

# **PROJECT REPORT**

## **CS101 Project 2015**

**Time-Table Scheduling and Optimisation**

**Group CUSE**

**R.Basuhi, 140010037**

**Yashaswini K Murthy, 140010054**

**Shachi Shailesh Deshpande, 140110047**

## Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>3</b>
1.1	Definitions, Acronyms and Abbreviations .....	3
1.2	References .....	3
<b>2</b>	<b>Overall Description .....</b>	<b>3</b>
<b>3</b>	<b>Details.....</b>	<b>5</b>
3.1	Functionality .....	5
3.2	Supportability .....	5
3.3	Design Constraints .....	5
3.4	On-line User Documentation and Help System Requirements .....	6
3.5	Interfaces .....	6
3.5.1	User Interfaces .....	6
3.5.2	Hardware Interfaces .....	6
3.5.3	Software Interfaces.....	6
3.5.4	Communications Interfaces .....	6
<b>4</b>	<b>Quality control .....</b>	<b>6</b>
<b>5</b>	<b>Risk management.....</b>	<b>7</b>
<b>6</b>	<b>Weekly Schedule.....</b>	<b>7</b>
<b>7</b>	<b>Implementation.....</b>	<b>8</b>
<b>8</b>	<b>Challenges Faced.....</b>	<b>17</b>
<b>9</b>	<b>Instructions To Run Our Code.....</b>	<b>17</b>
<b>10</b>	<b>Future Prospects.....</b>	<b>17</b>

# 1 Introduction

The following document describes the functional and non-functional requirements for the Time-Table Scheduling and Optimisation program to be submitted as a Software Project as part of our CS101 course-Spring Semester 2015. The features expressed in this Software Requirements Specification document are intended to be implemented to the maximum possible extent in our project. The system will be developed in such a way to provide easy addition of enhanced features, which may be desired in subsequent versions, if any.

## 1.1 Definitions, Acronyms and Abbreviations

- **Time-Table Scheduling:** In the context of our project, Time-Table scheduling is a constraint based problem wherein one inputs the constraints pertaining to a universities' course subjects, professor's availability and time-slots.
- **Soft Constraints:** Constraints that are negotiable to a certain extent for achieving the optimal solution.
- **Hard Constraints:** If the problem mandates that the constraints be satisfied, they are called hard constraints.
- **Perturbation:** In our context, the perturbation method means we take a feasible solution and make random changes (called perturbations) and try to find the most optimum timetable among the ones generated.
- **BackTracking:** Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons each partial candidate, *c*, ("backtracks") as soon as it determines that *c* cannot possibly be completed to a valid solution. It usually follows a recursive approach.
- **Randomized Subsets:** In this method, we create random subsets of all the time-tables possible (the number is huge!) and find the pseudo-optimal solution
- **Sequential Allocation:** This is based on some pattern, maybe on a First-Come-First-Serve basis.
- **TA:** Teaching Assistants. In our project, they teach tutorials and take lab classes.

## 1.2 References

[http://en.wikipedia.org/wiki/Constrained\\_optimization](http://en.wikipedia.org/wiki/Constrained_optimization)

# 2 Overall Description

## Product Perspective

This product intends to automate the procedure of Time-Table Scheduling and thus create an optimal version of the Time-Table subject to a few constraints. The professors and TAs get to input their name, subject of interest, preferred time-slots and other details. Then the program generates an optimal timetable. As envisioned so far, it is self-contained.

## Product Functions

From a bird's eye view, the program aims at the primary function of generating Time-Tables for a class subdivided into batches. From this optimal Time-Table, two sets, one for the Professors and one for each batch is generated.

## User Characteristics

The users are Professors and TAs who provide their preferences and availability and in turn, our program generates a Time-Table for them and each batch of students. In an extrapolated sense, even though students don't actively participate in the input process, they are also users who indirectly determine certain constraints.

## Constraints

- Time allotted for the completion and submission of the project is 4 weeks. Due to this we are trying to demonstrate a simplistic version.
- The language of the program is supposed to be C++ (i.e. at-least eighty percent of the coding must be done in C++) and the development environment will be Code::Blocks IDE.

## Assumptions and Dependencies

(Some of them may be upgraded to add more flexibility to the program if time permits.)

**A1:** Each professor teaches exactly one subject and exactly 2 batches.

**A2:** The subject allocated to the professor is on a First-Come-First-Serve basis.

**A3:** A time-table satisfying all the hard-constraints can be generated, but in order to find the optimal one, it may forego some soft-constraints.

**D1:** The subjects, the difficulty level (namely, the number of credits assigned), the time slots, the number of hours each subject requires, the classrooms, the number of working days, is all predetermined by us.

**D2:** Professor's preference is a soft-constraint, liable to be not met in the optimal time-table.

## Requirements Subsets

A particular subject must have all its lectures (and tutorials and labs, if applicable) assigned as per the course policy.

## 3 Details

### 3.1 Functionality

Our program primarily carries out the function of Time-Table generating and that, in layman's terms, includes:

1. Input: Professor's/TA's Input- We take console input from them.

(Since taking inputs for as many as 10 professors and as many TAs is time-consuming, we also have some default scenarios set up, just for testing purposes)

2. Time-Table Generation: Since the whole program revolves around this, we have several sub-functions dedicated for this purpose. These subfunctions are heavily based upon the mathematical algorithm that we are using.
3. Output: The optimal Time-Table generated is displayed to the user.

#### Modularisation

Also, this project has various classes, for *Professors*, for Lab and Tutorial *TAs* who have been inherited from Professors, for the *Time-Table* itself, which has *Cells* for slots. Each should be modularized accordingly along with the global functions that use them.

Owners of described modules:

Input/Output and File I/O: R.Basuhi and Yashaswini K. Murthy

Time-table Generating and Optimisation module: Shachi Shailesh Deshpande

### 3.2 Supportability

The computer on which the program is to be run must have software supportability for C++ programs. We have used simple-Codeblocks with the GNU GCC Compiler. No toolchain, packages, hardware requirements are needed.

