```c
#ifndef COMMANDS_H
#define COMMANDS_H
#include "status.h"
#include <stdio.h>
#include "new_acc.h"
#include "pending.h"
#include <string.h>
#include "friends.h"
void remove_newline(char * str)//really needed
{

	int i;
	for(i=0; i<strlen(str); i++)
	{
	if(str[i] == '\n')
		str[i] = '\0';
	}
}

void print_sep();

//==============================profile=====================================


void profile(Account * acc_head, Profile ** prof_Array, Profile * User , int
arr_len)
{

	printf("Your status:%s", User->status);
	putchar('\n');
	printf("Your friends status:\n");
	print_friends_status(prof_Array, User, arr_len) ;
	return;

}
//============================checkRequest================
void checkRequests(Profile * User, Profile ** prof_Array, int arr_len)
{
	char instruction[65];
	Profile * helper;
	printf("Dear %s, you have %d friend requests\n",User->username,
num_of_nodes(User->RequestHead) );
		printf("The following people are requesting you add them as
friends:\n");
		print_requests(User->RequestHead);
	Input: printf("\nTo approve a request enter: ''approve::<username>''\n");
		printf("To refuse and delete a request enter:
''refuse::<username>''\n");
		printf("To go back to the main selection menu etner: '&'\n");
		printf("Input:");

	scanf ("%[^\n]%*c", instruction);//Fix exception


	if(  strstr(instruction,"approve::")==instruction  )
	{

		if(is_user_in_requests(User->RequestHead, instruction+9))
```

```
                {
                helper = find_profile(instruction+9, prof_Array, arr_len) ;
                remove_request(&(User->RequestHead), instruction+9);
                add_friend(instruction+9,&(User->friends));
                add_friend(User->username,&(helper->friends));
                printf("%s was successfully add as a friend \n", instruction+9);
                return;
                }

                else
                {
                        printf("\nThere is no such user in the list\n");
                        goto Input;
                }
        }
        else if(  strstr(instruction,"refuse::")==instruction  )
        {
                if(is_user_in_requests(User->RequestHead, instruction+8))
                {
                remove_request(&(User->RequestHead), instruction+8);
                printf("%s was successfully removed from your pending list. \n",
instruction+8);
                return;
                }

                else
                {
                        printf("\nThere is no such user in the list\n");
                        goto Input;
                }

        }
        else if(strstr(instruction,"&")==instruction  )
        {
                putchar('\n');
                return;
        }

        else
        {
                printf("\nInvalid input\n");
                goto Input;
        }


}
//==========================request==============
void request(Profile ** prof_Array, char*dest, int arr_len ,Profile * user)
{
        {
        Profile * target;

        if(strcmp(dest, user->username)==0)
        {
                printf("\nYou can't send request to yourself.");
                return;
        }
```

```
	target = find_profile(dest, prof_Array, arr_len);
	if(target == NULL)
	{
		printf("\nNo such user");
		return;
	}

	else if(is_user_in_requests(target->RequestHead, user->username))
	{
		printf("You already sent a request to this user");
		return;
	}

	else if(is_in_friend(dest, user->friends))
	{
		printf("This user is already your friend");
		return;
	}
	else
	{
		add_request(&(target->RequestHead), user->username);
		printf("The request was succesfuly sent, waiting for the user to
accept you.");
	}




}
}
//=======================status=============================================


void status(Account * acc_head, Profile ** prof_Array, Profile * User)
{
	char option;
	char status[514];
	printf("Update your SocioPath status to share with your friends\n");
	Input:printf("Input:");
	fflush(stdin);
	fgets(status, 514, stdin);
	remove_newline(status);
	if(strlen(status) == 513)
	{
		printf("This length of this status exceeds the possible length,\nDo
you wish to update your status with first 512 characters?\ntype 'y' for yes 'n'
for no\nInput:)");
		Input2:option = getchar();
		switch (option)
		{
		case 'y':
			status[513] = '\0';
			change_status(status, User);
			printf("The status was successfully changed\n");
			fflush(stdin);
			return;
		case 'n':
			printf("Please pick other status.\n");
```

```
                goto Input;
            default:
                printf("Invalid input, type y/n:\nInput:");
                fflush(stdin);
                goto Input2;

            }


        }


        change_status(status, User);
        printf("The status was successfully changed\n");
        return;

}


//======================unfriend====================

void unfriend(char*friend_n, char** friends_of_current_user, Profile **
prof_Array, int arr_len, Profile * User)
{
        Profile * helper;
        if(is_in_friend(friend_n, *friends_of_current_user))
        {
            helper = find_profile(friend_n, prof_Array, arr_len);
            remove_friend(friend_n, friends_of_current_user);
            remove_friend(User->username, &(helper->friends));
            printf("%s was removed from your friend list", friend_n);
        }

        else
            printf("%s is not in your friend list", friend_n);


}

//============================checkStatus=======================================
==

void checkStatus(Account * acc_head, Profile ** prof_Array, Profile * User, char *
friend_n, int arr_len )
{
        Profile * friend_prof;
        if(is_in_friend(friend_n, User->friends))
            {
                friend_prof = find_profile(friend_n, prof_Array, arr_len);
                printf("%s Status:%s\n",friend_n,  friend_prof->status);
                return;

            }

            printf("Friend was not found\n");
            return;

}
```

```
//==================================printFriends====================
int string_comp( char * c1, char * c2)
	//lexicographical comaprison between 2 strings: if c1>c2 return1; else
return 0.
