P-I

Memoria de documentación del Proyecto

DIVOC_
MIGUEL PASTORIZA

INDICE

El Módulo Principal

El módulo principal del proyecto escrito en el archivo divoc.c tiene la función de dar forma al proyecto tanto de manera externa para el usuario como de manera interna para los programadores del mismo.

Este primer archivo está compuesto únicamente por la función main:

Int main()

Esta primera función tiene como finalidad mostrar el menú principal del programa y dejar acceder al usuario a sus distintas opciones. Por otra parte, es aquí donde se inicializa nuestra struct unPaciente losPacientes mediante la que almacenaremos los pacientes de nuestro programa.

```
int main (){

  char opcion[2];
  int salir = 0;

  struct unPaciente losPacientes[250];
  int numPacientes;
  numPacientes=0;

  archivo_pacientes(losPacientes,&numPacientes);

  caratula();


  do{
    printf("\n\nR) Register a patient\n");
    printf("S) Search for a patient\n");
    printf("D) Discharge a patient\n");
    printf("L) List patients by age\n");
    printf("P) Mark positive\n");
    printf("\nX) Exit the program");

    get_character("\nChoose an option","RSDLPX",opcion);


    switch(toupper(opcion[0])){
    case P_REGISTER:
      p_register(losPacientes, &numPacientes);
      break;
    case P_SEARCH:
      p_search(losPacientes,&numPacientes);
      break;
    case P_DISCHARGE:
      p_discharge(losPacientes,&numPacientes);
      break;
    case P_LIST:
      p_list(losPacientes,&numPacientes);
      break;
    case P_MARK:
      p_mark(losPacientes,&numPacientes);
      break;
    case P_EXIT:
      salir=p_exit(losPacientes,&numPacientes);
      if (salir==1){
      }
      break;
    default:
      printf("\nError de seleccion\n");


    }


  }while(salir!=1);

  return 0;
}
```

En primer lugar, declaramos las funciones “opción” y “salir” que usaremos a lo largo del main para elegir entre las distintas opciones y para salir del programa. Declaramos nuestra struct losPacientes con espacio para almacenar a 250 pacientes y en un primer momento diremos que el número de pacientes que tenemos es igual a cero.

Llamamos a la función archivo_pacientes pasándole por referencia losPacientes y el número de los mismos. Esta función leerá del archivo correspondiente los pacientes con los que comenzaremos nuestro programa.

Con la función caratula imprimimos en pantalla el título de nuestro programa y posteriormente entrando en nuestro bucle do-while mostraremos al usuario las distintas funcionalidades del programa. La función get_character invitará al usuario a elegir entre las distintas opciones que le ofrecemos al usuario (RSDLPX) y con las que trabajaremos en el nuevo bucle switch.

En este bucle switch accederemos a las distintas opciones del programa según el usuario desee hasta que en su caso seleccionemos la función de p_exit y tras verificar que realmente queremos salir del programa saldremos del mismo.

Módulo Gesto de la Base de Datos

Este es el módulo encargado de gestionar las distintas funcionalidades del programa desde registrar un paciente, hasta eliminarlo o buscarlo en la tabla de pacientes. Su finalidad es trabajar con la struct unPaciente y sus distintas opciones.

int p_register (struct unPaciente *nuevoPaciente, int *numero)

La función p_register recibirá por referencia los datos de nuestra struct y el número de pacientes. Como su nombre indica tiene como finalidad registrar a un nuevo paciente en nuestra base de datos.

```
int p_register(struct unPaciente *nuevoPaciente, int *numero){

  int tos;
  int fiebre;
  short int edad;
  char sintoma[256];
  char nombre[256];
  char dni[256];

  printf("\nRegister\n");

  get_string("\nName (1-24 char):",nombre,1,24);

  do{
    get_string("\nDNI (9-9 char):",dni,9,9);
    if (verify_dni(dni)==0){
      printf("\nInvalid DNI");
    }
  } while(verify_dni(dni)==0);

  get_integer("\nDate [1900-2020]:",&edad,1900,2020);

  fiebre=yes_or_no("\nFever (y/n):");

  tos=yes_or_no("\nCough (y/n):");

  get_character("Symptom","FSTMN",sintoma);

  strcpy(nuevoPaciente[*numero].nombre,nombre);
  strcpy(nuevoPaciente[*numero].dni,dni);
  nuevoPaciente[*numero].edad=edad;
  nuevoPaciente[*numero].fiebre=fiebre;
  nuevoPaciente[*numero].tos=tos;
  strcpy(nuevoPaciente[*numero].sintoma,sintoma);

  printf("\nNew patient:\n");

  display_patient(nuevoPaciente,numero);

  (*numero)++;


  return 0;
```

