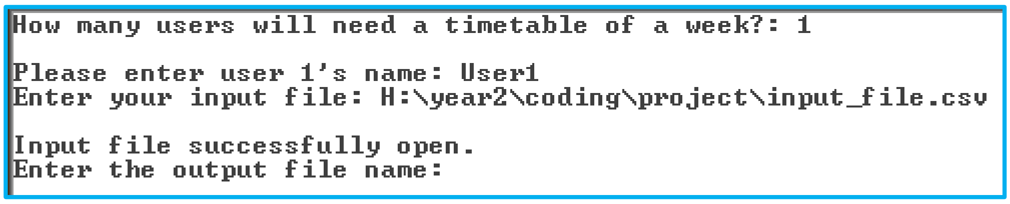


```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: H:\year2\coding\project\empty_file.csv

Input file successfully open.
Enter the output file name:
```

Figure 34 - Empty File Used for Input File

7.1.6 Using a CSV File Which Contained Data

Using a filled CSV file would cause the program to operate in exactly the same manner as Figure 34. This is due to both files being of the correct CSV file format. This can be seen in Figure 35, when the file from Figure 9 was used.



```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: H:\year2\coding\project\input_file.csv

Input file successfully open.
Enter the output file name:
```

Figure 35 - Correct CSV File Used as Input File

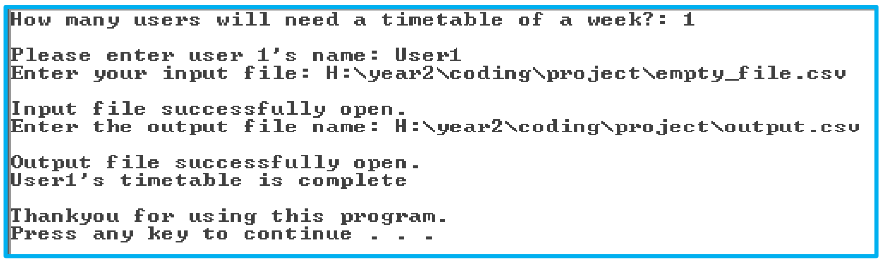
The first piece of functionality could now be said to be functioning correctly.

7.2 Organise a Time in the Week for the User Activities

For this functionality, only time defined activities would be considered as time undefined activities would be tested at a later stage. The file in Figure 38, was created to establish a small amount of test data to be considered. Since the file being used was a defined template, no errors should occur. Two tests would be carried out for this functionality, the first would be an empty CSV file to establish what the output would be for an empty input file. The second would have five activities all with a lab lasting two hours and a lecture lasting for one hour.

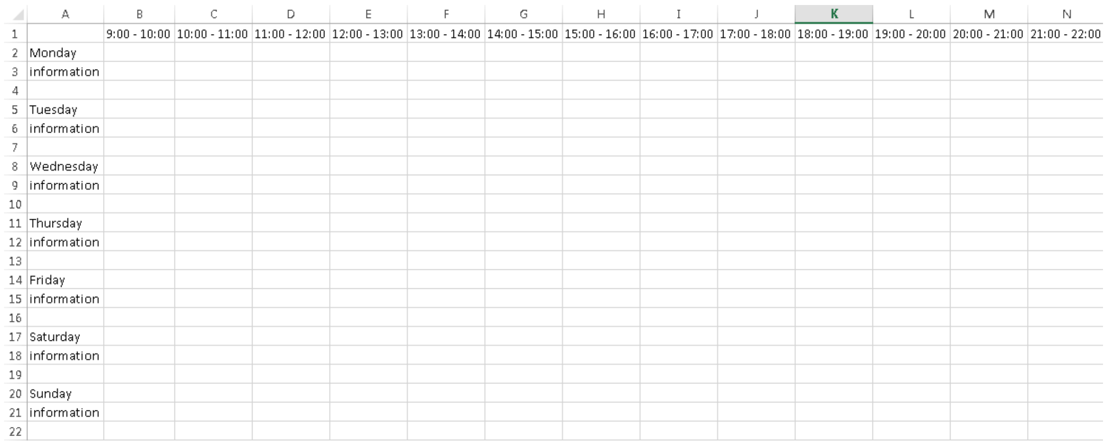
7.2.1 Empty Input File

The empty file was used to establish the output timetable when no input data had been entered. The command window can be seen in Figure 36, while the output timetable could be seen in Figure 37.



```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: H:\year2\coding\project\empty_file.csv

Input file successfully open.
Enter the output file name: H:\year2\coding\project\output.csv

Output file successfully open.
User1's timetable is complete

Thankyou for using this program.
Press any key to continue . . .
```

Figure 36 - Command Window of Empty CSV File Being Entered



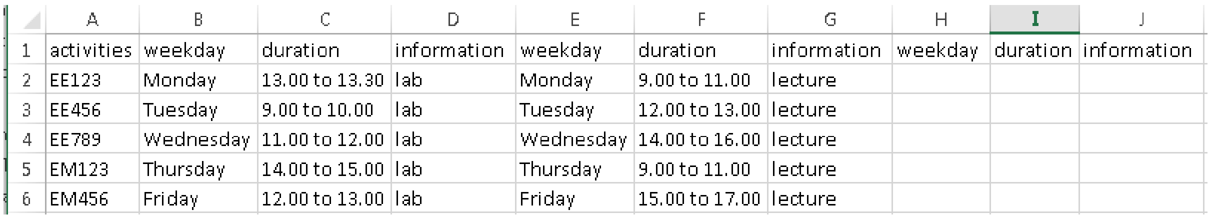
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday													
3	information													
4														
5	Tuesday													
6	information													
7														
8	Wednesday													
9	information													
10														
11	Thursday													
12	information													
13														
14	Friday													
15	information													
16														
17	Saturday													
18	information													
19														
20	Sunday													
21	information													
22														

Figure 37 - Empty Timetable Generated for Empty Input File

It can be seen the timetable which was generated was empty. This meant the program was able to correctly deal with a file which contained no data.

7.2.2 Input File Containing Test Data.

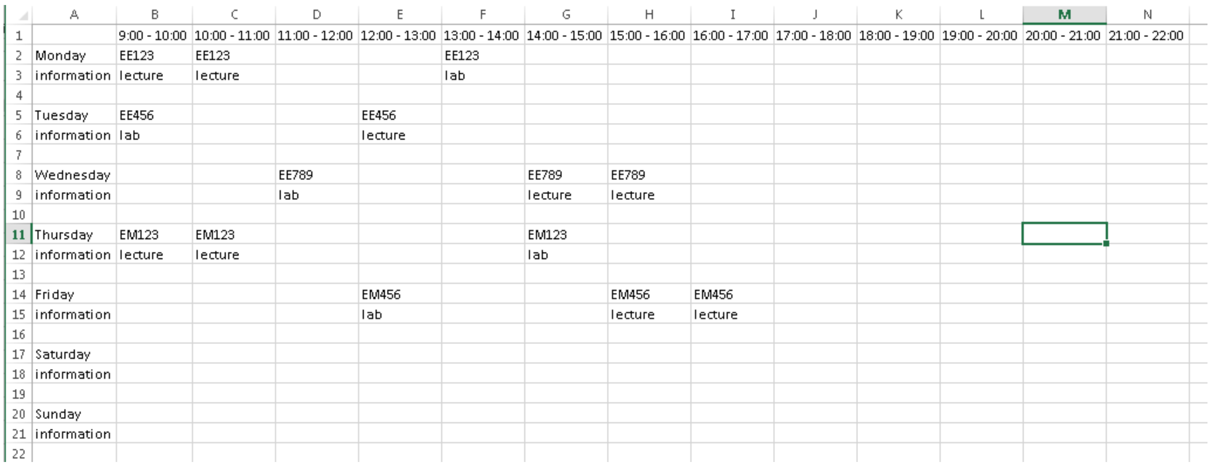
The file was created with five different fictional academic activities, all with a one hour lab and a two hour lecture. These all occurred at different times. For the EE123 activity, the lab was set to half an hour, to show how the program would deal with an activity which did not last for one whole hour. The input file can be seen in Figure 38.