{

	int c, len1, len2, i, Flag;


		if(c1[0]!='\0' && c2[0]!='\0')
		{
		len1 = strlen(c1);
		len2 = strlen(c2);
			c=0;

		 if(len1>len2)
			{
				len1 = len2;
				c =1;
			}

			i=0;
			Flag=1;
			while(Flag)
			{
				if(i==len1)
					Flag = 0;

				else if(chk_if_cletter(c1[i]) && chk_if_sletter(c2[i]))
				{
					Flag = 0;
					c = 1;
				}

				else if(chk_if_cletter(c2[i]) && chk_if_sletter(c1[i]))
				{
					Flag = 0;
					c = 0;
				}

				else if(c1[i]>c2[i])
				{
					Flag = 0;
					c = 1;
				}

				else if(c2[i]>c1[i])
				{
					Flag = 0;
					c = 0;
				}



				i++;
			}
		}
```

```
		else
			c = 1;


		return c;


}



void lex_print(char * friends)//works
{
	int Flag, length, i, k;
	char last[51],min[51], helper[51]; ;
	Flag = 1;
	length = strlen(friends);

	if(friends[0]!='\0');
	{
		last[0] = '\0';

		while(Flag)
		{
		Flag = 0;
		min[0] = '\0';
		k=0;
			for(i=0;i<=length;i++)
			{
				if( friends[i] =='\0' || friends[i] ==';')
				{
					helper[k] = '\0';
					if(string_comp(min, helper))
						if(string_comp(helper, last))
						{
							Flag = 1;
							strcpy(min, helper);
						}

						k=0;
				}

				else
				{
					helper[k] = friends[i];
					k++;
				}

			}
			if(min[0]!='\0')
			{
			printf("%s\n", min);
			strcpy(last, min);
			}

		}
	}
```

```
}



void printFriends(Profile * User)
{
	if((User->friends)[0] =='\0')
	{
		printf("You don't have any friends\n");
		return;
	}

	else
	{
		printf("Friends:\n");
		lex_print(User->friends);
		return ;

	}

}

//======================print_social_netowrk=========================

char* create_list(char*friends, int * status, int arr_len, Profile ** prof_Array)
{

char helper[52];
char * new_list=NULL;
int i=0, k=0, length=0, n=0;

new_list = (char*)malloc(1);

	if(new_list==NULL)
	{
		printf("Error");
			return NULL;
	}

new_list[0] = '\0';
length = strlen(friends);

for(i=0;i<=length;i++)
{
	if(friends[i] =='\0')
	{
		helper[k] = '\0';
		n=find_profile_index(helper, prof_Array, arr_len);
		if(status[n]==0)
		{
			status[n] = 1;
			new_list = (char*)realloc(new_list,  strlen(new_list) +
strlen(helper) + 1);

				if(new_list==NULL)
				{
				printf("Error");
				return NULL;
```

```
                        }

                strcat(new_list, helper);//assume there is null terminator
                }

                else
                {
                        if(new_list[strlen(new_list)-1]==';')
                                new_list[strlen(new_list)-1]='\0';
                }

        }



        else if(friends[i] ==';')
        {

                helper[k] = '\0';
                n=find_profile_index(helper, prof_Array, arr_len);
                helper[k] = ';';
                helper[k+1] = '\0';
                if(status[n]==0)
                {
                        status[n] = 1;
                        new_list = (char*)realloc(new_list,  strlen(new_list) +
strlen(helper) + 1);
                        strcat(new_list, helper);
                }
                k=0;
        }

        else
        {
                helper[k] = friends[i];
                k++;
        }

}

return new_list;
}


char * fof(char*friends, int * status, int arr_len, Profile ** prof_Array)//fof =
friends of friends
{

        Profile * friend_p=NULL;
        char helper[51];
        int length=0, i=0, k=0;

        char * sum=NULL;//needs malloc
        char * new_list=NULL;

        sum = (char*)malloc(1);

                if(sum==NULL)
```

```
		{
		printf("Error");
		return NULL;
		}

	sum[0]  = '\0';



	length = strlen(friends);
	for(i=0; i<=length; i++)
	{


		if(friends[i] =='\0')
		{
			helper[k]= '\0';
			friend_p = find_profile(helper, prof_Array, arr_len);
			new_list = create_list(friend_p->friends, status, arr_len,
prof_Array);
			sum = (char*)realloc(sum, strlen(sum) + strlen(new_list) + 1);
			strcat(sum, new_list);
			free(new_list);
			k=0;
		}

		else if(friends[i] ==';')
		{
			helper[k]= '\0';
			friend_p = find_profile(helper, prof_Array, arr_len);
			new_list = create_list(friend_p->friends, status, arr_len,
prof_Array);
			if(sum[0]!='\0' && new_list[0] !='\0')
			{
				sum = (char*)realloc(sum, strlen(sum) + 1 + 1);
				strcat(sum, ";");
			}
			sum = (char*)realloc(sum, strlen(sum) + strlen(new_list) + 1);
			strcat(sum, new_list);
			free(new_list);
			k=0;
		}

		else
		{
			helper[k] = friends[i];
			k++;
		}
	}

	return sum;

}
```

```
void print_string_properly(char * string)//print strings but if the char is ';' it
replaces it with ", "
{

	int n, i, k;
	n = strlen(string);
	k=0;
	for(i=0; i<n ;i++)
	{
		if(string[i] == ';')
		{
			if(k==5)
			{
			putchar('\n');
			k=0;
			}
			else
			{
			printf(", ");
			k++;
			}
		}
		else
			putchar(string[i]);
	}
}




void printNetowrk(Profile** prof_Array, int arr_len, Profile * user)
