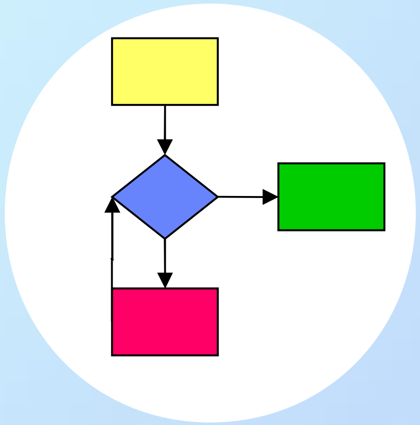
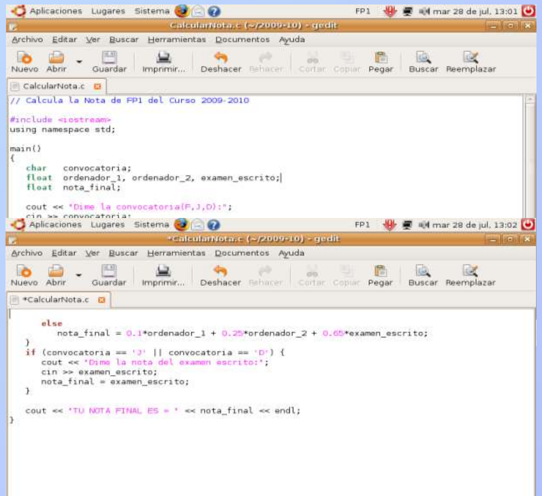


Programming 1

Lecture 6
Structured data types. Arrays



Aplicaciones Lugares Sistema
FP1 mar 28 de jul, 13:01
CalcularNota.c (~/2009-10) - gedit
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Nuevo Abrir Guardar Imprimir... Deshacer Rehacer Cortar Copiar Pegar Buscar Reemplazar
CalcularNota.c

```
// Calcula la Nota de FP1 del Curso 2009-2010

#include <iostream>
using namespace std;

main()
{
   char   convocatoria;
   float  ordenador_1, ordenador_2, examen_escrito;
   float  nota_final;

   cout << "Dime la convocatoria(F,J,D):";
   cin >> convocatoria;
```

Aplicaciones Lugares Sistema
FP1 mar 28 de jul, 13:02
*CalcularNota.c (~/2009-10) - gedit
Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Nuevo Abrir Guardar Imprimir... Deshacer Rehacer Cortar Copiar Pegar Buscar Reemplazar
*CalcularNota.c

```
      else
         nota_final = 0.1*ordenador_1 + 0.25*ordenador_2 + 0.65*examen_escrito;
   }
   if (convocatoria == 'J' || convocatoria == 'D') {
      cout << "Dime la nota del examen escrito:";
      cin >> examen_escrito;
      nota_final = examen_escrito;
   }

   cout << "TU NOTA FINAL ES = " << nota_final << endl;
}
```

Dpnt. de Ciència de la Computació i Intel·ligència artificial
Dpto. de Ciencia de la Computación e Inteligencia artificial

Objectives

- Understand the difference between simple and structured data types
- Manage the following structured data types: one-dimensional and two-dimensional arrays
- Manage one-dimensional and two-dimensional arrays in C language

Topics

Reminder: Simple data types

- All the variables that we have used so far are of **simple type**
- A variable of simple type can only store one **unique value** at a time
  - For instance, if x is of integer type, only one integer value can be stored at a time
    - X = 7;
    - X = 10;
    - X = 2000;

Structured data types

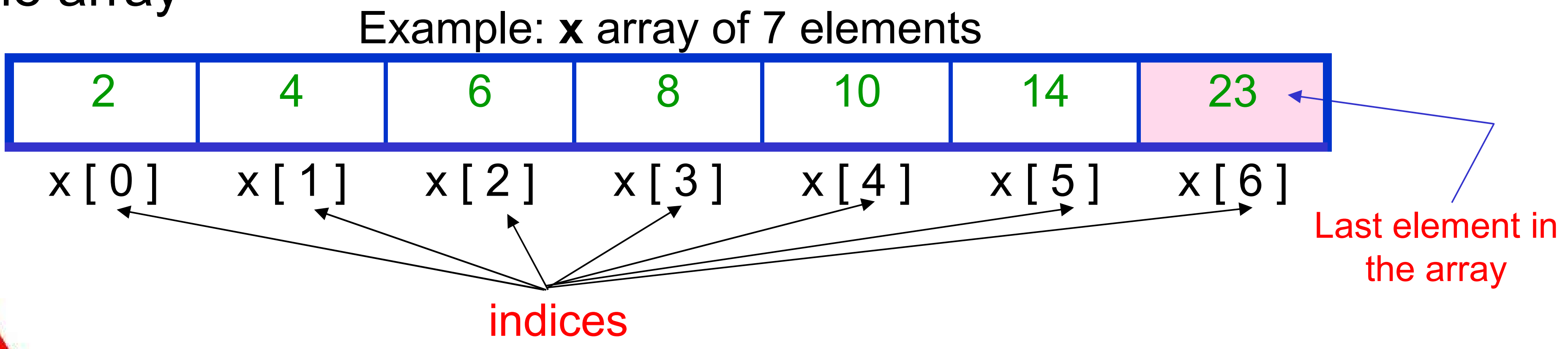
- A variable of a structured type is a collection of data of simple type
- A structured type can store more than one element (value) at a time
  - **Array Type**: all the elements stored in an array variable must be of the same type
  - **Record or Struct Type**: a struct variable can store elements of several types
- Example: *z* is a variable to store the winning numbers in the lottery. So, 6 values are stored at a time
  - *z* = (1, 4, 6, 24, 13, 2);
  - *z* = (3, 9, 12, 15, 23, 27);

