Design Document for OO Design for Case Study 1 (Course Scheduling)

1 Overview

The goal of this project is to develop a system for scheduling the courses in a computer science department, based on the input about classrooms, lecture times, and time preferences of the different instructors. Different conditions have to be satisfied by the final schedule and which have been specified in the SRS.

This document specifies the final system design for the system. It also gives some explanations on how the design evolved and why some design decisions were taken.

2 Class Diagram

We have discussed how the design of this case study evolved in the book. Here we just mention some of the key observations and design decisions that led to the final class diagram.

- From the problem specification, we identify the following classes: `TimeTable, Course, Room`, `LectureSlot`, `CToBeSched` (course to be scheduled), `InputFile_1`, and `InputFile_2`.
- `TimeTable` is an important class. It is an aggregation of many `TimeTableEntry`, each of which is a collection of a `Course`, a `Room` where the course is scheduled, and a `LectureSlot` in which the course is scheduled.
- From file 1, we get: `RoomDB, CourseDB`, and `SlotDB`, each of which is an aggregation of many members of `Room, Course`, and `Slot`, respectively.
- From file 2, we get a `TableOfCToBeSched`, which is an aggregation of many `CToBeSched`.
- For scheduling, courses were divided into four four different types: `PGwithPref, UGwithPref, PGwithoutPref`, and `UGwithoutPref`.
- We have a class `ConflictTable` into which different conflicts for the different time slots of different courses are stored, and from where they are later printed. This is a list of `ConflictTableEntry`.
- Implementation concerns also added a template class `List`, an internal class `PGReserve`,

The final class diagram after the design is shown in Figure 1.

In this design, mapping between the modules in the design (which essentially is the module view) and the component view given in the architecture document is not straightforward. Most of the classes are used by all the three components. However, the processing of the system is as per the architecture. Let us look at the main() function in this design:

```
main (int argc, char *argv[]) {
   InputFile_1 in1;
   InputFile_2 in2;
   TableOfCtoBeSched tbl;
   CourseDB cDB;
   RoomDB rDB;
   SlotDB sDB;
   TimeTable *timeTable;
   ConflictTable *conflictTable;

   // From main, first the databases are built by in1.build_CRS_DBs()
   // Then table from file 2 is built by in2.buildCtoBeSched( )
   // Rooms are then sorted, and scheduling is done by tbl.scheduleAll( )
   // Timetable and conflict table are then printed.
}
```

As this shows, the processing is still done in three stages, each stage passing the information to the next stage. The pipes are really the parameters and objects being passed between the various objects created when the system executes.

3 Design Specification

Here we specify the design shown in Figure 1 precisely. We use C++ structures to specify the design.

```
// This is the specification for the Object-Oriented Design for the Case Study.
// It gives all the major classes and most of the major data members and
// operations for these classes. The major parameters of the various operations
// are also given, which shows how objects are being made visible. All the
// major object declarations are also given. Comments are added only where
// they are needed over and above the discussions above.

class Course {
  private:
     char *courseID;
  public:
     *tellCourseID ();
     isAValidID ();  // To test whether courseID is invalid
     bool isAPGCourse ();
     void  setAttr (char *);
};

class Room  {
  private:
     int roomNo;
     int capacity;
  public:
      Room (int num, int cap);
      bool tellAttrib (int &num, int &cap);
      bool  areAttribValid (); // Atrribute validation
};
```

