**Team Project 2: Selection Sort and**

**Encryption Decryption Algorithms**

**Group 6 Team Report**

Fall 2021 CS351

Southeast Missouri State University

**11/29/2021**

|  |  |
| --- | --- |
| Name | Email |
| Zach Healy | zjhealy1s@semo.edu |
| Jalik Smith | Jmsmith13s@semo.edu |
| Momoh Brema | mbrema1s@semo.edu |

**Team Member Contributions**

Each team member’s contribution can be found in Table 1.

Table 1 - Team member contributions

| Team Member  Email | Contributions |
| --- | --- |
| Zach Healy  Zjhealy1s@semo.edu | * Selection Sort Algorithm * Encryption Decryption Algorithm * Documentation * Managed Team * Attended and fully participated in group zoom meetings |
| Jalik Smith  Jmsmith13s@semo.edu | * Helped write main code for selection sort * Helped write main code for encryption decryption * Made diagrams and tables * Attended and fully participated in group zoom meetings |
| Momoh Brema  Mbrema1s@semo.edu | • Sent initial email to get team together to begin work • Coordinated all zoom meetings • Helped with documentation • Made diagrams and tables • Drafted Introduction of report • Attended and fully participated in group zoom meetings |

Video Link to Team Presentation

The link to our team presentation is found at <https://youtu.be/SHqou1_Sb-4>

Table of Contents

[Abstract 3](#_Toc89179301)

[Introduction 3](#_Toc89179302)

[Main body 5](#_Toc89179303)

[Discussion 12](#_Toc89179318)

[Conclusion 13](#_Toc89179319)

[References 13](#_Toc89179320)

[Appendix 13](#_Toc89179321)

Table of Figure

[Figure 1 – scanning file and assigning to array 5](#_Toc89179304)

[Figure 2 – Selection Sort alogrithm 6](#_Toc89179305)

[Figure 3 – assigning array to file 6](#_Toc89179306)

[Figure 4 – selection sort running 7](#_Toc89179307)

[Figure 5 – running result of selection sort 7](#_Toc89179308)

[Figure 6 – scanning file and assigning to array (encryption) 8](#_Toc89179309)

[Figure 7 – Encryption algorithm 9](#_Toc89179310)

[Figure 8 – write encryption to file 9](#_Toc89179311)

[Figure 9 – read encryption file and assign to array 10](#_Toc89179312)

[Figure 10 – decryption algorithm 10](#_Toc89179313)

[Figure 11 – running decryption and write to file 11](#_Toc89179314)

[Figure 12 – encryption and decryption running 11](#_Toc89179315)

[Figure 13 – encryption file 12](#_Toc89179316)

[Figure 14 – decryption file 12](#_Toc89179317)

[Figure 15 – selection sort diagram 17](#_Toc89179322)

[Figure 16 – encryption and decryption diagram 25](#_Toc89179323)

[Figure 17 – File input output table 28](#_Toc89179324)

Abstract

This project’s goal is to make two C programs, one that takes a file of numbers and sorts them into their numeric order and another program that will take a file that contains a message, will encrypt it into a new file and then will read the new file and decrypt it back to another new file to show the original message. Before we even started the project, we did our research since there were a few elements of this project that were new concepts to us, such as file reading and writing, as well as pointers were still overall a new concept for us. To do the projects, we use a combination of file input/output, arrays, pointers, and functions all while working inside of Ubuntu. For the encryption and decryption, we used a shift cipher, and for the selection sort we ended up using nested for loops in order to help compare two different elements of the array. We used pointers to help shift the elements around when need be.

Introduction

This is project 2 for group 6, consisting of Zach Healy, Jalik Smith, and Momoh Brema. This project consists of having 2 programs, one that is a selection algorithm which takes a file of numbers, adds them into an array, and then sorts the in numerical order and writes them to a file output. The other program that was written for this project consists of encryption and decryption algorithms. This program will take the input of a file that has a message, then encrypt it to a file output. After that it will take that same file and read it to then be decrypted and have the result message be outputted to a new file. The main goals that were to be focused on for these programs was help with learning when and how to use pointers and arrays in C programming, as well as how to write functions that work within your code. The last thing that we needed to learn for this assignment was to read and write files while using C programming in the Linux Ubuntu environment. We had to learn a lot to do this project to the professional standard that we strive for. For one, we had to learn how to most efficiently use the linux environment. Alongside that, the idea of file input and output in linux was a new concept for us in C. While we have a basis for things like arrays in other languages in Java and Python, every programming language seems to have different ways to read and store files.