In C language, the **struct** type is equivalent to the **record** type of other languages

Topics

The Array data type

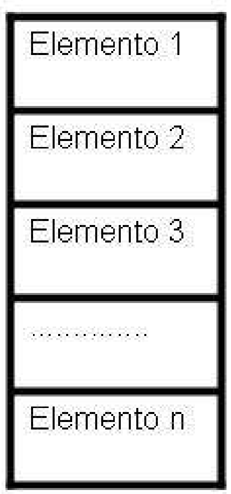
- An array is a data structure to store a **finite**, **homogeneous** and **ordered** collection of data:
  - **Finite**: The maximum number of elements to be stored must be determined
  - **Homogeneous**: Every element is of the same type
  - **Ordered**: the n-th element of the array can be determined
- To refer to a given element in an array, an **index** between squared brackets **[i]** is used. It specifies the relative position in the array



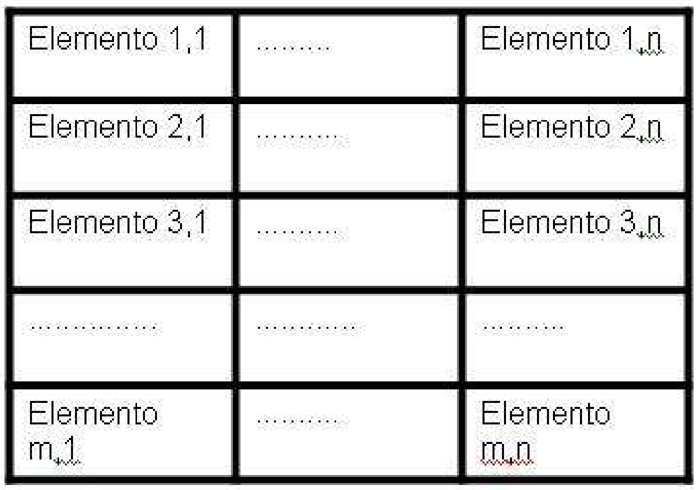
In C language, the first element in the array is placed in position (index) **zero**

Array classification

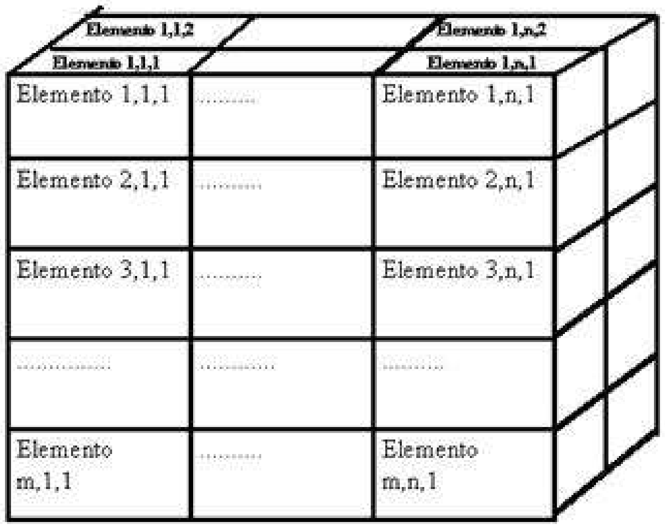
- Depending on the number of dimensions, the arrays can be classified as:
  - **One-dimensional** (vector)
  - **Two-dimensional** (matrix)
  - **Multi-dimensional** (three or more dimensions)
- The dimension of an array is the number of indices used to make reference to any of its elements