	A	B	C	D	E	F	G	H	I	J
1	activities	weekday	duration	information	weekday	duration	information	weekday	duration	information
2	EE123	Monday	13.00 to 13.30	lab	Monday	9.00 to 11.00	lecture			
3	EE456	Tuesday	9.00 to 10.00	lab	Tuesday	12.00 to 13.00	lecture			
4	EE789	Wednesday	11.00 to 12.00	lab	Wednesday	14.00 to 16.00	lecture			
5	EM123	Thursday	14.00 to 15.00	lab	Thursday	9.00 to 11.00	lecture			
6	EM456	Friday	12.00 to 13.00	lab	Friday	15.00 to 17.00	lecture			

Figure 38 - Input File Containing Test Data

The input file is simply the template containing test data. There are more cells which are empty which could be filled with another instance of each activity. The Output generated can be seen in Figure 39.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday	EE123	EE123			EE123								
3	information	lecture	lecture			lab								
4														
5	Tuesday	EE456			EE456									
6	information	lab			lecture									
7														
8	Wednesday			EE789			EE789	EE789						
9	information			lab			lecture	lecture						
10														
11	Thursday	EM123	EM123				EM123							
12	information	lecture	lecture				lab							
13														
14	Friday				EM456			EM456	EM456					
15	information				lab			lecture	lecture					
16														
17	Saturday													
18	information													
19														
20	Sunday													
21	information													
22														

Figure 39 - Timetable Produced From Input File Containing Test Data

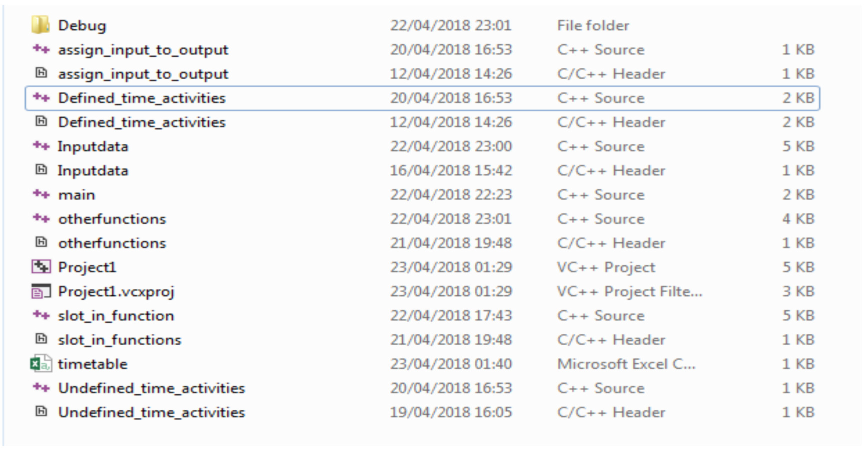
By comparing Figure 38 and Figure 39, it can be seen all activities are present in the generated timetable. For the activity which only lasts for half of one hour, the program simply fills the whole hour. This is due to the time incrementing in one hour steps. The code would have to be altered to increment in half hour steps in order to only fill a cell showing half of one hour. This piece of functionality was deemed fully functioning.

7.3 Open an Excel Output File Containing a Template Timetable

This piece of functionality operates in a slightly different way to what was specified in the User Specification. The output file does not contain a template timetable, but instead a template timetable is generated within the file each time the file is entered into the program. The output file is cleared to ensure there is no pre-existing data in the file, before the data is printed to the file. The program takes in the user output file using the same method as taking in the user input file. The program searches for the ".csv" term in the user input to determine whether the file is of the CSV format. This piece of the process has already been tested in section 7.1. The only aspect which had to be tested would be the actual output of timetable template. This has also been carried out in previous sections however this section will cover it in slightly more detail.

7.3.1Using the Name of a Non-Existing Output File

The code operated in entirely the same way as in section 7.1, however a new outcome occurred with certain inputs. If the user used a string accompanied by ".csv" as their input, the program would create a CSV file of the name that the user inserted as the string. This file would be saved in the location the entire project was saved in. This created a new piece of functionality which was unintentional but allowed the program to be more easily used as the user could easily create any file by simply entering, for example "timetable.csv". This meant the user did not require to have an output pre-made blank output file. It is recommended that the user would input a premade output CSV file however, as the location of the file would be known to the user. The input "timetable.csv", which was a file which did not exist, was saved in the location seen in Figure 40.



Debug	22/04/2018 23:01	File folder	
assign_input_to_output	20/04/2018 16:53	C++ Source	1 KB
assign_input_to_output	12/04/2018 14:26	C/C++ Header	1 KB
Defined_time_activities	20/04/2018 16:53	C++ Source	2 KB
Defined_time_activities	12/04/2018 14:26	C/C++ Header	2 KB
Inputdata	22/04/2018 23:00	C++ Source	5 KB
Inputdata	16/04/2018 15:42	C/C++ Header	1 KB
main	22/04/2018 22:23	C++ Source	2 KB
otherfunctions	22/04/2018 23:01	C++ Source	4 KB
otherfunctions	21/04/2018 19:48	C/C++ Header	1 KB
Project1	23/04/2018 01:29	VC++ Project	5 KB
Project1.vcxproj	23/04/2018 01:29	VC++ Project Filte...	3 KB
slot_in_function	22/04/2018 17:43	C++ Source	5 KB
slot_in_functions	21/04/2018 19:48	C/C++ Header	1 KB
timetable	23/04/2018 01:40	Microsoft Excel C...	1 KB
Undefined_time_activities	20/04/2018 16:53	C++ Source	1 KB
Undefined_time_activities	19/04/2018 16:05	C/C++ Header	1 KB

Figure 40 - Location of Timetable Created by Entering an Output File Which Did Not Exist

It can be seen the file would be saved inside the project folder which contains the debug folder along with all header and source files included in the project. This is not a location which the user would be familiar with thus creating the recommendation of the user entering a premade file with a known location.

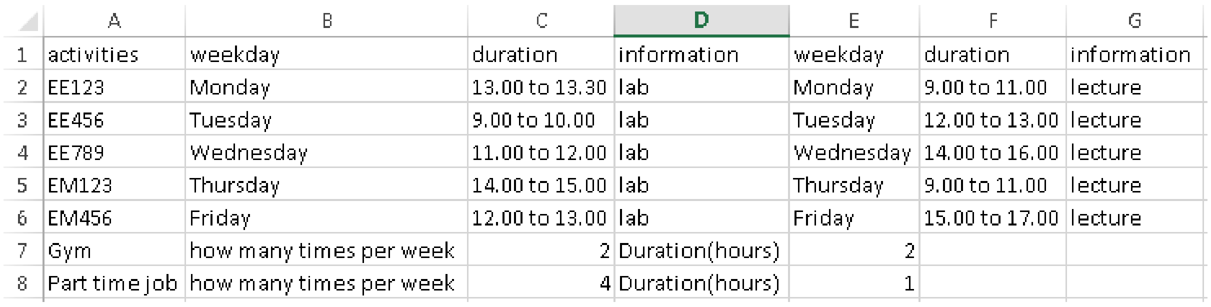
7.3.2 Using the Name and Address of a Premade Output File

Using a pre-existing file, the output would simply be a timetable depending on the input file. Using either a pre-existing file, or a non-existent file would create the same timetable output, just having a different location. The output created when using a blank input file, can be seen in Figure 37, as this had already been tested.

This meant the opening and outputting of a template timetable could be deemed fully functioning.

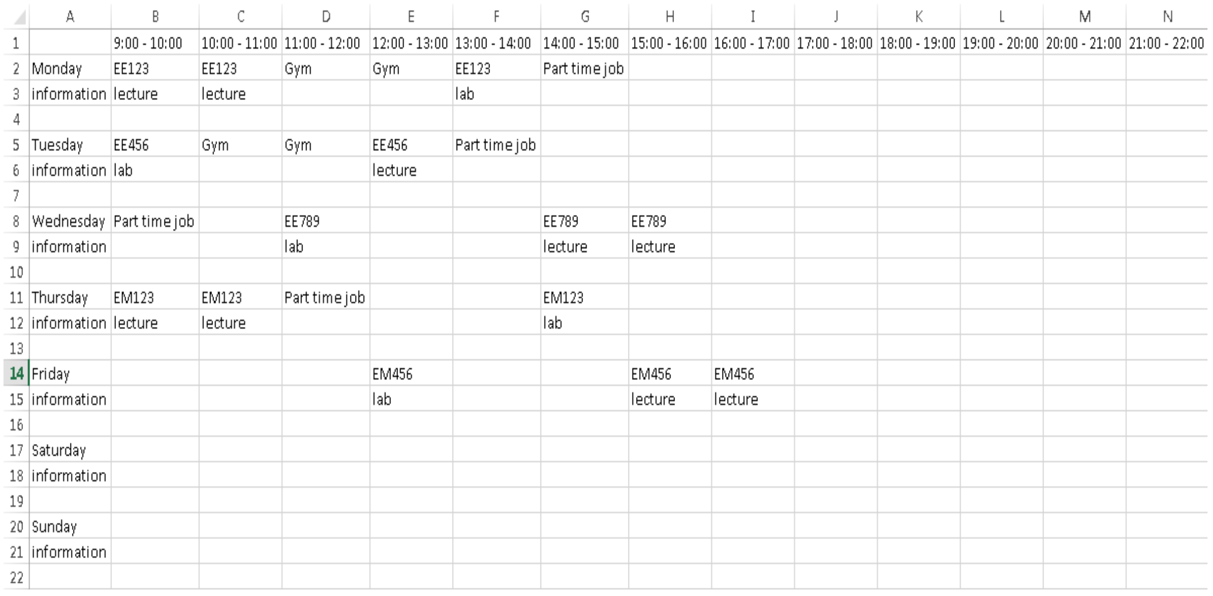
7.4 Assigning Activities to Slots Which Were Not Already Filled

This functionality refers to the programs ability to assign time slots for activities which were not time defined. To test this functionality, the input file from Figure 38 was edited to include two time undefined activities. The activities called “Gym” and “Part time job” were inserted under the time defined activities. As with all input files, the outlined format for the data had to be adhered to. The input file can be seen in Figure 41, and the timetable generated can be seen in Figure 42.



