The purpose of this assignment is to give you practice with command line arguments, inputting from files, using hash tables, and implementing hash tables using chaining. It will also give you practice with insertion order lists.

For this assignment you are to write a program which will input student registration information from two different files. One file contains student oriented information, and the other contains course oriented information. This information is to be stored in two separate "databases", one for students and one for courses. The program is then to use the information stored in both of these databases to generate schedules for each student, and to generate rosters for each course.

The names of the two files are to be obtained from the command line. The first command line argument is to be the name of the student information file, and the second is to be the course information file. If there are not the correct number of arguments, or if either of these files cannot be opened, the program is to print an appropriate error message and abort (calling `exit` (from *stdlib.h*) with a non-zero argument).

Students are uniquely recognized by their id. Each course has a unique course reference number (crn). The names of students and courses **should not** be considered to be unique.

The student file contains a number of sets of information, one set per student. Each set consists of

```
name: id numCourses crn crn ...
```

The name starts at the beginning of a line and is terminated by a colon. The name may contain spaces. The id is a space delimited string. The number of courses the student is currently registered for is an integer, as are each of the course reference numbers (crns). There is a crn for each course the student is taking. The following is a sample student file (which happened to be called students.dat)

```
Flintstone, Fred: 000-12SA 3 121314 12333 12116
Flintstone, Wilma: 000-45SA 2 12332 12111
Glotz, Joe Q: 901-9984 3 12332 12116 12111
Rubble, Barney: 001-01SA 3 121314 12111 12116
```

The course file has one set of information per course section being taught (one course name may have several sections). Each course has the following information in the file:

```
name section crn credits numSeats numStudents id id ...
```

The course name is a space delimited string. The section, crn, credits, number of seats, and number of students are all integers. The ids are space delimited strings. There is an id for each student registered for the course. The following is a sample student file (which happened to be called courses.dat)

```
CS001 1 12111 1 10 3 000-45SA 901-9984 001-01SA
CS515 2 121314 4 45 2 000-12SA 001-01SA
CH302 1 12116 5 15 3 000-12SA 901-9984 001-01SA
```

```
MA111 1 12333 4 15 1 000-12SA
PH999 1 12999 2 10 0
PY000 3 12332 6 5 2 000-45SA 901-9984
```

You may assume that all the information in the files is in the correct format and is consistent. Students registered for a course will appear in the student file, and courses the students are taking will appear in the course file.

The program is to input all the information from the two files and insert it into the databases **prior** to printing schedules or rosters. Information from the course database is necessary to print student schedules, and information from the student database is needed to print course rosters.

The schedule for each student in the student file is to be printed. The schedules are to be printed in the same order as the students appear in the file. Each schedule is to be separated from the other schedules in the manner shown in the sample run below. Each schedule is to contain the student's name and their id. Each course they are registered for is to be printed, printing the course name, and section number in addition to the course reference number. The courses are to be printed in the same order as they appear in the student file. The total number of credits registered for is to be printed.

The roster for each course in the course file is to be printed. The rosters are to be printed in the same order as the courses appear in the file. Each roster is to be separated from the other rosters in the manner shown in the sample run below. Each roster is to contain the course name and section (separated by a hyphen), the number of students and the number of seats in the course (students/seats) and the number of credits. The names and ids of each of the students in the course is to be printed. The students are to be printed in the same order as they appear in the course file.

The following shows a run of the sample solution using the above files to load the databases.

```
blj(mithrandir): a.out students.dat courses.dat

==================================
Flintstone, Fred -- 000-12SA
   CS515 (2)  [121314]
   MA111 (1)  [12333]
   CH302 (1)  [12116]
total credits: 13
==================================

==================================
Flintstone, Wilma -- 000-45SA
   PY000 (3)  [12332]
   CS001 (1)  [12111]
total credits: 7
==================================
```

```
==================================
Glotz, Joe Q -- 901-9984
   PY000 (3)  [12332]
   CH302 (1)  [12116]
   CS001 (1)  [12111]
total credits: 12
==================================

==================================
Rubble, Barney -- 001-01SA
   CS515 (2)  [121314]
   CS001 (1)  [12111]
   CH302 (1)  [12116]
total credits: 10
==================================

**********************************
CS001-1 -- 12111 (3 / 10) {1 credits}
   Flintstone, Wilma [000-45SA]
   Glotz, Joe Q [901-9984]
   Rubble, Barney [001-01SA]
**********************************

**********************************
CS515-2 -- 121314 (2 / 45) {4 credits}
   Flintstone, Fred [000-12SA]
   Rubble, Barney [001-01SA]
**********************************

**********************************
CH302-1 -- 12116 (3 / 15) {5 credits}
   Flintstone, Fred [000-12SA]
   Glotz, Joe Q [901-9984]
   Rubble, Barney [001-01SA]
**********************************

**********************************
MA111-1 -- 12333 (1 / 15) {4 credits}
   Flintstone, Fred [000-12SA]
**********************************

**********************************
PH999-1 -- 12999 (0 / 10) {2 credits}
**********************************
```

