2018

CPS1011: Programming Principles in C – Assignment Report

JAMIE GRECH

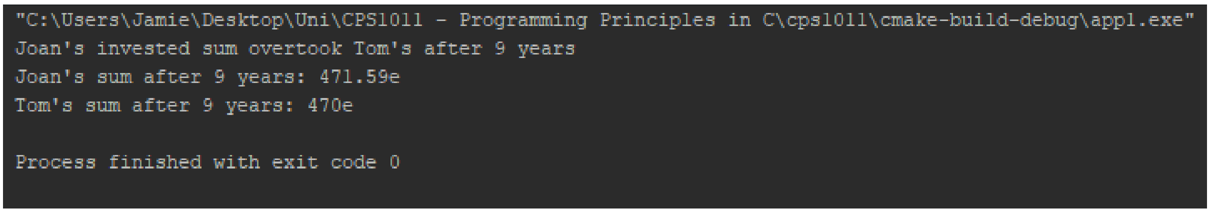
Contents

Question 1A

Code

```
#include <stdio.h>

int main() {

    int T = 200;//this is the initial investment by Tom
    double J = 200;//this is the initial investment by Joan, represented as a
double since it is i
    int i = 0;//this is the counter that will be used to count the amount of years
taken

    while(J <= T){//a while loop is used since the looping condition is based on
variables calculated within the loop and not on a predefined number of loops
        T = T + 30;//15% of 200 is 30 which is added to Tom's account annually
        J = J*1.1;//Joan's account balance is multiplied by 110% every year since
it is based on compound interest
        i++;//the counter is incremented at the end since every pass of the loop
represents a year passing
    }

    printf("Joan's invested sum overtook Tom's after %d years \n", i);//display for
years taken for Joan's investment to overtake Tom's
printf("Joan's sum after %d years: %.2lfe \n", i, J); //%.2lf identifier used for
Joan's balance since it is not an integer value to round it to 2 d.p.
printf("Tom's sum after %d years: %de \n", i, T);//Tom's balance at the time when
Joan's exceeds his for the first time


    return 0;//program termination

}
```

Testing



```
"C:\Users\Jamie\Desktop\Uni\CPS1011 - Programming Principles in C\cps1011\cmake-build-debug\app1.exe"
Joan's invested sum overtook Tom's after 9 years
Joan's sum after 9 years: 471.59e
Tom's sum after 9 years: 470e

Process finished with exit code 0
```

Figure 1

Explanation

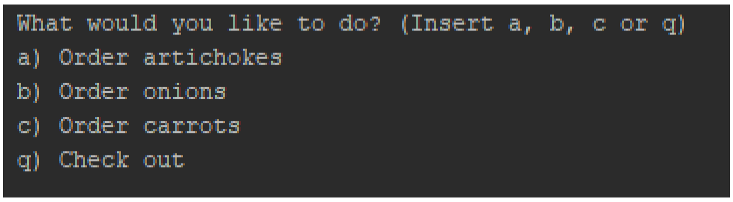
In the above program, iteration is used in order to calculated the interest accumulated by Tom and Joan, represented by T and J, on a yearly basis. The iteration is implemented by using a while loop which repeats until Joan’s total sum overtakes Tom’s. A counter i is declared and initialised with a value of 0 before the loop. For every iteration through the loop, the counter is incremented by one to represent a year passing. The value in i after the loop stops is the amount of years it takes for Joan’s invested sum to overtake Tom’s invested sum. This number of years as well as Joan and Tom’s respective sums after the years have passed are then outputted through 3 separate printf statements at the end of the program. The identifier %.2lf is used to represent Joan’s sum so as to have it outputted to 2 decimal places.

Question 1B

Code

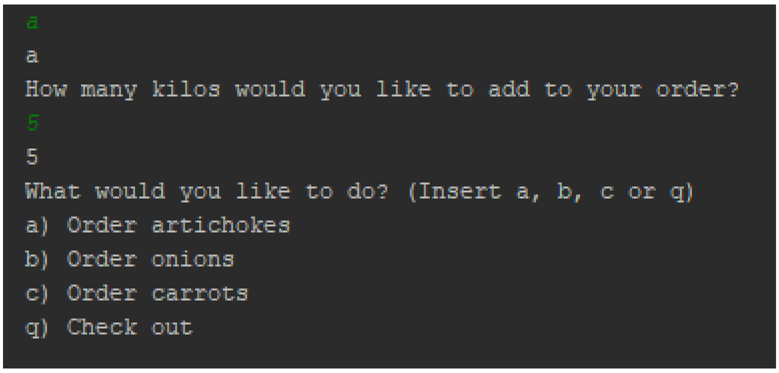
```
#include <stdio.h>

int main() {

    FILE *fp;//points to the file where the receipt is to be saved
    int a, o, c, tempa, tempo, tempc, totalWeight, exit;//declaration of variables
to be used
    a = 0;//kilos of artichokes
    o = 0;//kilos of onions
    c = 0;//kilos of carrots
    totalWeight = 0;//total weight in kilos
    exit = 0;//exit condition for while loop, set to 1 when user wishes to checkout
and therefore exit the loop
    char response, checkout;//stores user's response from main menu
    double grossPrice, netPrice;//stores the total price of the user's order
    int scanValid = 0;//used to check if input from scanf is valid;

    while(exit == 0){
        printf("What would you like to do? (Insert a, b, c or q)\n");
        printf("a) Order artichokes \n");
        printf("b) Order onions \n");
        printf("c) Order carrots \n");
        printf("q) Check out \n");
        fflush(stdin);//flush the input buffer before taking user input to clear
the input stream of unwanted content
        scanf("%c", &response);//takes user's choice for the main menu
        if(response == 'a'){
            printf("How many kilos would you like to add to your order? \n");
            fflush(stdin);
            scanValid = scanf("%d", &tempa);
            if(tempa < 0 || scanValid!=1){
                printf("Invalid input, please insert a positive integer value \n");
                tempa = 0;
            }
            a += tempa;//adds the user's input to the total amount of artichokes to
be ordered
        }else if(response == 'b'){
            printf("How many kilos would you like to add to your order? \n");
            fflush(stdin);
            scanValid = scanf("%d", &tempo);
            if(tempo < 0 || scanValid!=1){
                printf("Invalid input, please insert a positive integer value \n");
                tempo = 0;
            }
            o += tempo;//adds the user's input to the total amount of onions to be
ordered
        }else if(response == 'c'){
            printf("How many kilos would you like to add to your order? \n");
            fflush(stdin);
            scanValid = scanf("%d", &tempc);
            if(tempc < 0 || scanValid!=1){
                printf("Invalid input, please insert a positive integer value \n");
                tempc = 0;
            }
            c += tempc;//adds the user's input to the total amount of carrots to be
ordered
        }else if(response == 'q'){//exits menu loop and proceeds to checkout
            printf("Are you sure you want to check out? (Type Y to check out, type
anything else to return to main menu)\n");
            fflush(stdin);
            scanf("%c", &checkout);//checkout confirmation
            if(checkout == 'Y'){
                exit = 1;
            }else{
                printf("Returning to main menu\n");
            }
        }else{//user input doesn't match one of the menu options
            printf("Invalid input, please try again. \n");
        }
    }

    totalWeight = a + o + c;//adds up the total amount of kilos, in the order, used
to calculate shipping cost
    if(totalWeight > 0){//no checkout if nothing is ordered
        double tap = a*2.05;//calculating total price of artichokes
        double top = o*1.15;//calculating total price of onions
        double tcp = c*1.19;//calculating total price of carrots
        grossPrice = tap + top + tcp;//calculating total price before discount and
shipping
        double bulkDiscount = 0;
        double shippingCost = 0;
        netPrice = grossPrice;//initializing net price with value of gross price
before deducting discount/adding shipping

        if(grossPrice >= 100){//calculating bulk discount if total price before
shipping exceeds 100 euro
            bulkDiscount = grossPrice*0.05;
            netPrice -= bulkDiscount;
        }
        if(totalWeight <= 5){//calculation of shipping costs
            shippingCost = 6.5;
            netPrice += shippingCost;
        }else if(totalWeight > 5 & totalWeight < 20){
            shippingCost = 14;
            netPrice += shippingCost;
        }else if(totalWeight >= 20) {
            shippingCost = 14 + 0.5 * totalWeight - 20;
            netPrice += shippingCost;
        }

        printf("Final Bill \n");//outputting the bill to the user
        printf("Artichokes: %d kilos, %.2lfe \n", a, tap);
        printf("Onions: %d kilos, %.2lfe \n", o, top);
        printf("Carrots: %d kilos, %.2lfe \n", c, tcp);
        printf("Gross Price: %d kilos, %.2lfe \n", totalWeight, grossPrice);
        printf("Shipping Cost: %.2lfe \n", shippingCost);
        if(bulkDiscount != 0)
            printf("Bulk Discount: -%.2lfe \n", bulkDiscount);
        printf("Net Price: %.2lfe \n", netPrice);

        fp=fopen("bill.txt", "w");//opening or creating the file where the bill is
to be saved
        fprintf(fp, "Final Bill \n");//printing the bill to a text file
        fprintf(fp, "Artichokes: %d kilos, %.2lfe \n", a, tap);
        fprintf(fp, "Onions: %d kilos, %.2lfe \n", o, top);
        fprintf(fp, "Carrots: %d kilos, %.2lfe \n", c, tcp);
        fprintf(fp, "Gross Price: %d kilos, %.2lfe \n", totalWeight, grossPrice);
        fprintf(fp, "Shipping Cost: %.2lfe \n", shippingCost);
        if(bulkDiscount != 0)
            fprintf(fp, "Bulk Discount: -%.2lfe\n", bulkDiscount);
        fprintf(fp, "Net Price: %.2lfe \n", netPrice);
    }

    return 0;//program termination

}
```

Testing



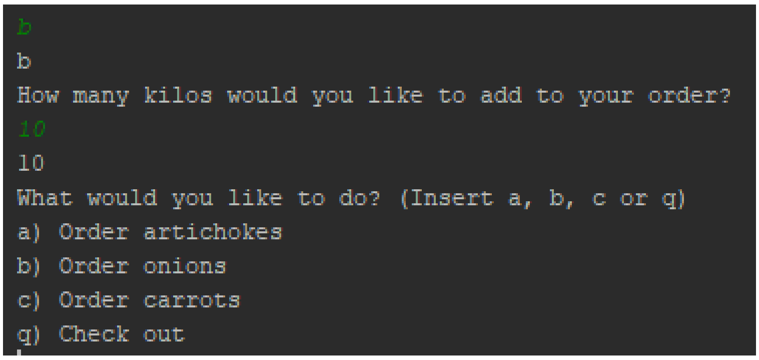
```
What would you like to do? (Insert a, b, c or q)
a) Order artichokes
b) Order onions
c) Order carrots
q) Check out
```

Figure 2 - Main Menu



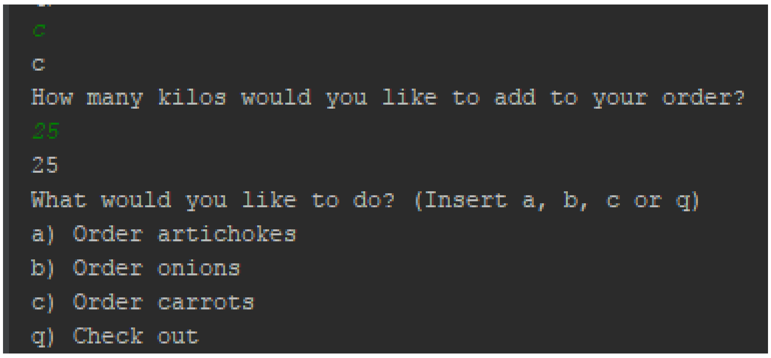
```
a
a
How many kilos would you like to add to your order?
5
5
What would you like to do? (Insert a, b, c or q)
a) Order artichokes
b) Order onions
c) Order carrots
q) Check out
```

Figure 3 - Adding artichokes to order



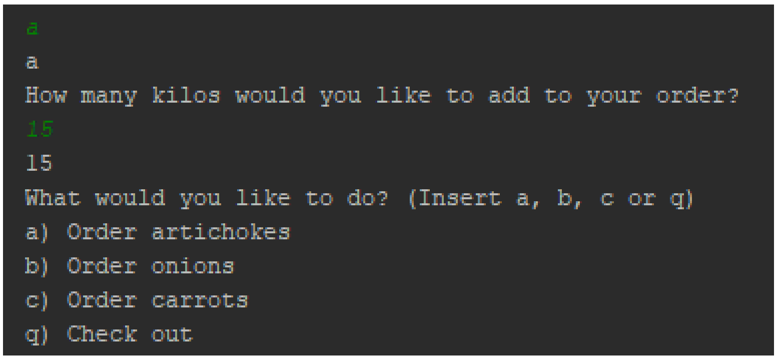
```
b
b
How many kilos would you like to add to your order?
10
10
What would you like to do? (Insert a, b, c or q)
a) Order artichokes
b) Order onions
c) Order carrots
q) Check out
```

Figure 4 - Adding onions to cart



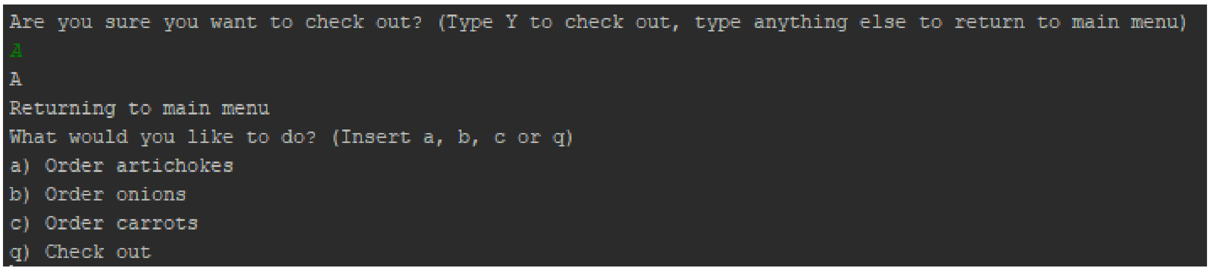
```
c
c
How many kilos would you like to add to your order?
25
25
What would you like to do? (Insert a, b, c or q)
a) Order artichokes
b) Order onions
c) Order carrots
q) Check out
```

