

PES UNIVERSITY

Electronic City Campus, 1 KM before Electronic City, Hosur Road,
Bangalore-100



PROJECT REPORT

on

“Personal Information Manager”

Submitted in partial fulfillment of the requirements for the IV Semester
Secure programming with C (UE19CS257C)

Bachelor of Engineering
IN
COMPUTER SCIENCE AND ENGINEERING

For the Academic year
2020-2021

BY

Student Name-Nikhil Adyapak	SRN PES2UG19CS257
Student Name-Anirudh Joshi	SRN PES2UG19CS042
Student Name-Jaywanth J	SRN PES2UG19CS165

Under the Guidance of
Vishwachetan D
Assistant Professor

Department of Computer Science and Engineering
PES UNIVERSITY EC CAMPUS
Hosur Road, Bengaluru -560100

PES UNIVERSITY EC CAMPUS

Hosur Road, Bangalore -560100

Department of Computer Science and Engineering



CERTIFICATE

Certified that the project work entitled **“Personal Information Manager”** is a Bonafede work carried out by **Nikhil Adyapak PES2UG19CS257, Jaywanth J PES2UG19CS165, Anirudh Joshi PES2UG19CS042**: of **PES University EC CAMPUS** in partial fulfillment for the special topic course **Secure Programming with C** of IV Semester in **Computer Science and Engineering** of the **Pes University, Bangalore** during the year 2020-2021.

Signatures:

Project Guide:
Vishwachetan D
Assistant Professor, Dept. of CSE,
PES UNIVERSITY EC CAMPUS,
Bengaluru

Dr. Sandesh B J
Head, Dept of CSE
PES UNIVERSITY EC CAMPUS,
Bengaluru

Declaration

I hereby declare that the project entitled **“Personal Information Manager”** submitted for the special topic course **Secure Programming with C** of IV Semester in **Computer Science and Engineering** of the **Pes University, Bangalore**, Bangalore is my original work.

Signature of the Student: Nikhil Adyapak

Place: Bengaluru

Date:1/5/2021

Signature of the Student: Anirudh Joshi

Place: Bengaluru

Date:1/5/2021

Signature of the Student: Jaywanth J

Place: Bengaluru

Date:1/5/2021

Table of Content

1.	Abstract : Problem Definition	
2.	Introduction	
3.	Requirement Specification	Software Requirement Software features and influence of following 1: C Programming Language 2: Compiler 3: Memory
4.	Security Requirement specification	<ul style="list-style-type: none">• Safety features.• List the safety recommendation required for your software• Justify the recommendation.
5.	Design	<ul style="list-style-type: none">• Data flow diagram• Modularity• Readability• Abstraction
6.	Implementation	Code
7.	Testing and Analysis	vulnerable test cases (Input) non Vulnerable test cases (Input) Static analysis report (splint)
8.	Maintenance	How easy for <ul style="list-style-type: none">• Modification (future enhancement)• Creating Multiple Version• Scalability
9.	Conclusion	

Abstract of the Project

Problem Statement-

The personal details of users are encrypted and is made available to user only when the user enters the correct password.

Need of safe programming:

With the good the technology has done to our world it has also brought in the bad, hackers can easily plant a bug in your software and your software will be in control of bad hands. For a simple example, when your software asks a new user to create an account hackers can give command prompt instructions instead of the details required to make an account and when your software tries to process the input it might end up executing vulnerable instructions. It is that simple to destroy your entire software. Secure programming makes a noticeable attempt in protecting your software, the risk of being hacked reduces by a huge amount but not completely. This opens the scope for two things, one to use secure programming to your software to protect it and two to develop higher security standards. This project can do both, we could use secure programming to develop a secure software and we also had the scope to wisely use the recommendations such that we could rise the security standards implemented in our project. We needed safe programming for consistent results for mathematical calculations, encryption systems, safe string processing

and many other core applications. The preferences and recommendations prescribed by the CERT has helped us in writing a safe and secure code. Starting from variable description to the complicated memory allocation, the CERT rules have played a major role in minimizing the syntax and run-time errors and provided a solution to encounter them. Our application needs a safe program and CERT recommendations is of high technical importance to us.

Synopsis-

The user creates his account by entering his details like Aadhar number, pan number and KYC number. The system allots a unique SRN number, and the user enters his password. After account creation and profile updating, the user logs in using his unique SRN and password. The user then requests for his personal details which would have been encrypted and stored in a file. On successful login, the user will be given his details in an output file that will be valid only for 2 minutes.

INTRODUCTION

DESIGN AND WORKING

The user is first presented with 3 options, 1. Existing user, 2. New user, 3. exit.

NEW USER

Under New user option, the user gets a unique SRN generated as PESXXX where XXX is a number ranging from 100 to 999. This SRN must be always remembered by the user. The user then must enter a 6-character password and re-enter it to confirm it. On successful account creation, we append the details of the user in a Admin.txt file and instead of using a database, we have used a text file that will be encrypted using the adminshift function. This file contains information about the users in this way-

Line 1- User1 SRN

Line 2- User1 password encrypted (using special function and last 3 digits of user SRN)

Line 3- User1 information file name that is also encrypted.

Line 4- New line

Line 5- User2 SRN

... and so, on

Then the user is prompted to enter his personal information that is Name, Aadhar card number, PAN

number and KYC details. These details are encrypted using a special function based on a shift function that gives a shift amount based on the users last 3 digits of SRN and stored in a file having format SRNInfo.txt

IDEA BEHIND THE ENCRYPTION ALGORITHM USED

The encryption algorithm is a slight variation of Caesar Cypher encryption Algorithm. The Caesar Cypher encryption follows shifting only alphabets from a shift of 1 to 25. Shifting a greater amount would lead into a loop around. The encryption algorithm we have used shifts all characters unlike the Caesar Cypher algorithm and the shift amount is between 1 and 255 since there are 256 ASCII characters and shifting a greater amount would lead into a loop around. The shift function gives a unique shift amount based on the users last 3 digits of SRN.

Similarly, the Admin.txt is also encrypted and decrypted at the end and start of the program, respectively using a special Admin shift function to ensure the contents of the file is not misused.

EXISTING USER

At the start of the program, the user SRN's registered is read out from Admin.txt file and appended to a Trie tree data structure and Queue data structure.

The user needs to first enter their unique SRN, and this is checked quickly by comparing user entered SRN to Trie

tree nodes. If not found, the user has 2 more attempts to enter the correct SRN.

On successful SRN matching with our record, the user must enter his password. The Queue data structure is used to get the line number in Admin.txt file at which the users encrypted password is stored. By dequeuing the SRNs from the Queue, we get the line number at which the password is stored, and we extract the same. We then compare the user entered password that will be encrypted with the encrypted password extracted from Admin.txt file. If incorrect, then the user is given 2 more attempts.