	A	B	C	D	E	F	G
1	activities	weekday	duration	information	weekday	duration	information
2	EE123	Monday	13.00 to 13.30	lab	Monday	9.00 to 11.00	lecture
3	EE456	Tuesday	9.00 to 10.00	lab	Tuesday	12.00 to 13.00	lecture
4	EE789	Wednesday	11.00 to 12.00	lab	Wednesday	14.00 to 16.00	lecture
5	EM123	Thursday	14.00 to 15.00	lab	Thursday	9.00 to 11.00	lecture
6	EM456	Friday	12.00 to 13.00	lab	Friday	15.00 to 17.00	lecture
7	Gym	how many times per week	2	Duration(hours)	2		
8	Part time job	how many times per week	4	Duration(hours)	1		

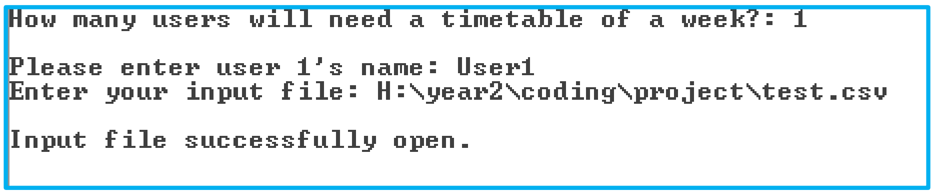
Figure 41 - Input File to Test Time Slotting for Activities of Undefined Time



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday	EE123	EE123	Gym	Gym	EE123	Part time job							
3	information	lecture	lecture			lab								
4														
5	Tuesday	EE456	Gym	Gym	EE456	Part time job								
6	information	lab			lecture									
7														
8	Wednesday	Part time job		EE789			EE789	EE789						
9	information			lab			lecture	lecture						
10														
11	Thursday	EM123	EM123	Part time job			EM123							
12	information	lecture	lecture				lab							
13														
14	Friday				EM456			EM456	EM456					
15	information				lab			lecture	lecture					
16														
17	Saturday													
18	information													
19														
20	Sunday													
21	information													
22														

Figure 42 - Timetable Generated Showing Undefined Time Activities Included

The undefined time activities were included for the number of times they occur, for the length of time which they occur. If the week was completely taken up with time defined activities, which is a very unlikely case, the time undefined activities should in theory simply be ignored as time defined activities have higher priority. This was not the case however as is seen in the command window in Figure 43. The input file user contained an activity for each day which lasted the entire day meaning there would be no spaces for time undefined activities. The program entered an infinite loop trying to assign undefined time activities in spaces which did not exist.



```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: H:\year2\coding\project\test.csv

Input file successfully open.
```

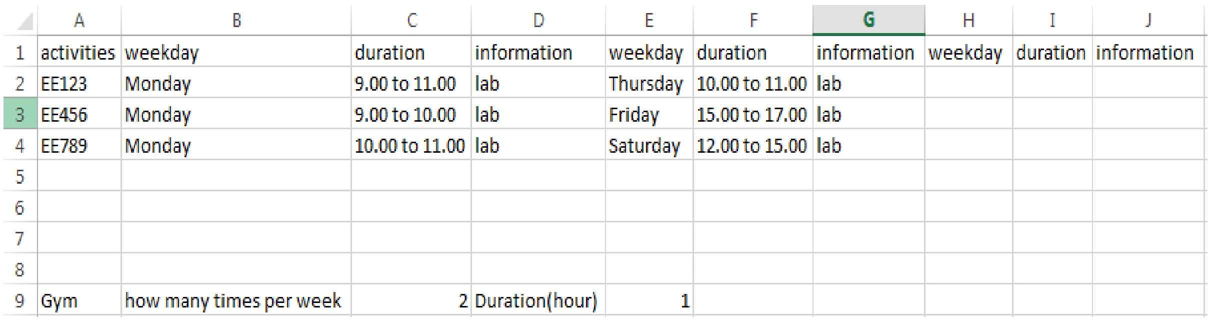
Figure 43 - Command Window Crashing Due to Infinite Loop

There was no feasible method to correct this error as there was no simple way to implement a piece of code which would handle the condition of no free space being available. An unavoidable fault had been found which reduced the programs robustness but was an event which was highly unlikely as the user would rarely have time defined activities taking up the entirety of their week.

This piece of functionality could be deemed partially operational due to the fault which had been found.

7.5 If a Conflict Occurs the User Should be Notified

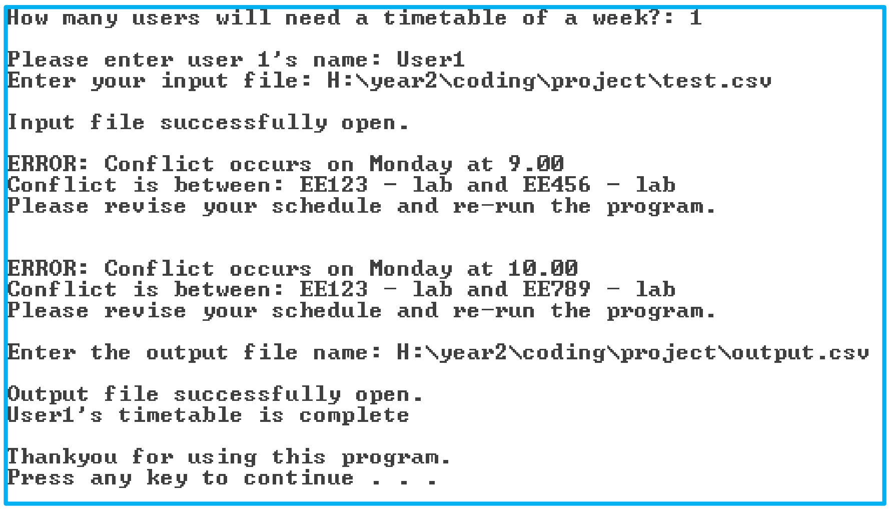
The fifth piece of functionality which would be tested was the conflict detection system. From the testing specification in Figure 5, the expected output can be seen to be "A clash has occurred, please revise your input file". This was updated to present an error message which contained what time the conflict occurs and what activities the conflict occurs between. A new input file which contained three specific conflicts was created as seen in Figure 44. The program should inform the user there is a conflict however the timetable would still be generated. The first activity to be read in would be the entry presented in the output timetable. The command window is present in Figure 45, and the created timetable in Figure 46.



	A	B	C	D	E	F	G	H	I	J
1	activities	weekday	duration	information	weekday	duration	information	weekday	duration	information
2	EE123	Monday	9.00 to 11.00	lab	Thursday	10.00 to 11.00	lab			
3	EE456	Monday	9.00 to 10.00	lab	Friday	15.00 to 17.00	lab			
4	EE789	Monday	10.00 to 11.00	lab	Saturday	12.00 to 15.00	lab			
5										
6										
7										
8										
9	Gym	how many times per week	2	Duration(hour)	1					

Figure 44 - Input File Containing Conflicts

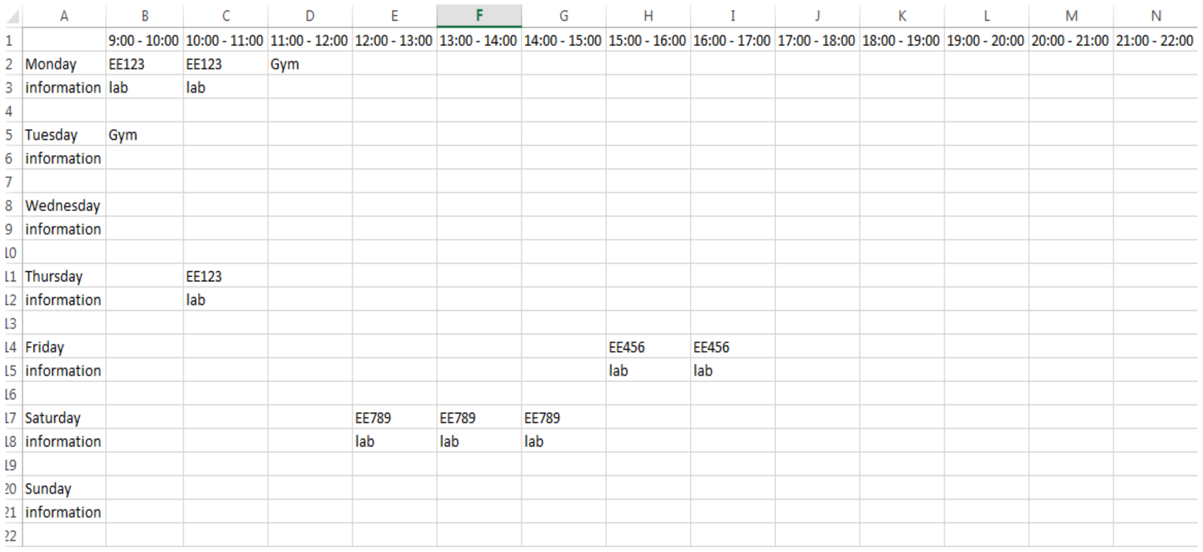
The conflicts contained within the file occur from 9.00 to 11.00. The program highlights every hour that a conflict occurs meaning there should be two error messages as there two conflicts. Running the program using this input created the following command window.



```
How many users will need a timetable of a week?: 1

Please enter user 1's name: User1
Enter your input file: H:\year2\coding\project\test.csv

Input file successfully open.

ERROR: Conflict occurs on Monday at 9.00
Conflict is between: EE123 - lab and EE456 - lab
Please revise your schedule and re-run the program.


ERROR: Conflict occurs on Monday at 10.00
Conflict is between: EE123 - lab and EE789 - lab
Please revise your schedule and re-run the program.

Enter the output file name: H:\year2\coding\project\output.csv

Output file successfully open.
User1's timetable is complete

Thankyou for using this program.
Press any key to continue . . .
```

Figure 45 - Command Window Informing User of Conflicts

The conflicts can be clearly seen to show what time the conflict occurs and also what activities the conflict is between.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday	EE123	EE123	Gym										
3	information	lab	lab											
4														
5	Tuesday	Gym												
6	information													
7														
8	Wednesday													
9	information													
10														
11	Thursday		EE123											
12	information		lab											
13														
14	Friday							EE456	EE456					
15	information							lab	lab					
16														
17	Saturday				EE789	EE789	EE789							
18	information				lab	lab	lab							
19														
20	Sunday													
21	information													
22														

Figure 46 - Output Created From Conflicted Input File

The timetable only contains the first activity at the time the conflict occurs. This is the expected output as the program has instructed the user to revise their input their input file meaning the user will know this timetable is invalid. This resulted in this piece of functionality being fully functional.

