LAB 1:

fixstring.c:

```
#include <stdbool.h>
#include <assert.h>
#include "fixstring.h"

unsigned int fstring_length(fixstring s) {
      unsigned int i =0u;
      while(s[i] != '\0' && i<FIXSTRING_MAX){
            i++;
      }
      assert(i<FIXSTRING_MAX);
      return i;
}

bool fstring_eq(fixstring s1, fixstring s2) {
      unsigned int i =0u,tams1,tams2;
      bool eq_flag;
      tams1 = fstring_length(s1);
      tams2 = fstring_length(s2);

      eq_flag = tams1 == tams2;
      while(eq_flag && s1[i] != '\0' && s2[i] != '\0'){
            eq_flag = eq_flag && (s1[i] == s2[i]);
            i++;
      }
      return eq_flag;
}
bool fstring_less_eq(fixstring s1, fixstring s2) {
      unsigned int i,tams1,tams2;
      tams1 = fstring_length(s1);
      tams2 = fstring_length(s2);
      i = 0u;
      bool less_eq_flag=true;
      while(s1[i]==s2[i] && i<tams1 && i<tams2){
            i++;
      }
      less_eq_flag = s1[i]<=s2[i];
      return less_eq_flag;
}

void fstring_set(fixstring s1, const fixstring s2) {
      int i = 0;
      while (i<FIXSTRING_MAX && s2[i]!='\0') {
      s1[i] = s2[i];
            i++;
      }
      s1[i] = '\0';
}

void fstring_swap(fixstring s1,  fixstring s2) {
      fixstring aux;

      fstring_set(aux,s1);
      fstring_set(s1,s2);
      fstring_set(s2,aux);
}
```

EJ-5 - MORTI:

```
void fstring_swap(fixstring s1,  fixstring s2) {
    fixstring aux;
    fstring_set(aux,s1);
    fstring_set(s1,s2);
    fstring_set(s2,aux);
}

void swap(fixstring a[], unsigned int i, unsigned int j) {
    fstring_swap(a[i], a[j]);
}
```

SORT-HELPERS:

```
void swap(int a[], unsigned int i, unsigned int j){
    int tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}

bool goes_before(int x, int y) {
    return x <= y;
}
```

SELECTION SORT:

```
void selection_sort(int a[], unsigned int length) {
    for (unsigned int i = 0; i < length; ++i) {
        unsigned int min_pos = min_pos_from(a, i, length);
        swap(a, i, min_pos);
    }
}
```

INSERTION SORT:

```
static void insert(int a[], unsigned int i, unsigned int
length) {
    unsigned int j = i;
    //Forzado a iniciar en j = 1 adaptado al lenguaje en C:
    while(j>0 && goes_before(a[j],a[j-1]) && j<length){
        array_dump(a, length);
        swap(a,j-1,j);
        j = j-1;
    }
}

void insertion_sort(int a[], unsigned int length) {
    for (unsigned int i = 1; i < length; ++i) {
        assert(array_is_sorted(a,i));
        insert(a, i, length);
    }
}
```

QUICK SORT - ADAPTADO A GOES_BEFORE:

```
static unsigned int partition(int a[], unsigned int izq,
unsigned int der) {
    /* PRECONDITION:
       0 <= izq < der < length of the array

     Permutes elements of a[izq..der] and returns pivot such
that:
     - izq <= pivot <= der
     - elements in a[izq,pivot) all 'go_before' (according to
function goes_before) a[pivot]
     - a[pivot] 'goes_before' all the elements in a(pivot,der]
    */

    unsigned int ppiv = izq;
    unsigned int i = izq+1;
    unsigned int j = der;

    while(i<=j){
        if(goes_before(a[i],a[ppiv])){
            i = i+1;
        }else if(goes_before(a[ppiv],a[j])){
            j = j-1;
        }else{
            swap(a,i,j);
            i = i+1;
            j = j-1;
        }
    }
    swap(a,ppiv,j);
    ppiv = j;
    return ppiv;

}
```

LAB 2:

K-ESIMO ELEMENTO MÁS CHICO:

```
int k_esimo(int a[], int length, int k) {
    unsigned int izq;
    unsigned int der;
    int ppiv;

    izq = 0;
    der = length-1;

    array_dump(a,length);
    printf("lft = %d, rgt = %d, ppiv = 0\n",izq,der);

    ppiv = partition(a,izq,der);

    array_dump(a,ppiv+1);
    printf("lft = %d, rgt = %d, ppiv = %d\n",izq,der,ppiv);

    while(ppiv != k){
        if(ppiv < k){
            izq = ppiv+1;
        }else{
            der = ppiv-1;
        }
        ppiv = partition(a,izq,der);
        array_dump(a,ppiv+1);
        printf("lft = %d, rgt = %d, ppiv = %d\n",izq,der,ppiv);
    }
    /*
    array_dump(a,length);
    printf("izq = %d, der = %d, ppiv = %d \n",izq,der,ppiv);
    */
    return a[k];
}
```