On successful login/password matching, the user is given a file, output.txt that has the decrypted contents of the user's personal information file (SRNInfo.c) and this file is available only for 2 minutes to ensure that the file is secure. Afterwards, the file will contain a message "2 minutes are up".

Since we have not used a database, yet we have managed to make the program secure by encrypting data at different layers, having double encryptions, and even encrypting file names so that even the encrypted data in those files are not misused.

IDEA BEHIND APPENDING SRN IN TRIE TREE

Handling SRN is the first step of the program and as such is reliant on an efficient data structure to perform operations on it.

Depending on the use cases, we determined that searching for SRN should take minimal time and since it would always have a particular format, a section of the SRN could be skipped to further decrease search time.

After thorough examination of the availability of our resources we decided to implement a Trie tree as it was already the most optimal data structure on its own. However, we realized during development that an ordinary Trie tree would not suffice as most of any SRN was in fact composed of numbers which were a limitation of any generic Trie. So, we decided to modify the Trie tree index range from 26 (All alphabets) to 36 (Alphabets as well as numbers included) and a function to map index values to the corresponding character.

After this modification, we determined that the speed of the Trie had improved, and the file handling time was minimized. It also allowed an improved transition from the initialization to the main function.

Requirement Specification

- 1: C Programming Language
- 2: GCC Compiler
- 3: GDB Debugger
- 4: Splint Flawfinder tool

Platforms used to coordinate work

- 1: Replit.com (common coding platform where we could integrate our work)
- 2: We used discord app as a platform to synchronize and communicate.
- 3: We used Microsoft teams to do complete this project report (created a common word doc where we could edit together)

SAFETY REQUIREMENT SPECIFICATION

SAFETY FEATURES

The program ensures encryption at many levels. The user data is encrypted using a special function that shifts based on the digits of the user's unique SRN. This data is stored in a file SRNinfo.txt, the file name too is encrypted. Since we have not used any database, we implemented a file called Admin.txt that stores the users SRN, encrypted password and encrypted file name that has users encrypted data. This file, Admin.txt is also encrypted using a special function called adminShift to ensure double encryption so that the data is not misused. When the user requests his information, it will be decrypted and provided to him in an output.txt file that will be valid for 2 minutes after which the contents of the output file gets erased. Time constraints prevented us from assigning different user modes to different files implemented in our project. We planned to assign executable permission to all our .c files, read permissions to the output file only to group and allow all permissions for owner and deny all other permissions to user and group.

SEI CERT C CODING STANDARD

IMPLEMENTATIONS

WHAT WE IMPLEMENTED FROM THE TEXT BOOK

1) DCL30-C. Declare objects with appropriate storage durations

We kept local dynamically allocated variables and we declared and freed them accordingly.

2) DCL31-C. Declare identifiers before using them

We have declared identifiers in some files like srnhandle.c before using them.

3) DCL37-C. Do not declare or define a reserved identifier
Initially we had declared a reserved identifier and later corrected the same.

4) FLP30-C. Do not use floating-point variables as loop counters

In time delay in the file existing.c, we have used double data type since time delay was large and we had no choice

5) MEM30-C. Do not access freed memory

Initially we had declared a reserved identifier and later corrected the same.

6) FIO46-C. Do not access a closed file

We ensured that we did not access any closed file pointers.

7) MSC37-C. Ensure that control never reaches the end of a non-void function

The control of our function reaches the end of our main function, PersonalInfo.c

8) INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data

Here we used fputc used to copy file contents of temp.c to SRNinfo.c, and we ensured that the numbers were not lost or misinterpreted

WHAT WE IMPLEMENTED FROM CLASS

1)Using format specifiers everywhere

We have used %s,%c,%d,.. and other format specifiers wherever applicable

2) STRO2-C SANITIZE DATA PASSED TO COMPLEX SUBSYSTEMS

We passed the encrypted password to the GetName function and compared it with the encrypted password stored, so that the actual/real password does not get leaked anywhere.

3) STR30-C: DO NOT ATTEMPT TO MODIFY STRING LITERALS

We ensured that we did not treat strings like character arrays modifying the contents.

4) STR31-C: Guarantee that storage for strings has sufficient space for character data and the null terminator

We kept `sizeof(string)+1` to ensure space for null character, `'\0'`.

5) STR32-C: DO NOT PASS A NON-NULL TERMINATOR CHARACTER SEQUENCE TO A LIBRARY FUNCTION THAT EXPECTS A STRING

We ensured that we did not pass any non-null terminating character sequence to `fgets`, `fputs`, `fgetc` and other library functions.

6) Using `fgets` instead of `gets`.

7) Using `strncmp` instead of `strcmp`

8) Using `strncat` instead of `strcat`

The above three practices are good coding practices instructed by our guide during class.

9) MEM34-C. Only free memory allocated dynamically

We freed `tries`, `queue`, and other dynamically allocated memory by calling the `free()` function dynamically.

10) MEM35-C. Allocate sufficient memory for an object

We ensured sufficient allocation of memory for tries and queues by using malloc subroutine.

11) MEM31-C. Free dynamically allocated memory when no longer needed

We freed tries, queues, character pointers and other dynamically allocated memory at the end of the program.

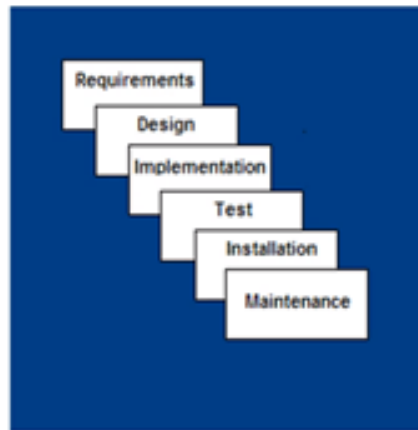
12) FIO42-C. Close files when they are no longer needed

We closed all file pointers pointing to admin.txt, SRNinfo.txt, output.txt, temp.txt.