{	int * status;//saves which user was already printed


	int i=0, cycle=0;
	char *str1;
	char *str2;

	str1=NULL;
	str2=NULL;


		status = (int*)malloc(sizeof(int)*arr_len);

		if(status==NULL)
		{
		printf("Error");
		return;
		}



	for(i=0; i<arr_len; i++)
	{
		status[i]=0;// 0 not printed, 1 printed

	}
```

```
        printf("Dear %s, your social netowrk:\n", user->username);
        printf("You: %s\n", user->username);
        status[find_profile_index(user->username ,prof_Array, arr_len)]  = 1;

        str1 = fof(user->username, status, arr_len, prof_Array);//last

        if(str1[0] == '\0')
        return;

        printf("Your friends:" );//1
        print_string_properly(str1);//2

        str2 = str1;
        str1 = fof(str2, status, arr_len, prof_Array); //4
        free(str2);

        putchar('\n');//6

        if(str1[0] == '\0')
        return;

        printf("Friends of friends:" );//1
        print_string_properly(str1);//2

        str2 = str1;
        str1 = fof(str2, status, arr_len, prof_Array); //4
        free(str2);

        putchar('\n');//6

        cycle = 3;

        while(str1[0]!='\0')
        {
        printf("Circle %d of friends:", cycle);//1
        print_string_properly(str1);//2

        str2 = str1;
        str1 = fof(str2, status, arr_len, prof_Array); //4
        free(str2);

        putchar('\n');//6
        cycle++;
        }

        free(str1);
        free(status);

}


//============================search=====================
char * ci_strstr( char *arg1, char *arg2)//case insensitive strstr
{
   const char *a, *b;

   for(;*arg1;*arg1++) {
```

```
      a = arg1;
      b = arg2;

      while((*a++ | 32) == (*b++ | 32))
        if(!*b)
          return (arg1);

    }

    return(NULL);
}

void search(Profile ** prof_Array, int arr_len, char * query)
{
       int i;
       printf("The system found the following users for the query: \"%s\"\n",
query);

       for(i=0; i<arr_len; i++)
       {
              if(ci_strstr(prof_Array[i]->username, query)!=NULL)
                     printf("-%s\n", prof_Array[i]->username);
       }
}



//============================read_command==============

void read_commands(Account * acc_head, Profile ** prof_Array, Profile * User, char
* command, int arr_len)
{
       if(strcmp(command,"$")==0)
              get_out(acc_head, prof_Array, arr_len);

       else if(  strcmp(command,"profile")==0  )
       {
              print_sep();
              profile(acc_head, prof_Array, User, arr_len);
              print_sep();
                     return;
       }

       else if(  strcmp(command,"status")==0  )
       {
              print_sep();
              status(acc_head, prof_Array, User);
              print_sep();
              profile(acc_head, prof_Array, User, arr_len);
                     return;
       }

       else if(  strstr(command,"checkStatus ")==command  )
       {
              print_sep();
              checkStatus(acc_head, prof_Array, User, command+12, arr_len);
                     print_sep();
                     return;
```

```
}
else if(  strcmp(command,"printFriends")==0  )
{
	print_sep();
	printFriends(User);
		print_sep();
		return;

}
else if(  strcmp(command,"printNetwork")==0  )
{
	print_sep();
	printNetowrk(prof_Array,arr_len, User);
		print_sep();
		return;

}
else if(  strcmp(command,"checkRequests")==0  )
{
	print_sep();
	checkRequests(User, prof_Array, arr_len);
	print_sep();
		return;

}


else if(  strstr(command,"request ")==command  )
{
	print_sep();
	request(prof_Array,command+8, arr_len, User);
		print_sep();
		return;

}

else if(  strstr(command,"unfriend ")==command  )
{

	unfriend(command+9, &(User->friends), prof_Array, arr_len, User);
		print_sep();
		return;

}
else if(  strstr(command,"search ")==command  )
{
	print_sep();
	search(prof_Array, arr_len, command+7);
		print_sep();
		return;

}


else
{
	putchar('\n');
```

```
                printf("Unknown command\n");
                print_sep();
                return;
        }
}



#endif
```

```
#ifndef DEF_H
#define DEF_H
#include <stdio.h>


typedef struct acc {
        char Username[51];
        int enc_Pass;
        int randomNum;
        char answer[200];
        struct acc * next;
} Account;


#endif
```

```
#ifndef DEF_2_H
#define DEF_2_H

#include <stdio.h>
#include <stdlib.h>

typedef struct StructProfile {

char * username;
char * status;
char * friends;
struct StructProfile * RequestHead;
}Profile;

#endif
```

```
#ifndef ENC
#define ENC

#include <stdio.h>
char AND_chars(char* pass)
{
        if(*(pass+1)=='\0')
```

```
                return *pass;
        return *pass  & AND_chars( pass+1);
}

int encryption(char * pass, int randomNum)
{
        int temp;
        temp = AND_chars(pass);
