```
//Ej 1 parcial a
//Order first all red, then white and finally blue
void flag_sort(color_t a[], int length) {
    int i = 0, j = 0, k = length-1;

    while (j <= k) {
        if (a[j] == red) {
            swap(a, i, j);
            i++;
            j++;
        } else if (a[j] == blue) {
            swap(a, j, k);
            k--;
        } else if (a[j] == white) {
            j++;
        }
    }
}

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

```
//Ej 2 parcial a
//Given a year returns rainfalls on the whole year
int year_rainfall(WeatherTable a, int year) {
    int i = 0;
    while (i+1980 != year) {
        i++;
    }

    int rains = 0;
    for (int month = 0; month < MONTHS; ++month)
{
        for (int day = 0; day < DAYS; ++day) {
            rains += a[i][month][day]._rainfall;
        }
    }

    return rains;
}
```

```
//Ej 1 parcial b DyV
//Returns position of first 0 [1,...1,...0,...0]
//Note: add first_zero_rec to .h
int first_zero(int a[], int length) {
   return first_zero_rec(a, 0, length-1);
}

int first_zero_rec(int a[], int lft, int rgt) {
   int result;
   if (lft == rgt) {
      if (a[lft] == 0)
         result = lft;
      else
         result = -1;
   } else if (lft < rgt) {
      int mid = (lft+rgt) / 2;
      if (a[mid] == 0)
         result = first_zero_rec(a, lft, mid);
      else if (a[mid] == 1)
         result = first_zero_rec(a, mid+1, rgt);
   } else
      result = -1;

   return result;
}

/*
Another way without recursion and without add .h
int first_zero(int a[], int length) {
   int lft = 0, rgt = length-1, mid, result;

   if (lft > rgt)
      return -1;

   while (lft < rgt) {
      mid = (lft+rgt)/2;
      if (a[mid] == 0)
         rgt = mid;
      else if (a[mid] == 1)
         lft = mid+1;
   }

   if (lft == rgt) {
      if (a[lft] == 0)
         result = lft;
      else
         result = -1;
   }

   return result;
}
*/
```

```
//Ej 2 parcial b
//Returns highest temperature of a day in the
indicated year
   int year_max_temp(WeatherTable a, int year) {
      int i = 0, max = 0;
      while (i+1980 != year)
         i++;

      for (int month = 0; month < MONTHS; ++month)
{
         for (int day = 0; day < DAYS; ++day) {
            if (a[i][month][day]._max_temp > max)
              max = a[i][month][day]._max_temp;
         }
      }

      return max;
   }
```

```
    //Lab03 ej1 weather_table
    //Write the content of the table into stdout
    void table_dump(WeatherTable a) {
        for (unsigned int year = 0u; year < YEARS;
++year) {
            for (month_t month = january; month <=
december; ++month) {
                for (unsigned int day = 0u; day < DAYS;
++day) {
                    // imprimir año, mes y día
                    fprintf(stdout, "%u %u %u ", year +
FST_YEAR, month + 1, day + 1);

                    // imprimir datos para ese día
                    weather_to_file(stdout,
a[year][month][day]);

                    // imprimir salto de línea
                    fprintf(stdout, "\n");
                }
            }
        }
    }
```

```
    //Lab03 ej1 weather_table
    //Read the table information from file
    void table_from_file(WeatherTable a, const char
*filepath) {
        FILE *file = NULL;

        file = fopen(filepath, "r");
        if (file == NULL) {
            fprintf(stderr, "File does not exist.\n");
            exit(EXIT_FAILURE);
        }

        unsigned int k_year = 0u;
        unsigned int k_month = 0u;
        unsigned int k_day = 0u;
        while (!feof(file)) {
            int res = fscanf(file, " %u %u %u ", &k_year,
&k_month, &k_day);
            if (res != 3) {
                fprintf(stderr, "Invalid table.\n");
                exit(EXIT_FAILURE);
            }

            // Ir a la función 'weather_from_file' en
weather.c y completar!
            Weather weather = weather_from_file(file);

            // También completar acá:
            // Guardar la medición de clima en el arreglo
multidimensional.
            a[k_year-FST_YEAR][k_month-1][k_day-1] =
weather;
            //Luego en cada posicion, ejemplo a[0][0][0]
tendriamos:
            //a[0][0][0]._average_temp
            //a[0][0][0]._max_temp
            //Y asi con los demas, para no hacer eso
campo a campo se usa weather
        }

        fclose(file);
    }
```

```
//Lab03 ej1 weather.c
#include <stdlib.h>
#include "weather.h"

