CIS 22C Data Structures
Team Project – Part 2

Suggested Data Structures

The system's data structure is to contain a hashed list array and two binary search trees. The input file should contain at least 25 records.

As the records are read, they are placed in dynamic memory. The memory location (address) is then inserted into a hashed array and into the two BSTs (shared data). The primary key (a unique key, like the isbn of a book) is the key of the hash table and one of the BSTs. The secondary key (like the title of a book, that is not unique) is the key of the other BST.

Test your program with two hash functions: a "bad" hash function, that gives lots of collisions, and a "good" hash function, that minimizes the number of collisions. The key of the hash table is the unique key, and it must be a string (this is one of the project requirements). You may use any of the hash functions discussed in class or explained in your text, or you could create new hash functions.

Collisions will be resolved using a variation of the linear probe method described below. In a `linear probe` when data cannot be stored in the home address, we resolve the collision by adding 1 to the current address. However, if this address is also filled, we add another 1 to the address and so on. As a variation of the linear probe, we can add 1, subtract 2, add 3, subtract 4, and so forth until we locate an empty element.

Example: Assume the size of the hash table is 10, and the hash function is modulo-division.
Keys: 25, 35, 95, 15, 85 // all synonyms

0	1	2	3	4	5	6	7	8	9
			85	95	25	35	15		

25 % 10 => 5
35 % 10 => 5, 5 + 1 = 6
95 % 10 => 5, 5 + 1 = 6, 6 – 2 = 4
15 % 10 = > 5, 5 + 1 = 6, 6 – 2 = 4, 4 + 3 = 7
85 % 10 => 5, 5 + 1 = 6, 6 – 2 = 4, 4 + 3 = 7, 7 – 4 = 3

Draw a Data Structure Diagram to show the hash table and the two BSTs (make sure to show how data are shared). Update this diagram depending on the variation of the project you have (i.e. number of students in your team).

Suggested Team Assignments

Each member of the team must write at least one unit of code as outlined below. If the team is small, team members may have to write multiple units.

> **Unit 1:** Team Coordinator: Data structure design coordination, main(), Menu and other related functions, such as Insert Manager, Delete Manager, Integration, Testing, Project presentation coordination, Weekly reports submission.
> **Unit 2:** BST Algorithms: Insert, Delete, Print indented trees, Search, etc.
> **Unit 3:** Hash list algorithms: Hash functions, Collision resolution functions, Insert, Delete, Search, Statistics (load factor, number of collisions, etc.). The hash table must be a dynamically allocated array.
> **Unit 4:** Screen Output: Primary Key Search Manager (prompt the user to enter a key, call search, then display the search results or a message if not found), Secondary Key Search Manager (if found display all matches), List Manager: List unsorted data (in hash table sequence), List all data sorted by the primary key, List all data sorted by the secondary key. *Undo Delete – only for teams of 4 or 5 students: The user can undo the delete in the reverse order of the delete sequence. When the user selects "Save to file", the undo stack is cleaned out (no undo possible unless more delete occurs first). Add one more option to the menu: "Undo delete".*
> **Unit 5:** File I/O: Determine Hash Size (count the lines in the input file, multiply it by 2, and choose the next prime number). Read data from the input file and insert them into the hash table and the two BSTs, Save to file, Re-hashing*: – only for teams of 4 or 5 students: when load factor is 75%, allocate another hash table (2 * actual size, and choose the next prime number), then traverse the hash table and hash each item into the new table using the same hash function and collision resolution method, and delete the old hash table)*

Detailed Suggested Team Assignments: are given based on the number of students in a team. It is intended for guidance only and may be changed by the team with the approval of the instructor. Note, however, that regardless of how the team assignments are determined, they must be clearly stated in the documentation.