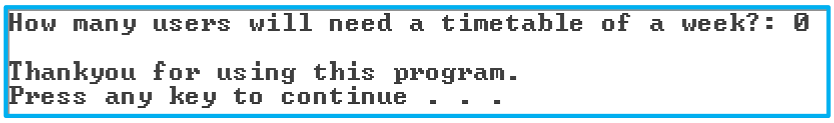
7.6 Multi-User Functionality

The multi-user functionality was implemented through the use of a loop. To test this a variety of different inputs were used:

- Setting the number of users to zero.
- Inserting a character/string as the number of users.
- Setting number of users to three.
- Setting number of users to a large number such as one hundred.
- Setting number of users to a negative number.
- Setting number of users to a float.

7.6.1 Setting Number of Users to Zero

When the number of users was set to zero, the program should simply skip to the end of the code and thank the user for using the program. This is evident from the command window in Figure 47.



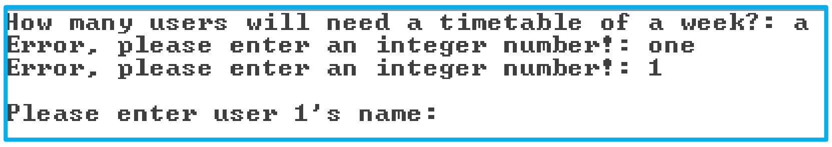
```
How many users will need a timetable of a week?: 0

Thankyou for using this program.
Press any key to continue . . .
```

Figure 47 - Command Window for Zero Users

The expected output of thanking the user for using the program occurred meaning the program operated as expected.

7.6.2 Setting Number of Users to a Character or String

The input of a character or string as the number of users should theoretically prompt the user to re-enter a number which is an integer. This is present in the command window in Figure 48.



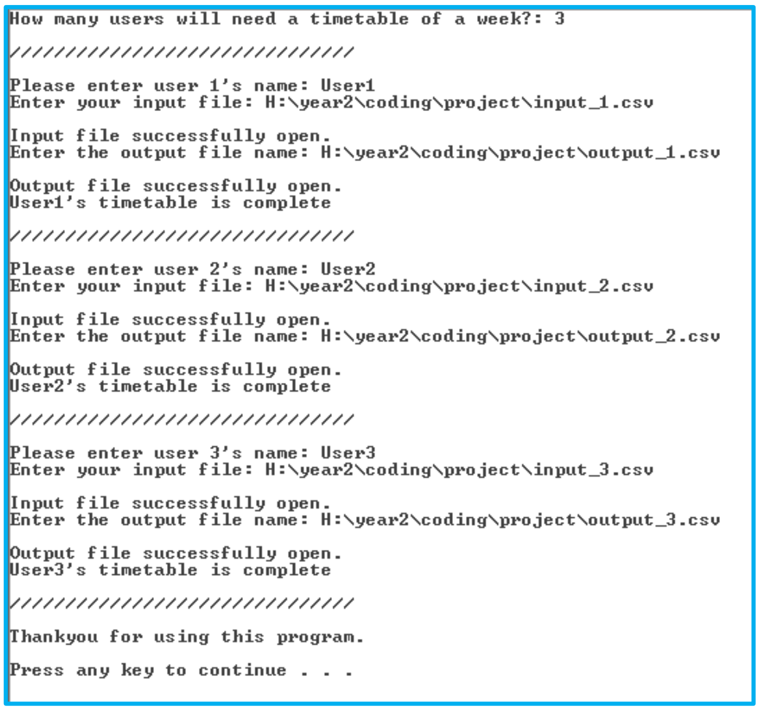
```
How many users will need a timetable of a week?: a
Error, please enter an integer number!: one
Error, please enter an integer number!: 1

Please enter user 1's name:
```

Figure 48 - Command Window for String and Character Number of Users

The program prompts the user to re-enter their input as an integer meaning the program responds correctly.

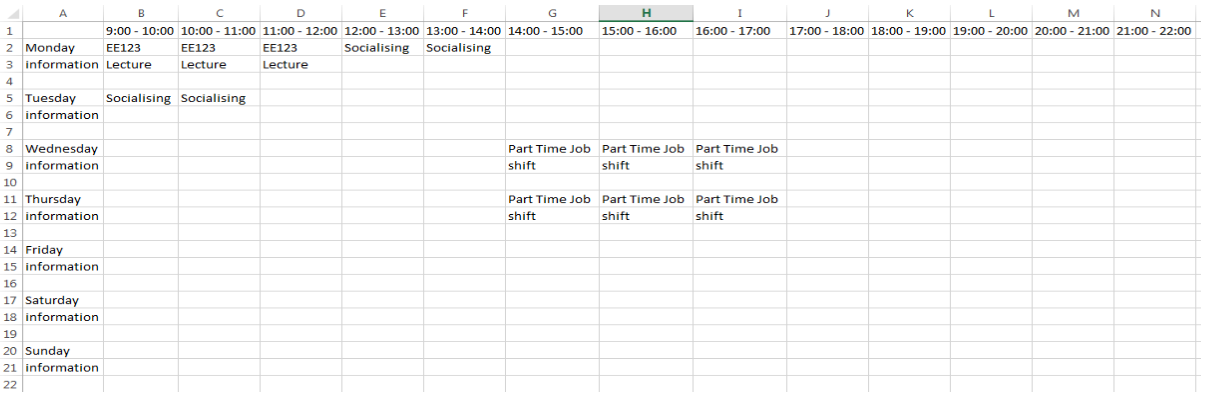
7.6.3 Setting Number of Users to Three

Entering a realistic number of users such as three should cause the program to generate three different timetables. Three basic input files were created to generate three different output files one for each user. An output of "/////////////////////////" was used to increase the readability of the command window by separating each users inputs. The command window can be seen in Figure 49. The three created timetables can be seen in Figure 50, Figure 51 and Figure 52.



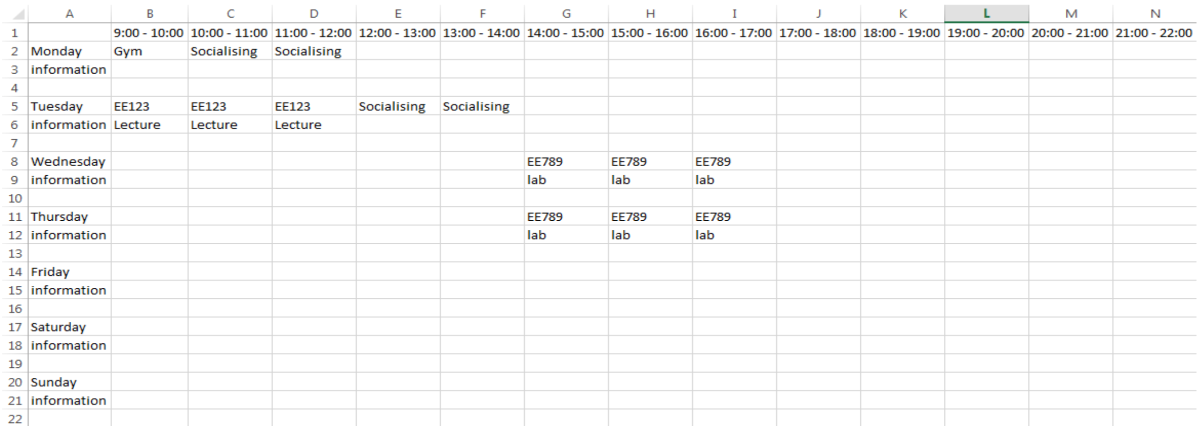
```
How many users will need a timetable of a week?: 3

///////////////////////////////

Please enter user 1's name: User1
Enter your input file: H:\year2\coding\project\input_1.csv

Input file successfully open.
Enter the output file name: H:\year2\coding\project\output_1.csv

Output file successfully open.
User1's timetable is complete

///////////////////////////////

Please enter user 2's name: User2
Enter your input file: H:\year2\coding\project\input_2.csv

Input file successfully open.
Enter the output file name: H:\year2\coding\project\output_2.csv

Output file successfully open.
User2's timetable is complete

///////////////////////////////

Please enter user 3's name: User3
Enter your input file: H:\year2\coding\project\input_3.csv

Input file successfully open.
Enter the output file name: H:\year2\coding\project\output_3.csv

Output file successfully open.
User3's timetable is complete

///////////////////////////////

Thankyou for using this program.

Press any key to continue . . .
```

