**R.V. COLLEGE OF ENGINEERING, Bengaluru-560059**

(Autonomous Institution Affiliated to VTU, Belgaum).

****

**Timetable Architect**

Experiential Learning Report

submitted by:

Vineeth Rao (RVCE23BCS035)

Shreya Prasad(RVCE23BCS130)

Submitted to:

**Prof. Vanishree K**

Assistant Professor

Department of Information Science

and Engineering

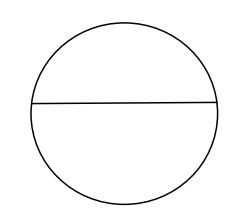
**(R.V. COLLEGE OF ENGINEERING, BENGALURU - 560059 Autonomous Institution Affiliated to VTU, Belgaum) DEPARTMENT OF INFORMATION SCIENCE ENGINEERING.**

****

**CERTIFICATE**

It is certified that the Experiential learning titled **Timetable Architect** is carried out by **Vineeth Rao** and **Shreya Prasad** who are bonafide students of R.V College of Engineering, Bengaluru, during the first semester, in the year **2023-2024**. It is also certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the report. The report has been approved as it satisfies the academic requirements in respect of Experiential learning.

Marks Awarded: **Signature of staff In-charge.**

** Signature of Head of the Department.**

**Table of contents**

|  |  |
| --- | --- |
| 1 . Abstract | 4 |
| 2. Objective | 4 |
| 3. Introduction | 5 |
| 4. Methodology | 5 |
| 5. Code | 7 |
| 6. output | 31 |
| 7. References | 31 |
| 8. Conclusion | 32 |

## Abstract

The increasing complexity of university schedules, with diverse courses, electives, and student preferences, necessitates the development of advanced tools to streamline the timetable creation process. This report presents Timetable Architect, a comprehensive desktop application designed to create optimal timetables for universities, with a focus on avoiding class collisions, accommodating electives, and optimizing student performance.

Timetable Architect employs a sophisticated algorithm that considers various constraints and preferences to generate conflict-free timetables. The application allows administrators to input course details, faculty availability, room constraints, and elective choices. It then intelligently allocates resources, minimizes schedule conflicts, and optimizes the use of available time slots.

By addressing the challenges associated with timetable creation in universities, Timetable Architect contributes to enhancing the overall educational experience for both students and faculty. The application aims to streamline scheduling processes, reduce administrative burdens, and ultimately create an environment conducive to academic success.

## Objectives

* **Conflict-free timetable:** Develop a robust algorithm that minimizes class collisions, ensuring that students can enroll in their desired courses without schedule conflicts.
* **Streamlined Administrative Processes:** Streamline the timetable creation process to reduce administrative burdens, save time, and enhance overall efficiency in managing academic schedules.
* **Optimized Student Performance:** Design the application to go beyond collision avoidance and actively seek to optimize student performance by considering factors such as course sequencing, study patterns, and workload distribution.
* **Elective Accommodation:** Enable the efficient allocation of elective courses, considering both student preferences and faculty availability, to create well-balanced and diverse timetables.

## Introduction

In today's dynamic educational landscape, efficient timetable management plays a pivotal role in ensuring the smooth functioning of academic institutions. The task of creating comprehensive timetables for colleges is a complex and time-consuming process that requires meticulous planning, consideration of various constraints, and the optimization of resources. Recognizing the need for an automated solution, we have undertaken the development of a project aimed at creating intelligent timetables for colleges using C++.

The primary objectives of our project are to streamline the timetable generation process, prevent class collisions, and optimize workload management for students. The significance of an automated system lies in its ability to not only save time but also enhance the overall efficiency of the educational institution. By leveraging the power of C++, we aim to create a robust and scalable solution that addresses the intricate challenges associated with timetable creation.

## Methodology

1. Class Structure:

section Class:

Manages core and lab subjects, blocking time slots, and generating timetables.

Handles storage of timetables in CSV files.

Allocates teachers, subjects, and rooms based on constraints.

teacher Class:

Represents teachers with attributes like name and timetable.

subject Class:

Represents subjects with attributes like name, credits, hours per credit, and room preferences.

2. Timetable Generation:

addCore Method:

Adds core subjects and teachers to respective vectors.

addLab Method:

Adds lab subjects, teachers, and the number of labs to respective vectors.

block Method:

Blocks specific time slots with a teacher and subject.

storeTimeTable Method:

Stores the generated timetable in a CSV file, separating tables for the timetable, teacher, and room.

makeTIMETABLE Method:

Implements a complex algorithm for allocating core and lab subjects to available teachers, subjects, and rooms.

Handles collisions and updates the timetable, teacher table, and room table.

3. Lab Allocation:

The makeTIMETABLE method allocates labs to available rooms and teachers based on various constraints and preferences.

Uses sophisticated algorithms to find suitable combinations of teachers, subjects, and rooms.

4. Elective Time Slot Analysis:

The intersectElectives function analyzes the availability of time slots for elective subjects based on the given teacher list and credit requirements.

5. Error Handling:

Includes some error checks and collision detection mechanisms in the makeTIMETABLE method.

6. Documentation and Comments:

Lack of comments makes understanding the code challenging. Comprehensive comments and documentation would improve readability and assist other developers.

7. Refactoring and Optimization:

Some parts of the code, especially within the makeTIMETABLE method, are lengthy and complex. Refactoring for clarity and optimization may be beneficial.

8. User Interface:

Consider adding a user interface to enhance user interaction and make the program more accessible.

9. Testing:

Thoroughly test the timetable generation algorithm to ensure correctness, especially under different scenarios and constraints.

