```c
#include <stdio.h>
#include <string.h>

// size ofchars
#define SIZE 10
// number of modules
#define MOD 4
// number of students max 13+9+14+6=42
#define MAXST 42

//one list for all surnames
struct allsurnames
{
    char surnameslist[SIZE];
};
//all names/surnames in all modules
struct names
{
    char firstname[SIZE];
    char surname[SIZE];
};
//structure for all 4 modules
struct modules
{
    char code[SIZE];
    // 1 FULL -- 0 PART
    int type;
    int maximum;
    int current;
    struct names students[MAXST];
};



void join(struct modules *modsf);
void leave(struct modules *modsf);
void display(struct modules *modsf);

void sorted_surnames( struct allsurnames *surnameslist,struct modules *modsf);
void merge_sort(int low, int high, struct allsurnames *surnameslist);
void merge(int low, int mid, int high, struct allsurnames *surnameslist);

void linear_search(struct modules *modsf);

int binary_search(struct names student[], int n, char searchkey[]);

int main()
{
```

```c
    struct modules mods[MOD] = {{"DT265A", 0, 13, 0, {' ', ' '}},
                                {"DT265C", 0, 9, 0, {' ', ' '}},
                                {"DT265B", 1, 14, 0, {' ', ' '}},
                                {"DT8900", 1, 6, 0, {' ', ' '}}};
    struct allsurnames surnames[MAXST];
    // menu choice
    int choice;
    // end while loop
    int end = 1;
    char searchkey[SIZE];

    do
    {
        // menu
        printf("\n\n__________\n");
        printf("1.FULL-TIME students\n");
        printf("2.Join module\n");
        printf("3.Leave module\n");
        printf("4.Display modules data\n");
        printf("5.Sorted surnames\n");
        printf("6.Whats your module\n");
        printf("7.Quit\n");

        printf("\nEnter your choice:\n");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            //tells you what students are enrolled in a fulltime course
            printf("\nFull-time students:\n");
            linear_search(mods);

            break;
        }
        case 2:
        {
            //add students to the modules
            join(mods);
            break;

        } // end case2
        case 3:
        {
            //remove students from the modules
            leave(mods);
            break;
```

```
        } // end case3
        case 4:
        {
            //display modules data
            display(mods);
            break;
        } // end 4
        case 5:
        {
            //all students in one list in alphabetical order
            sorted_surnames(surnames,mods);
            break;
        }
        case 6:
        {
            //tells you what module youare enrolled in based on your surname
            printf("\nEnter your Surname:");
            scanf("%s", searchkey);
            int find = -1;
            int result;
            for (int i = 0; i < MOD; i++)
            {
                if (mods[i].current >0)
                {
                    result
=binary_search(mods[i].students,mods[i].current,searchkey);
                    if(result !=-1)
                    {
                        find = i;
                        break;
                    }//end inner if
                }//end outer if
            }//end for


            if( find != -1)
            {
                printf("\nYour name has been found\n");
                printf("\n%s enrolled in %s\n",searchkey,mods[find].code);
            }//end if
            else
            {
                printf("\nStudent not found\n");
            }
            break;
        } // end case5
        case 7:
        {
```

```
            end = -1;
            break;
        } // end case6
        default:
        {
            printf("\nPlease enter an option from the menu\n");
            break;
        }

        } // end switch

    } while (end == 1); // end while
    return 0;
} // end main

void join(struct modules *modsf)
{
    // modulechoice
    int modch = 0;
    char mchoice[SIZE];
    int find = 0;
    char firstname[SIZE];
    char surname[SIZE];
    // enter what module to join
    printf("what module do you want to join:\n");
    scanf("%s", mchoice);
    printf("Please enter your name:\n");
    scanf("%s", firstname);
    printf("Please enter your surname:\n");
    scanf("%s", surname);

    for (int i = 0; i < MOD; i++)
    {

        // compare/ check that it exists
        modch = strcmp(modsf[i].code, mchoice);
        if (modch == 0)
        {
            // module has been found
            find = 1;
            // if number of students is less than the max amount
            if (modsf[i].current < modsf[i].maximum)
            {
                // add one person to the module
                // register them
                modsf[i].current++;
                strcpy(modsf[i].students[modsf[i].current - 1].firstname,
firstname);
```

```
                strcpy(modsf[i].students[modsf[i].current - 1].surname,
surname);
                printf("\nYou have been added to the module\n");

            }    // end if
            else // number of students is greater than max
            {
                printf("\nModule is full\n");
            } // end else
            break;
        } // end if

    } // end for
    if (find == 0)
    {
        printf("\nPlease chose an existing
module\nDT265A\nDT265C\nDT265B\nDT8900\n");
    } // end if
} // end function join

void leave(struct modules *modsf)
{
    // modulechoice
    int modch = 0;
    int nameexist = 0;
    char mchoice[SIZE];
    int find = 0;
    char surname[SIZE];
    // enter what module to join
    printf("\nwhat module do you want to leave:\n");
    scanf("%s", mchoice);
    printf("\nPlease enter your surname:\n");
    scanf("%s", surname);
    for (int i = 0; i < MOD; i++)
    {
