F28HS Report - C

Introduction

This report will cover the code for the C part of the Hardware-Software Interface coursework. I shall first explain the overall functionality of the code and my approach to tackling the task then I will go into more details into each function. The task was to take in an encoded string and use the provided rotNfunction as well as a dictionary in the form of a text file to decode the encoded string and print the key that was used to decode it. If the string given does not have a valid encoded key or if the decoded words did not exist in the dictionary, then a value of -1 is returned.

My approach was the same as the hint given in the specification. I will use a brute force approach to try all possible keys from 1 to 26 then compare the words in the given string to the words in the dictionary.

The strings provided to the main "crack_rot" function are parsed by another function which splits each word up and stores then in an array to be used by the "crack_rot" function.

In addition to the functions, we were provided I also added a few of my own, all functions will be explained in the next section of the report.

Functions

- **Int get_dict_len(char * file){}** – In this function, a the address character array is given as the parameter, this is the string of the file name to be used in the function to find and return the number of words in the dictionary. A function to return the dictionary length is already given but my code was written in CLion and I had issues running it so I created my own. First, a file pointer is declared then using "fopen" I assigned the string given to open the file to be used in the function. I declared and initialized a counter variable to "0" then using "fgets" and a line variable to hold each of the lines, I iterate over the dictionary and increment the value of counter. The while loop will run as long as fgets(line, MAX_LINE,fp) does not return the end of file value of NULL. The counter is then returned at the end of the function which should now hold the value of the number of lines in the dictionary.
- **Int try_key(char **dict, char **text, int key){}** – This function takes in the dictionary array and the text array as parameters. The address of the parametes are assigned to array pointers inside the function. I used a "for" loop and set up another pointer to the text array that will iterate over each value in the array until null is reached. Since a segmentation fault will be triggered upon reaching the null value, an if statement is placed at the end of this function to check if the next value is going to be NULL, if it is then "break" is used to exit the "for" loop before it triggers this. The main body of the first for loop simply uses the "rotN" function with the given key to decode each word in the string. Since pointers are used this change is in-place. The second "for" loop aims to compare each word in the text array with each word in the dictionary array. The variables "checker" and "not found" are declared and initialised out for loops. The second for loop is a nested one and in the inner loop the pointer to the dictionary array is compared to the text array pointer in the outer loop and if the values are the same then "0" is set to the value of "checker" and the inner loop is exited using break. The inner loop also contains an if statement to prevent the pointer from reaching NULL like the one in the outer nested loop. By the time the outloop has finished the value in checker is returned as the output of the "try_key" function.

- **Int crack_rot(char **dict, char **text){}** – This function takes in the dictionary array and the text array as parameters. Character pointers are assigned to the addresses of the parameters and in a “for” loop, a variable “found” is assigned value -1. This for loop will start at 1 and will increment value,“I”, until it reaches 26. Further inside the “for” loop the “found” variable is set to the value returned by a “try_key” function. If the output of the “try_key” function is different from “-1” then the current value at “I” is returned. This is because the “try_key” function passes 1 as the key value each loop as the text array is modified in-place. Instead of reverting the array values to it’s original value then reassigning the key value I chose to use this method instead, this way “try_key” can modify the already modified sentence in each iteration of the for loop until the key is found. If the key is not found and the loop is exhausted then -1 is returned from the entire function.
- **void rotN(char *str, int n) { }** – This function is given and serves to modify the each word given as a string by the specified “n” times. Each character in the word is shifted by the “n” amount
- **char *to_string(char *str) {}** – This is a function I have added which will take in a character array and add a string terminating character to the end. This is done by a “for” loop that points to the address of the string then iterates over the string, appending each character to a temporary string “temp” while the pointer is not pointing to a ‘\n’ , next line, character. The “temp” uses memory allocation so that it’s value can be preserved and returned by the function.
- **int get_len(char * str){}** – This function is used to get the number of words in a sentence passed as a character array. The function uses a “for” loop that points to the address of the string and checks if each value of the pointer is a space, if it is then a “len” variable is incremented by 1. Outside the loop an if statement checks if the current pointer is the same as the value at the start of the “str” array, if not then “len” is further incremented by 1 again and then the value is returned by the function.
- **void show_arr_words (char **arr_words){}** – This function takes in a two dimensional array then prints each value to the console. This is done by a “for” loop with the pointer assigned to the address of the array. The pointer is incremented by 1 each time until a NULL value is reached and then the value at each pointer is printed as well as a “counter” variable which tracks which position in the array the value belongs to. An if statement at the bottom of the “for” loop checks if the next value is NULL then exits the “for” loop if it is.
- **char** read_dict(char* file){}** – This function takes in a string as a character array and returns a 2-dimentional array. The function will take in the name of the file that contains the dictionary of words and will add each value in the file into a new 2-dimentional array. A file pointer is declared and the file is accessed using “fopen()” and in saved in the file pointer,”fp”. A previous function, “get_dict_len()” is used to return the length of the dictionary and assign it to variable “dict_len” then using memory allocation, a two dimensional array in the form of a character pointer to a pointer is declared and assigned empty space the size of a character pointer times the length of the dictionary plus an additional space at the end for a NULL pointer. The main body of this function has a buffer variable the size of the “MAX_LINE_LINE”, defined at the top of the code, and an “arr_count” variable assigned to 0 to track the current position in the diction. An “if” statement will use “Fopen()” to check if the file is not empty and if not, a while loop will append the buffer value to a temporary string variable, “temp”, after using the “to_string()” function to add a ‘\0’ to each line. The “temp” character array is then assigned to the current position in the array and then “arr_count” is incremented by 1. The function then checks if

the “arr_count” is equal to the length of the dictionary then the last element is set to NULL and the file is closed as well as the array return.