Figure 49 - Command Window for Three Users



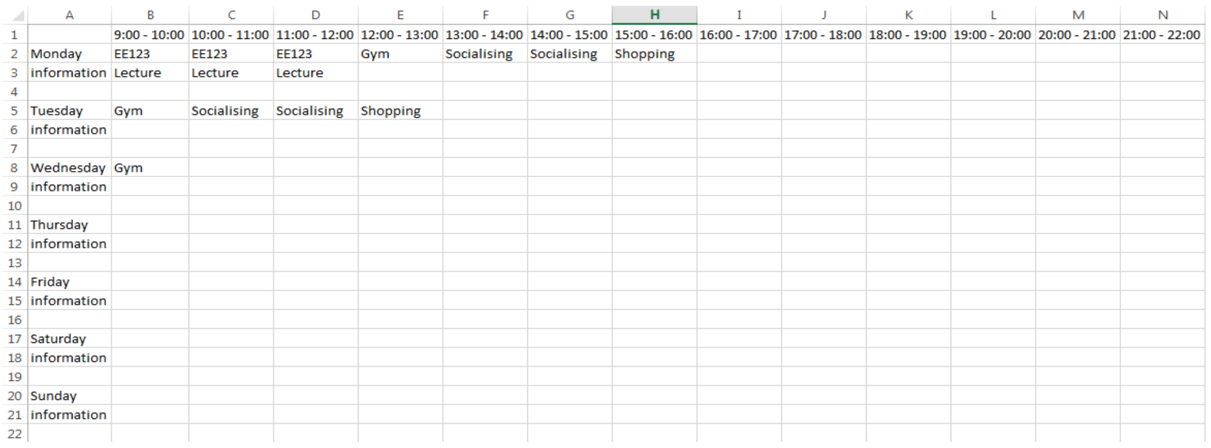
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday	EE123	EE123	EE123	Socialising	Socialising								
3	information	Lecture	Lecture	Lecture										
4														
5	Tuesday	Socialising	Socialising											
6	information													
7														
8	Wednesday						Part Time Job	Part Time Job	Part Time Job					
9	information						shift	shift	shift					
10														
11	Thursday						Part Time Job	Part Time Job	Part Time Job					
12	information						shift	shift	shift					
13														
14	Friday													
15	information													
16														
17	Saturday													
18	information													
19														
20	Sunday													
21	information													
22														

Figure 50 - Output Timetable for User1



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday	Gym	Socialising	Socialising										
3	information													
4														
5	Tuesday	EE123	EE123	EE123	Socialising	Socialising								
6	information	Lecture	Lecture	Lecture										
7														
8	Wednesday						EE789	EE789	EE789					
9	information						lab	lab	lab					
10														
11	Thursday						EE789	EE789	EE789					
12	information						lab	lab	lab					
13														
14	Friday													
15	information													
16														
17	Saturday													
18	information													
19														
20	Sunday													
21	information													
22														

Figure 51 - Output Timetable for User2



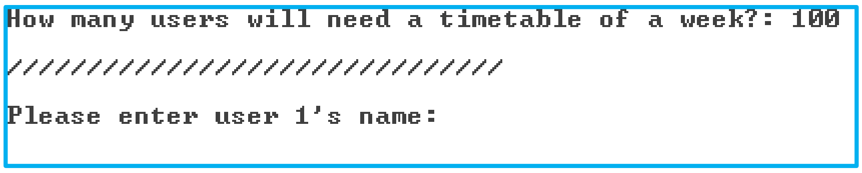
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 13:00	13:00 - 14:00	14:00 - 15:00	15:00 - 16:00	16:00 - 17:00	17:00 - 18:00	18:00 - 19:00	19:00 - 20:00	20:00 - 21:00	21:00 - 22:00
2	Monday	EE123	EE123	EE123	Gym	Socialising	Socialising	Shopping						
3	information	Lecture	Lecture	Lecture										
4														
5	Tuesday	Gym	Socialising	Socialising	Shopping									
6	information													
7														
8	Wednesday	Gym												
9	information													
10														
11	Thursday													
12	information													
13														
14	Friday													
15	information													
16														
17	Saturday													
18	information													
19														
20	Sunday													
21	information													
22														

Figure 52 - Output Timetable for User3

The program generated three unique timetables, with a clear and easy to use command window meaning this test operated as expected.

7.6.4 Entering a Large Number of Users

A large number of users was entered which should run as normal. No input files or Output files were used as only the command window would be required to show the program could deal with a large number of users. An input of "100" was used to test this.



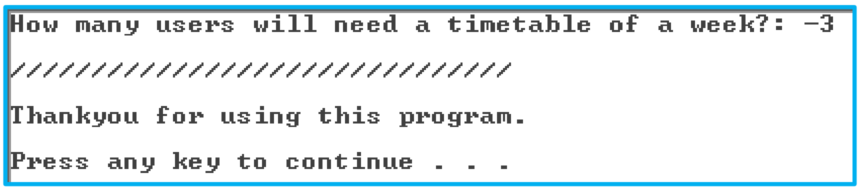
```
How many users will need a timetable of a week?: 100

////////////////////////////////

Please enter user 1's name:
```

Figure 53 - Command Window for 100 Users

For a large number of users the program operates as expected.

7.6.5 Entering a Negative Number of Users

Integers are defined as numbers which are not fractions. This means that entering a negative number will be a valid input. The code theoretically treat a negative number as a zero and thus not iterate through the loop, thus ending the program. Shows this.



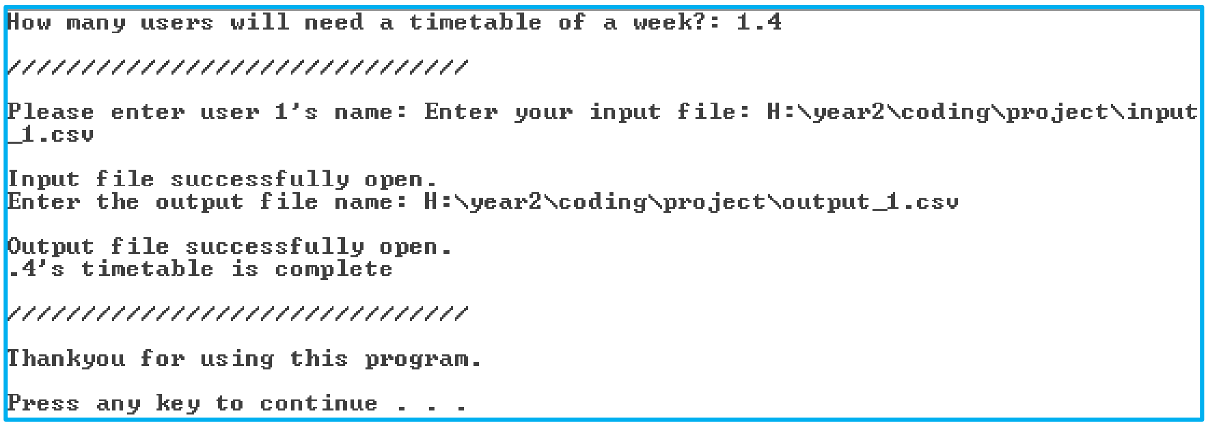
```
How many users will need a timetable of a week?: -3

////////////////////////////////

Thankyou for using this program.

Press any key to continue . . .
```

Figure 54 - Command Window for Negative Number of Users

For a negative number of users the program can be seen to simply not iterate through the loop thus treating the number of users as zero. This is the expected operation of the program under this input.

7.6.6 Entering a Float

When a float is entered instead of an integer number, the program theoretically should request the user to re-enter their input. From Figure 54, it can be seen that this is not the case.



```
How many users will need a timetable of a week?: 1.4

/////////////////////////////////

Please enter user 1's name: Enter your input file: H:\year2\coding\project\input
_1.csv

Input file successfully open.
Enter the output file name: H:\year2\coding\project\output_1.csv

Output file successfully open.
.4's timetable is complete

/////////////////////////////////

Thankyou for using this program.

Press any key to continue . . .
```

Figure 55 - Command Window for Float Entry as Number of Users

The outcome of a float entry is that it uses the number before the decimal point as the number of users. It then sets the decimal point and the following number as a string which it sets as the first users name in this case ".4". This was not the expected outcome however it did not cause the code to crash so it could be ignored.

This functionality was deemed to be operating correctly.

7.7 Edit a Timetable after it Had Been Created

The final piece of functionality did not need to be tested as the user has two options to edit an existing timetable. Due to the timetable being created on excel file the user could easily alter it by altering the excel file manually. The user could also alter their input file and re-run the program to create an updated file. This allowed the user to have freedom when editing their timetables. The user could also print their timetable easily using excel printing capabilities. This was the reason excel files were used over a graphical user interface.

All functionalities of the code had now been tested meaning the program was deemed fully operational. The code was fairly robust as it would only crash if the week of the user was full of time defined activities and they had entered any undefined time activities. The readability of the code was improved by using comments to indicate the operation of the code at various points and also through the use of indentation to visualise loops and functions.

8 Further Work and Evaluation

The code could be developed further in many ways.

