```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_LENGTH 80
#define MAX_QA 30
#define MAX_LINE_LENGTH 80
#define MAX_ID_LENGTH 10
#define MAX_USERS 100
#define MAX_ATTEMPTS 3
#define MAX_FILENAME 30

struct User
{
      char userID[10];
      char password [10];
      char name [20];
      char address [30];
      char contact [12];
      char sex[10];
      int vacc_year;
      int vacc_month;
      int vacc_day;
      char vacc_name [10];
      char vacc_loc [10];
      int sec_year;
      int sec_month;
      int sec_day;
      char sec_name [10];
      char sec_loc [10];
      int booster_year;
      int booster_month;
      int booster_day;
      char booster_name [10];
      char booster_loc [10];
};
      struct User date; // User field 1
      struct User users; // User field 2
      struct User exp_users;
      struct User u[MAX_USERS]; // array of User structures
      int num_users = 0; // used to keep track of the set number of users/
application

struct Appointment
{
      char appID [10];
      char name [20];
      char loc [10];
      char vacc_name [10];
      int hour;
      int min;
      int year;
      int month;
      int day;
      char dose [10];
};

      struct Appointment applic; // Appointment field 1
      struct Appointment Date; // Appointment field 2
      struct Appointment time; // Appointment field 3
      struct Appointment a[MAX_USERS]; // array of Appointment structures
      int numUsers = 0;

struct Chatbot
{
      char user_question[MAX_LENGTH];
    char file_question[MAX_LENGTH];
    char file_answer[MAX_LENGTH];
    char newQuestion[MAX_LENGTH];
      char newAnswer[MAX_LENGTH];
};
      struct Chatbot cb;
      struct Chatbot c[MAX_QA];
      struct Chatbot QA[MAX_LENGTH];
      int numQA = 6;


/
***********************************************************************************
**************************************/
// ************************************ V A C C I N A T I O N A P P O I N T M E N
T ************************************/
/
***********************************************************************************
**************************************/

// function declarations to avoid implicit
void AppointmentRequest();
void Registration_app();
int VaccinationRegistration (int vaccReg);
void ManageAppointmentMenu();
int Menu (int menu);

/* Vacc_App returns the choice of the user whether he/she opt to request or manage
his/her appointment
@param vaccapp - contains the choice
@return - none
Pre-condition: choice contains only numbers 1-3
*/
int
Vacc_App (int vaccapp){

      int vaccReg;
      int choice;
      char c;

      printf("\n\n                                              Enter: ");
      scanf("%d",&choice); // user's input the choice
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

      switch (choice){

      case 1:
            displayAppReq();
            printf("\n");
            Registration_app();
            DisplayVA();
            Vacc_App (vaccapp);
            break;

      case 2:
            displayManReq();
            printf("\n");
            ManageAppointmentMenu();
            DisplayVA();
            Vacc_App (vaccapp);
            break;

      case 3:
            VaccinationRegistration(vaccReg);
            break;

      default:
            printf("\n                                        *** INVALID CHOICE.
TRY AGAIN! ***\n");
            Vacc_App (vaccapp);
            break;
    }

}

/* Application_Date returns whether the date is invalid or valid
@param daysInMonth - contains number of days in a month
@param applic.year - contains the input of the user for year
@param applic.month - contains the input of the user for month
@param applic.year - contains the input of the user for day
@return 1 if the date is valid
@return 0 if the date is invalid
Pre-condition: numbers should not be negative
*/
// function to validate date
int
Application_date(){

      // Array of months and their number of days
      int daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

      //checks is months and day are within the valid ranges
      // Booster Shot
      int validYear = applic.year >= 0;
      int validMonth = applic.month >= 1 && applic.month <= 12;
      int validDay = applic.day >= 1 && applic.day <= daysInMonth [applic.month -
1];

      // checks if it's leap year
      // Booster
      if (applic.year % 4 == 0 && (applic.year % 100 != 0 || applic.year % 400 ==
0)){
            daysInMonth [1] = 29; // [1] = Feb
      }

      if (validYear && validMonth && validDay ){
            return 1;
      }
      else {
            return 0;
      }
}

/* isValidTime returns whether the time is invalid or valid
@param applic.hour - contains the input of the user for hour
@param applic.min - contains the input of the user for min
@return 1 if the time is valid
@return 0 if the time is invalid
Pre-condition: numbers should not be negative
*/
// function to validate time
int
isValidTime (){

      return (applic.hour >= 0 && applic.hour <= 23 && applic.min >= 0 &&
applic.min <= 59);
}

/* isAppID_Valid returns whether the application ID is invalid or valid
@param applic.appID - contains the input of the user for application ID
@return 1 if the application ID is valid
@return 0 if the application ID is invalid
Pre-condition: application number should be unique
*/
// function to check if the application ID is valid (i.e., not already taken)
int isAppID_Valid(char* appID) {
    for (int i = 0; i < numUsers; i++) {
        if (strcmp(appID, a[i].appID) == 0) {
            printf("\n                                       *** APPLICATION ID IS
ALREADY TAKEN. PLEASE ENTER A DIFFERENT ID. ***\n");
            return 0; // ID is invalid
        }
    }
    return 1; // ID is valid
}

// function declaration to avoid implicit
int isValidTime();
int isDateValid();


/* Registration_app gets the input of the user for appointment request
Pre-condition: application number should be unique
*/
// function for appointment scheduling
void Registration_app() {
    char userID[11];
    char password[11];
    int validDate = 0;
    int validTime = 0;
    char c;
      FILE *fp;


      fp = fopen("Appointment.txt", "a");

      if(fp == NULL)
      printf("\n                                      [ERROR: FILE NOT FOUND] ");
```

```
            else if(fp != NULL)
            {
                  while (1)
                  {
                  printf("\n                                       Application ID: 
");
                  fgets(applic.appID, 10, stdin);
                  //fprintf(fp, "%s", applic.appID);

                  if (isAppID_Valid(applic.appID))
                        {
                        printf("\n                                       Name: ");
                        fgets(applic.name, 20, stdin);
                        //fprintf(fp, "%s", applic.name);

                        printf("\n                                       Location: 
");
                        fgets(applic.loc, 10, stdin);
                        //fprintf(fp, "%s", applic.loc);

                        printf("\n                                       Vaccine: 
");
                        fgets(applic.vacc_name, 10, stdin);
                        //fprintf(fp, "%s", applic.vacc_name);

                        // checks if date is valid
                        while (!validDate)
                              {
                        printf("\n                                       Date 
(YYYY-MM-DD): ");
                        scanf("%d %d %d", &applic.year, &applic.month, 
&applic.day);
                        //fprintf(fp,"%d-%d-%d ", applic.year, applic.month, 
applic.day);

                        validDate = Application_date(applic.year, applic.month, 
applic.day);
                        if (!validDate)
                                    {
                              printf("\n                                       *** 
INVALID DATE. PLEASE TRY AGAIN! ***\n");
                        }
                        }

                        // checks if time is valid (24-hour format)
                         while (!validTime)
                              {
                        printf("\n                                       Time 
(HH:MM, 24-hour format): ");
                        scanf("%d %d", &applic.hour, &applic.min);
                        //fprintf(fp, "%d:%d ", applic.hour, applic.min);
                        while((c = getchar()) != '\n' && c != EOF)
                                    {
                                    }
                        validTime = isValidTime(applic.hour, applic.min);
                        if (!validTime)
                                    {
                              printf("\n*** INVALID TIME. PLEASE TRY AGAIN! ***\
n");
```

```
                        }
                        }

                        printf("\n                                        Dose: ");
                        fgets(applic.dose, 10, stdin);
                        //fprintf(fp, "%s", applic.dose);
                        printf("\n                                        ***
APPOINTMENT CONFIRMED! ***\n");

                        // adds the appointment's details to the a[] and increments
the numUsers variable
                        a[numUsers] = applic;
                        numUsers++;
                              break;
                  }
            }
            }

                  else
                  printf("\n                                      [SYNTAX ERROR]
");

      fclose(fp);
}


// function declarations to avoid implicit
int isValidTime();
int Application_date();

/* ManageAppointMenu gets the user's input whether he/she opt to change something
on his/her appointment
Pre-condition: application number should be unique
*/
void ManageAppointmentMenu() {
    int i, choice, validDate = 0, validTime = 0;
    char appID[10], c;
    FILE *fp;

    fp = fopen("Appointment.txt", "r");

    if(fp == NULL)
    printf("\n                                      [ERROR: FILE NOT FOUND] ");

      else if(fp != NULL)
      {
            // prompt user to enter application ID
            printf("\n                                       Enter Application ID:
");
            fgets(appID, 10, stdin);

            fgets(a->appID, 10, fp);
            fclose(fp);

            // search for appointment with matching appID
            for (i = 0; i < numUsers; i++)
                  {
                  if (strcmp(a[i].appID, appID) == 0)
                        {
```

```
                        // display appointment management menu
                        printf("\n");