10. Scalability:

Evaluate the program's scalability, especially when dealing with a large number of subjects, teachers, and rooms.

11. Future Development:

Plan for future developments, improvements, and additional features that may enhance the program's functionality.

Conclusion:

The "TimeTable Architect" program showcases a comprehensive approach to timetable generation, considering various constraints and preferences. Enhancements in code readability, documentation, user interface, and testing can contribute to the program's usability and maintainability. Consideration of future developments and scalability will ensure the program remains effective as requirements evolve.

## Code

#include<fstream>

#include<algorithm>

#include<list>

#include<string>

#include<iostream>

#include<vector>

#define days 6

#define periods 6

class subject {

public:

std::string name;

std::string elective;

bool lab;

int credits, hoursPerCredit;

unsigned short int bFactor;

std::string\* rooms;

unsigned short int noRooms;

bool readData(std::string inp);

std::string convertToString();

subject() {

credits = 0;

hoursPerCredit = 0;

bFactor = 0;

};

};

bool subject::readData(std::string inp) {

enum format {

namen,

electiven,

labn,

creditsn,

hoursPerCreditn,

bFactorn,

roomsn,

};

int commaCount = 0;

try {

for (int i = 0; i < inp.size(); i++) {

if (inp[i] == ',') {

commaCount++;

continue;

}

switch (commaCount) {

case namen:

name.push\_back(inp[i]);

break;

case electiven:

elective.push\_back(inp[i]);

break;

case labn:

lab = inp[i] - '0';

break;

case creditsn:

credits = credits \* 10 + inp[i] - '0';

break;

case hoursPerCreditn:

hoursPerCredit = hoursPerCredit \* 10 + inp[i] - '0';

break;

case bFactorn:

bFactor = bFactor \* 10 + inp[i] - '0';

break;

case roomsn:

int commaCount = 0;

i++;

std::vector<std::string> temp;

while (true) {

if (inp[i] == ']') {

noRooms = commaCount + 1;

rooms = new std::string[noRooms];

for (commaCount; commaCount >= 0; commaCount--) {

rooms[commaCount] = temp[commaCount];

}

break;

}

else if (inp[i] == ',') {

commaCount++;

}

else {

if (temp.size() > commaCount) {

temp[commaCount].push\_back(inp[i]);

}

else {

temp.push\_back("");

temp[commaCount].push\_back(inp[i]);

}

}

i++;

}

}

}

}

catch (...) {

return 0;

}

}

class room {

public:

std::string name;

int capacity = 0;

bool labOrNot = 0;

std::string building;

bool timeTable[days][periods];//when room is free. 0 for free 1 for occupiued.

int timeTableName[days][periods];//what class the rooms is holding. changes done by program, not taken from user.

bool readData(std::string inp);

void showTimeTable();

std::string convertToString();

room() {

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

for (int j = 0; j < periods; j++) {

timeTableName[i][j] = 0;

}

}

}//initialise all classes to zero.

};

void room::showTimeTable() {

// for(int i=0;i<days;i++){

// for(int j=0;j<periods;j++){

// std::cout<<timeTable[i][j]<<" , ";

// }

// std::cout<<std::endl;

// }

}

bool room::readData(std::string inp) {

enum format {

namen,

capacityn,

labOrNotn,

buildingn,

timeTablen

};

int commacount = 0;

try {

for (int i = 0; i < inp.size(); i++) {

if (inp[i] == ',') {

commacount++;

continue;

}

switch (commacount) {

case namen:

name.push\_back(inp[i]);

break;//read name

case capacityn:

capacity = capacity \* 10 + inp[i] - '0';

break;//read branch

case labOrNotn:

labOrNot = inp[i] - '0';

break;//read workhours

case buildingn:

building.push\_back(inp[i]);

break;//read name

case timeTablen:

commacount = 0;

int strptr = i;

while (inp[strptr]) {

if (inp[strptr] == ',') {

commacount++;

}

else if (!(commacount % 2)) {

timeTable[commacount / 12][(commacount % 12) / 2] = inp[strptr] - '0';

}

else {

timeTableName[commacount / 12][(commacount % 12) / 2] = timeTableName[commacount / 12][(commacount % 12) / 2] \* 10 + inp[strptr] - '0';

}

strptr++;

}

goto a;

break;

}

}

}

catch (...) {

return 0;

}

a:

return 1;

}

std::string room::convertToString() {

std::string output;

output += name + "," + std::to\_string(capacity) + "," + std::to\_string(labOrNot) + "," + building;

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

for (int j = 0; j < days; j++) {

output += "," + std::to\_string(timeTable[i][j]) + "," + std::to\_string(timeTableName[i][j]);

}

}

return output;

}

std::string subject::convertToString() {

std::string out;

out = name + ",";

out += elective + ",";

out += std::to\_string(lab) + ",";

out += std::to\_string(credits) + ",";

out += std::to\_string(hoursPerCredit) + ",";

out += std::to\_string(bFactor) + ",[";

for (int i = 0; i < noRooms - 1; i++) {

out += rooms[i] + ",";

}

out += rooms[noRooms - 1] + "]";

return out;

}

class teacher {

public:

std::string name;//name of teacher

std::string branch;//branch of teacher

//unsigned int workHours=0; //how many hours the teacher would work

bool timeTable[days][periods];//when teacher is free. 1 for free 0 for occupiued.

unsigned int timeTableName[days][periods];//what teacher is teaching when occupied. changes done by program, not taken from user.

bool readData(std::string inp);//function which converts std::string input from teacherdata to the objects data

std::string convertToString();//reverse of above

void showTimeTable();

teacher() {

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

for (int j = 0; j < periods; j++) {

timeTableName[i][j] = 0;

}

}

}

//initialise all classes to zero.