1. The implementation of a graphical user interface would allow the code to become more visually pleasing and also more versatile. Excel provides a more simplistic timetable but one which is easily edited and easily printed.
2. The code could be altered to allow a string input for the input file to only cause one error rather than a number of errors depending on the number of words in the string. This can be seen in 7.1.1 Input of a String.
3. From section 7.1.3 Entering the Phrase ".csv", a method of establishing whether a file exists or not could be developed to ensure the file entered actually exists. This would eliminate the possibility of the user entering the name of a file which does not exist.
4. Referring to section 7.3.1Using the Name of a Non-Existing Output File, a method to ensure the user entered only an output file could be implemented. This would ensure that no output timetables would be created inside the project folder. This could be implemented through if loops and a function to ensure the users file already exists.
5. From section 7.4 Assigning Activities to Slots Which Were Not Already Filled, a method could be implemented to ensure that if no spaces were available the project would exit the loop rather than remain inside it indefinitely. This would remove the incorrect operation of this aspect of the code. This would be done through if loops and break statements and could inform the user their week is full.
6. Section 7.5 If a Conflict Occurs the User Should be Notified, highlights that if a conflict does occur, a timetable will still be generated. This generated timetable would be incorrect so a method to not create an incorrect timetable could be implemented to increase the programs reliability. This could be done using if loops and break statements.
7. A method which could handle float inputs as the number of users would solve problem arising from section 7.6.6 Entering a Float. The program would simply ask the user to re-enter the number rather than setting the decimal point and the number which follows to the name of the first user. This could be done through the use of while loops which are present throughout the program.
8. A method to allocate group work between users which are in the same group could also be implemented. This was investigated and the potential method to do so would be to create another activity class which could be called "co-operate activities". It would be of a higher priority than time undefined activities meaning co-operate activities would be assigned time after time defined activities. The program could ask the user if they have any group work and what user they are in a group with or the reading in of the data could be altered to read in co-operating activities. The program will then try to assign the same slot of time in both users' timetables. This is a very complex operation and thus was only briefly looked into.

Implementing these functionalities would create a well-rounded code which would have a much higher level of functionality and increased robustness.

9 Conclusion

This project covered the creation of a program which could create a weekly timetable using a user defined input file containing their weekly commitments. It was established at the beginning that a graphical user interface would not be used but instead excel files in the format of comma separated values (CSV) would be used. Overall the project was a success as only a small number of improvements and added functionalities could be used to improve the program, which are highlighted in section 8 Further Work and Evaluation. The program was thoroughly tested to highlight any potential errors or bugs in the code. These are all present in section 7 Results and Discussion. Final versions of all files used can be found in the appendix along with the user guide.

10 Contribution

Both group members contributed equally to allow the program to be developed in parallel which decreased development time. Effort was applied both inside and outside of lab time in order to ensure project was completed to a high level. Communication played a key part in ensuring both members were kept up to date with all progress made.

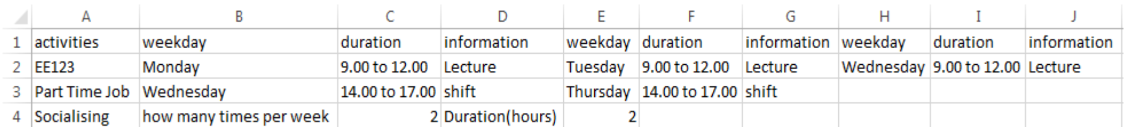
11 References

(http://www.cppblog.com/totti1006/archive/2014/04/28/95545.html, 2014) [1]

12 Appendix

12.1 Appendix (1) - User Guide

1. An input file must be created to use the program. The user must create an excel file which must be saved using the CSV format. The first row of the file must be the same as Figure 56.



	A	B	C	D	E	F	G	H	I	J
1	activities	weekday	duration	information	weekday	duration	information	weekday	duration	information
2	EE123	Monday	9.00 to 12.00	Lecture	Tuesday	9.00 to 12.00	Lecture	Wednesday	9.00 to 12.00	Lecture
3	Part Time Job	Wednesday	14.00 to 17.00	shift	Thursday	14.00 to 17.00	shift			
4	Socialising	how many times per week	2	Duration(hours)	2					

Figure 56 - Example Format of User Input File

   The user must have activities in the first column followed by three instances of “weekday”, “duration” and “information”. To enter time defined activities, the user must not have any spaces after their weekday and also have no spaces after the duration. To enter time undefined activities, the user must use “how many times per week” and “Duration(hours)” whilst having no spaces following any entries. The user may enter instances of activities in a horizontal manner.

2. An output file should be created in the CSV format also. There is no data required for this file.

3. The user may then run the program and enter the number of users that seek to create a timetable into the command window and press return. An input file similar to that of Figure 56 and an output file must be created for each user.

4. The first user will then enter their name into the command window and press return.

5. The first user will then drag their input file into the command window.

6. The first user will then drag their output file into the command window.

7. This will be repeated for each user requiring a timetable.

8. If the user wishes to alter an existing timetable they may manually alter it on excel or they can alter their input file and repeat the above process.

9. If any errors occur from the user input files, the use will be notified and instructed via the command window.

10. The user timetables will be attainable by opening their output file once the program has ended.

12.2 Appendix (2) – Main.cpp

```cpp
// Timetable generator project
// main.cpp
// Created by Jiayi Zhang and Arran Moffat on 06/03/2018.
//Last update implementing welcome message and readability lines
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <new>
#include "Undefined_time_activities.h"
#include "Defined_time_activities.h"
#include "Inputdata.h"
#include "otherfunctions.h"
#include "slot_in_functions.h"
using namespace std;
int main(int argc, const char * argv[])
{
    welcome_message();
    string name;
    int nn = ask_number_of_users();
    for (int i = 0; i < nn; i++)
{
//use a gap to increase reability between each user input
        cout << "\n/////////////////////////////\n";
        cout << "\nPlease enter user " << i + 1 << "'s name: ";
        cin >> name;
//create 18 time defined activities which will fill up slots in a week
        Time_defined_activities de_activity[18];
//create another 10 time undefined activities which will fill up free time in a week
        Time_undefined_activities un_activity[10];
//input all the data from the input file, and give value to the defined activities and unde
fined activities
//it is defined in the "Inputdata.h"
        defineallactivities(de_activity, un_activity);
//creat 7 days for a week, in a new class called "Day_of_week", in the file "assign_input_t
o_output"
        Day_of_week Monday[12], Tuesday[12], Wednesday[12], Thursday[12], Friday[12], Satur
day[12], Sunday[12];
//first set the time defined activities, in the file "other functions"
        set_each_day(de_activity, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, S
unday); //then to set other time undifined activities
        time_undefined_activities_slot_in(un_activity, Monday, Tuesday, Wednesday, Thursday
, Friday, Saturday, Sunday); //print them to the output file.
        Print_output_file(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
//inform each user their timetable is ready
        cout << name << "'s timetable is complete" << endl;
    }
    cout << "\n/////////////////////////////\n";
    cout << "\nThankyou for using this program.\n\n";
    system("PAUSE");
    return 0;
}
```

12.3 Appendix (3) – Inputdata.cpp

```
// Timetable generator project
// Inputdata.cpp
// Created by Jiayi Zhang on 12/03/2018.
// Last update implementing conflict detection system
#include "Inputdata.h"
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <new>
#include "Undefined_time_activities.h"
#include "Defined_time_activities.h"
using namespace std;

void gettime(string timestring, int time[])
{
    char char_time[15];
    strcpy_s(char_time, timestring.c_str()); //string copy (to here, from here)
    char buf[20];
    int i = 0;
    while (char_time[i] != '\0')
    {
        //if its a digit it will be put in buf which contains only numbers and spaces
        if (isdigit(char_time[i])) buf[i] = char_time[i];
        else buf[i] = ' ';
        ++i;
    }
    buf[i] = '\0';
    //%d = digits assigns them to each time
    sscanf_s(buf, "%d %d %d %d", & time[0], & time[1], & time[2], & time[3]);
}

void defineallactivities(Time_defined_activities de_activity[], Time_undefined_activities u
n_activity[])
{
    string myfile, mystring;
    //take in the users file
    cout << "Enter your input file: ";
    cin >> myfile;
    //ensure the user input file is of csv format
    size_t found;
    found = myfile.find(".csv");
    ifstream infile;
    //ask user to re-enter if file is not in csv format
    while (found == std::string::npos)
    {
        cout << "ERROR: The Input file is not in the format of csv, please re-enter: ";
        cin >> myfile;
        found = myfile.find(".csv");
    }
    infile.open(myfile.c_str());
   //print a message to the command window stating the file has been opened successfully
    cout << "\nInput file successfully open." << endl;

    string activity, day, information;
    string check, na, dua;

    int frequency, duration;
    int i = 0;
    int k = 0;
    int * p;
    //ignores the first line as it is not the users data
    getline(infile, check, '\n');
```

```cpp
//counts number of activities by calculating the number of filled cells from the second line to the last
    size_t n = count(check.begin(), check.end(), ',');
    //divides by 3 as 3 boxes correspndes to 1 activity
    n = n / 3;
    //while the file is open
    while (infile.good())
    {
//gets each activity using deliminators of a comma since the input file is of CSV format
        getline(infile, activity, ',');
        getline(infile, check, ',');
        //search for "times" in the cell containing "how many times per week
        size_t found;
        found = check.find("times");
        if (found != std::string::npos)
        {
//if it is a time undefined activity the un_activity array will begin to fill the activity name
            un_activity[i].set_undefined_activity_name(activity);
//if a comma is detected the next section to be filled is the number of times per week (frequency)
            getline(infile, check, ',');
//the ferequency is converted to an integer from ascii
            frequency = atoi(check.c_str()); //the frequency for this index of the array is then set
            un_activity[i].set_activity_times_per_week(frequency); //again uses commas as a delimeter to move onto the next "cell"
            getline(infile, na, ',');
            getline(infile, check, ','); //once the word duration has been found it takes in the next cell which contsins the numerical duration
            duration = atoi(check.c_str()); //converts again from ascii to an integer and assigns that value to the duration
            un_activity[i].set_activity_duration(duration); //has all the commas at end they are useless so looks for new line
            getline(infile, na, '\n'); //iterates to the next index in the array
            i = i + 1;
        }
        else
        {
       //if "how many times per week" is not detected it must be a time defined activity
            //thus check now = "day" rather than "how many times per week"
            day = check;
            for (int j = 0; j < n; j++)
            {
                        de_activity[k].set_defined_activity_name(activity);
                        de_activity[k].set_weekday(day);
                        getline(infile, dua, ',');
                        p = new int[4];
                        //function used to extract integer duration from string duration
                        gettime(dua, p);
                        de_activity[k].set_start_time_hour(p[0]);
                        de_activity[k].set_start_time_minute(p[1]);
                        de_activity[k].set_end_time_hour(p[2]);
                        de_activity[k].set_end_time_minute(p[3]);
                        //needs to reuse p every time
                        delete[] p;
                        if (j == 2)
                        {
                            //set activity information
                            getline(infile, information, '\n');
                            de_activity[k].set_information(information);
                            k = k + 1;
                            break;
                        }
                        else
                        {
                            getline(infile, information, ',');
                        }
                        de_activity[k].set_information(information);
                        getline(infile, day, ',');
                        k = k + 1;
                        //if empty, move to the next line
                        if (day == "")
                        {
                            getline(infile, na, '\n');
                            break;
                        }
                    }
                }
            }
        }
```