Tras mostrar en pantalla que la opción seleccionada ha sido la de “Register” a un paciente invita al usuario a introducir sus datos personales. En primer lugar, será su nombre, una cadena de caracteres de mínimo un carácter y máximo 25 que será copiado en la variable “nombre” a través de la función get_string.

En segundo lugar, mediante la misma función get_string invitaremos al usuario a introducir su DNI, compuesto por 8 números y un carácter, este DNI será pasado a la función verify_dni para comprobar que es un DNI correcto, en el caso de que no lo sea se le pedirá en bucle al usuario que introduzca un DNI válido.

Con la función get_integer invitaremos y copiaremos en la función edad el año de nacimiento de nuestro usuario comprendido entre 1900 y 2020.

Posteriormente mediante la función yes_or_no invitaremos al usuario a introducir si tiene los síntomas de la fiebre y la tos cuya respuesta correcta sólo podrá ser “yes” or “no” y que almacenaremos en las correspondientes variables declaradas al inicio de nuestra fusión.

En último lugar invitaremos al usuario a añadir uno de los síntomas que tenga, entre las opciones elegidas estará permitido seleccionar, “**F**atiga”, “Pérdida del olfato (**S**)”, “Pérdida del gusto (**T**)”,” Molestias **M**usculares” o “**N**inguno”.

Tras preguntar al usuario esta información procederemos a guardar esta misma en nuestra struct o base de datos. Esto se hará copiando mediante strcpy la cadena de caracteres solicitadas (nombre, DNI, y el síntoma) y guardando el valor entero de las funciones edad, fiebre y tos en sus respectivas variables de la struct.

Finalmente, tras almacenar toda la información imprimiremos en pantalla los datos guardados del usuario mediante la función display_patient e incrementaremos el número de pacientes de nuestro programa.

int p_search(struct unPaciente *elPaciente, int *numero)

```
int p_search(struct unPaciente *elPaciente,int *numero){

  char busqueda[10];
  int i;
  int comprobacion=0;

  printf("\nSearch\n\n");

  if((*numero)==0)
    printf("\nNo patients yet\n");
  else{

    do{
     get_string("\nDNI (9-9 char):",busqueda,9,9);
     for (i=0;i<(*numero);i++){
       if(strcmp(busqueda,elPaciente[i].dni)==0){
         printf("\nPatient data:\n");
         display_patient(elPaciente,&i);
         i=(*numero);
         comprobacion=1;
       }
       else
         comprobacion=0;
     }
     if (comprobacion!=1){
       printf("\nUnknown patient\n");
       comprobacion=1;
     }
    } while(comprobacion!=1);

  }
  return 0;
}
```

La función p_search recibirá por referencia los datos almacenados en la struct de losPacientes, llamada en este caso elPaciente y el número de los mismos. Su finalidad será la de buscar un paciente según su DNI y mostrar en pantalla la información del mismo.

Tras declarar las variables locales que vamos a usar en la función imprimiremos que la opción seleccionada en el menú ha sido la de Search, buscar a un paciente.

En primer lugar, comprobaremos que el número de pacientes pasado por referencia no es cero, es decir, no hay ningún usuario registrado en nuestra base de datos, en ese caso imprimiremos en pantalla “No patients yet” y volveremos al menú de inicio.

En el caso de que si haya pacientes registrados llamando a la función get_string solicitaremos al usuario el DNI que quiere buscar, este DNI será de 9 caracteres y lo copiaremos en la variable “búsqueda”. Tras recibir el DNI que el usuario quiere encontrar con el bucle for y la función strcmp compararemos el DNI (o cadena) escrita por el usuario con el dni de todos los pacientes registrados en nuestra strutc. Si la función strcmp devuelve un cero, es decir, las cadenas son iguales imprimiremos en pantalla los datos del paciente encontrado, haremos que nuestro contador “i” sea igual al número de pacientes, es decir, termine el bucle for y diremos que nuestra variable comprobación sea igual a uno.