We will use GNU Coding Standards, and the naming convention is such that the variables and corresponding entities are self-evident.

### 3.3 Design Constraints

The Time-Table has to adhere to certain pre-described standards like, classes only happen on weekdays. Each time-slot has a fixed time-interval and each course has it's one requirements in terms of number of hours. The courses are also pre-determined and have their own respective course policies that govern their number and duration of classes.

### 3.4 On-line User Documentation and Help System Requirements

In case of queries or reference to a user-manual, check out-

<https://docs.google.com/document/d/1qB2m7P0fkZFfyLMXEUUZ-sOWwLWe7YEvqB0bVdT8CM/edit?usp=sharing>

### 3.5 Interfaces

#### 3.5.1 User Interfaces

Console Input and Output.

#### 3.5.2 Hardware Interfaces

No hardware interface required.

#### 3.5.3 Software Interfaces

No software interfaces are required.

#### 3.5.4 Communications Interfaces

No communication interfaces required.

## 4 Quality Control

### 4.1 Test Data

1. If the teaching requirements of a particular subject are fulfilled, then, the next Professor who demands that subject will be intimated to change his subject or exit the form. Input through these forms will not cease until all the courses have a professors and TAs.
2. Slot clashes have to be avoided at all costs. If, say, all the professors give high preference for a particular day, maybe Monday, then the optimisation will be very low and perturbations won't be effective in generating such a time-table. Many will have to compromise on their preferences.
3. If there is a preference clash between TAs and Professors, Professors are given more weightage.

Testing Methodology:

Default cases will be generated by us to check and remove logical errors, if any. The default cases will be selected such that they cover both generic as well as extreme scenarios

List of test-cases:

- a) All professors/TAs want the same slots. In this case optimisation beyond a certain limit is not possible. An objective function (a measure of level of optimisation to be used throughout the program) will thus indicate the extent of optimisation thus done.

- b) Half the professor's want slots that are ranked low in the other half's preference. In this scenario, the optimisation should be self-evident.
- c) A default test case, where pseudo-random allocation of preferences happens, the objective function must also vary accordingly.
- d) Professor's input is inconsistent, does not adhere to the input standards clearly mentioned in the input portal, then he would be promptly asked to enter again. Also if vacancies for a particular subject has been filled (it is on first-come-first-serve basis) then, he can either choose a different subject or exit.
- e) As for generic case, any set of input must be optimised under the value of the objective function.

In the project development, the easy module includes generating a simple, unoptimised time-table as a base to work upon while the most difficult module to work on is perturbing the systems while keeping track of an objective function. Taking Input/Displaying Output can be listed of moderate difficulty though very crucial to link with the other modules.

## 5 Risk Management

- 1) The heart of the program is a mathematical algorithm that is used to generate an optimal solution to the constraint-based problem, i.e. Time-Table Scheduling. While an algorithm to find the absolute optimum is immensely complex and in some cases, not achievable, we have made an order of possible algorithms that are feasible, reliable and possible to implement in the limited time available to us.
  - Using Perturbation
  - Backtracking
  - Optimum under randomized subsets.
  - Sequential allocation
- 2) If time eludes the possibility of generating a full-fledged time-table generator, we might reduce the number of variables and increase the number of pre-determined constants.
- 3) They may rather give us relatively optimal solutions under reduced number of constraints. We will try to stick to the former ones, but worst case scenario, we might have to settle for a simple Time-Table generator.

## 6 Weekly Schedule

**Week 1:** Researching upon a suitable algorithm.

**Week 2:** Listing classes and functions that might be used to implement the said algorithm.  
Submission of SRS.

**Week 3:** Beginning to Code and Managing I/O

**Week 4:** Debugging the Code written for Prototype.

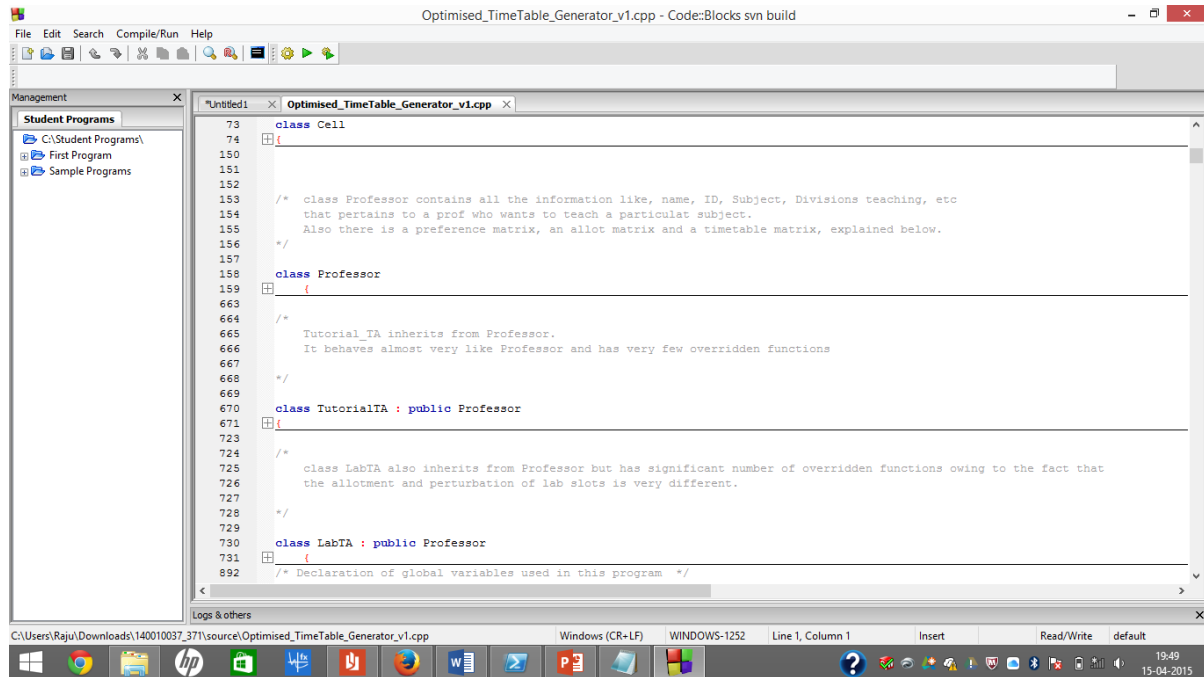
**Week 5:** Prototype Testing and Submission

**Week 6:** Improvements on the current prototype in the limited time available.

**Week 7:** Class demonstration (along with video tutorial and PowerPoint presentation) of the project and Final Submission.