One-dimensional



Two-dimensional



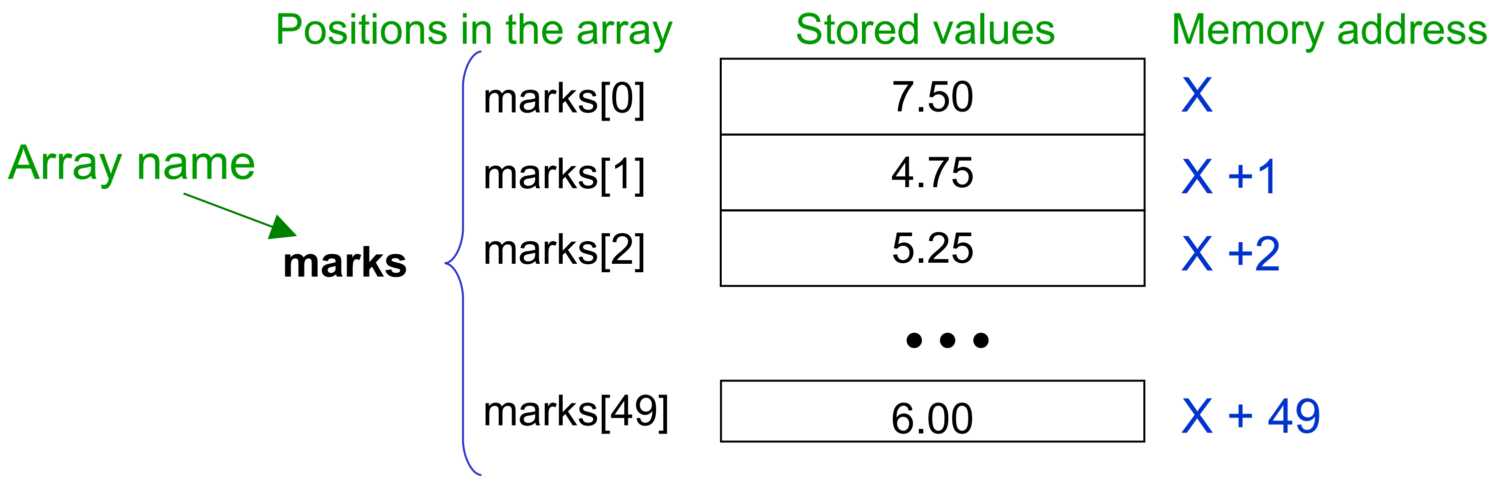
Multi-dimensional

Topics

1. Structured data types
2. Array data type
3. **One-dimensional arrays**
4. Two-dimensional arrays
5. Type definition using *typedef*
6. Information sources

One-dimensional arrays

- A one-dimensional array is a structured data type in which elements are stored in consecutive memory positions, each position being accessible through the use of an index
- Example: define the data type to store the mark of the Programming 1 exam for 50 students. The following steps are needed:
  1. Allocate 50 memory positions
  2. Give the array a name
  3. Associate each position in the array to each student
  4. Assign the mark for each position



Array declaration in C language

- First the array type variable (one-dimensional) must be declared in order to use it.
- Syntax

```
elements_type  array_name [ num_elem] ;
```

  - **elements_type**: indicates the type of each element in the array; every element is of the same type
  - **array_name**: indicates the name of the array; it can be any valid identifier
  - **[num_elem]**: indicates the maximum amount of elements in the array; it must be an integer constant value
- Example: **float marks [50];**

Initialization and access to an array

- As any other variable type, an array must be initialized before being used
- A possible way of **initializing** an array is accessing every element by using a loop and assigning them a value
- To **access** an array position the following syntax is used:

```
array_name [index] ;
```

- Example: access the mark of the student placed in 5th position in the array: **marks[4];**

Using values for the indices that are **out of the range** determined by the array size, produces unwanted errors when executing the program. These errors are difficult to detect.

Example 1: initializing an array

- If the values of every component of the array are known when the array is defined, definition and initialization can be done simultaneously:

```
// example array inicialization
#include <iostream>
using namespace std;


main () {
        int vectorA[4] = {1, 5, 3, 9};
        int vectorB [] = {1, 5, 3, 9};
        int vectorC[10] = {1, 5, 3, 9};
}
```

The size of the array is the number of values

The array can be partially inizialized

Example 2: initializing an array

- An array can be initialized by the user entering the data by keyboard, as follows:

```
// example array inicialization
#include <iostream>
using namespace std;

void inicialize_Array(float marks[ ]);

main () {
      float  marks[50];

      inicialize_Array(marks);
}
```

```
// procedure to inicialize the array
void inicialize_Array(float marks[ ])
{
   int i;

   for ( i=0 ; i < 50 ; i++ ) {
      cout << "Enter the mark “ << i << “:”;
      cin >> marks[i];
   }
}
```

In C language, the arrays are always passed **by reference** when used as parameters in a module.

In C language, functions **cannot return** an array type. For the array to be modified, it must be passed as a parameter.

Linear search of an element in an array

- When the elements in the array are **not sorted**
  - To search an element in the array a **linear search** can be used
    - The array is accessed consecutively from the first position until the searched element is found

```
// Linear search
// Function to find an element "elem" in an array with MAX_SIZE elements
// It returns the position of "elem" in the array if it is found, or -1 otherwise
int Linear_Search(int name_array[], int elem)
{
   int     pos;
   bool  found;

   pos = 0;
   found = false;
   // the search is finished if the array end is achieved or if the element is found
   while ( pos < MAX_SIZE  &&   ! found) {
         if  (name_array[pos] == elem)
               found = true;
         else
               pos = pos +1;
    }
    if (! found)
          pos = -1;

   return(pos);
}
```

Binary search of an element in an array

- When the elements in the array are **sorted**
  - To search an element in a sorted array a **binary, dichotomic or half-interval search** can be used
    - The search is reduced dividing the array in two, so that the search interval is smaller depending on the value to be searched