En el caso de que la cadena de caracteres de búsqueda no coincida con ningún dni almacenado comprobación seguirá valiendo el valor de cero y por lo tanto el programa imprimirá que no hay ningún paciente con ese dni registrado en nuestra base de datos y volverá a mostrar el menú de inicio.

int p_discharge(struct unPaciente *elPaciente, int *numero)

La función p_discharge recibe por referencia los datos almacenados en nuestra struct y el número de pacientes registrados. Su finalidad, igual que en el p_search, es buscar a un paciente en la base de datos mediante el dni y en este caso eliminarlo de nuestra tabla de pacientes.

```
int p_discharge (struct unPaciente *elPaciente, int *numero){

  char busqueda[10];
  int i,j;
  int comprobacion=0;

  printf("\nDischarge\n");

  if((*numero)==0)
    printf("\nNo patients yet\n");
  else{

    do{
     get_string("\nDNI (9-9 char):",busqueda,9,9);
     for (i=0;i<(*numero);i++){

       if(strcmp(busqueda,elPaciente[i].dni)==0){
         for(j=i;j<(*numero);j++){
           elPaciente[j]=elPaciente[j+1];
         }
         printf("\nDischarged patient\n");
         i=(*numero);
         comprobacion=1;
         (*numero)--;
       }
     }
     if (comprobacion!=1){
       printf("\nUnknown patient\n");
       comprobacion=1;
     }

    } while(comprobacion!=1);

  }
  return 0;
}
```

Como en cada función en las primeras líneas declaramos las variables que vamos a usar. Imprimimos en pantalla que la opción seleccionada por el usuario en el menú de inicio ha sido “Discharge” y en primer momento comprobamos que haya pacientes guardados en nuestra struct, en el caso de que no haya ningún paciente imprimiremos en pantalla “No patients yet” y volveremos al menú de inicio.

En el caso de que si haya pacientes registrados con la función get_string solicitaremos al usuario que escriba el dni de la persona que quiere eliminar de la tabla de pacientes. En el momento que la cadena solicitada es correcta comienza nuestro primer bucle for encargado de comparar que la cadena de búsqueda recibida por usuario, con todos los dnis guardados en nuestra base de datos. En el caso de que algún dni coincida con la cadena recibida, es decir, la variable strcmp valga cero, comenzará nuestro segundo bucle for encargado de mover todos los pacientes registrados a partir del eliminado una posición hacia atrás en nuestra tabla de pacientes. En el momento de que este bucle se finalice imprimiremos “Discharged patient” para que el usuario sepa que hemos eliminado el paciente solicitado. En este momento nuestra variable comprobación valdrá uno, “i” valdrá el valor del número de pacientes para acabar el primer bucle y no seguir analizando los demás pacientes, y el número de pacientes se reducirá en una unidad. Con comprobación=1 finalizará nuestro bucle do-while y volveremos al menú de inicio.

int p_list(struct unPaciente *elPaciente, int *numero)

La función p_list recibe por referencia la tabla da datos de los pacientes y el número de los mismos, su finalidad es pedir al usuario que introduzca un año de nacimiento y mostrar todos los pacientes registrados con un año de nacimiento inferior.

```
int p_list (struct unPaciente *elPaciente, int *numero){

  short int edad_busqueda;
  int i;

  printf("\nList\n\n");

  if((*numero)==0){
    printf("No patients yet\n");
  }
  else{
    do{
      get_integer("\nDate [1900-2020]:",&edad_busqueda,1900,2020);
      printf("\nPatients born before %d:\n",edad_busqueda);
      for(i=0;i<(*numero);i++){
        if (edad_busqueda>=elPaciente[i].edad){
          display_patient(elPaciente,&i);
        }
        else {
        }
      }
    }while(edad_busqueda<1900 || edad_busqueda>2020);
  }

  return 0;
}
```

En primer lugar, imprimiremos en pantalla que la opción seleccionada por el usuario ha sido la de “List”. Tras esto comprobaremos si hay pacientes registrados en la base de datos, en el caso de que esto no sea así el programa mostrará en pantalla “No patients yet” y volverá al menú principal.