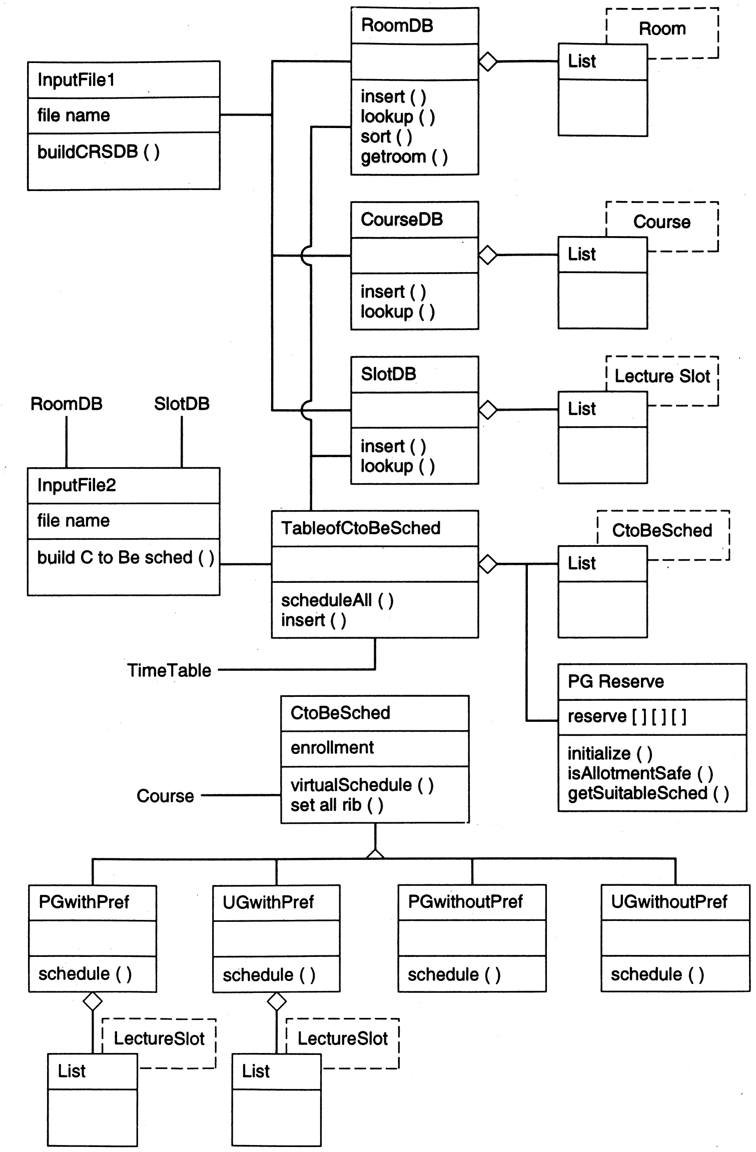

Figure 1: Final class diagram for the case study

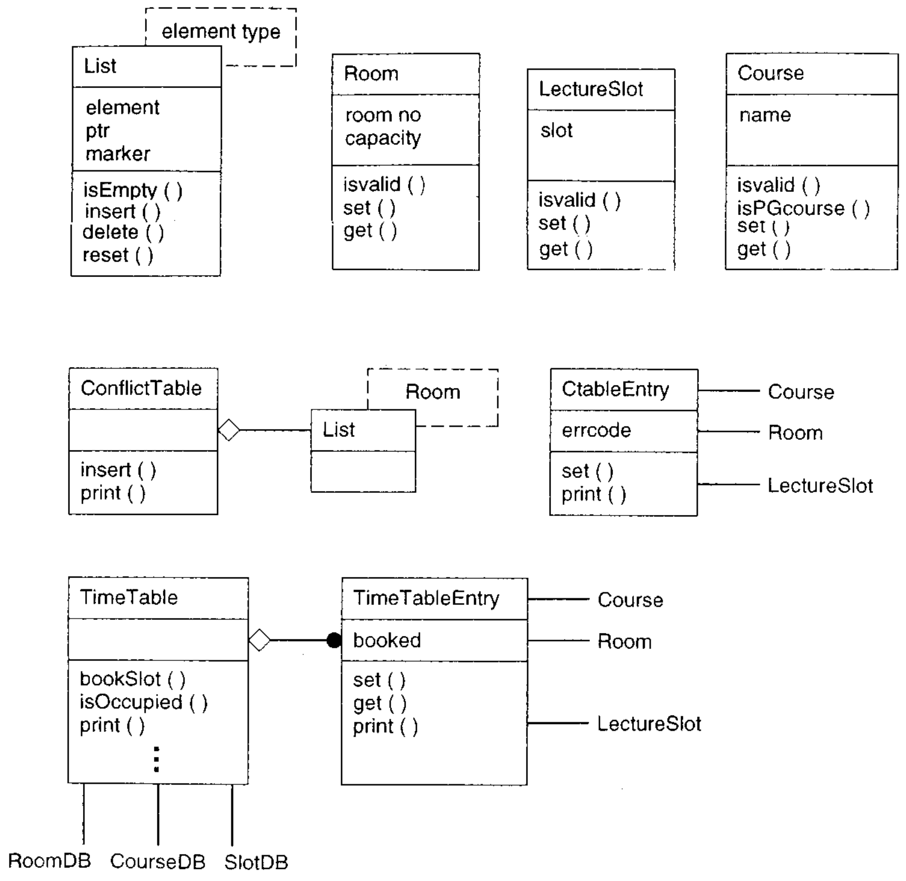

Figure 1: Final class diagram for the case study (cont.)

```
class LectureSlot {
  private:
    char  *slotID;
  public:
    char  *tellSlotID ();
    bool  isAValidSlot ();
    void  setAttr (char *id);
};

//  Generic List Abstraction
template  <class Element_t>
class List {
  private:
      Element_t listElem[ MAX_ELEMENTS ];
      int currentPtr;
      int stateMarker;
  public:
      bool isListEmpty ();
      bool insertElement (Element_t  element);
```

```
          // Next two operations needed to traverse the list
     void reset (); // Reset the CurrentPtr, if you want to scan from start
     bool getNextElement (Element_t  &element);
         // In tables we will represent a room, course, etc. as an index in its DB.
         // Operations for random access in the list are therefore needed.
     bool getIthElement (int, Element_t &);
     bool setIthElement (int, Element_t);
};

// Abstraction  for  room database
class RoomDB {
  private:
    List<Room *> roomDB;
  public:
    bool insert (Room *);
    void sortRooms (); // Sort on capacity - makes looking for a room easier
    bool getSuitableRoom (enrol); // get smallest room that fits
};

// Abstraction  for  course database
class CourseDB {
 private:
   List<Course  *>  courseDB;
 public:
    bool insert (Course  *);
    bool lookUp (int &, Course &);  // Search the DB and return the index
};

// Abstraction  for  class database
class SlotDB {
  private:
    List<LectureSlot *> slotDB;
  public:
    bool insert (LectureSlot *);
    bool lookUp (int &, LectureSlot &);  // Return index of the lecture slot
};

// Object for File 1
class InputFile_1 {
  private:
     ifstream inFile;
     char *fileName;
         // Parsing functions
     bool semicolonExists (char *); // TRUE if semicolon is present in the line
     bool buildRoomDB (RoomDB  &);  // Parse file until roomDB is built
     bool build_CS_DB (CourseDB &, SlotDB &); // Build course and slot DBs
     char skipOverWhiteSpaces ();
     void getNextRoomEntry (int &, int &, int &);