DESIGN

SOFTWARE DESIGN MODEL

Waterfall Model



Data Structures

1. Trie trees- to update the SRN numbers of users.

```
#define ALPHABETSIZE 36

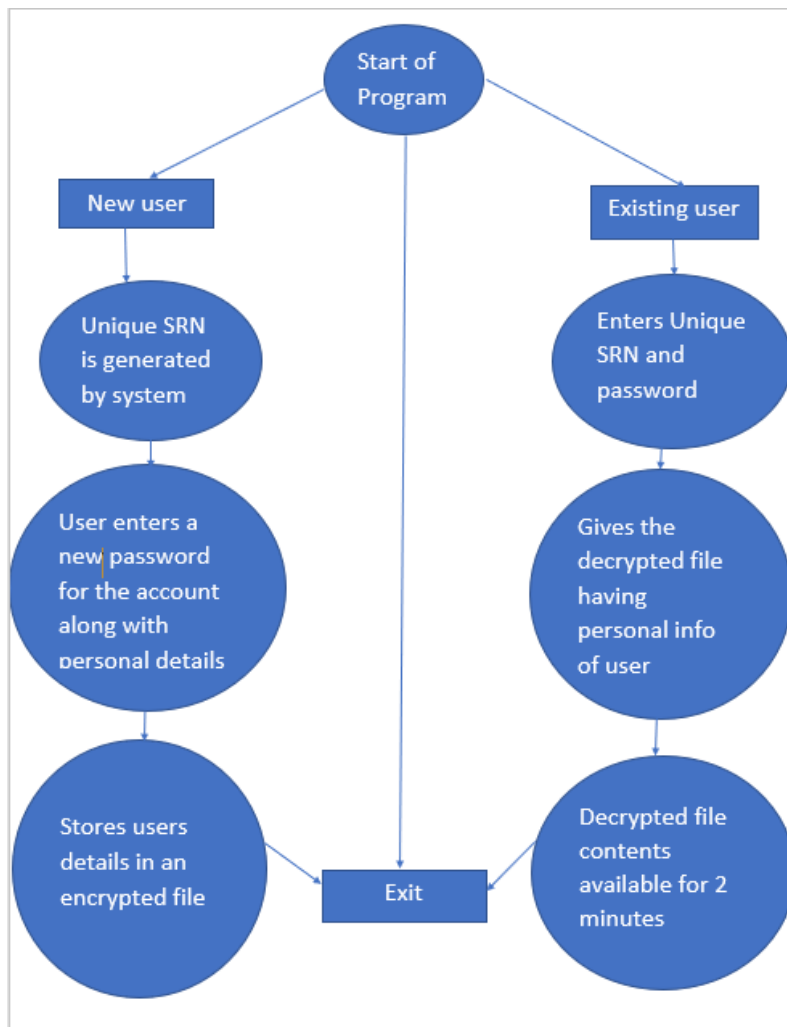
struct TrieNode
{
    struct TrieNode *child [ALPHABETSIZE];
    bool isEndOfWord;
};
```

2. Queue- for having user details, so that the location of the encrypted password could be found in the admin.txt file.

```
typedef struct node{
    char *data;
    struct node *next;
}node;
```

```
typedef struct queue {  
    struct node *front,*rear;  
}queue;
```

DATA FLOW DIAGRAM



Modularity

Modularity refers to the extent to which a software can be divided into smaller modules. In our project every function performs a particular operation, this also gives the programmers an additional feature of re-usability. The design of our project comprises of many systems which contains several sub-systems, and those sub-systems further contains their sub-systems. So, designing a complete system in one go comprising of required functionality is a hectic work, and the process can have many errors because of its vast size. Thus, in order to solve this problem, we had to breakdown the complete software into various modules. Every module is independent and can be modified without disturbing any other module in the program.

When every module talks about a specific part of the project, safeguarding it, applying safety recommendations on it becomes easier. Unlike a situation where the entire code is dumped into the main function, where everything becomes messy and safeguarding it then becomes a tough challenge. If the application of any safety recommendation causes an error as a side effect, modularity helps us in solving it.

Our project is very modular. Our project demonstrates security in three ways: Data sanitization, data validation and the most important of all, encryption, and decryption, all the three of the above ways are demonstrated as a separate module. We have split our program into 11 c files and each file has a minimum of 2 or 3 functions. We have made functions that perform elementary operations that

we require like extracting SRN's, passwords from Admin.txt file that acts like our database. We have implemented getPass function that is a separate module to handle the password. The password must be 6 characters or less and the user must re-enter the password to confirm it. There are separate files in our program like trie.c to handle operations on trie data structure like appending into trie and searching in trie, there is Qfunct.c that takes care of enqueueing, dequeueing functions for queue data structure. There are also files such as srnhandle.c that further handles trie.c file and extract.c handles Qfunct.c . The main file handles new.c and existing.c that further handles extract.c and srnhandle.c, respectively. These files also implement the encrypt.c and decrypt.c that has functions for encrypting, decrypting files, or specific strings. The encryption and decryption functions take shift amount from Adminshift, specialfunction and **srnshift** functions present in shift.c file.

Abstraction

Abstraction is the process of hiding unnecessary details from the user. In our program, we have implemented abstraction in verifying passwords for an existing user. The idea we implemented is simple and unique. When the user enters their password to get their details from the output file, we extract that SRN's specific password from Admin.txt file that acts like our database. This password is encrypted according to the users SRN. Now we encrypt the user entered password and compare it to the extracted password from admin.txt file. We do not decrypt the actual password, store it, or pass it around to functions to

ensure that this sensitive data may not be accessed through a possible memory leak.

Another area we have implemented abstraction is the encryption and decryption algorithm where the user is just given assurance by the developers that his data is secure but has no information about how his data is being stored. The encryption algorithm is a slight variation of Caesar Cypher encryption Algorithm. The Caesar Cypher encryption follows shifting only alphabets from a shift of 1 to 25. Shifting a greater amount would lead into a loop around. The encryption algorithm we have used shifts all characters unlike the Caesar Cypher algorithm and the shift amount is between 1 and 255 since there are 256 ASCII characters and shifting a greater amount would lead into a loop around. The shift function gives a unique shift amount based on the users last 3 digits of SRN. Similarly, the users and other outsiders do not have any clue of how our database file, Admin.txt is stored. This too encrypted and decrypted at the end and start of the program, respectively using a special Admin shift function to ensure the contents of the file is not misused. In case there is a hack, it will be almost impossible to get the correct contents of the file since this is a 256-bit encryption algorithm that has been encrypted several times at many different places with different shifts.

IMPLEMENTATION

CODE

WE USED THE FOLLOWING HEADER FILES IN ALL FILES

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
#include<time.h>
#include "functions.h"
```

Here 'functions.h' is a header file we have used to define all the functions we used.

Functions.h (Header file)

```
//anirudhs side
typedef struct node{
    char *data;
    struct node *next;
}node;

typedef struct queue {
    struct node *front,*rear;
}queue;

struct queue* create_queue();
```

```
struct node* create_node(char *);
```

```
void enqueue(queue *q,char *srn);
```

```
void dequeue(queue *q);
```

```
void deleteQueue(queue *q);
```

```
void Extract(queue *q,char *srn,char *password);
```

```
int GetLineNo(queue *q,char *srn);
```

```
void GetData(char *pass,char *file_name,int line_no);
```

```
char* GetName(char *srn,char *pass,char *password,char *file_name);
```

```
void DisplayFile(char *file_name);
```

```
//end of anirudhs side
```

```
//nikhils side
```

```
char * SentenceEncrypt(char *srn,int shf);
```

```
char * SentenceDecrypt(char *srn,int shf);
```

```
void EncryptFile(char file[],int shf);
```

```
void DecryptFile(char file[],int shf);
```

```
int SRNshift(char *ptr);
```

```
int Adminshift();
```

```
int specialfunct(int srn);
```

```
void NewUser(struct queue *,struct TrieNode *);
```

```
void ExistingUser(struct TrieNode *,queue *);
```

```
char *getPassword(char *);
```

```
void appendAdmin(char *, char *,char *);
```

```
void appendInfo(char *,char *,char *,char *,char *);
```

```
void display(char *,char *);
```

```
void delay();
```

```
void fileRemove();
```

```
//end of nikhils side
```

```
//Jaywanth's Side
```

```
#define BUFFERSIZE 255
```

```
#define ALPHABETSIZE 36
```

```
//#define CHAR_TO_INDEX(c) ((int)c - (int)'A')
```

```
#define LOWER_BOUND 200
```

```
#define UPPER_BOUND 600
```



```
struct TrieNode
{
    struct TrieNode *child [ALPHABETSIZE];
    bool isEndOfWord;
};