Weather weather_from_file(FILE* file)
{
    Weather weather;

    int aux = fscanf(file, " %d %d %d %u %u %u ",
                     &weather._average_temp,
                     &weather._max_temp,
                     &weather._min_temp,
                     &weather._pressure,
                     &weather._moisture,
                     &weather._rainfall);

    if (aux != 6) {
      fprintf(stderr, "Invalid table format\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);
}
```

```
//Lab03 ej 1 weather.utils.c
unsigned int lowest_temp_hist(WeatherTable table) {
  int result = table[0][0][0]._min_temp;
  for (unsigned int year = 0u; year < YEARS; ++year)
{
     for (unsigned int month = 0u; month < MONTHS;
++month) {
        for (unsigned int day = 0u; day < DAYS;
++day) {
           if (table[year][month][day]._min_temp <
result)
              result =
table[year][month][day]._min_temp;
        }
     }
  }
  return result;
}


void highest_temp_year(WeatherTable table,
unsigned int res[YEARS]) {
  for (unsigned int year = 0u; year < YEARS; ++year)
{
     int max = table[year][0][0]._max_temp;
     for (unsigned int month = 0u; month < MONTHS;
++month) {
        for (unsigned int day = 0u; day < DAYS;
++day) {
           if (table[year][month][day]._max_temp >
max)
              max =
table[year][month][day]._max_temp;
        }
     }
     res[year] = max;
  }
}


void most_rain_year(WeatherTable table, month_t
res[YEARS]) {
  for (unsigned int year = 0u; year < YEARS; ++year)
{
     month_t max_month = january;
     unsigned int max_rain = 0u;
     //Calculate rains by months
     for (unsigned int month = 0u; month < MONTHS;
++month) {
        unsigned int month_rain = 0u;
        for (unsigned int day = 0u; day < DAYS;
++day) {
           month_rain +=
table[year][month][day]._rainfall;
        }

        if (month_rain > max_rain) {
           max_rain = month_rain;
           max_month = month;
        }
     }
     res[year] = max_month;
  }
}
```

```
//Lab02 ej1 k_esimo.c
//Return element that will be in k position if ordered
// FUNCIONES INTERNAS DEL MÓDULO:
int partition(int a[], int izq, int der);
bool goes_before(int x, int y);
void swap(int a[], int i, int j);
#ifdef DEBUG
void array_dump(int a[], int length);
#endif

int k_esimo(int a[], int length, int k) {
   if (length < 0 || k < 0 || k >= length)
      return -1;

   int lft = 0, rgt=length-1;
   int ppiv = partition(a, lft, rgt);
   #ifdef DEBUG
   array_dump(a, length);
   printf("\nleft: %u\nright: %u\nppiv: %u\n\n", lft, rgt,
ppiv);
   #endif

   while (ppiv != k) {
      if (ppiv < k)
         lft = ppiv + 1;
      else
         rgt = ppiv - 1;
      ppiv = partition(a, lft, rgt);
      #ifdef DEBUG
      array_dump(a, length);
      printf("\nleft: %u\nright: %u\nppiv: %u\n\n", lft,
rgt, ppiv);
      #endif
   }

   return a[k];
}

int partition(int a[], int izq, int der) {
   int i, j, ppiv;
   ppiv = izq;
   i = izq + 1;
   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);
      }
   }
   swap(a, ppiv, j);
   ppiv = j;

   return ppiv;
}

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

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

#ifdef DEBUG
void array_dump(int a[], int length) {
   fprintf(stdout, "%u\n", length);
   for (int i = 0; i < length; ++i) {
      fprintf(stdout, "%d", a[i]);
      if (i < length - 1) {
         fprintf(stdout, " ");
      } else {
         fprintf(stdout, "\n");
      }
   }
}
#endif
```

