Data Structures and Algorithms

Assignment 1

Submitted by:
Apnatva Singh Rawat
D Stream
21354929

Things to Note

1. My code inputs the name and returns the frequency of the name, but is case INSENSITIVE (it was coded to be so). Even while reading the CSV file it has a decreased number of values. For the values given as 

in the CSV file, it will treat Waller and waller as the same. This is how I expected we were supposed to write the program hence I only tested it after I had modified it to be so. This is very intentional. This decreased the number of terms and hence decreased the load and had an increase in the number of collisions because of the way that I was storing these values.

2. I have not used the skeleton provided, I have not even taken inspiration of it. Each function was well thought out and planned and written by only me. Some of them have been edited multiple times and some have been completely rewritten.

3. For the hash function in Task 2 I added one line hash = (hash + numberOfChars) % numberOfEntries; to the hash function given in the PDF. The reasons for this was fairly simple,
   1. Even if two different names have the same hash value, the offset number of characters will cause will help in differentiation names of different length, as is important with surnames.
   2. It is one non-looped line, hence barely adds to the overhead and has little to no effect on the final runtime of the program as compared to the original basic hashing function. I have provided the entire code below.

4. After implementing the hashing function in task 3, I found an increase in the number of collisions. Whether this is due to an unsuccessful implementation of the said hashing function (which I doubt) or just the fact that the double hashing function is just not suitable.

5. I have used linked listed to build my hash table for all the tasks. Since we could use any approach in task 4, I have used chaining to resolve all collisions in my hash table. Each node is created and then a 'key' (hash value) is assigned to it. Then on the basis of this hash value it is placed at a particular position so that the linked list is sorted with respect to keys even as I am building it. This helps me as I will not have to sort the linked list after it is formed in the first three tasks.

6. My outputs differ on my local machine (MacOS, using VSCode) and on the submit website. I have attached screenshots of both the outputs and also the code in this pdf.

Task 1

SUBMITTY RESULT:

RESULT OBTAINED AT LOCAL MACHINE:

```
apnatvarawat@Lay-Z-MacBook-Pro A1 % ./A1_P1
File (null) loaded
 Capacity: 59
 Num Terms: 37
 Collisions: 47
 Load: 0.627119
Enter term to get frequency or type "quit" to escape
Stafford
>>> Stafford - 4
Murphy
>>> Murphy - 0
Wagstaffe
>>> Wagstaffe - 2
Doe
>>> Doe - 0
sghhj
>>> sghhj - 0
quit
apnatvarawat@Lay-Z-MacBook-Pro A1 %
```

CODE:#INCLUDE <STDIO.H>

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

#define numberOfEntries 59
//can also choose a random larger prime number for unkown number
of entries

int numTerms;
int collisions;
float load;
```

```
typedef struct surnames surnames;
struct surnames{
    int key;
    int freq;
    char surname[20];
    surnames* next;
};

surnames* hashTableTop = NULL;

int hash(char* sampleName){
    int hash = 0;
    while(*sampleName){
        if (*sampleName >= 'a' && *sampleName <= 'z')*sampleName =
*sampleName - 32;
        hash = (hash + *sampleName) % numberOfEntries;
        sampleName++;
    }
    return hash;
}

void addNewName(char* sampleName, int hashValue){
    surnames* newNode = (surnames*) malloc(sizeof(surnames));
    newNode->key = hashValue;
    newNode->freq = 1;
    strcpy(newNode->surname,sampleName);
    newNode->next = hashTableTop;
    hashTableTop = newNode;
    numTerms++;
}

void checkAndAdd(char* sampleName){
    int hashValue = hash(sampleName);
    int flag = 1;
    if (hashTableTop != NULL){ // no values in hashtable hence new
node must be added
        surnames* tempPointer = hashTableTop;
        while (tempPointer != NULL){ // will look for the same
surname without key as hash table is not ready for retrieval yet
            if (strcmp(tempPointer->surname, sampleName) == 0){
```

