PES UNIVERSITY

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

PROJECT REPORT
on

EMPLOYEE LEAVE MANAGEMENT SYSTEM

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

BHARGAV NARAYANAN P **PES2UG19CS088**
SATYAM RUDRADUTTA DWIVEDI **PES2UG19CS370**
SAI GRUHEETH N **PES2UG19CS352**

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 "**EMPLOYEE LEAVE MANAGEMENT SYSTEM**" is a bonafide work carried out by **Satyam R.D Dwivedi** bearing USN:PES2UG19CS370, **Bhargav Narayanan P** bearing USN:PES2UG19CS088, **Sai Gruheeth N** bearing USN:PES2UG19CS352 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 **"Employee Leave Management System"** 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: Satyam Rudradutta Dwivedi
Place:Bengaluru
Date:06/05/2021

Signature of the Student: Sai Gruheeth N
Place:Bengaluru
Date:06/05/2021

Signature of the Student: Bhargav Narayanan P
Place:Bengaluru
Date:06/05/2021

EMPLOYEE LEAVE MANEGMENT SYSTEM safeguarded by CERT recommendations

INDEX

AUTHORS

BHARGAV NARAYANAN P

PES2UG19CS088

SAI GRUHEETH N

PES2U19CS352

SATYAM RUDRADUTTA DWIVEDI

PES2UG19CS370

DATE AND PLACE OF DRAFT

6th May 2021

BENGALURU

Abstract of the project

The issue of security is very paramount in any code made for enterprise-scale operational capability, A well-documented and enforceable coding standard is an essential element of coding in the C programming language.

In our project we aim at a secure transaction and usage of holidays of any enterprise-level corporation by reading the employee credentials from the file and filling or populating an array of structures which is treated as a global variable and this data structure is made global with the help of EXTERN keyword and on the termination of the execution by the user in an orderly fashion generates a new file using system time as a reference for naming the text file.

The implementation and amalgamation of features without compromising on the integrity and security provisions of the coding standards was our prime concern and we not only achieved this target but also made our code resistant to the malicious intention of the intruder to cause an unforeseen stack smashes and buffer overflows which are a very easy and commonly used paths by intruders to enter a system and steal data.
We have implemented this project over four .c files of implementation to keep the code highly easy for maintenance and expansion aspects if we have any in the future and the ability to add extra fields is highly feasible by adding functions in an appropriate .c file and pass flag values to indicate the successful completion of a task.

Our efforts have been to uphold the coding ethics and also deliver a quality code that does not terminate abruptly and gives user an option at every step ahead and keeps the user at the centre of attention and stores the results back into a file for future reference and names it according to the date and time it was created giving us the colossal capability of having every unique file to be created every single second for wider range of use.

Introduction of the project

Problem definition

To develop a complete employee leave management system with file access methods with the use of safety recommendations provided by CERT C coding recommendations.

Need of safe programming:

In today's world of complex software and sophisticated user systems it is quite easy to miss loopholes in the implementation of Ideas into codes that run on our machines, and the most vital right any human needs is privacy, which is at the core aim of the production level codes to provide user data security and integrity intact, this is just the trust and faith the user has in an organization which allows him to share his/her important details for instance banking details to payment vendors and personal details with social websites like Instagram and Facebook and online payment vendors like google pay, Paytm and PayPal, thus for any organization the foundation for any software company has to be a solid and rigid code which does not leave out breakpoints for the hackers to enter the database and make sensitive information public or use it for worse malicious actions.

Thus, there is a need for improving the standards of code and making it more secure which forms a solid foundation of trust with the firm handling the sensitive data.

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.

Implementation .c files used in the project

1) main_function.c

This function acts like min function in any modular code, it reads and validates the input given by the command argument and passes on the name of the file to table function in the main2_table function to continue with the execution of the program.

2) main2_table.c

This acts as main handler and calls the table() function to populate the array of structures and this is the core function called every time the user wants to login and also handles level 1 of input validation and the other level being handled In login deduction which takes care of negative inputs.

3) login_deduction.c

This function has the code for authentication of user credentials allow access for the deduction of leaves from appropriate quota and of the right person and deliver the message of deduction in case of success and an appropriate error in case of a failure of deduction and it updates the structure values of the logged-in user, all input points are safeguarded from buffer overflow and stack smashing cases by flushing the stdin buffer.

4) quit_again_final.c

This function gives our project the ability to let user dictate the terms of termination of the program, here we make the user at

centre of the program by giving him/her the power to re-login without termination of the code without user permission.
If the user decides to finally quit after the prompts the quit function is called to ask for one last time to confirm that unintentional inputs are not taken straightaway and user is given a chance to correct his mistake if it was committed and calls print() function.
Final_print() is a multipurpose function which does the job of creating a file name using the reference of the system time and manages the format and creates the file and handles the appropriate errors and writes the output to the file and to the terminal at the same time and ends the execution of the program.

Requirement Specification

Software Requirement

Software features and influence of the following.

1: C Programming Language

2: Compiler

3: Memory

Software features

Data types used in the application

The Abstract data type used in the code are:

```
int argc, char *argv[]
char file_name[30];
int casual;
int medical;
int earned;
int leave_days;
int choice;
int log;
int res=0;
struct Empdetials emp[5];  // definition of emp
int linematched;
FILE *fp=NULL;//defining a file pointer
//char ch;
char  line[256];
char *token;
int tokenposition;
```

int lineposition;

int success=fclose(fp);

int login()

int i;

int c;

char userid[9];

char userpassword[5];

int xyz;

char ans[5];

int c;

char ans1[5];

FILE *fp=NULL;

char text[150];

time_t now = time(NULL);

struct tm *t = localtime(&now);

text[150] = '₩0';

char *filename;

int success=fclose(fp);

Array of structures used

```
extern struct Empdetials
{
    char name[25];
    char id[9];//one extra character to store /n
    char password[5];
    int casual;//casual,medical,earned leave as per the question
    int medical;
    int earned;
}emp[5];//declaring array of structures which will later be populated
```

using table() function

Functions used and its descriptions:

Here is the list of functions used in our application:

1) extern void main2() ;

This contains the main part of the code workings, its purpose is explained below just before the function

main function has a call to table() and invoking main() for re-login used to reset the structure and the deductions were lost,so we copied everything from the main function into another function called main2() which is basically the main function without the table() invoking so the structure is not reset after every call from again() and quit() during re-login and structure keeps a track of the deductions that happened and in this way we also removed the Welcome sentence to the same user, but the main reason is to avoid rewriting of the structure on every call and losing the deduction data.

2) extern void final_print();

Prints the details of the user on quitting the wizard and also creates a file and names it based on the time of creation and writes the details to the newly generated file to keep a track of history and changes made so far.

3) extern void quit();

This is one function which handles the termination of code prompting the user to take another trial for re-login and deduction if the user wants to do so and if user is not interested

in login then it invokes the final_print( ) function which continues its task.

4) extern void again();

This function handles the login if user wishes to take another leave,this is a feature that allows user to enter his choice once again as in case of a mistake from the user's side the code will end its execution so this is an additional safety feature to confirm the exit.

5) extern void table(char file_n[]);

Populating the structure from data elements from the file by using specific delimiter, which is a comma in our case, this can be expanded to hold other details and this takes care of the expansion phase of the code and also maintenance.

6) extern int login();

This function verifies the password with the one present in the file database and lets the user pass through only if the provided password is correct and sends an integer to the calling function to tell if the login authentication was successful.

7) extern int deduction(int linematched,int days,int type_of_leave);

This performs another validation on the requested leave count and checks for negative entries and deducts the days user desires for taking the leave and returns an integer stating if it was a success or not and the calling function will take action appropriately.

Influence of the following:

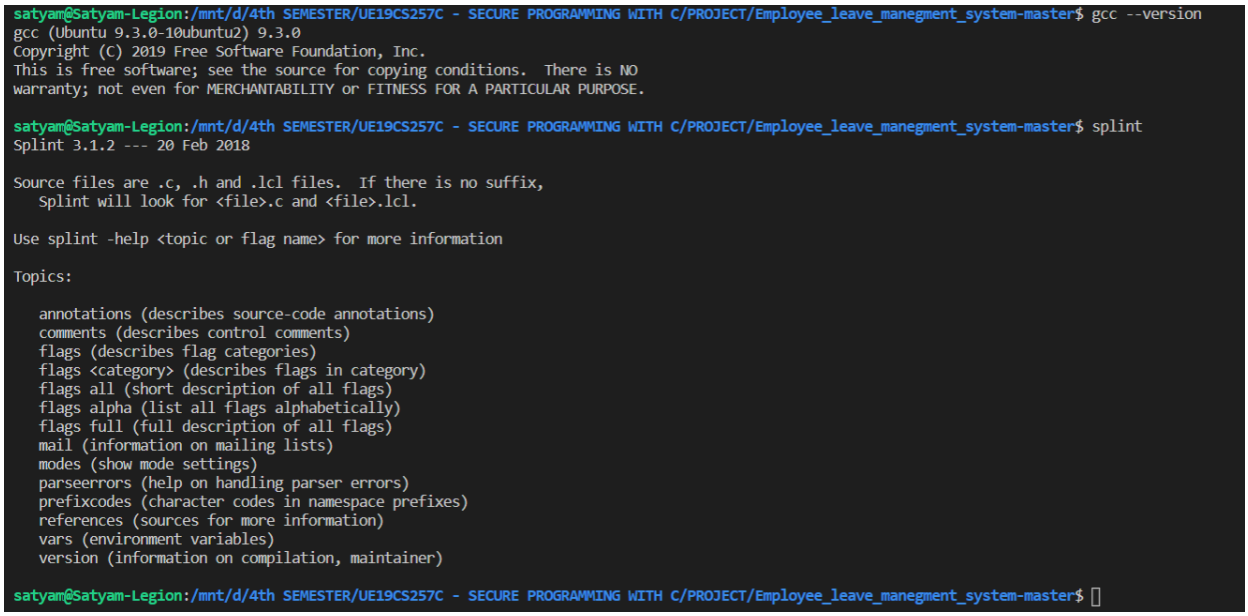
PROGRAMMING LANGUAGE

C is a programming language that is very close to the memory (hardware) and does not have a heavy software built around it. C language assumes that the user has taken care of all aspects the program and doesn't check for anything, as a developer it is so much more challenging and adding to these the non object oriented approach of C makes it tough to deal with data structure passing between the functions and this lack of data encapsulation and abstraction urges us to develop safe programs. For instance if the input exceeds the memory size allocated C language does not throw an error and there is high risk of the data being lost or in some cases overwritten and this leads to a breakpoint in the code and also leads to a possible loophole entry point in the code for invaders and steal passwords by breaking the code, therefore we have had to take care of minute details of all possible edge cases in developing our application.

