#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
struct element{int num; char text[15];};  
struct list{int Nb; struct element Tab[600];};  
int Search\_Sorted\_List(struct list List\_in, int search\_ref);  
  
*//declare the function  
  
  
//----------------------------------------------------  
// procedure that read the list of 500 words from a file. Each word has a number.*void Read\_list(struct list \*L)  
{  
 FILE \*fp;  
 int k;  
  
 fp=fopen("../data.txt","r");  
 (\*L).Nb=500;  
 for(k=0;k<(\*L).Nb;k=k+1)  
 {  
 fscanf(fp,"%d%s",&(\*L).Tab[k].num,(\*L).Tab[k].text);  
 }  
}  
  
*//----------------------------------------------------  
// procedure to swap 2 items of type struct element*void Swap\_elements(struct element E\_in1, struct element E\_in2, struct element \*E\_out1, struct element \*E\_out2)  
{  
 \*E\_out1=E\_in2;  
 \*E\_out2=E\_in1;  
}  
  
  
*//##################### YOUR TASK BEGINS THERE ###############################  
//----------------------------------------------------  
// procedure to sort a list of type struct list  
/\* Sub-algorithm Ranking\_Sort\_List  
 In : List\_in : Type\_List2  
 Out : List\_out : Type\_List2  
Variables  
 index\_in\_element, scan\_index : integer  
 nb\_smaller : integer  
Instructions  
 for index\_in\_element from 1 to List\_in.Nb step 1  
 nb\_smaller  0  
 for scan\_index from 1 to List\_in.Nb step 1  
 if (List\_in.Tab[scan\_index].ref < List\_in.Tab[index\_in\_element].ref) then  
 nb\_smaller  nb\_smaller+1  
 endif  
 endfor  
 List\_out.Tab[nb\_smaller+1]  List\_out.Tab[index\_in\_element]  
 endfor  
 List\_out.Nb  List\_in.Nb \*/  
//void Ranking\_Sort\_List... - to be written in C by adapting the above sub-algorithm*struct list Ranking\_Sort\_List(struct list List\_in)  
{  
 int index\_in\_element, scan\_index;  
 int nb\_smaller;  
 struct list List\_out;  
 for (index\_in\_element = 1; index\_in\_element<= List\_in.Nb ;index\_in\_element++) {  
 nb\_smaller = 0;  
 for (scan\_index = 1; scan\_index <= List\_in.Nb; scan\_index++) {  
 if (List\_in.Tab[scan\_index].num < List\_in.Tab[index\_in\_element].num) {  
 nb\_smaller = nb\_smaller+1;  
 }  
 }  
 List\_out.Tab[nb\_smaller+1] = List\_out.Tab[index\_in\_element];  
  
 }  
 List\_out.Nb = List\_in.Nb;  
 return List\_out;  
}  
  
  
  
*//----------------------------------------------------  
//procedure to search a word in the list given its number  
/\*Sub-algorithm Search\_Sorted\_List  
 In : List\_in : Type\_List2, search\_ref : integer  
 Out : Found: Boolean, position : integer  
Variables :  
 index\_begin, index\_end, index\_mid : integer  
Instructions  
 Found <-- FALSE  
 if List\_in.Nb>0 then  
 index\_begin <-- 1  
 index\_end <-- List\_in.Nb  
 while ((index\_begin ≤ index\_end) and (Found=FALSE)) do  
 index\_mid <-- Integer\_Part((index\_begin + index\_end) /2)  
 if List\_in.Tab[index\_mid].ref = search\_ref then  
 Found <-- TRUE  
 position <-- index\_mid  
 elseif List\_in.Tab[index\_mid].ref < search\_ref then  
 index\_begin <-- index\_mid+1  
 else  
 index\_end <-- index\_mid-1  
 endif  
 endwhile  
 endif \*/  
//void Search\_Sorted\_List... - to be written in C by adapting the above sub-algorithm*int Search\_Sorted\_List(struct list List\_in, int search\_ref)  
{  
 int Found, position;  
 int index\_begin, index\_end, index\_mid;  
 Found = 0;  
 if (List\_in.Nb>0) {  
 index\_begin = 1;  
 index\_end = List\_in.Nb;  
 while ((index\_begin <= index\_end) && (Found == 0)) {  
 index\_mid = fabs((index\_begin + index\_end) /2);  
 if (List\_in.Tab[index\_mid].num = search\_ref) {  
 Found = 1;  
 position = index\_mid;  
 }  
 else if (List\_in.Tab[index\_mid].num < search\_ref) {  
 index\_begin = index\_mid+1;  
 }  
 else {  
 index\_end = index\_mid-1;  
 }  
 }  
 }  
  
 return Found, position;  
}  
  
  
*//----------------------------------------------------  
// procedure to display an element on the screen  
/\*Sub-algorithm display  
 In : List\_in : Type\_List2, index\_search\_item : integer  
 Out : -  
Variables : -  
Instructions  
 write("The word number",List\_in.Tab[index\_search\_item].ref," is : ",List\_in.Tab[index\_search\_item].text)  
\*/  
//void display... - to be written in C by adapting the above sub-algorithm*void display(struct list List\_in, int index\_search\_item)  
{  
 printf("The word number %d is %c:\n ", List\_in.Tab[index\_search\_item].num, List\_in.Tab[index\_search\_item].text);  
  
}  
  
*//----------------------------------------------------*int main(void)  
{  
 struct list The\_list;  
 int k, repetition; *// k : dummy variable, repetition : index of the main for loop* int Found; *// to implement a boolean, true when the searched number is found in the list* int index\_search\_item; *// entry number of a searched word* int num\_word; *// the number of the searched word* struct list List\_in;  
 int search\_ref;

found = 1;  
 Read\_list(&The\_list);  
 *// Ranking\_Sort\_List(The\_list!The\_list) - to be changed to C* Ranking\_Sort\_List(List\_in);  
  
 for(repetition=0;repetition<5;repetition=repetition+1)  
 {  
 printf("enter the integer ID of the word you are looking for (between 1 and 3000) : ");  
 scanf("%d",&num\_word);  
 *//Search\_Sorted\_List(The\_list,num\_word!Found,index\_search\_item) - to be changed to C* Search\_Sorted\_List(List\_in, search\_ref);  
  
 if (Found)  
 {  
 *//display(The\_list,index\_search\_item!) - to be changed to C* display( List\_in,index\_search\_item);  
  
 }  
 else  
 {  
 printf("The word number %d is not in the list.\n",num\_word);  
 }  
 }  
 scanf("%d",&k);  
 return 0;  
}