```
                tempPointer->freq++;
                flag = 0;
                //numTerms++;
            }
            tempPointer = tempPointer->next;
        }
        tempPointer = hashTableTop;
        while (tempPointer != NULL){ // if this loop is raeched, 
that surname does not exist in the hash table
            if (tempPointer->key == hashValue){ // search if 
duplicate key exists for differnet surname
                hashValue = (hashValue + 1) % numberOfEntries; // 
if it does exist, use linear probing to find next available hash 
value
                tempPointer = hashTableTop;
                collisions++;
                continue;
            }
            tempPointer = tempPointer->next;
        }
    }
    if (flag)addNewName(sampleName, hashValue);
}

void sortList(){
    surnames* tempPointer;
    int flag = 1;
    int swapKey;
    int swapFreq;
    char swapSurname[20];
    while(flag){
        flag = 0;
        tempPointer = hashTableTop;
        while (tempPointer->next != NULL){
            if (tempPointer->key > tempPointer->next->key){
                flag = 1;

                swapKey = tempPointer->key;
                swapFreq = tempPointer->freq;
                strcpy(swapSurname,tempPointer->surname);
```

```
                tempPointer->key = tempPointer->next->key;
                tempPointer->freq = tempPointer->next->freq;
                strcpy(tempPointer->surname,tempPointer->next-
>surname);

                tempPointer->next->key = swapKey;
                tempPointer->next->freq = swapFreq;
                strcpy(tempPointer->next->surname,swapSurname);
                //can values be swapped in a better / faster way?
                //i was unable to find a more efficent way of
swapping values
            }
            tempPointer = tempPointer->next;
        }
    }
}

void findName(char* sampleInput, char* actualInput){
    int hashValue = hash(sampleInput);
    surnames* tempPointer = hashTableTop;
    while(hashValue < tempPointer->key){
        hashValue--;
        tempPointer = tempPointer->next;
    }//at the end of this loop tempPointer will be pointing to the
hash value of the input string
    //since we use linear probing to store values after collision,
    //the actual string may not be at that value and hence we have
to go further
    while (tempPointer != NULL && strcmp(sampleInput,tempPointer-
>surname) != 0){
        tempPointer = tempPointer->next;
    }
    if (tempPointer != NULL)printf(">>> %s - %d \n", actualInput,
tempPointer->freq);
    else printf(">>> %s - 0\n",actualInput);
}

int main(int argc, char *argv[]){
```

```
    // FILE* fileName = fopen("/Users/anandrawat/Desktop/Sem\ 5/
Data\ Structures/A1/names.csv","r");
    FILE* fileName = fopen(argv[1], "r");
    if (fileName == NULL){
        printf("File did not open");
        return 1;
    }
    printf("File %s loaded\n",argv[1]);
    numTerms = 0;
    collisions = 0;
    char storeFromFile[20];
    while (fgets(storeFromFile,20,fileName)){
        int length = strlen(storeFromFile);
        storeFromFile[length-1] = '\0';//removing last character
and replacing it with end string
        checkAndAdd(storeFromFile);
    }
    sortList(); // now the hash table is ready for retrieval of
values based solely on keys
    load = (float) numTerms / numberOfEntries;
    printf(" Capacity: %d\n", numberOfEntries);
    printf(" Num Terms: %d\n", numTerms);
    printf(" Collisions: %d\n", collisions);
    printf(" Load: %.6f\n", load);
    char inputString[20];
    char inputStringUpper[20];
    printf("Enter term to get frequency or type \"quit\" to
escape\n");
    while(1){
        scanf("%[^\n]%*c", inputString);
        if (strcmp(inputString,"quit")==0)break;
        strcpy(inputStringUpper,inputString);
        //printf("%s - %s\n",inputString,inputStringUpper);
        findName(inputStringUpper,inputString);
    }
    return 0;
}
```

Task 2

SUBMITTY RESULT:

C Testing part 2 - you are expected to have lower collisions than the expected output - marks for this will be awarded manually Hide Details

Visualize whitespace characters

Student STDOUT.txt