COMPILER

We use **WSL terminal provided by ubuntu linux environment** on Microsoft Visual Studio Code, we used gcc compiler just that compiler is unique to the C language. This is possibly the best compiler that we found that would suit our application, and the VSC IDE makes us recognize the problems beforehand and prompts suitable changes and solutions to the possible errors, for example, and we used splint static analysis tool version 3.1.2 to find the vulnerabilities in the project. this way we have found an optimal way to make the safest possible code powered by CERT recommendations and rules in check and not compromising on the integrity or functionality of the code.

Screenshot showing the WSL environment on Microsoft Visual studio code with GCC VERSION-9.3.0, SPLINT VERSION-3.1.2



MEMORY

We use An array of structures user defined data type for storing user name, ID, passwords and holiday quota and kept it global by exploiting the use of extern keyword and the functions referring to this data structure are declared extern in order to access the data type and we took assistance of StackOverflow to implement this idea of global data.

And we declared local file pointer to handle file reading and writing and to indicate opening and closing of the files we used proprietary variables in the functions. We've taken care of overflow conditions during input from the user and flushed the buffer in case of an overflow to avoid a possible breakpoint in the code and this breakpoint leaves a very large loophole to enter the code and read the passwords or change the data with malicious intent and this defeats the entire purpose of automation.

Security Requirement Specifications

The Need for Ensuring Security

C as a programming language was developed with the intention of utilizing the native infrastructure with almost direct interaction with the machine. As a result, C works close to the hardware and compared to other programming languages, it is fast and memory efficient. But the use of raw pointers and arrays call for good coding practices that are critical to avoid pointer-related memory problems. The responsibility is on the programmer to ensure proper functioning of the program since unsafe programming practices lead to common anomalies such as data corruption, buffer overflows and pointers accessing data outside their intended scope. These flaws are often left unchecked and can interfere with the normal execution of the program.

Hence, there is a need to enforce a well documented and enforceable set of coding standards since a bad design can lead to undefined program behaviors and exploitable vulnerabilities.

Safety Features

Given below is a list of safety features that we have implemented in our project:-

1. Input validation for command line arguments.

   Validation is the process of ensuring that the data entered by the user falls within the expected domain of valid program input. The entry point to our program is through command line arguments to the main function() and hence it is a critical section that has to be handled carefully.

   We have ensured that correct number of arguments are entered and the length of the file name does not exceed 30 characters. In case of a breach, an appropriate message is returned to the user. Special care has been taken to prevent buffer overflows by reading only a specific number of characters at once from the input stream into the required string as entered by the user. The excessive characters are then flushed out.

2. Working on the copy of the original data and use of Global Variables.

   Due to our modular approach, functions share common data values. Hence the idea throughout the implementation has been to work on the copy of the data rather than on the original data to prevent corruption of frequently used data.

   We have made use of few global variables wherever appropriate instead of pointers to prevent the possibility of referencing objects outside their lifetime.

3. Safe use of file pointers and protection against injection attacks.

   Care has been taken to ensure that the file pointer points to the desired location and does not point to NULL. This is an important step since it forms the basis for the values that we are going to work with. We have used fgets() to read data from the file. This function includes the buffer size in its arguments and hence characters beyond the size mentioned are not processed.

   This idea particularly comes into light when an attacker supplies untrusted input causing an injection attack against the system. For example in our case, an attacker might load an enormous file with no new-line characters (\n) with an attempt to corrupt data by overflow.

   We have designed our implementation such that the file instance is active at only one point in the entire code flow, followed the practice of closing the file after use and removed any other references to it across all the program files.

4. Uniformity in username and password size.

   We have set bounds on user inputs for the username and password and flushed the input stream each time after a scanf statement to minimize possibilities of circumvention leading to unauthorized access.

5. No instances of unreachable codes.

   The control flow will not encounter the case of certain sections of the program being unreachable and the use of break statements

has been done judiciously in this regard, limiting their use to only switch statements.

6. Use of malloc() to map date and time to the final statement.
   While generating the file with the final leave statement, we record the system date and time at that instant and append it to the filename. To achieve this, we have utilized malloc() to dynamically change the size of the string that holds the file name. This is a measure to prevent buffer overflows due to static allocation of small buffers and also wastage of memory by allocating more than what is required to store the file name.

Safety Recommendations Used for the Project:

We have adopted coding standards developed by the CERT Coordination Center in order to raise the security standards of our project. Given below is a list of the safety recommendations that we have conformed to.

While we have briefly listed out the recommendations that our code complies with, for some of the rules we have included the code snippets of our previous non-compliant code and added the snippet of the compliant code with the changes made.

1. DCL30-C : *Declare objects with appropriate storage durations.*
   For our implementation, we minimized the need for automatic storage duration by defining the array of structures under global scope and using the extern keyword wherever necessary.

2. DCL31-C : *Declare identifiers before using them.*
   Every function has to be explicitly declared before it can be called.

3. DCL36-C : *Do not declare an identifier with conflicting linkage classifications.*

   This means that use of an identifier (within one translation unit) classified as both internally and externally linked is undefined

behavior. Here we have made the correct use of externally linked classifiers:-

```
extern int linematched; //externally linked
extern struct Empdetials  //externally linked
 {
    char name[25];
    char id[9];//one extra character to store /n
    char password[5];
    int casual;//casual,medical,earned leave
    int medical;
    int earned;
}emp[5];//declaring array of structures which will later be popul
ated using table() function
```

4. DCL37-C : *Do not declare or define a reserved identifier.*
   We must remember that all identifiers that begin with an underscore and either an uppercase letter or another underscore are always reserved for any use.

5. DCL40-C : *Do not create incompatible declarations of the same function or object.*
   Two or more incompatible declarations of the same function or object must not appear in the same program as they result in undefined behavior.

6. DCL41-C : *Do not declare variables inside a switch statement before the first case label.*
   Switch case statements are only labels, the compiler interprets it as a jump to the label skipping all the declarations it contains. Furthermore, we have provided scope to our cases with curly braces for the purpose of safety.

```
 switch(type_of_leave)
    {
        case 1://casual leave
        {
            if(days>emp[linematched].casual || days<=0)//second validation
happens here with respect to the specified user
            {
                printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE E
NTERED A NEGATIVE VALUE\n\n");
                again();
                xyz=0;
                //return 0;
                //break;
            }
            else
            {
                emp[linematched].casual=emp[linematched].casual-days;
                xyz=1;
                //return 1;
                //break;
            }
            break;
        }
        case 2://medical leave
        {
            if(days>emp[linematched].medical || days<=0)
            {
                printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE E
NTERED A NEGATIVE VALUE\n\n");
                again();
                xyz=0;
                //return 0;
                //break;
            }
            else
            {
                emp[linematched].medical=emp[linematched].medical-days;
                xyz=1;
                //return 1;
                //break;
            }
            break;
        }
        case 3://earned leave
        {
            if(days>emp[linematched].earned || days<=0)
            {
                printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE E
NTERED A NEGATIVE VALUE\n\n");
                again();
                xyz=0;
```

```
                //return 0;
                //break;
            }
            else
            {
                emp[linematched].earned=emp[linematched].earned-days;
                xyz=1;
                //return 1;
                //break;
            }
            break;
        };
        default:
         {
               printf("SORRY! YOU HAVE NOT ENTRERED A VALID TYPE OF LEAVE \
n\n\n");
                again();
                xyz=0;

         }
    };
```

7. EXP33-C : *Do not read uninitialized memory.*
   If not initialized explicitly, variables take indeterminate/garbage values.

8. EXP34-C : *Do not dereference null pointers.*
   Ensuring this rule is followed is essential, since this leads to undefined behavior.

9. EXP37-C : *Call functions with the correct number and type of arguments.*

10. EXP45-C : *Do not perform assignments in selection statements.*
11. FLP30-C : *Do not use floating-point variables as loop counters.*

12. STR31-C : *Guarantee that storage for strings has sufficient space for character data and the null terminator.*

This means that, copying data to a buffer not large enough to hold the data results in buffer overflow.

We have implemented this while accepting any input from the user, we limit the number of characters that can be read at once.

Non-Compliant Code:-

```
int login()
{
    int i;//for the loop
    char userid[9];//user ID int he majn function,this will be passed to th
e the login function
    char userpassword[5];//user password in the main function

    printf("ENTER YOUR 8 CHARACTER USER ID\n");
    scanf("%s",userid);
    printf("ENTER THE 4 CHARACTER PASSWORD\n");
    scanf("%s",userpassword);

    /* Remaining Statements
                            */
}
```

In the above snippet for example, if the user enters more than 8 characters for the username, there will be an overflow.