        if(temp%2==0)
                temp = temp<<4;
        else
                temp = temp>>6;
        temp  = temp ^ randomNum;
        return temp;
}

#endif
```

```
#ifndef EXIT_PROGRAM_H
#define EXIT_PROGRAM_H


#include "definitions.h"
#include <stdlib.h>
#include "write_files.h"
#include "write_files_2.h"

//NEED TO PROFILES TOO
void print_help()
{
}

void free_user_db(Account * head)
{

        if(head==NULL)
        return ;

        free_user_db(head->next);
                free(head);

}

void free_profile(Profile * prof)//belong to exit_program
{


        if(prof==NULL)
        return;
        else
        {

                free_profile(prof->RequestHead);
```

```
		free(prof->friends);
		free(prof->status);
		free(prof->username);
		free(prof);

	}
}

void free_prof_array(Profile ** Array, int length)//main function for releasing
memory of profiles belong to exit_program
{
	int i;

	for(i=0; i<length; i++)
	{
		free_profile(Array[i]);
	}

}

void get_out(Account * acc_head, Profile ** prof_Array, int arr_len)//Profile **
prof_array
{

	FILE * fp;


	fp = fopen("validation.txt", "w");
	if (fp==NULL)
	{
		printf("Error");
		return ;
	}
	write_validation(acc_head, fp);
	fclose(fp);



	fp = fopen("profile.txt", "w");
	if (fp==NULL)
	{
		printf("Error");
			return;
	}

	write_prof(prof_Array, fp, arr_len);
	fclose(fp);

	free_prof_array(prof_Array,arr_len);
	free_user_db(acc_head);
	free(prof_Array);
	printf("\nThank you for using SocioPath, Good bye!\n");



	exit(EXIT_SUCCESS);
}
#endif
```

```
#ifndef FORGOT_P
#define FORGOT_P



#include "definitions.h"
#include <stdio.h>
#include <string.h>
#include "encoding.h"

int compare_ans(Account * user, char * ans)
{
	if(strcmp(user->answer, ans)==0)
		return 1;
	else
		return 0;
}

void change_pass(Account * user, char * new_pass)
{
	int new_enc;
	new_enc = encryption(new_pass, user->randomNum);
	user->enc_Pass = new_enc;
}
#endif
```

```
#ifndef FRIENDS_H
#define FRIENDS_H
#include <string.h>
#include "new_acc.h"
#include <stdlib.h>

int is_in_friend(char * user, char * friends)
{
	int i, k, len, Flag, outcome;
	char helper[51];

	len=strlen(friends);
	Flag = 0;//1 means done

	outcome = 0;
	i=0;
	k=0;
	while(i<=len && !Flag)
	{
		if (friends[i]=='\0')
		{
			helper[k]='\0';
			if(strcmp(helper, user)==0)
			{
				outcome=1;
				Flag = 1;
			}

		}

		else if(friends[i]==';')
		{
```

```
                        helper[k]='\0';
                        if(strcmp(helper, user)==0)
                        {
                                outcome=1;
                                Flag = 1;
                        }

                        k=0;
                }

                else
                {
                        helper[k]=friends[i];
                        k++;
                }


        i++;

        }
        return outcome;

}

void remove_friend(char *user, char**friends)
{
        char * helper;
        char* holder;
        int length, i, k;

        helper = strstr(*friends, user);
        holder = *friends;
        k=0;
        length = strlen(user);
        for(i=0; i<length; i++)
        {
                helper[i] = '0';
                k++;
        }

        if(helper[i]==';')
        {
                helper[i] = '0';
                k++;
        }
        else if(&(helper[i-k-1])>=holder)
        {
                helper[i-k-1]='0';
                k++;
        }

        *friends = (char*)malloc(strlen(holder)+1-k);

        if(*friends==NULL)
                {
                printf("Error");
                return;
```

```
                }


        length = strlen(holder);

        k=0;
        for(i=0; i<=length; i++)
        {
                if(holder[i]!='0')
                {
                        (*friends)[k] = holder[i];
                        k++;
                }
        }

        free(holder);


}

void add_friend(char *user, char**friends)


{
        if(*friends[0] == '\0')//if list is empty
        {
                *friends = (char*)realloc(*friends,
strlen(*friends)+strlen(user)+1);

                strcat(*friends, user);
        }

        else
        {
        *friends = (char*)realloc(*friends, strlen(*friends)+strlen(user)+2);
        strcat(*friends, ";");
        strcat(*friends, user);
        }

}



#endif
```

```c
#ifndef LOGIN
#define LOGIN


#include "definitions.h"
#include <string.h>
#include "read_files.h"
#include "encoding.h"

char * separation(char * string)
{
	char * helper;
	helper = strstr( string, "::");
		if (helper ==NULL)
		{
			return NULL;
		}
		return helper;
}


Account * find_username( char * user, Account * head)
	//Found-return account*
	//Not found-return NULL
{
	if (head==NULL)
		return NULL;
	if(strcmp(head->Username, user)==0)//Is it right?