                              ManApp();

                              // read user's choice
                        scanf("%d", &choice);
                        //to avoid skipping fgets because of previous scanf
                        while((c = getchar()) != '\n' && c != EOF)
                              {
                              }

                        // perform selected action
                        switch(choice)
                              {
                              case 1: // cancel appointment
                        // shift remaining appointments left to fill gap
                        for (int j = i; j < numUsers - 1; j++)
                                    {
                              a[j] = a[j + 1];
                        }
                              numUsers--;
                              printf("\n                                       ***
APPOINTMENT CANCELLED! ***\n");
                              break;

                        case 2: // reschedule appointment
                        // checks if date is valid
                        while (!validDate)
                                    {
                              printf("\n
Enter new Date (YYYY-MM-DD): ");
                              scanf("%d %d %d", &a[i].year, &a[i].month,
&a[i].day);
                              //fprintf(fp, "%d-%d-%d", a[i].year, a[i].month,
a[i].day);

                              // validate new date
                              validDate = Application_date(a[i]);

                              if (!validDate)
                                          {
                            printf("\n                                       ***
INVALID DATE. TRY AGAIN! ***\n");
                              }
                        }

                        // checks if time is valid (24- hr format)
                        while (!validTime)
                                    {
                              printf("\n
Enter new Time (24-hour format): ");
                              scanf("%d %d", &a[i].hour, &a[i].min);
                              //fprintf(fp,"%d:%d", a[i].hour, a[i].min);

                              validTime = isValidTime(a[i]);

                              if (!validTime)
                                          {
```

```
                            printf("\n                                        ***
INVALID TIME. TRY AGAIN!!***\n");
                              }
                        }

                        printf("\n                                        ***
APPOINTMENT RESCHEDULED! ***\n");
                        break;

                        case 3: // change vaccination center location
                               // prompt user to enter new location
                        printf("\n                                        Enter new
Vaccination Center Location: ");
                        scanf("%s", a[i].loc);
                        //fprintf(fp, "%s", a[i].loc);
                        printf("\n                                        ***
VACCINATION CENTER LOCATION CHANGED!***\n");
                        break;

                        case 4: // change vaccination brand
                              // prompt user to enter new brand
                        printf("\n                                        Enter new
Vaccination Brand: ");
                        scanf("%s", a[i].vacc_name);
                        //fprintf(fp,"%s",a[i].vacc_name);
                        printf("\n                                        ***
VACCINATION BRAND CHANGED! ***\n");
                        break;

                                    default:
                        printf("\n                                        ***
INVALID CHOICE. TRY AGAIN! *** \n");
                        ManageAppointmentMenu();
                        break;
                        }

            return;
                  }
            }
                  // appointment with matching appID not found
                  printf("\n                                      *** APPOINTMENT
NOT FOUND! ***\n");
            }

                  else
                  printf("\n                                     [SYNTAX ERROR]
");

}



/
*************************************************************************************
*/
/************************************** M E N U
**************************************/
/
*************************************************************************************
```

```
*/

// function declaration to avoid implicit
int VaccinationRegistration (int vaccReg);
int DataManagement (int dataMan);

/* Menu displays the main menu, gets the user id and password of he user if he/she
opt to choose data management menu
@param menu - contains the choice of the user
Pre-condition: user id should be unique
*/
int
Menu (int menu){
      char line[MAX_LINE_LENGTH];
      int attempts = 0;
      char userID [11];
      char password [11];
      int vaccReg, dataMan;
      char choice;
      char c;
      //char avoid[5];
      FILE *fp;

      fp = fopen("Users_Log_In.txt", "r");
      DisplayTitle();
      printf("\n\n                                              Enter: ");
      scanf("%d", &choice); // user's input the choice
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

      //switch (choice){

      //case 1:
      if(choice == 1)
      {
            VaccinationRegistration(vaccReg);
            return 0;
      }


      //case 2:
      else if(choice == 2)
      {
            login();
            //fgets(avoid, 5, stdin); //no input, to avoid skipping fgets
            if(fp == NULL)
            printf("\n\n                                              [ERROR: FILE
NOT FOUND] ");

      else if(fp != NULL)
      {
            //to read user id and password store in User_Log_In file
            while (fgets(line, MAX_LINE_LENGTH, fp))
            {
            fgets(u->userID, MAX_LINE_LENGTH, fp);
                  fgets(u->password, MAX_LINE_LENGTH, fp);
      }
```

```
            while (attempts < MAX_ATTEMPTS)
            {
            printf("\n                                       User ID: ");
            fgets(userID,11,stdin);
            printf("\n                                       Password: ");
            fgets(password,11,stdin);

            int userFound = 0;
            for (int i = 0; i < num_users; i++)
            {
                  if (strcmp(u[i].userID, userID) == 0 && strcmp(u[i].password,
password) == 0)
                  {
                        userFound = 1;
                        break;
                  }
            }


            if (userFound)
            {
            DataManagement(dataMan);
            break;
            }
                  else
                  {
                        printf("\n                                     ***
INCORRECT USER ID OR PASSWORD. TRY AGAIN! ***\n");
                        attempts++;
                        if (attempts >= MAX_ATTEMPTS)
                        {
                              printf("\n                                     ***
LOGIN ATTEMPTS EXCEEDED. ***\n");
                              return 0;
                        }
                  }
            }
      }

            else
            printf("\n\n                                                [SYNTAX
ERROR] ");
      return 0;
      }


      //case 3:
      else if(choice == 3)
      {
            printf("\n                                   ***** THANK YOU *****");
            return 0;
      }


      //default:
      else
      {
            printf("\n                                       *** INVALID CHOICE.
TRY AGAIN! ***\n");
```

```
        Menu(menu);
            return 0;
      }




}

/
************************************************************************************
********************/
/************************************ D A T A M A N A G E M E N T
************************************/
/
************************************************************************************
********************/

// function declarations to avoid implicit
int User (int user);
int Appointment (int app);
int Chatbot_2(int chat);
void Export();
void Import();

/* DataManagement displays the data management menu and ask for the user's input if
he/she opt to manage user, appointment, chatbot, export, or import
@param dataMan - contains the choice of the user
*/
int
DataManagement (int dataMan){

      char c;
      int i = 0;
      int user, app, chat, exp;
      char userID [5];
      char password [10];
      int attempts = 0;
      int menu;


      DisplayDM();
      printf("\n\n                                               Enter: ");
      scanf("%d",&dataMan); // user's input the choice
      //to avoid skipping fgets because of previous scanf
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

switch(dataMan){

      case 1:
            User(user);
            break;

      case 2:
            Appointment(app);
            break;
      case 3:
            Chatbot_2(chat);
            break;


      case 4:
            DisplayExport();
            Export();
            break;

      case 5:
            DisplayImport();
            Import();
            break;

      case 6:
                  Menu (menu);
                  break;

      default:
            printf("\n                                      *** INVALID CHOICE.
TRY AGAIN! ***\n");
            DataManagement(dataMan);
            break;

      return 0;
}
}


// function declaration to avoid implicit
void AddUser();
void ViewUser();
void EditUser();
void DeleteUser();

/* User displays the menu for user in data management
@param user - contains the input of the user
*/
int
User (int user){

      char c;
      int dataMan;
      DisplayUser();
      printf("\n\n                                          Enter: ");
      scanf("%d",&user); // user's input the choice
      //to avoid skipping fgets because of previous scanf
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

switch(user){

      case 1:
            DisplayAdd();
            printf("\n");
            AddUser();
            break;
      case 2:
            DisplayView();
            printf("\n");
            ViewUser();
            break;

      case 3:
            DisplayEdit();
            printf("\n");
            EditUser();
            break;

      case 4:
            DisplayDelete();
            printf("\n");
            DeleteUser();
            break;

      case 5:
            DataManagement(dataMan);
            break;

      default:
            printf("\n                                       *** INVALID CHOICE.