Compliant Code:-

```
int login()
{
    int i;//for the loop
    int c;
    char userid[9];//user ID in the main function,this will be passed to th
e the login function
    char userpassword[5];//user password in the main function

    printf("ENTER YOUR 8 CHARACTER USER ID\n");
    (void)scanf("%8s",userid);
    while ((c = fgetc(stdin)) != (int)'\n' && c != EOF); /* Flush stdin */
    printf("ENTER THE 4 CHARACTER PASSWORD\n");
    (void)scanf("%4s",userpassword);
    while ((c = fgetc(stdin)) != (int)'\n' && c != EOF); /* Flush stdin */
 }
```

In this code snippet, the format specifier '%8s' tells scanf to read only the first 8 characters in the input stream. Hence, this code is compliant with the above rule.

13. FIO42-C : *Close files when they are no longer needed.*

A call to fopen() must always be matched with a call to fclose(). From the below code segment, please make a note that the intermediary codes have been commented out for clarity.

The compliant code:-

```
extern void table(char file_n[])//this function is responsible for populati
ng the structure by reading the file
{
   FILE *fp=NULL;//defining a file pointer
   //char ch;
   /*
   char  line[256];//definig 256 here,any arbitrary number can be used
   char *token;//token pointer that points to the elements of a line
   int tokenposition=0;//token here means the particular text separated by
comma
   int lineposition=0;//line here is the entire line which has entire detai
ls of the an emmployee
   fp=fopen(file_n,"r");//opening the file in read mode as we are only acce
ssing information and not rewriting it
   //-------------------------------------------------------------------------
-------------------------------------------------------------------------

if (fp==NULL)
{
    printf("File access error!!\n");
    exit(0);
}
    while(fgets(line,256,fp) !=NULL) //accessing the lines one by one and t
rying to recognize the end of a line
    {
        tokenposition=0;//when we tokenize the line we need the count to ke
ep a track of where we are int he string
        //printf("%s \n",line);//used to check which line is not being read
 by the compiler
        token = strtok(line,",");
            //this seperates a line into token(entity) based on a delimiter
,here the delimiter is a comma
        //strtok-tring tokenization method
        while(token != NULL)
        {
            switch(tokenposition)//acessing the required element through th
e tokenposition counter
            {
                case 0://NAME
                    strcpy(emp[lineposition].name,token);
                    break;
                case 1://ID
                    strcpy(emp[lineposition].id,token);
```

```
                    break;
                case 2://PASSWORD
                    strcpy(emp[lineposition].password,token);
                    break;
                case 3://CASUAL LEAVE
                    emp[lineposition].casual=atoi(token);
                    break;
                case 4://MEDICAL LEAVE
                    emp[lineposition].medical=atoi(token);
                    break;
                case 5://EARNED LEAVE
                    emp[lineposition].earned=atoi(token);
                    break;
            }
            token =strtok(NULL,",");
        tokenposition++;//to access the next token in the same line
        }
        lineposition++;//after all the tokens are put in a structure,we mov
e to the next line and redo the entire process of reading a line
    }
    */
        int success=fclose(fp); //Don't forget to close the file when finis
hed
    if(success==0)
    {
        printf("file closed successfully\n");
    }
    else{
        printf("ERROR WHILE CLOSING THE FILE\n");
    }
 }
```

14. FIO47-C : *Use valid format strings.*

    Use of incorrect conversion specifier character can lead to undefined behavior.

15. MSC37-C : *Ensure that control never reaches the end of a non-void function.*

    If control reaches the closing curly brace ' }' of a non-void function without evaluating a return statement, using the return value of the function call is undefined behavior.

Non-compliant code:-

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include"server.h"
int login()
{
    int i;//for the loop
    char userid[9];//user ID int he main function,this will be passed to th
e the login function
    char userpassword[5];//user password in the main function
    printf("ENTER YOUR 8 CHARACTER USER ID\n");
    scanf("%s",userid);
    printf("ENTER THE 4 CHARACTER PASSWORD\n");
    scanf("%s",userpassword);
    linematched=0;
    for(i=0;i<5;i++)
    {
        if((strcmp(emp[i].id,userid)==0)&&(strcmp(emp[i].password,userpassw
ord)==0))//checking if login credentils are correct or wrong
        {
            linematched=i;
            printf("WELCOME %s TO THE LEAVE MANEGMENT SYSTEM\n",emp[linemat
ched].name);
            return 1;//TO INDICATE THAT USER WAS FOUND AND LOGIN WAS SUCCES
FUL
            break;
        }

    }
 }
```

From the above code snippet, we find that the function returns a value (ie. 1) to the caller only when the login was successful but does not return anything if the login is unsuccessful.

Compliant Code:-

```
int login()
{
    int i;//for the loop
    int c;
    char userid[9];
    char userpassword[5];//user password in the main function

    printf("ENTER YOUR 8 CHARACTER USER ID\n");
    (void)scanf("%8s",userid);
```

```
    while ((c = fgetc(stdin)) != (int)'\n' && c != EOF); /* Flush stdin */
    printf("ENTER THE 4 CHARACTER PASSWORD\n");
    (void)scanf("%4s",userpassword);
    while ((c = fgetc(stdin)) != (int)'\n' && c != EOF); /* Flush stdin */
    linematched=0;
    for(i=0;i<5;i++)
    {
        if((strcmp(emp[i].id,userid)==0)&&(strcmp(emp[i].password,userpassw
ord)==0))//checking if login credentils are correct or wrong
        {
            linematched=i;
            printf("WELCOME %s TO THE LEAVE MANEGMENT SYSTEM\n",emp[linemat
ched].name);
            return 1;//TO INDICATE THAT USER WAS FOUND AND LOGIN WAS SUCCES
FUL
            //break;
        }