1	**2**	**1**	**2**	**3**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**5**	**Assignment**
X		X			X				X					**Unit 1:** Team Coordinator
	X			X			X				X			**Unit 2:** BST Algorithms
X			X			X				X				**Unit 3:** Hash list algorithms
	X	X						X					X	**Unit 4:** Screen Output
	X			X			X					X		**Unit 5:** File I/O
X			X	X	X	X	X	X	X	X	X	X	X	**Test Plan:** Options and data so that anyone could use it to demonstrate the project. **Presentation Outline:** Activity, duration, etc.
X				X	X		X		X		X		X	**Project Documentation:** (see below)

For instance, for a team of two: **Student#1** will be responsible for Unit 1, Unit 3, Test Plan and Presentation Outline, and Project Documentation. **Student#2** will be responsible for Unit 2, Unit 4, and Unit 5.

Documentation

The team leader will create a folder named 22C_Team_No_x, compress it and submit the .zip file (one submission per team). The 22C_Team_No_x will include two sub-folders as described below.

First sub-folder, named **program_docs**:

1. Source Files and Header File(s). For each programmer, clearly indicate the assignment on the project. Each student's documentation is to be organized as shown below.
    a. Description - a short description of the purpose of this part of the project. For example, the documentation for the BST functions should describe the role of the BST in the application.
    b. Individual source/header files. It should contain only the functions used in the consolidated project; do not include the unit test driver code.
2. Input Data File
3. Demonstration Test Plan (final version) - should contain enough detail (options and data) so that anyone could use it to demonstrate the project. The test plan must also demonstrate collision resolution; that is it must contain at least one insertion that is a synonym.
4. Output (based on the demonstration test plan)
5. Output file
6. Executable version of the project (school computers, including project files)

Second folder, named **presentation**:

1. Presentation Outline (final version)
2. Power Point presentation, a Word document, or a .PDF file containing the following sections:
    a. Team number, Project title, team members' names (and picture(s) if you wish)
    b. Introduction – a short management summary describing the project application.
    c. Data Structure Design (diagram) with typical data, showing the relationships among the data
    d. Class Diagrams (UML)
    e. Structure Charts (A Structure Chart is a tree; you may use either the general tree representation, or the indented representation).
    f. A short description of each person's assignment.
    g. Hash functions (the actual code)
    h. Collision Resolution Method
    i. Anything else you consider relevant

Presentation

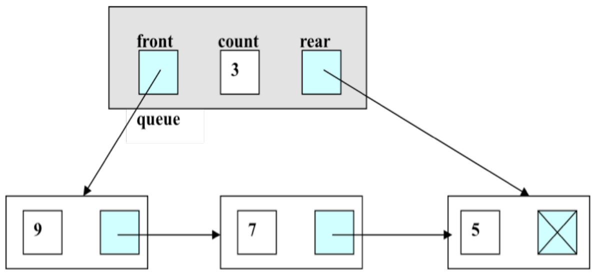
The presentation consists of three parts:

1. Project scenario and design. (Sections 2.a – 2.i listed above).
2. Complete demonstration of all major features of the program (all options on menu). The demonstration must include at least one synonym insertion, and deletion of the root (it should have two children).
3. Questions & Answers