// teacher(const teacher &bob){

// name=bob.name;

// branch=bob.branch;

// for(int i=0;i<days;i++){

// for(int j=0;j<periods;j++){

// timeTable[i][j]=bob.timeTable[i][j];

// timeTableName[i][j]=bob.timeTableName[i][j];

// }

// }

// }

};

bool teacher::readData(std::string inp) {

enum format {

namen,

branchn,

//workHoursn,

timeTablen

};//Used for switch statements. each field corresponds to its location in the std::string

int commacount = 0;//numebr of commas encountered

try {

for (int i = 0; i < inp.size(); i++) {

if (inp[i] == ',') {

commacount++;

continue;

}//equivalent to next column

switch (commacount) {

case namen:

name.push\_back(inp[i]);

break;//read name

case branchn:

branch.push\_back(inp[i]);

break;//read branch

/\*case workHoursn:

workHours=workHours\*10+inp[i]-'0';

break;//read workhours

\*/

case timeTablen:

commacount = 0;

int strptr = i;

while (inp[strptr]) {

if (inp[strptr] == ',') {

commacount++;

}

else if (!(commacount % 2)) {

timeTable[commacount / 12][(commacount % 12) / 2] = inp[strptr] - '0';

}

else {

timeTableName[commacount / 12][(commacount % 12) / 2] = timeTableName[commacount / 12][(commacount % 12) / 2] \* 10 + inp[strptr] - '0';

}

strptr++;

}

goto a;

break;

}

}

}

catch (...) {

return 0;

}

a:

return 1;

}

std::string teacher::convertToString() {

std::string output;

output = name;

output.push\_back(',');

output += branch;

//output.push\_back(',');

//output+=std::to\_string(workHours);

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

for (int j = 0; j < days; j++) {

output += "," + std::to\_string(timeTable[i][j]) + "," + std::to\_string(timeTableName[i][j]);

}

}

return output;

}

void teacher::showTimeTable() {

// for(int i=0;i<days;i++){

// for(int j=0;j<periods;j++){

// std::cout<<timeTable[i][j]<<" ";

// }

// std::cout<<std::endl;

// }

}

//#include<bits/stdc++.h>

/\*

First step:

Configure teachers and subjects

1) use addElective(teacher for subject,subject) to assign teachers and subject . passed arguments are the teacher and subject objects that have been provided;

2)use addLabs(list of teachers who are can take the class,number of the teachers given in the prev parameter,lab subject).

3)use block(day,period,message at blocked place,teacher at blocked place) to prevent the program from assigning class at that time also alters timetable name at t the places mentioned.

\*/

class section {

public:

int name;

std::vector<teacher> allTeachers;

std::vector<teacher> coreTeachers;

std::vector<subject> coreSubjects;

void addCore(teacher Teacher, subject Subject);

//std::vector<std::vector<teacher>> electiveTeachers;

//std::vector<subject> electiveSubjects;

std::vector<std::vector<teacher>> labTeachers;

std::vector<subject> labSubjects;

std::vector<int> noOfLabs;

void addLab(teacher Teacher[], int noteachers, subject Subject, int noLabs);

std::vector<room> allRooms;

std::vector<std::string> defaultRooms;

std::string timeTable[days][periods];

std::string teacherTable[days][periods];

std::string roomTable[days][periods];

void displayTimeTable();

void displayTeacherTable();

void displayClassTable();

void storeTimeTable();

//void addElective(teacher Teacher[],int numberOfTeachers,subject Subject);

void block(int a, int b, std::string subject, std::string teacher);

void makeTIMETABLE();

section() {

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

for (int j = 0; j < periods; j++) {

timeTable[i][j] = "f";

teacherTable[i][j] = "f";

roomTable[i][j] = "NA";

}

}

}

int dayfactor[days] = { 0 };

subject& returnSubject(std::string inp) {

error\_ = false;

for (int i = 0; i < coreSubjects.size(); i++) {

if (coreSubjects[i].name == inp) {

return coreSubjects[i];

}

}

for (int i = 0; i < labSubjects.size(); i++) {

if (labSubjects[i].name == inp) {

return labSubjects[i];

}

}

error\_ = true;

return coreSubjects[coreSubjects.size() - 1];

}

bool error\_ = false;

int weight(int& dayfactor, int time, int bfactor) {

return (time \* 10 + bfactor \* 5 + dayfactor);

}

room& returnRoom(std::string name) {//edit actual room

error\_ = false;

for (int i = 0; i < allRooms.size(); i++) {

if (allRooms[i].name == name) {

return allRooms[i];

}

}

error\_ = true;

return allRooms[allRooms.size() - 1];//fix and make program throw error saying room not there

}

teacher& returnTeacher(std::string inp) {//return teacher function

error\_ = false;

for (int i = 0; i < allTeachers.size(); i++) {

if (allTeachers[i].name == inp) {

return allTeachers[i];

}

}

error\_ = true;

return allTeachers[allTeachers.size() - 1];

}

};

void section::displayTimeTable() {

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

for (int j = 0; j < periods; j++) {

std::cout << timeTable[i][j] << " ";

}

std::cout << " bfactor: " << dayfactor[i] << std::endl;

}

}

void section::displayTeacherTable() {

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

for (int j = 0; j < periods; j++) {

std::cout << teacherTable[i][j] << " ";