TRY AGAIN! ***\n");
            User(user);
            break;

}
}

// function declaration to avoid implicit
void Registration_user();

/* AddUser returns the info of the new user
*/
void AddUser()
{
      char choice, c;
      int user;

      if (num_users == MAX_USERS)
      {
            printf("\n                                         *** MAXIMUM NUMBER
OF USERS REACHED. ***\n");
            return;
      }

      //function call
      Registration_user();
      printf("\n                                        *** USER ADDED
SUCCESSFULLY. ***\n");

      printf("\n                                         Do you want to add another
user? [Y/N] ");
      scanf(" %c", &choice);
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

      if (choice =='y' || choice == 'Y')
      AddUser();
      else
      {
      User (user);
      }
}

// function allows user to view other users
void ViewUser()
{

    int user;

    // Check if there are no users to display
    if (num_users == 0){
        printf("\n                                         *** NO USERS TO DISPLAY
***\n");
        return;
    }

    // Display the header for the table
    printf("\n                                         LIST OF USERS\n");

    printf("\
n----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
---------------------------------------------------------------------\n");
    printf("| %-8s | %-8s | %-20s | %-20s | %-15s | %-5s | %-15s | %-15s | %-15s |
%-15s | %-15s | %-15s | %-15s | %-15s |\n", "USER NO.","USER ID", "NAME",
"ADDRESS", "CONTACT", "SEX", "F.D DATE", "F.D VACCINE", "F.D LOCATION", "S.D DATE",
"S.D VACCINE", "S.D LOCATION", "B.S Date", "B.S VACCINE");

printf("---------------------------------------------------------------------------
-----------------------------------------------------------------------------------
----------------------------------------------------------------------------\n");

     // Loop through each user and display their details
    for (int i = 0; i < num_users; i++) {
        printf("| %-8d | %-8s | %-20s | %-20s | %-15s | %-5c | ", i+1, u[i].userID,
u[i].name, u[i].address, u[i].contact, u[i].sex);
        // Check if the first dose details are available, otherwise display N/A
            if (u[i].vacc_year == 0 && u[i].vacc_month == 0 && u[i].vacc_day == 0)
{
            printf("%-15s | %-15s | ", "N/A", "N/A");
        } else {
            printf("%-4d-%02d-%02d      | %-15s | %-15s | ", u[i].vacc_year,
u[i].vacc_month, u[i].vacc_day, u[i].vacc_name, u[i].vacc_loc);
        }
        // Check if the second dose details are available, otherwise display N/A
            if (u[i].sec_year == 0 && u[i].sec_month == 0 && u[i].sec_day == 0) {
            printf("%-4s-%02s-%02s      | %-15s | %-15s | ", "N/A", "N/A", "N/A",
"N/A", "N/A");
        } else {
            printf("%-4d-%02d-%02d      | %-15s | %-15s | ", u[i].sec_year,
u[i].sec_month, u[i].sec_day, u[i].sec_name, u[i].sec_loc);
        }
        // Check if the booster details are available, otherwise display N/A
            if (u[i].booster_year == 0 && u[i].booster_month == 0 &&
u[i].booster_day == 0) {
            printf("%-4s-%02s-%02s      | %-15s |\n", "N/A", "N/A", "N/A", "N/A");
        } else {
            printf("%-4d-%02d-%02d      | %-15s |\n", u[i].booster_year,
u[i].booster_month, u[i].booster_day, u[i].booster_name);
        }
    }


printf("---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------\n");
    User(user);
}

int isDateValidFirst();
int isDateValidSec();
int isDateValidBoost();

// allows the user to edit a user from the system by entering their name
void EditUser() {
    char c;
    int validDate = 0;
    char nameEdit[20], choice;

    // used to store the index of the user in u[] that matches the name entered by
user.
      // this make it easier to access the user's data.
      // allows to direct access and modify the data.
      // using of u[i] could lead to errors if there are multiple users with the
same name in the array.
    int index = -1;
    int user;

// checks if there are any users to edit
if (num_users == 0) {
    printf("\n                                       *** NO USERS TO EDIT! ***\
n");
    return;
}

printf("\n                                     Enter name of user to edit: ");
fgets(nameEdit, 20, stdin);

// Find the index of the user to be edited
index = -1;
for (int i = 0; i < num_users; i++) {
    if (strcmp(u[i].name, nameEdit) == 0) {
        index = i;
        break;
    }
}

if (index == -1) {
    printf("\n                                       *** USER NOT FOUND! ***\n");
    return;
}
```

```
// Keep asking for a unique and valid user ID //
char newUserID[MAX_ID_LENGTH];
int idAlreadyTaken;
do {
    idAlreadyTaken = 0;
    printf("\n                                        Enter new User ID: ");
    fgets(newUserID, MAX_ID_LENGTH, stdin);

    if (strcmp(newUserID, u[index].userID) == 0) {
        printf("\n                                        *** You cannot use your
current user ID! ***\n");
        idAlreadyTaken = 1;
        continue;
    }

    for (int i = 0; i < num_users; i++) {
        if (i != index && strcmp(u[i].userID, newUserID) == 0) {
            printf("\n                                        *** USER ID ALREADY
TAKEN! ***\n");
            idAlreadyTaken = 1;
            break;
        }
    }
} while (idAlreadyTaken);

// Store the new user ID and display it
strcpy(u[index].userID, newUserID);

      // if match is found, index of user is stored in the index
      for (int i = 0; i < num_users; i++){
            if (strcmp(u[i].name, nameEdit) == 0){
      printf("\n                                        Password: ");
      fgets(u[index].password, 10, stdin);
      printf("\n                                        Name: ");
      fgets(u[index].name, 20, stdin);
      printf("\n                                        Address: ");
      fgets(u[index].address, 30, stdin);
      printf("\n                                        Contact: ");
      fgets(u[index].contact, 12, stdin);
      printf("\n                                        Sex: ");
      scanf("%s", u[index].sex);

      // check if date is valid
      // input first dose information
      // input first dose information
      int validDate = 0;
      while (validDate == 0) {
    printf("\n                                        First Dose (YYYY-MM-DD): ");
    scanf("%d %d %d", &u[index].vacc_year, &u[index].vacc_month,
&u[index].vacc_day);

    // update date with the inputted date for first dose
    users.vacc_year = u[index].vacc_year;
    users.vacc_month = u[index].vacc_month;
    users.vacc_day = u[index].vacc_day;

    validDate = isDateValidFirst(date);
    if (validDate == 0) {
        printf("\n                                       *** INVALID DATE. TRY
AGAIN! ***\n");
    }
}

printf("\n                                        First Dose Vaccine: ");
scanf("%s", u[index].vacc_name);
printf("\n                                       First Dose Location: ");
scanf("%s", u[index].vacc_loc);

// input second dose information
validDate = 0;
while (validDate == 0) {
    printf("\n                                       Second Dose (YYYY-MM-DD): ");
    scanf("%d %d %d", &u[index].sec_year, &u[index].sec_month, &u[index].sec_day);

    // update date with the inputted date for second dose
    users.sec_year = u[index].sec_year;
    users.sec_month = u[index].sec_month;
    users.sec_day = u[index].sec_day;

    validDate = isDateValidSec(date);

    if (validDate == 0) {
        printf("\n                                       *** INVALID DATE. TRY
AGAIN! ***\n");
    }
}

printf("\n                                        Second Dose Vaccine: ");
scanf("%s", u[index].sec_name);
printf("\n                                        Second Dose Location: ");
scanf("%s", u[index].sec_loc);

// input booster dose information
validDate = 0;
while (validDate == 0) {
    printf("\n                                        Booster Dose (YYYY-MM-DD):
");
    scanf("%d %d %d ", &u[index].booster_year, &u[index].booster_month,
&u[index].booster_day);

    // update date with the inputted date for booster dose
    users.booster_year = u[index].booster_year;
    users.booster_month = u[index].booster_month;
    users.booster_day = u[index].booster_day;

    validDate = isDateValidBoost(date);

    if (validDate == 0) {
        printf("\n                                       *** INVALID DATE. TRY
AGAIN! ***\n");
    }
}

printf("\n                                        Booster Dose Vaccine: ");
scanf("%s ", u[index].booster_name);
printf("\n                                        Booster Dose Location: ");
```

```
scanf("%s", u[index].booster_loc);
      printf("\n                                      User details updated
successfully.\n");

      printf("\n                                               Do you want to edit
another user? [Y/N] ");
      scanf(" %c", &choice);
      while((c = getchar()) != '\n' && c != EOF)
      {
      }
      if (choice =='y' || choice == 'Y')
      EditUser();
      else
      {
      User (user);
      }
            }
      }


}

// allows the user to delete a user from the system by entering their name.
void DeleteUser() {
    char nameDelete[30], choice, c;
    int user;

    if (num_users == 0) {
        printf("\n                                               *** NO USERS TO
DELETE. ***\n");
        return;
    }

    printf("\n                                               Enter name of user to
delete: ");
    fgets(nameDelete, 30, stdin);

    for (int i = 0; i < num_users; i++) {
        if (strcmp(u[i].name, nameDelete) == 0) {
            // Move all the elements after the deleted user one position to the
left
            for (int j = i; j < num_users - 1; j++) {
                u[j] = u[j + 1];
            }
            // Decrement the number of users
            num_users--;
            printf("\n                                               User deleted
successfully.\n");
            printf("\n                                               Do you want to
delete another user? [Y/N] ");
            scanf(" %c", &choice);
            while((c = getchar()) != '\n' && c != EOF)
            {
                  }
            if (choice == 'y' || choice == 'Y') {
                DeleteUser();
            } else {
                User(user);
            }
            return;
        }
    }

    // If the function reaches this point, the user was not found
    printf("\n                                              *** USER NOT FOUND!