```
1  File names.csv loaded
2   Capacity: 59
3   Num Terms: 37
4   Collisions: 40
5   Load: 0.627119
6  Enter term to get frequency or type "quit" to escape
7  >>> Stafford - 0
8  >>> Murphy - 0
9  >>> Wagstaffe - 0
10 >>> Doe - 0
11 >>> sghhj - 0
12
```

Expected STDOUT.txt

```
1  File names.csv loaded
2   Capacity: 59
3   Num Terms: 42
4   Collisions: 38
5   Load: 0.711864
6  Enter term to get frequency or type "quit" to escape
7  >>> Stafford - 4
8  >>> Murphy - 0
9  >>> Wagstaffe - 2
10 >>> Doe - 0
11 >>> sghhj - 0
12 >>>
```

RESULT OBTAINED AT LOCAL MACHINE:

```
apnatvarawat@Lay-Z-MacBook-Pro A1 % ./A1_P2
File (null) loaded
 Capacity: 59
 Num Terms: 37
 Collisions: 40
 Load: 0.627119
Enter term to get frequency or type "quit" to escape
Stafford
>>> Stafford - 4
Murphy
>>> Murphy - 0
Wagstaffe
>>> Wagstaffe - 2
Doe
>>> Doe - 0
sghhj
>>> sghhj - 0
quit
apnatvarawat@Lay-Z-MacBook-Pro A1 %
```

CODE:#INCLUDE <STDIO.H>

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

#define numberOfEntries 59 //closest prime number to number of entries
//can also choose a random large prime number for unkown number of entries
```

```
int numTerms;
int collisions;
float load;

typedef struct surnames surnames;
struct surnames{
    int key;
    int freq;
    char surname[20];
    surnames* next;
};

surnames* hashTableTop = NULL;

int hash(char* sampleName){
    int hash = 0;
    int numberOfChars = 0;
    while(*sampleName){
        if (*sampleName >= 'a' && *sampleName <= 'z')*sampleName =
*sampleName - 32;
        hash = hash + *sampleName;
        sampleName++;
        numberOfChars++;
    }
    hash = (hash + numberOfChars) % numberOfEntries;
    return hash;
}

void addNewName(char* sampleName, int hashValue){
    surnames* newNode = (surnames*) malloc(sizeof(surnames));
    newNode->key = hashValue;
    newNode->freq = 1;
    strcpy(newNode->surname,sampleName);
    newNode->next = hashTableTop;
    hashTableTop = newNode;
    numTerms++;
}

void checkAndAdd(char* sampleName){
```

```
    int hashValue = hash(sampleName);
    int flag = 1;
    if (hashTableTop != NULL){ // no values in hashtable hence new
node must be added
        surnames* tempPointer = hashTableTop;
        while (tempPointer != NULL){ // will look for the same
surname and return 0 if found.
            if (strcmp(tempPointer->surname, sampleName) == 0){
                tempPointer->freq++;
                flag = 0;
            }
            tempPointer = tempPointer->next;
        }
        tempPointer = hashTableTop;
        while (tempPointer != NULL){ // if this loop is raeched,
that surname does not exist in the hash table
            if (tempPointer->key == hashValue){ // search if
duplicate key exists for differnet surname
                hashValue = (hashValue + 1) % numberOfEntries; //
if it does exist, use linear probing to find next available hash
value
                tempPointer = hashTableTop; // check for this new
hash value from the top
                collisions++;
                continue;
