UNIT 6.
STRUCTURED DATA TYPES
PART 1: ARRAYS

Programming

Year 2017-2018

Industrial Technology Engineering

Paula de Toledo

Contents

Structured data types vs simple data types

- Data can have an internal data structured
  - **Unstructured (simple) data types**
    - Data with a single element and a single value
      - *Numbers*: integer , float
      - *Characters*: char
      - Pointers
      - void
  - **Structured data types**
    - Data with an internal structure, not a single element
      - Character strings
      - Arrays and matrices

ARRAYS

Concepto of array

- **Collection of elements** of the same type named with the same global identifier

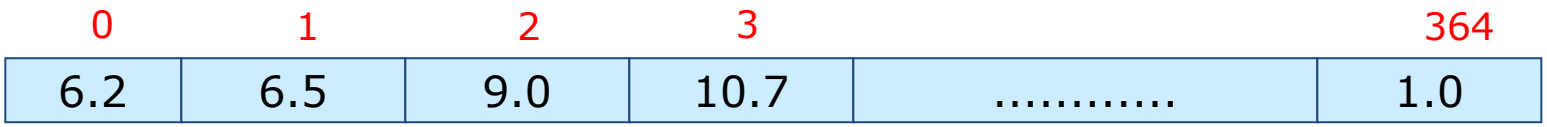
- Individual elements of the array are identified by an **index** corresponding to the position of in the array
  - The index is ALWAYS an integer expression

- Dimensions of an array
  - One-dimensional array: vector
  - More than one dimension: matrix
    - Two-dimension array: table, with rows and columns

Data structure to store the mean temperature of Madrid of each month of the year

One dimension = vector

`float temperature[365]`



- All elements of the same type (float)
- Share a name: temperature
- Each element has a different value
- Each element is identified with and index: [0], [1], ..., [11]
- Use the index to acess the element
  - E.g.: Assign March temperature (third month)

```
temperature [2] = 17.5;
```

element is 0 (not 1)

Data structure to store information regarding a movie theatre

- Value
  (0 free, 1 occupied)

0	1	1	1	1
0	0	1	1	1
1	1	1	1	1
0	0	1	0	0

Dos dimensiones

- All elements share the name (**theatre**)
- Individual elements are identified the indexes
  - In this case index will be row and column

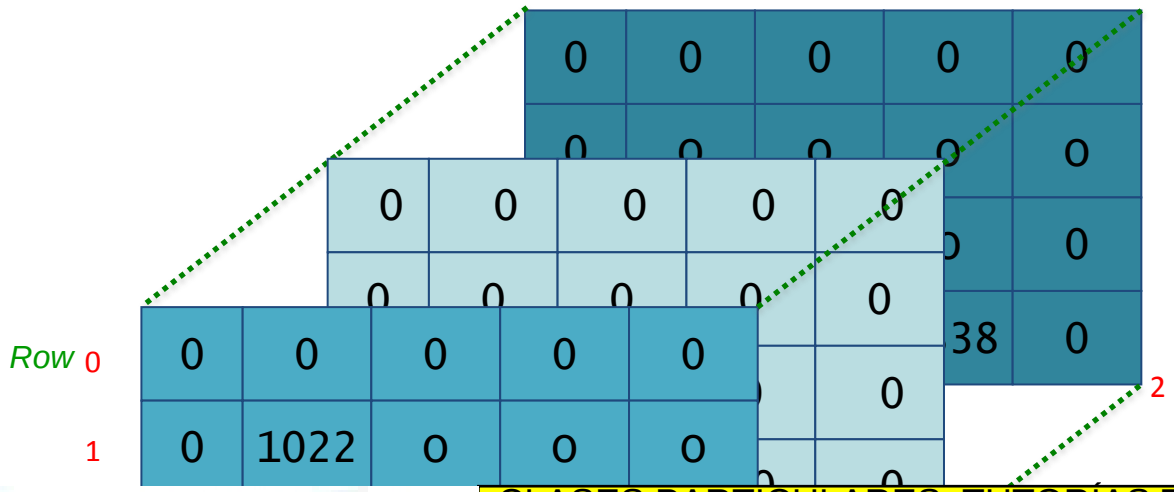
```
theatre [0] [0] = 0;
```

Row *Column*

Data structure to store information from three computer labs – who is using a computer?

