**Project Title: Contact Management System**

**1.0 Introduction**

Our project comprises a contact management system that enables users to quickly manipulate their contact list. This project aims to organise, manage, and streamline communication with contacts, which is crucial to both individuals and enterprises. Some of the key features included in our system are the ability to add, edit, delete, search, and display the user's contacts.

**2.0 Installation**

To install and set up our contact management system, open a power shell/command prompt and copy the location of your file into the terminal. Please ensure that the latest version of MinGW-w64 via MSYS2, which provides up-to-date native builds of GCC, is downloaded on your system. Then, enter gcc -o main.exe main.c into the terminal.



The program should then be installed automatically.

**For Mac:**

Check if clang is installed on your system by opening your macOS terminal window and entering the following command:

clang --version

If clang isn’t already installed, enter the following command line to include clang:

xcode-select –install

The program should then be installed automatically.

**3.0 Usage**

To run the program, enter the following command into the terminal:

**Windows VScode:** ./program\_name.exe

**macOS****:** ./program\_name

**Windows Command Prompt:** program\_name

**4.0 Features**

Some of the main features of our program include:

* Feature 1: Add contacts
* Feature 2: Edit contacts
* Feature 3: Delete contacts
* Feature 4: Search contacts
* Feature 5: Display contacts
* Feature 6: Delete contacts.txt file
* Feature 7: Input Validation

**Add Contacts**

The **add();** function adds contacts to the contact management system. It creates a **<contacts.txt>** file and a temporary file, **<temp.txt>.** The system will prompt the users to enter the *contacts.first\_name*, *contacts.last\_name*, *contacts.phone\_num*and*contacts.email*.Later, the system will ask if the user wants to put it into Favourite using the **yesNo()** function. If yes, *contacts.fav* contains ‘y’ else it contains ‘n’. This will be stored in the temporary file as the following format, [First Name\n Last Name\n Phone Number\n Email\n Favourite (as either y / n) and Archive (always n)]. The inputs are all converted to lowercase. The system will then proceed to confirm if the user is sure the user wants to add this contact, only then will it write from the **<temp.txt>** in to the **<contacts.txt>** file. The **<temp.txt>** file is removed. The system will ask the user if they want to continue, if yes, the **add();** function repeats, else it returns to the main menu.

**Edit Contacts**

The **edit();** function modify previously added contacts in the contact management system. It opens the **<contacts.txt>** file and creates a temporary file, **<temp.txt>**, if there is an issue when opening both files, an error message is shown. The system will prompt the users to enter the *first\_name*, *last\_name*of the contact that the user would like to modify. By using the *first\_name*, *last\_name*the user inputted, the system will scan 6 lines of text each line with the following format, [First Name\n Last Name\n Phone Number\n Email\n Favourite and Archive]. The system will then only use the *contacts.first\_name*, *contacts.last\_name*from the fileto compare with the user’s input. If both are true, the system will prompt the users to enter the *contacts.first\_name*, *contacts.last\_name*, *contacts.phone\_num*and*contacts.email*. and ask for *contacts.fav* using the **yesNo()** function again. The input will be converted to lowercase and stored as the same format in a **<temp.txt>** file and when the user confirms to be sure of the changes the changes will be store in the then to the **<contacts.txt>** file. Another error message will be shown when the transfer was unsuccessful. The system will ask the user if they want to continue, if yes, the **edit();** function repeats, else it returns to the main menu.

**Delete Contacts**

The **del()**; function deletes contacts from the contact management system. It involves searching for a contact in the original file <contacts.txt> and deleting it. The function accesses the main contact file (fcontact) in read mode before opening the temporary file <temp.txt> in write mode to store new contact information. The code prompts the user for a first and last name and checks that the input is not empty with the **r\_empty\_str()** function. To maintain a consistent comparison, the names are changed to lowercase with the **to\_lowercase()** function. The program then reads each contact from the <contacts.txt> file and compares the *first\_name* and *last\_name* attributes to the user's inputs. If a match is detected, the **yes\_no()** function will prompt for confirmation before deleting the record. Otherwise, it will be written to <temp.txt> (including non-matching ones). Both files are then closed, the original file updated, and the temporary file removed. Some of the issues addressed are file opening errors, no matching contact found errors, and temporary file replacement.

**Search Contacts**

The **search()** function allows users to search for contacts by first name, last name, phone number, or email. Additionally, users have the option to enter partial keywords or a single alphabet, which enables suggestions to be display for matching results. It provides a search that is case-insensitive and the ability to go back to the main menu or continue.

The **get\_suggestions(const char\* partial\_input, int search\_choice, Suggestion suggestions[], int\* suggestion\_count)** function retrieves corresponding contact suggestions. It performs case-insensitive comparisons and formats the results for display.

**Display Contacts**

Depending on your choice, the display function will either run the **display();** function if you enter 1, **fav();** if you choose 2 or return to main menu if you choose 3. If it is none of the above, the system will prompt the user to re-enter their choice.

The **display();** function will scan 6 lines of text each line with the following format, [First Name\n Last Name\n Phone Number\n Email\n Favourite and Archive]. If *contacts.archive* is ‘y’, it will not be displayed, but if *contacts.*archive is ‘n’ only the *contacts.first\_name*, *contacts.last\_name*, *contacts.phone\_num*and*contacts.email* will be displayed.

