University of Toronto
csc228S— File Structures and Data Management, Winter 2002

Project — Restaurant finder

Deadline A :	*Plan (4%). Beginning of your tutorial section, Friday March 15th, 2002.*
Deadline B :	*Presentation (3%). Beginning of your tutorial section, TBA.*
Deadline C :	*Report (15%). Beginning of your lecture* **Monday** *April 8th, 2002.*

Introduction

People like big cities, which offer something for everyone. But when it comes to find those extraordinary places people usually need the help of a seasoned veteran of the streets, or a guidebook which catalogues the various attractions. Restaurants are no exception; With so many choices, so many styles, so many budgets and so little time, people need help finding the appropriate restaurant for their situation.

Your Job

You are part of a team which is developing a new pocket electronic restaurant guidebook. Your job is to design and implement the basic functionality of the tool. Once you have done this you plan to hire a new grad to create a flashy user interface but you realize that the success of your product first depends on you. You are to implement the data structures and associated code that will provide at least the following functionally:

Locate Address Given an address (a street name and a number) return its coordinate[1] relative to the system's city map.

Lookup Restaurant Given the name and address of a restaurant, report the other information you have on file for it (cuisine, service etc.)

List Restaurants Given a set of constraints (cuisine, service and/or price) list the restaurants (i.e. name and address) which satisfy the constraints.

List Closest Restaurant(s) Given a coordinate relative to the system's city map (returned from Locate Address) return the closest restaurant(s) (i.e. its name, address and distance from the given coordinate) that meets your requirements of cuisine/service/price.

Add a Restaurant Insert into the system the information about a restaurant.

Add a Street Insert into the system the information about a street.

[1]This will be clarified later.

You will need to write a small program to test your code. Do not build a fancy interface just provide enough to demonstrate and test your code.

System Description and Implementation

There are many elements that participate in this system.

- Restaurant. A restaurant is described by its **Name**, unique **Address**, **Cuisine**(e.g. Asian, Indian, Italian), **Price Range** (e.g. 20$ per person) and **Quality of Service** (e.g. 1-5 scale). Restaurants are to be stored in a file indexed by Name/Cuisine/Price-Range and Quality of Service. At least one of the indexes should be implemented as either a B tree or a B+ tree. Restaurants will also be indexed by another structure, the Map, which will facilitate finding the closest restaurant to a coordinate.
- Street. A street is described by its **Name** and a list of **Corners**, which constitute the trajectory of the street as straight lines connecting corners. Streets should be stored in a hash table, where the key is the street name.
- Corner. A corner is associated with one street only and is described by its **Number** (as in a house's number at certain street, e.g. **22** Sussex Dr.) and its **Coordinate**.
- Coordinate. A coordinate is a pair of numbers (x, y) which contain the location of a Corner or a Restaurant. Coordinates can be expressed in the unit of your choice (such as miles or meters) and they can be represented either by integer or real values.
- Map. A map is the 2D table that results of subdividing the area where coordinates can exist into a rectangular grid. Every cell in that grid will correspond to an entry in the table, where pointers to the restaurants with coordinates falling in the cell exist. This set of cells is for all purposes a hashing table where coordinates are mapped to a table entry. Collisions occur when more than one coordinate get mapped to the same cell[2].

  Is is up to you to make decisions of the size of the table and how you will deal with collisions. Given a coordinate, a map facilitates the search for the closest restaurant as outlined in the following algorithm:

  1. given coordinate (x,y), find cell (m,n) that contains it.
  2. look for matching restaurants in cell (m,n) and pick closest(s) one(s).[3]
  3. if there are no matching restaurants, look for them in the neighbor cells (there are many ways of searching neighboring cells).

[2] At first you may be confused about the notion of coordinates and cells. You can think of the coordinates as the real location of the item in meters or miles or some distance measure of your choice. The cells correspond to the grid lines added to maps to make them easier to use. Even if coordinates are integers, cells are not the same. Cells are much larger and hold many coordinates. Cells are not necessarily all the same size.

[3] Of course this doesn't always pick the absolute closest match. If a coordinate is very near the edge of that cell a closer match might be in the neighboring cell. It is ok to ignore this and consider restaurants in the same cell to be closer.