}

std::cout << std::endl;

}

}

void section::displayClassTable() {

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

for (int j = 0; j < periods; j++) {

std::cout << roomTable[i][j] << " ";

}

std::cout << std::endl;

}

}

void section::addCore(teacher Teacher, subject Subject) {

coreTeachers.push\_back(Teacher);

coreSubjects.push\_back(Subject);

}

void section::addLab(teacher Teacher[], int noteachers, subject Subject, int noLabs) {

std::vector<teacher> a;

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

bool flag = 1;

a.push\_back(Teacher[i]);

}

labTeachers.push\_back(a);

labSubjects.push\_back(Subject);

noOfLabs.push\_back(noLabs);

}

void section::block(int i, int j, std::string Teacher, std::string Subject) {

timeTable[i][j] = Teacher;

teacherTable[i][j] = Subject;

}

void section::storeTimeTable() {

std::fstream storage;

std::string bob = "storage/" + std::to\_string(name) + ".csv";

storage.open(bob, std::ios::app);

storage << "TIME TABLE: \n";

if (storage.is\_open()) {

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

bob = "";

for (int j = 0; j < periods - 1; j++) {

bob += timeTable[i][j] + ",";

}

bob += timeTable[i][periods - 1] + "\n";

storage << bob;

}

storage << "\nTEACHER TABLE:\n";

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

bob = "";

for (int j = 0; j < periods - 1; j++) {

bob += teacherTable[i][j] + ",";

}

bob += teacherTable[i][periods - 1] + "\n";

storage << bob;

}

storage << "\nROOM TABLE:\n";

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

bob = "";

for (int j = 0; j < periods - 1; j++) {

bob += roomTable[i][j] + ",";

}

bob += roomTable[i][periods - 1] + "\n";

storage << bob;

}

storage.close();

}

}

void section::makeTIMETABLE() {

int creditsl;

bool collision = true;

collision = true;

for (int i = 0; i < labSubjects.size(); i++) {

std::vector<std::vector<room>> candidates;

std::vector<int> intersectionCount;

int intersections;

for (int j = 0; j < labSubjects[i].noRooms; j++) {

for (int k = j + 1; k < labSubjects[i].noRooms; k++) {

intersections = 0;

room curRoomA = returnRoom(labSubjects[i].rooms[j]);

room curRoomB = returnRoom(labSubjects[i].rooms[k]);

for (int a = 0; a < days; a++) {

for (int b = 0; b < periods; b += 2) {

if (!curRoomA.timeTable[a][b] && !curRoomA.timeTable[a][b + 1] && !curRoomB.timeTable[a][b] && !curRoomB.timeTable[a][b + 1] && timeTable[a][b] == "f" && timeTable[a][b + 1] == "f") {

intersections++;

}

}

}

if (intersections >= labSubjects[i].credits) {

std::vector<room> selection = { curRoomA,curRoomB };

candidates.push\_back(selection);

intersectionCount.push\_back(intersections);

}

}

}

int highest = 0, highestindex = 0;

for (int j = 0; j < candidates.size(); j++) {

if (intersectionCount[j] > highest) {

highestindex = j;

highest = intersectionCount[j];

}

}

room curRoomA = candidates[highestindex][0];

room curRoomB = candidates[highestindex][1];

bool intersectionTimes[days][periods] = {};

for (int a = 0; a < days; a++) {

for (int b = 0; b < periods; b += 2) {

intersectionTimes[a][b] = !curRoomA.timeTable[a][b] && !curRoomA.timeTable[a][b + 1] && !curRoomB.timeTable[a][b] && !curRoomB.timeTable[a][b + 1];

intersectionTimes[a][b + 1] = !curRoomA.timeTable[a][b] && !curRoomA.timeTable[a][b + 1] && !curRoomB.timeTable[a][b] && !curRoomB.timeTable[a][b + 1];

}

}

int creditsForLab;

std::vector<teacher> teacherListForLab = labTeachers[i];

std::vector<std::vector<teacher>> comb;

std::vector<teacher> selectedTeachers;

int lowest = 36;

int intersectionTable[days][periods] = { 0 };

for (int a = 0; a < teacherListForLab.size() - 3; ++a) {

for (int j = a + 1; j < teacherListForLab.size() - 2; ++j) {

for (int m = j + 1; m < teacherListForLab.size() - 1; ++m) {

for (int n = m + 1; n < teacherListForLab.size(); ++n) {

creditsForLab = labSubjects[i].credits;

int tempTable[days][periods] = { 0 };

for (int k = 0; k < days; k++) {

for (int l = 0; l < periods; l++) {

if (intersectionTimes[k][l]) {

if (!returnTeacher(teacherListForLab[a].name).timeTable[k][l] && !returnTeacher(teacherListForLab[a].name).timeTable[k][l + 1] && !returnTeacher(teacherListForLab[j].name).timeTable[k][l] && !returnTeacher(teacherListForLab[j].name).timeTable[k][l + 1] && !returnTeacher(teacherListForLab[m].name).timeTable[k][l] && !returnTeacher(teacherListForLab[m].name).timeTable[k][l + 1] && !returnTeacher(teacherListForLab[n].name).timeTable[k][l] && !returnTeacher(teacherListForLab[n].name).timeTable[k][l + 1]) {

tempTable[k][l] = 1;

tempTable[k][l + 1] = 1;

creditsForLab--;