***\n");
    User(user);
}


// function declaration to avoid implicit
void AddApp();
void ViewApp();
void EditApp();
void DeleteApp();

/* Appointment displays the menu for appointment in data management
@param app - contains the input of the user
*/
int
Appointment (int app){
      char c;
      int dataMan;
      DisplayAppointment();
      printf("\n\n                                              Enter: ");
      scanf("%d",&app); // user's input the choice
      while((c = getchar()) != '\n' && c != EOF) //to avoid skipping fgets
    {
      }

switch(app){

      case 1:
            DisplayAddApp();
            printf("\n");
            AddApp();
            break;

      case 2:
            DisplayViewApp();
            printf("\n");
            ViewApp();
            break;

      case 3:
            DisplayEditApp();
            printf("\n");
            EditApp();
            break;

      case 4:
            DisplayDeleteApp();
            printf("\n");
            DeleteApp();
            break;

      case 5:
            DataManagement (dataMan);
            break;

        default:
             printf("\n                                       *** INVALID CHOICE.
TRY AGAIN! ***\n");
             Appointment(app);
             break;
}
}

// allows user to add an appointment to the system
void AddApp(){
      char choice,c;
      int app;

      // checks if the max number of users has been reached
      if ( num_users == MAX_USERS){
            printf("\n                                         *** MAXIMUM NUMBER
OF APPOINTMENTS REACHED. ***\n");
            return;
      }

      // function call
      Registration_app();
      printf("\n                                        *** APPOINTMENT ADDED
SUCCESSFULLY. ***\n");

      printf("\n                                         Do you want to add another
appointment? [Y/N] ");
      scanf(" %c", &choice);
      while((c = getchar()) != '\n' && c != EOF) //to avoid skipping fgets
    {
      }
      if (choice =='y' || choice == 'Y')
      AddApp();
      else{
      Appointment(app);
}
}


// function allows user to view other users
void ViewApp() {

      int app;

    if (numUsers == 0) {
        printf("\n                                            ** NO APPOINTMENTS
TO DISPLAY. ***\n");
        return;
    }

    printf("\n                                                              LIST
OF APPOINTMENTS\n");
    printf("\
n-----------------------------------------------------------------------------------
------------------------\n");
    printf("| %-8s | %-8s | %-20s | %-20s | %-15s | %-15s|\n", "USER
NO.","APPLICATION ID","NAME", "LOCATION", "DATE", "TIME");
```

```
printf("--------------------------------------------------------------------------------
-------------------------------\n");

    for (int i = 0; i < numUsers; i++) {
        printf("| %-8d | %-8s       | %-20s | %-20s | %4d-%02d-%02d      | %02d:
%02d      |\n", i+1, a[i].appID, a[i].name, a[i].loc, a[i].year, a[i].month,
a[i].day, a[i].hour, a[i].min);
    }


printf("--------------------------------------------------------------------------------
-------------------------------\n");
    Appointment(app);
}

int Application_date();
int isValidTime();


// allows the user to edit an appointment from the system by entering their name
void EditApp() {

    int validTime = 0;
      int validDate = 0;
      char nameEdit[20], choice, c;
      // used to store the index of the appointment in a[] that matches the name
entered by user.
      // this make it easier to access the appointment's data.
      // allows to direct access and modify the data.
      // using of a[i] could lead to errors if there are multiple appointments with
the same name in the array.
      int index = -1;
      int app;

      // checks if there's an appointment to edit
      if ( num_users == 0){
            printf("\n                                       *** NO APPOINTMENTS TO
EDIT. ***\n");
            return;
      }

      printf("\n                                       Enter name of appointment
to edit: ");
      fgets(nameEdit, 20, stdin);

      // searching for a user in the a[] by iterating
      // checks the name of the current appointment at index i
            for (int i = 0; i < num_users; i++){
            if (strcmp(a[i].name, nameEdit) == 0){
                  index = i;
                  break;
            }
      }

      // if no match is found
      if (index == -1){
            printf("\n                                       *** APPOINTMENT NOT
FOUND. ***\n");
```

```
            return;
      }

      // if match is found, index of appointment is stored in the index
      for (int i = 0; i < num_users; i++){
            if (strcmp(a[i].name, nameEdit) == 0){

      printf("\n                                        Application ID: ");
      fgets(a[index].appID, 10, stdin);
      printf("\n                                        Name: ");
      fgets(a[index].name, 20, stdin);
      printf("\n                                        Location: ");
      fgets(a[index].loc, 10, stdin);
      printf("\n                                        Vaccine: ");
      fgets(a[index].vacc_name, 10, stdin);

      // reset validation
      validDate = 0;
      validTime = 0;

      // checks if date is valid
      while (!validDate){

      printf("\n                                        Date [YYYY/MM/DD]: ");
      scanf ("%d %d %d", &a[index].year, &a[index].month, &a[index].day );

      validDate = Application_date(Date);

      if (!validDate){
      printf("\n                                       *** INVALID DATE. TRY AGAIN!
***\n");
      }
      }

      // checks if time is valid
      while(!validTime){
      printf("\n                                       Time (24-hour format): ");
      scanf("%d %d", &a[index].hour, &a[index].min);

      validTime = isValidTime(time);

      if (!validTime){
      printf("\n                                       *** INVALID DATE. TRY AGAIN!
***\n");
      }
      }

      printf("\n                                        Dose: ");
      scanf ("%s", a[index].dose);

      printf("\n                                      *** APPOINTMENT DETAILS
UPDATED SUCCESSFULLY! ***\n");
      printf("\n                                               Do you want to edit
another appointment? [Y/N] ");
      scanf(" %c", &choice);
      while((c = getchar()) != '\n' && c != EOF) //to avoid skipping fgets
      {
      }
      if (choice =='y' || choice == 'Y')
      EditApp();
      else{
      Appointment(app);
      }

            }
      }


}

// allows user to delete an appointment
void DeleteApp() {

      char nameDelete [30], choice,c;
      int app;

      // checks if there;s an appointment to be deleted
    if (num_users == 0){
            printf("\n                                        *** NO APPOINTMENT
TO DELETE. ***\n");
            return;
      }

      printf("\n                                        Enter name of appointment
to delete: ");
      fgets(nameDelete, 30, stdin);

      // responsible for deleting an appointment from the array of appointments
      // if match is found, appointment will be removed from the array
      for( int i = 0; i < num_users; i++){
            if ( strcmp(a[i].name, nameDelete) == 0)
            {
            for (int j = 0 ; j < num_users - 1; j++){
                  a[j] = a[j+1];
            }

      // decremented
      num_users--;

      printf("\n                                        *** APPOINTMENT DELETED
SUCCESSFULLY! ***\n");
      printf("\n                                        Do you want to delete
another appointment? [Y/N] ");
      scanf(" %c", &choice);
      while((c = getchar()) != '\n' && c != EOF)
      {
      }
      if (choice =='y' || choice == 'Y')
      DeleteApp();
      else
      {
      Appointment(app);
      }

      }
      }
      printf("\n                                      *** USER NOT FOUND! ***\n");
      Appointment(app);
```

```
}

//to avoid implicit declarations
void AddQuestionAndAnswer();
void ViewQuestionAndAnswer();
void EditQuestionAndAnswer();
void DeleteQuestionAndAnswer();

/* Chatbot_2 displays the menu for chatbot in data management
@param chat - contains the input of the user
*/
int
Chatbot_2 (int chat){

      char c;
      int dataMan;
      DisplayChatBot();
      printf("\n\n                                              Enter: ");
      scanf("%d",&chat); // user's input the choice
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

switch(chat){

      case 1:
            DisplayAddQuestion();
            printf("\n");
            AddQuestionAndAnswer();
            break;

      case 2:
            DisplayViewQuestion();
            printf("\n");
            ViewQuestionAndAnswer();
            break;

      case 3:
            DisplayEditQuestion();
            printf("\n");
            EditQuestionAndAnswer();
            break;

      case 4:
            DisplayDeleteQuestion();
            printf("\n");
            DeleteQuestionAndAnswer();
            break;

      case 5:
            DataManagement(dataMan);
            break;

      default:
            printf("\n                                        *** INVALID CHOICE.