- Three labs with four rows and five columns of desks
  - We store the students' id of the student using the computer

```
int lab[3][4][5]
```



Declaring an array:

- What info do we need to specify to declare an array
  - Data type of the array elements
  - Name of the array
  - Number of dimensions
  - Number of elements for each dimension

- This tells the computer how much memory to allocate for this variable
  - n variables of the same type
  - Stored in consecutive cells in memory

- Datatypes of array elements
  - Array elements can be

Declaring a vector

- Declaring a vector (one dimension)
  - Template

    <data type> <array name> [size];
    - Size has to be an integer literal or an integer constant
  - Example:

```
int vectorInt[10];
```

    - array (vector) of 10 values of type int
    - Individual elements identified by the index

5	7	15	1	.....	250
Index: 0	1	2	3	….	9

Declaring a matrix

- More than one dimensions

<data type> <array name> [constant 1] [cte2].... [cteN];

  - Each constant indicates the number of elements in that dimension
- Example 1: Two dimensional array to store an image of size 800 x 600 in black and white

```
int image [800] [600];
```

- Example 2: Three dimensional array to store the initial letter of the name of the people at a movies theatre with rows, 15 columns and three levels

```
char theatre [30] [15] [3];
```

Declaring arrays: arrays of arrays

- A two dimension matrix can be interpreted as a vector whose elements are vectors

```
int image [800] [600];
```

  - Can be seen as a vector of 800 elements, where each element is a vector of 600 elements

- This can be generalized to more dimensions
  - A thee dimensional matrix can be seen as a vector where each element is a two dimensional matrix

Array and pointers

- In C there is a close relationship between arrays and pointers
  - The name of the array is a variable that stores the memory address of the first element of the array
    - i.e. the name of the array is a pointer: the memory address of the first element in the array
  - You can access array elements using the address of the first element + the distance of your element to the first
    - **`elemento3= *(array+2)`**
    - **`// equal to elemento3= array[2]`**
    - We will not use this notation,

Using arrays: element by element

- In C you can't do operations with an array as a block
  - Print, scan, assign, compare – element by element
  - Other languages can handle arrays as a whole

- Array names are pointers ..
  - `int myTable [800] [600]`
  - `printf("%i", myTable );`
    - Prints the memory address of the first element of the array (the pointer)

Assigning values

- Assign value to an element identified by its index

```
theatre [1][3][1]= 1;
marks[25]= 10;
image[0][0]=1;
```

  - Indexes are integer variables, literals or expressions
    - Temperature[11] Temperature[i+j-7]
  - Indexes have to be in the correct range
    - Form 0 to size-1

- It's not possible to assign a value to the array as a whole

```
image =0; //error
```

Initializing

- You can declare the array and then assign initial values element by element

```
int list[5];
list[0]=6;
list[1]=2;
list[2]=7;
list[3]=4;
list[4]=8;
```

- Or you can declare and initialize in one single instruction
  - As with simple datatypes (int a=6;)

Declare + initialize: vectors

- Only exception where you can handle your array as a unit
  - `int list[5]={6,2,7,4,8};`
  - You can omit the number of elements only if you initialize
    - The number of elements will be used by the computers to assign vector size
    - `int list []={6,2,7,4,8};`

Declare + initialize: matrix

- Declaring and initializing arrays of more than one dimension
  - Initialized as a vector of vectors

```
int list [3][2]={
                    {0,1},
                    {10,11},
                    {20,21}
                    };
```

  - list is a vector of three elements, where each element is a vector of two elements
  - You can omit the size of the first dimension…

```
int list [][2]={
                    {0,1},
                    };
```

Printing arrays

- One element
  - printf ("This is the colour of the third pixel in the fourth column %i:", image [2][3]);

- Whole array (remember elements one by one)

```
int array[4][2];
int i, j;
for (i=0; i<4; i++) {
  for (j=0; j<2; j++) {
    printf ("%i\t", array[i][j]);
  }
  printf("\n");
```



```
Presione una tecla para continuar . . .
```

Reading arrays:

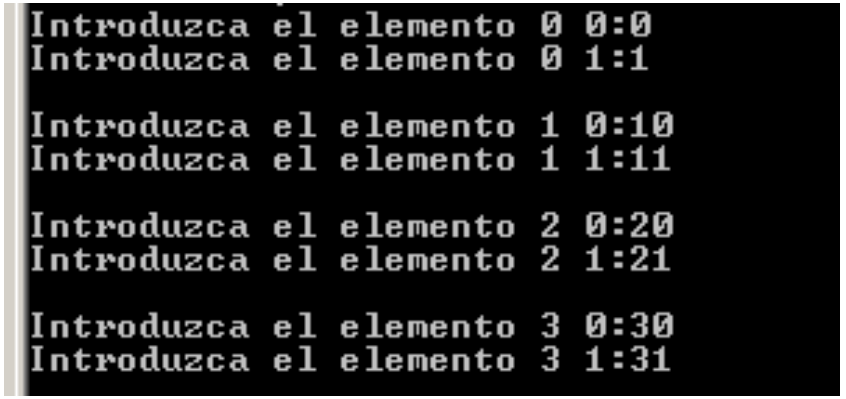
- One element

```
printf ("Enter the colour of the third pixel in the
second column\n");

scanf("%i", & image[2][1]); // use &, we read an int
```

- Whole array

```
int myArray[4][2];
for (i=0; i<4; i++) {
    for (j=0; j<2; j++) {
        printf ("Enter element %i %i:", i, j);
}
```



6.1.2 STRINGS

String variables

- Strings are a vector where the elements are chars
- But with one distinctive feature
  - An extra char is added at the end of the string
  - This extra char is the null character , '\0' , whose ASCII code 0
  - This null character is added automatically by the computer

- Strings are declared and used as vectors with some distinctive features
  - Declare and initialize as a vector or ..
    - Can also be initialized to a string literal ("Hello")
  - Assign and compare as vector (element by element) or..