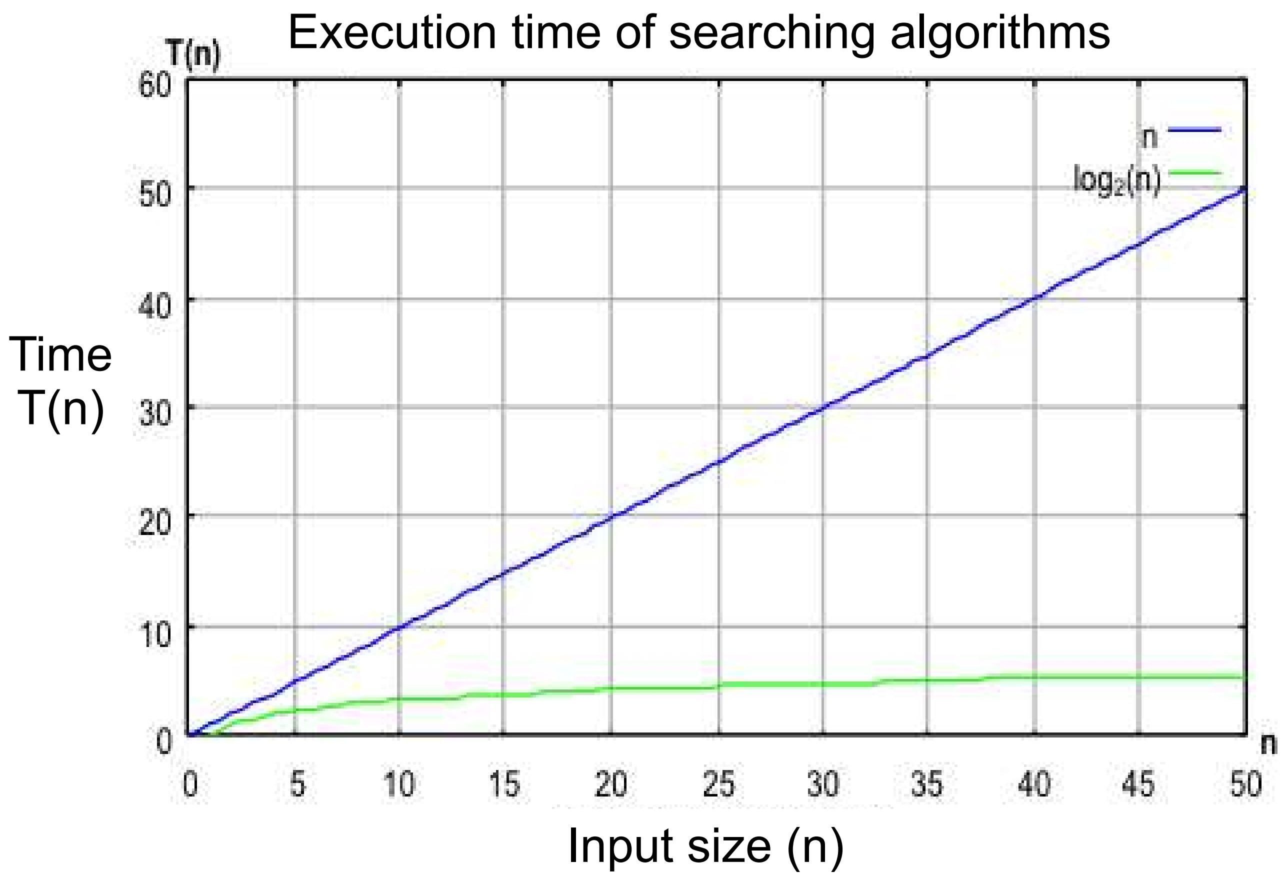
```
// Binary search of an element in an array of size MAX_SIZE. The elements are sorted in an ascending order
int Binary_Search(int array_name[], int elem)  {
    int      pos_begin, pos_end, pos_half;
    bool  found;

    // [pos_begin, pos_end] = current search interval
    pos_begin = 0;   // begin position in the array
    pos_end = MAX_SIZE -1;   // end position in the array
    found = false;
    while ( pos_begin <= pos_end   &&   ! found) {
        pos_half = (pos_begin + pos_end) / 2;   // half position in the array
        if  (elem == array_name[pos_half])  // element found at pos_half
            found = true;
        else if (elem > array_name[pos_half] )
            pos_begin = pos_half +1;   // the element must be searched in the upper half
        else
            pos_end = pos_half -1;   // the element must be searched in the lower half
    }
    if (! found)
       pos_half = -1;

    return(pos_half);
}
```

Search (temporary cost)

- Linear search: linear execution time
- Binary search: logarithmic execution time



Topics

Character strings

- A **character string** (or just string) is a finite sequence of consecutive characters
- We will use **character arrays** to store them
- A character array can store:
  - Words
  - Sentences
  - People's names, city names...
  - Alphanumeric codes
  - etc.