            }
            tempPointer = tempPointer->next;
        }
    }
    if (flag)addNewName(sampleName, hashValue);
}

void sortList(){
    surnames* tempPointer;
    int flag = 1;
    int swapKey;
    int swapFreq;
    char swapSurname[20];
    while(flag){
        flag = 0;
```

```
        tempPointer = hashTableTop;
        while (tempPointer->next != NULL){
            if (tempPointer->key > tempPointer->next->key){
                flag = 1;

                swapKey = tempPointer->key;
                swapFreq = tempPointer->freq;
                strcpy(swapSurname,tempPointer->surname);

                tempPointer->key = tempPointer->next->key;
                tempPointer->freq = tempPointer->next->freq;
                strcpy(tempPointer->surname,tempPointer->next-
>surname);

                tempPointer->next->key = swapKey;
                tempPointer->next->freq = swapFreq;
                strcpy(tempPointer->next->surname,swapSurname);
                //can values be swapped in a better / faster way?
            }
            tempPointer = tempPointer->next;
        }
    }
}

void findName(char* sampleInput, char* actualInput){
    int hashValue = hash(sampleInput);
    surnames* tempPointer = hashTableTop;
    while(hashValue < tempPointer->key){
        hashValue--;
        tempPointer = tempPointer->next;
    }//at the end of this loop tempPointer will be pointing to the
hash value of the input string
    //since we use linear probing to store values, the actual
string may not be at that value and hence we have to go further
    while (tempPointer != NULL && strcmp(sampleInput,tempPointer-
>surname) != 0){
        tempPointer = tempPointer->next;
    }
    if (tempPointer != NULL)printf(">>> %s - %d \n", actualInput,
tempPointer->freq);
```

```
    else printf(">>> %s - 0\n",actualInput);
}

int main(int argc, char *argv[]){
    // FILE* fileName = fopen("/Users/anandrawat/Desktop/Sem\ 5/
Data\ Structures/A1/names.csv","r");
    FILE* fileName = fopen(argv[1], "r");
    if (fileName == NULL){
        printf("File did not open");
        return 1;
    }
    printf("File %s loaded\n",argv[1]);
    numTerms = 0;
    collisions = 0;
    char storeFromFile[20];
    while (fgets(storeFromFile,20,fileName)){
        int length = strlen(storeFromFile);
        storeFromFile[length-1] = '\0';
        checkAndAdd(storeFromFile);
    }
    sortList();
    load = (float) numTerms / numberOfEntries;
    printf(" Capacity: %d\n", numberOfEntries);
    printf(" Num Terms: %d\n", numTerms);
    printf(" Collisions: %d\n", collisions);
    printf(" Load: %.6f\n", load);
    char inputString[20];
    char inputStringUpper[20];
    printf("Enter term to get frequency or type \"quit\" to
escape\n");
    while(1){
        scanf("%[^\n]%*c", inputString);
        if (strcmp(inputString,"quit")==0)break;
        strcpy(inputStringUpper,inputString);
        findName(inputStringUpper,inputString);
        //printf("%s - %s\n",inputString,inputStringUpper);
    }
    return 0;
}
```