Figure 5 - Adding carrots to cart



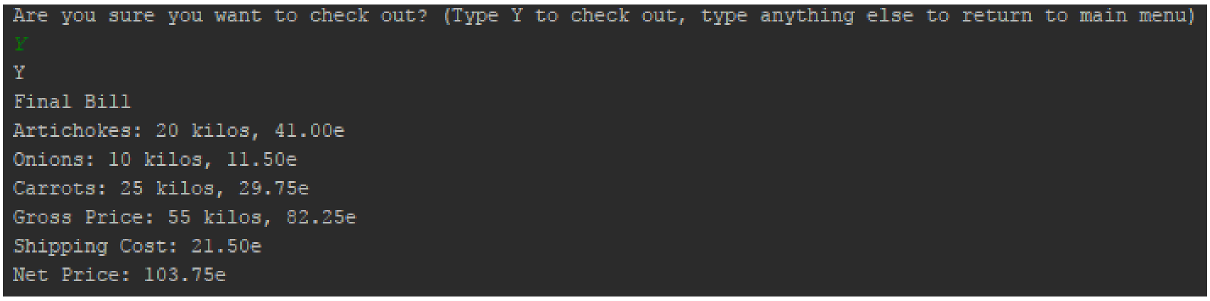
```
a
a
How many kilos would you like to add to your order?
15
15
What would you like to do? (Insert a, b, c or q)
a) Order artichokes
b) Order onions
c) Order carrots
q) Check out
```

Figure 6 - Adding more artichokes after having already ordered some



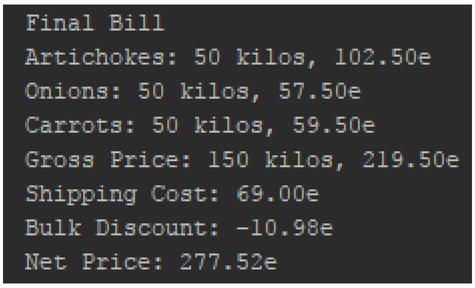
```
Are you sure you want to check out? (Type Y to check out, type anything else to return to main menu)
A
A
Returning to main menu
What would you like to do? (Insert a, b, c or q)
a) Order artichokes
b) Order onions
c) Order carrots
q) Check out
```

Figure 7 - Cancelling checkout



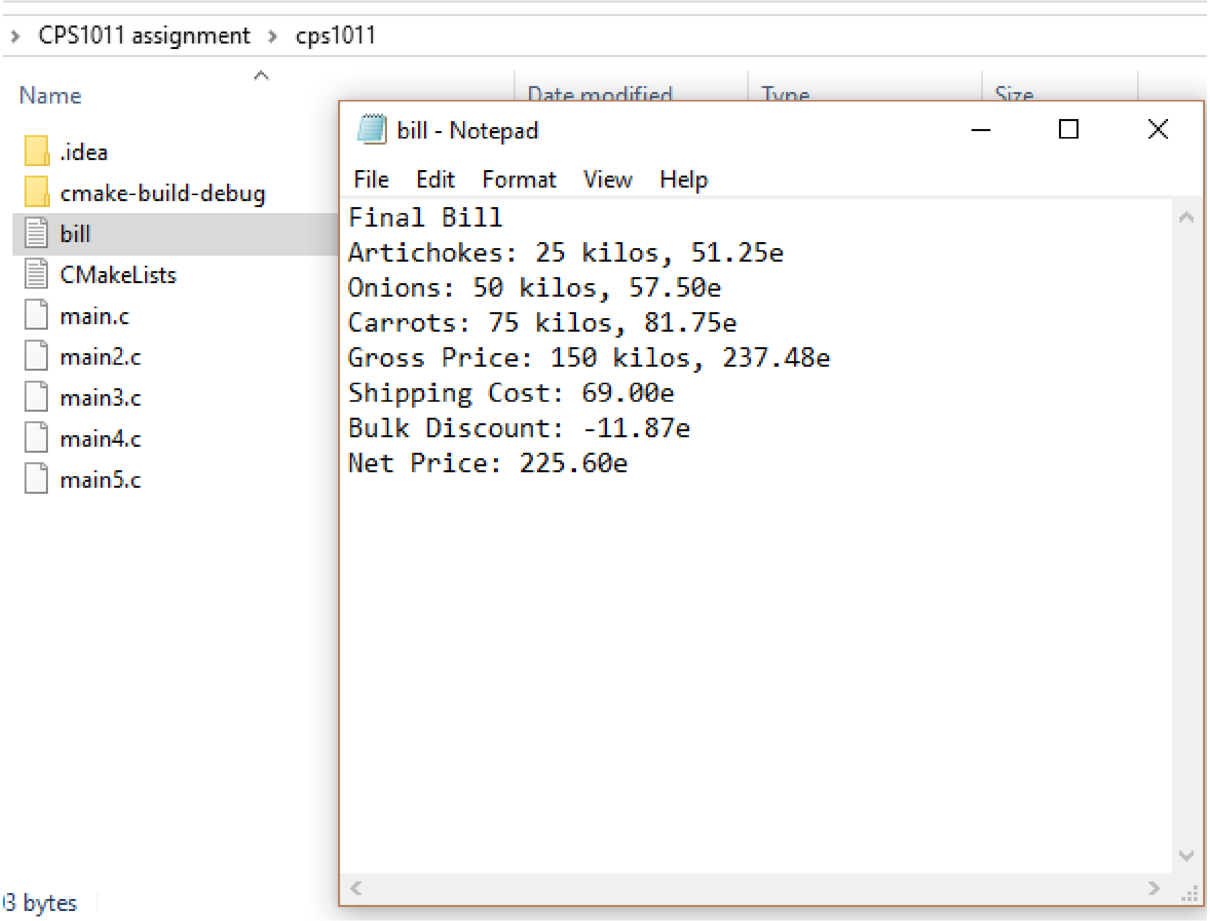
```
Are you sure you want to check out? (Type Y to check out, type anything else to return to main menu)
Y
Y
Final Bill
Artichokes: 20 kilos, 41.00e
Onions: 10 kilos, 11.50e
Carrots: 25 kilos, 29.75e
Gross Price: 55 kilos, 82.25e
Shipping Cost: 21.50e
Net Price: 103.75e
```

Figure 8 - Confirmation of checkout + bill without bulk discount



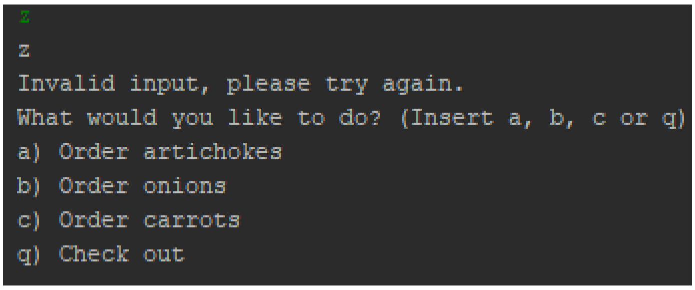
```
Final Bill
Artichokes: 50 kilos, 102.50e
Onions: 50 kilos, 57.50e
Carrots: 50 kilos, 59.50e
Gross Price: 150 kilos, 219.50e
Shipping Cost: 69.00e
Bulk Discount: -10.98e
Net Price: 277.52e
```

Figure 9 - Bill with bulk discount



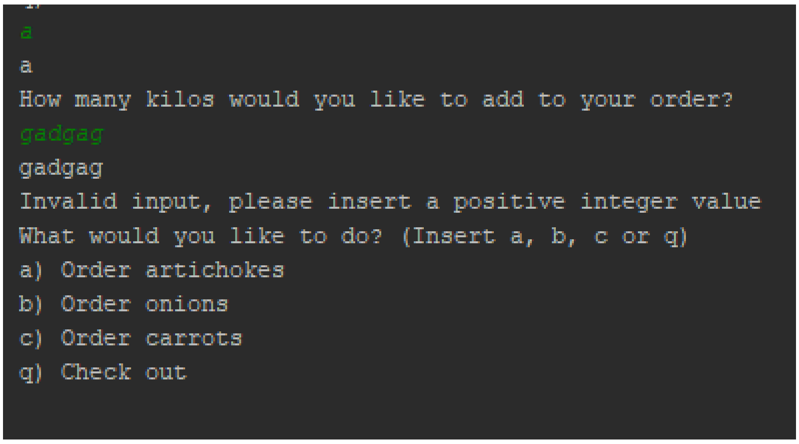
```
Final Bill
Artichokes: 25 kilos, 51.25e
Onions: 50 kilos, 57.50e
Carrots: 75 kilos, 81.75e
Gross Price: 150 kilos, 237.48e
Shipping Cost: 69.00e
Bulk Discount: -11.87e
Net Price: 225.60e
```

Figure 10 - Bill saved to text file in project directory



```
z
z
Invalid input, please try again.
What would you like to do? (Insert a, b, c or q)
a) Order artichokes
b) Order onions
c) Order carrots
q) Check out
```

Figure 11 - Handling of incorrect input into menu



```
a
a
How many kilos would you like to add to your order?
gadgag
gadgag
Invalid input, please insert a positive integer value
What would you like to do? (Insert a, b, c or q)
a) Order artichokes
b) Order onions
c) Order carrots
q) Check out
```

Figure 12 - Handling of incorrect input when adding to order

Explanation

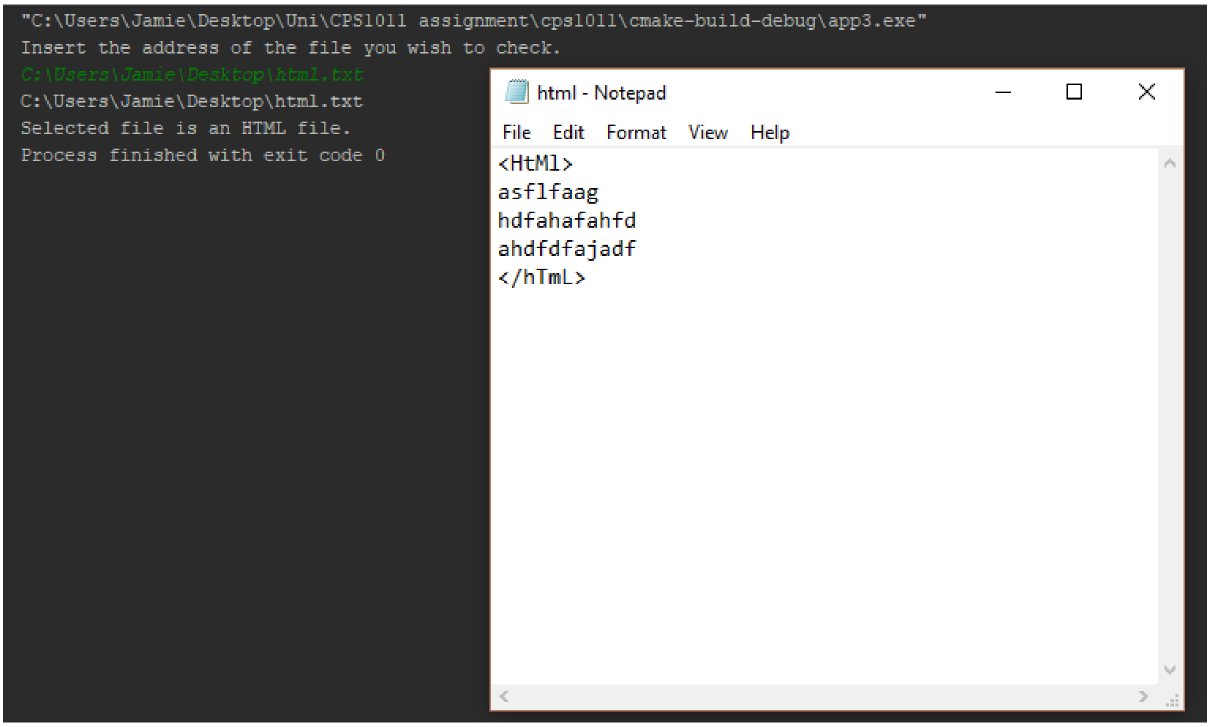
For this question, the majority of the code where user input is required is stored in a while loop. This is done so that the user will keep being redirected to the main menu after adding to his order until he confirms that he wishes to proceed to the bill. The input buffer is flushed before every user input so as to remove any unwanted content from the buffer. An if-else-if ladder is used for navigation of both the main menu and the shipping costs since only one of the options is to be used in both of these situations. The scanValid int is used to store the return value of the scanf functions which take the kilos to be added to the order for input. In the case that the user inputs an integer, as requested by the %d identifier, the scanf function stores the value inserted into the respective temporary integer and then adds it to the total weight for that particular product if the integer is positive. The file is saved to a text file named 'bill.txt' in the project directory, pointed to by the pointer fp.

