Cosc 242 Assignment 1

Due: 4pm Friday August 27th 2021

Overview

In this assignment you will expand and modify code written during the labs. You will be using the hash table data structure from lab 10 and 11.

Group and individual work

Everyone in the class should form groups of three to work on this assignment. Please send an email to iain.hewson@otago.ac.nz with the names and University user codes of your group members (e.g. stuad123, brofr456, jonsu789). You have until 4pm Friday July 30th to select your own groups. If you don't select your own group then one will be chosen for you containing students who have completed a similar amount of internal assessment. You will be informed of your group members via email, so please check your University email regularly and let Iain know if there are any problems. All of the programming for this assignment should be done working in your groups.

You must not collaborate with, or discuss issues related to the assignment with anyone who is not a member of your group.

Version Control

All code for this assignment must be stored in the departmental GitLab server https://altitude.otago.ac.nz/. You can log in using your regular Computer Science usercode and password. One group member should create a new blank project called "242-a1" and initialize it with a README containing the names and usercodes of the other group members. The project should have private visibility. After creating the project go to "Project information -> members" and add the other group members and Iain Hewson "ihewson" giving everyone the role of "Maintainer". Make sure that you have a top level directory in your project called "asgn" containing all of the .c and ,h files needed for your assignment. You can put other files in the project as you see fit e.g. combined directories for lab 10 and 11 or separate directories for each group member. The only files that will be marked are those found in the top level asgn directory.

Part of the marks for this assignment come from appropriate use of version control. There is a comprehensive freely available book "Pro Git" that you might find helpful https://git-scm.com/book/en/v2. Note that we don't require you to use branching.

Provided files

We have provided some files for you to use when completing this assignment. The files can be found in the directory `/home/cshome/coursework/242/asgn-files/` and are as follows:

- `print-stats.txt` — contains extra functions which you should add to your `htable` files.
- `sample-asgn` — an executable, compiled on Linux, which you can use to see if your program is working correctly. If your program is correct then it should produce exactly the same output as the sample program.

Comparing hashing strategies

One of the purposes of this assignment is to compare hashing strategies. In order to do this you will modify the hash tables that you develop during labs to add some new features.

Your program needs to meet the following requirements:

- The default size of the hash table should be 113.
- Add a `htable_print_entire_table()` function to print the entire contents of the hash table like `sample-asgn` using the format string `"\%5d␣\%5d␣\%5d␣␣␣\%s\n"` to print each line (spaces have been made visible so you can count them).
- You should be able to use either *linear probing* or *double hashing* as a collision resolution strategy, based on an enumerated type which gets passed to `htable_new()`. You can find more information about enumerated types in the rbt labs. The definition of the enumerated type in `htable.h` should look like this:

  ```
  typedef enum hashing_e { LINEAR_P, DOUBLE_H } hashing_t;
  ```

- Information should be collected about the state of a hash table as it is being built. You will use this information to compare the two collision resolution strategies. There are a number of different attributes of a hash table which could give us an idea of how effective our hashing strategy is. We are going to look at three.

  1. The proportion of keys which were placed in their home cells. i.e. the percentage of keys that were placed into the hash table without colliding with another key.
  2. The maximum number of collisions that occurred before placing a key into the hash table.
  3. The average number of collisions that occurred before placing a key into the hash table.

  There is an easy but ingenious way that we can record all of these statistics as the table builds. All we need to do is add a `stats` array to our hash table struct. Whenever we add a new key to our hash table we just record how many collisions occurred before the key found an empty cell. (Note that unlike our `keys` and `freqs` arrays this one gets filled in sequential order, indexed by `num_keys`).

  Your hash table struct in `htable.c` should now look like this:

```
struct htablerec {
   int capacity;
   int num_keys;
   char **keys;
   int *freqs;
   int *stats;
   hashing_t method;
};
```

We have provided a couple of functions in the file `print-stats.txt` for you to use when displaying the information collected while building your hash table. Please don't alter the output of these two functions, so that your program's output is consistent with `sample-asgn`.

Run your program using different amounts of input and table size to determine the relative merits of linear probing and double hashing. Update the README file in your project to include a summary of your findings.