}

}

}

}

if (creditsForLab <= 0) {

std::vector<teacher> temp = { returnTeacher(teacherListForLab[a].name),returnTeacher(teacherListForLab[j].name),returnTeacher(teacherListForLab[m].name),returnTeacher(teacherListForLab[n].name) };

comb.push\_back(temp);

if (lowest > labSubjects[i].credits - creditsForLab) {

lowest = labSubjects[i].credits - creditsForLab;

selectedTeachers = temp;

for (int k = 0; k < days; k++) {

for (int l = 0; l < periods; l++) {

intersectionTable[k][l] = tempTable[k][l];

}

}

}

}

}

}

}

}

creditsForLab = labSubjects[i].credits;

for (int k = 0; k < days; k++) {

for (int l = 0; l < periods; l++) {

if (intersectionTable[k][l]) {

creditsForLab--;

break;

}

}

}

if (creditsForLab <= 0)

collision = false;

creditsForLab = labSubjects[i].credits;

//make collisiosn handelling

//update teacher timetable

if (!collision) {

for (int k = 0; k < days; k++) {

for (int l = 0; l < periods; l++) {

if (intersectionTable[k][l] && creditsForLab) {

dayfactor[k] += labSubjects[i].bFactor \* 5;

timeTable[k][l] = labSubjects[i].name;

timeTable[k][l + 1] = labSubjects[i].name;

teacherTable[k][l] = selectedTeachers[0].name + " , " + selectedTeachers[1].name + " , " + selectedTeachers[2].name + " , " + selectedTeachers[3].name;

teacherTable[k][l + 1] = selectedTeachers[0].name + " , " + selectedTeachers[1].name + " , " + selectedTeachers[2].name + " , " + selectedTeachers[3].name;

roomTable[k][l] = candidates[highestindex][0].name + " , " + candidates[highestindex][1].name;

roomTable[k][l + 1] = candidates[highestindex][0].name + " , " + candidates[highestindex][1].name;

room& tempUpdate = returnRoom(candidates[highestindex][0].name);

tempUpdate.timeTable[k][l] = 1;

tempUpdate.timeTable[k][l + 1] = 1;

tempUpdate.timeTableName[k][l] = name;

tempUpdate.timeTableName[k][l + 1] = name;

room& tempUpdateA = returnRoom(candidates[highestindex][1].name);

tempUpdateA.timeTable[k][l] = 1;

tempUpdateA.timeTable[k][l + 1] = 1;

tempUpdateA.timeTableName[k][l] = name;

tempUpdateA.timeTableName[k][l + 1] = name;

for (int p = 0; p < 4; p++) {

returnTeacher(selectedTeachers[p].name).timeTable[k][l] = 1;

returnTeacher(selectedTeachers[p].name).timeTable[k][l + 1] = 1;

returnTeacher(selectedTeachers[p].name).timeTableName[k][l] = 1;

returnTeacher(selectedTeachers[p].name).timeTableName[k][l + 1] = 1;

}

creditsForLab--;

break;

}

}

}

}

else {

//std::cout<<"bob";

}

}

//core subjects allocation

for (int i = 0; i < coreTeachers.size(); i++) {

// std::cout<<"\niterating through: "<<coreSubjects[i].name;

creditsl = 0;

std::vector<room> defRooms;

room roomDefault; int highest = 0;

for (int j = 0; j < defaultRooms.size(); j++) {

defRooms.push\_back(returnRoom(defaultRooms[j]));

int free = 0;

for (int k = 0; k < days; k++) {

for (int l = 0; l < periods; l++) {

if (!defRooms[j].timeTable[k][l]) {

free++;

}

}

}

if (free > highest) {

roomDefault = defRooms[j];

}

}

if (coreSubjects[i].hoursPerCredit == 1) {

for (int j = 0; j < days; j++) {

for (int k = 0; k < periods; k++) {

if (!returnTeacher(coreTeachers[i].name).timeTable[j][k]) {

if (timeTable[j][k] == "f") {

creditsl++;

}

}

}

}

if (creditsl >= coreSubjects[i].credits) {

collision = false;

}

else {

collision = true;

}

if (!collision) {

//std::cout<<"\nno collision for: "<<coreSubjects[i].name;

int currentAssigned = 0;

int count = 0;

repeat:

count++;

if (count < 37) {

std::vector<int> weights;

int numberclasses = coreSubjects[i].credits - currentAssigned;

for (int j = 0; j < days; j++) {

int discouragementFactor = 1;

for (int k = 0; k < periods; k++) {

if (timeTable[j][k] == coreSubjects[i].name) {

discouragementFactor = 200;

}

if (!returnTeacher(coreTeachers[i].name).timeTable[j][k]) {

if (timeTable[j][k] == "f") {

weights.push\_back(weight(dayfactor[j], k, coreSubjects[i].bFactor \* discouragementFactor));

}

}

}

}

std::list<int> weightsl;

for (int o = 0; o < weights.size(); o++) {

weightsl.push\_back(weights[o]);

}

weightsl.sort();

std::list<int>::iterator ptr = weightsl.begin();

int j = 0;

while (ptr != weightsl.end()) {

weights[j] = \*ptr;

ptr++;

j++;

}

for (int j = 0; j < days; j++) {

int discouragementFactor = 1;

for (int k = 0; k < periods; k++) {

if (timeTable[j][k] == coreSubjects[i].name) {

discouragementFactor = 200;

