```
#include "part2.h"

/*  Considers all appointments in the array, builds and returns a linkedlist
    including all appointment requests in the order of their time.  */
node_t* build_ll(Appointment_t appointmens[], int size, const Working_hours_t*
hours)
{
    node_t* linked_list;
    node_t* current;
    node_t* head=NULL;
    char same[1]={'\0'};
    int i,k=0,j;
    int work_hour[24];
    int hour;

    if(DEBUG)
    {
        for(i=0;i<size;++i)
        {
            printf("%d,",appointmens[i].app_id);
            printf("%d,",appointmens[i].patient_id);
            printf("%d\n",appointmens[i].hour);
        }
    }

    for(i=0;i<size;++i)
    {
        if(appointmens[i].hour>=(hours->start)  && appointmens[i].hour<=(hours->end))
        {
            hour=EMPTY_HOUR;

            work_hour[k]=appointmens[i].hour;

            for(j=0;j<k;++j)
            {
                if(work_hour[j]==work_hour[k])
                {
                    hour=REPLETE_HOUR;
                }
            }
            ++k;
        }

        if(hour==EMPTY_HOUR)
        {
            if(appointmens[i].hour>=(hours->start)  && appointmens[i].hour<=(hours->end))
            {
                linked_list=(node_t*)malloc(sizeof(node_t));
                linked_list->app_id=appointmens[i].app_id;
                linked_list->patient_id=appointmens[i].patient_id;
                linked_list->hour=appointmens[i].hour;
                strcpy(linked_list->name,same);
                linked_list->history=NULL;

                if(head==NULL)
                {
                    head=linked_list;
                    current=linked_list;
                }
                else
                {
                    current->next=linked_list;
                    current=linked_list;
                }
            }
        }
    }

    return head;
}

/*  Write all appointments in the list into the accepted appointment file in
```

```c
    the described format.   */
void write_accepted_app(node_t* head, const Files_t* files)
{
    node_t* temp_head;
    int i=1,no=1,null_check=1;

    FILE *Appointments_csv_output_file;

    Appointments_csv_output_file=fopen(files->accepted_appo_file_n,"w");

    if(Appointments_csv_output_file == NULL){

        printf("ERROR!! Appointments csv file could not be opened to write.\n");
        exit(1);
    }


    fprintf(Appointments_csv_output_file,"no;id;patient_id;name;history;hour\n");

    if(head==NULL)
    {
        null_check=0;
    }

    if(null_check)
    {
        temp_head=head;

        while(i==1)
        {
            fprintf(Appointments_csv_output_file,"%d;",no);
            fprintf(Appointments_csv_output_file,"%d;",temp_head->app_id);
            fprintf(Appointments_csv_output_file,"%d;",temp_head->patient_id);
            fprintf(Appointments_csv_output_file,"%s;",temp_head->name);
            fprintf(Appointments_csv_output_file,"%s;",temp_head->history);
            fprintf(Appointments_csv_output_file,"%d\n",temp_head->hour);

            if(temp_head->next!=NULL)
            {
                temp_head=temp_head->next;
            }
            else
            {
                i=0;
            }
            ++no;
        }
    }

    fclose(Appointments_csv_output_file);

}

/*  Takes personal data from the patients file and adds the corresponding
    name and history information to each appointment.  */
void add_personal_data(node_t* head, const Files_t* files)
{
    FILE *Patients_input_file;
    node_t* list=head;
    node_t* temp_linked=head;
    char string[1000];
    char string2[1000];
    char symbol;
    char symbol_less_than='<';
    char symbol_more_than='>';
    char more_symbol;
    char array[10000];
    int found=1;
    char *id_id="ID";
    char *records="/Records";
    int equal_id,equal_records,size=0,i;
    char * pch;
    char * pch2;
    char * pch3;
    char * history_check="/History>";
    char * space=" ";
    int id;
    int a;
    int check;
    int one_word;


    while(list!=NULL)
    {
        list=list->next;
        ++size;
    }

    list=head;

    do
    {

        Patients_input_file=fopen(files->patients_file_n,"r");

        if(Patients_input_file == NULL)
        {
            printf("ERROR!! Patients file could not be opened to read.\n");
            exit(1);
        }

        do
        {
            fscanf(Patients_input_file," %c",&symbol);

            if(symbol_less_than==symbol)
            {
                fscanf(Patients_input_file,"%s",string);
                pch = strtok (string,">");

                equal_id=strcmp(pch,id_id);
                equal_records=strcmp(pch,records);

                if(equal_records==0)
                {
                    found=0;

                }

                if(equal_id==0)
                {
                    pch = strtok (NULL,"<");
                    id=atoi(pch);

                    if(list->patient_id==id)
                    {

                        /*  Add the corresponding name */
                        do
                        {
                            fscanf(Patients_input_file," %c",&symbol);

                            if(symbol_less_than==symbol)
                            {
                                fscanf(Patients_input_file,"%s",string);
                                pch = strtok (string,">");
                                pch = strtok (NULL,"<");
                                fscanf(Patients_input_file,"%s",string2);
                                pch2 = strtok (string2,"<");
                                strcat(pch,space);
                                strcat(pch,pch2);
                                strcpy(list->name,pch);
                            }
                        }while(symbol_less_than!=symbol);
                        /***********************************/


                        /*  Add the corresponding history */
                        do
                        {
                            fscanf(Patients_input_file," %c",&symbol);

                            if(symbol_less_than==symbol)
                            {
                                fscanf(Patients_input_file,"%s",string);
                                pch = strtok (string,">");
                                pch = strtok (NULL,"<");
                                strcpy(pch2,pch);
                                check=strcmp(pch2,history_check);

                                if(check!=0)
                                {
                                    do
                                    {
                                        fscanf(Patients_input_file,"%c",&more_symbol);
                                        array[0]=more_symbol;

                                        if(more_symbol=='\n')
                                        {
                                            more_symbol=symbol_less_than;
                                        }

                                        else if(more_symbol!='<')
                                        {
                                            strcat(pch2,&array[0]);
                                        }


                                    }while(symbol_less_than!