    }
    return 0;
 }
```

In the compliant code above, the function always returns a value whether the login is successful or unsuccessful.

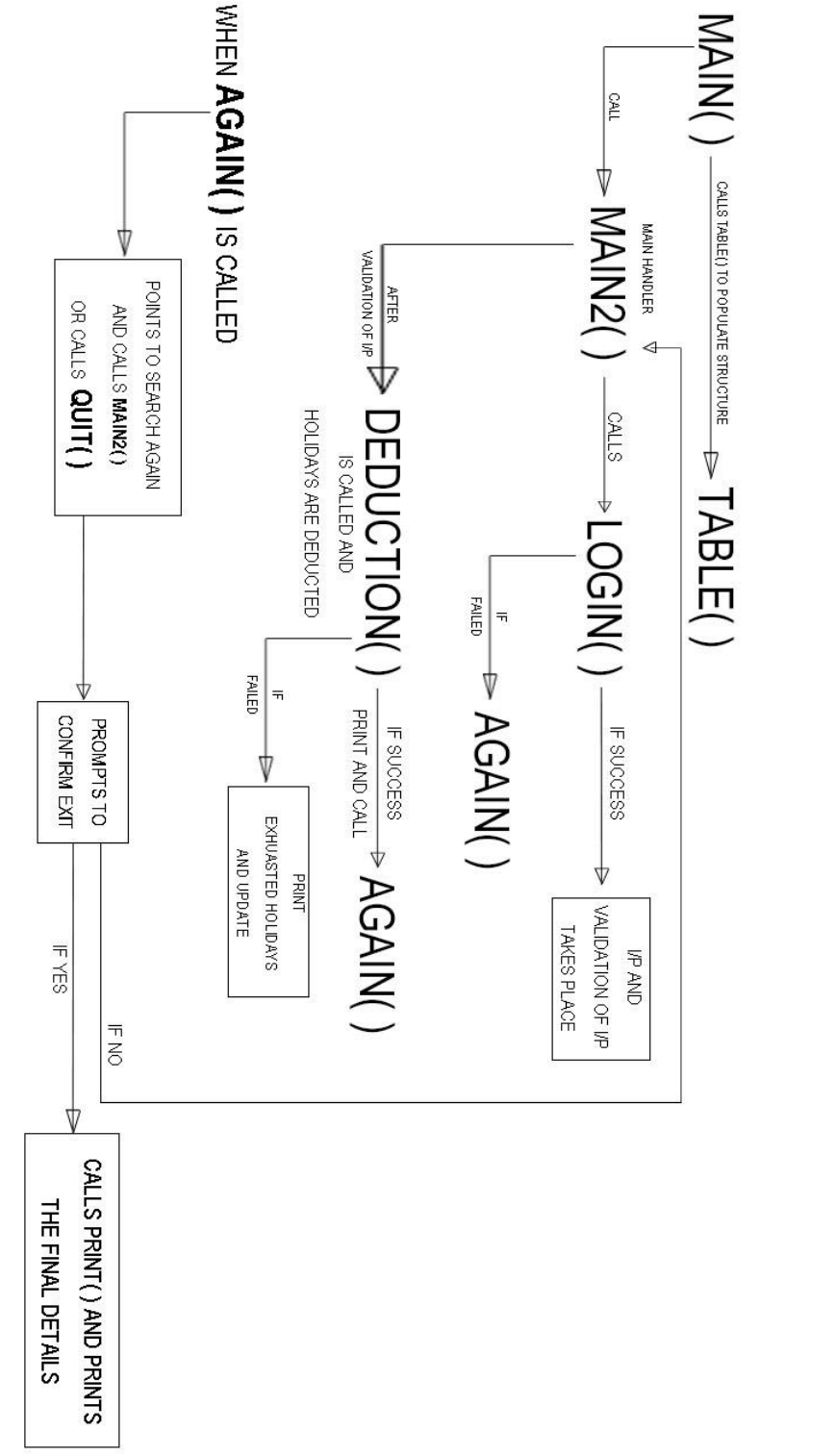
Justification for the Recommendations Used:

The primary goal of studying these rules and recommendations is to learn how to develop safe, reliable, and secure systems. We have put these rules to good use to the best of our abilities and we have contributed to the maximum extent possible to minimize the chances of undefined program behaviors or unexpected program terminations with unsafe inputs.

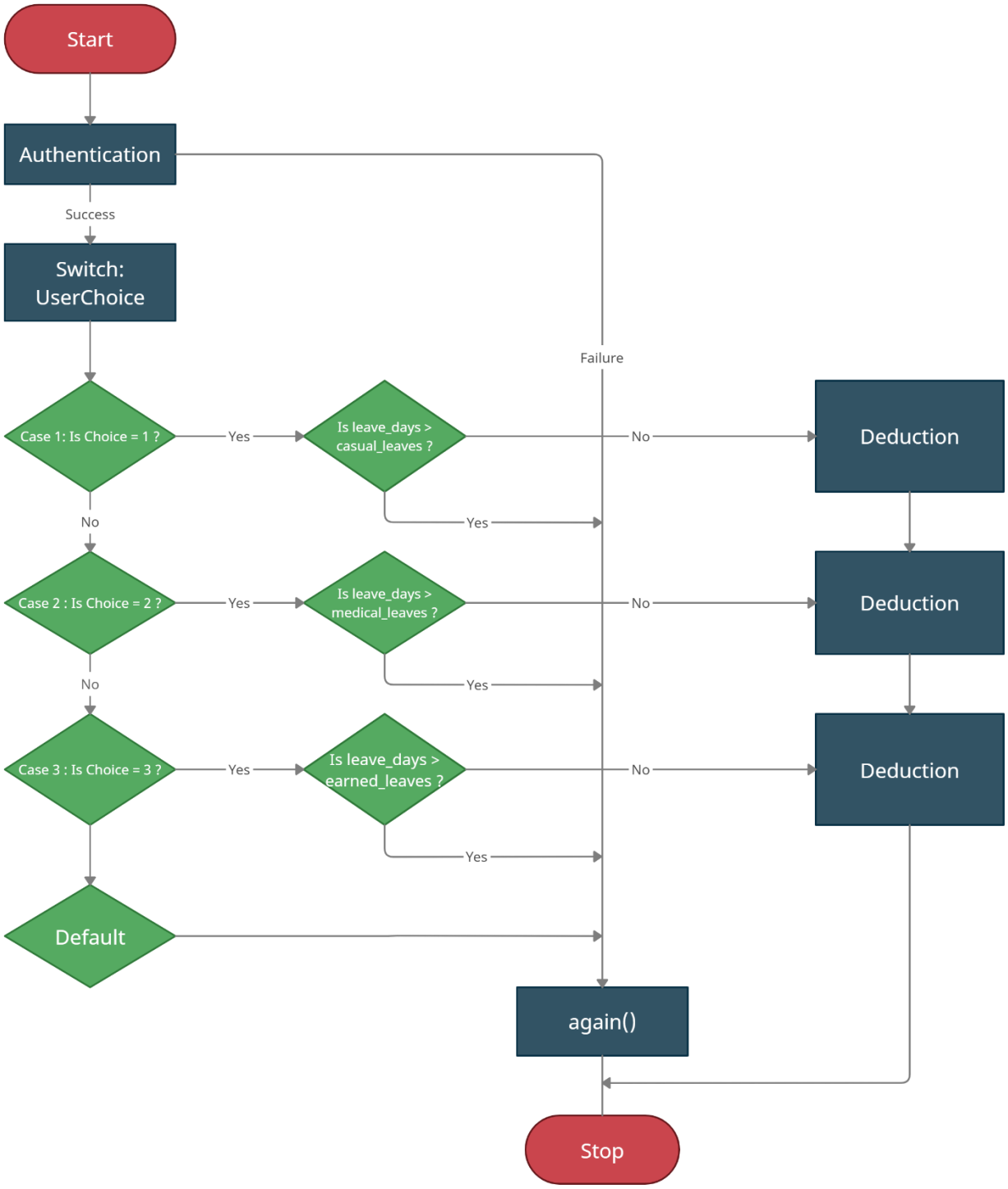
Few of the recommendations were fundamental to the implementation (e.g.:-passing correct no. of arguments to a function) which have been stated in a concise manner, but we have included snippets of our previously non-compliant code and snippets of the compliant solution that we have come up with in the previous section of this report. We have tried to address all the possible vulnerabilities in our program for the code to conform with the rules that have been listed.

DESIGN

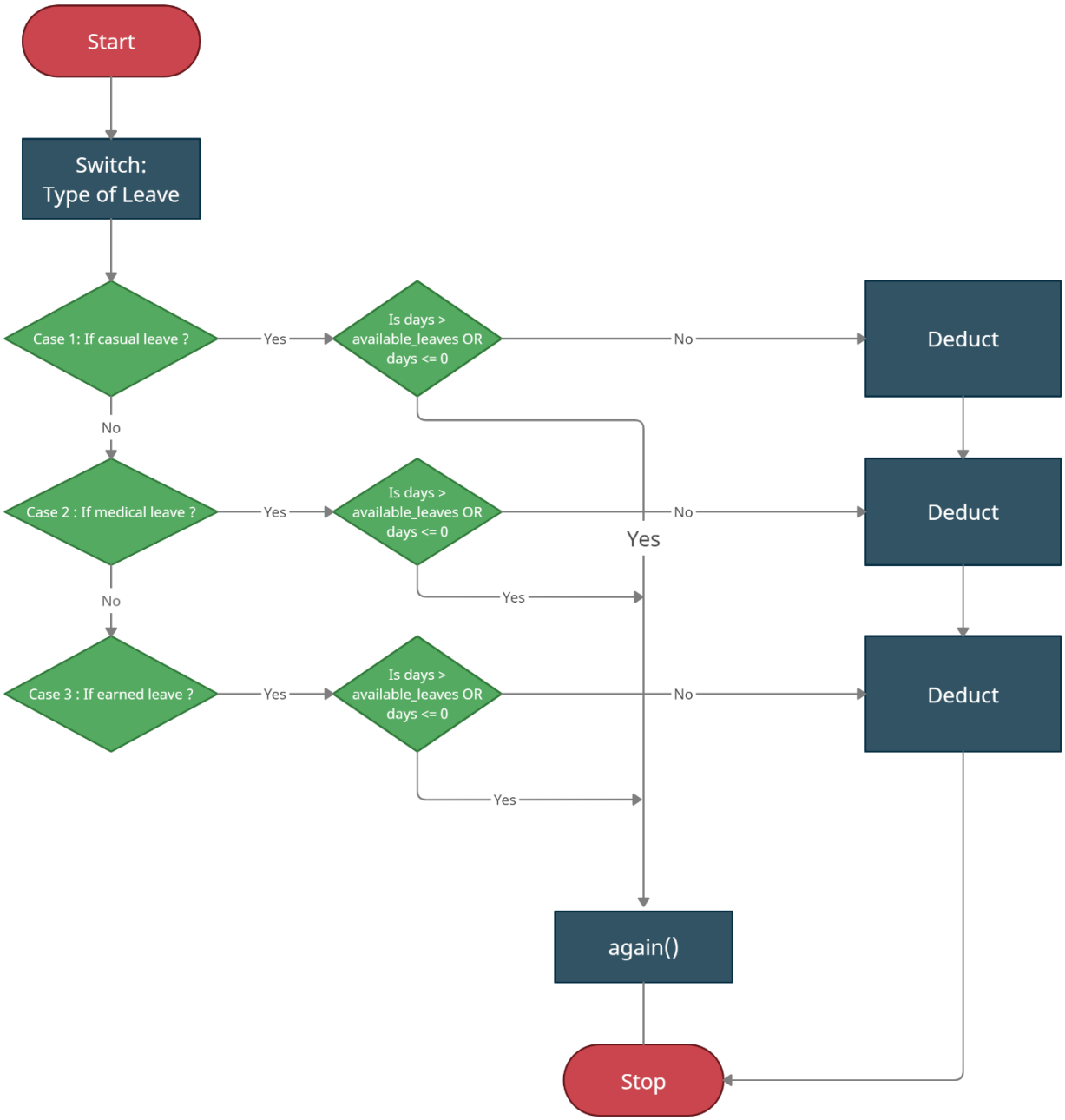
DATA FLOW DIAGRAM AND FLOWCHARTS



ABOVE SHOWN IS THE DATA FLOW DIAGRAM OF THE ENTIRE CODE WITH FUNCTION CALLS AND FLOW OF DATA



ABOVE SHOWN IS THE DECISION MAKING FLOWCHART OF main2() FUNCTION IN main2_table.c IMPLEMENTATION FILE



ABOVE SHOWN IS THE FLOWCHART OF THE THE SWITCH BLOCK DECISION MAKING IN THE deduction() FUNCTION IN login_deduction.c IMPLEMENTATION FILE

Modularity:

Modularity is the practice of breaking the code into several reusable blocks and this makes the code easier to maintain and upgrade and debug as well.
This factor makes the code very powerful as the same snippet is used repeatedly saving the space and time and complexity.
In our project implementation we have heavily manifested the advantage of having a modular code as we have many functions that invokes for re-logging and quit.
This way we have reduced the length of the entire code by up-keeping the same functionality, our project focuses on developing a more protected system, if developing a useful software solution is one side of a coin, protecting it from hackers is another side and passing the values between functions is a major security concern and this needs to be addressed as sometimes sensitive data is flowing between subroutines and this leads to major security threats to the integrity of the whole system.

Applying safety recommendations to the code without subroutines is a hectic and tedious task and there are chances of leaving something behind in any copy of the code and in this aspect as well the modularity helps us with handling the vulnerability as the only one snippet of code is made with utmost care and precaution thus upholding the quality of the overall implementation of the project.

Readability:

We have put in a lot of our efforts to increase the readability of our code ADTs where by adding explanation and adding comments to part of codes and indicate their role in the execution of the code and this way the tracing of the code is a much easier task and makes the code on the whole a lot more discernible and concise to explain.
By having a lot of comments to explain the code working we can implement the expansion phase by knowing which variables to change and how to handle the returning values and all these are handled in our code.

Abstraction:

We used user defined data type which is an array of structures and that is defined using extern keyword and this data structure is made global, this not only eases hassle of passing the pointers to every function as reading an array of structures using pointers leads to serious consequences if a null value is encountered and this crashes the entire execution and that is an indicative of a poorly implemented code and we didn't allow any such places to go unchecked and applied buffer overflow and overwrite protection by flushing the entire stream and this in one way is a form of abstraction as the user is not aware of the background process that deals with inputs that might otherwise crash the program or leave behind a loophole in the code on breakdown. CERT standards and recommendations helped us to keep the simplicity to the user and left non trace of any DS used in the code and in this way the abstraction requirement was fulfilled in our project's implementation.

Code Implementation:

.c FILES

We have included the source code used for the implementation of this project. Suitable comment statements are included for better readability and understanding of the thought process behind every code segment.

1) main_function.c :-

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include <time.h>
#include"server.h"

struct Empdetials emp[5];  // definition of emp
int linematched;

int main(int argc,char *argv[])//command line arguments takes the name of t
he file to be read
{
    char file_name[30];//to copy the name of the file into a character arra
y
    if(argc != 2)   //Argument Validation
    {
        printf("PLEASE ENTER DATA IN CORRECT FORMAT \n");
        exit(0);
    }
    //if(argv[1]=='\0')
    if (strlen(argv[1])>30)    //To bound the file name size to avoid overf
lows
    {
        printf("Length of file name exceeds the limit\n");
        exit(0);
    }
    strcpy(file_name,argv[1]);//copying the name here
    //printf("%s\n\n",file_name);//testing if the name is taken sucessfully

    table(file_name);//calling the function to initiate the process of fill
ing a structure with all the data from the files
    printf("--------------------------------HELLO-------------------------
-----\n");
```

```
    printf("----------------------WELCOME TO LEAVE MANAGEMENT SYSTEM-------
-------\n");
    main2();//the use of this is explained in the next .c file
    return 0;
 }
```