}

if (!returnTeacher(coreTeachers[i].name).timeTable[j][k]) {

if (timeTable[j][k] == "f") {

for (int a = 0; a < coreSubjects[i].credits - currentAssigned; a++) {

if (weight(dayfactor[j], k, coreSubjects[i].bFactor \* discouragementFactor) == weights[a] && numberclasses) {

numberclasses--;

dayfactor[j] += coreSubjects[i].bFactor \* 5;

timeTable[j][k] = coreSubjects[i].name;

teacherTable[j][k] = coreTeachers[i].name;

returnTeacher(coreTeachers[i].name).timeTable[j][k] = 1;

returnTeacher(coreTeachers[i].name).timeTableName[j][k] = name;

if (coreSubjects[i].rooms[0] == "0") {

if (!roomDefault.timeTable[j][k]) {

roomDefault.timeTable[j][k] = 1;

roomDefault.timeTableName[j][k] = name;

roomTable[j][k] = roomDefault.name;

}

else {

for (int s = 0; s < defRooms.size(); s++) {

if (!defRooms[s].timeTable[j][k]) {

defRooms[s].timeTable[j][k] = 1;

defRooms[s].timeTableName[j][k] = name;

roomTable[j][k] = defRooms[s].name;

goto a;

}

}

roomTable[j][k] = "?????";

}

}

else {

for (int q = 0; q < coreSubjects[i].noRooms; q++) {

room temp = returnRoom(coreSubjects[i].rooms[q]);

if (!temp.timeTable[j][k]) {

temp.timeTableName[j][k] = name;

temp.timeTable[j][k] = 1;

roomTable[j][k] = temp.name;

goto a;

}

}

}

a:

break;

}

}

}

}

}

}

if (numberclasses > 0) {

currentAssigned = coreSubjects[i].credits - numberclasses;

goto repeat;

}

}

}

else {

//std::cout<<"collision of subject "<<coreSubjects[i].name<<std::endl;

int assigned = coreSubjects[i].credits;

for (int j = 0; j < days; j++) {

for (int k = 0; k < periods; k++) {

if (!returnTeacher(coreTeachers[i].name).timeTable[j][k]) {

if (timeTable[j][k] == "f" && assigned) {

dayfactor[j] += coreSubjects[i].bFactor \* 5;

timeTable[j][k] = coreSubjects[i].name;

teacherTable[j][k] = coreTeachers[i].name;

returnTeacher(coreTeachers[i].name).timeTable[j][k] = 1;

returnTeacher(coreTeachers[i].name).timeTableName[j][k] = name;

assigned--;

if (coreSubjects[i].rooms[0] == "0") {

if (!roomDefault.timeTable[j][k]) {

roomDefault.timeTable[j][k] = 1;

roomDefault.timeTableName[j][k] = name;

roomTable[j][k] = roomDefault.name;

}

else {

for (int s = 0; s < defRooms.size(); s++) {

if (!defRooms[s].timeTable[j][k]) {

defRooms[s].timeTable[j][k] = 1;

defRooms[s].timeTableName[j][k] = name;

roomTable[j][k] = defRooms[s].name;

goto b;

}

}

roomTable[j][k] = "?????";

}

}

b:

break;

}

if (assigned) {

teacher& present = returnTeacher(teacherTable[j][k]);

subject& presentSub = returnSubject(timeTable[j][k]);

if (error\_) {

break;

}

for (int l = 0; l < days; l++) {

bool flag = 1;

for (int m = 0; m < periods; m++) {

if (timeTable[l][m] == presentSub.name) {

if (l != j || k != m) {

l++;

flag = 0;

}

}

}

if (flag) {

for (int m = 0; m < periods; m++) {

if (timeTable[l][m] == "f" && !present.timeTable[l][m] && assigned) {

subject& presentSub = returnSubject(timeTable[j][k]);

timeTable[j][k] = coreSubjects[i].name;

teacherTable[j][k] = coreTeachers[i].name;

dayfactor[j] += coreSubjects[i].bFactor \* 5 - presentSub.bFactor \* 5;

returnTeacher(coreTeachers[i].name).timeTable[j][k] = 1;

returnTeacher(coreTeachers[i].name).timeTableName[j][k] = name;

present.timeTable[j][k] = 0;

//check if default rooms for new subject

timeTable[l][m] = presentSub.name;

teacherTable[l][m] = present.name;

dayfactor[l] += presentSub.bFactor \* 5;

present.timeTable[l][m] = 1;

present.timeTableName[l][m] = name;

assigned--;

if (coreSubjects[i].rooms[0] == "0") {

if (!roomDefault.timeTable[l][m]) {

roomDefault.timeTable[l][m] = 1;

roomDefault.timeTableName[l][m] = name;

roomTable[l][m] = roomDefault.name;

}

else {

for (int s = 0; s < defRooms.size(); s++) {

if (!defRooms[s].timeTable[l][m]) {

defRooms[s].timeTable[l][m] = 1;

defRooms[s].timeTableName[l][m] = name;

roomTable[l][m] = defRooms[s].name;

break;

}

roomTable[j][k] = "?????";

}

}

break;

}

}

}

}

}

}

}

}

}

}

}

else if (coreSubjects[i].hoursPerCredit == 2) {

for (int j = 0; j < days; j++) {

for (int k = 0; k < periods; k += 2) {

if (!returnTeacher(coreTeachers[i].name).timeTable[j][k] && !returnTeacher(coreTeachers[i].name).timeTable[j][k + 1]) {

if (timeTable[j][k] == "f") {

creditsl++;

}

}

if (creditsl >= coreSubjects[i].credits) {

collision = false;

break;

}

}

}

if (!collision) {

int classesAssigned = 0;

int count = 0;

repeat2:

count++;

if (count < 37) {

std::vector<int> weights;

int numberclasses = coreSubjects[i].credits - classesAssigned;

for (int j = 0; j < days; j++) {

int discouragementFactor = 1;

for (int k = 0; k < periods; k += 2) {

if (timeTable[j][k] == coreSubjects[i].name) {

discouragementFactor = 200;