```
//Lab02 ej2 cima.c
/**
 * @brief Indica si el arreglo tiene cima.
 * Un arreglo tiene cima si existe una posición k tal
que el arreglo es
 * estrictamente creciente hasta la posición k y
estrictamente decreciente
 * desde la posición k.
 * @param a Arreglo.
 * @param length Largo del arreglo.
 */
bool tiene_cima(int a[], int length) {
    if (length==1)
        return true;

    int i = 0;

    while (i < length-1 && a[i] < a[i+1])
        i++;

    if (i != 0 && i != length-1) {
        while (i < length-1 && a[i] > a[i+1])
            i++;
    } else
        return false;

    return i == length-1;
}

/**
 * @brief Dado un arreglo que tiene cima, devuelve la
posición de la cima.
 * Un arreglo tiene cima si existe una posición k tal
que el arreglo es
 * estrictamente creciente hasta la posición k y
estrictamente decreciente
 * desde la posición k.
 * La cima es el elemento que se encuentra en la
posición k.
 * PRECONDICION: tiene_cima(a, length)
 *
 * @param a Arreglo.
 * @param length Largo del arreglo.
 */
int cima(int a[], int length) {
    int cima = 0;
    for (int i=1; i < length-1; i++)
        if (a[i] > a[i-1])
            cima = i;

    return cima;
}
```

```
//Lab02 ej3 cima_log.c
/**
 * @brief Dado un arreglo que tiene cima, devuelve la
posición de la cima
 * usando la estrategia divide y venceras.
 *
 * Un arreglo tiene cima si existe una posición k tal
que el arreglo es
 * estrictamente creciente hasta la posición k y
estrictamente decreciente
 * desde la posición k.
 * La cima es el elemento que se encuentra en la
posición k.
 * PRECONDICION: tiene_cima(a, length)
 *
 * @param a Arreglo.
 * @param length Largo del arreglo.
 */
int cima_log(int a[], int length) {
    int lft = 0, rgt = length-1, mid;
    if (lft > rgt || length == 1)
        return 0;

    if (length == 2) {
        if(a[0] > a[1]) {
            return 0;
        } else
            return 1;
    }

    while (lft <= rgt) {
        mid = (lft + rgt)/2;
        if (a[mid-1] < a[mid] && a[mid] > a[mid+1])
            return mid;
        else if (a[mid-1] > a[mid] && a[mid-1] > a[mid+1])
            rgt = mid -1;
        else if (a[mid+1] > a[mid] && a[mid+1] >
a[mid-1])
            lft = mid +1;
    }
    return 0;
}
```

//Lab01 ej0 fixstring.c

```
/*
 * Get the length of a string.
 * Iterates starting in the first position until hit '\0'
 * counting the how many characters the string has.
 * Note we cannot use string.h library.
 */
unsigned int fstring_length(fixstring s) {
    unsigned int length = 0;

    while (s[length] != '\0')
        length++;

    return length;
}

/*
 * Check if two strings has the same content.
 * Iterates checking character by character if are equal
or if they are '\0'.
 * If are not the same character, return false, if are the
same return true
 */
bool fstring_eq(fixstring s1, fixstring s2) {
    unsigned int i = 0;
    while (s1[i] == s2[i] && s1[i] != '\0')
        i++;

    return s1[i] == s2[i];
}

/*
 * Check if s1 is smaller alphabeticaly than s2
 * Iterates checking character by character if are equal
or if they are '\0'.
 * If are not the same character, return true if s1 is
smaller, false if not.
 */
bool fstring_less_eq(fixstring s1, fixstring s2) {
    unsigned int i = 0;
    while (s1[i] == s2[i] && s1[i] != '\0')
        i++;

    return s1[i] <= s2[i];
}
```

//Lab01 ej1 sort.c

```
/*
 * Note: insert_proc from theory is insert_sort here.
 * We have to implement insert starting from 0.
 * void(lenght) is to avoid unused variable and allow
debugging.
*/
static void insert(int a[], unsigned int i, unsigned int
length) {
    (void)length;
    #ifdef DEBUG
    array_dump(a, length); printf("\n");
    #endif

    unsigned int j = i;
    while (j > 0 && goes_before(a[j], a[j-1])) {
        swap(a, j-1, j);
        j--;
    }
}

/*
 * Added assert inside for to check from [0, i)
 */
void insertion_sort(int a[], unsigned int length) {
    for (unsigned int i = 1; i < length; ++i) {
        insert(a, i, length);
        assert(array_is_sorted(a, i));
    }
}
```