		return head;
	else
		find_username(user, head->next);
}

int compare_password(Account * node, char * pass)
{
	return node->enc_Pass == encryption(pass, node->randomNum);
}
#endif
```

```c
#ifndef NEW_ACC
#define NEW_ACC
#include "definitions.h"
#include <string.h>
#include <stdlib.h>


int chk_if_sletter(char c)
{
//	int i;
//for(i=(int)('a'); i<=(int)('z'); i++)
	//if (c==(char)(i))
	//	return 1;


return (c>96 && c<123);
}
```

```
int chk_if_cletter(char c)
{
//     int i;
//for(i=(int)('A'); i<=(int)('Z'); i++)
       //if (c==(char)(i))
       //     return 1;
return (c>64 && c<91);
}

int chk_if_digit(char c)
{
       //int i;
//for(i=(int)('0'); i<=(int)('9'); i++)
       //if (c==(char)(i))
       //     return 1;

return (c>47 && c<58);

}

int user_vaild_rec(char * user)
{
       if(*(user+1)=='\0')
              return (chk_if_sletter(*user) || chk_if_cletter(*user) || *user==' 
');

              return (chk_if_sletter(*user) || chk_if_cletter(*user) || *user==' 
') && (user_vaild_rec(user+1));


}

int is_username_valid(char * user)
{
       if(*user=='\0')
              return 0;



       return user_vaild_rec(user);
}

int pass_vaild_rec(char *pass)
{
       if (*pass == '\0')
              return 8;
       else if(chk_if_sletter(*pass))
              return 1 |  pass_vaild_rec(pass+1);
       else if(chk_if_cletter(*pass))
              return 2 |  pass_vaild_rec(pass+1);
       else if(chk_if_digit(*pass))
              return 4 |  pass_vaild_rec(pass+1);
       else
              return 0;
}
```

```
int answer_valid_rec(char*ans)
{
		if(*(ans+1)=='\0')
		return (chk_if_sletter(*ans) || chk_if_cletter(*ans) || *ans==' ' ||
chk_if_digit(*ans));

		return (chk_if_sletter(*ans) || chk_if_cletter(*ans) || *ans==' ' ||
chk_if_digit(*ans)) && (user_vaild_rec(ans+1));
}

int is_ans_valid(char * ans)
{
	if(*ans=='\0')
		return 0;



	return answer_valid_rec(ans);
}




int pass_valid(char *pass)//check
{
	if(pass_vaild_rec(pass)==15)
		return 1;
	else
		return 0;
}







void add_account(Account ** head, Account * new_one)
{

	Account * helper;
	if(new_one==NULL)
		return;

	if(*head==NULL)
	{
		*head = new_one;
		return ;
	}

	helper = *head;

	while(helper->next!=NULL)
		helper = helper->next;

	helper->next = new_one;
```

```
}


Account * new_account_node(char * username,  char *  password, char * Answer)
{

	Account * new_acc;
	int RanNum;

	new_acc = (Account*)malloc(sizeof(Account));

	if(new_acc==NULL)
		{
		printf("Error");
		return;
		}


	strcpy(new_acc->Username, username);//Part
	RanNum  = rand();//
	//printf("%d", RanNum);
	new_acc->enc_Pass = encryption(password, RanNum);//
	new_acc->randomNum = RanNum;//
	strcpy(new_acc->answer, Answer);//
	new_acc->next=NULL;//


	return new_acc;
}


#endif
```

```
#ifndef PENDING_H
#define PENDING_H
#include <stdlib.h>
#include <string.h>
#include "definitions_2.h"

void remove_request(Profile ** reqHead, char * user)
{

	Profile * helper;

	while(*reqHead!=NULL)
	{
		if(strcmp((*reqHead)->username, user)==0)
		{
			free((*reqHead)->username);
			free((*reqHead)->status);
			free((*reqHead)->friends);
			helper = (*reqHead)->RequestHead;
			free(*reqHead);
			*reqHead = helper;
			break;
		}
```

```
                reqHead = &((*reqHead)->RequestHead);
        }

}


void add_request(Profile ** reqHead, char * user)
{
        Profile * helper;

        while(*reqHead!=NULL)
        {
                reqHead = &((*reqHead)->RequestHead);
        }

        helper = (Profile*)malloc(sizeof(Profile));

                if(helper==NULL)
                {
                printf("Error");
                return;
                }


        helper->friends = (char*)malloc(1);

                if(helper->friends==NULL)
                {
                printf("Error");
                return;
                }


        helper->friends[0] = '\0';
        helper->status = (char*)malloc(1);

        if(helper->status==NULL)
                {
                printf("Error");
                return;
                }

        helper->status[0] = '\0';
        helper->RequestHead=NULL;
        helper->username = (char*)malloc(strlen(user)+1);

        if(helper->username==NULL)
                {
                printf("Error");
                return;
                }

        strcpy(helper->username, user);
        *reqHead = helper;

}

void print_requests(Profile *reqHead)
```

```
{
	if(reqHead==NULL)
		return;
