Assignment 6

Version 1.02 (last update: Nov. 10, 20:00)
Changes highlighted in yellow
Due date: Tue, Nov 17, 11:59 PM

In the following, the word **actor** refers to a person who acts regardless of gender.

Summary and Purpose

For this assignment, you will be creating a small database system, consisting of 5 programs. You will use hash tables and binary files to understand how a practical application might be constructed.

You program will make use of a C file of supporting functions called **`util.c`** and an accompanying header file called **`util.h`**.

When you program is compiled, it will be supplied with another C file containing the code for a hashing function called **`hashfn.c`** and an accompanying header file called **`hashfn.h`**. I will supply an example **`hashfn.c`**, but your program may be tested with a different **`hashfn.c`**.

Deliverables

You will be submitting:

1) A file called **`buildidx.c`** that will be compiled to create a program called **`buildidx`**.
2) A file called **`key2val.c`** that will be compiled to create a program called **`key2val`**.
3) A file called **`val2key.c`** that will be compiled to create a program called **`val2key`**.
4) A file called **`actors.c`** that will be compiled to create a program called **`actors`**.
5) A file called **`bacon1.c`** that will be compiled to create a program called **`bacon1`**.
6) A **`makefile`** that contains the instructions to compile your code.

You will submit all of your work via git to the School's gitlab server. (As per instructions in the labs.)

This is an individual assignment. Any evidence of code sharing will be investigated and, if appropriate adjudicated using the University's Academic Integrity rules.

The programs

`buildidx`
This program will accept 2 command line arguments (you must not read from the standard input, nor prompt the user). If a different number of command line arguments are provided, then your program shall:

```
fprintf( stderr, "Usage: %s filename.kv capacity\n", argv[0] );
```

Otherwise, the first argument should be interpreted as a **kv** file which can be accessed using the **read_key**, **read_val**, and **read_keyval** function provided in **util.c**. You may assume that the **kv** file exists.

The program should open the given file name for reading, along with two other files for reading and writing. All files are binary files.

The other two files will have the same name as the read file, except that instead of extension **".kv"** they will have extensions **".khs"** and **".vhs".**

The **khs** and **vhs** files will be initialized by calling the **write_empty** function from **util.c** with the command line parameter **capacity** passed as **capacity**.

Your program should read the contents of the **kv** file using **read_keyval** in the **util.c** file keeping track of the index of the key and value in the **kv** file (i.e. the first key, value pair red has index 0, the second key, value pair read has index 1, etc.). For each key and value read, your program should compute a hash value using the **hashfn** in **hashfn.c** (1 hash value for the key and a different hash value for the value).

It should then try to **write_index** the value of the index at the location of the key's hash value within the **khs** file, and at the location of the value's hash value within the **vhs** file. Of course, if the location in the hash file is already occupied (**read_index** returns an index value that's not -1), you will have to move to the next location. Naturally, once you hit the end of the file, you will need to start again from the beginning until you have found an empty location.

You may assume that the user will provide a command-line capacity parameter large enough to accommodate all the entries in the **kv** file.

Make sure to free any memory that you allocated and to close your files when you are done.

Your program should not print anything (other than the error condition, above).

key2val

This program will accept 2 command line arguments (you must ***not*** read from the standard input, nor prompt the user). If a different number of command line arguments are provided, then your program shall:

```
fprintf( stderr, "Usage: %s filename.kv 'search term'\n", argv[0] );
```

Otherwise, the first argument should be interpreted as a **kv** file which can be accessed using the **read_key**, **read_val**, and **read_keyval** function provided in **util.c**.

The program should open the given file name for reading, along with the corresponding **".khs"** file.

This program should perform a search for the search term, beginning at the index computed by the **hashfn** in **hashfn.c**. It should use that value as the hash in the **read_index** function and check the return value of **read_key** with the value generated by **read_index**. If the key value retrieved from the **kv** file matches the search term, then you can use **read_value** to retrieve the corresponding value from the **kv** file and print it followed by **'\n'**. You should only print the value retrieved by **read_value** and the **'\n'**, nothing else.

Of course, if the key value doesn't match the search term, you must continue through the hash table, looping back to zero when you hit the end of the file, until you have searched the entire table.

If you do not fine the 'search term', you will

```
printf( "NOT FOUND\n" );
```

val2key
This program is the exact analog of **key2val**, except that you use the **vhs** file to search for a value and return the corresponding key value. If you do not find the 'search term', you will

```
printf( "NOT FOUND\n" );
```

The Last 20%

actors
Write a program, actors, that takes a movie name as input (as the first command-line argument) and lists all the actors in it. Each actor name should be printed as represented in the **name.basics.kv** file, one per line. You may assume the movie name exists and is correctly entered.

bacon1
Play the game, "one degree of Kevin Bacon". This program will take one command-line argument which is the name of an actor. It will print one movie that includes both the actor named on the command line and Kevin Bacon. The program will print the title of the movie followed by a single new-line and nothing else. You may assume that the actor name on the command line has a Bacon number of 1.

You can write additional helper functions as necessary to make sure your code is modular, readable, and easy to modify.

Testing

You are responsible for testing your code to make sure that it works as required. The CourseLink web-site will contain some test programs to get you started. However, we will use a different set of test programs to grade your code, so you need to make sure that your code performs according to the instructions above by writing more test code.

Your assignment will be tested on the standard SoCS Virtualbox VM (http://socs.uoguelph.ca/SoCSVM.zip) which will be run using the Oracle Virtualbox software (https://www.virtualbox.org/wiki/Downloads). If you are developing in a different environment, you will need to allow yourself enough time to test and debug your code on the target machine. We will NOT test your code on YOUR machine/environment.

Full instructions for using the SoCS Virtualbox VM can be found at: https://wiki.socs.uoguelph.ca/students/socsvm.

Makefile

You will create a makefile that supports the following targets:

`all:`

this target should generate the executable files **`buildidx`**, **`key2val`**, **`val2key`**, and optionally **`actors`**, and **`bacon1`**.

`clean:`

this target should delete all **`*.o`** and executable files.

Each, **`*.c`** file should correspond to a rule in the **`makefile`** that generates a corresponding **`*.o`** file.

Additionally, there should be an individual rule for each executable file.

All compilations and linking must be done with the **`-Wall -pedantic -std=c99`** flags and compile and link **without any warnings or errors**.

Git

You must submit your **`.c`** files, and **`makefile`** using git to the School's git server. Only code submitted to the server will be graded. Do ***not*** e-mail your assignment to the instructor. We will only grade one submission; we will only grade the last submission that you make to the server and apply any late penalty based on the last submission. So once your code is complete and you have tested it and you are ready to have it graded make sure to commit and push all of your changes to the server, and then do not make any more changes to the A2 files on the server.

Academic Integrity

Throughout the entire time that you are working on this assignment. You must not look at another student's code, now allow your code to be accessible to any other student. You can share additional test cases (beyond those supplied by the instructor) or discuss what the correct outputs of the test programs should be, but do not share ANY code with your classmates.

Also, do your own work, do not hire someone to do the work for you.

Grading Rubric

buildidx	8
key2val	2
val2key	2
style	2
makefile	2
actors	2
bacon1	2
Total	20

Ask Questions

The instructions above are intended to be as complete and clear as possible. However, it is YOUR responsibility to resolve any ambiguities or confusion about the instructions by asking questions in class, via the discussion forums, or by e-mailing the course e-mail.