int char_to_index(char );

void srnExtract(FILE *,struct TrieNode *,queue *);

void srnAppend(struct TrieNode *,char *);

char *generateSRN();

int srnSearch(struct TrieNode *,const char *);

struct TrieNode *getTrieNode();

void insertIntoTrie(struct TrieNode *, const char *);

int searchTrie(struct TrieNode *, const char *);

void freeTrie(struct TrieNode *);

//end of jaywanths side
```

1) PersonalInfo.c (Main file)

Coded by Nikhil Adyapak, Jaywanth J, Anirudh Joshi

```
int main(void)
{
    FILE *fptr;//, *fp
    fptr=fopen("Admin.txt","r");
    if(fptr==NULL)
    {
        fptr=fopen("Admin.txt","w");
    }
    struct TrieNode* SRNTrie = getTrieNode();
    queue* Q= create_queue();
    //Extract Srn's first
    srnExtract(fptr, SRNTrie,Q);
    (void)fclose(fptr);//Problem 2
    printf("PERSONAL INFO MANAGER\n");
    printf("1.Existing user\n2.New user\n3.Exit\n");
    printf("Enter choice\n");
    int choice;
    (void)scanf("%d",&choice);
    switch(choice)
    {
        case 1:
            ExistingUser(SRNTrie,Q);
            break;
        case 2:
```

```

    NewUser(Q,SRNTrie);//Problem 3
    break;
case 3:
    exit(0);
default:
    printf("Invalid choice\n");
}
free(SRNTrie);
deleteQueue(Q);
return 0;
}

```

2)trie.c (Implementing all functions related to Trie trees data structure)

Coded by Jaywanth J

```

int char_to_index(char ch)
{
    if(ch-'A'>=0)
    {
        return (int)(ch-'A');
    }
    return (26 + (int)(ch - '0'));
}

```

```

struct TrieNode *getTrieNode(void)
{
    struct TrieNode *pNode = (struct TrieNode *)malloc(sizeof(struct TrieNode));
    if (pNode)

```

```

{
    int i;

    pNode->isEndOfWord = false;
    for (i = 0; i < ALPHABETSIZE; i++)
        pNode->child[i] = NULL;
}
return pNode;
}

```

```

void insertIntoTrie(struct TrieNode *root, const char *key)

```

```

{
    int level;
    int length = strlen(key);
    int index;
    struct TrieNode *pCrawl = root;
    for (level = 0; level < length; level++)
    {
        index = char_to_index(key[level]);
        if (!pCrawl->child[index])
            pCrawl->child[index] = getTrieNode();
        pCrawl = pCrawl->child[index];
    }
    pCrawl->isEndOfWord = true;
}

```

```

int searchTrie(struct TrieNode *root, const char *key)

```

```

{
    int level;
    int length = strlen(key);
    int index;
    struct TrieNode *pCrawl = root;

```

```

for (level = 0; level < length; level++)
{
    index = char_to_index(key[level]);
    if (!pCrawl->child[index])
        return 0;
    pCrawl = pCrawl->child[index];
}
return (pCrawl != NULL && pCrawl->isEndOfWord);
}

```

```

void freeTrie(struct TrieNode *root)
{
    int i;
    if(!root)
    {
        return;
    }
    for(i=0;i<ALPHABETSIZE;i++)
    {
        freeTrie(root->child[i]);
    }
    free(root);
}

```

3)QFunct.c (Implementing all functions related to Queues data structure)

Coded by Anirudh Joshi

```

struct queue* create_queue()

```

```
{  
    struct queue *q=(struct queue*)malloc(sizeof(struct queue));  
    q->front=q->rear=NULL;  
    return q;  
}
```

```
struct node* create_node(char *srn)  
{  
    struct node *n=(struct node*)malloc(sizeof(struct node));  
    n->data=(char *)malloc(sizeof(char)*6);  
    strcpy(n->data,srn);  
    n->next=NULL;  
    return n;  
}
```

```
void enqueue(queue *q,char *srn)  
{  
    node *temp=create_node(srn);  
    if(q->rear==NULL)  
    {  
        q->front = q->rear = temp;  
        return;  
    }  
    (q->rear)->next=temp;  
    q->rear=temp;  
}
```

```
void dequeue(queue *q)  
{  
    if(q->front==NULL)  
    {
```

```

        return;
    }
    node *temp=q->front;
    q->front=(q->front)->next;
    if(q->front==NULL)
    {
        q->rear=NULL;
    }
    free(temp);
}

void deleteQueue(struct queue *q)
{
    struct node* next;
    while (q->front!= NULL) {
        next = q->front->next;
        free(q->front);
        q->front= next;
    }
    free(q);
}

```

4)srnHandle.c (To insert the SRN's into the trie and Queue and to extract the SRN's from admin.txt file)

Coded by Jaywanth J

//Run Initially to add srn's from admin.txt

```

void srnExtract(FILE *fp,struct TrieNode *srnTrie,queue *q)
{
    if(fp==NULL)
    {
        printf("Error opening file\n");
        exit(-1);
    }
    int i = 1;
    srand(time(0));
    while(!feof(fp))
    {
        //fgets(line,BUFFERSIZE,fp); can be used instead
        char *line=(char *)malloc(sizeof(char)*6);
        char *temp=(char *)malloc(sizeof(char)*6);
        fgets(line,sizeof(line),fp);
        strncpy(temp,line,6);
        if((i-1)%4==0 && temp[0]=='P')
        {
            //printf("%s %d\n",temp,i);
            srnAppend(srnTrie,temp);
            enqueue(q,temp);
        }
        i++;
    }
}

```

//To add any srn to Trie

```

void srnAppend(struct TrieNode *srnTrie,char *srn)
{
    insertIntoTrie(srnTrie,srn);
}

```