In C++ language, a special type called **string** exists. Nevertheless, we will **not use** it in P1

Character strings in C language

- In C language, a character string is written between **double quotation** marks

“hola”

- In C language, the character strings must always finish with the null character '\0' that must be stored in the array next to the last character in the string

‘h’	‘o’	‘l’	‘a’	‘\0’					
0	1	2	3	4	5	6	7	8	9

- Example: string “hola”
  - It is stored in a character string of size 10
  - It is made of 4 characters (length=4) but it occupies 5 characters in the array, because the character '\0' is also stored

```
char cad[10]=“hola”;
```

Functions in C to manage character strings

Function	Description	Use
cin.getline(*string*, *SIZE*)	Read a character string by keyboard up to the end of the line or until reaching the maximum size specified by SIZE (positive integer). The read sequence of characters is stored in *string* (character array of size ≤ SIZE)	As a procedure
strcpy(destination_string, origin_string)	Copy the string. It copies the content of *origin_string* into *destination_string*	As a procedure
strcmp(*string1, string2*)	Alphabetic comparison of strings If *string1* < *string2* then return a number < 0 If *string1* == *string2* then return 0 If *string1* > *string2* then return a number > 0	As a function
strlen(*string*)	Return a number of type *int* that indicates the length of the character string specified as a parameter, that is, the amount of valid characters of *string* (up to the special character of string end '\0', not including it)	As a function

Topics

Example: character string

- Function to return the length of a character string

```
// Return the length of a character string
// this function is equivalent to the predefined function of C strlen( )
int String_Length(char  str[ ])
{
      int len;

      len = 0;
      while (str[len] != ‘\0’)
         len++;

      return(len);
}
```

Example: one-dimensional arrays (I)

- Procedure to display the content of an array of elements of type *double*

```
// Display the element of an array of elements
// of double type
void print_Array(double a[], int len)
{
      int i;
       for (i=0; i < len; i++)
            cout << "[" << i << "] = " << a[i] << endl;
}
```

- Function to calculate the average mark of the all the students

```
// Average of  "len" marks if type float
float calculate_Average(float a[], int len)
{
      int     i;
      float  sum;

      sum = 0.0;
      for (i=0; i < len; i++)
          sum = sum + a[i];

      // assume len > 0
      return(sum / len);
}
```

Example: one-dimensional arrays (II)

- Given an array of LMAX integers, move all its elements one position to the right. The displacement is circular, that is, the last element will become the first one

```
void move_Circular (int  v[ ])
{
   int i, last;

   // store the value in the last position of the array (LMAX is the assumed length)
   last = v[LMAX-1];

   // move all the elements one position to the right, but the last one
   for (i=LMAX-1; i > 0; i--)
      v[i] = v[i-1];

   // store the last element in the first position
   v[0] = last;
}
```

Example: one-dimensional arrays (III)

- Given an array of LMAX integers, return the greatest value, the amount of occurrences of this value, and the first and last position where this value is stored

```
void Occurrencies(int  v[ ], int &greatest, int &amount_occur, int &pos_first, int &pos_last)
{
  int i;

  greatest = v[0]; //  initially, the greatest value is the one in the first position
  amount_occur = 1;
  pos_first = 0;
  pos_last = 0;

  // access the elements in the array: from the second position to the last one (assumed constant LMAX)
  for (i=1; i < LMAX; i++) {
    if (v[i] > greatest) {  //a new greatest is found
      greatest = v[i];
      amount_occur = 1;
      pos_first = i;
      pos_last = i;
    }
    else if (v[i] == greatest) {
      // a new occurrence of the current greatest value is found
      amount_occur = amount_occur +1;
      pos_last = i;
    }
  }
}
```

Topics

Array sorting algorithms

- The operation of sorting an array is frequent and interesting
  - Example: maintain the array of marks sorted, so that the best five marks can be obtained quickly. The array can be sorted in descending order, and access the first five positions in the array
- There are a lot of sorting algorithms for arrays. An efficient algorithm is the **Direct Insertion Algorithm**. For descending order:
  - Each element is compared with the ones on its left and it is inserted in its **correct position**
    - The correct position is reached when the element on the left is greater or when the left end of the array is found
  - During the search for the correct position, each lesser element is scrolled one position to the right

Example: array sorting algorithm (I)

- The **Direct Insertion Sorting Algorithm** can be compared with the sorting in a hand of cards
  - Each time a card is taken from the table, it is inserted in the correct position among the ordered cards that you have in your hand
- The arrays are divided into two parts:
  - The **first one** (that represents the cards in your hand) **is ordered and it grows** as the sorting algorithm is performed
  - The **second one** (the cards on the table) **is not ordered and it decreases** as the sorting algorithm is performed

Trace: Direct Insertion Sorting Algorithm

🟦 Sorted part

🟧 Unsorted part

Initial array	🟧 84	🟧 69	🟧 76	🟧 86	🟧 94	🟧 91
	🟦 84	🟧 69	🟧 76	🟧 86	🟧 94	🟧 91
	🟦 84	🟦 69	🟧 76	🟧 86	🟧 94	🟧 91
	🟦 84	🟦 76	🟦 69	🟧 86	🟧 94	🟧 91
	🟦 86	🟦 84	🟦 76	🟦 69	🟧 94	🟧 91
	🟦 94	🟦 86	🟦 84	🟦 76	🟦 69	🟧 91
Sorted array	🟦 94	🟦 91	🟦 86	🟦 84	🟦 76	🟦 69