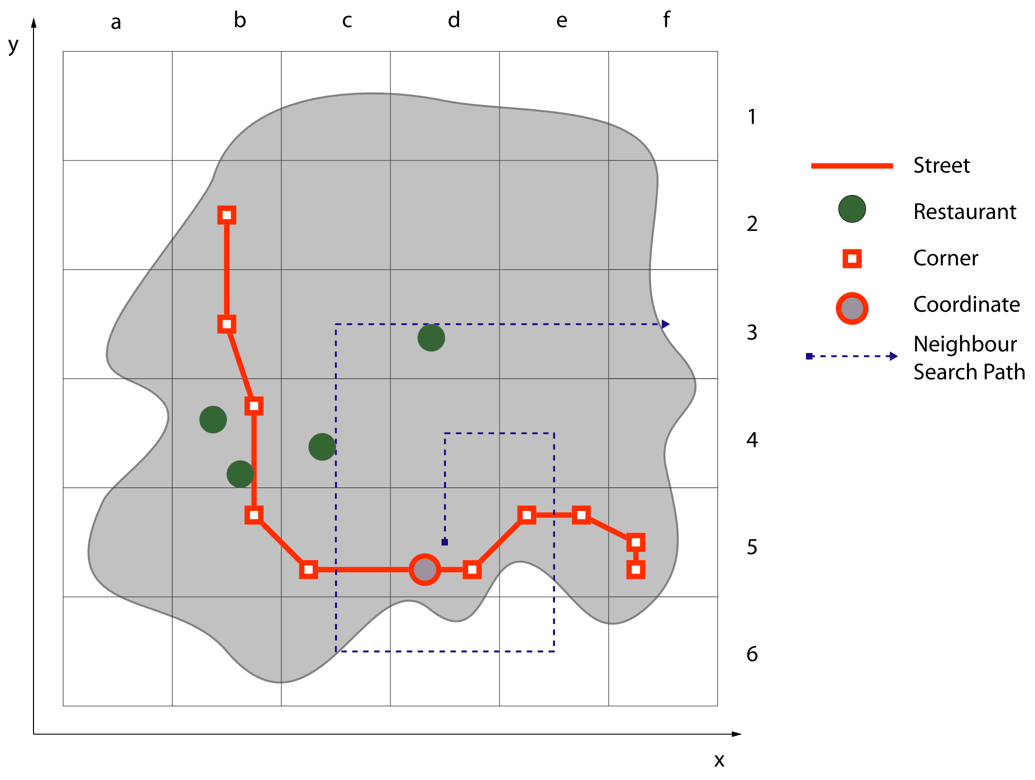

Figure 1: The elements of the system.

Extra Work for Groups of 3 people

Groups of three students are expected to implement the minimum functionality described above and then do some additional work. Here are some possibilities. You should do at least two of these extra items.

- Make the map hashing table extendable. In this case, make the grid denser when the number of restaurants in a cell grows over a pre-determined threshold.
- Allow the deletion of restaurants.
- Allow the rating of restaurants to be modified.
- add the function **Locate Intersection** which given two street names returns the coordinate of their intersection relative to the system's city map.
- Other extensions you have cleared with your instructor.

Working in a group

You will complete this project in a group of two or three people. Groups of three will be expected to complete a more ambitious project than groups of two, as described above.

Be sure to reread sections 5 and 6.2–6.3 of the 228 Course Guide; they address how to set up and organize your group, and how your group work will be marked.

I hope that working in a group will be one of the most valuable experiences you have in this course. However, you will have to make an extra effort to deal with the dynamics of a group if you wish it to be a valuable *positive* experience. :-)

If your group is having difficulties working effectively, don't sweep it under the rug and hope things will improve; they probably won't. Talk about it, and if that doesn't work, see your professor as soon as possible.

Using the code from your textbook

The appendices of your textbook contain c++ classes that partially implement B-trees. You are welcome to use this code by adapting it as necessary for your project. Because many of the B-tree routines are already implemented, using this code may save you a considerable amount of time. However, there are some hurdles you will have to overcome in order to use the code:

- The code uses objects and methods from other classes presented in earlier appendices. These include SimpleIndex, IOBuffer, Record and possibly others. In order to understand the B-tree code, you will need to understand these other classes.
- It does not implement deletion. If you need deletion you will have to write it yourself.
- It does not quite implement a B^+-tree. By B^+-tree we mean a tree in which the actual records are stored only in the leaves and there is a linked list joining the leaves together. The book implements B^+-trees in which the records are only at the leaves, but does not contain the pointers to join the leaves into a linked list. You would have to modify the code to add and maintain these pointers.
- There may be bugs in the code. We have not used it before.

In the real world, programmers often have to make decisions about when to use and adapt legacy code and when to create routines from scratch. You need to consider this decision as a group as you plan your project.

The phases of the project

Your project plan

Your project plan should include the following information:

- **Group**
  The name of your group, your group members their individual TAs, and the TA to whom you are submitting your project. Write this on the project cover sheet, available on the course web site.

Remember that you needn't all be in the same tutorial; in fact, it is an advantage if you are not, because your group will have the benefit of getting guidance from more than one tutor.

- **Domain**
  What properties of the domain (an electronic restaurant finder) affect your design decisions (*e.g.*, "There are far more of this data than that data, so ..." or "Operation such-and-such will occur very often, so ..."). If you don't think carefully about the domain, the rest of your project may not make sense.