## 7 Implementation (and features):

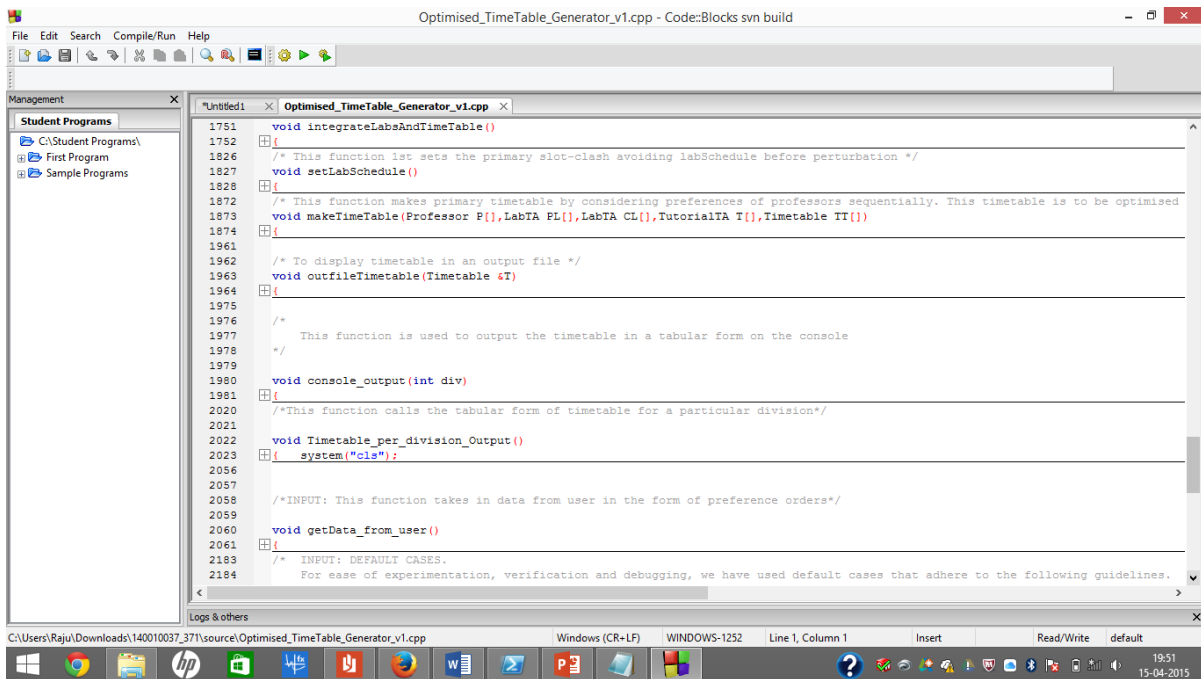
The entire code has been divided into a number of classes, for various entities of our program such as Professors, TAs, etc :



```
73 class Cell
74 {
150
151
152
153 /* class Professor contains all the information like, name, ID, Subject, Divisions teaching, etc
154 that pertains to a prof who wants to teach a particular subject.
155 Also there is a preference matrix, an allot matrix and a timetable matrix, explained below.
156 */
157
158 class Professor
159 {
663
664 /*
665 Tutorial_TA inherits from Professor.
666 It behaves almost very like Professor and has very few overridden functions
667 */
668
669 class TutorialTA : public Professor
670 {
723
724 /*
725 class LabTA also inherits from Professor but has significant number of overridden functions owing to the fact that
726 the allotment and perturbation of lab slots is very different.
727 */
728
729 class LabTA : public Professor
730 {
731
892 /* Declaration of global variables used in this program */
```

And a couple of global functions such as:



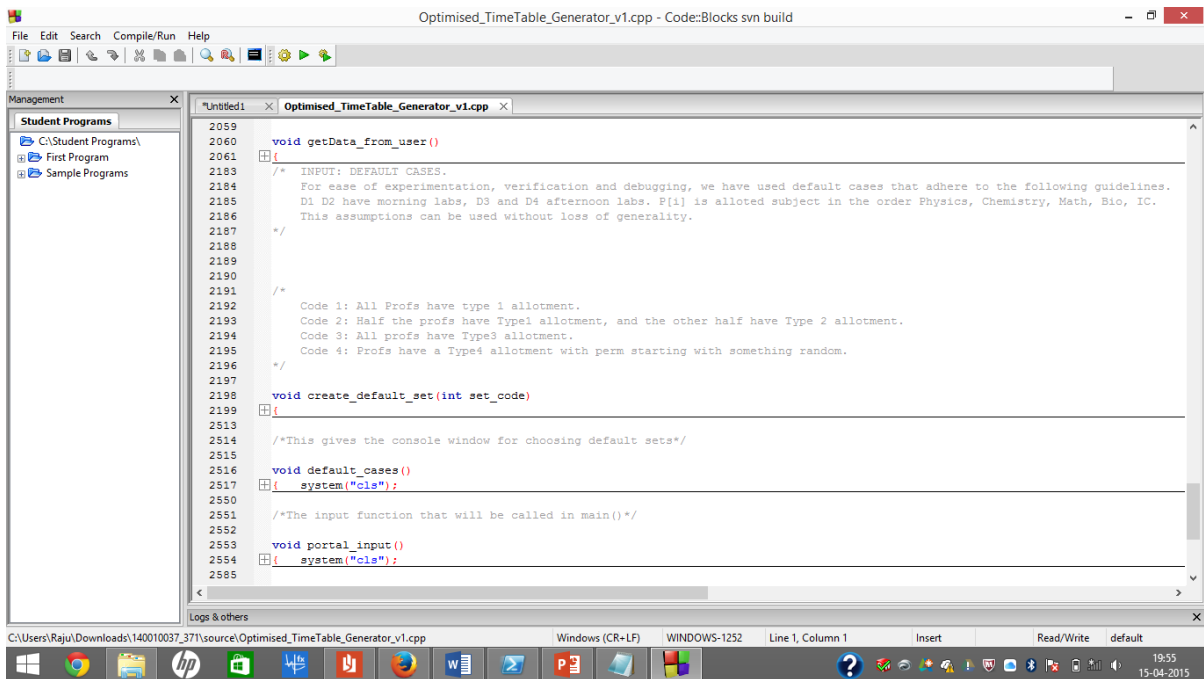


```
1751 void integrateLabsAndTimeTable()
1752 {
1826 /* This function 1st sets the primary slot-clash avoiding labSchedule before perturbation */
1827 void setLabSchedule()
1828 {
1872 /* This function makes primary timetable by considering preferences of professors sequentially. This timetable is to be optimised
1873 void makeTimeTable(Professor P[], LabTA FL[], LabTA CL[], TutorialTA T[], Timetable TT[])
1874 {
1961
1962 /* To display timetable in an output file */
1963 void outfileTimetable(Timetable &T)
1964 {
1975
1976 /* This function is used to output the timetable in a tabular form on the console
1977 */
1978
1979
1980 void console_output(int div)
1981 {
2020 /*This function calls the tabular form of timetable for a particular division*/
2021
2022 void Timetable_per_division_Output()
2023 {
2023     system("cls");
2056
2057
2058 /*INPUT: This function takes in data from user in the form of preference orders*/
2059
2060 void getData_from_user()
2061 {
2183 /* INPUT: DEFAULT CASES.
2184 For ease of experimentation, verification and debugging, we have used default cases that adhere to the following guidelines.
```