Main body

For the selection sort file written, we figured out a lot of good basic information that helped us throughout the rest of the project. The first thing we knew we needed to figure out was how to import the file and set it to the array we would use for the rest of the algorithm. For that we used the line the fopen function that is a basic part of our environment and defined it to a variable called “\*file”. We run a check to make sure that the file is there and that there are things in the file. Then we use malloc to allocate the size. We then use a while loop to add each element to the pointer. Then we used a for loop, to check that if we made it through the file to assign each number to an element from the array by using the variable ‘i’ to increment it.

Figure 1 – scanning file and assigning to array

Graphical user interface, text, application

Description automatically generated

Next we ran the function which would go through and sort the array. This went through the whole array two times, comparing the first number and the one that came after it. It would compare them and then move the array elements if needed, then move onto the next set.

Figure 2 – Selection Sort alogrithm

Text

Description automatically generated

The last step is to use the fprintf function in order to print out each element of the array into the file that we opened up right before. This will be using the sorted array since we are fprintf-ing it after the sorting as taken place.

Figure 3 – assigning array to file

Text

Description automatically generated

This shows us making the file, as well as showing that when ran, there are no errors.

Figure 4 – selection sort running

Text

Description automatically generated

Here is the output of the file that is created while running the program.

Figure 5 – running result of selection sort

Graphical user interface, application

Description automatically generated

For the encryption and decryption code we needed to write, we used the basis that we learned from the selection sort and knew we needed to first open up the file that was going to be used, and this time scan each element and added it to a char array. We create a malloc to a pointer, then open a file and make sure they aren’t NULL. We used the same for and while loops to check and see if we made it to the end of the file, and if not continue looping while assigning each element to the incrementing position using ‘i’ to the malloc then to the array after.

Figure 6 – scanning file and assigning to array (encryption)

Text

Description automatically generated

The next step we needed to take was to run the newly made array through the encryption function. For this process, we used a simple encryption process known as a shift cipher. A shift cipher is a cipher that take a character, and shifts it x amount of places in the character list. So if you have a shift of 3, the character a becomes a d since that would be 3 positions.

Figure 7 – Encryption algorithm

Text

Description automatically generated

We then assigned the array to a file, which because this is a character array makes it much easier. We first open up a new file that we named “encrypted\_message.txt” and then we used the fputs command to assign the variable to that file.

Figure 8 – write encryption to file

A picture containing graphical user interface

Description automatically generated

After we assign the array to that file, we can then read that file again and assign it back to the array it was just taken from.

Figure 9 – read encryption file and assign to array

Text

Description automatically generated

We will then run it through our decryption algorithm, just shifting the encrypted message back the same amount of characters that we shifted it in the first place.

Figure 10 – decryption algorithm

Text

Description automatically generated

We then end up assigning it back to the final file that will be called “result\_message.txt”. This once again can just use the fputs command instead of running a for loop with fprintf.

Figure 11 – running decryption and write to file

A picture containing diagram

Description automatically generated

Figure 12 – encryption and decryption running

This shows us making the file, as well as showing that when ran, there are no errors

Text

Description automatically generated

Figure 13 – encryption file

This shows the file that is the running result of the encryption function

Text

Description automatically generated

Figure 14 – decryption file

This shows the file that is the running result of the decryption function

Graphical user interface, text, application

Description automatically generated

Discussion

Our results were very pleasing at the end! We ended up getting the programs to work perfectly as we had planned them to, however that was not without its hiccups. The main issues that we faced were with getting the program to write its results to a file for the output. We’ve done this in other languages, but C was just different enough that it gave us quite a bit of trouble. Once we read the textbook and figured out more about the syntax regarding the writing format it made a lot more sense. Along with that, a smaller hiccup we encountered was that our linux environment we were working in crashed and was constantly failing boot in the middle of the project. Luckily we had our code stored somewhere else so we were able to re-install Ubuntu and rewrite the programs in a timely manner.

Conclusion

Overall, this project was very interesting for us as a group. We learned a lot about C, specifically some really important concepts that we are going to be using in the field in the future. Between learning about file input and output and diving deeper into a core concept like arrays this entire project gave us great real world experience. Alongside the basic concepts, the assignments themselves seemed very heavily based on a real world assignment. The idea of making an encryption or sorting algorithms is something that we can see being used daily in the work force with ideas like sorting a list of products from least expensive to most expensive.

References

King, K. N. (2008). C Programming: A modern approach (2nd ed.). W. W. Norton.