     void  getNextToken (int &, char *&);
  public:
     build_CRS_DBs (char *fName,CourseDB &cDB, RoomDB &rDB, SlotDB &sDB);
};

// A course that has to be scheduled - from file 2. Forms the base class
class CtoBeSched {
  protected:
    int enrollment;
    int coursePtr;
  public:
    virtual void getScheduled (TimeTable *&, RoomDB *, SlotDB  *,
           ConflictTable *&, TableOfCtoBeSched  *&, PGReserve *&);
    int setAttr (int enrollment, int course);
    bool tellAttr (int &cptr, int &enroll);
};


class PGwithPref: public CtoBeSched {
  private:
    List<int> prefList; // List of preferences
    void getScheduled (TimeTable *&, RoomDB *, SlotDB  *,
           ConflictTable *&, TableOfCtoBeSched  *&, PGReserve *&);
  public:
    bool insertPref (int); // into prefList
};

class UGwithPref: public CtoBeSched {
  private:
     List<int> prefList; // List of preferences
  public:
    void getScheduled (TimeTable *&, RoomDB *, SlotDB  *,
           ConflictTable *&, TableOfCtoBeSched  *&, PGReserve *&);
    bool insertPref (int); // into prefList
};

class PGwithoutPref: public CtoBeSched {
  public:
    void getScheduled (TimeTable *&, RoomDB *, SlotDB  *,
           ConflictTable *&, TableOfCtoBeSched  *&, PGReserve *&);
};

class UGwithoutPref: public CtoBeSched {
  public:
    void getScheduled (TimeTable *&, RoomDB *, SlotDB  *,
           ConflictTable *&, TableOfCtoBeSched  *&, PGReserve *&);
};

class PGReserve {
  private:
     int reserve[ MAX_COURSES ] [MAX_SLOTS] [MAX_ROOMS];

  public:
      void initialize (List<CtoBeSched *> *,SlotDB *,RoomDB *,TimeTable *);
      bool isAllotmentSafe (List<CtoBeSched *> *, int room, int slot);
           // given the PG courses without prefs, is this allotment safe?
      bool getSuitableSchedule (TimeTable *&,List<CtoBeSched *> *,int cPtr, int rNum,
           int &slot, int &room,bool &); // To schedule PGwithoutPref courses
};

class TableOfCtoBeSched {  // Table of courses to be scheduled
  private:
     List<CtoBeSched *> table [NUM_OF_CATEGORIES];
     PGReserve  *pgReserve;  // Reservation chart for PGsansP
  public:
     bool scheduleAll (TimeTable *&, SlotDB *,RoomDB *, ConflictTable *&);