```

//Generates unique SRN within Range 100 to 999

/*
    Add SRN to trie , queue , admin.txt
*/
char *generateSRN(struct TrieNode *srnTrie)
{
    int num;
    int generated=0;
    char *temp1,*temp2;
    while(!generated)
    {
        temp1 = (char *)malloc(sizeof(char)*4);
        temp2 = (char *)malloc(sizeof(char)*3);
        srand(time(0));
        num = (rand()%(UPPER_BOUND - LOWER_BOUND + 1)) + LOWER_BOUND;
        sprintf(temp2,"%d",num);
        strcpy(temp1,"PES");
        strncat(temp1,temp2,3);
        if(!srnSearch(srnTrie,temp1))
        {
            srnAppend(srnTrie, temp1);
            return temp1;
        }
        else
        {
            free(temp1);
            free(temp2);
        }
    }
}

```

```

int srnSearch(struct TrieNode *srnTrie,const char *srn)
{
    if(srnTrie==NULL)
    {
        return 0;
    }
    return searchTrie(srnTrie,srn);
}

```

5)Extract.c (To get the line number and extract the encrypted password from admin.txt file and compare it with user password)

Coded by Anirudh Joshi

```

void Extract(queue *q,char* srn,char* password)
{
    printf("extract\n");
    int line_no=GetLineNo(q,srn);
    printf("%d\n",line_no);
    char pass[15];
    char file_name1[15];
    char *file_name2 = (char *)malloc(sizeof(char)*15);
    GetData(pass,file_name1,line_no);
    printf("\n%s\n%s",pass,file_name1);
    file_name2=GetName(srn,pass,password,file_name1);
    if(strncmp(file_name2,"failed",7)==0)
    {

```

```
        printf("\nIncorrect Password\n");
    }
    else
    {
        DisplayFile(file_name2);
    }
}
```

```
int GetLineNo(queue *q,char *srn)
{
    int i=1;
    while(strncmp(q->front->data,srn,7)!=0)
    {
        dequeue(q);
        i+=3;
    }
    return i;
}
```

```
void GetData(char *pass,char *file_name,int line_no)
{
    FILE *fp=fopen("Admin.txt","r");
    if(fp==NULL){
        perror("Error");
        exit(-1);
    }
    char *line=(char *) (malloc(sizeof(char)*15));
    int i=1;
    while(fscanf(fp,"%s",line))
    {
        if(i==line_no)
```

```

    {
        fscanf(fp,"%s",pass); //encrypted password for the srn_no
        fscanf(fp,"%s",file_name); //encrypted file name with personal details of that
srn_no
        break;
    }
    i+=1;
}
}

```

```

char* GetName(char srn[],char pass[],char password[],char file_name[])

```

```

{
    //compare user password after encrypting with password from file
    //char *arr=(char*)malloc(sizeof(pass)+1);
    char *arr=(char*)malloc(sizeof(char)*16);
    arr=SentenceEncrypt(password,SRNshift(srn));
    if(strncmp(arr,pass,7)==0){
        //Now decrypt file name and accesses it to get personal details
        char *arr=(char*)malloc(sizeof(char)*(15));
        arr=SentenceDecrypt(file_name,SRNshift(srn));
        return arr;
    }
    else
        return "failed";
}

```

```

void DisplayFile(char *file_name)

```

```

{
    FILE *fp;
    if((fp=fopen(file_name,"r"))==NULL)
    {

```

```

        printf("error");
        exit(1);
    }
    char Name[20],Aadhar[20],Pan[20],Kyc[20];
    fscanf(fp,"%[^\\n]*c",Name);
    fscanf(fp,"%[^\\n]*c",Aadhar);
    fscanf(fp,"%[^\\n]*c",Pan);
    fscanf(fp,"%[^\\n]*c",Kyc);
    printf("\\nName:%s",Name);
    printf("\\nAadhar:%s",Aadhar);
    printf("\\nPan:%s",Pan);
    printf("\\nKyc:%s",Kyc);
    fclose(fp);
}

```

6)Encrypt.c (To encrypt the file/string based on a special shift function got from the last 3 digits of the SRN)

Coded by Nikhil Adyapak

```

void EncryptFile(char file[],int shf)
{
    FILE *fptr2;
    FILE *fptr3;
    fptr3=fopen("temp.txt","w");
    fptr2=fopen(file,"r");
    if(fptr2==NULL || fptr3==NULL)

```

```
{  
printf("\nUnable to open the file\n");  
exit(-1);  
}  
  
char str1,str2;  
  
int ch;  
  
str1 = fgetc(fp2);  
while (str1 != EOF)  
{  
ch=(int)str1;  
if(ch+shf>255)  
ch=ch+shf-255;  
else  
ch+=shf;  
str2=(char)ch;  
fputc(str2,fp3);  
str1 = fgetc(fp2);  
}  
  
fclose(fp2);  
fclose(fp3);  
  
FILE *fp4;  
FILE *fp5;  
  
fp5=fopen("temp.txt","r");  
fp4=fopen(file,"w");  
if(fp4==NULL || fp5==NULL)  
{  
printf("\nUnable to open the file\n");  
exit(-1);  
}  
  
str1 = fgetc(fp5);
```

```
while (str1 != EOF)
{
fputc(str1,fptr4);
str1 = fgetc(fptr5);
}
fclose(fptr4);
fclose(fptr5);
}
```

```
char * SentenceEncrypt(char *srn,int shf)
{
int n = strlen(srn);
int i=0;
char* temp = (char*)malloc(sizeof(srn)+1);
char str1=srn[i];
int ch;
while (str1 != '\0')
{
ch=(int)str1;
if(ch+shf>255)
ch=ch+shf-255;
else
ch+=shf;
temp[i]=(char)ch;
i+=1;
str1=srn[i];
}
temp[i]='\0';
return temp;
}
```

7)Decrypt.c (To decrypt the file/string based on a special shift function got from the last 3 digits of the SRN)

Coded by Nikhil Adyapak

```
void DecryptFile(char file[],int shf)
{
FILE *fptr2;
FILE *fptr3;
fptr3=fopen("output.txt","w");
fptr2=fopen(file,"r");
if(fptr2==NULL || fptr3==NULL)
{
printf("\nUnable to open the file\n");
exit(-1);
}
char str1,str2;
int ch;
str1 = fgetc(fptr2);
while (str1 != EOF)
{
ch=(int)str1;
if(ch-shf<0)
ch=ch-shf+256;
else
ch-=shf;
str2=(char)ch;
fputc(str2,fptr3);
```



```

str1 = fgetc(fp2);
}
fclose(fp2);
fclose(fp3);
}

char * SentenceDecrypt(char *srn,int shf)
{
int n = strlen(srn);
int i=0;
char* temp = (char*)malloc(sizeof(srn) + 1);
char str1=srn[i];
int ch;
while (str1 != '\0')
{
ch=(int)str1;
if(ch-shf<0)
ch=ch-shf+255;
else
ch-=shf;
temp[i]=(char)ch;
i+=1;
str1=srn[i];
}
return temp;
}

```

8)shift.c (A special shift function that gets a unique shift to encrypt the users personals details based on the last 3 digits of the SRN)

Coded by Nikhil Adyapak