Implementation: Direct Insertion Sorting Algorithm

```
// Sort an array in DESCENDING order using DIRECT INSERTION
void Sort_Array( int elem[ ], int amount_elem)
{
     int     left;    // position on the left of the inserted element in the sorted part
     int     right;   // position of the first element of the currently unsorted part
     int     current;  // first element in the currenly unsorted part. The one to be inserted
     bool   pos_found;

     // Initially, the ordered part (left part) is made up of the first position only,
     // so, the search begins with the first element of the unsorted part (right part)
     // from the second position (index=1)
    for (right = 1; right < amount_elem; right++)  {
          current = elem[right];
          left = right -1;
          pos_found = false;
          while ( left >= 0   &&   !pos_found) {
               if  (current > elem[left]) {
                    elem[left+1] = elem[left];  // the lesser elements are scrolled to the right
                    left = left -1;
               }
               else
                    pos_found = true;
          }
          // insert the first element of the unsorted part in its correct position
          // within the currently sorted part
          elem[left+1] = current;
     }
}
```

Another array sorting algorithm

- Another method to sort arrays is the **Direct Selection Sorting Algorithm**
- To sort an array in **descending order**:
  - **Step 1**: search and select the greatest element among the ones that are not sorted yet
  - **Step 2**: exchange the positions of this element and the one on the very left side of the unsorted part

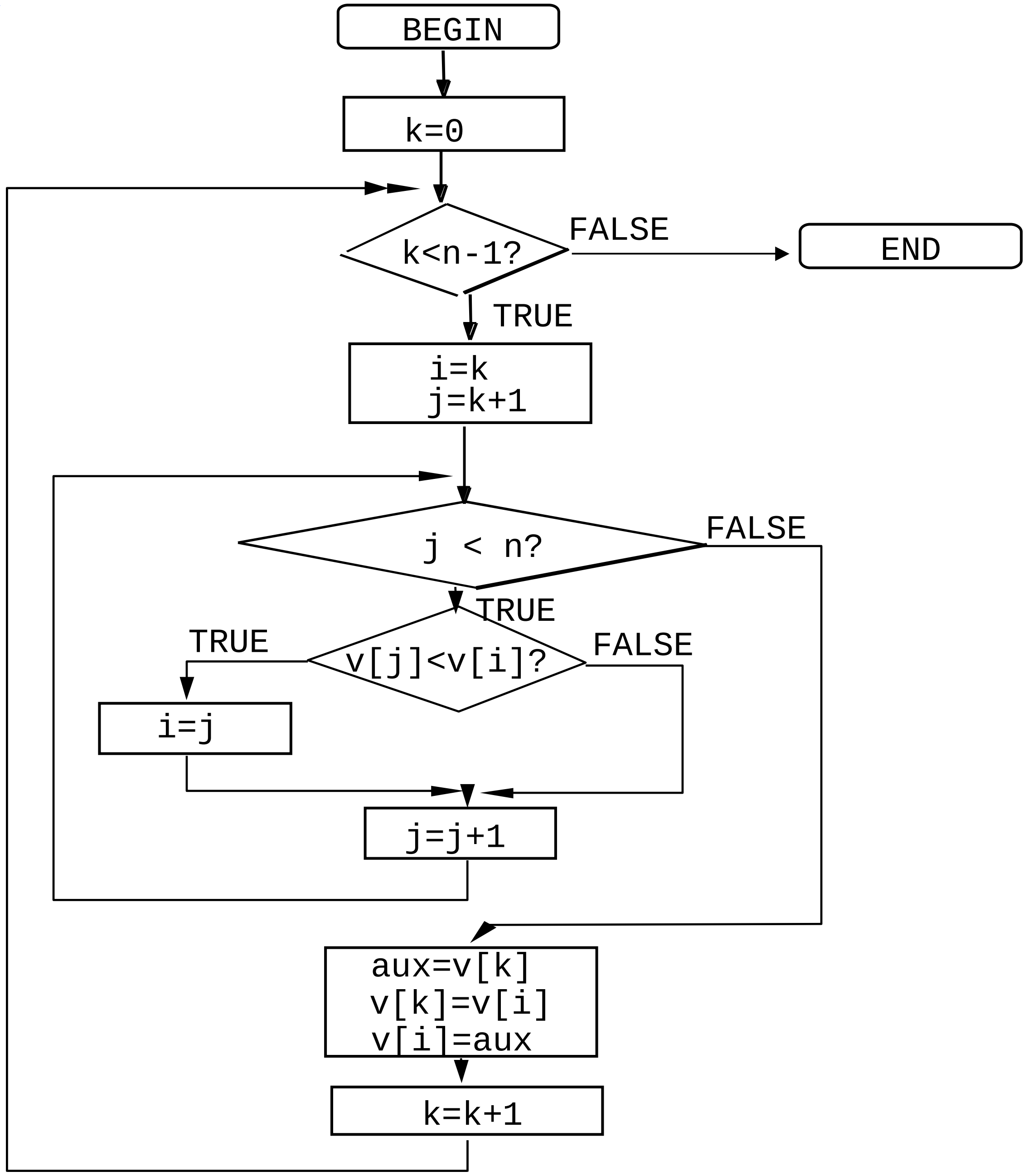
Trace: Direct Selection Sorting Algorithm