Question 1C

Code

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

int main(){


    FILE *fP;
    char address[256];
    char textLine[150];
    char normalCase[150];
    char upperCase[150];
    printf("Insert the address of the file you wish to check.\n");
    scanf("%s", address);
    while((fP = fopen(address, "r")) == NULL){//note: spaces in file address cause
an error
        printf("Invalid address, please try again.\n");
        fflush(stdin);
        scanf("%s", address);
    }
    fP = fopen(address, "r");
    fgets(textLine, 150, fP);//gets the first line in the file
    strcpy(normalCase, textLine);//stores the first line in the file
    strcpy(upperCase, textLine);//stores the first line in the file
    while(!feof(fP)){
        fgets(textLine, 150, fP);//keeps going till it reaches the last line in the
file
    }
    for(int i = 0; i < strlen(upperCase); i++) {//converts one of the copies of the
first line in the file to uppercase so that a case insensitive comparison can be
made to check for an html file
        upperCase[i] = (char)toupper(upperCase[i]);
    }
    for(int i = 0; i < strlen(textLine); i++){//converts the last line in the file
to uppercase so that a case insensitive comparison can be made to check for an html
file
        textLine[i] = (char)toupper(textLine[i]);
    }
    if(strstr(normalCase,"#include")!=NULL) {//strstr used for case sensitive
string comparison
        printf("Selected file is a C File.");
    }else if(strstr(upperCase,"<HTML>")!=NULL && strstr(textLine, "</HTML>"))
{//case insensitive comparison for HTML file, comparing uppercase with uppercase
        printf("Selected file is an HTML file.");
    }else{
        printf("Selected file is neither a C file nor an HTML file");
    }

    return 0;

}
```

Testing

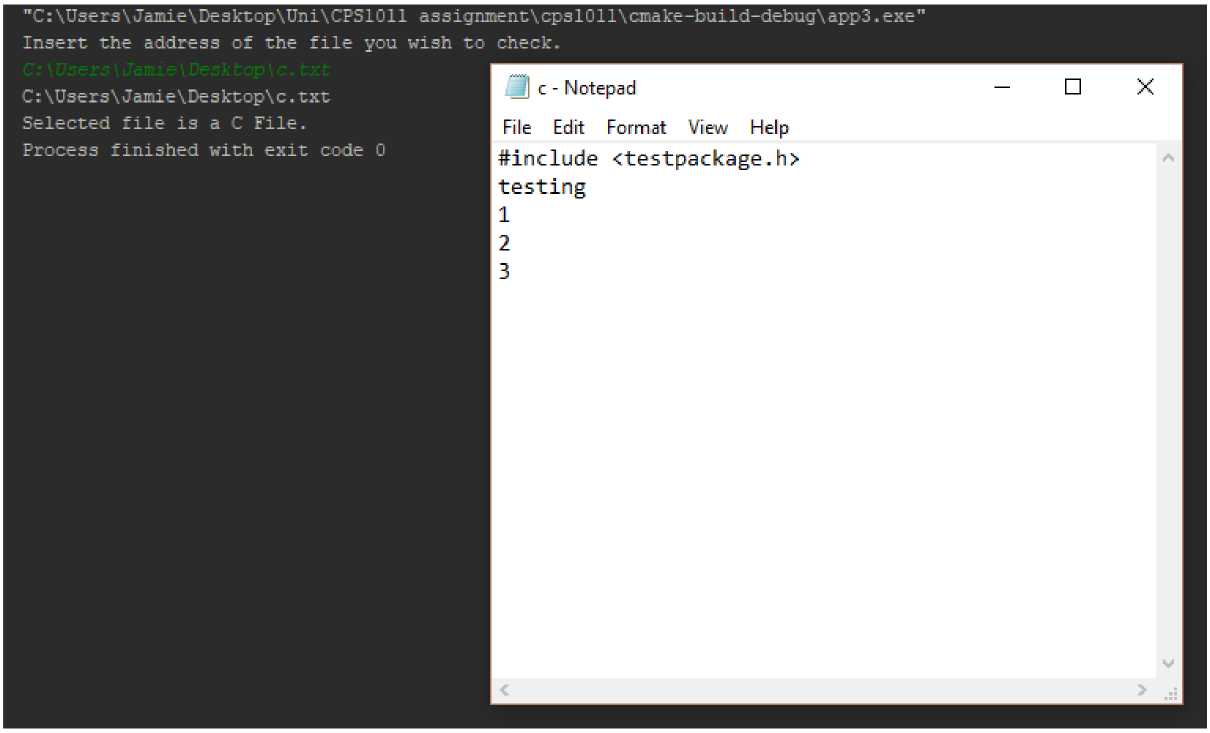


```
"C:\Users\Jamie\Desktop\Uni\CPS1011 assignment\cps1011\cmake-build-debug\app3.exe"
Insert the address of the file you wish to check.
C:\Users\Jamie\Desktop\html.txt
C:\Users\Jamie\Desktop\html.txt
Selected file is an HTML file.
Process finished with exit code 0
```

html - Notepad

```
<HtMl>
asflfaag
hdfahafahfd
ahdfdfajadf
</hTmL>
```

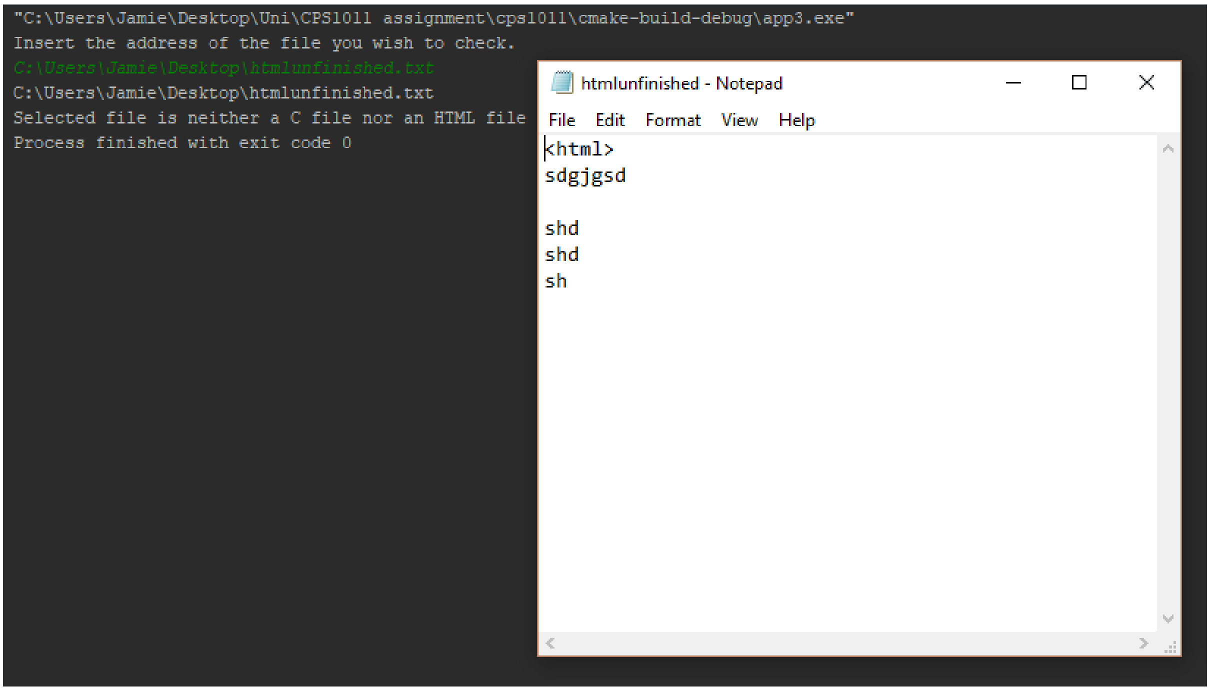
Figure 13 - Detecting HTML file



```
"C:\Users\Jamie\Desktop\Uni\CPS1011 assignment\cps1011\cmake-build-debug\app3.exe"
Insert the address of the file you wish to check.
C:\Users\Jamie\Desktop\c.txt
C:\Users\Jamie\Desktop\c.txt
Selected file is a C File.
Process finished with exit code 0
```

c - Notepad

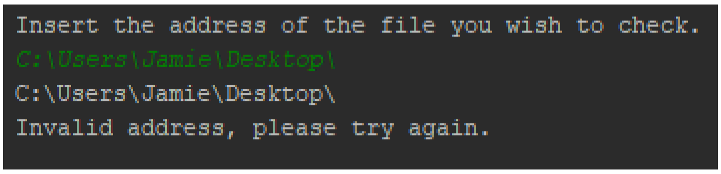
```
#include <testpackage.h>
testing
1
2
3
```



```
"C:\Users\Jamie\Desktop\Uni\CPS1011 assignment\cps1011\cmake-build-debug\app3.exe"
Insert the address of the file you wish to check.
C:\Users\Jamie\Desktop\htmlunfinished.txt
C:\Users\Jamie\Desktop\htmlunfinished.txt
Selected file is neither a C file nor an HTML file
Process finished with exit code 0
```

```
<html>
sdgjgsd

shd
shd
sh
```

Figure 14 - File is neither C nor HTML



```
Insert the address of the file you wish to check.
C:\Users\Jamie\Desktop\
C:\Users\Jamie\Desktop\
Invalid address, please try again.
```

Figure 15 - Invalid address inserted

Explanation

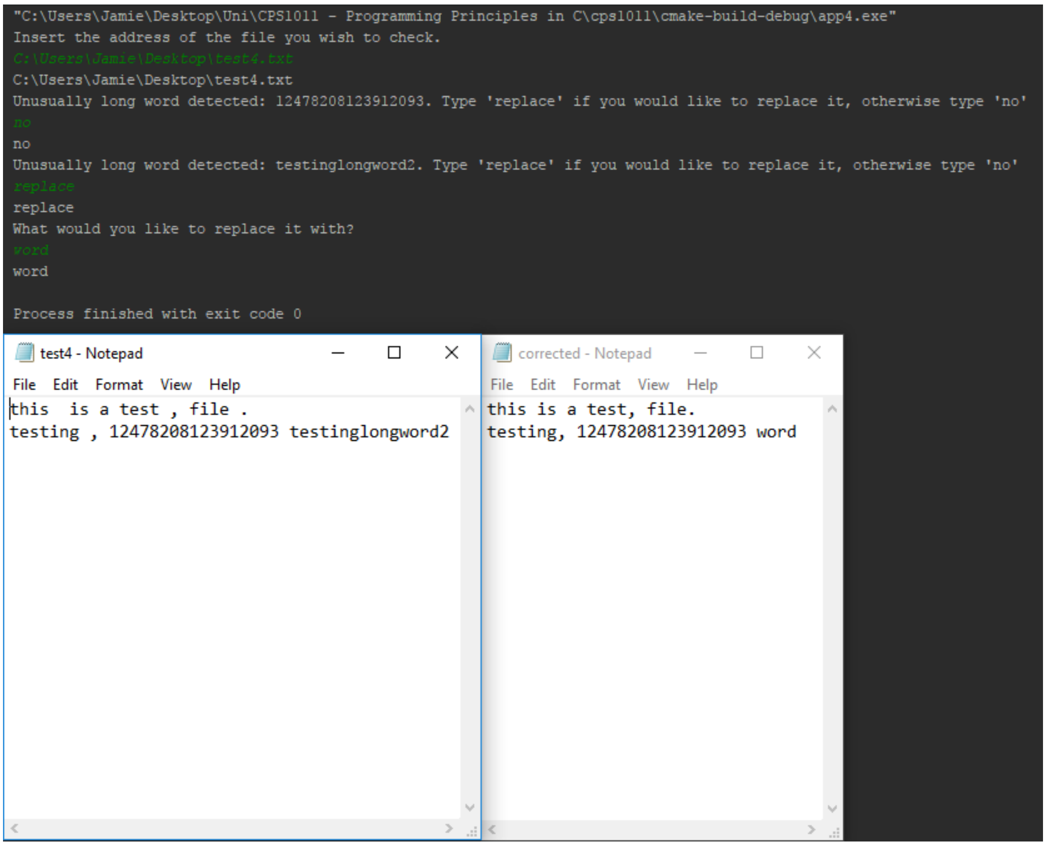
After the user inputs a valid address of a text file, the program reads the first line and stores two copies of it. One of the copies is kept as is while the other copy is converted to uppercase using the toupper function within a for loop which repeats for every character in the line. The program then proceeds through a while loop to find the last line which is kept in the texLine array, which is also converted to uppercase. The lowercase version of the first line is compared to the string #include and is classified as a C file if it contains #include. This has a margin of error since if #include isn't the first thing in the line, it is still classified as a C file. If the file is not classified as a c file, it checks if the uppercase versions of the first and last line contain <HTML> and </HTML> respectively. The html tags in the file are not case sensitive since the program converts the respective lines to uppercase before the comparison. If the program finds that none of these conditions are met by the given text file, it displays "Selected file is neither a C file nor an HTML file"

Question 1D

Code

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

char* replace(char *sentence, char *toBeReplaced, char *replacementWord){

    char* newSentence;
    int counter = 0;
    size_t toBeReplacedLen, replacementLen;
    toBeReplacedLen = strlen(toBeReplaced);
    replacementLen = strlen(replacementWord);
    int i;//counter to represent length of old sentence

    for(i = 0; sentence[i] != '\0'; i++){//checking how many times the string to be replaced appears in the line
        if(strstr(&sentence[i], toBeReplaced) == &sentence[i]){
            counter++;
            i += toBeReplacedLen - 1;//skip the instance of the string to be replaced which was already counted
        }
    }

    newSentence = (char*)malloc(i + counter * (replacementLen - toBeReplacedLen) + 1);//allocating memory according to the length of the new sentence (+1 for end of string character)

    i = 0;

    while(*sentence){//goes through the old sentence and replaces all instances of the desired string, placing the result in newSentence
        if(strstr(sentence, toBeReplaced) == sentence){//checks if the char pointer has reached one of the instances to be replaced
            strcpy(&newSentence[i], replacementWord);//
            i += replacementLen;
            sentence += toBeReplacedLen;
        }else{
            newSentence[i++] = *sentence++;
        }
    }
    newSentence[i] = '\0';
    return newSentence;
}

char * replaceLongWord(char *sentence){

    size_t letterCounter = 1;
    char* longWord;
    char replacementString[50];
    char replaceConfirmation[50];
    for(int i = 0; sentence[i] != '\0'; i++){//checking how many times the string to be replaced appears in the line
        if(sentence[i] != ' ' && sentence[i] != '-' && sentence[i+1] != ' ' && sentence[i+1] != '-' && sentence[i+1] != '\0') {
            letterCounter++;
        }else if(letterCounter > 12){//if a space is found but the letter counter is greater than 12, enter replacement case
            longWord = (char*)malloc(letterCounter * sizeof(char) + 1);//allocating memory for word to be replaced
            for(int x = 0; x <= letterCounter; x++){
                longWord[x] = sentence[i - (letterCounter-x)];//retrieving word to be replaced from sentence
            }
            longWord[letterCounter + 1] = '\0';//appending end of string character to the word to be replaced
            printf("Unusually long word detected: %s. Type 'replace' if you would 
like to replace it, otherwise type 'no'\n", longWord);
            fflush(stdin);
            scanf("%s", replaceConfirmation);//prompting user for input
            while(strcmp(replaceConfirmation, "replace") != 0 && 
strcmp(replaceConfirmation, "no") != 0){//user input validation
                printf("Invalid input, please type 'replace' or 'no'\n");
                fflush(stdin);
                scanf("%s", replaceConfirmation);
            }
            if(strcmp(replaceConfirmation, "replace") == 0){
                printf("What would you like to replace it with?\n");
                fflush(stdin);
                scanf("%s", replacementString);
                sentence = replace(sentence, longWord, replacementString);
                i += strlen(replacementString) - strlen(longWord);//adjusting i 
according to changes made in length of sentence so as to not interrupt the loop for 
instances of other long words
                letterCounter = 0;
            }else if(strcmp(replaceConfirmation, "no") == 0){
                letterCounter = 0;
            }
        }else{
            letterCounter = 0;
        }
    }
    return sentence;//if no word
}

int main(){

    FILE *fP;
    FILE *correctedFP;
    char *textLine;
    char address[256];
    printf("Insert the address of the file you wish to check.\n");
    fflush(stdin);
    scanf("%s", address);
    while((fP = fopen(address, "r")) == NULL){//note: spaces in file address cause 
an error
        printf("Invalid address, please try again.\n");
        fflush(stdin);
        scanf("%s", address);
    }
    fP = fopen(address, "r");
    correctedFP = fopen("corrected.txt", "w");
    while(!feof(fP)){
        textLine = (char*)malloc(256 * sizeof(char));
        fgets(textLine, 256, fP);
        textLine = replace(textLine, " .", ".");
        textLine = replace(textLine, " ,", ",");
        textLine = replace(textLine, "  ", " ");
        textLine = replaceLongWord(textLine);
        textLine = replace(textLine, " .", ".");
        textLine = replace(textLine, " ,", ",");
        textLine = replace(textLine, "  ", " ");
        fputs(textLine, correctedFP);
    }

    return 0;

}
```