2) main2_table.c :-

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <time.h>
#include<ctype.h>
#include"server.h"

/*main function has a call to table() and invoking main() for re-
login used to reset the structure and the deductions were lost,so we copied

everything from the main function into another function called main2() whic
h is basically the main function without the table() invoking so the
structure is not resetted after every call from again() and quit() during R
e-login and structure keeps a track of the deductions that happened and
in this way we also removed the Welcome sentence to the same user,but the m
ain reason is to avoid rewriting of the structure on every call
and losing the deduction data*/


void main2()
{
    int casualm=10;//defining the maximum possible leaves here
    int medicalm=15;
    int earnedm=7;
    int leave_days=0;//handles the number of leaves that a user will enter
    int choice=0;//type of leave is handled by this variable 1 is casual,2
is emergency,3 is earned
    int log=0;//CHECKS IF LOGIN WAS SUCCESFULL OR NOT
    int res=0;//CHECKS IF DEDUCTION WAS SUCCESFULL OR NOT
    //-------------------------------------------------------------------
-------------------------------------------------------------------
-
    //-------------------------------------------------------------------
-------------------------------------------------------------------
--
    log=login();//login is invoked and log stores the value which determine
s whether the login was succesful or not
    if(log==1)//if log=1,implies that the login was succesful and the user
was found
    {
        printf("PLEASE SELECT THE TYPE OF LEAVE YOU WANT TO TAKE\n");
        printf("1.CASUAL LEAVE\n");
        printf("2.MEDICAL LEAVE\n");
```

```
        printf("3.EARNED LEAVE\n");
        (void)scanf("%d",&choice);
        printf("PLEASE ENTER THE NUMBER OF LEAVES YOU WANT TO APPLY FOR\n")
;
        (void)scanf("%d",&leave_days);
    }
    else
    {
        printf("SORRY THE LOGIN CREDENTIALS DON'T MATCH WITH ANY USER\n\n\n
");//when the login fails this message is displayed
        again();//gives user another chance to take give the input
    }
    //first validation of the input values,this compares the input with the
 largest value of holidays available
    int flag=1;//this controls whether the variable entered is within the m
aximum limit and that deduction() function can be called

     if(choice==1)
    {
        if(leave_days>casualm)
        {
            printf("THE ENTERED NUMBER OF DAYS CROSSES THE MAXIMUM LIMIT \n
");
            again();
            flag=0;
        }
    }

    if(choice==2)
    {
        if(leave_days>medicalm)
        {
            printf("THE ENTERED NUMBER OF DAYS CROSSES THE MAXIMUM LIMIT \n
");
            again();
            flag=0;
        }
    }

    if(choice==3)
    {
        if(leave_days>earnedm)
        {
            printf("THE ENTERED NUMBER OF DAYS CROSSES THE MAXIMUM LIMIT \n
");
            again();
            flag=0;
        }
    }
    if(choice>3||choice<0)
    {
```

```
        printf("sorry!! wrong input\n");
        again();
        flag=0;
    }

    if(flag==1)//if the validation is succesful the value of the variable
remains 1 and this loop is entered
    {
        res=0;
        res=deduction(linematched,leave_days,choice);//if res==1 then thern
 the validation was succesful and then the deduction is called and paramete
r is passed
        if(res==1)//if deduction was succesful then the
        {
            printf("DEDUCTION  WAS SUCCESFUL\n\n");
            again();
        }
        // else{
        //     printf("failed\n");
        //     again();
        // }
    }
}


//Here, we convert the file to a structure and then use the structure to ac
cess the data and also using files to store the information
extern void table(char file_n[])//this function is responsible for populati
ng the structure by reading the file
{
   FILE *fp=NULL;//defining a file pointer
   //char ch;
   char  line[256];//definig 256 here,any arbitrary number can be used
   char *token;//token pointer that points to the elements of a line
   int tokenposition=0;//token here means the particular text separated by
comma
   int lineposition=0;//line here is the entire line which has entire detai
ls of the an emmployee
   fp=fopen(file_n,"r");//opening the file in read mode as we are only acce
ssing information and not rewriting it
   //-------------------------------------------------------------------
-------------------------------------------------------------------
if (fp==NULL)
{
    printf("File access error!!\n");
    exit(0);
}
    while(fgets(line,256,fp) !=NULL) //accessing the lines one by one and t
rying to recognize the end of a line
    {
```

```
        tokenposition=0;//when we tokenize the line we need the count to ke
ep a track of where we are int he string
        //printf("%s \n",line);//used to check which line is not being read
 by the compiler
        token = strtok(line,",");
            //this seperates a line into token(entity) based on a delimiter
,here the delimiter is a comma
        //strtok-tring tokenization method
        while(token != NULL)
        {
            switch(tokenposition)//acessing the required element through th
e tokenposition counter
            {
                case 0://NAME
                    strcpy(emp[lineposition].name,token);
                    break;
                case 1://ID
                    strcpy(emp[lineposition].id,token);
                    break;
                case 2://PASSWORD
                    strcpy(emp[lineposition].password,token);
                    break;
                case 3://CASUAL LEAVE
                    emp[lineposition].casual=atoi(token);
                    break;
                case 4://MEDICAL LEAVE
                    emp[lineposition].medical=atoi(token);
                    break;
                case 5://EARNED LEAVE
                    emp[lineposition].earned=atoi(token);
                    break;
            }
            token =strtok(NULL,",");
        tokenposition++;//to access the next token in the same line
        }
        lineposition++;//after all the tokens are put in a structure,we mov
e to the next line and redo the entire process of reading a line
    }
        int success=fclose(fp); //Don't forget to close the file when finis
hed
    if(success==0)
    {
        printf("file closed successfully\n");
    }
    else{
        printf("ERROR WHILE CLOSING THE FILE\n");
    }
}
```

3) login_deduction.c :-

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <time.h>
#include<ctype.h>
#include"server.h"

int login()
{
    int i;//for the loop
    int c;
    char userid[9];//user ID in the main function,this will be passed to th
e the login function
    char userpassword[5];//user password in the main function

    printf("ENTER YOUR 8 CHARACTER USER ID\n");
    (void)scanf("%8s",userid);
    while ((c = fgetc(stdin)) != (int)'\n' && c != EOF); /* Flush stdin */
    printf("ENTER THE 4 CHARACTER PASSWORD\n");
    (void)scanf("%4s",userpassword);
    while ((c = fgetc(stdin)) != (int)'\n' && c != EOF); /* Flush stdin */
    linematched=0;

    for(i=0;i<5;i++)
    {
        if((strcmp(emp[i].id,userid)==0)&&(strcmp(emp[i].password,userpassw
ord)==0))//checking if login credentils are correct or wrong
        {
            linematched=i;
            printf("WELCOME %s TO THE LEAVE MANEGMENT SYSTEM\n",emp[linemat
ched].name);
            return 1;//TO INDICATE THAT USER WAS FOUND AND LOGIN WAS SUCCES
FUL
            //break;
        }

    }
    return 0;
}


int deduction(int linematched,int days,int type_of_leave)//here all inputs
are validated for the second time after the validation in main function
{
```

```
    int xyz=0;

    switch(type_of_leave)
    {
        case 1://casual leave
        {
            if(days>emp[linematched].casual || days<=0)//second validation
happens here with respect to the specified user
            {
                printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE E
NTERED A NEGATIVE VALUE\n\n");
                again();
                xyz=0;
                //return 0;
                //break;
            }
            else
            {
                emp[linematched].casual=emp[linematched].casual-days;
                xyz=1;
                //return 1;
                //break;
            }
            break;
        }
        case 2://medical leave
        {
            if(days>emp[linematched].medical || days<=0)
            {
                printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE E
NTERED A NEGATIVE VALUE\n\n");
                again();
                xyz=0;
                //return 0;
                //break;
            }
            else
            {
                emp[linematched].medical=emp[linematched].medical-days;
                xyz=1;
                //return 1;
                //break;
            }
            break;
        }
        case 3://earned leave
        {
            if(days>emp[linematched].earned || days<=0)
            {
                printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE E
NTERED A NEGATIVE VALUE\n\n");
```

```
                again();
                xyz=0;
                //return 0;
                //break;
            }
            else
            {
                emp[linematched].earned=emp[linematched].earned-days;
                xyz=1;
                //return 1;
                //break;
            }
            break;
        };
        // default:
        // {
        //     printf("SORRY! YOU HAVE NOT ENTRERED A VALID TYPE OF LEAVE \
n\n\n");
        //     again();
        //     //xyz=0;
        //     break;
        // }
    };
    return xyz;
}
```