Task 3

SUBMITTY RESULT:

RESULT OBTAINED AT LOCAL MACHINE:

```
apnatvarawat@Lay-Z-MacBook-Pro A1 % ./A1_P3
File (null) loaded
 Capacity: 59
 Num Terms: 37
 Collisions: 43
 Load: 0.627119
Enter term to get frequency or type "quit" to escape
Stafford
>>> Stafford - 4
Murphy
>>> Murphy - 0
Wagstaffe
>>> Wagstaffe - 2
Doe
>>> Doe - 0
sghhj
>>> sghhj - 0
quit
apnatvarawat@Lay-Z-MacBook-Pro A1 %
```

CODE:#INCLUDE <STDIO.H>

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

#define numberOfEntries 59 //closest prime number to number of
entries
//can also choose a random large prime number for unkown number of
entries

int numTerms;
int collisions;
float load;

typedef struct surnames surnames;
struct surnames{
    int key;
    int freq;
    char surname[20];
    surnames* next;
};

surnames* hashTableTop = NULL;

int hash1(char* sampleName){
    int hash = 0;
    while(*sampleName){
        if (*sampleName >= 'a' && *sampleName <= 'z')*sampleName =
*sampleName - 32;
        hash = (hash + *sampleName) % numberOfEntries;
        sampleName++;
    }
    return hash;
}

int hash3(char* sampleName){
    int hash = 0;
    while (*sampleName){
        hash = 1 + ((hash + *sampleName) % (numberOfEntries-1));
        sampleName++;
    }
    return hash;
```

```
}

void addNewName(char* sampleName, int hashValue){
    surnames* newNode = (surnames*) malloc(sizeof(surnames));
    newNode->key = hashValue;
    newNode->freq = 1;
    strcpy(newNode->surname,sampleName);
    newNode->next = hashTableTop;
    hashTableTop = newNode;
    numTerms++;
}

void checkAndAdd(char* sampleName){
    int hashValue = hash1(sampleName);
    int flag = 1;
    if (hashTableTop != NULL){ // no values in hashtable hence new
node must be added
        surnames* tempPointer = hashTableTop;
        while (tempPointer != NULL){ // will look for the same
surname and return 0 if found.
            if (strcmp(tempPointer->surname, sampleName) == 0){
                tempPointer->freq++;
                flag = 0;
            }
            tempPointer = tempPointer->next;
        }
        tempPointer = hashTableTop;
        while (tempPointer != NULL){ // if this loop is raeched,
that surname does not exist in the hash table
            if (tempPointer->key == hashValue){ // search if
duplicate key exists for differnet surname
                hashValue = (hashValue + hash3(sampleName)) %
numberOfEntries; // if it does exist, use linear probing to find
next available hash value
                tempPointer = hashTableTop;
                collisions++;
                continue;
            }
            tempPointer = tempPointer->next;
        }
```

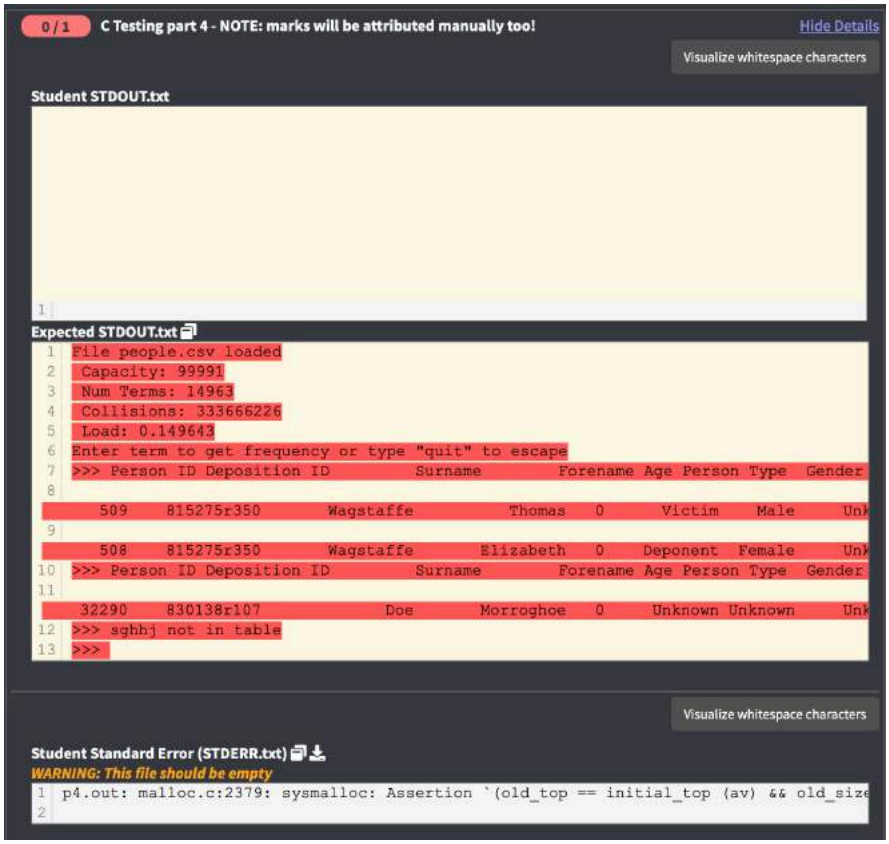
```
    }
    if (flag)addNewName(sampleName, hashValue);
}