Por otra parte, si hay pacientes registrados, mediante la función get_integer le pedimos al usuario que introduzca una fecha entre 1900 y 2020, en el momento que tengamos ese año guardado en la variable edad_búsqueda imprimiremos en pantalla “Patients before (edad_busqueda)” e imprimiremos todos los pacientes con edad de nacimiento menor a la solicitada, en el caso de que no haya ninguno que cumpla esos requisitos no imprimirá nada.

void p_mark(struct unPaciente *elPaciente, int *numero)

La función p_mark recibe por referencia los datos de nuestra tabla de pacientes y el número de los mismos, su finalidad es la de marcar cual de los pacientes es positivos en covid según unos parámetros y en este caso mostrar toda su información registrada.

```
void p_mark(struct unPaciente *elPaciente,int *numero){
  int i;

  printf("\nPositives\n\n");

  if((*numero)==0){
    printf("No patients yet\n");
  }
  else{
    printf("Positive patients:\n");
    for(i=0;i<(*numero);i++){
      if(elPaciente[i].fiebre==1 && elPaciente[i].tos==1 && strcmp("N",elPaciente[i].sintoma)!=0){
        display_patient(elPaciente,&i);
      }}}
}
```

Tras inicializar la variable “i” que utilizaremos en el bucle for, imprimiremos en pantalla que la opción seleccionada por el usuario ha sido la de “Positivies”, es decir, mostrar los pacientes positivos. En primero lugar comprobaremos que haya pacientes registrados comprobando si el número de pacientes es igual a cero, en este caso mostraremos en pantalla “No patients yet” y volveremos al menú principal.

En el caso de que si haya pacientes registrados imprimiremos en pantalla “Positives patients:” y mostraremos la información de todos los pacientes que tenga fiebre, tos y cualquier otro síntoma que no sea **N**inguno. En el caso de que no haya ningún paciente registrado con síntomas se mostrará igual la cadena de “Positives patients:” y se volverá al menú principal sin mostrar nada.

int p_exit(struct unPaciente *elPaciente, int *numero)

La función p_exit recibe por referencia los datos almacenados en nuestra strcut de unPaciente y el número de pacientes de la misma, su finalidad es asegurarse de que el usuario quiere salir

```
int p_exit(struct unPaciente *elPaciente,int *numero){

  FILE *paciente;
  int i;

  printf("\nExit\n");

  if(yes_or_no("\nAre you sure you want to exit the program? (y/n) : ")==1){

    paciente=fopen("patients.txt","w");

    for(i=0;i<(*numero);i++){
      fprintf(paciente,"%s %s %d %d %d %s\n",elPaciente[i].nombre,elPaciente[i].dni,elPaciente[i].edad,
              elPaciente[i].fiebre,elPaciente[i].tos,elPaciente[i].sintoma);
    }
    fclose(paciente);
    return 1;
  }
  else
    return 0;
}
```

del programa y guardar todos los datos de los pacientes en el correspondiente archivo .txt.

Al inicio de la función declaramos la variable “i” para el posterior bucle for y la variable “FILE *pacientes” con la trabajaremos en el archivo patients.txt.

En primero lugar imprimiremos en pantalla que la opción seleccionada por el usuario ha sido la de “Exit” es decir, salir del programa. Con la función yes_or_no nos aseguramos de que el usuario realmente quiere salir del programa, es decir, esta función retorna una unidad si quiere salir o un cero si no es así. Si realmente no quiere salir volveremos al menú de inicio.

En el caso de que el usuario haya asegurado que quiere salir del programa abriremos el archivo “patients.txt” en modo de escritura, es decir, borraremos todo almacenado en el y con el bucle for imprimiremos en el los datos de todos los pacientes que en ese momento están almacenados en nuestra struct. Tras imprimir todos estos datos en el archivo cerraremos el archivo patients y retornaremos un uno y el programa se cerrará definitivamente.

Módulo de Entrada/Salida

El módulo de entrada y salida almacena las funciones secundarias del programa y son las encargadas de revisar que la información recibida por el usuario es correcta y transmitirla a las funciones principales para realizar los procesos solicitados. Su finalidad es dar robustez al programa y dar una mayor solidez al mismo.

void caratula ()

```
void caratula(){
  stripe();
  headline();
  stripe();
  printf("\n");
  return;
}
```

Esta función es la que llama a las distintas funciones que imprimen el menú de nuestro programa “Divoc_” en pantalla.