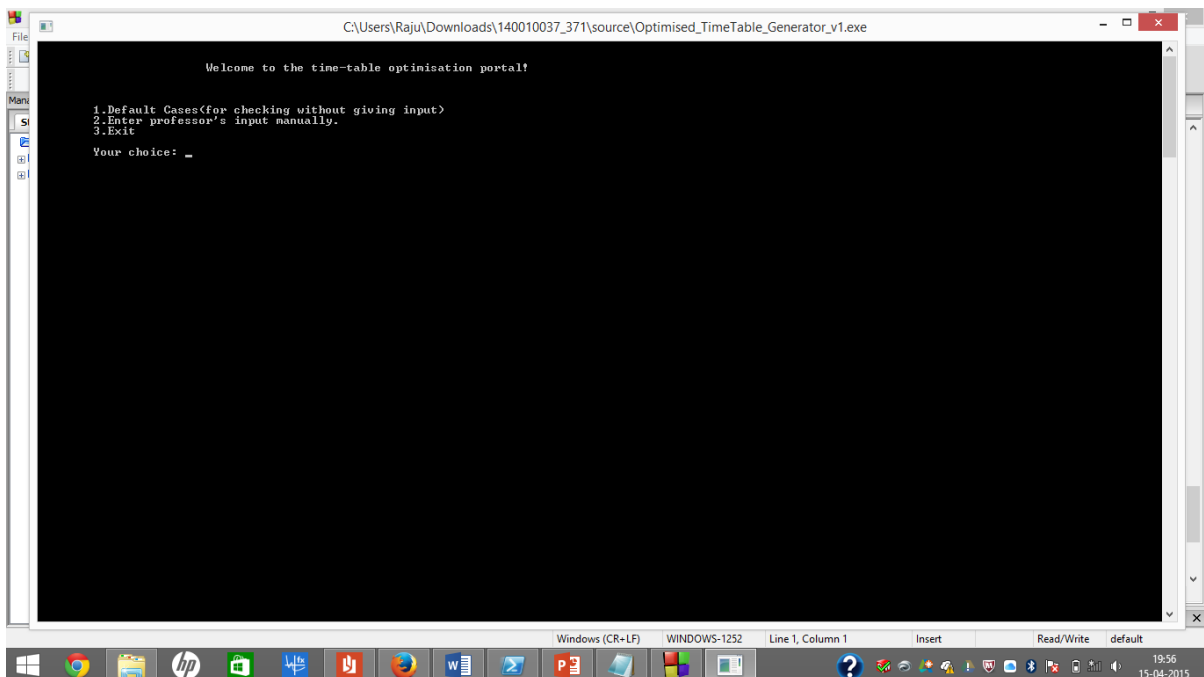
As we can see, some of classes which are common to other classes have inherited the superior class.

The input basically occurs through console where the user has the opportunity to either choose to enter all the preferences of the teachers manually or gets to choose the option which assigns the preferences of the teachers with the most likely set of values (default values which also serve as test cases).

The functions concerning input are shown below:



And the input on the console is of this manner:



On entering option 1, we get to choose the the default case we want.

Optimised\_TimeTable\_Generator\_v1.cpp - Code::Blocks svn build  
C:\Users\Raju\Downloads\140010037\_371\source\Optimised\_TimeTable\_Generator\_v1.exe

```
Default Sets <for easy review>

The default cases are as follows:

1.All the professors have a descending order of preference, with Monday being the most preferred and Friday being the least, all of them preferring morning slots
2.First Five professors will have their preference order as in case 1 and the next five will have ascending order of preference without loss of generality, that is Monday being the least preferred and Friday being the most, all of them preferring the afternoon slots
3.All the professors have a descending order of preference, with Monday being the most preferred and Friday being the least with then preferring alternate slots on consecutive days
4.All the professors have a descending order of preference starting from a random day, that is starting from any day from Monday to Friday, all preferring morning slots.

Your choice:2
```

Perhaps, we choose to go with option 2. The interface is of the following type:

Optimised\_TimeTable\_Generator\_v1.cpp - Code::Blocks svn build  
C:\Users\Raju\Downloads\140010037\_371\source\Optimised\_TimeTable\_Generator\_v1.exe

```
Welcome. This is the data input portal for Professors.

Professor # 1
Enter your choice of subject from the following:
1. PH107
2. CH107
3. MH106
4. IC102
5. BB101
Enter the serial number to choose a particular subject 2

Enter your name(only first name):Sunder

Now you will be giving the preference order of you available days. If Monday is least preferred assign it 1, and if most preferred assign it 5. And so on...Also each day must have a distinct preference value. And thus 1-5 numbers must be covered.

Day: Monday
Enter Preference for Monday 5
Enter your slot preference:
1. Morning
2. Afternoon
3. Both
Your choice:2

Day: Tuesday
Enter Preference for Tuesday 3
Enter your slot preference:
1. Morning
2. Afternoon
3. Both
Your choice:1

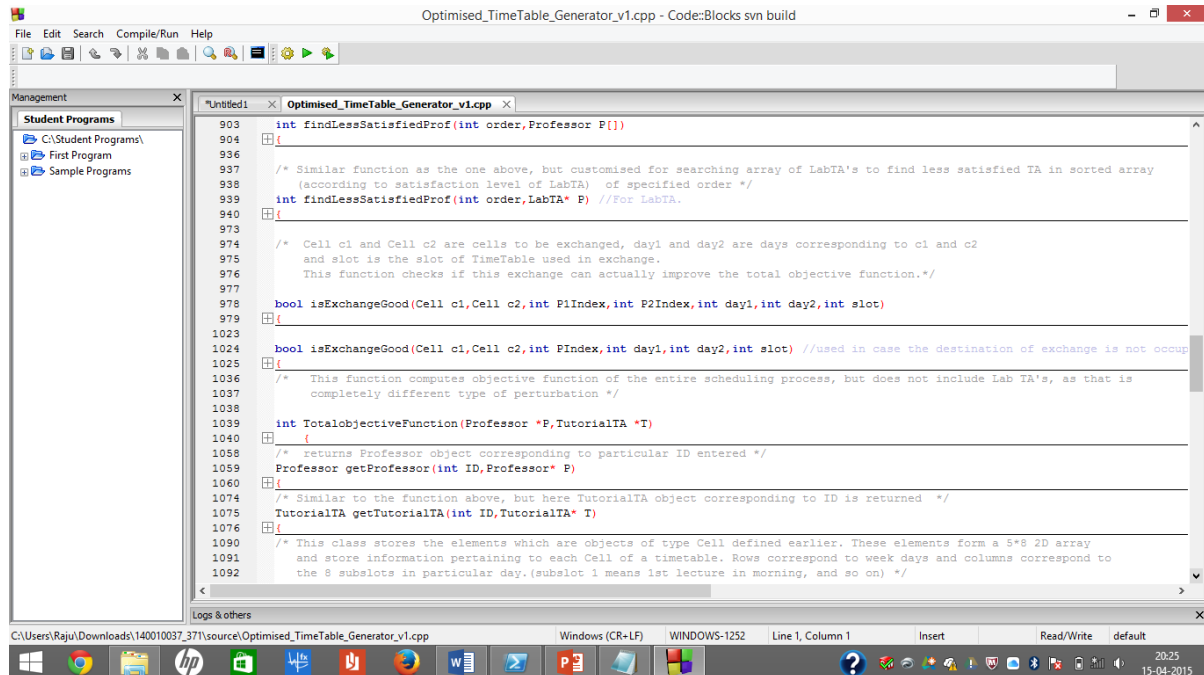
Day: Wednesday
Enter Preference for Wednesday 4
Enter your slot preference:
1. Morning
2. Afternoon
3. Both
Your choice:_
```

The same will be followed for all the professors and teaching assistants.

The above basically deals with input module of our code.

The algorithm module of the code is implemented in this manner:

Basically our algorithm is on perturbations which has been implemented through various global functions such as:



```
Optimised_TimeTable_Generator_v1.cpp - Code::Blocks svn build

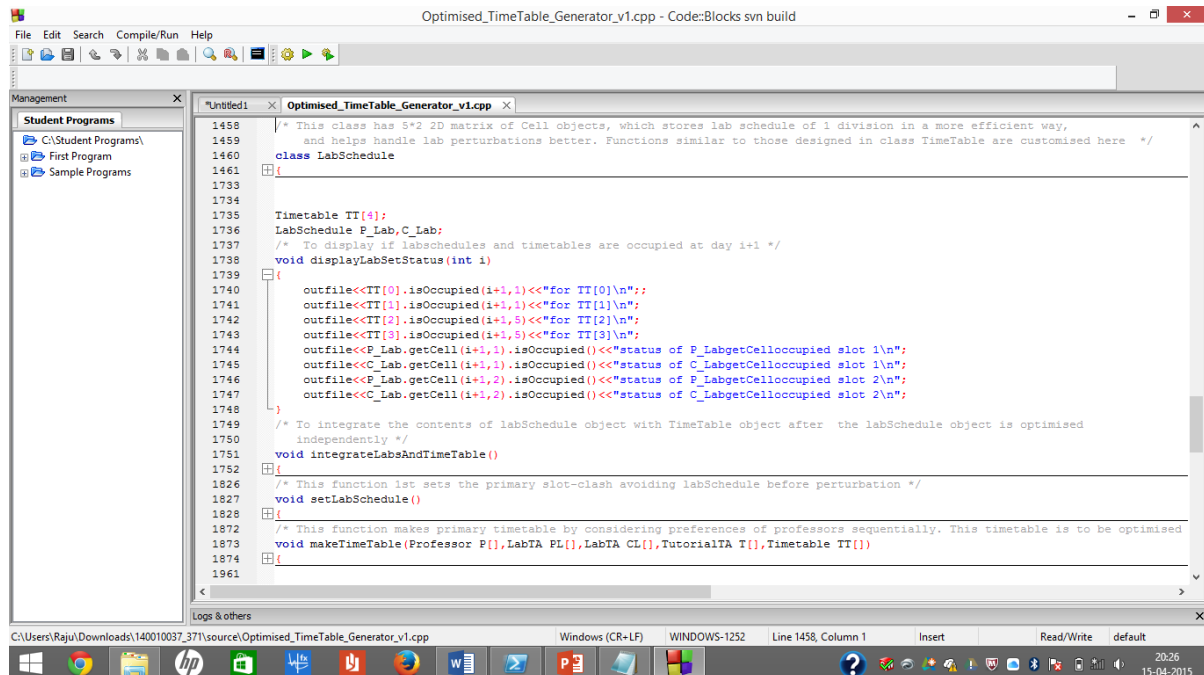
File Edit Search Compile/Run Help

Management
Student Programs
  C:\Student Programs\
  First Program
  Sample Programs

903 int findLessSatisfiedProf(int order, Professor P[])
904 {
905     /* Similar function as the one above, but customised for searching array of LabTA's to find less satisfied TA in sorted array
906     (according to satisfaction level of LabTA) of specified order */
907     int findLessSatisfiedProf(int order, LabTA* P) //for LabTA.
908     {
909         /* Cell c1 and Cell c2 are cells to be exchanged, day1 and day2 are days corresponding to c1 and c2
910         and slot is the slot of TimeTable used in exchange.
911         This function checks if this exchange can actually improve the total objective function.*/
912         bool isExchangeGood(Cell c1, Cell c2, int P1Index, int P2Index, int day1, int day2, int slot)
913         {
914             bool isExchangeGood(Cell c1, Cell c2, int PIndex, int day1, int day2, int slot) //used in case the destination of exchange is not occup
915             {
916                 /* This function computes objective function of the entire scheduling process, but does not include Lab TA's, as that is
917                 completely different type of perturbation */
918                 int TotalObjectiveFunction(Professor *P, TutorialTA *T)
919                 {
920                     /* returns Professor object corresponding to particular ID entered */
921                     Professor getProfessor(int ID, Professor* P)
922                     {
923                         /* Similar to the function above, but here TutorialTA object corresponding to ID is returned */
924                         TutorialTA getTutorialTA(int ID, TutorialTA* T)
925                         {
926                             /* This class stores the elements which are objects of type Cell defined earlier. These elements form a 5*8 2D array
927                             and store information pertaining to each Cell of a timetable. Rows correspond to week days and columns correspond to
928                             the 8 subslots in particular day. (subslot 1 means 1st lecture in morning, and so on) */
929                             ...
930                         }
931                     }
932                 }
933             }
934         }
935     }
936 }

C:\Users\Raju\Downloads\140010037_371\source\Optimised_TimeTable_Generator_v1.cpp Windows (CR-LF) WINDOWS-1252 Line 1, Column 1 Insert Read/Write default 20:25 15-04-2015
```

And...



```
Optimised_TimeTable_Generator_v1.cpp - Code::Blocks svn build

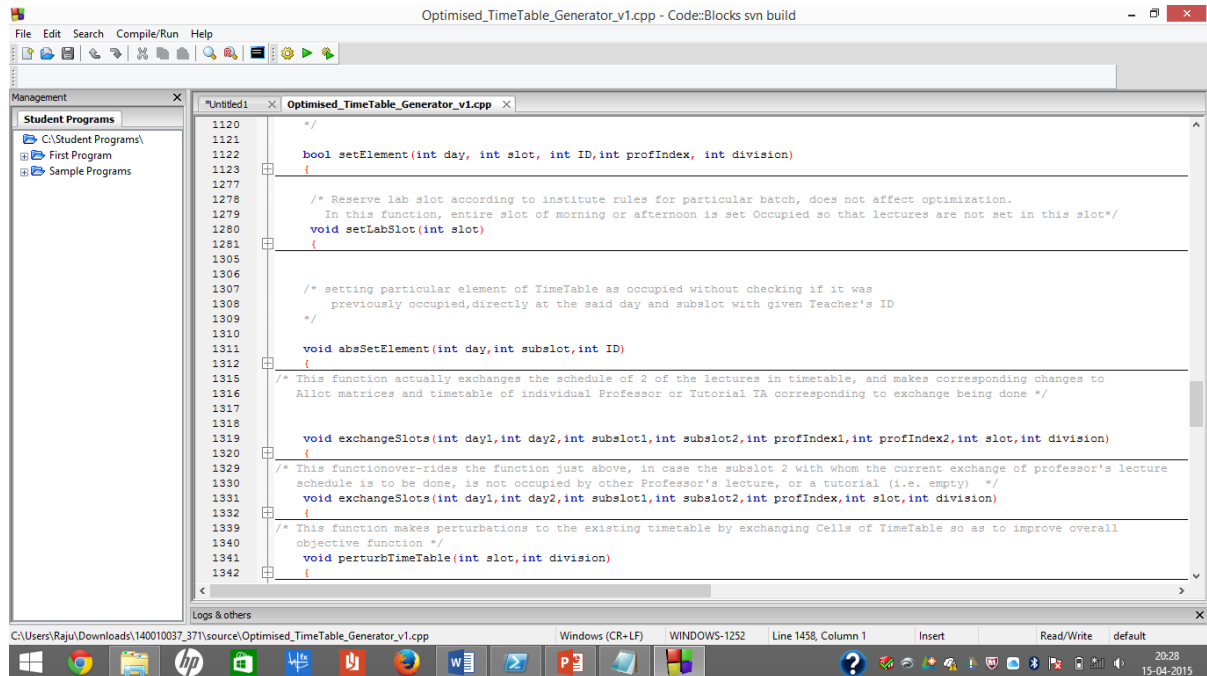
File Edit Search Compile/Run Help

Management
Student Programs
  C:\Student Programs\
  First Program
  Sample Programs

1458 /* This class has 5*2 2D matrix of Cell objects, which stores lab schedule of 1 division in a more efficient way,
1459 and helps handle lab perturbations better. Functions similar to those designed in class TimeTable are customised here */
1460 class LabSchedule
1461 {
1462     Timetable TT[4];
1463     LabSchedule P_Lab, C_Lab;
1464     /* To display if LabSchedules and timetables are occupied at day i+1 */
1465     void displayLabSetStatus(int i)
1466     {
1467         outfile<<TT[0].isOccupied(i+1,1)<<"for TT[0]\n";
1468         outfile<<TT[1].isOccupied(i+1,1)<<"for TT[1]\n";
1469         outfile<<TT[2].isOccupied(i+1,5)<<"for TT[2]\n";
1470         outfile<<TT[3].isOccupied(i+1,5)<<"for TT[3]\n";
1471         outfile<<P_Lab.getCell(i+1,1).isOccupied()<<"status of P_LabgetCelloccupied slot 1\n";
1472         outfile<<C_Lab.getCell(i+1,1).isOccupied()<<"status of C_LabgetCelloccupied slot 1\n";
1473         outfile<<P_Lab.getCell(i+1,2).isOccupied()<<"status of P_LabgetCelloccupied slot 2\n";
1474         outfile<<C_Lab.getCell(i+1,2).isOccupied()<<"status of C_LabgetCelloccupied slot 2\n";
1475     }
1476     /* To integrate the contents of labSchedule object with TimeTable object after the labSchedule object is optimised
1477     independently */
1478     void integrateLabSetAndTimeTable()
1479     {
1480         /* This function 1st sets the primary slot-clash avoiding labSchedule before perturbation */
1481         void setLabSchedule()
1482         {
1483             /* This function makes primary timetable by considering preferences of professors sequentially. This timetable is to be optimised
1484             void makeTimeTable(Professor P[], LabTA PL[], LabTA CL[], TutorialTA T[], Timetable TT[])
1485             {
1486                 ...
1487             }
1488         }
1489     }
1490 }

C:\Users\Raju\Downloads\140010037_371\source\Optimised_TimeTable_Generator_v1.cpp Windows (CR-LF) WINDOWS-1252 Line 1458, Column 1 Insert Read/Write default 20:26 15-04-2015
```