Testing

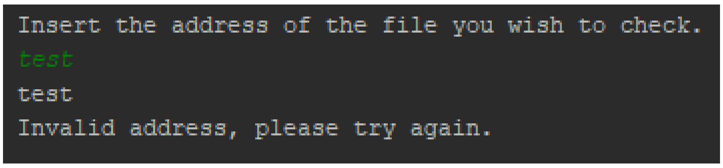


```
"C:\Users\Jamie\Desktop\Uni\CPS1011 - Programming Principles in C\cps1011\cmake-build-debug\app4.exe"
Insert the address of the file you wish to check.
C:\Users\Jamie\Desktop\test4.txt
C:\Users\Jamie\Desktop\test4.txt
Unusually long word detected: 12478208123912093. Type 'replace' if you would like to replace it, otherwise type 'no'
no
no
Unusually long word detected: testinglongword2. Type 'replace' if you would like to replace it, otherwise type 'no'
replace
replace
What would you like to replace it with?
word
word

Process finished with exit code 0
```

```
this  is a test , file .
testing , 12478208123912093 testinglongword2
```

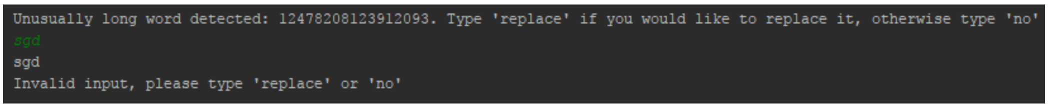
```
this is a test, file.
testing, 12478208123912093 word
```

Figure 16 - Program running successfully



```
Insert the address of the file you wish to check.
test
test
Invalid address, please try again.
```

Figure 17 - Invalid address inputted



```
Unusually long word detected: 12478208123912093. Type 'replace' if you would like to replace it, otherwise type 'no'
sgd
sgd
Invalid input, please type 'replace' or 'no'
```

Figure 18 - Invalid input when replacing long word

Explanation

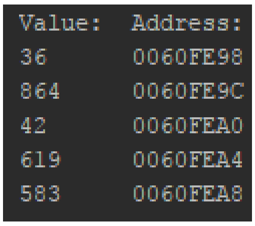
The first function in the program is replace. It takes 3 character pointers as parameters. These are the sentence to be searched, the word to be replaced and the word to replace it with. The function first counts how many instances of the word to be replaced are in the sentence. It then takes this counter, along with the length of the string to be replaced, the length of the string to replace it with and the length of the original sentence to allocate memory for the original sentence. It then goes through the sentence again and replaces all the instances. The second function is the replaceLongWord. It reads a sentence passed as a parameter and then checks for instances of more than 12 consecutive characters without a space or hyphen between them. When one such word is found it asks the user whether they wish to replace it. If they type 'no' it proceeds to check the rest of the sentence. If they type 'replace' it asks the user what they would like to replace the word with, then takes the input, adjusts the sentence and the counter controlling the for loop accordingly to continue checking the line. The main method simply takes an address as input from the user and keeps looping until a valid address to a text file is given. When a valid text file is given, it checks it for extra spaces and unusually long words line-by-line until the end of the file is reached, and the corrected file is saved in the project directory.

Question 1E

Code

```
#include <stdio.h>

void view_stack_frame(int array[], size_t length)
{
    printf("Value:\tAddress:\n");
    for(int i = 0; i < length; i++)
    {
        printf("%d\t%p\n", array[i], &array[i]);
    }
}

int main(){

    int numbers[] = {36, 864, 42, 619, 583};
    size_t numbersSize = sizeof(numbers)/sizeof(int);
    view_stack_frame(numbers, numbersSize);

    return 0;
}
```

Testing



```
Value:  Address:
36      0060FE98
864     0060FE9C
42      0060FEA0
619     0060FEA4
583     0060FEA8
```

Figure 19 - Printing of values and their respective memory address

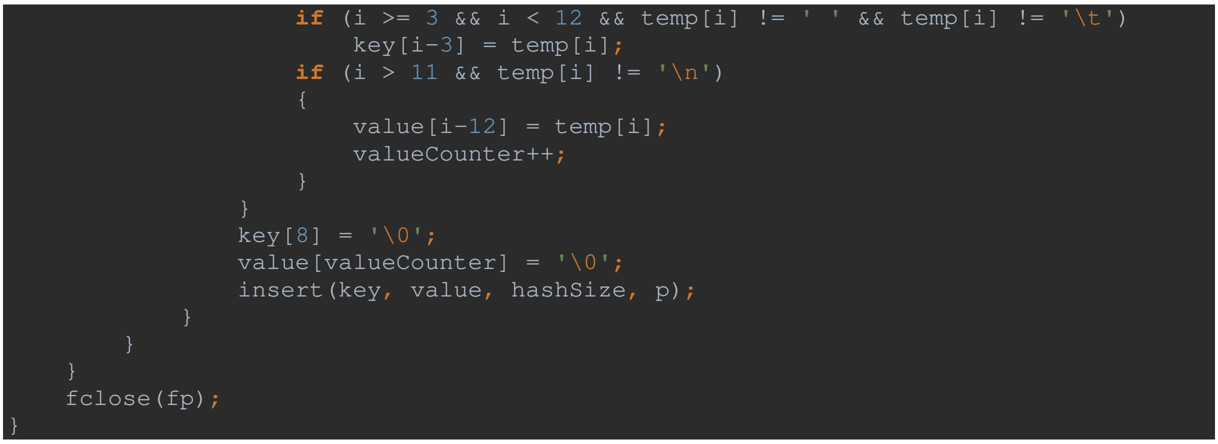
Explanation

For simplicity's sake, the given function is assuming that all variables in the stack frame are of type int. The function takes the variables of the stack frame in the form of an array as well as the size of said array as inputs. The function then goes through the array using a for loop and prints out the value of the integer (%d) and the pointer address of where the integer is stored on the stack (%p – Pointer address). In the given program, an array 'numbers' of type int is used to test out the function. Its size is calculated by using sizeof to find its size in bytes then dividing it by the size of an integer in bytes, and then passed along with the array itself as parameters to the function.

Question 2A

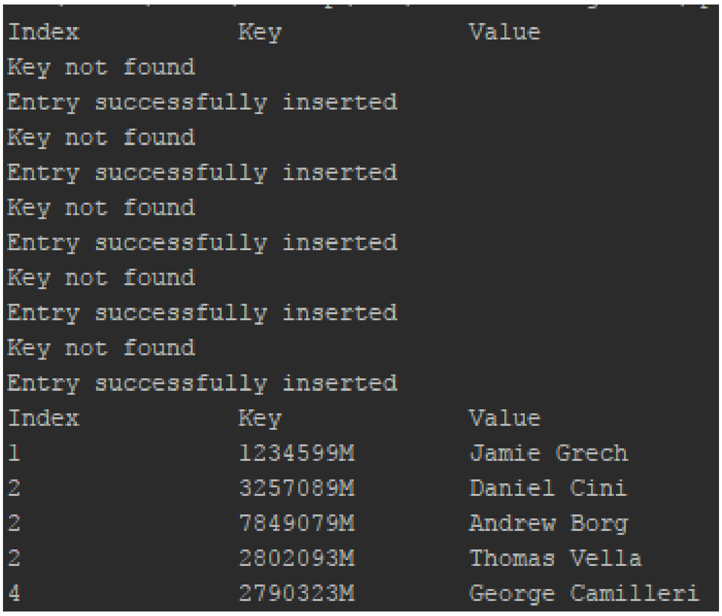
Code

```
#include "hashtable.h"

typedef struct hashTable//defining the properties we need for our table
{
    size_t maxCollisions;
    size_t usedCollisions;
    pair array[5];//store the collisions for every hash index
}hashTable;

void * createTable(size_t hashSize)
{
    hashSize = 5;//hash space is predefined
    hashTable * p = (hashTable*)malloc(hashSize * sizeof(hashTable));//memory is
allocated for 5 rows of 5 collisions since the max collisions and hash space are
predefined
    return p;
}

void initializeTable(size_t hashSize, void * p)//initializes all the keys and
values as empty strings
{
    hashSize = 5;//hash space is predefined
    hashTable * start = (hashTable*)p;//typecasts the void pointer to a pointer of
type hashtable
    for(int x = 0; x < hashSize; x++)
    {
        start[x].maxCollisions = 5;//the maximum collisions for every hash index is
5
        start[x].usedCollisions = 0;//initially there are no collisions since no
entries are inputted
        for(int i = 0; i < 5; i++)
        {
            strcpy(start[x].array[i].key, "\0");
            strcpy(start[x].array[i].value, "\0");
        }
    }
}

int hashFunction(char key[9], size_t hashSize){//uses the key of an entry to
calculate the index where it should be stored
    hashSize = 5;//hash space is predefined
    int keyLength = strlen(key);
    int sum =0;
    for(int c = 0; c<keyLength; c++)
    {
        sum += key[c];
    }
    sum = sum%hashSize;//the algorithm consists of adding the ASCII value value of
every character and calculating its mod by the hash space size
    return sum;
}

int lookUp(char key[9], size_t hashSize, void * p)
{
    hashSize = 5;//hash space is predefined
    hashTable * start = (hashTable*)p;
    int hash = hashFunction(key, hashSize);//find the hash index of the key to
search for
    for(int i = 0; i < start[hash].maxCollisions; i++)//searches all the collisions
for that hash index
    {
        if(strcmp(start[hash].array[i].key, key) == 0)//key inputted must be an
exact match with a key in the table
        {
            printf("Key found at location [%d][%d]\n", hash, i);
            return i;
        }
    }
    printf("Key not found\n");
    return -1;
}

int insert(char key[9], char value[25], size_t hashSize, void * p)//inserts new
entry into the table
{
    hashTable * start = (hashTable*)p;
    hashSize = 5;//hash space is predefined
    int hashValue = hashFunction(key, hashSize);//uses key and hash size to
calculate index
    if(strlen(key)!= 8)//checks if key is of appropriate size
    {
        printf("Key must be 8 characters long\n");
        return -1;
    }
    if(key == "" || value == "")//checks that neither the key nor the value or
empty
    {
        printf("Key or value cannot be blank\n");
        return -1;
    }
    if(lookUp(key, hashSize, p) != -1)//checks if the key is unique
    {
        printf("Key is not unique\n");
        return -1;
    }
    if(start[hashValue].maxCollisions == start[hashValue].usedCollisions)//checks
if the maximum collisions for the particular index has been reached
    {
        printf("Maximum collisons for hash value %d reached\n", hashValue);
        return -1;
    }
    for(int i = 0; i < start[hashValue].maxCollisions; i++){
        if(start[hashValue].array[i].key[0]=='\0')
        {
            strcpy(start[hashValue].array[i].key, key);
            strcpy(start[hashValue].array[i].value, value);
            printf("Entry successfully inserted\n");
            start[hashValue].usedCollisions += 1;
            i = start[hashValue].maxCollisions+1;
            return 1;
        }
    }
}

int delete(char key[9], size_t hashSize, void * p)//searches for a given key and
deletes the entry associated with it
{
    hashSize = 5;//hash space is predefined
    hashTable * start = (hashTable*)p;
    int hashValue = hashFunction(key, hashSize);//finds the hash index for the
given key
    int position = lookUp(key, hashSize, p);//finds the position of the key in the
respective array if it already exists
    if(position == -1)
    {
        return -1;//if the key doesn't already exist, exit the function
    }else{//if the key is found, replace it and the respective value with empty
strings and deduct the used collisions for that hash value by 1
        strcpy(start[hashValue].array[position].key, "\0");
        strcpy(start[hashValue].array[position].value, "\0");
        start[hashValue].usedCollisions-=1;
        printf("Key successfully deleted\n");
        return 1;
    }
}

void printContents(size_t hashSize, void * p)
{
    hashSize = 5;//hash space is predefined
    hashTable * start = (hashTable*)p;
    printf("Index\t\tKey\t\tValue\n");
    for(int i = 0; i < hashSize; i++)
    {
        for(int collisionCounter = 0; collisionCounter < start[i].maxCollisions; collisionCounter++){
            if(start[i].array[collisionCounter].key[0]!='\0')//prints all elements which are not empty
                printf("%d\t\t%s\t%s\n", i, start[i].array[collisionCounter].key, start[i].array[collisionCounter].value);
        }
    }
}

void saveFile(size_t hashSize, void * p)//same as printContents but prints to file
{
    hashSize = 5;//hash space is predefined
    hashTable * start = (hashTable*)p;
    FILE *fp = fopen("hashPartA.txt", "w");
    if (fp == NULL){
        printf("Error opening file!\n");
    }else{
        fprintf(fp, "Index\t\tKey\t\tValue\n");
        for(int i = 0; i < hashSize; i++)
        {
            for(int collisionCounter = 0; collisionCounter < start[i].maxCollisions; collisionCounter++){
                if(start[i].array[collisionCounter].key[0]!='\0')
                    fprintf(fp, "%d\t\t%s\t%s\n", i, start[i].array[collisionCounter].key, start[i].array[collisionCounter].value);
            }
        }
        printf("File successfully saved\n");
    }
    fclose(fp);
}

void readFromFile(size_t hashSize, void * p)
{
    hashSize = 5;//hash space is predefined
    FILE *fp = fopen("hashPartA.txt", "r");
    if (fp == NULL)
    {
        printf("Error opening file!\n");
    }else{
        initializeTable(hashSize, p);//empties any elements which currently contain data to override them with the data in the given file
        char temp[100];
        char key[9];
        char value[25];
        int counter = 0;//used so that data is not read from the first line which is a header
        int valueCounter;
        while(!feof(fp))
        {
            valueCounter = 0;//used to find the length of the value loaded so as to add a \0 to the end of it
            fgets(temp, 100, fp);
            counter++;
            if(!feof(fp) && counter > 1) {
                for (int i = 0; i < strlen(temp); i++) {
```