```
int SRNshift(char *ptr)
{
    int srn=0;
    for(int i=3;i<=5;i++)
        srn=(srn*10)+(ptr[i]-'0');
    int shiftamt=specialfunct(srn);
    return shiftamt;
}
```

```
int Adminshift()
{
    int i=2*3*5+1;
    return i;
}
```

```
int specialfunct(int srn)
{
    int val=(srn+100-srn*2-4+(232+srn))%256;
    return val;
}
```

9)New.c (To generate SRN for new user and have them enter a new password and personal details)

Coded by Nikhil Adyapak and Anirudh Joshi

```

void NewUser(struct queue *q,struct TrieNode* srnTrie)
{
    char *srn = generateSRN(srnTrie);
    printf("\nThis is your SRN,remember it all times");
    printf("%s\n",srn);
    //enqueue(q,srn);
    char *encrPass = getPassword(srn);
    printf("\nEnter your details in this order");
    printf("\n1.Name\n2.Aadhar\n3.Pan\n4.Kyc");
    printf("\nEnter your Name ");
    char *Name = (char *)malloc(sizeof(char)*50);
    scanf(" %[^\\n]%*c",Name);
    printf("\nEnter your Aadhar number ");
    char *Aadhar = (char *)malloc(sizeof(char)*50);
    scanf("%s",Aadhar);
    printf("\nEnter your Pan number ");
    char *Pan = (char *)malloc(sizeof(char)*50);
    scanf("%s",Pan);
    printf("\nEnter your Kyc number ");
    char *Kyc = (char *)malloc(sizeof(char)*50);
    scanf("%s",Kyc);
    //append srn,password,SRNinfo.txt into admin.txt
    //append name,aadhar,pan,kyc into SRNinfo.txt
    char *fileName=(char *) (malloc(sizeof(char)*15));
    char *temp1=(char *) (malloc(sizeof(char)*9));
    strcpy(fileName,srn);
    strcpy(temp1,"info.txt");
    strncat(fileName,temp1,8);
    appendAdmin(srn,encrPass,fileName);
    appendInfo(Name,Aadhar,Pan,Kyc,fileName);
}

```

```
    EncryptFile(fileName,SRNshift(srn));  
}
```

```
void appendAdmin(char *srn, char *encrPass,char *file_name)  
{  
    FILE *fptr;  
    fptr=fopen("Admin.txt","a");  
    if(fptr==NULL)  
    {  
        printf("File doesn't exist");  
        exit(1);  
    }  
    fputs(srn,fptr);  
    fputs("\n",fptr);  
    fputs(encrPass,fptr);  
    fputs("\n",fptr);  
    fputs(file_name,fptr);  
    fputs("\n",fptr);  
    fputs("\n",fptr);  
    fclose(fptr);  
}
```

```
void appendInfo(char *Name,char *Aadhar,char *Pan,char *Kyc,char *fileName)  
{  
    FILE *fptr;  
    fptr=fopen(fileName,"w");  
    if(fptr==NULL)  
    {  
        printf("File doesn't exist");  
        exit(1);  
    }  
}
```

```

fputs(Name,fptr);
fputs("\n",fptr);
fputs(Aadhar,fptr);
fputs("\n",fptr);
fputs(Pan,fptr);
fputs("\n",fptr);
fputs(Kyc,fptr);
fputs("\n",fptr);
fclose(fptr);
}

```

10)Existing.c (To check if user has an account, has entered the right password, and display the decrypted file having Personal details available only for 2 minutes)

Coded by Nikhil Adyapak and Jaywanth J

```

void ExistingUser(struct TrieNode* SRNTrie, queue * Q)
{
    char *SRN = (char *)malloc(sizeof(char)*7);
    printf("Enter Your SRN\n");
    for(int i=0;i<3;i++)
    {
        char *srn = (char *)malloc(sizeof(char)*7);
        fscanf(stdin,"%s",srn);
        printf("%s\n",srn);
        bool flag=searchTrie(SRNTrie,srn);
        if(flag==false && i!=2)

```

```

{
    printf("SRN does not exist\n");
    printf("Re-enter SRN\n");
}
else if(flag==false && i==2)
{
    printf("Too many invalid attempts\n");
    exit(-1);
}
else
{
    strncpy(SRN,srn,7);
    break;
}
}

int line=GetLineNo(Q,SRN);
FILE *fptr;
fptr=fopen("Admin.txt","r");
printf("Enter your password\n");
char *file_name=(char *) (malloc(sizeof(char)*15));
char *fileName=(char *) (malloc(sizeof(char)*15));
char *check=(char *) (malloc(sizeof(char)*15));
char *temp1=(char *) (malloc(sizeof(char)*9));
strcpy(fileName,SRN);
strcpy(temp1,"info.txt");
strncat(fileName,temp1,8);
for(int i=0;i<3;i++)
{
    char *pass = (char *) malloc(sizeof(char)*7);
    scanf("%s",pass);
    char *encrPass=(char *) malloc(sizeof(char)*7);

```

```

GetData(encrPass,file_name,line);
char *check=GetName(SRN,encrPass,pass,file_name);
if(strncmp(check,"failed",7)!=0)
    break;
else
{
    printf("Incorrect Password\n");
    printf("Please Re-enter Password\n");
    if(i==2)
    {
        printf("You have entered incorrect passwords\n");
        exit(-1);
    }
}
}
fclose(fptr);
display(SRN,fileName);
}

```

```

void display(char *srn,char *fileName)
{
    DecryptFile(fileName,SRNshift(srn));
    printf("This file will be available only for 2 minutes\n");
    delay();
    fileRemove();
}

```

```

void delay()
{
    for(double i=0;i<10500000000;i++); //delay of nearly 2 min
}

```

```
void fileRemove()
{
    FILE *fptr1;
    fptr1=fopen("output.txt","w");
    fputs("2 minutes up",fptr1);
    fclose(fptr1);
}
```

11)getPass.c (To get the password of 6 characters and reconfirm it failing which, 2 more attempts will be given)

Coded by Nikhil Adyapak