TRY AGAIN! ***\n");
            Chatbot_2(chat);
            break;
}
}

//allows the user to add new set of question and answer (however, the new set will
only be added after the program terminates)
void
AddQuestionAndAnswer()
{
      char line[MAX_LINE_LENGTH];
      int chat;
      char choice, C;
      FILE *fp;
      fp = fopen("Chatbot.txt", "a+"); //a+ allows us to read and append at the
same time


      if (numQA == MAX_QA)
      {
            printf("\n                                         *** MAXIMUM NUMBER
OF QUESTION AND ANSWER REACHED. ***\n");
            return;
      }

            if (fp == NULL)
            {
            printf("\n                                         [ERROR: FILE NOT
FOUND] ");
      }

      while (fgets(line, MAX_LINE_LENGTH, fp))
            {
            fgets(QA->file_question, MAX_LINE_LENGTH, fp);
                  fgets(QA->file_answer, MAX_LINE_LENGTH, fp);

      }

      printf("\n                                         Add question: ");
      fgets(cb.newQuestion, 80, stdin);
      printf("\n                                         Add answer: ");
      fgets(cb.newAnswer, 80, stdin);

            printf("\n                                        *** QUESTION AND
ANSWER ADDED SUCCESFULLY ***\n");
            c[numQA] = cb;
            numQA++;

            printf("\n                                         Do you want to add
another set of question and answer? [Y/N] ");
            scanf(" %c", &choice);
            while((C = getchar()) != '\n' && C != EOF)
            {
            }

            if (choice =='y' || choice == 'Y')
            AddQuestionAndAnswer();
                  else
                  {
                        Chatbot_2(chat);
                  }
```

```
    fclose(fp);

}

//allows the user to view all question and answer
void
ViewQuestionAndAnswer()
{
      int chat;
      FILE *fp;


      fp = fopen("Chatbot.txt", "r");

      /*if (numQA == 0)
      {
        printf("\n                                             ** NO SET OF
QUESTION AND ANSWER TO DISPLAY. ***\n");
        return;
    }*/
      if(fp == NULL)
      {
            printf("\n                                         [ERROR: FILE NOT
FOUND] ");
      }

            while (fgets(c->file_question, MAX_LENGTH, fp))
            {
                  fgets(c->file_answer, MAX_LENGTH, fp);
                  printf("\n                                        Question: ");
            printf("%s", c->file_question);
            printf("\n                                         Answer: ");
            printf("%s", c->file_answer);
            }
      fclose(fp);
    Chatbot_2(chat);
}

//allows the user to edit certain question and answer
void
EditQuestionAndAnswer()
{
      //struct Chatbot c[MAX_QA];
      int num_qa = 0, chat; // number of questions and answers in the array
    char filename[30]; // input filename
    char line[MAX_LINE_LENGTH]; // line buffer for reading file
    FILE *fp; // file pointer
      char C;
      int choice1;

    printf("\n                                         Enter filename (include .txt
on filename): ");
    scanf("%s", filename);

    // check if file exists
    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("\n                                         Error: file not found.\
n");
        return;
    }
    fclose(fp);

    //load question and answer from file
    fp = fopen(filename, "r");
    while(fgets(c->file_question, MAX_LENGTH,fp))
    {
      fgets(c->file_answer, MAX_LENGTH, fp);

      // add question and answer to array
        if (num_qa < MAX_QA)
            {
            strcpy(c[num_qa].file_question, c->file_question);
            strcpy(c[num_qa].file_answer, c->file_answer);
            num_qa++;
        }
                  else
                  {
                  printf("\n                                         Error: maximum
number of questions and answers reached.\n");
                  return;
            }
      }
      fclose(fp);

      // print questions and answers
    printf("\n                                       Questions and answers\n");
    for (int i = 0; i < num_qa; i++)
      {
        printf("%d.\n                                         Question: %s\n
Answer: %s\n", i+1, c[i].file_question, c[i].file_answer);
    }

    // edit questions and answers
    printf("\n                                       Enter question number to
edit (or 0 to exit): ");
    int choice;
    scanf("%d", &choice);
    while((C = getchar()) != '\n' && C != EOF)
    {
      }

    while (choice != 0)
      {
        if (choice < 1 || choice > num_qa)
            {
            printf("\n                                        Error: invalid
question number.\n");
        }
            else
            {
            // prompt user for new question and answer
            printf("\n                                        Enter new question:
");
            fgets(c[choice-1].newQuestion, MAX_LENGTH, stdin);
            printf("\n                                        Enter new answer:
");
```

```
            fgets(c[choice-1].newAnswer, MAX_LENGTH, stdin);

            /* update file with new questions and answers
            fp = fopen(filename, "a");
            for (int i = 0; i < num_qa; i++) {
                fprintf(fp, "%s\n%s\n", c[i].newQuestion, c[i].newAnswer);
            }
            fclose(fp);*/

            printf("\n                                         Question %d
updated.\n", choice);
        }


        // prompt user for next question number to edit
        printf("\n                                         Enter question number to
edit (or 0 to exit): ");
        scanf("%d", &choice1);
    }

    if(choice == 0 || choice1 == 0)
    Chatbot_2(chat);
}

//allows the user to delete certain question and answer
void
DeleteQuestionAndAnswer()
{

    FILE *fp, *temp;
    char line[100];
    char Chatbot[] = "Chatbot.txt"; // original file name
    char temporary_file[] = "Chatbot.txt"; // temporary file name
    char question[] = "Question1"; // question to be deleted
    char answer[] = "Answer1"; // answer to be deleted
    int found = 0;

    // Open original file in read mode
    fp = fopen(Chatbot, "r");

    // Open temporary file in write mode
    temp = fopen(temporary_file, "w");

    // Read each line from original file and write to temporary file
    while (fgets(line, 100, fp) != NULL)
      {
        // Check if line contains question and answer to be deleted
        if (strcmp(line, question) == 0 && strcmp(line, answer) == 0)
            {
            found = 1;
        }
        // Write line to temporary file
        fprintf(temp, "%s", line);
    }

    // Close both files
    fclose(fp);
    fclose(temp);
    if (found)
      {
        // Delete original file
        remove(Chatbot);

        // Rename temporary file to original file name
        rename(temporary_file, Chatbot);

        printf("Question and answer deleted successfully.");
    }
      else
      {
        printf("Question and answer not found in the list.");
    }

}

//allows the user to export new question and answer to the existing file
void
ExportChatbot()
{
      int dataMan;
      FILE *fp;

      fp = fopen("Chatbot.txt", "a");

      if(fp == NULL)
    {
      printf("\n                                         [ERROR: FILE NOT FOUND]
");
      }

            for(int i = 0; i < MAX_QA ; i++)
            {
                  fprintf(fp, "%s", c[i].newQuestion);
                  fprintf(fp, "%s", c[i].newAnswer);
            }

      printf("\n\n                           *** YOUR DATA HAS BEEN EXPORTED
SUCCESSFULLY! ***\n");
      DataManagement(dataMan);

    fclose(fp);
}

//allows the user to export all the appointment to a file
void
ExportAppt()
{
      int dataMan;
      FILE *fp;
      char line[MAX_LINE_LENGTH];

      fp = fopen("Appointment.txt", "a");

      if(fp == NULL)
    {
      printf("\n                                         [ERROR: FILE NOT FOUND]
");
```

```
      }


    while (fgets(line, MAX_LINE_LENGTH, fp))
      {
            fgets(a->appID, MAX_LINE_LENGTH, fp);
            fgets(u->userID, MAX_LINE_LENGTH, fp);
            fgets(a->name, MAX_LINE_LENGTH, fp);
            fgets(a->loc, MAX_LINE_LENGTH, fp);
            fgets(a->vacc_name, MAX_LINE_LENGTH, fp);
            scanf("%d %d %d", &a->year, &a->month, &a->day);
            scanf("%d %d", &a->hour, &a->min);
            scanf("%s", a->dose);
      }
            for(int i = 0; i < num_users; i++)
            {
                  fprintf(fp, "%s%s", a[i].appID, u[i].userID);
                  fprintf(fp, "%s", a[i].name);
                  fprintf(fp, "%s", a[i].loc);
                  fprintf(fp, "%s", a[i].vacc_name);
                  fprintf(fp, "%d-%d-%d\n", a[i].year, a[i].month, a[i].day);
                  fprintf(fp, "%d:%d\n", a[i].hour, a[i].min);
                  fprintf(fp, "%s\n", a[i].dose);
            }

            printf("\n\n                          *** YOUR DATA HAS BEEN EXPORTED
SUCCESSFULLY! ***\n");
            DataManagement(dataMan);

    fclose(fp);
}

//allows the user to export all the user's information to a file
void
ExportUser()
{

      char line[MAX_LINE_LENGTH];
      int dataMan;
      FILE *fp;
      fp = fopen("User.txt", "a");

      if(fp == NULL)
    {
      printf("\n                                         [ERROR: FILE NOT FOUND]
");
      }


                  while (fgets(line, MAX_LINE_LENGTH, fp))
            {