void sortList(){
    surnames* tempPointer;
    int flag = 1;
    int swapKey;
    int swapFreq;
    char swapSurname[20];
    while(flag){
        flag = 0;
        tempPointer = hashTableTop;
        while (tempPointer->next != NULL){
            if (tempPointer->key > tempPointer->next->key){
                flag = 1;

                swapKey = tempPointer->key;
                swapFreq = tempPointer->freq;
                strcpy(swapSurname,tempPointer->surname);

                tempPointer->key = tempPointer->next->key;
                tempPointer->freq = tempPointer->next->freq;
                strcpy(tempPointer->surname,tempPointer->next-
>surname);

                tempPointer->next->key = swapKey;
                tempPointer->next->freq = swapFreq;
                strcpy(tempPointer->next->surname,swapSurname);
                //can values be swapped in a easier / faster way?
            }
            tempPointer = tempPointer->next;
        }
    }
}

void findName(char* sampleInput, char* actualInput){
    int hashValue = hash1(sampleInput);
    //printf("%s\n",sampleInput);
    surnames* tempPointer = hashTableTop;
```

```
    int flag = 1;
    int i = 0;
    while (i < numberOfEntries && flag){
        i++;
        //printf("%d - %d\n",hashValue,i);
        while (tempPointer != NULL){
            if (tempPointer->key == 40){
                if (strcmp(tempPointer->surname,sampleInput) == 0)
{
                    printf(">>> %s - %d \n", actualInput,
tempPointer->freq);
                    flag = 0;
                    break;
                }
                //printf("%d - %s\n",strcmp(tempPointer-
>surname,sampleInput),tempPointer->surname);
            }
            tempPointer = tempPointer->next;
        }
        hashValue = (hashValue + hash3(sampleInput)) %
numberOfEntries;
        tempPointer = hashTableTop;
    }
    if (flag)printf(">>> %s - 0\n",actualInput);
}

int main(int argc, char *argv[]){
    // FILE* fileName = fopen("/Users/anandrawat/Desktop/Sem\ 5/
Data\ Structures/A1/names.csv","r");
    FILE* fileName = fopen(argv[1], "r");
    if (fileName == NULL){
        printf("File did not open");
        return 1;
    }
    printf("File %s loaded\n",argv[1]);
    numTerms = 0;
    collisions = 0;
    char storeFromFile[20];
    while (fgets(storeFromFile,20,fileName)){
        int length = strlen(storeFromFile);
```

```
        storeFromFile[length-1] = '\0';//removing last character
and replacing it with end string
        checkAndAdd(storeFromFile);
    }
    sortList();

    load = (float) numTerms / numberOfEntries;
    printf(" Capacity: %d\n", numberOfEntries);
    printf(" Num Terms: %d\n", numTerms);
    printf(" Collisions: %d\n", collisions);
    printf(" Load: %.6f\n", load);
    char inputString[20];
    char inputStringUpper[20];
    memset(inputStringUpper,'\0',20);
    int i = 0;
    printf("Enter term to get frequency or type \"quit\" to
escape\n");
    while(1){
        scanf("%[^\n]%*c", inputString);
        if (strcmp(inputString,"quit")==0)break;
        for (i = 0; inputString[i] != '\0'; i++){
            if (inputString[i] >= 'a' && inputString[i] <=
'z')inputStringUpper[i] = inputString[i] - 32;
            else inputStringUpper[i] = inputString[i];
        }
        //printf("%s - %s\n",inputString,inputStringUpper);
        findName(inputStringUpper,inputString);
        memset(inputStringUpper,'\0',20);
    }
    return 0;
}
```

Task 4

SUBMITTY RESULT:



RESULT OBTAINED AT LOCAL MACHINE:
ON TRUNCATED.CSV