Con la función stripe imprimimos los caracteres, tanto en la primera línea como en la tercera y en el medio, con la función headline imprimimos el título y un carácter a cada lateral.

void stripe ()

```
void stripe(){

  int j;

  for(j=0;j<50;j++){
    printf("%c",CARACTER_1); }

  return ;
}
```

Esta función es la encargada de imprimir en pantalla los caracteres en la primera línea de la caratula 50 veces a través de nuestro bucle for.

void headline()

```
void headline () {

  int i,j;

  int numEspacios;
  numEspacios=((50-2-strlen(TITULO))/2);

  printf("\n%c",CARACTER_2);

  for(i=0;i<numEspacios;i++){
    printf(" ");}
  printf("%s",TITULO);

  for(j=0;j<numEspacios;j++){
    printf(" ");}
  printf("%c\n",CARACTER_2);

  return;
}
```

Esta función es la encargada de imprimir en pantalla el titulo de nuestro programa “Divoc_” un carácter a cada extremo dejando espacios entre ellos.

Con numEspacios calculamos el número de los mismos que queremos imprimir, así si aun que cambiemos el título o la longitud de nuestro programa siempre estará centrado.

Con los bucles imprimimos los espacios, los caracteres y el titulo en el centro

int yes_or_no (char *mensaje)

La función yes or no es la encargada de asegurarse si el usuario que seleccionar “yes” or “no”, en determinadas opciones. En el caso en el que la respuesta sea que “si” retornara un entero con el valor de la unidad y en el caso de que sea un “no” retornará un cero.

```
int yes_or_no (char *mensaje) {

  char lectura[256];
  char decision;
  char extra_char[256];
  int num_valores;
  int salir=0;

  do{

    printf("%s",mensaje);
    fgets(lectura,sizeof(lectura),stdin);

    if (strlen(lectura)==1){
    }
    else {
      num_valores=sscanf(lectura,"%c%s",&decision,extra_char);

      if (num_valores==1){
        if(toupper(decision)=='Y' || toupper(decision)=='N')
          salir=1;
        else
          salir=0;
      }
    }
  } while (salir!=1);

  if (toupper(decision)=='Y')
    return 1;
  else
    return 0;
}
```

En primer lugar, declaramos las variables que vamos a usar en la función e iniciamos el bucle do-while. Dentro del mismo imprimimos el mensaje recibido por referencia e invitamos al usuario a escribir un carácter que tiene que ser “yes” or “no”.

Con el fgets leemos la cadena escrita por el usuario y en primero lugar analizamos su longitud, en el caso de que esta sea igual a la unidad el bucle se reiniciará a causa de que el usuario solo ya presionado la tecla Enter.

En el caso de que haya escrito algún carácter procedemos a analizar la cadena introducida. Con la función num_valores que toma el valor de sscanf comprobamos que los caracteres introducidos en este caso solo se almacenan en la primera variable, es decir, la que nosotros necesitamos.

Si num_valores toma el valor de la unidad quiere decir que no hay ningún valor almacenado en extra_char y por lo tanto solo ha introducido un carácter que es lo deseado en esta función. Tras esto analizamos que el carácter introducido sea uno de las dos opciones permitidas (y/n).

Si el usuario introduce una de estas dos opciones la variable salir toma el valor de la unidad y se termina el bucle do-while, al salir comprobamos cual de los dos ha escrito y retornamos la unidad si es “yes” o retornamos cero si es “no”. En el caso de que el usuario no introduzca ninguno de estos valores repetimos en bucle que escriba un carácter hasta que se seleccione uno correcto.

void get_string (char *mensaje, char *cadena, int min, int max)

La función get_string recibe por referencia dos cadenas de caracteres y dos números enteros. La primera cadena es el mensaje que mostramos al usuario para que escriba la información solicitada, ya sea el dni o el nombre. Con los dos números enteros limitamos el número de caracteres máximos y mínimos de la cadena que vamos a pedir.