🟦 Sorted part

🟧 Unsorted part

Initial array	🟧 84	🟧 69	🟧 76	🟧 86	🟧 94	🟧 91
	🟧 84	🟧 69	🟧 76	🟧 86	🟧 94	🟧 91
	🟦 94	🟧 69	🟧 76	🟧 86	🟧 84	🟧 91
	🟦 94	🟦 91	🟧 76	🟧 86	🟧 84	🟧 69
	🟦 94	🟦 91	🟦 86	🟧 76	🟧 84	🟧 69
	🟦 94	🟦 91	🟦 86	🟦 84	🟧 76	🟧 69
Sorted array	🟦 94	🟦 91	🟦 86	🟦 84	🟦 76	🟦 69

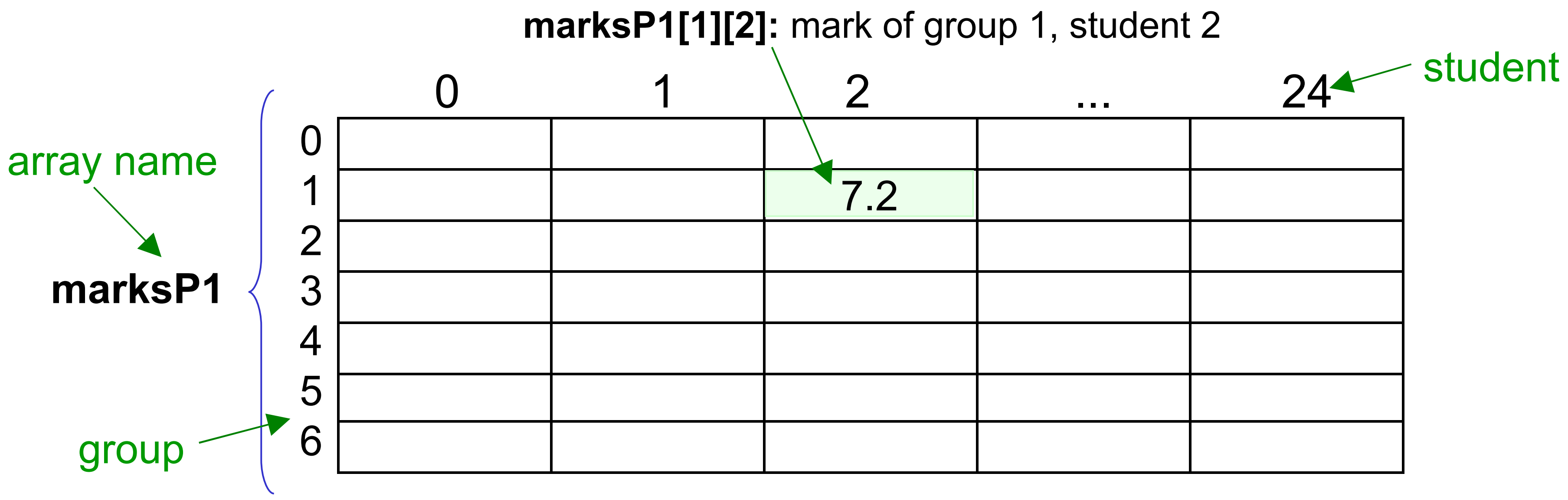
Direct Selection Sorting Algorithm (ascending order)



Topics

Two-dimensional arrays: matrices

- Two indices are needed to access any element
- Example: two-dimensional array to store the marks of 7 groups of P1, each one having 25 students



Two-dimensional array declaration in C language

- First a variable of two-dimensional array type must be declared in order to use it.
- Syntax

```
type array_name [n_elemR] [n_elemC] ;
```

  - **type**: type of each element in the array; every element is of the same type
  - **array_name**: array name
  - **[n_elemR]**: amount of rows (first dimension)
  - **[n_elemC]**: amount of columns (second dimension)

Initialization and access to a two-dimensional array

- As any other variable type, an array must be initialized before being used
- A possible way of **initializing** an array is accessing every element by using two loops (one loop for each dimension) and assigning them a value
- To **access** an array position the following syntax is used:

```
array_name [indexR][indexC] ;
```

  - **array_name**: array_name
  - **[indexR]**: position in the first dimension (row) to be accessed. It is a value in the interval **0 .. n_elemR-1**
  - **[indexC]**: position in the second dimension (column) to be accessed. It is a value in the interval **0 .. n_elemC-1**
- Example:
  - **marksP1[6][24];** //mark of student 24 of group 6
  - **marksP1[6];** //all the marks of group 6 (one-dimensional array made of row 6)