```
char *getPassword(char *srn)
{
    char *newPass = (char *)malloc(sizeof(char)*7);
    char *rePass = (char *)malloc(sizeof(char)*7);
    int cmpValue=1;
    while(cmpValue)
    {
        printf("Enter Password -> 6 characters \n");
        scanf("%s",newPass);
        printf("Re-enter password\n");
        scanf("%s",rePass);
        cmpValue=strncmp(newPass, rePass, 7);
    }
    return (SentenceEncrypt(newPass,SRNshift(srn))); }
```


ADDITIONAL FILES THAT ARE USED

1)Admin.txt

Instead of using a database, we have used a text file that will be encrypted using the adminshift function. This file contains information about the users in this way-

Line 1- User1 SRN

Line 2- User1 password encrypted (using special function and last 3 digits of user SRN)

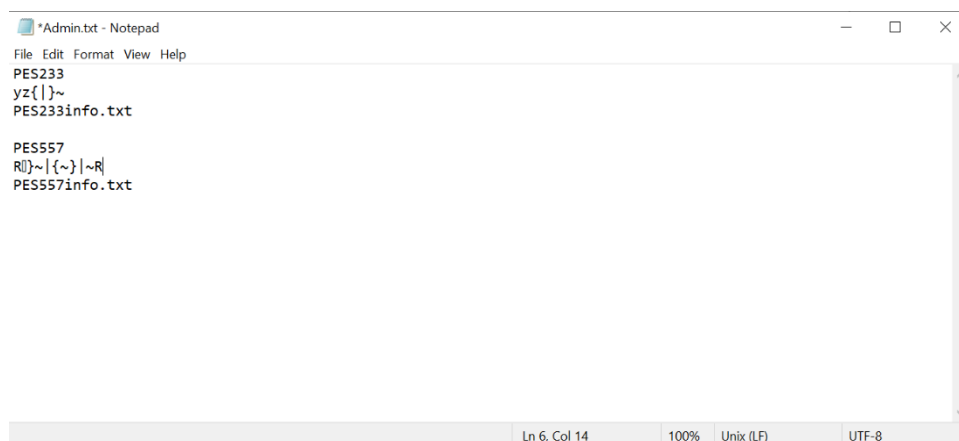
Line 3- User1 information file name that is also encrypted.

Line 4- New line

Line 5- User2 SRN

... and so on

Sample of admin.txt created-



```
*Admin.txt - Notepad
File Edit Format View Help
PES233
yz{|}~
PES233info.txt

PES557
R[]}~|{|~}|~R|
PES557info.txt

Ln 6, Col 14 100% Unix (LF) UTF-8
```

2)SRNinfo.txt

PES233info.txt - Notepad

File Edit Format View Help

%- 'ŠŒh'→ 'R' } { z€ | [] } R } ~ | { ~ } | z R [] } ~ | { ~ } | ~ R

Ln 1, Col 1100%Windows (CRLF)ANSI

PES557info.txt - Notepad

File Edit Format View Help

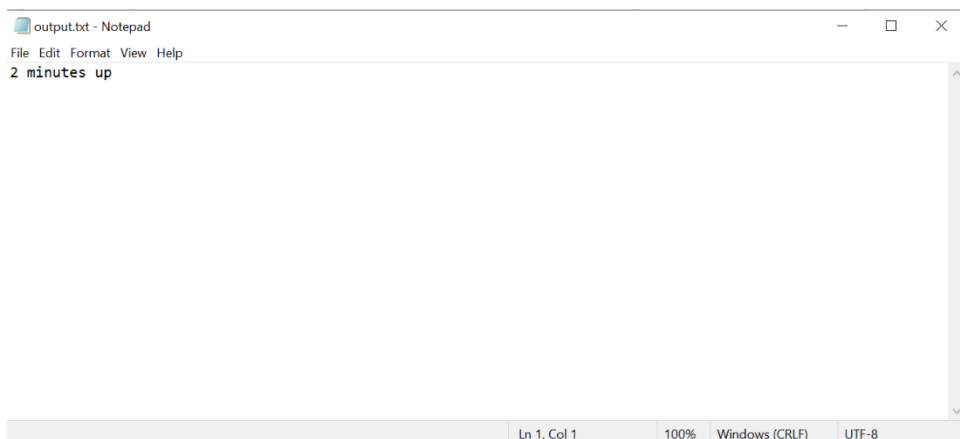
'%| Ÿ%-œh-ŸŒR[]€ { z ~ } | { z } R } [] { | ~ | { ~ R | [] { ~ [] | { R

Ln 1, Col 1100%Windows (CRLF)ANSI

3)output.txt



After 2 minutes,



TESTING AND ANALYSIS

Flaws initially found

- 1) If SRNs were exhausted, it fails.
- 2) Name of string data type with spaces was not accepted
- 3) Using Fgets continuously without flushing stdin, used to not accept data correctly.
- 4) Since Queue variable was used everywhere and was global, there was memory leak.
- 5) Password length greater than 6 and infinite loop for accepting password.

Vulnerabilities In Initial Code

- 1) When User inputs password of length greater than 6

```
(base) mint@mint-Precision-T3600:~/Desktop/personal-information-manager$ ./PersonalInfo
PERSONAL INFO MANAGER
1.Existing user
2.New user
3.Exit
Enter choice
2

This is your SRN,remember it all timesPES372
Enter Password -> 6 characters
amfsfjrfg
Re-enter password
amfsfjrfg

Enter your details in this order
1.Name
2.Aadhar
3.Pan
4.Kyc
Enter your Name anirudh

Enter your Aadhar number 947396584

Enter your Pan number 9469847369

Enter your Kyc number 9375284395
```

As visible we have input a password of length greater than 6.

```
(base) mint@mint-Precision-T3600:~/Desktop/personal-information-manager$ ./PersonalInfo
PERSONAL INFO MANAGER
1.Existing user
2.New user
3.Exit
Enter choice
1
Enter Your SRN
PES372
PES372
Enter your password
amfsfjrfg
Incorrect Password
Please Re-enter Password
amfsfjrfg
Incorrect Password
Please Re-enter Password
amfsfjrfg
Incorrect Password
Please Re-enter Password
amfsfjrfg
You have entered incorrect passwords
```

Password is not accepted

2) When User Inputs their full name

```
(base) mint@mint-Precision-T3600:~/Desktop/personal-information-manager$ ./PersonalInfo
PERSONAL INFO MANAGER
1.Existing user
2.New user
3.Exit
Enter choice
2

This is your SRN,remember it all timesPES201
Enter Password -> 6 characters
123456
Re-enter password
123456

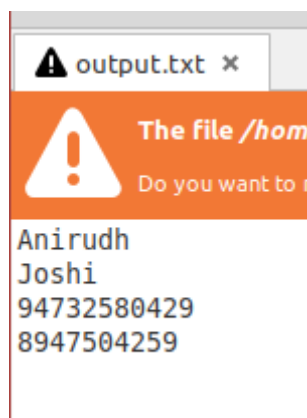
Enter your details in this order
1.Name
2.Aadhar
3.Pan
4.Kyc
Enter your Name Anirudh Joshi

Enter your Aadhar number
Enter your Pan number 94732580429

Enter your Kyc number 8947504259
```

Here as seen user's name has been taken with a space

```
(base) mint@mint-Precision-T3600:~/Desktop/personal-information-manager$ ./PersonalInfo
PERSONAL INFO MANAGER
1.Existing user
2.New user
3.Exit
Enter choice
1
Enter Your SRN
PES201
PES201
Enter your password
123456
This file will be available only for 2 minutes
(base) mint@mint-Precision-T3600:~/Desktop/personal-information-manager$ |
```



The program was taking the users last name as the Aadhar number.

This was an issue with scanf.

These problems were fixed later.

To fix the problem of the password length we made changes in the getPassword function where we made sure the input password was of length 6, We gave the user 2 more attempts to input password of length 6.

And to avoid last name of user being accepted as Aadhar number and the name of user to be his full name(last name included).

We altered the parameter in scanf function.