                  fgets(u->userID, MAX_LINE_LENGTH, fp);
                  fgets(u->password, MAX_LINE_LENGTH, fp);
                  fgets(u->name, MAX_LINE_LENGTH, fp);
                  fgets(u->address, MAX_LINE_LENGTH, fp);
                  fgets(u->contact, MAX_LINE_LENGTH, fp);
                  scanf("%s", u->sex);
                  scanf("%s", u->vacc_loc);
                  scanf("%d %d %d",&u->vacc_year, &u->vacc_month, &u->vacc_day);
                  scanf("%s", u->vacc_name);
                  scanf("%s", u->sec_loc);
                  scanf("%d %d %d",  &u->sec_year, &u->sec_month, &u->sec_day);
                  scanf("%s", u->sec_name);
                  scanf("%s", u->booster_loc);
                  scanf("%d %d %d", &u->booster_year, &u->booster_month, &u-
>booster_day);
                  scanf("%s", u->booster_name);
            }

                  for(int i = 0; i < num_users; i++)
                  {
                  fprintf(fp, "%s%s", u[i].userID, u[i].password);
                  fprintf(fp, "%s", u[i].name);
                  fprintf(fp, "%s", u[i].address);
                  fprintf(fp, "%s\n", u[i].contact);
                  fprintf(fp, "%s\n", u[i].sex);
                  fprintf(fp, "%s ", u[i].vacc_loc);
                  fprintf(fp, "%d-%d-%d ",u[i].vacc_year, u[i].vacc_month,
u[i].vacc_day);
                  fprintf(fp, "%s\n", u[i].vacc_name);
                  fprintf(fp, "%s ", u[i].sec_loc);
                  fprintf(fp, "%d-%d-%d ",u[i].sec_year, u[i].sec_month,
u[i].sec_day);
                  fprintf(fp, "%s\n", u[i].sec_name);
                  fprintf(fp, "%s ", u[i].booster_loc);
                  fprintf(fp, "%d-%d-%d ",u[i].booster_year, u[i].booster_month,
u[i].booster_day);
                  fprintf(fp, "%s", u[i].booster_name);

                  fprintf(fp, "\n");

                  if(u[i].userID == 0 && u[i].password == 0 && u[i].name == 0 &&
u[i].address == 0 && u[i].contact == 0 && u[i].sex == 0 && u[i].vacc_loc == 0 &&
u[i].vacc_year == 0 && u[i].vacc_month == 0 && u[i].vacc_day == 0 && u[i].vacc_name
== 0 && u[i].sec_loc == 0 && u[i].sec_year == 0 && u[i].sec_month == 0 &&
u[i].sec_day == 0 && u[i].sec_name == 0 && u[i].booster_loc == 0 &&
u[i].booster_year == 0 && u[i].booster_month == 0 && u[i].booster_day == 0 &&
u[i].booster_name == 0)
                  fprintf(fp, "N/A");

                  }

            printf("\n\n                          *** YOUR DATA HAS BEEN EXPORTED
SUCCESSFULLY! ***\n");
            DataManagement(dataMan);

      fclose(fp);
}

//Export - gets the filename to be search
void
Export()
{
      char filename[MAX_FILENAME], c;


      printf("\n                                        Enter filename: ",
MAX_FILENAME);
```

```
    scanf("%s", filename);
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

    if(strcmp(filename, "User") == 0)
    {
            DisplayExportUser();
            printf("\n                                         GATHERING DATA TO BE 
SAVED... ");
            ExportUser();
      }

            else if(strcmp(filename, "Appointment") == 0)
            {
                  DisplayExportAppt();
                  printf("\n                                         GATHERING DATA 
TO BE SAVED... ");
                  ExportAppt();
            }

                  else if(strcmp(filename, "Chatbot") == 0)
                  {
                        DisplayExportChatbot();
                        printf("\n                                         
GATHERING DATA TO BE SAVED... ");
                        ExportChatbot();
                  }
      else
      {
            printf("\n                                         FILE DOES NOT 
EXISTS! ");
            Export();
      }

}

//allows the user to import certain question and answer
void
ImportChatbot()
{
      int dataMan;
      FILE *fp;

      fp = fopen("Chatbot.txt", "r");

      if(fp == NULL)
    {
      printf("\n                                         [ERROR: FILE NOT FOUND] 
");
      }

            for(int i = 0; i < MAX_QA ; i++)
            {
                  fprintf(fp, "%s", c[i].file_question);
                  fprintf(fp, "%s", c[i].file_answer);
            }

      printf("\n\n                           *** YOUR DATA HAS BEEN IMPORTED
SUCCESSFULLY! ***\n");
      DataManagement(dataMan);

    fclose(fp);
}

//allows the user to import appointment
void
ImportAppt()
{
      int dataMan;
      FILE *fp;
      char line[MAX_LINE_LENGTH];

      fp = fopen("Appointment.txt", "a");

      if(fp == NULL)
    {
      printf("\n                                         [ERROR: FILE NOT FOUND]
");
      }


    while (fgets(line, MAX_LINE_LENGTH, fp))
      {
            fgets(a->appID, MAX_LINE_LENGTH, fp);
            fgets(u->userID, MAX_LINE_LENGTH, fp);
            fgets(a->name, MAX_LINE_LENGTH, fp);
            fgets(a->loc, MAX_LINE_LENGTH, fp);
            fgets(a->vacc_name, MAX_LINE_LENGTH, fp);
            scanf("%d %d %d", &a->year, &a->month, &a->day);
            scanf("%d %d", &a->hour, &a->min);
            scanf("%s", a->dose);
      }
            for(int i = 0; i < num_users; i++)
            {
                  printf("\n                                         Application
ID: %s\n                                        User ID: %s", a[i].appID,
u[i].userID);
                  printf("\n                                         Name: %s",
a[i].name);
                  printf("\n                                         Location: %s",
a[i].loc);
                  printf("\n                                         Vaccine name:
%s", a[i].vacc_name);
                  printf("\n                                         Date: %d-%d-
%d\n", a[i].year, a[i].month, a[i].day);
                  printf("\n                                         Time: %d:%d\
n", a[i].hour, a[i].min);
                  printf("\n                                         Dose: %s\n",
a[i].dose);
            }

            printf("\n\n                           *** YOUR DATA HAS BEEN IMPORTED
SUCCESSFULLY! ***\n");
            DataManagement(dataMan);

    fclose(fp);
}
```

```
//allows the user to import user information
void
ImportUser()
{
      char line[MAX_LINE_LENGTH];
      int dataMan;
      FILE *fp;
      fp = fopen("User.txt", "r");

      if(fp == NULL)
    {
      printf("\n                                         [ERROR: FILE NOT FOUND]
");
      }


                  while (fgets(line, MAX_LINE_LENGTH, fp))
            {
                  fgets(u->userID, MAX_LINE_LENGTH, fp);
                  fgets(u->password, MAX_LINE_LENGTH, fp);
                  fgets(u->name, MAX_LINE_LENGTH, fp);
                  fgets(u->address, MAX_LINE_LENGTH, fp);
                  fgets(u->contact, MAX_LINE_LENGTH, fp);
                  scanf("%s", u->sex);
                  scanf("%s", u->vacc_loc);
                  scanf("%d %d %d",&u->vacc_year, &u->vacc_month, &u->vacc_day);
                  scanf("%s", u->vacc_name);
                  scanf("%s", u->sec_loc);
                  scanf("%d %d %d",  &u->sec_year, &u->sec_month, &u->sec_day);
                  scanf("%s", u->sec_name);
                  scanf("%s", u->booster_loc);
                  scanf("%d %d %d", &u->booster_year, &u->booster_month, &u-
>booster_day);
                  scanf("%s", u->booster_name);
            }

                  for(int i = 0; i < num_users; i++)
                  {
                  printf("\n                                         User ID: %s \n
Passowrd: %s", u[i].userID, u[i].password);
                  printf("\n                                         Name: %s",
u[i].name);
                  printf("\n                                         Address: %s",
u[i].address);
                  printf("\n                                         Contact:%s\n",
u[i].contact);
                  printf("\n                                         Sex: %s\n",
u[i].sex);
                  printf("\n                                         First dose
location: %s ", u[i].vacc_loc);
                  printf("\n                                         First dose
(YYYY-MM-DD): %d-%d-%d ",u[i].vacc_year, u[i].vacc_month, u[i].vacc_day);
                  printf("\n                                         First dose
vaccine: %s\n", u[i].vacc_name);
                  printf("\n                                         Second dose
location: %s ", u[i].sec_loc);
                  printf("\n                                         Second dose
(YYYY-MM-DD): %d-%d-%d ",u[i].sec_year, u[i].sec_month, u[i].sec_day);
```

```
                  printf("\n                                         Second dose
vaccine: %s\n", u[i].sec_name);
                  printf("\n                                         Booster shot
location: %s ", u[i].booster_loc);
                  printf("\n                                         Booster shot
(YYYY-MM-DD): %d-%d-%d ",u[i].booster_year, u[i].booster_month, u[i].booster_day);
                  printf("\n                                         Booster shot
vaccine: %s", u[i].booster_name);

                  fprintf(fp, "\n");

                  if(u[i].userID == 0 && u[i].password == 0 && u[i].name == 0 &&
u[i].address == 0 && u[i].contact == 0 && u[i].sex == 0 && u[i].vacc_loc == 0 &&
u[i].vacc_year == 0 && u[i].vacc_month == 0 && u[i].vacc_day == 0 && u[i].vacc_name
== 0 && u[i].sec_loc == 0 && u[i].sec_year == 0 && u[i].sec_month == 0 &&
u[i].sec_day == 0 && u[i].sec_name == 0 && u[i].booster_loc == 0 &&