	else
	{
		printf("-%s\n", reqHead->username);
		print_requests(reqHead->RequestHead);
	}
}

int num_of_nodes(Profile * reqHead)
{
	if(reqHead==NULL)
		return 0;
	return 1 + num_of_nodes(reqHead->RequestHead);
}

int is_user_in_requests(Profile * reqHead, char * user)
{
	if(reqHead==NULL)
		return 0 ;
	if(strcmp(reqHead->username, user)==0)
		return 1;
	return is_user_in_requests(reqHead->RequestHead, user);

}
#endif
```

```
#ifndef PROFILES_H
#define PROFILES_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "definitions_2.h"

void print_pending(Profile * reqHead)//belong to pending
{

	if (reqHead == NULL)
	return;

	else
	{
		printf("%s\n", reqHead->username);
		print_pending(reqHead->RequestHead);
	}

}



Profile * find_profile(char * username, Profile ** prof_Array,int arr_len)
{
int i;

for(i=0;i<arr_len;i++)
{
	if(strcmp(prof_Array[i]->username, username)==0)
		return prof_Array[i];
```

```
}
return NULL;
}


int  find_profile_index(char * username, Profile ** prof_Array,int arr_len)
{
int i;

for(i=0;i<arr_len;i++)
{
	if(strcmp(prof_Array[i]->username, username)==0)
		return i;

}
return -1;
}



Profile * new_profile(char * username)
{
	Profile * new_p;

	new_p = (Profile*)malloc(sizeof(Profile));

		if(new_p==NULL)
		{
		printf("Error");
		return;
		}



	new_p->username = (char*)malloc(strlen(username)+1);

	if(new_p->username==NULL)
		{
		printf("Error");
		return;
		}

	strcpy(new_p->username, username);
	new_p->friends=(char*)malloc(1);

	if(new_p->friends==NULL)
		{
		printf("Error");
		return;
		}


	new_p->friends[0] = '\0';
	new_p->RequestHead = NULL;
	new_p->status=(char*)malloc(1);

	if(new_p->status==NULL)
```

```
                {
                printf("Error");
                return;
                }


        new_p->status[0] = '\0';
         return new_p;
}

void add_profile(Profile* new_p, Profile *** Array, int*arr_len)
{
        if (new_p==NULL)
                return;

        *arr_len = *arr_len + 1 ;
        *Array = (Profile**)realloc(*Array, (*arr_len)*sizeof(Profile*));


        *((*Array)+*arr_len-1) = new_p;

}

#endif
```

```
#ifndef READ_FILES
#define READ_FILES


#include "definitions.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void ReadLine( FILE *fp, char *str  )//Read a line from a file into an arrray of
characters.
{
        char c;
        int i; //Counter

        i=0;
        c=getc(fp);
        while (c!='\n' && c!=EOF )
        {
                str[i]=c;
                i++;
                c=getc(fp);
        }
        str[i]='\0';
}

char * detect_separation(char * str)// finds _$_ and give pointer to 3 chars
ahead/

{
        if(*str == '_' && *(str+1)=='$' && *(str+2)=='_')
```

```
                return str + 3;
        else
                return NULL;
        }

int bits2number(char * bits)
{
        int len, i, twos, number;
        twos=1;
        number=0;
        len = strlen(bits);
        for(i=len-1; i>=0; i--)
        {
                number = number + twos*( (int)bits[i] - (int)('0'));

                twos = twos*2;
        }
        return number;
}

char  * get_user(char * line, char * user_hlp)
{
        int i;
        char * helper;
        for(i=0; i<50; i++)
        {
                if (line[i]== '_')
                {
                        helper = detect_separation(line+i);
                                if (helper != NULL)
                                {
                                        user_hlp[i]='\0';
                                        return helper;
                                }
                }

                user_hlp[i] = line[i];
        }
}

char  * get_enc_pass(char * line, char * pass_hlp)
{
        int i;
        char * helper;
        for(i=0; i<33; i++)
        {
                if (line[i] == '_')
                {
                        helper = detect_separation(line+i);
                                if (helper != NULL)
                                {
                                        pass_hlp[i]='\0';
                                        return helper;
                                }
                }

                pass_hlp[i] = line[i];
        }
```

```
	}

int txt2int(char * number)
{
	int num;
	int i;
	num=0;
	for(i=0; i<strlen(number); i++)
	{
		num=num*10;
		num=num +  ( (int)(number[i]) - 48 );
	}
	return num;
}

	char  * get_ranNum(char * line, char * rnum_hlp)
{
	int i;
	char * helper;
	for(i=0; i<20; i++)