Example 1: two-dimensional array initialization

- If the values are known, a two-dimensional array can be initialized as follows:

```
// two-dimensional array initialization
#include <iostream>
using namespace std;

const int  N_ROWS = 4;
const int  N_COLUMNS = 2;

main () {
        float  matrix[N_ROWS][N_COLUMNS] =    {{3.6, 6.7},
                                               {2.9, 7.6},
                                               {8.9, 9.3},
                                               {1.9, 0.2}};

        Int matrix2 [][N_COLUMNS] = {{3, 6},
                                     {9, 7},
                                     {8, 3},
                                     {1, 0}};
}
```

In C language, it is not necessary to specify the size of the first dimension of the array

Example 2: two-dimensional array initialization

- An array can be initialized by the user entering the data by keyboard, as follows:

```
// two-dimensional array initialization
#include <iostream>
using namespace std;

const int  N_ROWS = 10;
const int  N_COLUMNS = 20;
void initialize_Matrix(float matrix[ ][N_COLUMNS]);

main () {
      float  matrix[N_ROWS][N_COLUMNS];

      initialize_Matrix(matrix);
}
```

```
// procedure to initialize the matrix
void initialize_Matrix(float matrix[ ][N_COLUMNS])
{
      int  i, j;

       for ( i=0 ; i < N_ROWS ; i++ ) {
              cout << "row " << i << ":" << endl;
             for ( j=0; j < N_COLUMNS; j++) {
                   cout << "column " << j << ":";
                   cin >> matrix[i][j];
             }
       }
}
```

In C language, when declaring or defining a module, it is not necessary to specify the size of the first dimension of the array

Topics

Example: two-dimensional arrays (I)

- Calculate and display the average mark for each student. There are 25 students and each one takes 7 subjects.

```
#include <iostream>
using namespace std;

const int N_STUDENTS = 25;
const int N_SUBJECTS = 7;
void  display_Average_Students(float marks[ ] [N_SUBJECTS]);

main () {
   float marks [N_STUDENTS][N_SUBJECTS];

   display_Average_Students(marks);
}

// calculate the average mark for each student and display on the screen
void display_Average_Students(float marks[ ] [N_SUBJECTS]) {
   int i;

   for (int i=0; i< N_STUDENTS; i++)
      cout << "Student " << i << " has an average mark " << calculate_Average(marks[i], N_SUBJECTS) << endl;
}
```

This function was defined in a previous example

Example: two-dimensional arrays (II)

- Given a squared matrix of integers, display in the following order: the elements of the diagonal, the elements of the upper triangle (above the diagonal) and the elements of the lower triangle (below the diagonal), accessing the rows and then the columns

```
// Version 1: the matrix is accessed three times
void Display_Matrix_3 (int  matrix[ ][LMAX])
{   int i, j;

   //  Display diagonal
   for (i=0; i < LMAX; i++)  // access rows
      for (j=0; j < LMAX; j++) // access columns
         if (i == j)
            cout << matrix[i][j];
   //  Display upper triangle
   for (i=0; i < LMAX; i++)
      for (j=0; j < LMAX; j++)
         if (j > i)
            cout << matrix[i][j];
  // Display lower triangle
   for (i=0; i < LMAX; i++)
      for (j=0; j < LMAX; j++)
         if (j < i)
            cout << matrix[i][j];
}
```

```
// Version 2: the matrix is accessed only once
void Display_Matrix_1(int  matrix[][LMAX])
{   int i, j;

   //  Display diagonal
   for (i=0; i < LMAX; i++)   // access rows
      cout << matrix[i][i];

   //  Display upper triangle
   for (i=0; i < LMAX-1; i++)
      for (j=i+1; j < LMAX; j++)
         cout << matrix[i][j];

  //  Display lower triangle
   for (i=1; i < LMAX; i++)
      for (j=0; j < i; j++)
         cout << matrix[i][j];
}
```

Topics

User-defined data types in C language

- The reserved word **typedef** is used to define structured data types (such as arrays and structs)
- It is useful to create new data types and to improve the readability of the programs

```
// Data types definitions
typedef   int     T_integer[20];
typedef   float  T_marks[50];
typedef   char   T_string[30];
typedef   int     T_matrix[3][3];

// Variable declaration
T_marks marks_P1, marks_P2;
T_string  name_student1, name_student2;
T_matrix matrix1, matrix2;
```

```
// If no data types are defined,
// the declaration of variables of type
// array would be:
float   marks_P1[50];
float   marks_P2[50]
char    name_student1[30];
char    name_student2[30];
int       matrix1[3][3];
int       matrix2[3][3];
```

Topics

1. Structured data types
2. Array data type
3. One-dimensional arrays
4. Two-dimensional arrays
5. Type definition using *typedef*
6. **Information sources**

Information sources

Fundamentos de Programación
Jesús Carretero, Félix García, y otros
Thomson-Paraninfo 2007.   ISBN: 978-84-9732-550-9

✓ Capítulo 8

Problemas Resueltos de Programación en Lenguaje C
Félix García, Alejandro Calderón, y otros
Thomson (2002) ISBN: 84-9732-102-2

✓ Capítulo 6

Resolución de Problemas con C++
Walter Savitch
Pearson Addison Wesley 2007.   ISBN: 978-970-26-0806-6

✓ Capítulo 10 (excepto apartado 10.4)
✓ Capítulo 11 (apartado 11.1)