The asgn1.c file

You need to create a main file called `asgn1.c`, which uses your hash table to perform a number of tasks. By default, words are read from *stdin* and added to your hash table before being printed out alongside their frequencies to *stdout*. This should be done by passing a `print_info` function, shown below, to `htable_print`.

```
static void print_info(int freq, char *word) {
   printf("%-4d %s\n", freq, word);
}
```

All memory allocated should be deallocated before your program finishes.

All words should be read using the `getword()` function from the lab book. Helper functions like `getword` and `emalloc` should be in your mylib.c file.

You should use the *getopt* library to help you process options given on the command line. Here is an example of how to use it:

```
const char *optstring = "ab:c";
char option;

while ((option = getopt(argc, argv, optstring)) != EOF) {
   switch (option) {
      case 'a':
         /* do something */
      case 'b':
         /* the argument after the -b is available
            in the global variable 'optarg' */
      case 'c':
```

```
            /* do something else */
        default:
            /* if an unknown option is given */
    }
}
```

You need to include `getopt.h` to use the getopt library. The letters listed in optstring are possible valid options. The colon following the letter `b` indicates that `b` takes an argument. As the options are being processed by getopt, they get shifted to the front of the argv array. After processing, the index of the first non-option argument is available in the global variable *optind*. For more information have a look at the man page for getopt.h (type `man 3 getopt`).

When your program is given the `-h` option, or an invalid option is given, then a usage message should be printed and your program should exit.

- Your program should respond to command line arguments as specified in this table:

Option	Action performed
-d	Use double hashing as the collision resolution strategy (linear probing is the default).
-e	Display entire contents of hash table on stderr using the format string `("%5d␣%5d␣%5d␣␣␣%sn"` to display the index, frequency, stats, and the key if it exists. (Note that spaces have been made visible in the format string so you can see how many there are).
-p	Print stats info using the functions provided in `print-stats.txt` instead of printing the frequencies and words
-s *snapshots*	Display up to the given number of stats snapshots when given `-p` as an argument. If the table is not full then fewer snapshots will be displayed. Snapshots with 0 entries are not shown.
-t *tablesize*	Use the first prime >= *tablesize* as the size of your hash table. You can assume that *tablesize* will be a number greater than 0.
-h	Print a help message describing how to use the program

Submission

Instead of requiring you to submit your assignment before the due date, we will pull the latest version from the shared repository. Your project should have a top level directory called `asgn` which contains the following files:

```
asgn1.c             mylib.c
htable.c            mylib.h
htable.h
```

- Your assignment should be ready for collection by 4pm on the due date.
- If you have any concerns about the contributions of any of your group members please send an email to iain.hewson@otago.ac.nz.

Marking

This assignment is worth 15% of your final mark for Cosc 242. It is possible to get full marks. In order to do this you must write correct, well-commented code which meets the specifications, and use the departmental GitLab server as specified.

Program marks are awarded for both implementation and style (although it should be noted that it is very bad style to have an implementation that doesn't work).

Allocation of marks	
Implementation	6
Style/Readability	4
Use of version control	5
Total	15

In order to maximise your marks please take note of the following points:

- Your code should compile without warnings on the Linux lab machines using the command:

  ```
  gcc -O2 -W -Wall -ansi -pedantic -lm *.c -o asgn1
  ```

  If your code does not compile, it is considered to be a very, very bad thing!
- Your program should use good C layout as demonstrated in the lab book, including no lines more than 80 characters long.
- Most of your comments should be in your function headers. A function header should include:
  - A description of what the function does.
  - A description of all the parameters passed to it.
  - A description of the return value if there is one.
  - Any special notes.

Overdue assignments will lose marks at a rate of 10% per day late.

You should not discuss issues pertaining to the assignment with anyone not in your group. All programs will be checked for similarity.

Part of this assignment involves you clarifying exactly what your program is required to do. Don't make assumptions, only to find out that they were incorrect when your assignment gets marked.

You should check your University email regularly, since email clarifications may be sent to the class email list.

If you have any questions about this assignment, or the way it will be assessed, please see Iain or send an email to iain.hewson@otago.ac.nz.