```
//Lab01 ej3 sort.c
/*
 * Chop the array and return a pivot with it's correct order and position
 * Note: there's no need to check (a[i] > a[ppiv] && a[j] < a[ppiv]) because
 * that cases are the final else.
 * Also, can't return if izq < der because it's not a void function.
 * Need to use if instead of while to check goes_before because a while will
 * iterate until the last i <= j and the other while will never start.
 * It's basically the same than the theory with the difference it leave the
 * elements == on the left side instead the right side.
 */
static unsigned int partition(int a[], unsigned int izq, unsigned int der) {
    unsigned int ppiv = izq, i = izq + 1, j = der;

    //assert(izq < der);

    #ifdef DEBUG
    printf("\nleft: %u\nright: %u\nppiv: %u\n\n", izq, der, ppiv);
    #endif

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

    swap(a, ppiv, j);
    ppiv = j;

    return ppiv;
}

/*
 * Note: partition return uint asserting lft <= ppiv <= rgt
 * We don't have to check izq <= ppiv because quick_sort_rec takes ppiv -1
 * IDEM with der <= ppiv.
 * Need to check der > izq to assure precondition of partition (0 <= izq < der)
 * it's useful to avoid swap in case length=0
 * Check if ppiv=0, if it's don't change the value, if not it's higher so ppiv-1
 * It's not necessary to check the same with der, in case that ppiv+1 >= der
 * then the if(der > izq) will fail
 */
static void quick_sort_rec(int a[], unsigned int izq, unsigned int der, unsigned int length) {
    if (der > izq) {
        (void)length;
        #ifdef DEBUG
        printf("BEFORE:\n"); array_dump(a, length);
        #endif

        unsigned int ppiv = partition(a, izq, der);

        (void)length;
        #ifdef DEBUG
        printf("AFTER:\n"); array_dump(a, length);
printf("\n\n\n");
        #endif

        //if (izq < ppiv)
        quick_sort_rec(a, izq, (ppiv == 0) ? 0 : ppiv -1, length);

        //if (der > ppiv)
        quick_sort_rec(a, ppiv + 1, der, length);
    }
}

void quick_sort(int a[], unsigned int length) {
    quick_sort_rec(a, 0, (length == 0) ? 0 : length - 1, length);
}
```

```
//Lab01 ej5 fixstring.c
/*
 * Get the length of a string.
 * Iterates starting in the first position until hit '\0'
 * counting the how many characters the string has.
 * Note we cannot use string.h library.
 */
unsigned int fstring_length(fixstring s) {
    unsigned int length = 0;

    while (s[length] != '\0')
        length++;

    return length;
}

/*
 * Check if two strings has the same content.
 * Iterates checking character by character if are equal
or if they are '\0'.
 * If are not the same character, return false, if not
return true
 */
bool fstring_eq(fixstring s1, fixstring s2) {
    unsigned int i = 0;
    while (s1[i] == s2[i] && s1[i] != '\0')
        i++;

    return s1[i] == s2[i];
}

/*
 * Check if s1 is smaller alphabeticaly than s2
 * Iterates checking character by character if are equal
or if they are '\0'.
 * If are not the same character, return true if s1 is
smaller, false if not.
 */
bool fstring_less_eq(fixstring s1, fixstring s2) {
    unsigned int i = 0;
    while (s1[i] == s2[i] && s1[i] != '\0')
        i++;

    return s1[i] <= s2[i];
}

/*
 * Works as a swap function but it uses fixstring type
 */
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';
}

/*
 * Equivalent to use unsigned int but it only takes two
arguments fixstring type
 */
void fstring_swap(fixstring s1,  fixstring s2) {
    fixstring aux;
    fstring_set(aux, s1);
    fstring_set(s1, s2);
    fstring_set(s2, aux);
}
```

```
//Lab01 ej5 sort_helpers.c
/*
 * Equivalent to use unsigned int with a[]
 */
void swap(fixstring a[], unsigned int i, unsigned int j) {
    //fixstring aux;
    //fstring_set(aux, a[i]);
    //fstring_set(a[i], a[j]);
    //fstring_set(a[j], aux);
    fstring_swap(a[i], a[j]);
}

/*
 * It will depend on how we wanna sort
 */
bool goes_before(fixstring x, fixstring y) {
    //Alphabeticaly
    return fstring_less_eq(x, y);

    //Length
    //return fstring_length(x) <= fstring_length(y);
}

bool array_is_sorted(fixstring array[], unsigned int
length) {
    unsigned int i = 1;
    while (i < length && goes_before(array[i-1],
array[i])) {
        i++;
    }
    return (i >= length);
}
```