La finalidad de la función es devolver una cadena de caracteres entre un mínimo y un máximo, en nuestro programa el nombre del usuario a registrar y su dni.

```
void get_string (char *mensaje,char *cadena,int min,int max){

  char palabra[max];
  char extra_char[256];
  char lectura[256];
  int valores;
  int longuitud;
  longuitud=0;

  do{
    printf("%s",mensaje);

    fgets(lectura,sizeof(lectura),stdin);

    if (strlen(lectura)==1){
    }
    else{
        valores=sscanf(lectura,"%s%s",palabra,extra_char);
        if (valores==1)
          longuitud=strlen(palabra);
    }
  } while (longuitud<min || longuitud > max);

  if (min==max){
    palabra[8]=toupper(palabra[8]);
  }
  strcpy(cadena,palabra);
  return;
}
```

Como en cada función declaramos las variables que vamos a usar y empezamos nuestro bucle do-while mientras la longitud de la palabra es correcta.

En primer lugar, imprimimos en pantalla el mensaje donde solicitamos al usuario que escriba una cadena de caracteres.

Con la función fgets leemos la cadena introducida por la entrada principal, es decir, el teclado. Tras esto analizamos la longitud de la cadena introducida, en el caso de que sea la unidad será a causa de que el usuario solo ha tecleado un Enter y por lo tanto reiniciamos el bucle.

En el caso de que haya introducido más de un carácter y analizamos cuantas palabras se guardan correctamente con el valor que devuelve sscanf a la variable valores. En el caso de que sea la unidad y la longitud de la palabra esté entre el máximo y el mínimo saldremos del bucle do-while. Al salir del bucle comprobamos si el máximo y el mínimo son iguales, en este caso el usuario ha escroto un dni y haremos un toupper devolver la letra del documento de identidad en mayúscula.

Justo antes de acabar la función haremos un strcpy de la cadena y la palabra, esta ultima es la cadena correcta introducida por el usuario y la primera es la cadena que viene por referencia por lo que la copiaremos ahí y saldremos del programa.

int get_integer (char *mensaje, short int *edad, int min, int max)

La función get_integer recibe por referencia una cadena de caracteres, un short int y el valor de dos números enteros. Su función es la de solicitar al usuario un número entero entre el mínimo y máximo pasado por referencia y comprobar que no es una cadena de caracteres o algo similar. Es decir, dar robustez al programa.

```
int get_integer(char *mensaje,short int *edad,int min, int max){

  int numero;
  int valores;
  int salir=0;
  char lectura[256];
  char extra_char[256];

  do{

    printf("%s",mensaje);

    fgets(lectura,sizeof(lectura),stdin);

    if(strlen(lectura)==1){
    }
    else{
      valores=sscanf(lectura,"%d%s",&numero,extra_char);
      if (valores==1)
        salir=1;
    }
  } while (numero<min || numero > max || salir!=1);

  return(*edad=numero);
}
```

Tras inicializar las variables que vamos a utilizar imprimiremos en pantalla el mensaje recibido por referencia, en nuestro programa un año de nacimiento. Con fgets leeremos una cadena de caracteres introducida por el usuario y con el primer if analizaremos si simplemente ha pulsado Enter o no. Con el valor que retorna sscanf podemos ver si el usuario ha introducido un número y por lo tanto salir tomará el valor de la unidad.

Con el bucle do-while veremos si ese número esta entre los valores solicitados y además de asegurarnos que la lectura por pantalla ha sido la correcta. En este caso la función retornara el short int recibido por referencia que ha tomado previamente el valor del número solicitado.

int verify_dni (char *dni)

La función verify_dni recibe por referencia una cadena de caracteres, un dni, y su finalidad es la de comprobar que este dni es correcto retornando en este caso un uno.

```
int verify_dni(char *dni){

  int i;
  char indice_dni[]={'T','R','W','A','G','M','Y','F','P','D','X','B','N','J','Z','S','Q','V','H','L','C','K','E'};

  i=atoi(dni);
  i=i%23;

  if (toupper(dni[8])==toupper(indice_dni[i]))
    return 1;
  else
    return 0;
}
```

Tras inicializar la variable “i” y una cadena de caracteres con los distintos casos de la letra del dni, utilizamos la función “atoi” para convertir la cadena de caracteres recibida por referencia en un número entero. Tras esto dividimos el mismo entre 23 y con el if comprobamos que el dni sea correcto. Para que esto pase le letra almacenada en el dni tiene que coincidir con la letra en la posición que tiene la variable “i”, en el caso en el que esto pase retornamos la unidad, al contrario, retornamos un cero.

void get_character (char *mensaje, char *comprobacion, char *la_cadena)