- **Data**
  An outline of what data you plan to store, which will include what we've listed. Describe any relationships across data sets (*e.g.*, " The address in the restaurant file relates to the street name in the streets file and the street number in the corners. ").

- **File Structures**
  For each set of data, a description of the file structure. Will it be hashed, or indexed? What sort of collision resolution scheme will be used? Will it be linked to any other file? Will the restaurant file be threaded or will you use accession lists?

  Here you are describing the structure of the file, not the details of how you'll lay it out (variable length or fixed length records, etc.). Think carefully about your decisions — they will affect what operations will be possible and efficient.

- **Operations**
  A list of the operations you plan to handle, with a brief but precise specification of what each one will do. Again, this will include the operations we've required.

- **Special Features**.
  Any special features you plan to include in your project.

- **Plan of Attack and Schedule**
  A detailed breakdown of tasks. For each task, specify who will focus on it, its planned completion date, and whether it is core or a fancy extra that you hope to have time for.

  You can think of the work to be done as a two-dimensional table, with components of the program down the rows and phases of development (settling the specifications, design, coding, testing, report writing) across the columns. It is acceptable for group members to each focus on different aspects; however, it is **not acceptable** to divide things up simply by row or column – for example to have one team member do everything to do with hashing, or everything to do with writing the report.

  The plan of attack and schedule is probably the most important part of your Project Plan. Do not neglect it.

You will be allowed to change your mind later about anything in your project plan, but of course, the better thought out your plan is, the fewer changes will be necessary. Be prepared also that your TA may find flaws in your plan, and that this may lead you to make changes part way through the project. Accommodating changes will be much easier if you have designed your code well.

I would rather you get a modest project done really nicely (including robust code, thorough testing, and well-written report), than over-extend yourself and end up with a very ambitious project done poorly. And yes, a well done but modest project will get a better mark.

A wise strategy would be to plan to complete a modest project, and then if you can, to improve or extend it. Explain this strategy in your Project Plan (including exactly how you hope to extend your project). In order to work this way, you *must* design your code well, so that pieces can be plugged in and out.

Your group presentation

During tutorials in late March and early April, each group will make a brief (roughly 5 minute) presentation of their project, discussing their overall design strategy and special features of their system. In most cases, the group should select one member to do the actual speaking, but every member should participate in planning the presentation.

We will say more about the presentations in an upcoming tutorial.

Your final report

Your final report will be significantly longer than the reports for assignments 1B and 2B, but shouldn't be longer than about 10 pages.

Include the usual sections of a report, as well as all the sections listed under the project plan above. It is okay to reuse parts of what you handed in for your plan, but you should to update it to reflect changes to your plans, and you should expand upon the design decisions you made, alternatives your rejected, tradeoffs, etc.

In addition, you must submit the following sections:

- **Who did what.**
  Each group member must submit their own brief assessment of who did which work on the project. Even if your group writes this "who did what" statement together, each person must hand in their own copy. These statements should be brief — substantially less than a page long.
- **Teamwork.**
  A brief description of how you handled the teamwork, beyond who did what. Was one person in charge of everything? Was a different person in charge of each of several sub-tasks? Or was it a more democratic team structure? How did you run your team meetings? What problems arose in your team? How did you solve them?

Approaching the project

You will save yourself a great deal of time and effort overall if you take the time to do a good job in the early stages of the project. You should have a clear specification of *what* you plan to do before beginning to do it. And you should design your code carefully from the beginning, rather than think that you will reverse-engineer a good design after it's done. Don't forget the lessons learned in CSC148 and beyond about modularity, abstraction, and information hiding.

You may find it necessary to reduce your ambitions, and you may learn things later that lead you to change aspects of your system. For both reasons, it is important to begin by isolating the essential operations of the system and designing a modular structure to accommodate later changes. This is not a difficult task, but you must remember not to rush straight into coding. If, on the other hand, you find that you have quickly succeeded in building a basic working system, you may wish to add extra features to your program.

It will be natural as you plan your system to think about how to describe it in your report. The report should not be written at the end of the project. Rather, you should write an outline at the beginning, and continue to expand it as you proceed, until by the end of coding and testing, the report is almost complete.

You should also begin work on your testing strategy very early in the project. If you leave testing to the last minute, your group will likely find itself in trouble. Try to build up the system from a simple skeleton to a fully functional package in gradual stages, testing various components as you proceed. With the confidence of having something simple working soon, you can go on to grow a bigger project successfully.

To which TA should you hand in your project?

This section applies only to St George students.

If all members of your group, or the majority of members, have the same TA, hand all phases of your project in to him or her. Otherwise, you may pick which one of your TAs to hand your project in to.

After the first phase (the Project Plan) is due, I may redistribute some groups to a different TA, to even out their marking workload.