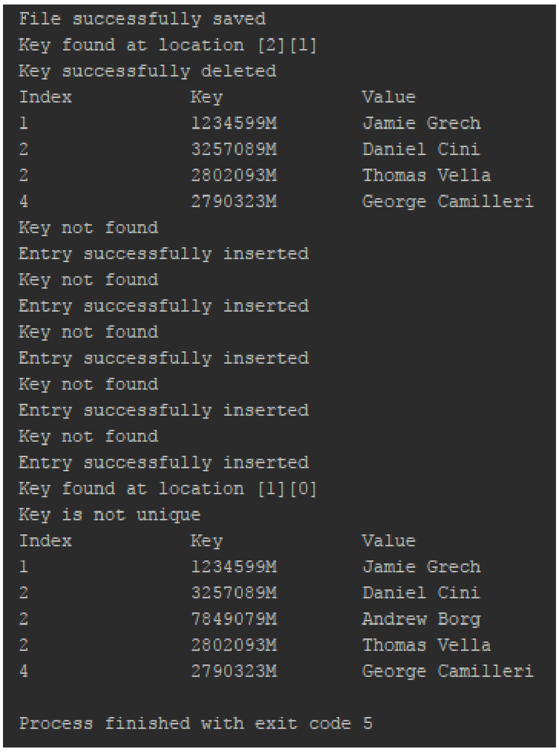
```c
                    if (i >= 3 && i < 12 && temp[i] != ' ' && temp[i] != '\t')
                        key[i-3] = temp[i];
                    if (i > 11 && temp[i] != '\n')
                    {
                        value[i-12] = temp[i];
                        valueCounter++;
                    }
                }
                key[8] = '\0';
                value[valueCounter] = '\0';
                insert(key, value, hashSize, p);
            }
        }
    }
    fclose(fp);
}
```

Testing



```
Index           Key             Value
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Index           Key             Value
1               1234599M        Jamie Grech
2               3257089M        Daniel Cini
2               7849079M        Andrew Borg
2               2802093M        Thomas Vella
4               2790323M        George Camilleri
```

Figure 20 - Printing empty list, adding 5 values to list then printing list with new values



```
File successfully saved
Key found at location [2][1]
Key successfully deleted
Index           Key             Value
1               1234599M        Jamie Grech
2               3257089M        Daniel Cini
2               2802093M        Thomas Vella
4               2790323M        George Camilleri
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key found at location [1][0]
Key is not unique
Index           Key             Value
1               1234599M        Jamie Grech
2               3257089M        Daniel Cini
2               7849079M        Andrew Borg
2               2802093M        Thomas Vella
4               2790323M        George Camilleri

Process finished with exit code 5
```

Figure 21 - Saving the list, deleting an entry then reloading the list before the key was deleted and printing it again

Explanation

To start off, a struct pair is defined containing a two character strings, a key and a value. This is because every element in the table is to contain a key-value and not a single character string. The key is mean to be a unique identifier of an entry while the value is not necessarily unique. In the test case shown above, keys are a typical Maltese I.D. number while values contain a name and surname. For simplicity's sake, keys are restricted to exactly 8 characters in length not including the end of string character while values are restricted to 24 or less characters not including the end of string character. Another struct is defined, called hashTable, which contains two unsigned integer values usedCollisions and maxCollisions as well as a one-dimensional array with 5 elements. Our actual hash table is to be represented by a pointer of this struct, which acts as a two dimensional array. When the createTable function is called, memory for a hashTable struct is allocated for every row in the hash table, also known as the hash space size. A void pointer is passed as a parameter to every function that directly interacts with the table so as to pass the pointer representing the table through as a parameter. The reason it is a void pointer and not a hashTable pointer is so that a struct with a different name can be used in questions b and c while still allowing for one test driver program to test all the different versions in part d. For every function where the hash size is passed as a parameter, the hash size is immediately set to 5 since the hash space is meant to be predefined in this question. Similarly, when the maximum collisions for a particular index are reached, instead of more memory being allocated to that index thus increasing the maximum possible collisions, the program simply prints that the maximum collisions for that index has been reached. When printing the contents and/or saving them to a file, the respective functions first check if the respective element is blank before printing it, so as not to unnecessarily print empty elements. When reading from the file, the key and value read are appended with an end of string character before being re inserted into the table, as well as the table being re-initialized so as to completely overwrite any data in the table before the loading occurs thus minimizing errors or unwanted behaviour.

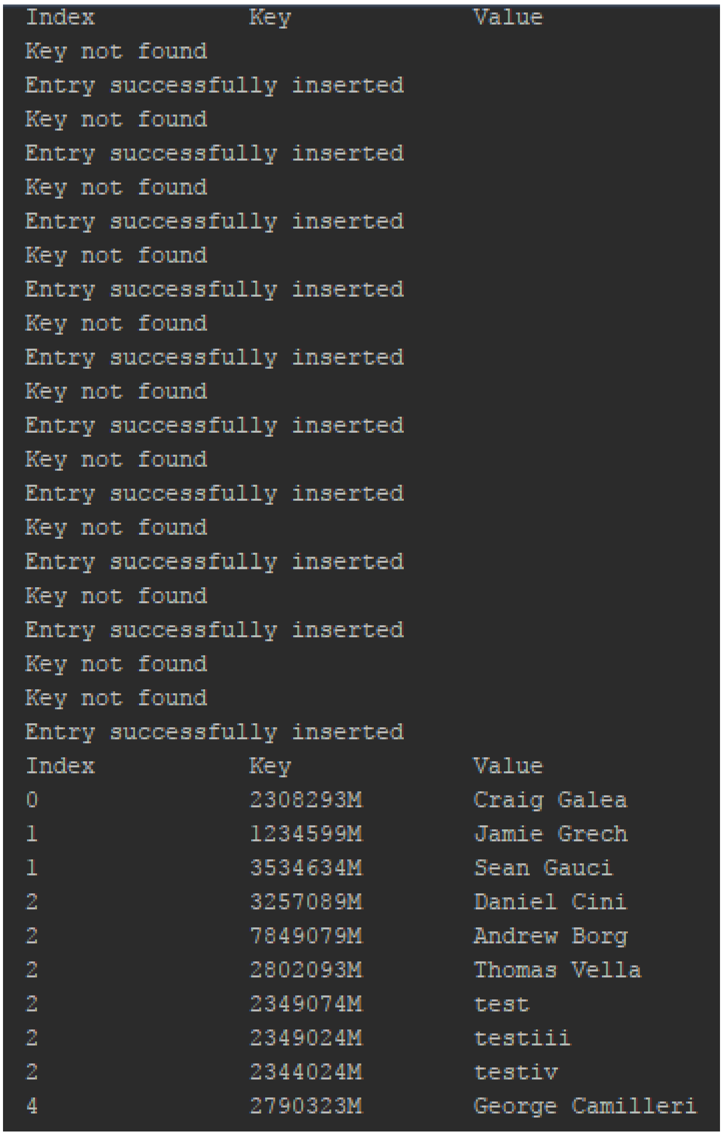
Question 2B

Code

```c
#include "hashtable.h"

typedef struct hashTable
{
    size_t maxCollisions;
    size_t usedCollisions;
    pair * array;//using a pointer since max collisions are not fixed
}hashTable;

void * createTable(size_t hashSize)
{
    hashTable * p = (hashTable*)malloc(hashSize * sizeof(hashTable));//allocating memory for the rows
    for(int i = 0; i < hashSize; i++)
    {
        p[i].array = (pair*)malloc(5 * sizeof(pair));//allocating memory for the columns/collisions, initially 5 columns
        p[i].maxCollisions = 5;
        p[i].usedCollisions = 0;
    }
    return p;
}

void initializeTable(size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    for(int x = 0; x < hashSize; x++)
    {
        for(int i = 0; i < start[x].maxCollisions; i++)
        {
            strcpy(start[x].array[i].key, "\0");
            strcpy(start[x].array[i].value, "\0");
        }
    }
}

int hashFunction(char key[9], size_t hashSize)
{
    int keyLength = strlen(key);
    int sum =0;
    for(int c = 0; c<keyLength; c++)
    {
        sum += key[c];
    }
    sum = sum%hashSize;
    return sum;
}

int lookUp(char key[9], size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    int hash = hashFunction(key, hashSize);
    for(int i = 0; i < start[hash].maxCollisions; i++)
    {
        if(strcmp(start[hash].array[i].key, key) == 0)
        {
            printf("Key found at location [%d][%d]\n", hash, i);
            return i;
        }
    }
    printf("Key not found\n");
    return -1;
}
```

```c
int insert(char key[9], char value[25], size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    int hashValue = hashFunction(key, hashSize);
    if(strlen(key)!= 8)
    {
        printf("Key must be 8 characters long\n");
        return -1;
    }
    if(key == "" || value == "")
    {
        printf("Key or value cannot be blank\n");
        return -1;
    }
    if(lookUp(key, hashSize, p) != -1)
    {
        printf("Key is not unique\n");
        return -1;
    }
    if(start[hashValue].maxCollisions == start[hashValue].usedCollisions)//if an entry is to be inserted to a full index, memory is allocated for one more collision at that index
    {
        start[hashValue].maxCollisions += 1;
        start[hashValue].array = (pair*)realloc((start[hashValue].array), (start[hashValue].maxCollisions)*sizeof(pair));
        strcpy(start[hashValue].array[start[hashValue].maxCollisions-1].key, "\0");//initializing newly allocated memory
        strcpy(start[hashValue].array[start[hashValue].maxCollisions-1].key, "\0");
        return insert(key, value, hashSize, p);//attempt to insert the entry again after allocating more memory for that index
    }
    for(int i = 0; i <  start[hashValue].maxCollisions; i++)
    {
        if(start[hashValue].array[i].key[0]=='\0')
        {
            strcpy(start[hashValue].array[i].key, key);
            strcpy(start[hashValue].array[i].value, value);
            printf("Entry successfully inserted\n");
            start[hashValue].usedCollisions += 1;
            return 1;
        }
    }
}

int delete(char key[9], size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    int hashValue = hashFunction(key, hashSize);
    int position = lookUp(key, hashSize, p);
    if(position == -1)
    {
        return -1;
    }else{
        strcpy(start[hashValue].array[position].key, "\0");
        strcpy(start[hashValue].array[position].value, "\0");
        start[hashValue].usedCollisions-=1;
        printf("Key successfully deleted\n");
        return 1;
    }
}

void printContents(size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    printf("Index\t\tKey\t\tValue\n");
    for(int i = 0; i < hashSize; i++)
    {
        for(int collisionCounter = 0; collisionCounter < start[i].maxCollisions;
collisionCounter++){
            if(start[i].array[collisionCounter].key[0]!='\0')
                printf("%d\t\t%s\t%s\n", i, start[i].array[collisionCounter].key,
start[i].array[collisionCounter].value);
        }
    }
}

void saveFile(size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    FILE *fp = fopen("hashPartB.txt", "w");
    if (fp == NULL)
    {
        printf("Error opening file!\n");
    }else{
        fprintf(fp, "Index\t\tKey\t\tValue\n");
        for(int i = 0; i < hashSize; i++)
        {
            for(int collisionCounter = 0; collisionCounter <
start[i].maxCollisions; collisionCounter++){
                if(start[i].array[collisionCounter].key[0]!='\0')
                    fprintf(fp, "%d\t\t%s\t%s\n", i,
start[i].array[collisionCounter].key, start[i].array[collisionCounter].value);
            }
        }
        printf("File successfully saved\n");
    }
    fclose(fp);
}

void readFromFile(size_t hashSize, void * p)
{
    initializeTable(hashSize, p);
    FILE *fp = fopen("hashPartB.txt", "r");
    if (fp == NULL)
    {
        printf("Error opening file!\n");
    }else{
        char temp[100];
        char key[9];
        char value[25];
        int counter = 0;
        int valueCounter;
        while(!feof(fp))
        {
            valueCounter = 0;
            fgets(temp, 100, fp);
            counter++;
            if(!feof(fp) && counter > 1) {
                for (int i = 0; i < strlen(temp); i++) {
                    if (i >= 3 && i < 12 && temp[i] != ' ' && temp[i] != '\t')
                        key[i-3] = temp[i];
                    if (i > 11 && temp[i] != '\n')
                    {
                        value[i-12] = temp[i];
                        valueCounter++;
                    }
                }
                key[8] = '\0';
                value[valueCounter] = '\0';
                insert(key, value, hashSize, p);
            }
        }
    }
    fclose(fp);
}
```

Testing



```
Index           Key             Value
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Key not found
Entry successfully inserted
Index           Key             Value
0               2308293M        Craig Galea
1               1234599M        Jamie Grech
1               3534634M        Sean Gauci
2               3257089M        Daniel Cini
2               7849079M        Andrew Borg
2               2802093M        Thomas Vella
2               2349074M        test
2               2349024M        testiii
2               2344024M        testiv
4               2790323M        George Camilleri
```

Figure 22 - Adding more than 5 entries to the same hash causing memory to be allocated for more collisions