12.4 Appendix (4) – Inputdata.h

```cpp
// Timetable generator project
// Inputdata.h
// Created by Jiayi Zhang on 12/03/2018.
// Last update defining functions
#ifndef INPUTDATA_H_
#define INPUTDATA_H_
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <new>
#include "Undefined_time_activities.h"
#include "Defined_time_activities.h"
using namespace std;

void gettime(string timestring, int time[]);
void defineallactivities(Time_defined_activities de_activity[], Time_undefined_activities u
n_activity[]);
#endif /* INPUTDATA_H_ */
```

12.5 Appendix (5) – Defined_time_activities.cpp

```
// Timetable generator project
// Defined_time_activities.cpp
// Created by Arran Moffat on 06/03/2018.
// Last update defining data setters and getters

#include <iostream>
#include <string>
#include "Defined_time_activities.h"
using namespace std;

//setters
void Time_defined_activities::set_defined_activity_name(string activity)
{
    defined_activity_name = activity;
}
void Time_defined_activities::set_weekday(string day)
{
    weekday = day;
}
void Time_defined_activities::set_information(string inf)
{
    information = inf;
}
void Time_defined_activities::set_start_time_hour(int time_hour)
{
    start_time_hour = time_hour;
}
void Time_defined_activities::set_start_time_minute(int time_minute)
{
    start_time_minute = time_minute;
}
void Time_defined_activities::set_end_time_hour(int time_hour)
{
    end_time_hour = time_hour;
}
void Time_defined_activities::set_end_time_minute(int time_minute)
{
    end_time_minute = time_minute;
}

//getters
string Time_defined_activities::get_defined_activity_name()
{
    return defined_activity_name;
}
string Time_defined_activities::get_weekday()
{
    return weekday;
}
string Time_defined_activities::get_information()
{
    return information;
}
int Time_defined_activities::get_start_time_hour()
{
    return start_time_hour;
}
int Time_defined_activities::get_start_time_minute()
{
    return start_time_minute;
}
int Time_defined_activities::get_end_time_hour()
{
    return end_time_hour;
}
int Time_defined_activities::get_end_time_minute()
{
    return end_time_minute;
}
```

12.6 Appendix (6) – Defined_time_activities.h

```
// Timetable generator project
// Defined_time_activities.h
// Created by Arran Moffat on 06/03/2018.
// Last update defining data members

#ifndef DEFINED_TIME_ACTIVITIES_H_
#define DEFINED_TIME_ACTIVITIES_H_
#include <stdio.h>
#include <string>
using namespace std;

//format of csv data being read in:
//activity name, day, start time, end time
class Time_defined_activities
{
    private:
    string defined_activity_name, weekday, information;
    int start_time_hour, start_time_minute, end_time_hour, end_time_minute;

    public:

    //constructor
    Time_defined_activities() {}

    //setters
    void set_defined_activity_name(string);
    void set_weekday(string);
    void set_information(string);
    void set_start_time_hour(int);
    void set_start_time_minute(int);
    void set_end_time_hour(int);
    void set_end_time_minute(int);

    //getters
    string get_defined_activity_name();
    string get_weekday();
    string get_information();
    int get_start_time_hour();
    int get_start_time_minute();
    int get_end_time_hour();
    int get_end_time_minute();

    //destructor
    virtual~Time_defined_activities() {}
};

#endif /* DEFINED_TIME_ACTIVITIES_H_ */
```

12.7 Appendix (7) – Undefined_time_activities.cpp

```cpp
// Timetable generator project
// Undefined_time_activities.cpp
// Created by Jiayi Zhang and Arran Moffat on 12/03/2018.
// Defining setters and getters

#include <iostream>
#include <string>
#include "Undefined_time_activities.h"
using namespace std;

//setters
void Time_undefined_activities::set_undefined_activity_name(string activity)
{
    undefined_activity_name = activity;
}
void Time_undefined_activities::set_activity_duration(int duration)
{
    activity_duration = duration;
}
void Time_undefined_activities::set_activity_times_per_week(int times_week)
{
    times_per_week = times_week;
}

//getters

string Time_undefined_activities::get_undefined_activity_name()
{
    return undefined_activity_name;
}
int Time_undefined_activities::get_activity_duration()
{
    return activity_duration;
}
int Time_undefined_activities::get_activity_times_per_week()
{
    return times_per_week;
}
```

12.8 Appendix (8) – Undefined_time_activities.h

```cpp
// Timetable generator project
// Undefined_time_activities.h
// Created by Jiayi Zhang and Arran Moffat on 12/03/2018.
// Defining data members

#ifndef UNDEFINED_TIME_ACTIVITIES_H_
#define UNDEFINED_TIME_ACTIVITIES_H_
using namespace std;
#include <iostream>
#include <string>

//if the time is not defined format:
//activity name, number of times per week, duration of activity
class Time_undefined_activities
{
private:
    int activity_duration, times_per_week;
    string undefined_activity_name;
public:
    //constructor
    Time_undefined_activities() {}

    //setters
    void set_undefined_activity_name(string);
    void set_activity_duration(int);
    void set_activity_times_per_week(int);

    //getters
    string get_undefined_activity_name();
    int get_activity_duration();
    int get_activity_times_per_week();

    //destructor
    virtual~Time_undefined_activities() {}

};

#endif /* UNDEFINED_TIME_ACTIVITIES_H_ */
```

12.9 Appendix (9) – Assign_input_to_output.cpp

```cpp
// Timetable generator project
// assign_input_to_output.cpp
// Created by Arran Moffat on 18/03/2018.
// Last update implementing getters and setters

#include "assign_input_to_output.h"
#include "Defined_time_activities.h"
#include <string>
using namespace std;

//setters
void Day_of_week::set_activity_name(string n)
{
    name = n;
}
void Day_of_week::set_activity_info(string i)
{
    info = i;
}

//getters
string Day_of_week::get_activity_name()
{
    return name;
}
string Day_of_week::get_activity_info()
{
    return info;
}
```

12.10 Appendix (10) - Assign_input_to_output.h

```
// Timetable generator project
// assign_input_to_output.h
// Created by Arran Moffat on 18/03/2018.
// Last update defining data members

#include <iostream>
#include <string>
using namespace std;

#ifndef ASSIGN_INPUT_TO_OUTPUT_H_
# define ASSIGN_INPUT_TO_OUTPUT_H_

//will have 7 arrays of size 12
class Day_of_week {
private:
    string name, info;
public:
    //constructor
    Day_of_week() {}

    //setters
    void set_activity_name(string);
    void set_activity_info(string);
    //getters
    string get_activity_name();
    string get_activity_info();
    virtual~Day_of_week() {}
};

#endif /*ASSIGN_INPUT_TO_OUTPUT_H_*/
```

