# Hash Tables

**Date assigned:** Wednesday, November 15, 2017
**Date due:** Monday, December 4, 2017
***There is no late grace period for this final assignment***
**Points:** 50

## Hash Table

You are to design and implement a **Hash Table ADT**. The HT must use chaining for collision handling and rely on your **Dynamic List** to implement chaining for collision handling. The key, data pair are void *'s and each has a corresponding size specifier.

The user of the HT must supply the HT, the hash function to the HT as a function pointer, a compare function as a function pointer to compare keys in the HT, and the HT size during the htCreate function call. You need to design the hash table header file consisting of the data structure(s) as well as the functions for the HT. The function names are to be htName, so for instance, htCreate, htTerminate, …

The HT has the expected set of functions to:

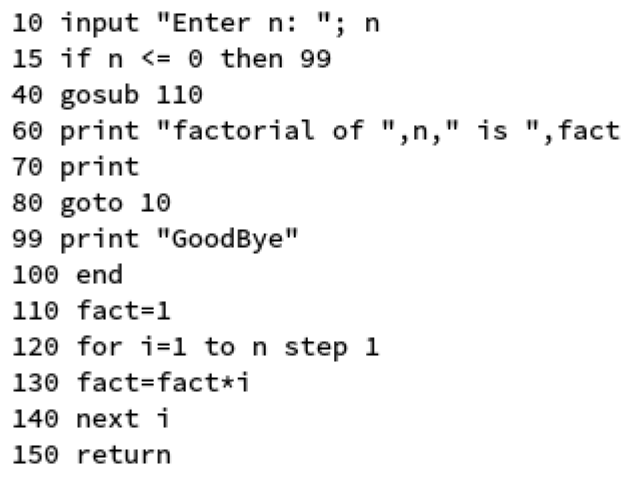
a) create a HT
b) terminate a HT
c) insert a key and associated data into the HT
d) delete a key (and associated data) from the HT returning the data
e) update a key's data
f) find a key in the HT returning the key's data
g) check if the HT is empty
h) check if the HT is full
i) print the HT
j) any other functions you so desire pertaining to a HT

In addition to implementing the data structure, you must provide a Makefile and test driver (htDriver.c that produces an executable named htdriver) that thoroughly tests your data structure.

## Renumber Program

Sooooo, what are we going to do with our hash table once it's implemented?

The introduction of microcomputers in the mid-1970s was the start for the explosive growth of the BASIC programming language. Here is a BASIC program (fact.bas) to find the factorial of a number.



```
10 input "Enter n: "; n
15 if n <= 0 then 99
40 gosub 110
60 print "factorial of ",n," is ",fact
70 print
80 goto 10
99 print "GoodBye"
100 end
110 fact=1
120 for i=1 to n step 1
130 fact=fact*i
140 next i
150 return
```

Notice that each line has an associated line number.

We used to have to type in a line number and then the BASIC code. You can imagine that the line numbering scheme gets messy and in some cases, one can even run out of line numbers. For instance, we can only put 4 more lines between line 10 and line 15. It would be nice to have a renumbering program and that's what I'm asking you to write.

Your renumbering program will be run as follows:

```
renumber inputprogram outputprogram startnumber incrementnumber
```

| | |
|---|---|
| where | renumber is the name of the renumbering program |
| | inputprogram is the BASIC program that is to be renumbered |
| | outputprogram is the renumbered BASIC program |
| | startnumber is the number of the first statement in outputprogram |
| | incrementnumber is the value added to each subsequent line number |

As an example, renumber fact.bas rfact.bas 100 10 will renumber a program called fact.bas in rfact.bas where the first line number will be 100, the second line number will be 110, then 120, and so on.

There are five kinds of BASIC statements that can cause problems during the renumbering process and they are:

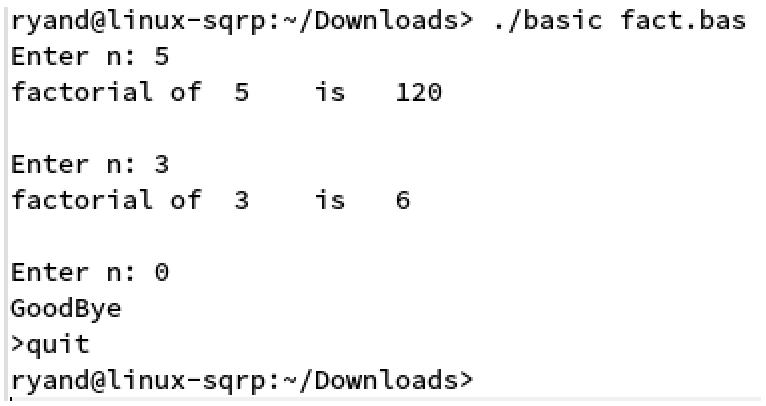
1. `if expression then ###`
2. `gosub ###`
3. `goto ###`
4. `on expr goto ### {, ###, …}`
5. `on expr gosub ### {, ###, …}`

For this assignment, you are to handle the first three BASIC statements (1. , 2., and 3.) and assume that the program passed to your renumber program is well-formed. There will not be any syntax errors in the input program and there cannot be any syntax errors in the output program following the execution of your renumbering program. There will be one BASIC statement per line. Line numbers will be in the range of 0 to 99999 inclusive. For this assignment, the reserved words **if**, **then**, **gosub**, and **goto** will all be lowercase.