4) quit_again_final.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include <time.h>
#include"server.h"

void quit()
{
    char ans[5];
    int c;
    printf("ARE YOU SURE YOU WANT TO QUIT THE WIZARD? Y OR N\n");
    (void)scanf("%4s", ans);
     while ((c = fgetc(stdin)) != (int)'\n' && c != EOF); /* Flush stdin */
    if (tolower(ans[0]) == (int)'y')
    {
        final_print();
        printf("\n\n");
        printf("---THANK YOU FOR USING OUR PORTAL--\n");
    }
    else
    {
```

```
        main2();
    }
}


void again()
{
    char ans1[5];
    int c;
    //char tt='y';
    printf("SEARCH AGAIN USING ID AND PASSWORD? Y OR N\n");
    (void)scanf("%4s", ans1);
    while ((c = fgetc(stdin)) != (int)'\n' && c != EOF); /* Flush stdin */
    if (tolower(ans1[0]) == (int)'y')
    {
        main2();//FOR RE-LOGIN PROCESS
    }
    else
    {
        quit();
    }
    printf("\n\n\n");
}

//PRINTS TO FILE AND THE TERMINAL THE FINAL RESULTS AND STATEMENT OF LEAVES
 OF EVERY EMPLOYEE
void final_print()
{
    FILE *fp=NULL;
    //------------------------------------------------------------------------
----------------------
    char text[150];
    time_t now = time(NULL);
    struct tm *t = localtime(&now);
    //strftime(text, sizeof(text)-1, "%dd %mm %YYYY %HH:%MM", t);
    (void)strftime(text, sizeof(text)-1,"%d_%B_%Y_%H-%M-%S", t);
    text[150] = '\0';
    // concat the date to file name
    char *filename;
    //C:\\Users\\Satyam_Dwivedi\\Desktop\\Employee_leave_manegment_system-
master
    //C:\\Users\\Sai_Gruheeth\\Desktop\\Employee_leave_manegment_system-
master
    if((filename = malloc(strlen("C:\\Users\\BHARGAV_NARAYANAN\\Employee_le
ave_manegment_system-master")+strlen(text)+1)) != NULL){
        filename[0] = '\0';   // ensures the memory is an empty string
        strcat(filename,"statement_");
        strcat(filename,text);
        strcat(filename,".txt");
    }
    //printf("filename: %s \n\n",filename);
```

```
   // use the file
    //-------------------------------------------------------------------------
---------------------
    if(filename!=NULL)
    {
        printf("filename: %s \n\n",filename);
        fp = fopen(filename, "w");// "w" means that we are going to write o
n this file
    }
    if (fp==NULL)
    {
        printf("File access error!!\n");
        exit(0);
    }
    printf("----------------FINAL STATEMENT OF LEAVE OF EACH EMPLOYEE------
------------------\n\n");
    fprintf(fp,"----------------FINAL STATEMENT OF LEAVE OF EACH EMPLOYEE--
----------------------\n\n");
    int i;
    for (i = 0; i < 5; i++)
    {
        printf("NAME : %s\n",emp[i].name);
        fprintf(fp,"NAME : %s\n",emp[i].name);
        printf("USER ID : %s\n",emp[i].id);
        fprintf(fp,"USER ID : %s\n",emp[i].id);
        printf("CASUAL LEAVE LEFT : %d\n",emp[i].casual);
        fprintf(fp,"CASUAL LEAVE LEFT : %d\n",emp[i].casual);
        printf("MEDICAL LEAVE LEFT : %d\n",emp[i].medical);
        fprintf(fp,"MEDICAL LEAVE LEFT : %d\n",emp[i].medical);
        printf("EARNED LEAVE LEFT : %d\n\n",emp[i].earned);
        fprintf(fp,"EARNED LEAVE LEFT : %d\n\n",emp[i].earned);
    }
    printf("---------THANK YOU FOR USING LEAVE MANAGEMENT PORTAL-----\n");
    fprintf(fp,"---------THANK YOU FOR USING LEAVE MANAGEMENT PORTAL-----
\n");
    // printf("STATMENT GENERATED ON %s \n \n", time_string);
    // fprintf("STATEMENT GENERATED ON %s\n \n", time_string);

    int success=fclose(fp); //Don't forget to close the file when finished
    if(success==0)
    {
        printf("file closed successfully\n");
    }
    else{
        printf("ERROR WHILE CLOSING THE FILE\n");
    }
    free(filename);
}
```

.h FILE:

server.h

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

extern int linematched;//global variable which stores the variable
extern struct Empdetials
{
    char name[25];
    char id[9];//one extra character to store /n
    char password[5];
    int casual;//casual,medical,earned leave
    int medical;
    int earned;
}emp[5];//declaring array of structures which will later be populated using
 table() function
//extern struct Empdetials emp[5]; // declaration of emp

extern void main2();//this containsthe main part of the code workings,its p
upose is explained below just before the function

extern void final_print();//prints the details of the user on quitting the
wizard
extern void quit();//this is one function which handles the termination of
code prompting the user to take another trail

extern void again();//this function handles the login if user wishes to tak
e another leave

extern void table(char file_n[]);//populating the structure from data eleme
nts from the file by using specific delimiter
extern int login();//this function verifies the password with the one prese
nt in the file database
extern int deduction(int linematched,int days,int type_of_leave);//this per
forms another validation and deducts the days user desires for taking the l
eave
```

Makefile:

```
a.out: main_function.o quit_again_final.o login_deduction.o main2_table.o
    gcc main_function.o quit_again_final.o login_deduction.o main2_table.o
quit_again_final.o: quit_again_final.c server.h
    gcc -c quit_again_final.c
login_deduction.o: login_deduction.c server.h
    gcc -c login_deduction.c
main2_table.o: main2_table.c server.h
    gcc -c main2_table.c
main_function.o: main_function.c server.h
    gcc -c main_function.c
```

Sample template of the Input File:

Employeeinfo.txt

Ramesh,QW120345,PO56,10,15,7

Rajesh,QW120905,IO56,10,15,7

Kajal,JI456987,IWQ9,10,15,7

Harleen,HJ782013,ZM12,10,15,7

Jim,BN784569,KL45,10,15,7

Final Statement (Output file):

file name template: *statement_dd_mm_yyyy_HH-MM-SS.txt*

----------FINAL STATEMENT OF LEAVE OF EACH EMPLOYEE----------

NAME : Ramesh

USER ID : QW120345

CASUAL LEAVE LEFT : 10

MEDICAL LEAVE LEFT : 15

EARNED LEAVE LEFT : 7

NAME : Rajesh
USER ID : QW120905
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

NAME : Kajal
USER ID : JI456987
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

NAME : Harleen
USER ID : HJ782013
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

NAME : Jim
USER ID : BN784569
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 5

---------THANK YOU FOR USING LEAVE MANAGEMENT PORTAL-----

Testing and Analysis

Vulnerable test cases

1.Input Validation :

```
^[[Agruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$ make -f makefile.mk
make: 'a.out' is up to date.
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$ ./a.out employeeinfo.txt
--------------------------------HELLO-------------------------------
----------------------WELCOME TO LEAVE MANEGMENT SYSTEM--------------
ENTER YOUR 8 CHARACTER USER ID
dsfjkhdsjhfldhjksflsdjhf
ENTER THE 4 CHARACTER PASSWORD
djfkj
*** stack smashing detected ***: terminated
Aborted (core dumped)
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$
```

A **stack smashing** error was observed every time we put up a large user id, as the string could not accept more than 9 characters for username and 5 characters for password.