**Delete contacts.txt File**

The **delete\_file()** function deletes the entire original file <contacts.txt> from the file system. This function uses the **yesno()** function to get user confirmation before deleting the <contact.txt> file. If the file is successfully deleted, it returns zero; otherwise, it returns a non-zero value and terminates the file deletion operation. When the file is successfully erased, a success message appears and returns to the main menu. Otherwise, an error notice will appear.

**Input Validation**

The **r\_empty\_str()** function ensures the presences of user input is neither empty nor only whitespace by prompting continuously until valid input is entered. The **is\_duplicate\_contact()** function checks if a contact's first and last names already exist in the <fcontact> file, it will return true if a duplicate is found. The **is\_valid\_email()** function verifies if the email entered contains one “@”, one “.”, and valid characters, ensuring “@” appears before “.”. The **is\_valid\_phone()** function checks a phone number for valid characters (digits, +, -, spaces), and the input is at least one digit long, and a minimum length of 7.

The bonus features in our program include:

* Bonus Feature 1: Sorting Contacts
* Bonus Feature 2: Encryption & Decryption
* Bonus Feature 3: Authentication
* Bonus Feature 4: Favourite Contacts
* Bonus Feature 5: Archive Contact
* Bonus Feature 6: Backup and Restore

**Sorting Contacts**

The **sort\_contacts(Contact contacts[], int count)** function sorts an array of contacts alphabetically by first name. If two contacts share the same first name, it arranges them by last name. The function implements a bubble sort algorithm, swapping contacts when necessary to keep them in order.

**Encryption**

The **encrypt()** function is used to encrypt the original file <contacts.txt> to ensure that the data is protected. The function initially declares two file pointers, *fp* and *temp*, which points to the original file for reading and the temporary file for writing the encrypted content. The original file is opened in read mode; if it cannot be opened, an error notice is displayed. The temporary file is then opened in write mode and used to temporarily store the encrypted data. If the file cannot be opened, an error notice will appear. The code then reads characters from the <contacts.txt> file and increases their ASCII values by three. The updated characters are written into the temporary file <temp.txt>, and this process continues until the file is complete. After processing, both the original and temporary files are closed, and the encrypted temporary file replaces the original.

**Decryption**

The **decrypt()** function is used to decrypt the original file <contats.txt>. The function initially declares two file pointers, *fp* and *temp*, which point to the original file for reading and the temporary file for writing the decrypted content. The original file is opened in read mode; if it cannot be opened, an error notice is displayed. The temporary file is then opened in write mode and used to temporarily store the encrypted data. If the file cannot be opened, an error notice will appear. The code then reads characters from the <contacts.txt> file and decreases their ASCII values by three. The updated characters are written into the temporary file <temp.txt>, and this process continues until the file is complete. After processing, both the original and temporary files are closed, and the decrypted temporary file replaces the original.

**Authentication**

The **authentication()** function is used to authenticate the user using the program. For security reasons, a passcode (dRSiMoN) is set. Before decrypting the <contacts.txt> file for reading, the application will prompt the user to provide their password. If the password is correct, a success message will be displayed, and the **decrypt()** function will be called. Otherwise, an error message will be displayed, followed by an exit to the main menu.

**Favourite Contacts**

The **fav();** function will act similarly to **display();**. It scans 6 lines of text each line with the following format, [First Name\n Last Name\n Phone Number\n Email\n Favourite and Archive]. It will only display the *contacts.first\_name*, *contacts.last\_name*, *contacts.phone\_num*and*contacts.email* when the *contacts.fav* of the contact is ‘y’ and *contacts.archive* of the contact is ‘n’. After that, the system asks the user if they want to add contacts as favourite. If add, *contacts.fav* is changed to ‘y’, or when the contact’s *contacts.fav* is ‘y’, it will tell the user the contact is already in Favourite. If remove, *contacts.fav* is changed to ‘n’, or when the contact’s *contacts.fav* is already in ‘n’ it will tell the user the contact is not in Favourite. If the user choose exit, the program returns to main menu.

**Archive Contact**

The **archive();** function will also act similarly to **display();**. It will display an archive menu with the choices, archive, unarchive, view and exit. When the user selects archive, the function will accept the user inputs *first\_name*  and *last\_name* and ask for confirmation. Once the user confirms their selection, the contact will be archived and won’t be displayed by the **display();** function. Users can also unarchive contacts if needed as well as view the contacts in their archive list. If the user chooses to exit, the program returns to the main menu.

**Backup**

The **back\_up()** function creates a **<backUp.txt>** file, copies all contents from **<contacts.txt>** and writes it into <backUp.txt>. Variable *c* is declared as an integer as **fgetc()** returns an integer instead of character. The while loop in this function compares value of each character return by fgetc(backUp) to **EOF** (end of file: a special constant that indicates no more characters can be read) and write each character to <contacts.txt> with **fputc(*c*, *fp)***, where *fp* is the file pointer of <contacts.txt>.

**Restore**

The **restore()** function is similar to back\_up(), however it requires an **existing backup file** which means the user must make a backup first. Thus, this function restores contents from <backUp.txt>. It uses **fgetc()** to get the character and compare the value of the character returned by fgetc() with **EOF** to see if it hits the end of file. Then, it uses **fputc(*c*, *fp*)** to write each character into <contacts.txt>. If the user has backed up the contacts, then he/she modified the contacts without backing up again and chooses to restore, the modified content wouldn’t be saved.