Section Exercises 3: Files and Strings

1) String Implementation Redux. Write an implementation of the `Substring` function from the string library. You can use `strlen` from the ANSI string library, and of course `GetBlock`, but don't use any other strlib or ANSI string functions.

2) Quick Data Structure Question. You need to store a collection of a particular type of structure. In general, think about the differences between declaring a static array of the structures, a dynamic array of those structures, and a static array of pointers to those structures. In what cases would you want to use each of these approaches?

3) Structures. You've been hired to track the information about students and courses in a small college. Your data structure will record all the students at the college and all of the courses that are offered. You would like to package this information in one large data structure. You can rely on having a limit of `MAX_STUDENTS` and `MAX_COURSES` in this database.

For each course offered, you should track the following information:

- course's title
- number of units the course is worth
- the instructor of the course (instructor's name is sufficient)
- whether or not the class will be graded Pass/Fail
- what school the course is in (engineering, humanities, science, business, law, or medical)

For each student, you will track of the following information:

- student's name
- student ID number (this is just like a Stanford ID: 7 digits)
- the courses that the student is taking and whether they are taking each course for a grade. No student is allowed to take more than `STUDENT_MAX_COURSES`.

Design the data structure needed to store all of the information outlined above, and show all the type declarations involved.

Write a function `FindUnits`. This function should take the database and a student's name, and return the number of units the student is taking for a grade. You may assume that the list of students is sorted by student name, so you can take advantage of a quicker binary search.

4) Structure & Pointer Craziness. Consider the following declaration, which declares a type for a compact disc which is the title of the disc, as well as a dynamic array of the track titles:

```
typedef struct {
   string title;
   string *trackTitles;
   int numTracks;
} CDtype;
```

Assume that we have a function ReadCD that will read in one CD worth of info from a file: read the CD title, dynamically allocate the array of track titles and read them in. Also assume that, for some strange reason, that we have two CDs that are identical, except for the title of the first song. We try creating the two CD structs using the following code, but something is fishy here—what is it?

```
main()
{
   CDtype CD1, CD2;

   ReadCD(&CD1);     // read all info about CD1
   CD2 = CD1;        // make CD2 a copy of the CD1 struct
   CD2.trackTitles[0] = "Red Barchetta";     // change 1st song in
CD 2
}
```

5) Super Quick File Questions.

a) There's some dreadfully wrong in the fscanf call in the following function (assuming it gets passed a valid FILE pointer). What's wrong and how would you fix it?

```
string GetNameFromFile(FILE *infile)
{
   string result;
   fscanf(infile, "%s", result);
   return result;
}
```

b) Say you have an input file which is filled with info for baseball players (opening day was two days ago—hooray!) in the following form:

```
Fred McGriff
Hits: 40 AtBats: 110 Walks: 20
```

Write a function called GetBaseballPlayer which will return a pointer to a newly allocated and filled-in PlayerRec. This struct and the function prototype are defined as follows:

```
typedef struct {
   string name;
   int hits, atBats, walks;
} PlayerRec;
```

```
PlayerRec *GetBaseballPlayer(FILE *infile);
```

6) Files, files, files. You're going to read credit card billing information from a file that is organized like this:

```
Kermit the Frog                 name of credit card holder
12
Macy's                          proprietor for a charge
123.45                          amount of charge
Tower Records                   etc.
45.12
.... 10 more charges here ...
Bill Gates                      next record follows immediately after the
end
120                             Bill has a gold card...
Fry's Electronics
12345.60
... etc. ...
```

A blank line will follow the last client in the file to signify the end.

There can be up to `MAX_RECORDS` records in the file, but is likely to be a lot less, so you don't really want to pre-allocate any records. You will have a complete array of struct pointers, but you should use `New()` to create actual structures only when needed. There is no limit on the number of charges that may be billed for each person, so you should allocate that space dynamically.

Design the data structure for this database and write the functions necessary to read this information from the file. What would it take to free all of the memory allocated for the data structure you designed?