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$ ./a.out emp.txt
Segmentation fault (core dumped)
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$
```

A **segmentation fault** occurred every time a wrong text file was given as an input, making the code end abruptly.

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$ ./a.out
Segmentation fault (core dumped)
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$
```

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$ ./a.out employee.txt emp.txt
Segmentation fault (core dumped)
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$
```

When no file was given as an input or when there were 2 or more text files given as an input, the code ended abruptly.

2.Negative inputs :

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/old$ ./a.out employeeinfo.txt
--------------------------------HELLO--------------------------------
---------------------WELCOME TO LEAVE MANEGMENT SYSTEM-------------
ENTER YOUR 8 CHARACTER USER ID
QW120345
ENTER THE 4 CHARACTER PASSWORD
PO56
WELCOME Ramesh TO THE LEAVE MANEGMENT SYSTEM
PLEASE SELECT HE TYPE OF LEAVE YOU WANT TO TAKE
1.CASUAL LEAVE
2.MEDICAL LEAVE
3.EARNED LEAVE
1
PLEASE ENTER THE NUMBER OF LEAVES YOU WANT TO APPLY FOR
-5
DEDUCTION  WAS SUCCESFUL

SEARCH AGAIN USING ID AND PASSWORD? Y OR N
N
ARE YOU SURE YOU WANT TO QUIT THE WIZARD? Y OR N
Y
---------------FINAL STATEMENT OF LEAVE OF EACH EMPLOYEE-----------------------

NAME : Ramesh
USER ID : QW120345
CASUAL LEAVE LEFT : 15
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

NAME : Rajesh
USER ID : QW120905
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

NAME : Kajal
USER ID : JI456987
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

NAME : Harleen
USER ID : HJ782013
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

NAME : Jim
USER ID : BN784569
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

---------THANK YOU FOR USING LEAVE MANEGMENT PORTAL-----
```

On adding negative values, the leave was getting added and this was one of the major concerns which had to be dealt with.

3.Incompatible types comparison:

```
char ans[5];
printf("ARE YOU SURE YOU WANT TO QUIT THE WIZARD? Y OR N\n");
scanf("%s", ans);
if (tolower(ans[0]) == 'y')
{
    final_print();
    printf("\n\n");
    printf("---THANK YOU FOR USING OUR PORTAL--\n");
}
```

In the first few functions, we have used equality to define/check for the equality of character. What this actually does is return a flag variable. This was indicated by a splint warning in most of our functions.

4.File pointer's argument passed with a non-null parameter:

```
while(fgets(line,256,fp) !=NULL) //accessing the lines one by one and trying to recognize the end of a line
{
    tokenposition=0;//when we tokenize the line we need the count to keep a track of where we are int he string
  //printf("%s \n",line);//used to check which line is not being read by the compiler
```

Again, this was referred from splint's warnings. Each line was being retrieved with no check for a Null character. It just carries on even if a Null pointer is found.

5.Ignoring the use of command line arguments:

```
void main(int argc,char *argv[])//command line arguments takes the name of the file to be read
{
    char file_name[30];//to copy the name of the file into a character array
    strcpy(file_name,argv[1]);//copying the name here
    //printf("%s\n\n",file_name);//tersting if the name is taken sucessfully
    table(file_name);//calling the function to initiate the process of filling a structure with all the data from the files
    printf("--------------------------------HELLO--------------------------------\n");
    printf("----------------------WELCOME TO LEAVE MANEGMENT SYSTEM-------------\n");
    main2();//the use of this is explained in the next .c file
}
```

6. Use of too many break statements for case blocks in switch statements:

```
switch(type_of_leave)
{
    case 1://casual leave
    {
        if(days>emp[linematched].casual)//second validation happens here
        {
            printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA\n\n");
            again();
            return 0;
            break;
        }
        else
        {
            emp[linematched].casual=emp[linematched].casual-days;
            return 1;
            break;
        }
        break;
    }
    case 2://medical leave
    {
        if(days>emp[linematched].medical)
        {
            printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA\n\n");
            again();
            return 0;
            break;
        }
        else
        {
            emp[linematched].medical=emp[linematched].medical-days;
            return 1;
            break;
        }
        break;
    }
    case 3://earned leave
    {
        if(days>emp[linematched].earned)
        {
            printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA\n\n");
            again();
            return 0;
            break;
        }
        else
        {
            emp[linematched].earned=emp[linematched].earned-days;
            return 1;
            break;
        }
        break;
    };
};
```

Non-Vulnerable test cases

1.Input Validation:

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ make -f makefile.mk
make: 'a.out' is up to date.
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ ./a.out employeeinfo.txt
file closed successfully
--------------------------------HELLO--------------------------------
---------------------WELCOME TO LEAVE MANAGEMENT SYSTEM--------------
ENTER YOUR 8 CHARACTER USER ID
sdfhsdjfhsdfhljhfds
ENTER THE 4 CHARACTER PASSWORD
kjfkjdfkjdf
SORRY THE LOGIN CREDENTIALS DON'T MATCH WITH ANY USER
```

The issue of stack smashing is solved. The number of characters in a string that had to be given as an was set and mentioned.

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ ./a.out emp.txt
File access error!!
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$
```

For a wrong file given as an input, it gives back a file access error when that particular file is not included in the folder of the files stored.

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ ./a.out
PLEASE ENTER DATA IN CORRECT FORMAT
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ ./a.out emp.txt employee.txt
PLEASE ENTER DATA IN CORRECT FORMAT
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$
```

Regarding no input text file or 2/more text files given as an input, a message is been provided to the user, which has solved the issue of the code ending abruptly.

2.Negative inputs :

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ ./a.out employeeinfo.txt
file closed successfully
-------------------------------HELLO-------------------------------
---------------------WELCOME TO LEAVE MANAGEMENT SYSTEM--------------
ENTER YOUR 8 CHARACTER USER ID
QW120345
ENTER THE 4 CHARACTER PASSWORD
PO56
WELCOME Ramesh TO THE LEAVE MANEGMENT SYSTEM
PLEASE SELECT THE TYPE OF LEAVE YOU WANT TO TAKE
1.CASUAL LEAVE
2.MEDICAL LEAVE
3.EARNED LEAVE
1
PLEASE ENTER THE NUMBER OF LEAVES YOU WANT TO APPLY FOR
-5
SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE ENTERED A NEGATIVE VALUE

SEARCH AGAIN USING ID AND PASSWORD? Y OR N
N
ARE YOU SURE YOU WANT TO QUIT THE WIZARD? Y OR N
Y
filename: statement_04_May_2021_21-33-58.txt

----------------FINAL STATEMENT OF LEAVE OF EACH EMPLOYEE-----------------------

NAME : Ramesh
USER ID : QW120345
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7

NAME : Rajesh
USER ID : QW120905
CASUAL LEAVE LEFT : 10
MEDICAL LEAVE LEFT : 15
EARNED LEAVE LEFT : 7
```

Negative value which was getting added to the leave quota has been taken care and provides a message to the user prompting him to enter an appropriate value.

3.Incompatible types comparison:

```
while ((c = fgetc(stdin)) != (int)'\n' && c != EOF);
if (tolower(ans[0]) == (int)'y')
{
    final_print();
    printf("\n\n");
    printf("---THANK YOU FOR USING OUR PORTAL--\n");
}
```

To avoid the splint warning, we have explicitly converted the value of 'y' to its ascii value and have proceeded with the comparison.

4.File pointer's argument passed with a non-null parameter:

```
extern void table(char file_n[])//this
{
    FILE *fp=NULL;//defining a file poi
    //char ch;
    char  line[256];//definig 256 here,
    char *token;//token pointer that po
    int tokenposition=0;//token here me
    int lineposition=0;//line here is t
    fp=fopen(file_n,"r");//opening the
    //--------------------------------

if (fp==NULL)
{
    printf("File access error!!\n");
    exit(0);
}

    while(fgets(line,256,fp) !=NULL) //
    {
        tokenposition=0;//when we toke
        //printf("%s \n",line);//used
        token = strtok(line,",");
```

This checks for a Null pointer and treat it as shown and now has nothing to worry even if a Null pointer shows up.

5.Ignoring the use of command line arguments:

```c
int main(int argc,char *argv[])//command line arguments takes the name of the file to be re
{
    char file_name[30];//to copy the name of the file into a character array
    if(argc != 2)   //Argument Validation
    {
        printf("PLEASE ENTER DATA IN CORRECT FORMAT \n");
        exit(0);
    }
    //if(argv[1]=='\0')
    if (strlen(argv[1])>30)    //To bound the file name size to avoid overflows
    {
        printf("Length of file name exceeds the limit\n");
        exit(0);
    }
    strcpy(file_name,argv[1]);//copying the name here
    //printf("%s\n\n",file_name);//testing if the name is taken sucessfully
    table(file_name);//calling the function to initiate the process of filling a structure
    printf("--------------------------------HELLO--------------------------------\n");
    printf("----------------------WELCOME TO LEAVE MANAGEMENT SYSTEM--------------\n");
    main2();//the use of this is explained in the next .c file
    return 0;
}
```

Use of command line arguments for checking the data format's accuracy. This particular variable is used for checking whether the appropriate file name is given as an input.

If there are any discrepancies in the input, it is directly dealt here and not allowed to go any further.

6.Use of too many break statements for case blocks in switch statements:

```
case 1://casual leave
{
    if(days>emp[linematched].casual || days<=0)//second validation happens here with respect to the
    {
        printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE ENTERED A NEGATIVE VALUE\n\n");
        again();
        xyz=0;
        //return 0;
        //break;
    }
    else
    {
        emp[linematched].casual=emp[linematched].casual-days;
        xyz=1;
        //return 1;
        //break;
    }
    break;
}
case 2://medical leave
{
    if(days>emp[linematched].medical || days<=0)
    {
        printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE ENTERED A NEGATIVE VALUE\n\n");
        again();
        xyz=0;
        //return 0;
        //break;
    }
    else
    {
        emp[linematched].medical=emp[linematched].medical-days;
        xyz=1;
        //return 1;
        //break;
    }
    break;
}
case 3://earned leave
{
    if(days>emp[linematched].earned || days<=0)
    {
        printf("SORRY YOU HAVE EXHAUSTED YOUR LEAVE QUOTA OR HAVE ENTERED A NEGATIVE VALUE\n\n");
        again();
        xyz=0;
        //return 0;
        //break;
    }
    else
    {
        emp[linematched].earned=emp[linematched].earned-days;
        xyz=1;
        //return 1;
        //break;
    }
    break;
};
```

Use of flag variables returning to the function call instead of break operations for further procession.