}

if (!returnTeacher(coreTeachers[i].name).timeTable[j][k] && !returnTeacher(coreTeachers[i].name).timeTable[j][k + 1]) {

if (timeTable[j][k] == "f") {

weights.push\_back(weight(dayfactor[j], k, coreSubjects[i].bFactor \* discouragementFactor));

}

}

}

}

std::list<int> weightsl;

for (int o = 0; o < weights.size(); o++) {

weightsl.push\_back(weights[o]);

}

weightsl.sort();

std::list<int>::iterator ptr = weightsl.begin();

int j = 0;

while (ptr != weightsl.end()) {

weights[j] = \*ptr;

ptr++;

j++;

}

for (int j = 0; j < days; j++) {

int discouragementFactor = 1;

for (int k = 0; k < periods; k += 2) {

if (timeTable[j][k] == coreSubjects[i].name) {

discouragementFactor = 200;

}

if (!returnTeacher(coreTeachers[i].name).timeTable[j][k] && !returnTeacher(coreTeachers[i].name).timeTable[j][k + 1]) {

if (timeTable[j][k] == "f") {

for (int a = 0; a < coreSubjects[i].credits - classesAssigned; a++) {

if (weight(dayfactor[j], k, coreSubjects[i].bFactor \* discouragementFactor) == weights[a] && numberclasses) {

numberclasses--;

dayfactor[j] += coreSubjects[i].bFactor \* 5;

timeTable[j][k] = coreSubjects[i].name;

teacherTable[j][k] = returnTeacher(coreTeachers[i].name).name;

returnTeacher(coreTeachers[i].name).timeTable[j][k] = 1;

returnTeacher(coreTeachers[i].name).timeTableName[j][k] = name;

timeTable[j][k + 1] = coreSubjects[i].name;

teacherTable[j][k + 1] = coreTeachers[i].name;

returnTeacher(coreTeachers[i].name).timeTable[j][k + 1] = 1;

returnTeacher(coreTeachers[i].name).timeTableName[j][k + 1] = name;

break;

if (coreSubjects[i].rooms[0] == "0") {

if (!roomDefault.timeTable[j][k] && !roomDefault.timeTable[j][k + 1]) {

roomDefault.timeTable[j][k] = 1;

roomDefault.timeTableName[j][k] = name;

roomTable[j][k] = roomDefault.name;

roomDefault.timeTable[j][k + 1] = 1;

roomDefault.timeTableName[j][k + 1] = name;

roomTable[j][k + 1] = roomDefault.name;

}

else {

for (int s = 0; s < defRooms.size(); s++) {

if (!defRooms[s].timeTable[j][k] && !defRooms[s].timeTable[j][k + 1]) {

defRooms[s].timeTable[j][k] = 1;

defRooms[s].timeTableName[j][k] = name;

roomTable[j][k] = defRooms[s].name;

defRooms[s].timeTable[j][k + 1] = 1;

defRooms[s].timeTableName[j][k + 1] = name;

roomTable[j][k + 1] = defRooms[s].name;

goto c;

}

}

roomTable[j][k] = "?????";

}

}

else {

for (int q = 0; q < coreSubjects[i].noRooms; q++) {

room temp = returnRoom(coreSubjects[i].rooms[q]);

if (!temp.timeTable[j][k]) {

temp.timeTableName[j][k] = name;

temp.timeTable[j][k] = 1;

roomTable[j][k] = temp.name;

goto c;

}

}

}

c:

break;

}

}

}

}

}

}

if (numberclasses > 0) {

classesAssigned = coreSubjects[i].credits - numberclasses;

goto repeat2;}

}

}

}

}

}

std::vector<std::vector<bool>> intersectElectives(std::vector<teacher> teacherList, int credits) {

int intersectionCount[days][periods] = { 0 };

std::vector<std::vector<bool>> returnVal(6, std::vector<bool>(6, false));

for (int i = 0; i < teacherList.size(); i++) {

for (int j = 0; j < days; j++) {

for (int k = 0; k < periods; k++) {

if (!teacherList[i].timeTable[j][k]) {

intersectionCount[j][k]++;}

}

}

}

for (int j = 0; j < days; j++) {

for (int k = 0; k < periods; k++) {

if (intersectionCount[j][k] > credits) {

returnVal[j][k] = 1;

break;