```
apnatvarawat@Lay-Z-MacBook-Pro A1 % ./A1_P4
File (null) loaded
 Capacity: 99991
 Num Terms: 20
 Collisions: 0
 Load: 0.000200
Enter term to get frequency or type "quit" to escape
Wagstaffe
>>> Wagstaffe not in table
Ward
Person ID Deposition ID Surname    Forename    Age   Person Type   Gender   Nationality   Religion   Occupation
37120     833080r062    WARD       zacarie     0     Victim        Male     Unknown       Unknown    Gentleman
quit
apnatvarawat@Lay-Z-MacBook-Pro A1 %
```

ON PEOPLE.CSV

```
apnatvarawat@Lay-Z-MacBook-Pro A1 % ./A1_P4
File (null) loaded
 Capacity: 99991
 Num Terms: 60171
 Collisions: 59222
 Load: 0.601764
Enter term to get frequency or type "quit" to escape
Wagstaffe
Person ID Deposition ID Surname    Forename    Age   Person Type   Gender   Nationality   Religion   Occupation
508       815275r350    WAGSTAFFE  Elizabeth   0     Deponent      Female   Unknown       Unknown    Unknown
509       815275r350    WAGSTAFFE  Thomas      0     Victim        Male     Unknown       Unknown    Unknown
quit
apnatvarawat@Lay-Z-MacBook-Pro A1 %
```

CODE:#INCLUDE <STDIO.H>

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

#define numberOfEntries 99991
int numTerms;
int collisions;
float load;

typedef struct people people;
struct people{
    int key;
    people* next;
    people* nextInChain;
    char personID[20];
    char depositionID[20];
    char surname[20];
    char forename[20];
    char age[3];
    char personType[20];
    char gender[20];
    char nationality[20];
    char religion[20];
    char occupation[20];
};
people* hashTableTop = NULL;
people* printThis = NULL;

int hash(char* sampleName){
    int hash = 0;
    while(*sampleName){
```

```
        if (*sampleName >= 'a' && *sampleName <= 'z')*sampleName =
*sampleName - 32;
        hash = hash + *sampleName;
        sampleName++;
    }
    hash = hash % numberOfEntries;
    return hash;
}

void addNode(char* dataSet){
    people* newNode = (people*) malloc(sizeof(people));
    sscanf(dataSet,"%s %s %s %s %s %s %s %s %s %s",newNode-
>personID,newNode->depositionID,newNode->surname,newNode-
>forename,newNode->age,newNode->personType,newNode-
>gender,newNode->nationality,newNode->religion,newNode-
>occupation);
    int hashValue = hash(newNode->surname);
    newNode->key = hashValue;
    newNode->nextInChain = NULL;
    people* chaining;
    if (hashTableTop == NULL){
        newNode->next = NULL;
        hashTableTop = newNode;
    }
    else if (hashTableTop->next == NULL){
        if (hashValue < hashTableTop->key){
            newNode->next = hashTableTop;
            hashTableTop = newNode;
        }
        else if (hashValue == hashTableTop->key){
            collisions++;
            chaining = hashTableTop;
            while(chaining->nextInChain!=NULL)chaining = chaining-
>nextInChain;
            chaining->nextInChain = newNode;
            newNode->next = NULL;
        }
        else{
            hashTableTop->next = newNode;
            newNode->next = NULL;
```

```
        }
    }
    else {
        people* tempPointer = hashTableTop;
        people* prevPointer = NULL;
        while(tempPointer != NULL){
            if (hashValue > tempPointer->key){
                prevPointer = tempPointer;
                tempPointer = tempPointer->next;
                if (tempPointer == NULL && hashValue > 
prevPointer->key){
                    prevPointer->next = newNode;
                    newNode->next = NULL;
                }
            }
            else if (hashValue == tempPointer->key){//collision