Static Analysis report

Splint warnings

Before :

```
gruheeth@gruheeth-VirtualBox:~/Desktop/SWPC$ splint quit_again_final.c
Splint 3.1.2 --- 20 Feb 2018

quit_again_final.c: (in function quit)
quit_again_final.c:10:5: Return value (type int) ignored: scanf("%s", ans)
  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)
quit_again_final.c:11:9: Incompatible types for == (char, char):
                           tolower(ans[0]) == 'y'
  A character constant is used as an int. Use +charintliteral to allow
  character constants to be used as ints.  (This is safe since the actual type
  of a char constant is int.)
quit_again_final.c: (in function again)
quit_again_final.c:26:5: Return value (type int) ignored: scanf("%s", ans1)
quit_again_final.c:27:9: Incompatible types for == (char, char):
                           tolower(ans1[0]) == 'y'

Finished checking --- 4 code warnings
```

i) Regarding scanf warning, there was no dedicated number of input characters due to which the stack smashing error happened to pop up. To resolve this we have mentioned the number of characters to be taken as an input.

ii) To resolve the incompatible type warning, we explicitly converted the value of the char to its actual ascii number so that comparison can become easier.

```
gruheeth@gruheeth-VirtualBox:~/Desktop/SWPC$ splint main_function.c
Splint 3.1.2 --- 20 Feb 2018

main_function.c:8:6: Function main declared to return void, should return int
  The function main does not match the expected type. (Use -maintype to inhibit
  warning)
main_function.c: (in function main)
main_function.c:8:15: Parameter argc not used
  A function parameter is not used in the body of the function. If the argument
  is needed for type compatibility or future plans, use /*@unused@*/ in the
  argument declaration. (Use -paramuse to inhibit warning)

Finished checking --- 2 code warnings
```

iii) Changes the void main() to int main() with a return 0.

iv) Command line argument usage was implemented (argc).

```
gruheeth@gruheeth-VirtualBox:~/Desktop/SWPC$ splint main2_table.c
Splint 3.1.2 --- 20 Feb 2018

main2_table.c: (in function main2)
main2_table.c:29:9: Return value (type int) ignored: scanf("%d", &choice)
  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)
main2_table.c:31:9: Return value (type int) ignored: scanf("%d", &lea...
main2_table.c:76:8: Test expression for if not boolean, type int: flag
  Test expression type is not boolean or int. (Use -predboolint to inhibit
  warning)
main2_table.c: (in function table)
main2_table.c:97:25: Possibly null storage fp passed as non-null param:
                       fgets (..., fp)
  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)
   main2_table.c:95:7: Storage fp may become null
main2_table.c:90:9: Variable ch declared but not used
  A variable is declared but never used. Use /*@unused@*/ in front of
  declaration to suppress message. (Use -varuse to inhibit warning)

Finished checking --- 5 code warnings
```

v) The same scanf warning has been shown again as the number of characters to be taken for input was not mentioned.

vi) Null pointer test case for file pointer was not taken as a check. It was implemented using a if condition to check if the pointer is Null and pop up a message regarding file access error.

vii) Regarding flag variable, its initiation was done in local scope. Its was shifted on a global scope with respect to that particular functioning. Later, this whole flag variable was replaced with two new variables which takes care of checks.

```
gruheeth@gruheeth-VirtualBox:~/Desktop/SWPC$ splint login_deduction.c
Splint 3.1.2 --- 20 Feb 2018

login_deduction.c: (in function login)
login_deduction.c:12:5: Return value (type int) ignored: scanf("%s", userid)
  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)
login_deduction.c:14:5: Return value (type int) ignored: scanf("%s", user...
login_deduction.c:23:13: Unreachable code: break
  This code will never be reached on any possible execution. (Use -unreachable
  to inhibit warning)
login_deduction.c: (in function deduction)
login_deduction.c:39:17: Unreachable code: break
login_deduction.c:45:17: Unreachable code: break
login_deduction.c:56:17: Unreachable code: break
login_deduction.c:62:17: Unreachable code: break
login_deduction.c:73:17: Unreachable code: break
login_deduction.c:79:17: Unreachable code: break
login_deduction.c:84:2: Path with no return in function declared to return int
  There is a path through a function declared to return a value on which there
  is no return statement. This means the execution may fall through without
  returning a meaningful result to the caller. (Use -noret to inhibit warning)

Finished checking --- 10 code warnings
```

viii) Scanf() warning was resolved by determining the number of characters to be taken as an input.

ix) Multiple break statements were used to get out of the loop. Instead of using break statements which used to end the program abruptly, we have made use of flag variables which get returned to the function call to take further necessary actions.

After :

```
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ splint login_deduction.c
Splint 3.1.2 --- 20 Feb 2018

Finished checking --- no warnings
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ splint main_function.c
Splint 3.1.2 --- 20 Feb 2018

Finished checking --- no warnings
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ splint main2_table.c
Splint 3.1.2 --- 20 Feb 2018

Finished checking --- no warnings
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$ splint quit_again_final.c
Splint 3.1.2 --- 20 Feb 2018

Finished checking --- no warnings
gruheeth@DESKTOP-9NV73NC:/mnt/c/Users/gruhe/Desktop/swpc$
```

MAINTAINANCE

Readability:

The program is written is a very simple way. All important lines of code have been commented on what that particular line does. For any person who views the program can easily understand it without any difficulty.

Modifications:

Modularity:

Any function can be designed/modifies according to the need of the user. The program is designed very flexibly that any action that has been reverted/changed can be reversed and any kind of variables can be modified according to the need of the user.

Complexity:

File pointer handling was one of the biggest tackles we faced. Handling the Null pointer and the regular data, fetching it and updating it according the user's actions was one big phase of the program that took time.

We have taken some ideas from banks where transaction data is maintained with date and time and the changes made. Similar idea has been implemented where in when any user accesses the application and takes a leave, a separate text file is created to keep the record of what changes are made and when they are made.

Future Enhancements:

A web application/mobile application can be created to access this from any remote place by the employee.

Create Multiple Version

Multiple versions can be created with any further updates that can be made with some implementable ideas.

The same idea can be implemented for goods processing in provision stores/stationary stores or for booking an appointment with doctors at clinics/hospitals.

Scalability

Leave management system is implemented for now. This can be later progressed onto salary management according to the leaves taken by the employee.

Allowances of any employee's leaves can also be integrated. Conditional excuses for employee's absence can also be monitored. This also can be extended to monitor the work progress done by the employee and all of these can be done changing a few variable values and altering the table() function with more switch cases to accommodate read more fields from the file, all the required counter and limit variables are placed in the function so the work of changing them is easier

CONCLUSION

Through this project we learnt a lot about the characteristics of the C programming language and how raw it is, it is an extremely powerful language that depends on the programmer's ability to resolve issues on his own ability and how data sharing can take place in multiple file systems and this not only introduced us to a lot of positive sides of the language that it is ultra-fast, compact in final form and but its highly vulnerable due to coder's negligence in maintaining the code security standards , so in a nutshell C language trades off security with speed and efficiency of execution and this turns out to be a very crucial factor that if a programmer is cautious and writes a good code we can have best of both worlds- speed and security, but the norms needed to monitor and govern them need to be universal and this is provided by CERT C STANDARDS, it not only helped us improve the code's standards and security, it also made the data flow more streamlined and organized.

Through the implementation of global data handling and passing across multiple implementation files we learnt how powerful static and extern keywords are and how the utilization of these help in safe handling of data, in order to solve the issue we had to use stackoverflow to understand these aspects.

We also came across the idea to name a file according to the system date and time of creation of the file and this too gave us immense challenge to find an appropriate library function to do the task of generating and concatenating the date and time into a legally correct format of specifying the name of the file and this painstaking

work of generation and writing to the file and closing it later endowed us with some valuable insights of using system and handling the know how of a few things and also we were made aware of the devastating consequences of buffer overflow leading to stack smashing and a vulnerable loophole in the code that can lead to breakdown and reveal sensitive data and thus compromise on security or worsen is overwrite the existing data leading to catastrophic effects on a working system which relies on these data elements.

The non-object-oriented nature of the languages gave us quite a lot of trouble to handle data, but the provision of subroutines and pointers made the work a lot easier.

The major takeaway of this project implementation phase was learning how a team works together on a code and how we integrate all the different parts and synchronize function calls and pass values without leaving a leak in the code and also learnt a lot about the the handling of global data and utilizing system time to name files in custom legal formats.

BIBLIOGRAPHY

- CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems.- SEI CERT,2016
- https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard
- https://stackoverflow.com/questions/61951837/i-am-trying-to-split-this-code-into-two-c-and-one-h-file-but-the-structure-is
- https://stackoverflow.com/questions/61726808/how-to-populate-the-structure-using-a-filecode-attached-below
- https://www.tutorialspoint.com
- https://www.geeksforgeeks.org
- Bert C. Seacord, "Secure coding in C and C++" , Second edition Addison Wesley Professional,2013

THANK YOU