```
char *Name = (char *)malloc(sizeof(char)*50);
scanf("%[^\\n]%c", Name);
```

This ensures that input will be taken until a new line character is read.

STATIC ANALYSIS TOOL Splint

Splint warnings initially:

```
(base) mint@mint-Precision-T3600:~/Desktop/personal-information-manager$ splint PersonalInfo.c
Splint 3.1.2 --- 20 Feb 2018

PersonalInfo.c: (in function main)
PersonalInfo.c:22:14: Possibly null storage fptr passed as non-null param:
    srnExtract (fptr, ...)
    A possibly null pointer is passed as a parameter corresponding to a formal
    parameter with no /*@null@*/ annotation. If NULL may be used for this
    parameter, add a /*@null@*/ annotation to the function parameter declaration.
    (Use -nullpass to inhibit warning)
PersonalInfo.c:17:10: Storage fptr may become null
PersonalInfo.c:23:3: Return value (type int) ignored: fclose(fptr)
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
PersonalInfo.c:28:3: Return value (type int) ignored: scanf("%d", &choice)
PersonalInfo.c:40:7: Unreachable code: break
    This code will never be reached on any possible execution. (Use -unreachable
    to inhibit warning)
PersonalInfo.c:45:12: Fresh storage SRNTrie not released before return
    A memory leak has been detected. Storage allocated locally is not released
    before the last reference to it is lost. (Use -mustfreefresh to inhibit
    warning)
PersonalInfo.c:19:44: Fresh storage SRNTrie created
PersonalInfo.c:45:12: Fresh storage Q not released before return
PersonalInfo.c:20:28: Fresh storage Q created

Finished checking --- 6 code warnings
```

We were able to fix 4 of these warnings and believe the memory leak for Queues is a false positive.

```
(base) mint@mint-Precision-T3600:~/Desktop/personal-information-manager$ splint PersonalInfo
Splint 3.1.2 --- 20 Feb 2018

spec file not found: PersonalInfo.lcl
PersonalInfo.c: (in function main)
PersonalInfo.c:22:13: Possibly null storage fptr passed as non-null param:
    srnExtract (fptr, ...)
    A possibly null pointer is passed as a parameter corresponding to a formal
    parameter with no /*@null@*/ annotation. If NULL may be used for this
    parameter, add a /*@null@*/ annotation to the function parameter declaration.
    (Use -nullpass to inhibit warning)
PersonalInfo.c:17:10: Storage fptr may become null
PersonalInfo.c:45:12: Fresh storage Q not released before return
    A memory leak has been detected. Storage allocated locally is not released
    before the last reference to it is lost. (Use -mustfreefresh to inhibit
    warning)
PersonalInfo.c:20:28: Fresh storage Q created

Finished checking --- 2 code warnings
```

We were able to fix the return value warnings in `fclose()` and `scanf()` by prefixing `(void)`.

```
(void)fclose(fp);  
printf("PERSONAL INFO MANAGER\n");  
printf("1.Existing user\n2.New user\n3.Exit\n");  
printf("Enter choice\n");  
int choice;  
(void)scanf("%d",&choice);
```

Removing the redundant `break` statement after `exit(0)` in case 3 of our switch statement helped us get rid of the unreachable code warning.

```
case 3:  
    exit(0);  
default:  
    printf("Invalid choice\n");  
}  
free(SRNTrie);  
deleteQueue(Q);  
return 0;
```

We had global variables in Queues and trie tree data structures. Freeing memory allocated using `malloc` for the data structures to avoid a memory leak.

Maintenance

Modification:

Since we follow the waterfall model, our problem statement is already very descriptive, and any such solutions based on it are straight forward.

However, this does not necessarily indicate there is no scope for modification in our code. In fact, the codebase has a lot of potential for improvement with a developed understanding of more such critical security failure scenarios. Our codebase has used some common but enhanced data structures. There is always some space for modifications in this regard.

Particularly with the Trie Tree, there is potential to incorporate more such special characters, boosting security there as well. Increasing the SRN range is also very simple if the need be.

Our password and file encryption algorithm could also be replaced with a more secure and efficient which does not require 2-layer encryption based on the skill of the developer.

A layer of GUI may also be developed while still maintaining the main functionality.

Creating Multiple Versions:

Our codebase supports multiple version development. We have levelled out each functionality using separate modules in separate files allowing additional functionality to aggregate effectively.

We have already based our code on Version Control Systems (GitHub platform) so we had already

considered the possibility of multiple versions each with their own advantages and disadvantages.

Our straight-forward approach allows even beginner developers to make modifications without ruining the main purpose of the application. This allows for a massive surge in ideas based on the base code and create much more refined applications.

Scalability:

Our application was already highly scalable as much of the development consisted of team effort through the replit platform. Throughout development, we realized the need for simple changes much quickly using the platform and modified appropriately. This allowed the opportunity for even more potential in terms of scalability. Any team or individual can very easily fork our codebase, suggest changes, or maintain their own codebase. Large development teams also can use the codebase to a much larger project without affecting their own application. Some CERT recommendations are also not affected by scaling up and with a little bit of sophistication can easily be addressed on larger applications without any memory leaks or security attacks.

An example of a very simple scale up is to generate a password-management system whose core philosophy is based on the personal information manager and add more functionalities such as support for Secure Notes, Bank Information, 2FA and much more.

Conclusion

Well now that we have reached the end, we wonder what was the point of this project?

Here are a few points we have learned:

- Security is a fundamental concern of any application.
- To create a completely faultless system is a limitation by itself.
- Based on certain CERT principles and compliant solutions, we can develop a reliable and highly secure application.
- Warnings and errors are a key part of development which allows innumerable scenarios to be tested virtually.

Throughout the development we have gained a fair amount of practice on CERT standards which play a significant role in understanding the mindset of hackers and phishers. It is safe to say that the course has been successful in not only convincing but upgrading our knowledge base on some fundamentals as well as advanced development concepts.

The fact that a whole range of possibilities regarding code syntax, semantics and logical vulnerabilities also accompany some significant security practices (which may have been ignored prior to this course) have also been quite beneficial.

Github link

<https://github.com/jaywanthgowda11/Personal-Information-Manager>

References

- 1) Secure Coding in C and C++, Second Edition,
Robert C. Seacord
- 2) SEI CERT, C Coding Standard, Rules for Developing
Safe, Reliable, and Secure Systems, 2016 Edition,
Carnegie Mellon University
- 3) SEI CERT C Coding Standard, wiki.sei.cmu.edu,
official website of Carnegie Mellon University
- 4) www.geeksforgeeks.org/trie-insert-and-search/
- 5) www.geeksforgeeks.org/queue-data-structure/