HEADERS DE LAB 3:

```
typedef enum {january, february, march, april, may, june,
july, august, september, october, november, december}
month_t;
#define MONTHS 12
#define DAYS 28
typedef Weather WeatherTable [YEARS][MONTHS][DAYS];
```

TIENE_CIMA:

```
bool tiene_cima(int a[], int length) {
   // primero recorremos la parte creciente
   // frenamos cuando termina el arreglo o deja de ser
creciente
   int k = 0;
   while (k < length - 1 && a[k] < a[k+1]) {
       k++;
   }
   // terminamos. acá vale:
   // - arreglo creciente hasta posición k
   // - termina en posición k (k == length-1)
   //   || el que sigue es mayor o igual (a[k] >= a[k+1])


   // ahora recorremos la parte decreciente
   // frenamos cuando termina el arreglo o deja de ser
decreciente
   while (k < length - 1 && a[k] > a[k+1]) {
       k++;
   }

   // tiene cima si y sólo si llegamos hasta el final del
arreglo
   return k == length - 1;
}
```

RECURSOS EXTRA:

Declaración de una struct:

```
struct NombreDeLaEstructura {
    tipo_campo1 nombre_campo1;
    tipo_campo2 nombre_campo2;
    // ...
};
```

Con typedef tmb podemos declararla asi:

```
typedef struct {
    tipo_campo1 nombre_campo1;
    tipo_campo2 nombre_campo2;
} NombreAlias;
Ejemplo: struct NombreDeLaEstructura {tipo_campo1
nombre_campo1;tipo_campo2 nombre_campo2; …
}; //… pagina 4 despues de fscanf y fprintf
```

CIMA_LOG:

```
int cima_log(int a[], int length) {
   return cima_rec(a, 0, length-1, length);
}


// PRE: tiene_cima(a, length)
int cima_rec(int a[], int lft, int rgt, int length) {
   int result;
   int mid = (lft + rgt) / 2;
   if (es_cima(a, mid, length)) {
       result = mid;
   } else if (izq_cima(a, mid, length)) {
       result = cima_rec(a, lft, mid-1, length);
   } else if (der_cima(a, mid, length)) {
       result = cima_rec(a, mid+1, rgt, length);
   }
   return result;
}

// PRE: tiene_cima(a, length)
// Indica si la posición i del arreglo a es la cima.
bool es_cima(int a[], int i, int length) {
   return (i == 0 || a[i] > a[i-1]) && (i == length-1 || a[i] >
a[i+1]);
}

// PRE: tiene_cima(a, length)
// Indica si la cima está a la izquierda de la posición i del
arreglo
bool izq_cima(int a[], int i, int length) {
   return i > 0 && a[i] < a[i-1];
}

// PRE: tiene_cima(a, length)
// Indica si la cima está a la derecha de la posición i del
arreglo
bool der_cima(int a[], int i, int length) {
   return i < length-1 && a[i] < a[i+1];
}
```

LAB 3 - CONVENIENCIAS:

```
// También completar acá:
// Guardar la medición de clima en el arreglo multidimensional.
a[k_year-FST_YEAR][k_month-1][k_day-1] = weather;
```

EJEMPLOS DE USO 'FSCANF' Y 'FPRINTF':

```
Weather weather_from_file(FILE* file)
{
    Weather weather;
    int res = fscanf(file,"%d %d %d %u %u %u \n",
&weather._average_temp, &weather._max_temp, &weather._min_temp,
&weather._pressure, &weather._moisture, &weather._rainfall);

    if (res != 6) {
      fprintf(stderr, "Invalid table.\n");
      exit(EXIT_FAILURE);
    }
    return weather;
}

void weather_to_file(FILE* file, Weather weather)
{
    fprintf(file, "%d %d %d %u %u %u", weather._average_temp,
            weather._max_temp, weather._min_temp,
weather._pressure, weather._moisture, weather._rainfall);
}
int main() {
    struct Persona p1;
    char nombre[] = "Juan";
    for (int i = 0; i < 5; i++) { // 5 incluye el '\0'
          p1.nombre[i] = nombre[i];
    }
    p1.edad = 30;
    // Acceso a los campos
    printf("Nombre: %s\n", p1.nombre);
    printf("Edad: %d\n", p1.edad);
    return 0;
}
```