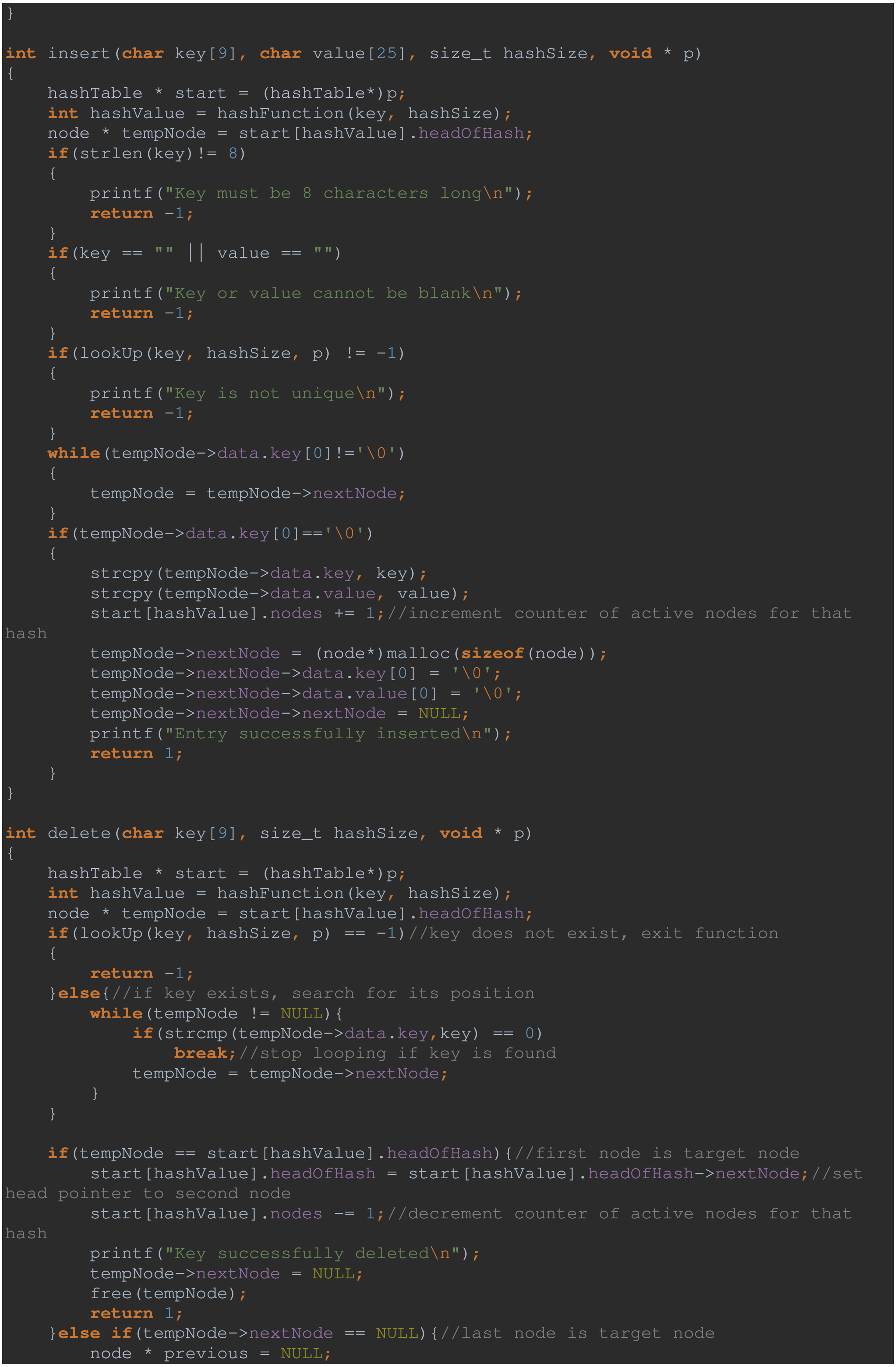
Explanation

Part 2b is very similar to 2a. The first difference is that the pair array in the hashTable definition is replaced with a pair pointer since the collisions are now only limited by system memory availability. This is put into practice in the insert method where if max collisions for a particular hash value are reached, instead of displaying an error message, more memory is allocated for a new entry using realloc and the entry is initialized with empty strings. After this, the function recursively calls itself to attempt to insert the entry into the newly allocated slot in memory.

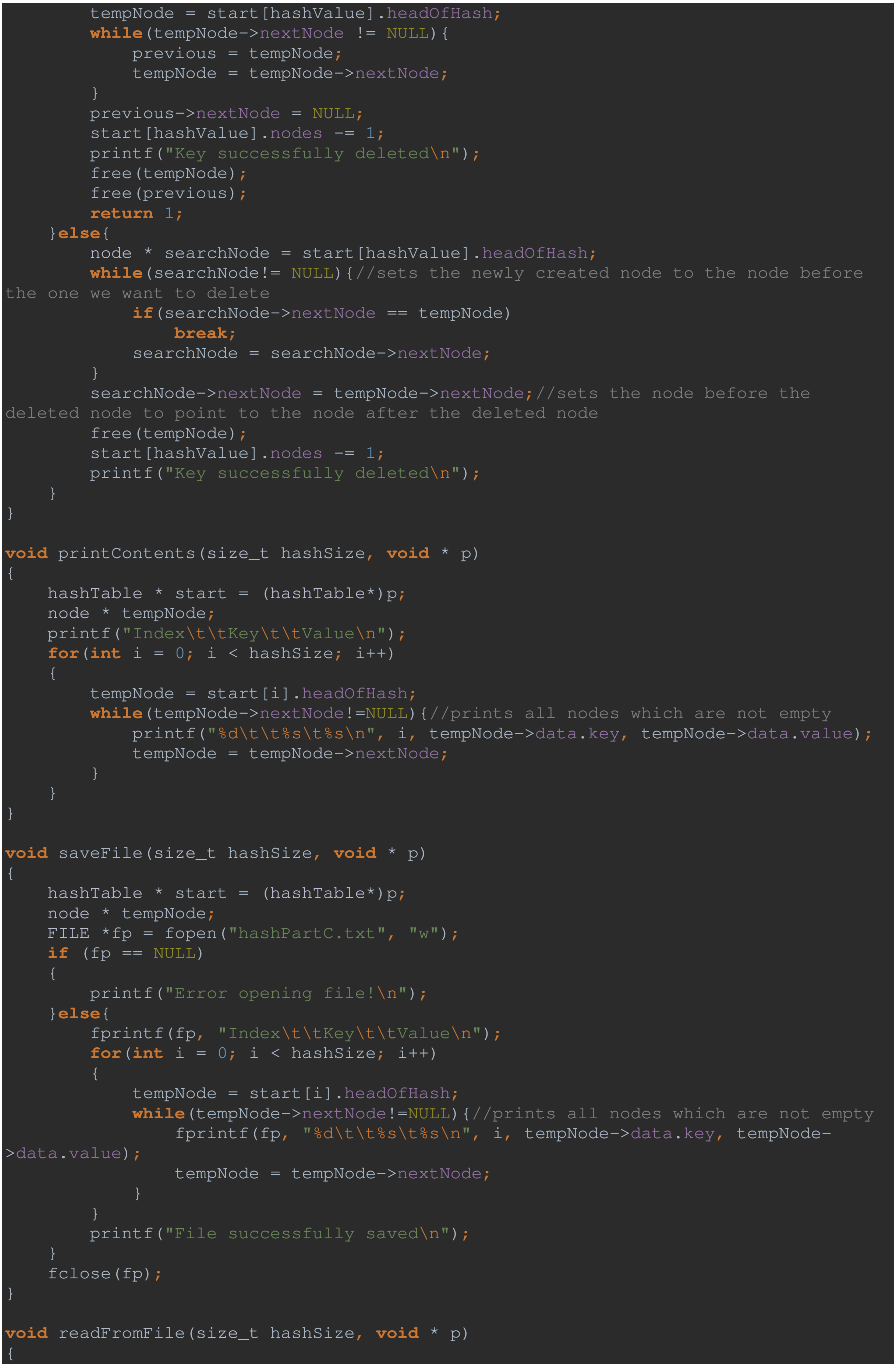
Question 2C

Code

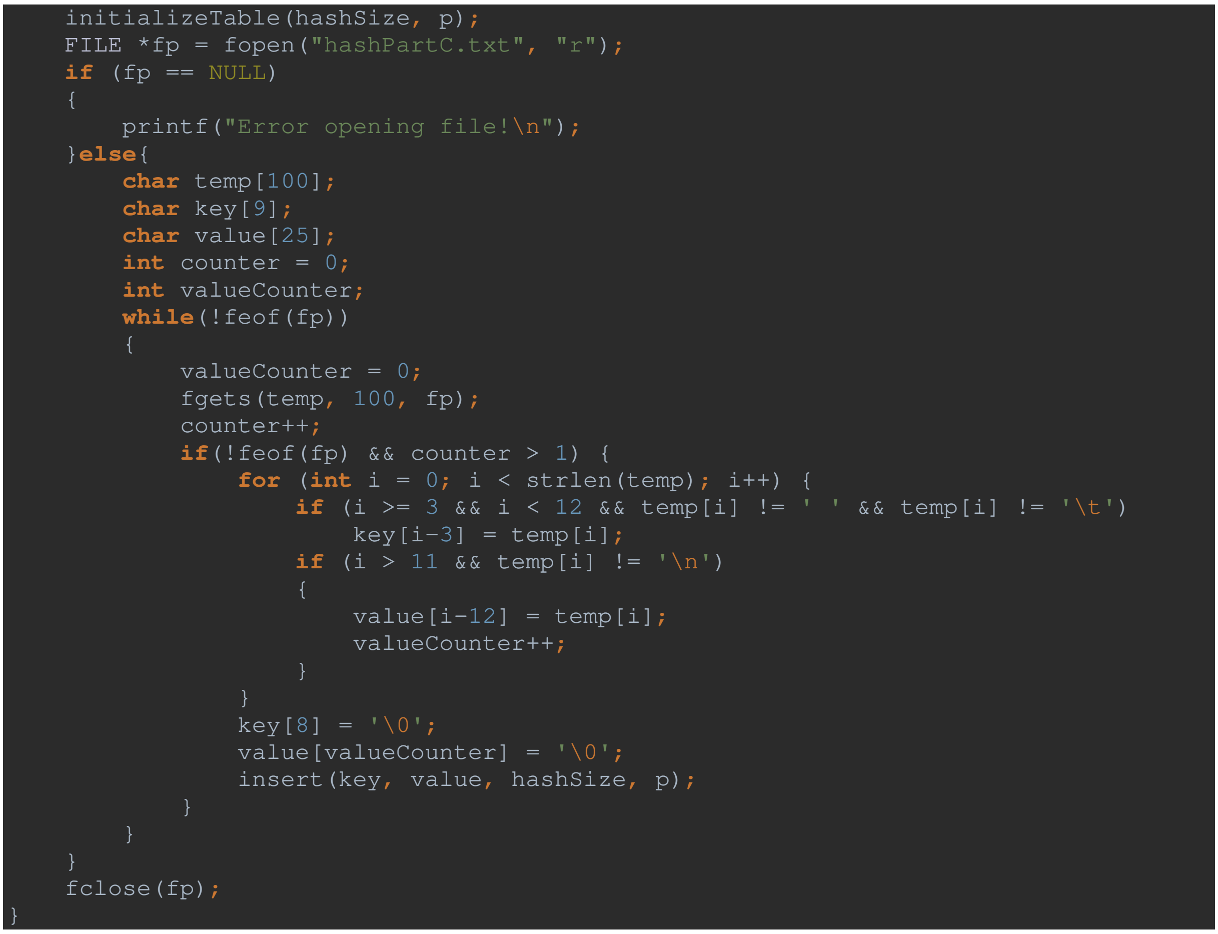
```
#include "hashtable.h"

typedef struct node
{
    pair data;
    struct node * nextNode;
}node;

typedef struct hashTable
{
    size_t nodes;//indicates the amount of nodes currently active for every hash
value
    node * headOfHash;//points to the head node for every hash value
}hashTable;

void * createTable(size_t hashSize)
{
    hashTable * p = (hashTable*)malloc(hashSize * sizeof(hashTable));
    return p;
}

void initializeTable(size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    for(int i = 0; i < hashSize; i++)
    {
        start[i].nodes = 0;
        start[i].headOfHash = (node*)malloc(sizeof(node));//allocating memory for
the first node
        strcpy(start[i].headOfHash->data.key, "\0");//initializing the first node
        strcpy(start[i].headOfHash->data.value, "\0");
        start[i].headOfHash->nextNode = NULL;
    }
}

int hashFunction(char key[9], size_t hashSize)
{
    int keyLength = strlen(key);
    int sum =0;
    for(int c = 0; c<keyLength; c++)
    {
        sum += key[c];
    }
    sum = sum%hashSize;
    return sum;
}

int lookUp(char key[9], size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    int hash = hashFunction(key, hashSize);
    node * tempNode = start[hash].headOfHash;//temporary node used  to search
through the list for that particular hash
    int counter = 0;
    while(tempNode->data.key[0]!='\0'){
        if (strcmp(tempNode->data.key, key) == 0) {
            printf("Key found\n");
            return counter;
        }
        tempNode = tempNode->nextNode;
        counter++;
    }
    printf("Key not found\n");
    return -1;
```



```
}

int insert(char key[9], char value[25], size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    int hashValue = hashFunction(key, hashSize);
    node * tempNode = start[hashValue].headOfHash;
    if(strlen(key)!= 8)
    {
        printf("Key must be 8 characters long\n");
        return -1;
    }
    if(key == "" || value == "")
    {
        printf("Key or value cannot be blank\n");
        return -1;
    }
    if(lookUp(key, hashSize, p) != -1)
    {
        printf("Key is not unique\n");
        return -1;
    }
    while(tempNode->data.key[0]!='\0')
    {
        tempNode = tempNode->nextNode;
    }
    if(tempNode->data.key[0]=='\0')
    {
        strcpy(tempNode->data.key, key);
        strcpy(tempNode->data.value, value);
        start[hashValue].nodes += 1;//increment counter of active nodes for that
hash
        tempNode->nextNode = (node*)malloc(sizeof(node));
        tempNode->nextNode->data.key[0] = '\0';
        tempNode->nextNode->data.value[0] = '\0';
        tempNode->nextNode->nextNode = NULL;
        printf("Entry successfully inserted\n");
        return 1;
    }
}

int delete(char key[9], size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    int hashValue = hashFunction(key, hashSize);
    node * tempNode = start[hashValue].headOfHash;
    if(lookUp(key, hashSize, p) == -1)//key does not exist, exit function
    {
        return -1;
    }else{//if key exists, search for its position
        while(tempNode != NULL){
            if(strcmp(tempNode->data.key,key) == 0)
                break;//stop looping if key is found
            tempNode = tempNode->nextNode;
        }
    }

    if(tempNode == start[hashValue].headOfHash){//first node is target node
        start[hashValue].headOfHash = start[hashValue].headOfHash->nextNode;//set
head pointer to second node
        start[hashValue].nodes -= 1;//decrement counter of active nodes for that
hash
        printf("Key successfully deleted\n");
        tempNode->nextNode = NULL;
        free(tempNode);
        return 1;
    }else if(tempNode->nextNode == NULL){//last node is target node
        node * previous = NULL;
```