u[i].booster_year == 0 && u[i].booster_month == 0 && u[i].booster_day == 0 &&
u[i].booster_name == 0)
                  fprintf(fp, "N/A");

                  }

            printf("\n\n                          *** YOUR DATA HAS BEEN IMPORTED
SUCCESSFULLY! ***\n");
            DataManagement(dataMan);

      fclose(fp);
}

//Import - gets the name of the file
void
Import()
{
      char filename[MAX_FILENAME], c;


      printf("\n                                         Enter filename: ",
MAX_FILENAME);
    scanf("%s", filename);
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

    if(strcmp(filename, "User") == 0)
    {
            DisplayImportUser();
            printf("\n                                         SCANNING DATA FROM
FILE... ");
            ImportUser();
      }

            else if(strcmp(filename, "Appointment") == 0)
            {
                  DisplayImportAppt();
                  printf("\n                                         SCANNING DATA
FROM FILE... ");
                  ImportAppt();
            }
```

```
                  else if(strcmp(filename, "Chatbot") == 0)
                  {
                        DisplayImportChatbot();
                        printf("\n                                          SCANNING 
DATA FROM FILE... ");
                        ImportChatbot();
                  }
      else
      printf("\n                                         FILE DOES NOT EXISTS! ");
}

/
*************************************************************************************
***************************************/
/*************************************** V A C C I N A T I O N R E G I S T R A T I O 
N ***************************************/
/
*************************************************************************************
***************************************/

void Registration();
void VaccApp();
void Chatbot();

/* VaccinationRegistration displays the menu for vaccination registration
@param vaccReg - contains the input of the user
pre-condition: user id should be unique
*/
int
VaccinationRegistration (int vaccReg)
{
      char line[100];
      char userID [MAX_LINE_LENGTH];
      char password [MAX_LINE_LENGTH];
      int vaccapp;
      int attempts= 0;
      char c;
      FILE *fp;
      int choice, menu;
      fp = fopen("Users_Log_In.txt", "r");

      DisplayVRM();
      printf("\n\n                                                Enter: ");
      scanf("%d",&vaccReg); // user's input the choice

      //to avoid skipping fgets because of previous scanf
      while((c = getchar()) != '\n' && c != EOF)
      {
      }

switch(vaccReg)
{
      case 1:
            DisplayReg();
            printf("\n");
            Registration_user();
            printf("\n                         *** CONGRATULATIONS! YOUR 
REGISTRATION HAS BEEN COMPLETED SUCCESSFULLY! ***\n");
            VaccinationRegistration (vaccReg);
            break;

      case 2:
            login();
            if(fp == NULL)
            printf("\n                                       [ERROR: FILE NOT FOUND]
");

                  else if(fp != NULL)
                  {

                        //to read user id and password store in User_Log_In file
                        while (fgets(line, MAX_LINE_LENGTH, fp))
                        {
                        fgets(u->userID, MAX_LINE_LENGTH, fp);
                              fgets(u->password, MAX_LINE_LENGTH, fp);

                  }
                        while (attempts != MAX_ATTEMPTS)
                              {

                              printf("\n                                       User
ID: ");
                              fgets(userID, MAX_LINE_LENGTH, stdin);
                              printf("\n
Password: ");
                              fgets(password, MAX_LINE_LENGTH, stdin);

                              int userFound = 0;

                                    for (int i = 0; i < MAX_USERS; i++)
                                    {
                                                //compares the user's input to the
data from the file
                                                if (strcmp(u[i].userID, userID) ==
0 && strcmp(u[i].password, password) == 0)
                                                {
                                                      userFound = 1;
                                                      break;
                                                }

                                    }

                              if (userFound)
                              {
                                    DisplayVA();
                                    printf("\n");
                                    Vacc_App (vaccapp);
                              }
                                    else
                                    {
                                          printf("\n
*** INCORRECT USER ID OR PASSWORD. TRY AGAIN! ***\n");
                                          attempts++;
                                                if (attempts >= MAX_ATTEMPTS)
                                                {
                                                      printf("\n
*** LOGIN ATTEMPTS EXCEEDED! ***\n");
                                                      return 0;
                                                }
                                        }
                                }

                        }
                                else
                                {
                                printf("\n                                      [SYNTAX
ERROR] ");
                        }
                break;

                case 3:
                        DisplayChat();
                        Chatbot();
                        break;


                case 4:
                        Menu(menu);
                        break;

                 default:
                        printf("\n                                      *** INVALID CHOICE.
TRY AGAIN! ***\n");
                        VaccinationRegistration(vaccReg);
                                break;
                }


}

 // Date checker in Users
int
isDateValidFirst()
{
        // Array of months and their number of days
        int daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

        //checks is months and day are within the valid ranges
        // First Vaccine
        int validYear = users.vacc_year >= 0;
        int validMonth = users.vacc_month >= 1 && users.vacc_month <= 12;
        int validDay = users.vacc_day >= 1 && users.vacc_day <= daysInMonth
[users.vacc_month - 1];

        // checks if it's leap year
        // First
        if (users.vacc_year % 4 == 0 && (users.vacc_year % 100 != 0 ||
users.vacc_year % 400 == 0)){
                daysInMonth [1] = 29;
        }

        if (validYear && validMonth && validDay ){
                return 1;
        }
        else {
                return 0;
        }
```

```
}

//function to check whether the date for second dose is valid
int
isDateValidSec()
{
      // Array of months and their number of days
      int daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

      //checks is months and day are within the valid ranges
      //Second Vaccine
      int validSecYear = users.sec_year >= 0;
      int validSecMonth = users.sec_month >= 1 && users.sec_month <= 12;
      int validSecDay = users.sec_day >= 1 && users.sec_day <= daysInMonth
[users.sec_month - 1];

      // Second
      if (users.sec_year % 4 == 0 && (users.sec_year % 100 != 0 || users.sec_year %
400 == 0)){
            daysInMonth [1] = 29;
      }

      if (validSecYear && validSecMonth && validSecDay){
            return 1;
      }
      else {
            return 0;
      }
}

//function to check whether the date for booster is valid
int
isDateValidBoost()
{
      // Array of months and their number of days
      int daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

      //checks is months and day are within the valid ranges
      // Booster Shot
      int validBoostYear = users.booster_year >= 0;
      int validBoostMonth = users.booster_month >= 1 && users.booster_month <= 12;
      int validBoostDay = users.booster_day >= 1 && users.booster_day <=
daysInMonth [users.booster_month - 1];

      // checks if it's leap year
      // Booster
      if (users.booster_year % 4 == 0 && (users.booster_year % 100 != 0 ||
users.booster_year % 400 == 0)){
            daysInMonth [1] = 29; // [1] = Feb
      }

      if (validBoostYear && validBoostMonth && validBoostDay ){
            return 1;
      }
      else {
            return 0;
      }
}
```

```
// Function to check if a given user ID is unique
int
isUserID_Valid()
{
    int valid_id = 1;

    // check if user ID is less than or equal to 10 digits long
    if (strlen(users.userID) > 10) {
        printf("\n                                User ID should be less than or
equal to 10 digits long.\n");
        valid_id = 0;
    }

    // check if user ID consists of integers only
    for (int i = 0; i < strlen(users.userID); i++) {
        if (users.userID[i] < '0' && users.userID[i] > '9') {
            printf("\n                                User ID should consist of
integers only.\n");
            valid_id = 0;
            break;
        }
    }

    // check if user ID is already taken
    for (int i = 0; i < num_users; i++) {
        if (strcmp(users.userID, u[i].userID) == 0) {
            printf("\n                                *** USER ID IS ALREADY TAKEN!