                collisions++;
                chaining = tempPointer;
                while(chaining->nextInChain!=NULL)chaining = 
chaining->nextInChain;
                chaining->nextInChain = newNode;
                newNode->next = NULL;
                tempPointer = NULL;
            }
            else {
                if (prevPointer == NULL){
                    newNode->next = hashTableTop;
                    hashTableTop = newNode;
                }
                else{
                prevPointer->next = newNode;
                newNode->next = tempPointer;
                }
                tempPointer = NULL;
            }
        }
    }
}

int printTheseValues(char* sampleInput){
```

```
    people* tempPointer = printThis;
    int flag = 1;
    int returnFlag = 1;
    while(tempPointer!=NULL){
        if (strcmp(tempPointer->surname, sampleInput) == 0){
            returnFlag = 0;
            if (flag){
                flag = 0;
                printf("Person ID Deposition ID Surname
Forename    Age   Person Type   Gender   Nationality   Religion
Occupation\n");
            }
            printf("%-9s %-13s %-10s %-11s %-5s %-13s %-8s %-13s
%-10s %-13s\n",tempPointer->personID,tempPointer-
>depositionID,tempPointer->surname,tempPointer-
>forename,tempPointer->age,tempPointer->personType,tempPointer-
>gender,tempPointer->nationality,tempPointer-
>religion,tempPointer->occupation);
        }
        tempPointer = tempPointer->nextInChain;
    }
    return returnFlag;
}

void findName(char* sampleInput, char* actualInput){
    int hashValue = hash(sampleInput);
    people* tempPointer = hashTableTop;
    int flag = 1;
    while (tempPointer != NULL){
        if (tempPointer->key == hashValue){
            printThis = tempPointer;
            flag = printTheseValues(sampleInput);
            break;
        }
        else tempPointer = tempPointer->next;
    }
    if (flag){printf(">>> %s not in table\n",actualInput);}
}

int main(int argc, char *argv[]){
```

```
    // FILE* fileName = fopen("/Users/anandrawat/Desktop/Sem\ 5/
Data\ Structures/A1/truncated.csv", "r");
    FILE* fileName = fopen(argv[1], "r");
    if (fileName == NULL){
        printf("File did not open");
        return 1;
    }
    printf("File %s loaded\n",argv[1]);
    int i;
    numTerms = 0;
    collisions = 0;
    char buffer[1000];
    fgets(buffer,1000, fileName);
    while (fgets(buffer,1000, fileName)) {
        for (i = 0; buffer[i]!='\n'; i++){
            if (buffer[i] == ' ') buffer[i] = '-';
            else if (buffer[i] == ',' || buffer[i] ==
'\"')buffer[i] = ' ';//processing data in a way that it is easy to
separate
        }
        addNode(buffer);
        numTerms++;
    }

    load = (float) numTerms / numberOfEntries;
    printf(" Capacity: %d\n", numberOfEntries);
    printf(" Num Terms: %d\n", numTerms);
    printf(" Collisions: %d\n", collisions);
    printf(" Load: %.6f\n", load);
    char inputString[20];
    char inputStringModified[20];
    memset(inputStringModified,'\0',20);
    printf("Enter term to get frequency or type \"quit\" to
escape\n");
    while(1){
        scanf("%[^\n]%*c", inputString);
        if (strcmp(inputString,"quit")==0)break;
        for (i = 0; inputString[i] != '\0'; i++){
            if (inputString[i] == ' ')inputStringModified[i] =
'-';
```

```
            else inputStringModified[i] = inputString[i];
        }
        findName(inputStringModified,inputString);
        memset(inputStringModified,'\0',20);
    }
    fclose(fileName);
    return 0;
}
```