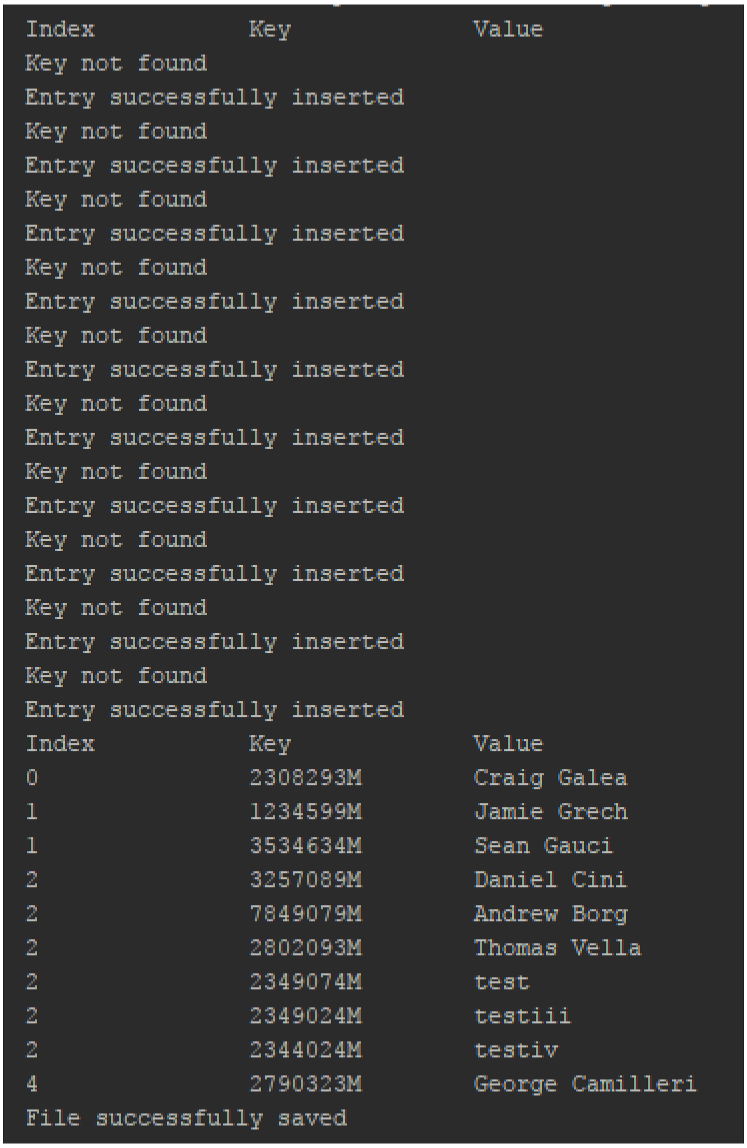


```
        tempNode = start[hashValue].headOfHash;
        while(tempNode->nextNode != NULL){
            previous = tempNode;
            tempNode = tempNode->nextNode;
        }
        previous->nextNode = NULL;
        start[hashValue].nodes -= 1;
        printf("Key successfully deleted\n");
        free(tempNode);
        free(previous);
        return 1;
    }else{
        node * searchNode = start[hashValue].headOfHash;
        while(searchNode!= NULL){//sets the newly created node to the node before
the one we want to delete
            if(searchNode->nextNode == tempNode)
                break;
            searchNode = searchNode->nextNode;
        }
        searchNode->nextNode = tempNode->nextNode;//sets the node before the
deleted node to point to the node after the deleted node
        free(tempNode);
        start[hashValue].nodes -= 1;
        printf("Key successfully deleted\n");
    }
}

void printContents(size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    node * tempNode;
    printf("Index\t\tKey\t\tValue\n");
    for(int i = 0; i < hashSize; i++)
    {
        tempNode = start[i].headOfHash;
        while(tempNode->nextNode!=NULL){//prints all nodes which are not empty
            printf("%d\t\t%s\t%s\n", i, tempNode->data.key, tempNode->data.value);
            tempNode = tempNode->nextNode;
        }
    }
}

void saveFile(size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    node * tempNode;
    FILE *fp = fopen("hashPartC.txt", "w");
    if (fp == NULL)
    {
        printf("Error opening file!\n");
    }else{
        fprintf(fp, "Index\t\tKey\t\tValue\n");
        for(int i = 0; i < hashSize; i++)
        {
            tempNode = start[i].headOfHash;
            while(tempNode->nextNode!=NULL){//prints all nodes which are not empty
                fprintf(fp, "%d\t\t%s\t%s\n", i, tempNode->data.key, tempNode-
>data.value);
                tempNode = tempNode->nextNode;
            }
        }
        printf("File successfully saved\n");
    }
    fclose(fp);
}

void readFromFile(size_t hashSize, void * p)
{
```



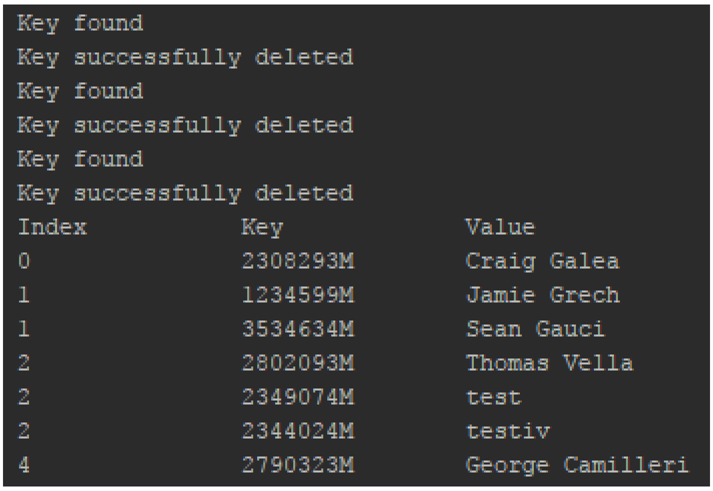
```
    initializeTable(hashSize, p);
    FILE *fp = fopen("hashPartC.txt", "r");
    if (fp == NULL)
    {
        printf("Error opening file!\n");
    }else{
        char temp[100];
        char key[9];
        char value[25];
        int counter = 0;
        int valueCounter;
        while(!feof(fp))
        {
            valueCounter = 0;
            fgets(temp, 100, fp);
            counter++;
            if(!feof(fp) && counter > 1) {
                for (int i = 0; i < strlen(temp); i++) {
                    if (i >= 3 && i < 12 && temp[i] != ' ' && temp[i] != '\t')
                        key[i-3] = temp[i];
                    if (i > 11 && temp[i] != '\n')
                    {
                        value[i-12] = temp[i];
                        valueCounter++;
                    }
                }
                key[8] = '\0';
                value[valueCounter] = '\0';
                insert(key, value, hashSize, p);
            }
        }
    }
    fclose(fp);
}
```

Testing



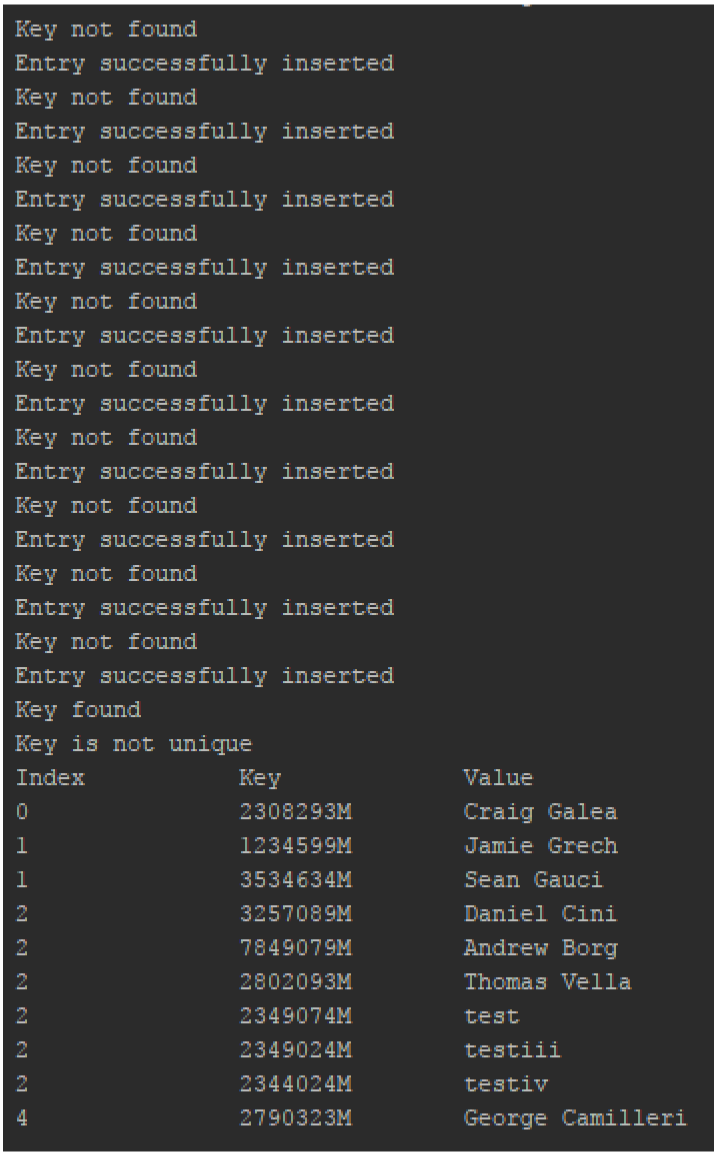
```
Index           Key             Value
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Index           Key             Value
0               2308293M        Craig Galea
1               1234599M        Jamie Grech
1               3534634M        Sean Gauci
2               3257089M        Daniel Cini
2               7849079M        Andrew Borg
2               2802093M        Thomas Vella
2               2349074M        test
2               2349024M        testiii
2               2344024M        testiv
4               2790323M        George Camilleri
File successfully saved
```

Figure 23 - Creating, initializing, saving and adding content to linked list



```
Key found
Key successfully deleted
Key found
Key successfully deleted
Key found
Key successfully deleted
Index           Key             Value
0               2308293M        Craig Galea
1               1234599M        Jamie Grech
1               3534634M        Sean Gauci
2               2802093M        Thomas Vella
2               2349074M        test
2               2344024M        testiv
4               2790323M        George Camilleri
```

Figure 24 - Deleting an item from the head, tail and middle of a row



```
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key not found
Entry successfully inserted
Key found
Key is not unique
Index           Key             Value
0               2308293M        Craig Galea
1               1234599M        Jamie Grech
1               3534634M        Sean Gauci
2               3257089M        Daniel Cini
2               7849079M        Andrew Borg
2               2802093M        Thomas Vella
2               2349074M        test
2               2349024M        testiii
2               2344024M        testiv
4               2790323M        George Camilleri
```

Figure 25 - Loading the linked list to the status at which it was saved i.e. before the delete in Fig. 21

Explanation

In the program given above, the nodes in the linked list are defined recursively within the node struct. This is done by every node in the list having a node pointer nextNode with it, which points to the next node. The hashTable struct contains a node pointer which points to the first node for every hash value, i.e. the head, denoted by headOfHash. It also contains an unsigned integer 'nodes' which is used to keep track of the amount of active nodes for every hash value. It is incremented when a node is added and decremented when a node is deleted from its respective hash value. When a new node is inserted, the key and value of the last node for that hash value are changed from empty strings to the key and value of the node to be added. Memory is then allocated in advance for the next node to be added to that row, the key and value in this empty node are set to empty strings and the empty node's next node pointer is set to null. When a node is to be deleted, the procedure differs depending on if the node is the first node in the row, the last node in the row or somewhere in the middle. In any case, the node to be deleted is stored in a temporary node pointer tempNode. If the node is the first in the row, the hashTable pointer to the head of the row for that hash index is set to the node pointed to by the nextNode pointer of the first node. The nextNode pointer of the temporary node is then set to NULL and the memory allocated to it is freed. If the node is the last in the list, the list is browsed again this time using two pointers, one for the current node and one for the node before it. When the current node pointer reaches the last node, the second pointer, i.e. 'previous', is pointing to the node before the one to be deleted, which also happens to be the new last node in the list. Therefore, previous' nextNode pointer is set to null and the tempNode pointer which contains the deleted node is freed. The previous pointer is also freed since it was just a temporary pointer used to set the new last node's nextNode pointer to null. If the node is in the middle of the row, a new node pointer called searchNode is initialized starting from the head of the row. This node then traverses the row using a while loop until it arrives at the node before the node to be deleted, currently represented by tempNode. When it reaches the destination node, it sets its nextNode pointer to the address pointed to by tempNode's nextNode pointer, thereby removing tempNode from the linked list. The memory allocated to tempNode is then freed. When printing/saving the contents of the list, the program prints/saves all nodes whose nextNode pointer does not point to null, i.e. are not empty. When loading from a file, the same procedure is taken as in part a and b.