	{
		if (line[i] == '_')
		{
			helper = detect_separation(line+i);
				if (helper != NULL)
				{
					rnum_hlp[i]='\0';
					return helper;
				}
		}

		rnum_hlp[i] = line[i];
	}
	}

	void  get_answer(char * line, char * ans_hlp)
	{
	int i;
	char * helper;
	for(i=0; i<100; i++)
	{
		if (line[i] == '\0')
		{
			ans_hlp[i] = '\0';
		return;
		}
		ans_hlp[i] = line[i];
	}
	}

	Account * create_node(char * line)//working
	{
		char Username[51];
		char Pass_bits[33];
		char RanNum[20];
		char Answer[201];
		char * h1, *h2,* h3;
		Account  * new_node;
```

```
		h1 = get_user(line, Username);
		h2 = get_enc_pass(h1, Pass_bits);
		h3 = get_ranNum(h2, RanNum);
		get_answer(h3, Answer);

		new_node = (Account*) malloc(sizeof(Account));

		if (new_node==NULL)
		{
			printf("Error allocating memory/n");
			return NULL;
		}

		strcpy(new_node->Username, Username);
		new_node->enc_Pass = bits2number(Pass_bits);
		new_node->randomNum = txt2int(RanNum);
		strcpy(new_node->answer, Answer);
		new_node->next=NULL;
		return new_node;
	}

void read_validation(FILE * fp ,Account ** head)//working
{
	char line[306];
	Account * helper;
	while(!feof(fp))
	{
		ReadLine(fp, line);
		helper = create_node(line);
		*head = helper;
		head = &(*head)->next;
	}
}

#endif
```

```
#ifndef READ_FILES_2
#define READ_FILES_2


#include <stdio.h>
#include <stdlib.h>
#include "definitions_2.h"

char * ReadLine_2(FILE * fp)
{
	int i;
	char * string;
	char chr;

	string = (char*)malloc(sizeof(char));

	if(string==NULL)
	{
		printf("Error allocating");
		return NULL;
```

```
	}

	string[0] = '\0';
	chr = fgetc(fp);
	i=0;
	while(chr!=EOF && chr!='\n')
	{
		string[i] = chr;
		i++;
		string = (char*)realloc(string, i+1*(sizeof(char)));
		string[i] = '\0';
		chr = fgetc(fp);
	}
	return string;
}

int find_sep(char * chr)
	//return 1 if there is a separation;
{
	if(*chr== '_' && *(chr+1)=='$' && *(chr+2)=='_')
		return 1;
	else
		return 0;
	}


Profile * create_pending(char * line)
{
	Profile * temp;
	int i, len;

	if(line[0]=='\0')
		return NULL;
	temp  = (Profile*)malloc(sizeof(Profile));

		if(temp==NULL)
		{
		printf("Error");
		return NULL;
		}

	len  = strlen(line);
	i=0;

	temp->username=NULL;
	for(i=0; i<=len; i++)
	{

		temp->username = (char*)realloc(temp->username, i+1);

		if(line[i]=='\0')
		{
			temp->username[i] = '\0';
			temp->RequestHead = NULL;
			break;
		}
```

```
		else if(find_sep(line+i))
		{
			temp->username[i] = '\0';
			temp->RequestHead = create_pending(line+ +i+ 3);
			break;
		}



		temp->username[i] = line[i];

	}
	temp->friends=NULL;
	temp->status  =NULL;
	return temp;
}


char * create_friends(char *line)

{
	char * helper=NULL;
	int length, i, k;
	length = strlen(line);

	i=0;
	k=0;
	while(i<=length)
	{


		 if(line[i]=='\0')
		{
			helper  = (char*)realloc(helper, k+1);
			helper[k] = '\0';
			i++;

		}

		else if(find_sep(line+i))
		{
			helper  = (char*)realloc(helper, k+1);
			helper[k] = ';';
			i = i+3;
		}



		else
		{
			helper  = (char*)realloc(helper, k+1);
			helper[k] = line[i];
			i++;
		}

		k++;
	}
```

```
	return helper;


	}

Profile * create_profile(FILE * fp)
{
	char * username_l, *username;
		char* status_l, *status;
		char* new_f;
		char * new_p;
		Profile * new_a;
		new_a = (Profile*)malloc(sizeof(Profile));


		username_l = ReadLine_2(fp);
		status_l = ReadLine_2(fp);
		new_f  =ReadLine_2(fp);
		new_p = ReadLine_2(fp);

		if(strstr(username_l,"Username_")==username_l)

		{
			username = (char*)malloc(strlen(username_l+9)+1);


			strcpy(username, username_l+9);//
			new_a->username  = username;
		}
		else
			goto Error;

		if(strstr(status_l,"Status_")==status_l)

		{
			status= (char*)malloc(strlen(status_l+7)+1);
			strcpy(status, status_l+7);//
			new_a->status  = status;
		}
		else
			goto Error;

		if(strstr(new_f,"Friends_")==new_f)
		{
		new_a->friends = create_friends(new_f+8);
		}
		else
			goto Error;
		if(strstr(new_p,"Pendings_")==new_p)
		new_a->RequestHead = create_pending(new_p+9);
		else
			goto Error;


	 free(username_l);
		free(status_l);
		free(new_f);
```

```
                free(new_p);


        return new_a;

        Error: free(username_l);
                free(status_l);
                free(new_f);
                free(new_p);
                printf("Error with profiles.txt\n");
                free(new_a);
                new_a=NULL;
                return NULL;
}

int read_profiles(FILE * fp, Profile *** head)