La función get_character recibe por referencia tres cadenas de caracteres y tiene como finalidad la de pedir al usuario que teclee un carácter, comprobar que este es una de los permitidos y en este caso copiarlo en la_cadena recibida por referencia.

```
void get_character (char *mensaje,char *comparacion,char *la_cadena){

  char lectura[256];
  char caracter;
  char extra_char[256];
  int num_valores;
  int salir=0;

  do{

    printf("\n%s (%s):",mensaje,comparacion);

    fgets(lectura,sizeof(lectura),stdin);

    if (strlen(lectura)==1){
    }
    else {
      num_valores=sscanf(lectura,"%c%s",&caracter,extra_char);
      if (num_valores==1){
        caracter=toupper(caracter);
        if (strchr(comparacion,caracter)!=NULL){
          strcpy(la_cadena,&caracter);
          salir=1;
        }
      }
    }
  }while (salir!=1);

  return;
}
```

En primer lugar, tras inicializar las variables y empezar nuestro bucle do-while, imprimiremos en pantalla el mensaje recibido y las opciones que el usuario tiene.

Con fgets leemos de la entrada principal la cadena introducida por el usuario, en el caso de que su longitud sea la unidad reiniciaremos el bucle ya que el usuario solo ha pulsado la tecla Enter.

Con el valor que retorna el sscanf en la variable num_valores analizaremos si solo ha almacenado un carácter en la primera variable o no. En este caso nuestra variable entera valdrá la unidad y por lo tanto procederemos a comprobar que el carácter introducido por el usuario es válido. Tras poner en mayúscula el carácter introducido utilizáramos la función strchr para comprobar que el carácter es igual a uno de los caracteres de la cadena recibida por referencia “comprobación”. En el caso de que esta función no sea NULL copiaremos el carácter introducido en la cadena de caracteres recibida por referencia y haremos que la variable “salir”, que controla el bucle while, valga uno para poder terminarlo.

void display_patient (struct unPaciente *elPaciente, int *numero)

La función display_patient recibe por referencia la struct de la tabla de pacientes y el numero de los mismos. Su finalidad es imprimir en pantalla los datos del paciente solicitado.

Con el valor del número del paciente recibido por referencia imprimimos en una línea los siguientes datos del paciente: nombre; dni; edad; fiebre; tos; síntoma.

```
void display_patient (struct unPaciente *elPaciente,int *numero){

  int i;
  i=(*numero);

  printf(">\t%s;%s;%d;%d;%d;%s;\n",elPaciente[i].nombre,elPaciente[i].dni,
         elPaciente[i].edad,elPaciente[i].fiebre,elPaciente[i].tos,elPaciente[i].sintoma);
}
```

void archivo_pacientes (struct unPaciente *elPaciente, int *numero)

La función archivo_pacientes recibe por referencia la struct con nuestros pacientes y el numero de los mismos, esta función es llamada nada más iniciar el programa y es la encargada de leer los pacientes guardados en el archivo patients.txt.

```
void archivo_pacientes (struct unPaciente *elPaciente, int *numero){


  char cadena[256];

  FILE *pacientes;

  if ((pacientes=fopen("patients.txt","r"))!=NULL){
    while((fgets(cadena,sizeof(cadena),pacientes))!=NULL){
      sscanf(cadena,"%s %s %hd %d %d %c",elPaciente[*numero].nombre,elPaciente[*numero].dni
             ,&elPaciente[*numero].edad,&elPaciente[*numero].fiebre,&elPaciente[*numero].tos,elPaciente[*numero].sintoma);
      (*numero)++;
    }

    fclose(pacientes);
    return;
  }
}
```

Declaramos una cadena de caracteres y nuestro archivo FILE para acceder al mismo. En primer momento comprobamos que el archivo no es NULL, es decir, que existe y hay algo escrito en él, en el caso de que esto no sea así la función terminará y no añadirá ningún dato en nuestra tabla de pacientes.

En el caso de que si haya datos escritos en el archivo patients.txt abrimos el archivo en forma de lectura y mientras la cadena no sea nula, es decir, hasta que finalice la lectura vamos almacenando con un sscanf los datos escritos en fila en nuestra struct y va sumando una unidad la variable del número de pacientes por cada línea correctamente leída. Al acabar de leer todos los datos cerramos el archivo y volvemos al programa.