FAQ

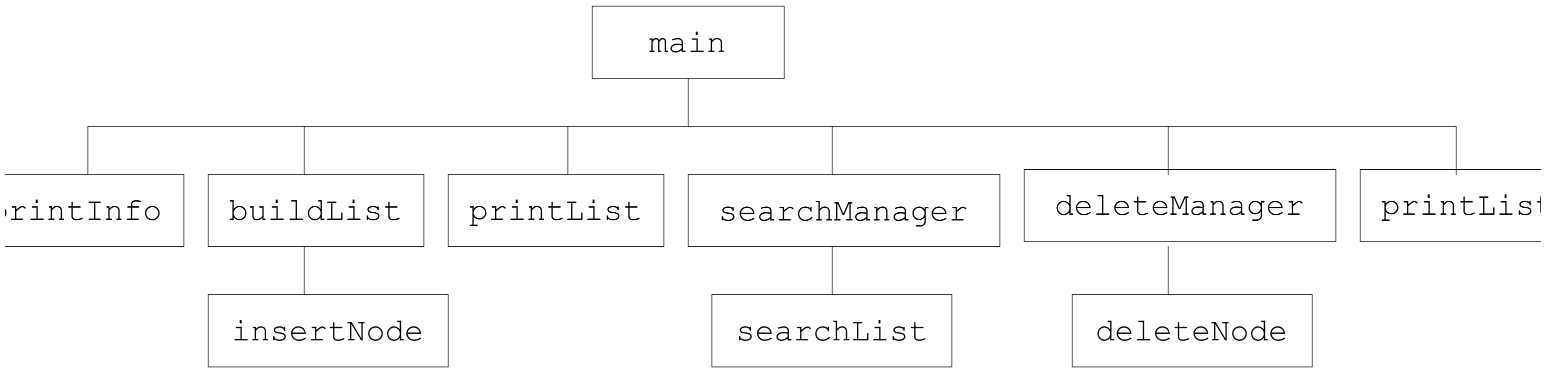
What is a Data Structure Diagram?

Let’s say that you are solving a problem that requires a queue. A data structure diagram for this project would show the queue:



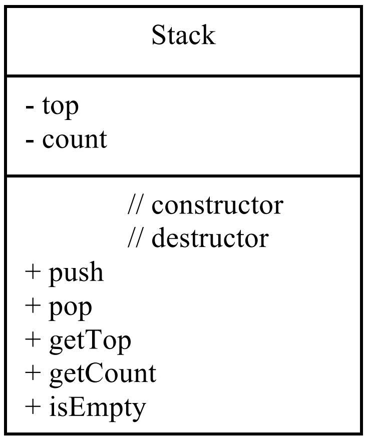
What is a Structure Chart?

A structure chart is a design tool that shows the structure of your program. It keeps track of the functions used in the program and also shows the calling called relationship between functions.