Appendix

Selection\_sort.c

#include<stdio.h>

#include<stdlib.h>

//Define function

double\* selection\_sort(double a[]);

int main(){

//define variables

int i = 0;

double num;

double \*p;

double arr[36];

double \*c;

//open file

FILE \*file = fopen("num.txt", "r");

//checks if file if found and returns error if not

if(file == NULL){

printf("File not found\n");

exit(0);

}

c = (double\*)malloc(100\*sizeof(double));

//assign file info to array

while(fscanf(file, "%le, ", &num) > 0){

c[i] = num;

i++;

}

for(int i = 0; i < 36; i++){

arr[i] = c[i];

}

//run function to sort array

p = selection\_sort(arr);

//load file to store array to, then store array to file

FILE \*aNum = fopen("ascended\_num.txt", "wb");

for(int i = 0; i < 36; i++){

fprintf(aNum, "%f\n", p[i]);

}

fclose(aNum);

}

//function for selection sort

double\* selection\_sort(double arr[]){

//define variables

int temp = 0;

double place = 0;

//selection sort alogrithm

for(int i = 0; i < 36; i++){

temp = i;

for(int j = i+1; j < 36; j++){

if(arr[j] < arr[temp]){

temp = j;

}

}

//after comparison, move array elements

place = arr[temp];

arr[temp] = arr[i];

arr[i] = place;

}

//return sorted array

return arr;

}

Figure 15 – selection sort diagram

Diagram

Description automatically generated

Diagram

Description automatically generated

Chart, diagram

Description automatically generated

encryptDecrypt.c

#include<stdio.h>

#include<stdlib.h>

//define functions that will be used

char\* encrypt(char arr[]);

char\* decrypt(char arr[]);

//define size that will be used for array

//and array scanning

#define SIZE 74

int main(){

//define variables used

char arr[SIZE];

char \*p;

char temp;

int i;

p = (char\*)malloc(100\*sizeof(char));

if(p == NULL){

exit(0);

}

//Read message.txt

FILE \*file = fopen("message.txt", "r");

//checks if file if found and returns error if not

if(file == NULL){

printf("File not found\n");

exit(0);

}

//assign to malloc

while(fscanf(file, "%c", &temp) > 0){

p[i] = temp;

i++;

}

for(int i = 0; i < SIZE; i++){

arr[i] = p[i];

}

//encryption and print to file.

encrypt(arr);

FILE \*encrypt = fopen("encrypted\_message.txt", "w");

fputs(arr, encrypt);

//read encrypted file

FILE \*encrypted = fopen("encrypted\_message.txt", "r");

//checks if file if found and returns error if not

if(encrypted == NULL){

printf("File not found\n");

exit(0);

}

//assign to array

while(fscanf(encrypted, "%c", &temp) > 0){

arr[i] = temp;

i++;

}

//decrypt and print to file

decrypt(arr);

FILE \*result = fopen("result\_message.txt", "w");

fputs(arr, result);

}

//encryption function (shift cipher)

char\* encrypt(char arr[]){

//runs through array

for(int i = 0; i < SIZE; i++){

//checks to see if char is lowercase then shifts

if(arr[i] >= 'a' && arr[i] <= 'z'){

arr[i] += 3;

//if char goes out of bounds, then it goes back to start of alphabet

if(arr[i] > 'z'){

arr[i] = arr[i] -  'z' + 'a' - 1;

}

//same as lower case, but for upper case letters

}else if(arr[i] >= 'A' && arr[i] <= 'Z'){

arr[i] += 3;

if(arr[i] > 'Z'){

arr[i] = arr[i] - 'Z' + 'A' - 1;

}

}

}

}

//decryption function (shift cipher)

char\* decrypt(char arr[]){

//runs through array

for(int i = 0; i < SIZE; i++){

//checks to see if char is lowercase then shifts

if(arr[i] >= 'a' && arr[i] <= 'z'){

arr[i] -= 3;

//if char goes out of bounds, then it goes back to start of alphabet

if(arr[i] < 'a'){

arr[i] = arr[i] + 'z' - 'a' + 1;

}

//same as lower case, but for upper case letters

}else if(arr[i] >= 'A' && arr[i] <= 'Z'){

arr[i] -= 3;

if(arr[i] < 'A'){

arr[i] = arr[i] + 'Z' - 'A' + 1;

}

}

}

}

Figure 16 – encryption and decryption diagram

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Figure 17 – File input output table

Table

Description automatically generated