But the heart of the program, the perturbation related functions are located in class Timetable and is shown below:



The screenshot shows a C++ IDE with the file `Optimised_TimeTable_Generator_v1.cpp` open. The code is organized into several functions with line numbers on the left margin. The functions shown are:

- `bool setElement(int day, int slot, int ID, int profIndex, int division)` (lines 1120-1123)
- `void setLabSlot(int slot)` (lines 1277-1280)
- `void absSetElement(int day, int subslot, int ID)` (lines 1306-1310)
- `void exchangeSlots(int day1, int day2, int subslot1, int subslot2, int profIndex1, int profIndex2, int slot, int division)` (lines 1316-1320)
- `void exchangeSlots(int day1, int day2, int subslot1, int subslot2, int profIndex, int slot, int division)` (lines 1329-1332)
- `void perturbTimeTable(int slot, int division)` (lines 1340-1342)

The code includes comments explaining the purpose of each function, such as "Reserve lab slot according to institute rules" and "This function makes perturbations to the existing timetable by exchanging Cells of TimeTable so as to improve overall objective function".

The above deals with the implementation of the algorithm module.

As for the output module implementation, we once again have a couple of functions as shown:

The last three of functions displayed in the below screen shot basically deals with the output display on the console:

```
1746 outfile<<P_Lab.getCell(i+1,2).isOccupied()<<"status of P_LabgetCelloccupied slot 2\n";
1747 outfile<<C_Lab.getCell(i+1,2).isOccupied()<<"status of C_LabgetCelloccupied slot 2\n";
1748
1749 /* To integrate the contents of labSchedule object with TimeTable object after the labSchedule object is optimised
1750 independently */
1751 void integrateLabsAndTimeTable()
1752 {
1753     /* This function 1st sets the primary slot-clash avoiding labSchedule before perturbation */
1754 }
1755 void setLabSchedule()
1756 {
1757     /* This function makes primary timetable by considering preferences of professors sequentially. This timetable is to be optimised
1758     void makeTimeTable(Professor P[],LabTA FL[],LabTA CL[],TutorialTA T[],Timetable TT[])
1759 }
1760
1761 /* To display timetable in an output file */
1762 void outfileTimetable(Timetable &T)
1763 {
1764 }
1765
1766 /*
1767 This function is used to output the timetable in a tabular form on the console
1768 */
1769 void console_output(int div)
1770 {
1771     /*This function calls the tabular form of timetable for a particular division*/
1772 }
1773 void Timetable_per_division_Output()
1774 {
1775     system("cls");
1776 }
1777
1778 /*INPUT: This function takes in data from user in the form of preference orders*/
```

Coming back to the input we had entered initially, for the default case 2, we are asked to enter the number of the division corresponding to the time table we would wish to have.

If we choose to have the timetable for division 1, we enter 1 and we get its timetable. The same follows for the other divisions.

```
Enter the division whose timetable you want to see.
1. D1
2. D2
3. D3
4. D4
5. Exit
Your choice: 1
```

And the timetable generated is:

The screenshot shows the application window titled "Optimised\_TimeTable\_Generator\_v1.cpp - Code::Blocks svn build". The main window displays the following text:

```

Enter the division whose timetable you want to see.
1. D1
2. D2
3. D3
4. D4
5. Exit
Your choice: 1

Day      08:30-09:30    09:30-10:30    10:30-11:30    11:30-14:00    14:00-15:00    15:00-16:00    16:00-17:00    17:00-18:00
Monday      0            0            0            0            1005 BB101      0            0
Tuesday     203 PH117L    203 PH117L    203 PH117L    203 PH117L    1005 BB101    102 MA106    0
Wednesday   203 PH117L    203 PH117L    203 PH117L    203 PH117L    1002 CH107    1004 IC102    101 PH107
Thursday     301 CH117L    301 CH117L    301 CH117L    301 CH117L    1001 PH107    1002 CH107    1003 MA106    1004 IC102
Friday       301 CH117L    301 CH117L    301 CH117L    301 CH117L    1001 PH107    1002 CH107    1003 MA106    1004 IC102

You can view time-tables for other divisions as well or you may choose to exit.

1. D1
2. D2
3. D3
4. D4
5. Exit
Your choice: -
  
```

And for division 2:

The screenshot shows the application window titled "Optimised\_TimeTable\_Generator\_v1.cpp - Code::Blocks svn build". The main window displays the following text:

```

1. D1
2. D2
3. D3
4. D4
5. Exit
Your choice: 2

Day      08:30-09:30    09:30-10:30    10:30-11:30    11:30-14:00    14:00-15:00    15:00-16:00    16:00-17:00    17:00-18:00
Monday      0            0            0            0            0            1005 BB101      0            0
Tuesday     303 CH117L    303 CH117L    303 CH117L    303 CH117L    102 MA106    1005 BB101    0            0
Wednesday   303 CH117L    303 CH117L    303 CH117L    303 CH117L    1004 IC102    1002 CH107    101 PH107    1005 BB101
Thursday     201 PH117L    201 PH117L    201 PH117L    201 PH117L    1002 CH107    1001 PH107    1004 IC102    1003 MA106
Friday       201 PH117L    201 PH117L    201 PH117L    201 PH117L    1002 CH107    1001 PH107    1004 IC102    1003 MA106

You can view time-tables for other divisions as well or you may choose to exit.

1. D1
2. D2
3. D3
4. D4
5. Exit
Your choice: -
  
```

And for division 3:

Optimised\_TimeTable\_Generator\_v1.cpp - Code::Blocks svn build

C:\Users\Raju\Downloads\140010037\_371\source\Optimised\_TimeTable\_Generator\_v1.exe

You can view time-tables for other divisions as well or you may choose to exit.

1. D1
2. D2
3. D3
4. D4
5. Exit

Your choice: 3

Day	08:30-09:30	09:30-10:30	10:30-11:30	11:30-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00
Monday	1006 PH107	1007 CH107	1008 MA106	1009 IC102	0	0	0	0
Tuesday	1006 PH107	1007 CH107	1008 MA106	1009 IC102	204 PH117L	204 PH117L	204 PH117L	204 PH117L
Wednesday	1007 CH107	1009 IC102	1010 BB101	103 PH107	204 PH117L	204 PH117L	204 PH117L	204 PH117L
Thursday	1010 BB101	104 MA106	0	0	302 CH117L	302 CH117L	302 CH117L	302 CH117L
Friday	1010 BB101	0	0	0	302 CH117L	302 CH117L	302 CH117L	302 CH117L

You can view time-tables for other divisions as well or you may choose to exit.

1. D1
2. D2
3. D3
4. D4
5. Exit

Your choice: -

And for division 4:

Optimised\_TimeTable\_Generator\_v1.cpp - Code::Blocks svn build

C:\Users\Raju\Downloads\140010037\_371\source\Optimised\_TimeTable\_Generator\_v1.exe

You can view time-tables for other divisions as well or you may choose to exit.

1. D1
2. D2
3. D3
4. D4
5. Exit

Your choice: 4

Day	08:30-09:30	09:30-10:30	10:30-11:30	11:30-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00
Monday	1007 CH107	1006 PH107	1009 IC102	1008 MA106	0	0	0	0
Tuesday	1007 CH107	1006 PH107	1009 IC102	1008 MA106	304 CH117L	304 CH117L	304 CH117L	304 CH117L
Wednesday	1009 IC102	1007 CH107	103 PH107	1010 BB101	304 CH117L	304 CH117L	304 CH117L	304 CH117L
Thursday	104 MA106	1010 BB101	0	0	202 PH117L	202 PH117L	202 PH117L	202 PH117L
Friday	0	1010 BB101	0	0	202 PH117L	202 PH117L	202 PH117L	202 PH117L

You can view time-tables for other divisions as well or you may choose to exit.

1. D1
2. D2
3. D3
4. D4
5. Exit

Your choice:

And we enter 5 to exit.

This basically gives a broad outlook into our implementation of the program.



## **8 Challenges Faced (and how the problems were solved):**

Getting the algorithm right was a tough one. Since so many functions were interlinked, coming up with a clear cut algorithm was time consuming and constant checks were needed. After a lot of trials and altering the logic of modules of code we were able to overcome the difficulties of the algorithm and received a functional result.

The most common challenge faced by any programmer is debugging. And debugging 2500 lines of code was on hell of a task. A lot of errors were occurring due to array limits going out of range and finding out each and every one of them was not easy. Syntax errors were considerably better to correct. One of the most common errors were global functions being declared as “out of scope”. This was because their definition and call positions were not appropriate. This was solved by including the function declarations at the beginning of our program.

Another set of errors occurred when we integrated all our pieces of code. Due to high interdependency among the functions, any alteration had a very high chance of resulting in an error. However, we overcame this by verifying every single line of code from various versions and correcting it into the valid one.

Even determining the test cases was a challenge. And redirecting the input into the system assigning preferences once again required a lot of labour. But after spending a lot of time and discussing it out, we settled on the default cases as explained in the implementation.

Time was also a very crucial factor and we had to sacrifice a lot to accomplish this.

## **9 Instructions To Run Our Code:**

Make sure codeblocks has been pre-installed.

Log into moodle or github and download the compressed folder of our code.

The link to download from moodle is: <http://moodle.iitb.ac.in/mod/assign/view.php?id=6533>

Decompress the folder and you will be able to view our .cpp file named:

Optimised\_TimeTable\_Generator\_v1

Open codeblocks, click on file option in the menu bar, click open and browse to open our .cpp file.

Click on compile option.

Click on run and you should definitely be able to carry out the timetable generation.

## **10 Future Prospects:**

We can attempt to create a timetable for the entire institute which would include a lot of courses and variety of different slot clashes, so obviously it would include a variety of variables. The number of

students registered for each course will vary. Also, one professor might be able to teach different courses ( more than one) and he can teach for students belonging to different years of engineering.

The slots clashes will once again have to be managed in this case, in a more diligent manner ofcourse.

We can also include constraints such as distance between lecture halls of consecutive classes for students to commute in a comfortable manner.

Also, we can generate additional department timetables too, along with the teachers' and student' timetables too

The algorithm too, will have to be improved to manage the above scenario in a more efficient manner.