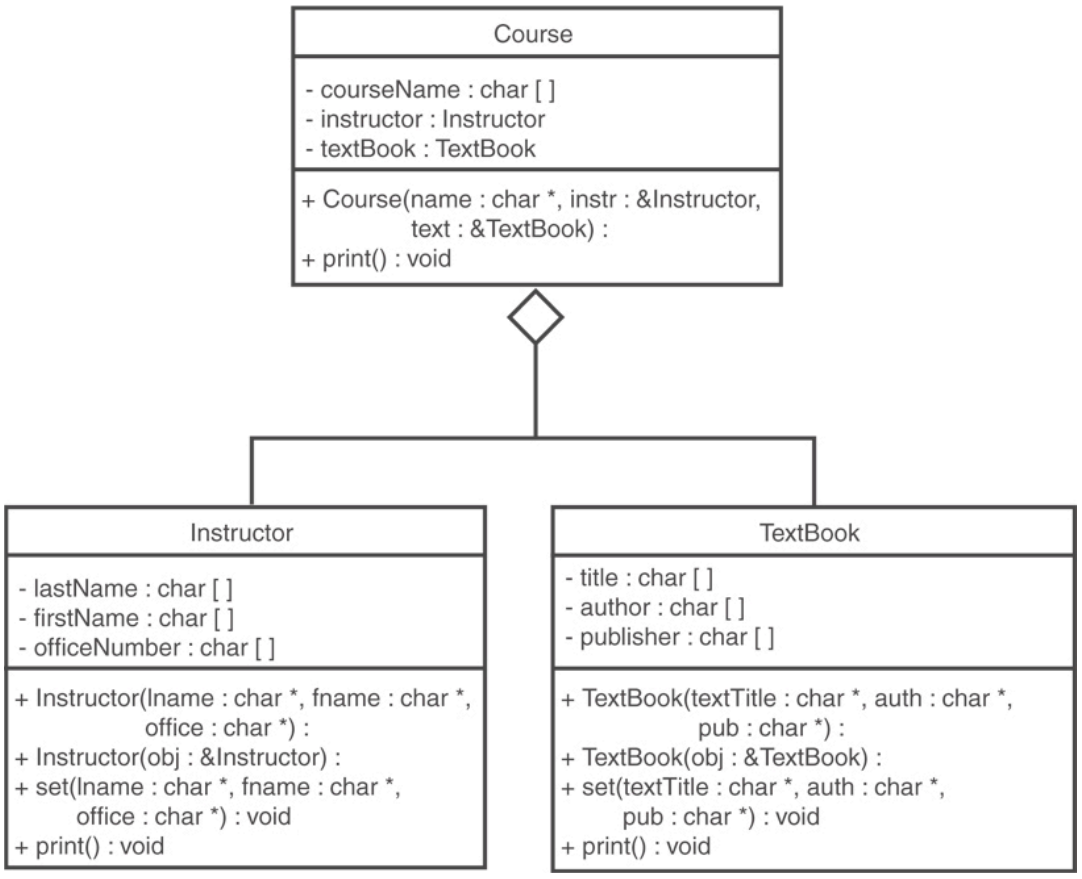


What is a Class Diagram?

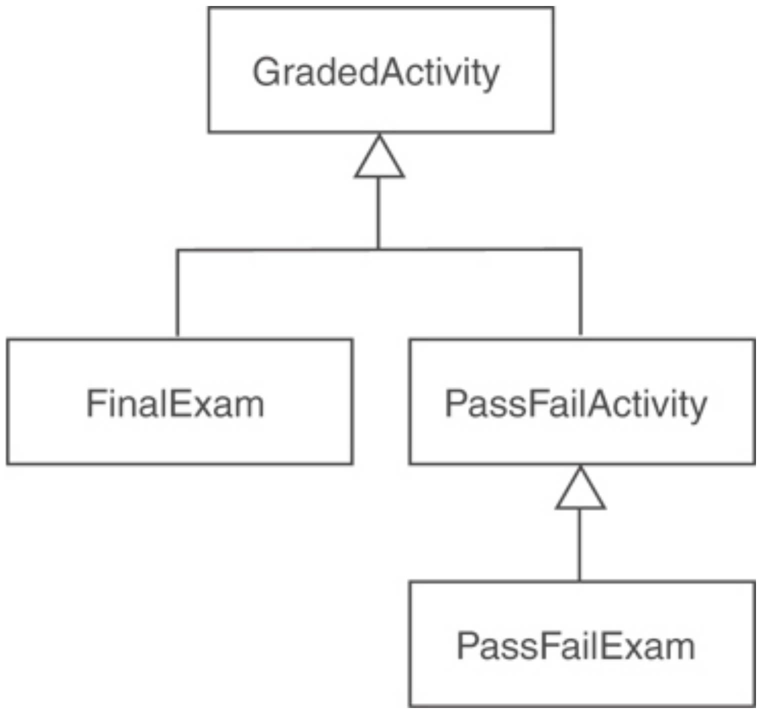
Here are some examples:



A Course “has” an “Instructor” and a “TextBook” (aggregation)


Course
- courseName : char [ ]
- instructor : Instructor
- textBook : TextBook
+ Course(name : char *, instr : &Instructor, text : &TextBook) :
+ print() : void
Instructor
- lastName : char [ ]
- firstName : char [ ]
- officeNumber : char [ ]
+ Instructor(lname : char *, fname : char *, office : char *) :
+ Instructor(obj : &Instructor) :
+ set(lname : char *, fname : char *, office : char *) : void
+ print() : void
TextBook
- title : char [ ]
- author : char [ ]
- publisher : char [ ]
+ TextBook(textTitle : char *, auth : char *, pub : char *) :
+ TextBook(obj : &TextBook) :
+ set(textTitle : char *, auth : char *, pub : char *) : void
+ print() : void

A “FinalExam” “is” a “GradedActivity” (derived classes)


GradedActivity
FinalExam
PassFailActivity
PassFailExam