I will test your renumbering program using Chipmunk BASIC which can be found at http://www.nicholson.com/rhn/basic.html. You can simply download the correct version of Chipmunk BASIC and run any syntactically correct BASIC program. For instance, to run the above program, use a text editor to type in the program as you see it. Then execute the following commands:

```
./basic fact.bas              OR         ./basic
quit                                     load "fact.bas"
                                         run
                                         quit
```

Here's how to run a Chipmunk BASIC program:



```
ryand@linux-sqrp:~/Downloads> ./basic fact.bas
Enter n: 5
factorial of  5    is   120

Enter n: 3
factorial of  3    is   6

Enter n: 0
GoodBye
>quit
ryand@linux-sqrp:~/Downloads>
```

Notes:

1. You will need at least two modules other than your list module. You will need a hash table module (**HashTable**) and a renumber module (**Renumber**) for sure.

2. Submit a file called **cs300_7_PUNetID.tar.gz** by 9:15am on the due date. This file must include at least the two new projects you are to write as well as **GenericDynamicList**. Each project is to be complete such that I can type make in any of the projects and execute any driver I so desire. Make sure you have all dependencies set correctly and that each Makefile builds the appropriate object files before building the executable. Also, each module's Makefile must have a target called **valgrind.** Typing **make valgrind** in the Renumber project executes the valgrind command:

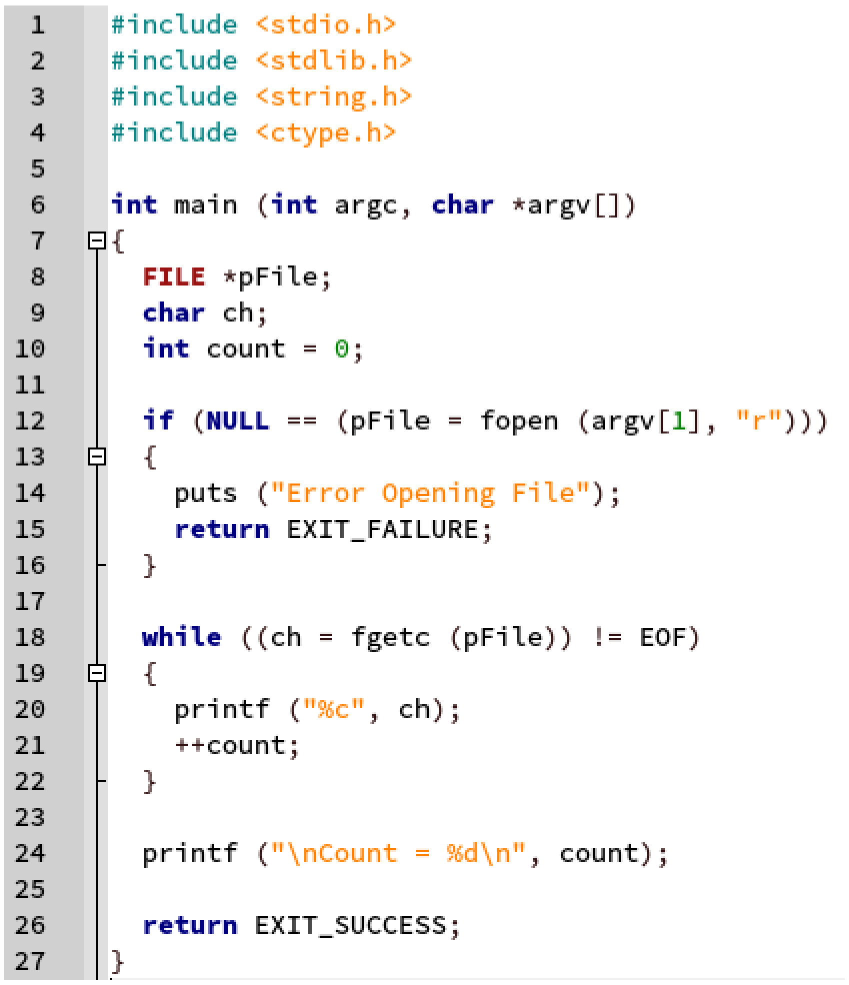
```
valgrind –v –leak-check=yes bin/renumber data/input.txt data/valgrind.txt 1 1
```

I will clean all projects BEFORE typing make in the Renumber project.

Data file input is to be from the command line.
Turn in a color, double sided, stapled packet of code by the due date. The packet must be in the following order:

- renumber.c (.h then .c if you have both, otherwise just .c)
- ht.c (.h then .c)
- Any extra .h/.c pairs you have. (do not include any code from previous projects)
- New Makefiles printed in any order

## Character Processing in C



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int main (int argc, char *argv[])
{
  FILE *pFile;
  char ch;
  int count = 0;

  if (NULL == (pFile = fopen (argv[1], "r")))
  {
    puts ("Error Opening File");
    return EXIT_FAILURE;
  }

  while ((ch = fgetc (pFile)) != EOF)
  {
    printf ("%c", ch);
    ++count;
  }

  printf ("\nCount = %d\n", count);

  return EXIT_SUCCESS;
}
```

You will most likely need to make 2 passes over the BASIC program. rewind sets the file position back to the beginning of the stream.

Other Information:

1. A line number can be a blank line
2. I will not do anything tricky like
   **`10 print "goto 10"`**