         // Sends getScheduled message to all courses
     bool insert (int, CtoBeSched *);
};

class  TimeTableEntry { // One entry in the timetable
  private:
     int coursePtr;  // Index into Course DB
     bool booked;  // Some other course booked it?
  public:
     void setAttr (int cptr, bool mark);
     bool isThisBooked ();
     int tellWho ();
     void printEntry (CourseDB *cDB);
};

class TimeTable {
  private:
    int PGOccupied[ MAX_SLOTS ];
    TimeTableEntry table[ MAX_ROOMS ][ MAX_SLOTS ];
  public:
    bool bookASlot (int, int, int); // Set table[i][j] = ptr ;
    bool isColumnEmpty (int colIndex);
    bool setPGOccupied (int slotIndex,int coursePtr);
    int PGwho (int  coursePtr);
    int who (int roomPtr, int slotPtr);
    void printTimeTable (CourseDB *, RoomDB *, SlotDB *);
    bool isAlreadyOccupied (int roomPtr, int slotPtr);
    huntForSuitableRoom (int, int, RoomDB *,  int &);
};

class ConflictTable_Entry {
  private:
     int errorCode;
     int coursePtr;
     int conflictCourse;
     int preference;
     int roomPtr;
 public:
     void setAttr (int E, int c,int cc,int p,int r);
     void printAttr (CourseDB *,SlotDB *,RoomDB *);
};

class  ConflictTable {
  private:
     List< ConflictTable_Entry *>  table;
  public:
      bool insertEntry (int, int, int, int, int);
      void printTable (CourseDB *,SlotDB *,RoomDB *);
};

class InputFile_2 {
  private:
     char *fileName;
         // Parsing functions
     char skipOverWhiteSpaces ();
     char *getNextToken ();
     char *getNextPref ();
     bool prefGiven ();
  public:
     bool buildCtoBeSched (char *fName, TableOfCtoBeSched &, CourseDB *, SlotDB *);
};


main (int argc, char *argv[]) {
   InputFile_1 in1;
   InputFile_2 in2;
   TableOfCtoBeSched tbl;
   CourseDB cDB;
   RoomDB rDB;
   SlotDB sDB;
   TimeTable *timeTable;
   ConflictTable *conflictTable;

   // From main, first the databases are built by in1.build_CRS_DBs()
   // Then table from file 2 is built by in2.buildCtoBeSched( )
   // Rooms are then sorted, and scheduling is done by tbl.scheduleAll( )
   // Timetable and conflict table are then printed.
}
```