12.11 Appendix (11) – Otherfunctions.cpp

```cpp
// Timetable generator project
// otherfunctions.cpp
// Created by Jiayi Zhang and Arran Moffat on 24/03/2018.
// Last update ensuring user enters csv file

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include "otherfunctions.h"
#include "Defined_time_activities.h"
#include "stdlib.h"
using namespace std;

void welcome_message()
{
    cout << "Welcome to the timetable generator project!";
    cout << "\n\nUse only the template input file to enter your commitments!";
    cout << "\n\nIf you enter an incorrect file please retry with your input template file!";
    cout << "\n\nYour output file can be any excel file.";
    cout << "\n\nIf you wish to change an existing file you can either edit your output file on \nexcel or change your input file and rerun the programme.";
    cout << "\n\nYou will be notified if any of your commitments cause a conflict, in which case please alter your schedule!";
    cout << "\n\nFinally, the programme will work for any number of users.\n\n";
}

int ask_number_of_users()
{
    //ask how many users require a timetable
    int nn;
    cout << "How many users will need a timetable of a week?: ";
    cin >> nn;
    //if the entry is not an integer value, prompt for re-entry until integer entered
    while (cin.fail())
    {
        cout << "Error, please enter an integer number!: ";
        cin.clear();
        cin.ignore(256, '\n');
        cin >> nn;
    }
    return nn;
}

void Print_output_file(Day_of_week Monday[], Day_of_week Tuesday[], Day_of_week Wednesday[], Day_of_week Thursday[], Day_of_week Friday[], Day_of_week Saturday[], Day_of_week Sunday[])
{
    string name, Output_File_Name;
    //take in user output file
    cout << "Enter the output file name: ";
    cin >> Output_File_Name;
    //make sure user output file is of csv format
    size_t found;
    found = Output_File_Name.find(".csv");
    ofstream Output_File;
    //ask user to re-enter if file is not csv format
    while (found == std::string::npos)
    {
        cout << "ERROR: The output file is not in the format of csv, please re-enter: ";
        cin >> Output_File_Name;
        found = Output_File_Name.find(".csv");
    }
    //inform user output file could be opened
    cout << "\nOutput file successfully open." << endl;
    Output_File.open(Output_File_Name.c_str());
    //clear output file before writing to it
    Output_File.clear();
    //print empty cell at top left
    Output_File << ",";
    //print top line of times
    for (int i = 9; i < 22; i++) {
        Output_File << i << ":00" << " - " << i + 1 << ":00" << ",";
    }
    Output_File << endl;

////////////// PRINT EACH DAYS SCHEDULE AND AN EMPTY ROW TO INCREASE READIBILITY BENEATH
    int wday = 1;
    Day_of_week * pointer = nullptr;
    while (wday <= 7)
    {
        {
            if (wday == 1)
            {
                pointer = Monday;
                Output_File << "Monday" << ",";
            }
            else if (wday == 2)
            {
                pointer = Tuesday;
                Output_File << "Tuesday" << ",";
            }
            else if (wday == 3)
            {
                pointer = Wednesday;
                Output_File << "Wednesday" << ",";
            }
            else if (wday == 4) {
                pointer = Thursday;
                Output_File << "Thursday" << ",";
            }
            else if (wday == 5)
            {
                 pointer = Friday;
                 Output_File << "Friday" << ",";
             }
             else if (wday == 6)
             {
                 pointer = Saturday;
                 Output_File << "Saturday" << ",";
             }
             else if (wday == 7)
             {
                 pointer = Sunday;
                 Output_File << "Sunday" << ",";
             }
        }
        //print the name of the activity
        for (int i = 0; i < 12; i++)
        {
            Output_File << pointer[i].get_activity_name() << ",";
        }
        Output_File << endl;
        //print the information of the activity
        Output_File << "information" << ",";
        for (int i = 0; i < 12; i++)
        {
            Output_File << pointer[i].get_activity_info() << ",";
        }
        Output_File << endl; //print out an empty row
        for (int i = 0; i < 12; i++)
        {
            Output_File << "" << ",";
        }
        Output_File << endl;
        wday = wday + 1;
        }
        //close the output file
        Output_File.close();
}
```

12.12 Appendix (12) – Otherfunctions.h

```
// Timetable generator project
// otherfunctions.h
// Created by Jiayi Zhang and Arran Moffat on 24/03/2018.
// Last update defining welcome_message function
#include <iostream>
#include <string>
#include "Defined_time_activities.h"
#include "assign_input_to_output.h"
using namespace std;

#ifndef CREATE_TEMPLATE_TIMETABLE_H_# define CREATE_TEMPLATE_TIMETABLE_H_

void welcome_message();

int ask_number_of_users();

void Print_output_file(Day_of_week Monday[], Day_of_week Tuesday[], Day_of_week Wednesday[], Day_of_week Thursday[], Day_of_week Friday[], Day_of_week Saturday[], Day_of_week Sunday[]);

#endif /*CREATE_TEMPLATE_TIMETABLE_H_*/
```

12.13 Appendix (13) – Slot_in_function.cpp

```cpp
// Timetable generator project
// slot_in_function.cpp
// Created by Jiayi Zhang on 27/03/2018.
// Last update using pointers to assign each day

#include "slot_in_functions.h"
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <new>
#include "Undefined_time_activities.h"
#include "Defined_time_activities.h"
#include "otherfunctions.h
using namespace std;

/// SET EACH DAY WITH THE TIME DEFINED ACTIVITIES
void set_each_day(Time_defined_activities de_activity[], Day_of_week Monday[], Day_of_week Tuesday[], Day_of_week Wednesday[], Day_of_week Thursday[], Day_of_week Friday[], Day_of_week Saturday[], Day_of_week Sunday[])
{
    int a;
    int b;
    Day_of_week * pointer = nullptr;
    //use pointers to assign each day
    //searches for the day "Monday" and "monday" to allow the user to use capital letters
    //or lowercase
    for (int i = 0; i < 18; i++)
    {
         if (de_activity[i].get_defined_activity_name() == "")
             break;
        else if (de_activity[i].get_weekday() == "Monday" || de_activity[i].get_weekday() == "monday") pointer = Monday;
        else if (de_activity[i].get_weekday() == "Tuesday" || de_activity[i].get_weekday() == "tuesday") pointer = Tuesday;
        else if (de_activity[i].get_weekday() == "Wednesday" || de_activity[i].get_weekday() == "wednesday") pointer = Wednesday;
        else if (de_activity[i].get_weekday() == "Thursday" || de_activity[i].get_weekday() == "thursday") pointer = Thursday;
        else if (de_activity[i].get_weekday() == "Friday" || de_activity[i].get_weekday() == "friday") pointer = Friday;
        else if (de_activity[i].get_weekday() == "Saturday" || de_activity[i].get_weekday() == "saturday") pointer = Saturday;
        else if (de_activity[i].get_weekday() == "Sunday" || de_activity[i].get_weekday() == "sunday") pointer = Sunday;

        a = de_activity[i].get_start_time_hour();
        b = de_activity[i].get_end_time_hour();

        //uses end time - start time to calculate what index of the array
        //to place the activity name and info
        int c = b - a;

        //if the activity starts at 10 am and finishes at 10.30 am,
        //the array must still iterate to assign that activity
        //it will be assigned for the entire hour as the timetable is
        //divided into hour sections
        if (c == 0)
        {
            c = c + 1;
        }
        for (int ii = 0; ii < c; ii++)
        {
            //if a conflict occurs inform the user
            if (pointer[a - 9 + ii].get_activity_name() != "")
            {
                cout << "\nERROR: Conflict occurs on " << de_activity[i].get_weekday() << " at " << de_activity[i].get_start_time_hour() << ".00" << endl;
                cout << "Conflict is between: " << pointer[a - 9 + ii].get_activity_name() << " - " << pointer[a - 9 + ii].get_activity_info();
                cout << " and " << de_activity[i].get_defined_activity_name() << " - " << de_activity[i].get_information() << endl;
                cout << "Please revise your schedule and re-run the program.\n\n";
                break;
            }
            pointer[a - 9 + ii].set_activity_name(de_activity[i].get_defined_activity_name());
            pointer[a - 9 + ii].set_activity_info(de_activity[i].get_information());
        }
    }
}

void time_undefined_activities_slot_in(Time_undefined_activities un_activity[], Day_of_week Monday[], Day_of_week Tuesday[], Day_of_week Wednesday[], Day_of_week Thursday[], Day_of_week Friday[], Day_of_week Saturday[], Day_of_week Sunday[])
{
    int aa = 0;
    int m = 0;
    Day_of_week * pointer = nullptr;
    while (un_activity[m].get_undefined_activity_name() != "")
    {
        int wday = 1;
        int hours = 0;
        int ff = 1;
        int f = un_activity[m].get_activity_times_per_week();
        int a = un_activity[m].get_activity_duration(); //while users to create pointer
        while (wday <= 7)
        {
            {
                if (wday == 1) pointer = Monday;
                else if (wday == 2) pointer = Tuesday;
                else if (wday == 3) pointer = Wednesday;
                else if (wday == 4) pointer = Thursday;
                else if (wday == 5) pointer = Friday;
                else if (wday == 6) pointer = Saturday;
                else if (wday == 7) pointer = Sunday;
            }
            //while loop used to search for free times to assign time undefined activity
            while (hours < 12)
            {
                {
                    for (aa = 0; aa < a; aa++)
                    {
                        if (pointer[hours].get_activity_name() != "")
                        {
                                   hours = hours + 1;
                                   break;
                        }
                        else hours = hours + 1;
                    }
                    if (aa == a)
                    {
                       for (int i = 0; i < aa; i++)
                       {
                                   pointer[hours - 1 - i].set_activity_name(un_activity[m].get_undefined_activity_name());
                       }
                       wday = wday + 1;
                       hours = 0;
```

```
                        ff = ff + 1;
                        break;
                   }
                   else break;
               }
          }
          if (ff > f)
          {
             m = m + 1;
             break;
          }
        }
       }
       }
```

12.14 Appendix (13) – Slot_in_functions.h

```
// Timetable generator project
// slot_in_functions.h
// Created by Jiayi Zhang on 27/03/2018.
// Last update defining functions

#ifndef SLOT_IN_FUNCTIONS_H_
#define SLOT_IN_FUNCTIONS_H_
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <new>
#include "Undefined_time_activities.h"
#include "Defined_time_activities.h"
#include "otherfunctions.h"
using namespace std;

void set_each_day(Time_defined_activities de_activity[], Day_of_week Monday[], Day_of_week Tuesday[], Day_of_week Wednesday[], Day_of_week Thursday[], Day_of_week Friday[], Day_of_week Saturday[], Day_of_week Sunday[]);

void time_undefined_activities_slot_in(Time_undefined_activities un_activity[], Day_of_week Monday[], Day_of_week Tuesday[], Day_of_week Wednesday[], Day_of_week Thursday[], Day_of_week Friday[], Day_of_week Saturday[], Day_of_week Sunday[]);

#endif /* SLOT_IN_FUNCTIONS_H_ */
```