{
        int i;
        i = 0;
        while(!feof(fp))
        {

                *head = (Profile**)realloc(*head, sizeof(Profile*)*(i+1));
                (*head)[i] = create_profile(fp);
                        i++;


        }


        return i;
}
#endif
```

```
#ifndef STATUS_H
#define STATUS_H

#include <stdio.h>
#include <string.h>
#include "profiles.h"

void print_friends_status(Profile ** prof_Array, Profile * user, int arr_len)
{
char helper[51];
int i, length, k;
Profile * friend_prof;

length = strlen(user->friends);
k=0;
for(i=0; i<=length; i++)
{
        if((user->friends)[i]=='\0')
        {
                helper[k] = '\0';
```

```
                friend_prof = find_profile(helper, prof_Array, arr_len);
                        if(friend_prof!=NULL)
                                printf("-%s Status:%s\n", friend_prof->username,
friend_prof->status);
        }
        else if((user->friends)[i]==';')
        {
                helper[k] = '\0';
                friend_prof = find_profile(helper, prof_Array, arr_len);
                if(friend_prof!=NULL)
                                printf("-%s Status:%s\n", friend_prof->username,
friend_prof->status);
                k=0;
        }

        else
        {
                helper[k] = user->friends[i];
                k++;
        }


        }
}

void change_status(char * new_status, Profile * User)
{
        int length;
        length = strlen(new_status);
        free(User->status);
        User->status = (char*)malloc(length+1);

        if(User->status==NULL)
                {
                printf("Error");
                return;
                }

        strcpy(User->status, new_status);
}
```

```
#endif
```

```
#ifndef WRITE_F
#define WRITE_F


#include "definitions.h"
#include <stdlib.h>
#include <stdio.h>
#include "definitions_2.h"
```

```
void int2bits(int num,  char * str)
{
	int i;
	str[32] = '\0';
	for (i=31; i>=0; i--)
	{
		str[i] = (int)('0') + (num % 2);
		num = num/2;
	}
}

int find_num_digits(int num)
{
	int ten;
	int i;
	ten=10;
	i = 1;
	while(num/ten>0)
	{
		ten = ten*10;
		i++;
	}
	return i;

}

char * int2txt(int num)// FIX
{
	//free the address after finishing;
	int i; //numbers of dgits in number;
char * helper;
i = find_num_digits(num);
helper = (char*)malloc(sizeof(char)*i+1);

if(helper==NULL)
		{
		printf("Error");
		return;
		}


helper[i] = '\0';

for(i=i-1; i>=0; i--)
{
helper[i]  =(int)('0') + num%10;
num = num/10;
}

return helper;
}



void write_validation( Account * current, FILE * fp)

{
```

```
        char bits[33];
        char * number;

        if(current==NULL)
        return ;



        fprintf(fp, "%s_$_", current->Username);//OK
        int2bits(current->enc_Pass, bits);//OK

        fprintf(fp, "%s_$_", bits);//OK
        number = int2txt(current->randomNum);//ok

        fprintf(fp, "%s_$_", number);



        fprintf(fp, "%s", current->answer);
        free(number);

        if(current->next!=NULL)
        fputc('\n', fp);

        write_validation(current->next, fp);


}



#endif
```

```
#ifndef WRITE_FILES_2
#define WRITE_FILES_2
#include "definitions_2.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void write_friends(char* friends, FILE * fp)//belongs to friends, can be used for
profile command
{
       int i, n;
       n = strlen(friends);
       i=0;

       while(i<n)
       {
              if(friends[i]==';')
                     fprintf(fp,"_$_");
              else
                     fputc(friends[i], fp);

              i++;
       }
}
```

```
void write_pending(Profile * reqHead, FILE * fp)
{


	if(reqHead==NULL)
		return;
	else if(reqHead->RequestHead==NULL)
	{
		fprintf(fp, "%s", reqHead->username);
	}

	else
	{
		fprintf(fp, "%s_$_", reqHead->username);
		write_pending(reqHead->RequestHead, fp);
	}

}




void write_prof(Profile ** head,  FILE * fp, int length)
{

	int  i;

	for(i=0; i<length-1; i++)
	{
	fprintf(fp, "Username_%s\n", head[i]->username);
	fprintf(fp, "Status_%s\n", head[i]->status);
	fprintf(fp, "Friends_");
	write_friends(head[i]->friends, fp);
	fprintf(fp, "\nPendings_");
	write_pending(head[i]->RequestHead, fp);
	fputc('\n', fp);
	}

	if(i==length-1)
	{

		fprintf(fp, "Username_%s\n", head[i]->username);
	fprintf(fp, "Status_%s\n", head[i]->status);
	fprintf(fp, "Friends_");
	write_friends(head[i]->friends, fp);
	fprintf(fp, "\nPendings_");
	write_pending(head[i]->RequestHead, fp);

	}
}

#endif
```