***\n");
            valid_id = 0;
            break;
        }
    }

    return valid_id;
}

//function that gets all the user information ofr user registration
void
Registration_user()
{

      int validDate = 0;
      int vaccReg;
      char letter, letter1;
      char user_id[11]; // variable to store user ID
    FILE *fp, *fp1;
    int validDate_sec = 0, validDate_boost = 0;
    char c;

    fp = fopen("Users.txt", "a");
    fp1 = fopen("Users_Log_In", "a");

    if(fp == NULL && fp1 == NULL)
    printf("\n                                      [ERROR: FILE NOT FOUND] ");

      else if(fp != NULL && fp1 != NULL)
      {
            while (1)
                  {

            printf("\n\n                                      User ID: ");
            fgets(users.userID,10,stdin);
            //fprintf(fp, "%s", users.userID);
            fprintf(fp1, "%s", users.userID);

                  printf("\n                                         Password: ");
                  fgets(users.password, 10, stdin);
                  //fprintf(fp,"%s", users.password);
                  fprintf(fp1,"%s\n", users.password);

                   // check if user ID is valid and not already taken
                   if (isUserID_Valid())
                  {



                        printf("\n                                       Name: ");
                        fgets(users.name, 20, stdin);
                        //fprintf(fp,"%s", users.name);

                        printf("\n                                       Address: 
");
                        fgets(users.address, 30, stdin);
                        //fprintf(fp,"%s", users.address);

                        printf("\n                                       Contact: 
");
                        fgets(users.contact, 12, stdin);
                        //fprintf(fp,"%s", users.contact);

                        printf("\n                                       Sex: ");
                        scanf("%s", users.sex);
                        //fprintf(fp,"%s\n", users.sex);

                        // checks if date is valid
                        while (!validDate)
                        {
                              printf("\n 
First Dose (YYYY-MM-DD): ");
                              scanf("%d %d %d", &users.vacc_year, 
&users.vacc_month, &users.vacc_day);
                              //fprintf(fp, "%d-%d-%d ", users.vacc_year, 
users.vacc_month, users.vacc_day);

                              validDate = isDateValidFirst(users);

                              if (!validDate)
                              {
                              printf("\n                                   *** 
INVALID DATE. TRY AGAIN! ***\n");
                              }

                        }
                              // preventing data from previous registrations from 
carrying over to the next registration
                              // for reset
                              users.sec_year = 0;
                              users.sec_month = 0;
                              users.sec_day = 0;
                              strcpy(users.sec_name, "");
                              strcpy(users.sec_loc, "");
                              strcpy(users.booster_name, "");
                              strcpy(users.booster_loc, "");

                              printf("\n
First Dose Vaccine: ");
                              scanf("%s", users.vacc_name);
                              //fprintf(fp, "%s ", users.vacc_name);
                              printf("\n
First Dose Location: ");
                              scanf("%s", users.vacc_loc);
                              //fprintf(fp,"%s", users.vacc_loc);
                              //
                              users.sec_year = 0;
                              users.sec_month = 0;
                              users.sec_day = 0;
                              strcpy(users.sec_name, "");
                              strcpy(users.sec_loc, "");
                              strcpy(users.booster_name, "");
                              strcpy(users.booster_loc, "");
                              printf("\n                                       Have
you received your second dose? [Y/N] ");
                              scanf(" %c", &letter);

      if ( letter == 'y' || letter =='Y')
      {
            // will continue to prompt the user to input a date until a valid date
is inputted.
                  while (!validDate_sec)
                  {

                        printf("\n                                       Second
Dose (YYYY-MM-DD): ");
                        scanf("%d %d %d", &users.sec_year, &users.sec_month,
&users.sec_day);
                        //fprintf(fp, "\n%d-%d-%d ", users.sec_year,
users.sec_month, users.sec_day);

                        // function check if the date is valid
                        validDate_sec = isDateValidSec(users);

                        if (!validDate_sec)
                        {
                        printf("\n                                       ***
INVALID DATE. TRY AGAIN! ***\n");
                        }
                  }

                  printf("\n                                       Second Dose
Vaccine: ");
                  scanf("%s", users.sec_name);
                  //fprintf(fp,"%s ", users.sec_name);
                  printf("\n                                       Second Dose
Location: ");
                  scanf("%s", users.sec_loc);
                  //fprintf(fp, "%s", users.sec_loc);
      }
            //else if(letter1 == 'n' || letter1 == 'N')
            //fprintf(fp, "\n");

      printf("\n                                       Have you received your
Booster shot? [Y/N] ");
      scanf(" %c", &letter1);

      if ( letter1 == 'y' || letter1 =='Y')
      {
            while (!validDate_boost)
                  {
                        printf("\n                                       Booster
Dose (YYYY-MM-DD): ");
                        scanf("%d %d %d", &users.booster_year,
&users.booster_month, &users.booster_day);
                        //fprintf(fp, "\n%d-%d-%d ", users.booster_year,
users.booster_month, users.booster_day);

                        // function check if the date is valid
                        validDate_boost = isDateValidBoost(users);

                        if (!validDate_boost)
                        {
                        printf("\n                                       ***
INVALID DATE. TRY AGAIN! ***\n");
                        }
                  }

                  printf("\n                                    Booster Dose
Vaccine: ");
                  scanf("%s", users.booster_name);
                  //fprintf(fp, "%s ", users.booster_name);
                  printf("\n                                    Booster Dose
Location: ");
                  scanf("%s", users.booster_loc);
                  //fprintf(fp, "%s\n", users.booster_loc);
      }
            //else if(letter1 == 'n' || letter1 == 'N')
            //fprintf(fp, "\n");
            // adds the user's details to the u[] and increments the num_users
variable
            u[num_users] = users;
            num_users++;
            //if(num_users > 0)
            //fprintf(fp, "\n");
            break;
                  }

            }
      }
                  else
                  printf("\n                                        [SYNTAX ERROR]
");
    fclose(fp);
}

//function that allows the user to talk to a chatbot
```

```
void
Chatbot()
{
      int vaccReg;
    FILE *fp;
    int found = 0;

      fp = fopen("Chatbot.txt", "a+");
    if (fp == NULL)
      {
        printf("\n                                         [ERROR: FILE NOT FOUND]
");
    }

    while (1)
      {
        printf("\n\n                                     User: ");
        fgets(c->user_question, MAX_LENGTH, stdin);
        c->user_question[strlen(c->user_question)-1] = '\0'; // Remove the newline
character at the end of the input string

        rewind(fp); // Move the file pointer back to the beginning of the file

        while (fgets(c->file_question, MAX_LENGTH, fp) != NULL)
            {
            c->file_question[strlen(c->file_question)-1] = '\0'; // Remove the
newline character at the end of the question string

            if (strcmp(c->user_question, c->file_question) == 0)
                  {
                found = 1;
                break;
            }
        }

            if(strcmp(c->user_question, "exit") == 0)
       {
            printf("\n                                    VacciBot: Thank you for
using VacciBot. Goodbye! ");
            VaccinationRegistration (vaccReg);
            break;
            }

                  if (found)
                  {
                  fgets(c->file_answer, MAX_LENGTH, fp);
                  c->file_answer[strlen(c->file_answer)-1] = '\0'; // Remove the
newline character at the end of the answer string
                  printf("\n                                    VacciBot: %s\n",
c->file_answer);
                  }
                        else if(!found)
                        {
                        printf("\n                                    VacciBot:
Sorry, I don't know the answer. Please type another question. ");

                        Chatbot();
                  }
     found = 0;
```

```
        }

fclose(fp);
}
```