Question 2D

Code

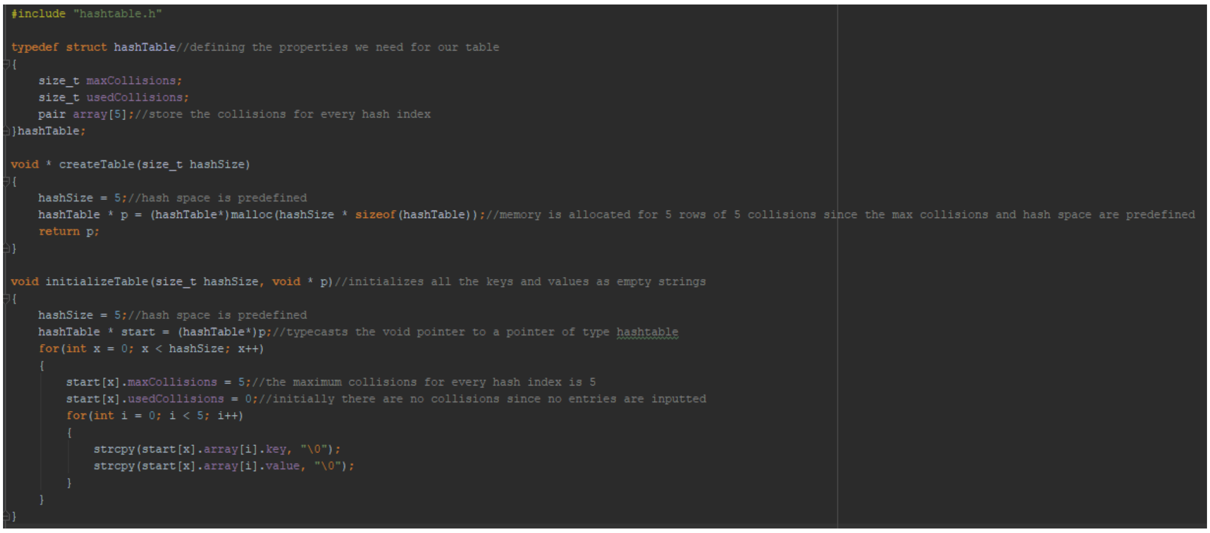
hashtable.h header file

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

#ifndef CPS1011Q2_HASHTABLE_H
#define CPS1011Q2_HASHTABLE_H

#endif //CPS1011Q2_HASHTABLE_H

//pair struct since it is the same in all versions of the table
typedef struct pair
{
    char key[9];
    char value[25];
}pair;

//Function prototypes
void * createTable(size_t hashSize);
void initializeTable(size_t hashSize, void * p);
int hashFunction(char key[9], size_t hashSize);
int lookUp(char key[9], size_t hashSize, void * p);
int insert(char key[9], char value[25], size_t hashSize, void * p);
int delete(char key[9], size_t hashSize, void * p);
void printContents(size_t hashSize, void * p);
void saveFile(size_t hashSize, void * p);
void readFromFile(size_t hashSize, void * p);
//Function prototypes
```

test.c test driver file

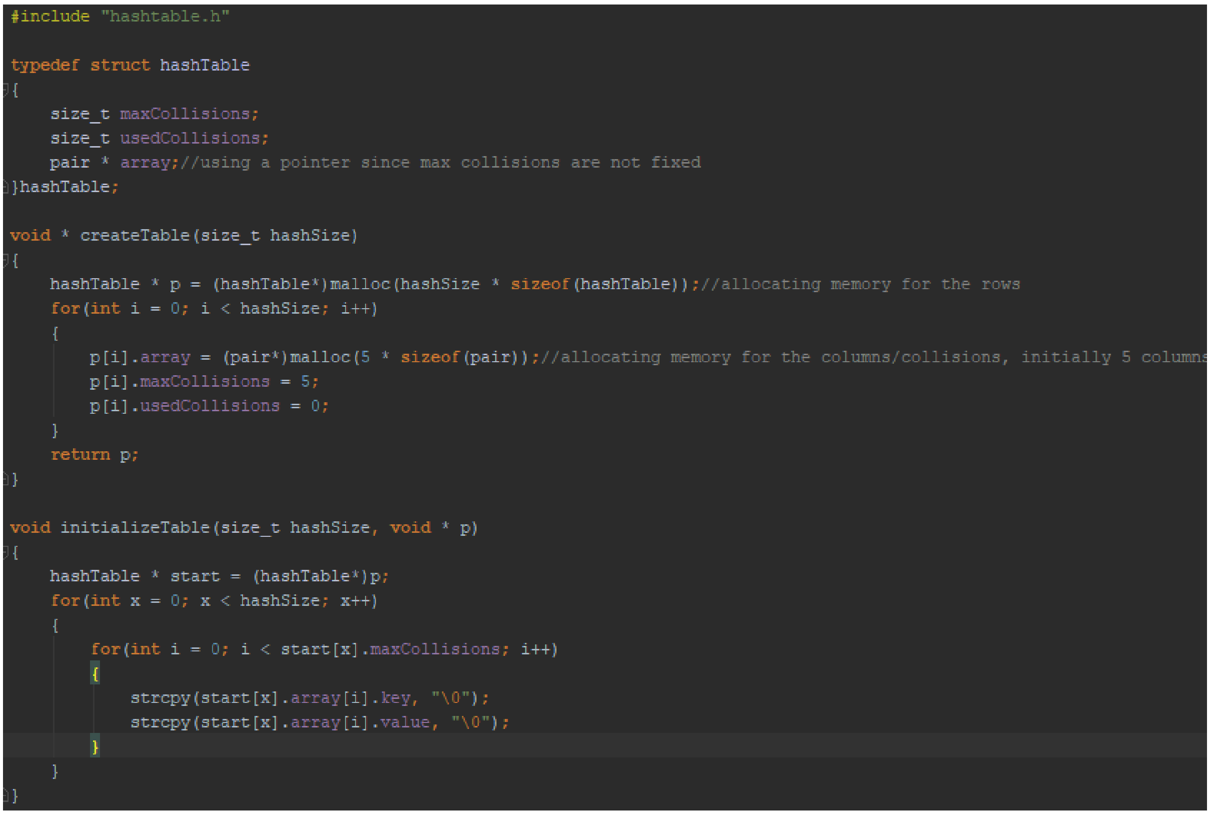
```
#include "hashtable.h"

void main()
{
    void * testTable = createTable(5);
    initializeTable(5, testTable);
    printContents(5, testTable);
    insert("1234599M", "Jamie Grech", 5, testTable);
    insert("2790323M", "George Camilleri", 5, testTable);
    insert("3257089M", "Daniel Cini", 5, testTable);
    insert("7849079M", "Andrew Borg", 5, testTable);
    insert("2802093M", "Thomas Vella", 5, testTable);
    insert("3534634M", "Sean Gauci", 5, testTable);
    insert("2308293M", "Craig Galea", 5, testTable);
    insert("2349074M", "test", 5, testTable);
    insert("2349024M", "testiii", 5, testTable);
    insert("2344024M", "testiv", 5, testTable);
    printContents(5, testTable);
    saveFile(5, testTable);
    delete("7849079M", 5, testTable);
    printContents(5, testTable);
    readFromFile(5, testTable);
    insert("1234599M", "testnonunique", 5, testTable);
    printContents(5, testTable);
}
```

Testing



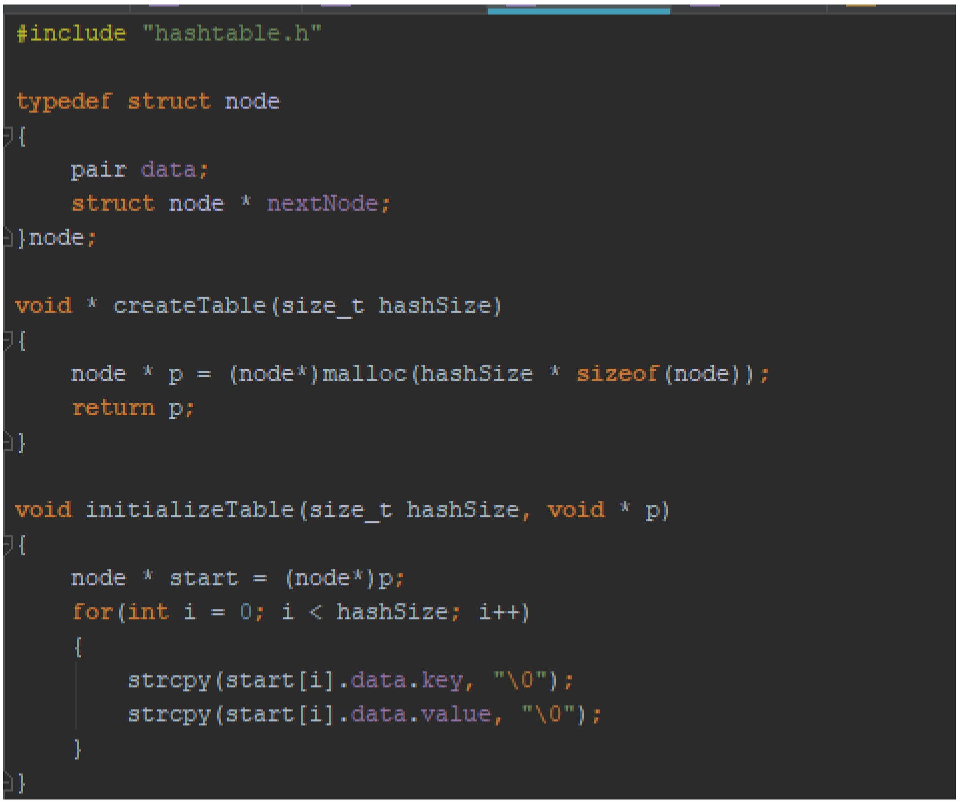
```c
#include "hashtable.h"

typedef struct hashTable//defining the properties we need for our table
{
    size_t maxCollisions;
    size_t usedCollisions;
    pair array[5];//store the collisions for every hash index
}hashTable;

void * createTable(size_t hashSize)
{
    hashSize = 5;//hash space is predefined
    hashTable * p = (hashTable*)malloc(hashSize * sizeof(hashTable));//memory is allocated for 5 rows of 5 collisions since the max collisions and hash space are predefined
    return p;
}

void initializeTable(size_t hashSize, void * p)//initializes all the keys and values as empty strings
{
    hashSize = 5;//hash space is predefined
    hashTable * start = (hashTable*)p;//typecasts the void pointer to a pointer of type hashtable
    for(int x = 0; x < hashSize; x++)
    {
        start[x].maxCollisions = 5;//the maximum collisions for every hash index is 5
        start[x].usedCollisions = 0;//initially there are no collisions since no entries are inputted
        for(int i = 0; i < 5; i++)
        {
            strcpy(start[x].array[i].key, "\0");
            strcpy(start[x].array[i].value, "\0");
        }
    }
}
```

Figure 26 - Successful implementation of hashtable library in question 2a



```c
#include "hashtable.h"

typedef struct hashTable
{
    size_t maxCollisions;
    size_t usedCollisions;
    pair * array;//using a pointer since max collisions are not fixed
}hashTable;

void * createTable(size_t hashSize)
{
    hashTable * p = (hashTable*)malloc(hashSize * sizeof(hashTable));//allocating memory for the rows
    for(int i = 0; i < hashSize; i++)
    {
        p[i].array = (pair*)malloc(5 * sizeof(pair));//allocating memory for the columns/collisions, initially 5 columns
        p[i].maxCollisions = 5;
        p[i].usedCollisions = 0;
    }
    return p;
}

void initializeTable(size_t hashSize, void * p)
{
    hashTable * start = (hashTable*)p;
    for(int x = 0; x < hashSize; x++)
    {
        for(int i = 0; i < start[x].maxCollisions; i++)
        {
            strcpy(start[x].array[i].key, "\0");
            strcpy(start[x].array[i].value, "\0");
        }
    }
}
```

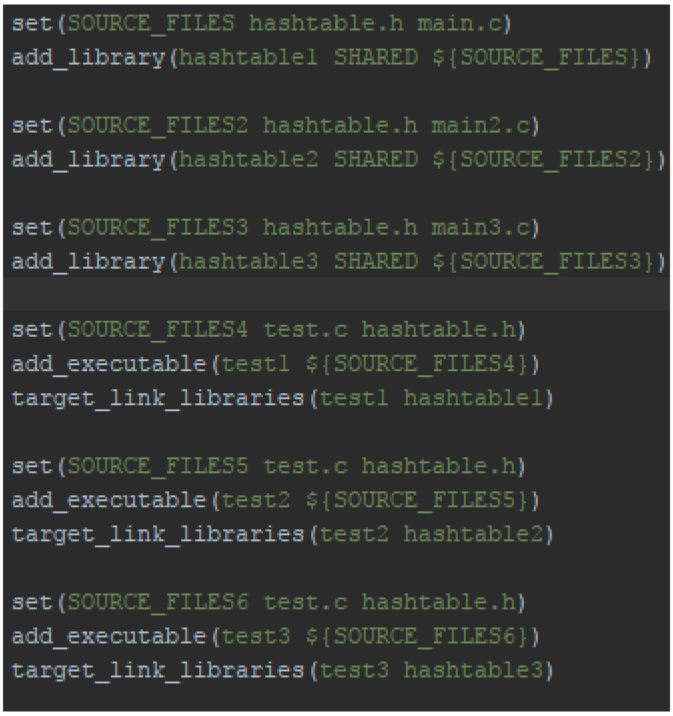
Figure 27 - Successful implementation of hashtable library in question 2b



```
#include "hashtable.h"

typedef struct node
{
    pair data;
    struct node * nextNode;
}node;

void * createTable(size_t hashSize)
{
    node * p = (node*)malloc(hashSize * sizeof(node));
    return p;
}

void initializeTable(size_t hashSize, void * p)
{
    node * start = (node*)p;
    for(int i = 0; i < hashSize; i++)
    {
        strcpy(start[i].data.key, "\0");
        strcpy(start[i].data.value, "\0");
    }
}
```

Figure 28 - Successful implementation of hashtable library in question 2c

Explanation



```
set(SOURCE_FILES hashtable.h main.c)
add_library(hashtable1 SHARED ${SOURCE_FILES})

set(SOURCE_FILES2 hashtable.h main2.c)
add_library(hashtable2 SHARED ${SOURCE_FILES2})

set(SOURCE_FILES3 hashtable.h main3.c)
add_library(hashtable3 SHARED ${SOURCE_FILES3})

set(SOURCE_FILES4 test.c hashtable.h)
add_executable(test1 ${SOURCE_FILES4})
target_link_libraries(test1 hashtable1)

set(SOURCE_FILES5 test.c hashtable.h)
add_executable(test2 ${SOURCE_FILES5})
target_link_libraries(test2 hashtable2)

set(SOURCE_FILES6 test.c hashtable.h)
add_executable(test3 ${SOURCE_FILES6})
target_link_libraries(test3 hashtable3)
```

As can be seen in the above screenshot from the CMakeLists.txt file for question 2, the header file shown in the code section is shared between the source files for all versions of the hash table. The same test driver program test.c is also implemented to test each version of the table. As can also be seen in the screenshots in the 'testing' section above, the hashtable.h header file was successfully implemented in each version of the table created in parts a, b and c. Regarding the header file, it contains a #include for all the external libraries which are used in all the versions of the table, so that each version requires only the hashtable.h header file and no external ones. The pointer used in the test driver is of type void since the hashTable struct is not included in the header file. This is because the struct differs between the versions of the data structure due to their different specifications.