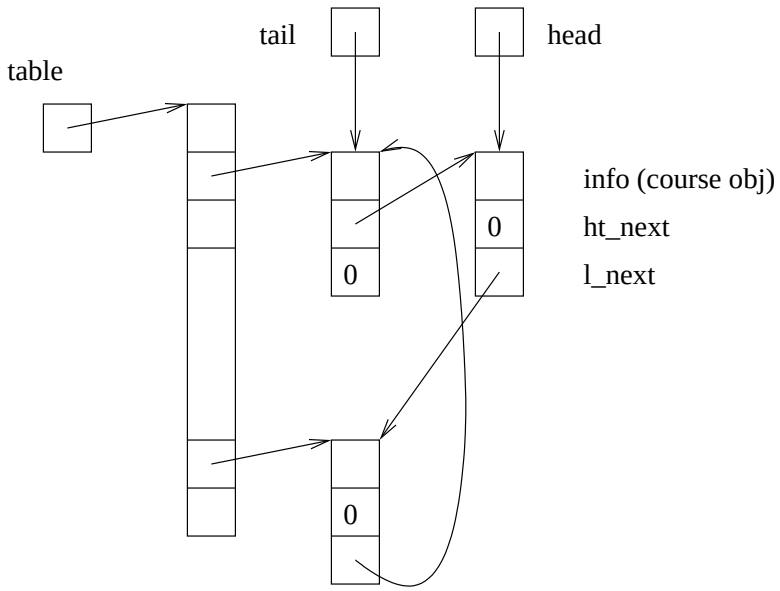
```
**********************************
PY000-3 -- 12332 (2 / 5) {6 credits}
   Flintstone, Wilma [000-45SA]
   Glotz, Joe Q [901-9984]
**********************************
blj(mithrandir):
```

Conceptually, there can be a very large number of students, and a large number of courses, which have a combined very large number of seats which can be filled. Thus, fast lookup is desired for both the students and the courses. The fastest would be a hash table. However, the order of the output for both the schedules and the rosters is specified to be the same as the input order, which is based on some unknown criteria. Although the files could be reread to get the order after the databases were built, file input may be at a snails pace when compared to the speed of memory and the CPU. Thus, although the reread method would work, it is far from an optimal solution.

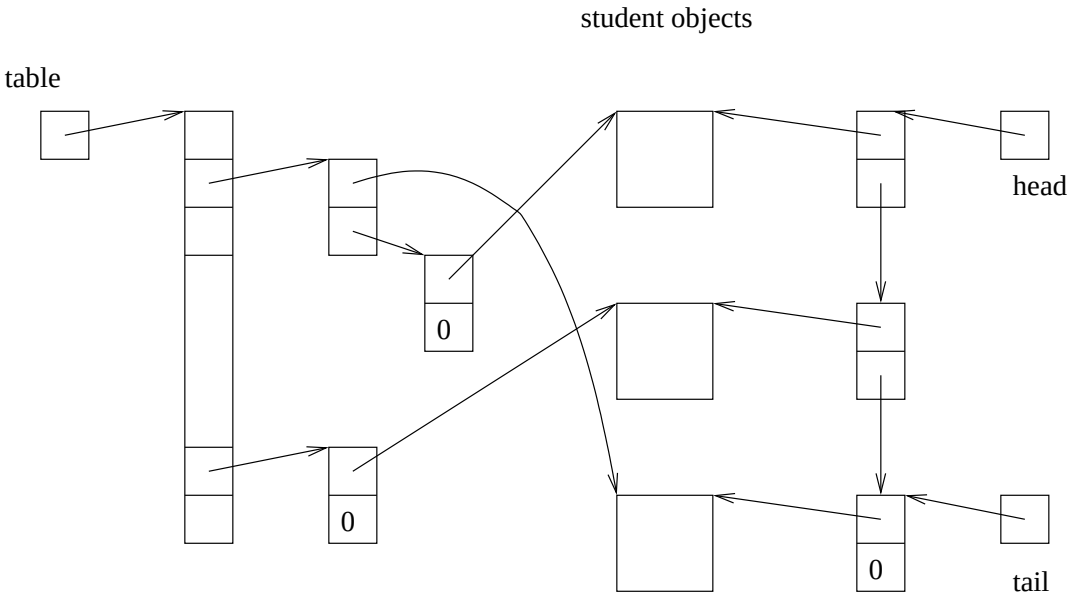
A way to solve the ordering problem would be to use an insertion order list, which then could be traversed in memory to provide the correct order. However, having the two separate storage structures could lead to storing the data twice.

The solution is to have a hybrid made up of the two mechanisms, sharing the information being stored. There are two general ways of approaching this sharing: (1) sharing the "elements" which contain the information; and (2) sharing the information between separate elements.

The course database uses the first approach. Each course is contained in exactly one "element". Each element has two pointers. One (`ht_next`) is used to chain the elements in the hash table buckets, and the other (`l_next`) is used to chain the elements in the insertion order list. A pictorial representation of this method is shown below.



The student database uses the second approach. Each student is dynamically allocated and a pointer to that element is placed in two separate elements. One of the elements is linked in the chain for a hash table bucket, and the other is linked into the insertion order list. The following is a pictorial representation using this method.



In both cases, finding the information is to be based on the hash table mechanism of the databases, while the order of producing the output is to be based on the insertion order list mechanism.

The implementation of the hash functions are up to you. You may assume that the data for the two classes (ids and course reference numbers) are reasonably random. The indexes returned should be reasonably distributed (for example, a hash function that always returned 0 would technically be a legal hash function, but it would not provide distributed indexes so would not be acceptable). You don't have to go to great effort to get a really good hash functions, but do avoid having really terrible ones.

A word of caution (prompted by almost two hours spent searching for a memory leak in my sample solution to this assignment – time wasted simply because I forgot to consider the following). In many respects, the input operation for a class is similar to assignment operator for that class. Both accept an **existing** object and put new values into the objects data members. Just as with assignment, if the existing object contains pointers to dynamically allocated memory, simply allocating new memory and linking the instance variables to it may cause a memory leak. The solution is the same as is done in the assignment operator – the input operation needs to perform the destructor code (unless the memory can be reused directly) to deallocate the old memory prior to allocating new memory. Be sure to do this only if something has been input, however.