=more_symbol);


                                    a=strlen(pch2);

                                    list->history=(char*)malloc(a*sizeof(char)+1);
                                    strcpy(list->history, pch2);
                                }

                                else
                                {
                                    a=strlen(space);
                                    list->history=(char*)malloc(a*sizeof(char)+1);
                                    strcpy(list->history, space);
                                }

                            }
                        }while(symbol_less_than!=symbol);
                        /***********************************/

                    }


                }


            }



        }while(found);

        list=list->next;
        ++i;
        found=1;

        fclose(Patients_input_file);
```

```c
    }while(i<size);

    if(DEBUG)
    {
        for(i=0;i<size;++i)
        {
            printf("**********************\n");
            printf("App_id:%d\n",temp_linked->app_id);
            printf("Patient:%d\n",temp_linked->patient_id);
            printf("Name:%s\n",temp_linked->name);
            printf("History:%s\n",temp_linked->history);
            printf("Hour:%d\n",temp_linked->hour);

            temp_linked=temp_linked->next;
        }
    }
}

/*  Deletes all records in the delete file from the linked list
    and returns the number of appointments deleted. */
int delete_appointments(node_t** head, const Files_t* files)
{
    int number_of_appointments_deleted=0,target,delete_id,status,found=0,i;
    node_t* list,*curr;
    FILE* delete_input_file;
    int check=0,check_head=0,size=0;

    list=*head;

    while(list!=NULL)
    {
        ++size;
        list=list->next;
    }

    list=*head;
    curr=*head;

    while(list!=NULL)
    {
        target=list->app_id;



        delete_input_file=fopen(files->delete_file_n,"r");

        if(delete_input_file == NULL)
        {
            printf("ERROR!! Delete file could not be opened to read.\n");
            exit(1);
        }

        status=fscanf(delete_input_file,"%d",&delete_id);


        while(status!=EOF)
        {

            if(delete_id==target)
            {
                if(check_head==check)
                {
                    curr=(*head)->next;
                    *head=(*head)->next;
                    free(list);
                    ++check;
                    ++number_of_appointments_deleted;
                }

                else
                {
                    curr->next=list->next;
                    list->next=NULL;
                    free(list);
                    found=1;
                    ++number_of_appointments_deleted;
                }
            }


            status=fscanf(delete_input_file,"%d",&delete_id);
        }


        fclose(delete_input_file);

        ++check_head;
        if(check_head==check)
        {
            list=*head;
            curr=*head;

        }

        else
        {
            if(size==number_of_appointments_deleted)
            {
                free(*head);
                *head=NULL;
                list=NULL;
            }

            else
            {
                if(!found)
                {
                    curr=list;
                }
                found=0;
                list=curr->next;
            }
        }

    }

    if(head==NULL)
    {
        printf("head Null\n");
    }

    if(DEBUG)
    {
        printf("%d appointments delete from the link list\n",number_of_appointments_deleted);
    }

    if(size==number_of_appointments_deleted)
    {
        *head=NULL;
    }

    return number_of_appointments_deleted;
}

/*  Frees all dynamically allocated memory in the list. */
void free_list(node_t* head)
{

    while(head!=NULL)
    {
        free(head->history);
        free(head);
        head=head->next;
    }

}
/*##########################################################################*/
/*                            End of HW10_part2.c                           */
/*##########################################################################*/
```