- **char **str_to_arr(char * str){}** – This function takes in a sentence in the form of a character array and returns a 2 dimensional array with each word in the sentence split up and appended into each index of the array. This is done by declaring and assigning a pointer to the start of the “str” array. A variable “len” is declared told to the length of words in the sentence using the function “get_len()”. A “for” loop iterates over the sentence till the index reaches the value stored in “len”. In each iteration a temporary character array is declared and initialized using memory allocation, then another “for” loop nested in the first will point to the start of the sentence and iterate over till a NULL value is reached. In each iteration of this inner loop, an “if” statement will check if the current pointer is not a space or a NULL character, if not then the pointer value is assigned to the temporary character array, “temp”, with a “c_counter” tracking the current character position of the temporary array. The variable “c_counter” is incremented each time and another variable “s_counter” will track the current character in the sentence, this is also incremented as the pointer value is appended to the temporary array. Inside the inner loop there is an else statement for when the next pointer is either a space character, ‘ ‘, or a NULL character,”’\0’, when this condition is met, the current pointer is appended to the temporary character array then the next position in the temporary character array is set to ‘\0’ and then space is allocated for the 2 dimensional array which then stores the temporary character array in the index of the outer “for” loop. At the bottom of the outer “for” loop, “c_counter” is set to 0 so that the temporary character array can be reused for the next words and the “s_counter” is incremented so that the inner “for” loop can point to the correct position in the sentence to carry on the parsing. The 2 dimensional array is returned at the end of the nested loops.
- **Int main(){}** – This is the main function and combines all the other functions to print the key and the decoded word. Depending on the command line argument the main function will output different values. If only “verbose” is give, -v, then the hard-coded test is displayed, implementing the other functions to decode the hard-coded text and display them. If ‘-f’ is passed alongside a file name, then the lines in the file are decoded and displayed onto the terminal. If debug, ‘-d’, is passed alongside ‘-f’ then an extra line showing the current line in the file is shown before it is decoded.

Test Results:

Below are the test results.

Test 1: main -v

```
C:\Users\josep\Desktop\C90HSI>main -v -d
Settings for running the program
Help is OFF
Verbose is ON
Debug is ON
Cracked text: gur dhvpx oebja sbk whzc bire gur ynml qbt Key: 13 (expected 13)
Testing
Cipher text:  'gur dhvpx oebja sbk whzc bire gur ynml qbt'
Cracked text: 'the quick brown fox jump over the lazy dog'
Plain text:   'the quick brown fox jump over the lazy dog'
with cracked key: 13
OK decrypted text.
```

Test 2: main -v -d -f filein.txt

```
C:\Users\josep\Desktop\CLion Projects>main -v -d -f filein.txt
Settings for running the program
Help is OFF
Verbose is ON
Debug is ON
Input file = filein.txt
line = gur jbeq vf guvf
Key: 13
Testing
Cipher text: 'gur jbeq vf guvf'
Cracked text: the word is this
with cracked key: 13

line = gur dhvpx oebja sbk whzc bire gur ynml qbt
Key: 13
Testing
Cipher text: 'gur dhvpx oebja sbk whzc bire gur ynml qbt'
Cracked text: the quick brown fox jump over the lazy dog
with cracked key: 13

line = gur yvggyr gbja snqr njnl vagb pbhagel ba bar fvqr pybfr gb gur ragenaprjnl bs n terng cnex
Key: 13
Testing
Cipher text: 'gur yvggyr gbja snqr njnl vagb pbhagel ba bar fvqr pybfr gb gur ragenaprjnl bs n terng cnex'
Cracked text: the little town fade away into country on one side close to the entranceway of a great park
with cracked key: 13

line = aol xbpjr iyvdu mve qbtw vcly aol shgf kvn
Key: 19
Testing
Cipher text: 'aol xbpjr iyvdu mve qbtw vcly aol shgf kvn'
Cracked text: the quick brown fox jump over the lazy dog
with cracked key: 19

line = yllihyiom wixy cm nby miolwy iz uff ypcf
Key: 6
Testing
Cipher text: 'yllihyiom wixy cm nby miolwy iz uff ypcf'
Cracked text: erroneous code is the source of all evil
with cracked key: 6
```

Test 3: main -v 13 “This is the text”

```
C:\Users\josep\Desktop\CLion Projects>main -v 13 "This is the text"
Settings for running the program
Help is OFF
Verbose is ON
Debug is OFF
Encrypting provided input string with provided key
Usage: main [options] <key> <string>
Rotation value:  '13'
Input:  '13'
Output: 'Guvf vf gur grkg'
```

Main -v 13 “Guvf vf gur grkg”

```
C:\Users\josep\Desktop\CLion Projects>main -v 13 "Guvf vf gur grkg"
Settings for running the program
Help is OFF
Verbose is ON
Debug is OFF
Encrypting provided input string with provided key
Usage: main [options] <key> <string>
Rotation value:  '13'
Input:  '13'
Output: 'This is the text'
```

Summary

To summarise, for the C part of the course work I was able to learn how to effectively use pointers to manipulate strings in c and modify variable in a memory level and how strings are represented as character arrays in c. I can understand how each of the character of a string are stored in memory and manipulate them in different ways. I also learnt how to use memory allocation to dynamic allocate values to variables. I also understand how memory is stored in c in terms of stack and heap as well as the lifetime of functions and local variables and how they interact with the main function and the rest of the code.