}

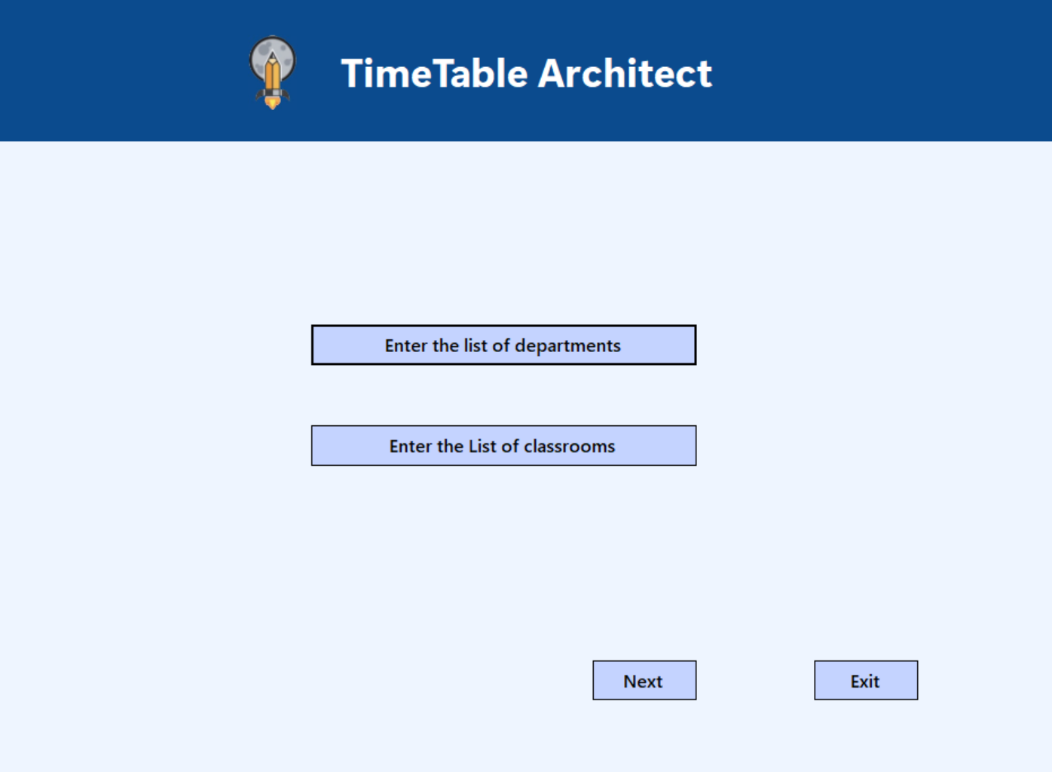
}

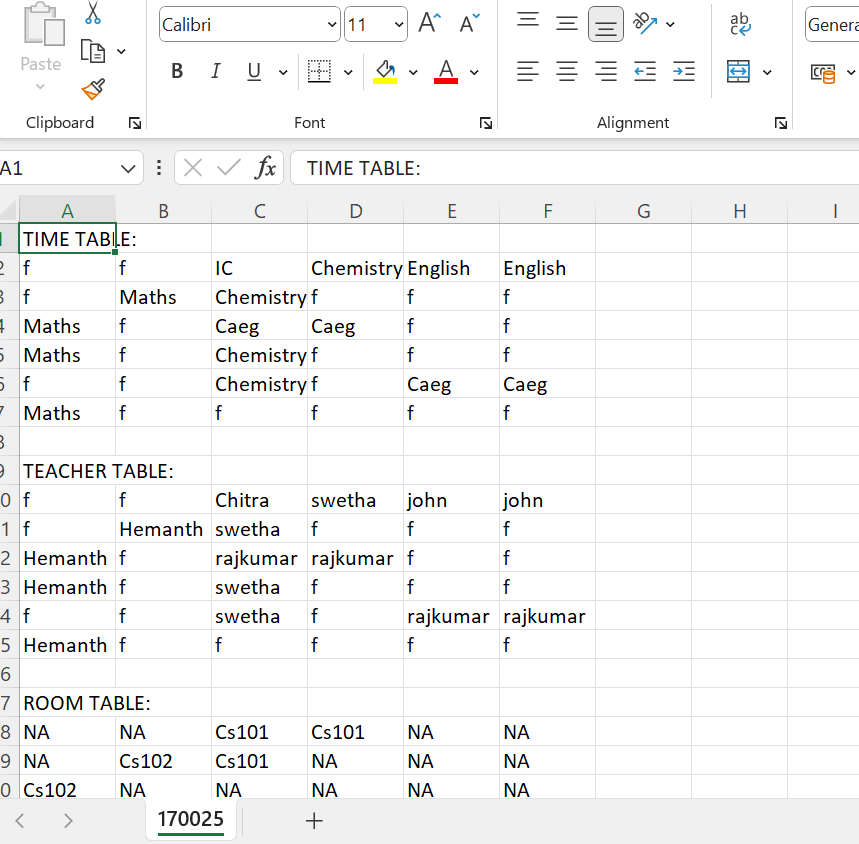
}

return returnVal;

}

## Output





## References

1. <https://en.cppreference.com/w/cpp/container/vector>
2. <https://books.google.co.in/books?id=-tahz50XWEQC&lpg=PA1&ots=byUMVhoinp&dq=vectors%20in%20c%2B%2B&lr&pg=PA126#v=onepage&q=vectors%20in%20c++&f=false>
3. R. Bonifácio, F. Carvalho, G. N. Ramos, U. Kulesza and R. Coelho, "The use of C++ exception handling constructs: A comprehensive study," 2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM), Bremen, Germany, 2015, pp. 21-30, doi: 10.1109/SCAM.2015.7335398. keywords: {Open source software;Java;Semantics;Data collection;Libraries;Time factors}

## Conclusion

The development of an Automated Timetable Generation System for Colleges using C++ is a significant step towards modernizing and optimizing academic operations. By addressing challenges such as class collisions and workload management, our project aims to contribute to the efficient functioning of educational institutions. As we progress with the implementation, we anticipate positive outcomes that will revolutionize the way colleges manage their timetables, ultimately benefiting both staff and students.

By automating the timetable creation process, our system aims to save valuable time for administrators and staff, allowing them to allocate their efforts towards more strategic and impactful tasks. The prevention of class collisions is a crucial feature, as it ensures a seamless schedule for both students and faculty, minimizing disruptions and enhancing the overall learning experience.

In summary, our project represents a step towards modernizing educational processes, enhancing organizational efficiency, and ultimately contributing to the overall advancement of the academic experience within colleges.