        // compare/ check that it exists
        modch = strcmp(modsf[i].code, mchoice);

        if (modch == 0)
        {
            // module has been found
            find = 1;
            for (int j = 0; j < modsf[i].current; j++)
            {
                nameexist = strcmp(modsf[i].students[j].surname, surname);
                if (nameexist == 0)
                {
                    // SUBSTRACT one person to the module
```

```
                    // UNregister them
                    modsf[i].current--;
                    for (int k = 0; k < modsf[i].current; k++)
                    {
                        strcpy(modsf[i].students[k].surname,
modsf[i].students[k + 1].surname);
                    }

                    printf("\nYou have exited the module\n");
                    break;
                } // end if

            } // end for

        } // end if

    } // end for
    if (find == 0)
    {
        printf("\nPlease chose an existing
module\nDT265A\nDT265C\nDT265B\nDT8900");
    } // end if
} // end function leave

void display(struct modules *mods)
{
    printf("\nFULL-TIME = 1\nPART-TIME = 0\n");
    printf("\nMODULE:  TYPE:  MAX:  CURRENT:\n ");
    for (int  i = 0; i < MOD; i++)
   {
        printf("\n%s     %d    %d     %d
",mods[i].code,mods[i].type,mods[i].maximum,mods[i].current);
        for (int j = 0; j < mods[i].current; j++)
        {
            printf(" %s %s \n",
mods[i].students[j].firstname,mods[i].students[j].surname);
        }//end for inner

   }//end fo outer


} // end display
void sorted_surnames(struct allsurnames *surnameslist, struct modules *modsf)
{
    int count = 0;

    for (int i = 0; i < MOD; i++)
    {
```

```
        for (int j = 0; j < modsf[i].current; j++)
        {
            //copy all surnames from modules structure to new structure with
only surnames
            strcpy(surnameslist[count].surnameslist,
modsf[i].students[j].surname);
            count++;
        }
    }

    //call merge sort with sorted surnames
    merge_sort(0, count - 1, surnameslist);

    //display sorted surnames
    printf("Sorted surnames:\n");
    for (int i = 0; i < count; i++)
    {
        printf("%s\n", surnameslist[i].surnameslist);
    }
}

void merge_sort(int low, int high, struct allsurnames *surnameslist)
{

    if(low < high)
    {
        //divide the list in low, mid and high positions
        int mid = (low + high) / 2;
        merge_sort(low, mid, surnameslist);
        merge_sort(mid+1, high, surnameslist);
        //merge function
        merge(low, mid, high, surnameslist);
    }//end if
}//end merge_sort

void merge(int low, int mid, int high, struct allsurnames *surnameslist)
{
    int left = low;
    int right = mid + 1;
    char temp[MAXST][SIZE];
    int ptrtemp = 0;

    while(left <= mid && right <= high)
    {
        //compare surnames left and right and swap
        if(strcmp(surnameslist[left].surnameslist,
surnameslist[right].surnameslist) < 0)
        {
```

```
            //swap if condition true
            //use ptertemp as temporary variable to store surname and excecute
the swap
            //increment indeces(go to the next name)
            strcpy(temp[ptrtemp], surnameslist[left].surnameslist);
            ptrtemp++;
            left++;
        } //enf if
        else
        {
            //copy surname to temporary variable
            //increment indeces(go to the next name)
            strcpy(temp[ptrtemp], surnameslist[right].surnameslist);
            ptrtemp++;
            right++;
        }//end else
    }//end while

    while(left <= mid)
    {
        //copy surname to temporary variable
        //increment indeces(go to the next name)
        strcpy(temp[ptrtemp], surnameslist[left].surnameslist);
        ptrtemp++;
        left++;
    }//end while

    while(right <= high)
    {
        //copy surname to temporary variable
        //increment indeces(go to the next name)
        strcpy(temp[ptrtemp], surnameslist[right].surnameslist);
        ptrtemp++;
        right++;
    }//end while

    for(int i = 0; i < ptrtemp; i++)
    {
        //copy the temp variable into structure with only surnames
        //structre now holds the sorted surnames
        strcpy(surnameslist[low + i].surnameslist, temp[i]);
    }//end for
}//end merge

void linear_search(struct modules *modsf)
{
    //full time is represented by 1 in the structure
    int key = 1;
```

```
    int found = 0;

    //outer loop go trough  4 modules
    for (int i = 0; i < MOD ; i++)
    {
        //inner loop go trough students
        for (int j = 0; j < MAXST; j++)
        {
            //find 1 (full-time)
            if (modsf[i].type == key)
            {
                //display students in full-time
                printf("%s %s\n",
modsf[i].students[j].firstname,modsf[i].students[j].surname);
                found = 1;
                break;
            }//end if

        }//end for inner
        if (found = 0)
        {
            printf("\nNo students in FULL-TIME modules\n");
        }
    }//end for outer

}//end linear search

int binary_search(struct names student[], int n, char searchkey[])
{
    //divide data in positions low, mid,high
    int low =0;
    int high= n-1;
    int middle;

    while (low <= high)
    {
        middle =(low +high)/2;

        int cmp = strcmp(student[middle].surname,searchkey);

        if (cmp == 0)
        {
            //searck key will be found (returned) when is equals to the middle
element.
            return middle;

        }//end if
        else if (cmp < 0)
```

```
        {
            high = middle +1;

        }//end else if
        else if (cmp > 0)
        {
            low = middle -1 ;

        }//end else if
    }
    return -1;
}//end binary
```