Declaring and initializing strings

- Declaring a vector of char and a string is identical
  - `char MyVector [LENGTH];`
  - `char MySstring [LENGTH];`
- Declaring and initializing
  - Can be initialized as vectors of chars but adding the null char at the end
    - Vectors of chars
      - `char vector_hello1 []= {'H', 'o', 'l', 'a' };`
      - `char vector_hello2 [4]= {'H', 'o', 'l', 'a' };`
    - String
      - `char string_hello1 []= {'H', 'o', 'l', 'a', '\0' };`
      - `char string_hello2 [5]= {'H', 'o', 'l', 'a', '\0' };`
  - Strings can also be initialized to string literals
    - If size is not specified an extra space is allocated for the null character
      - `char string_hello3 []= "Hola" ;`
      - `char string[1024]= "A random string in C";`

- Not to assign a value to a variable

Assigning values to strings: strcpy

- Assign operator not working with strings
  - It would copy a pointer into a pointer, not the strings
  - MyString = myName; // ¡¡no!!
  - MyString = "hola"; // ¡¡no!!
- Instead use function *string copy* **strcpy**
  - Library string.h
    - `strcpy (MyString, myName);`
    - `strcpy (MyString, "Paula");`
  - Arguments: two string variables or a string variable and a string literal
  - Assigns the value of the second string to the first
- Alternative?: copy element by element

```
string [4]='\0';
```

Comparing strings: strcmp

- Strings can't be compared as simple variables
  - If (myString == myName)
  - You'd be comparing pointers!
- Use function string compare: strcmp

```
int main(void) {
int result;
char example1[50], char example2[50];

// assign values to our strings
strcpy(example1, "C programming is useful");
strcpy(example2, "C programming is fun");
// Compare the two strings provided
result = strcmp(example1, example2);
if (result == 0)
```

Joining strings (concatenate): strcat

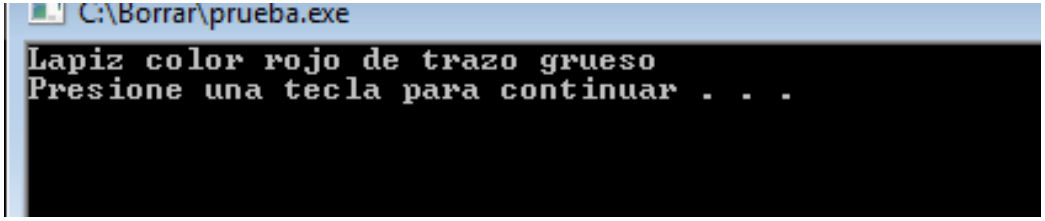
- strcat concatenates two strings

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

int main(void){
    char color[]="rojo";
    char grosor[]="grueso";

    char descripcion[1024];

    strcpy(descripcion, "Lapiz color ");
    strcat (descripcion, color);
    strcat(descripcion, " de trazo ");
    strcat (descripcion, grosor);
```



Example: strcpy and strcat

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

#define LEN   80

int main (void)
{
  //Variable declaration
  char name[LEN ];
  char surnames[ LEN ];
  char fullName[LEN *2];

  printf ("name?:\n");
  scanf("%s", name);

  printf ("surname(s)?:\n");
```

```
// 1. Initialize to empty string
  strcpy (fullName, "");

  //2. concatenate name
  strcat(fullName, name);

 // 3. concatenate blank space
  strcat(fullName, " ");

 // 4. concatenate surnames
  strcat(fullName, surnames);

 // 5. Print full name
printf("Your full name is : %s\n",
```

Finding the length of a string: strlen

```
//strlen – string length – gets the length of a string
tam=strlen(MyString) ;
```

- Example:
  - Write your own code to find a string length:
    - Strings end with a null character ('\0')
    - This is used to find the length of the string

```
i=0;
while (MyString[i] !='\0')
        i++;
tam = i; // or i+1 if we want to count the null char
```

Reading and printing strings

- As seen in Unit 3
  - Format specifier for string variables and string literals %s
  - When using scanf with strings, address of operator not needed, '&'
    - Function scanf takes a pointer as parameter (pass by reference)
    - The string name is already a pointer

```
char str[100];
printf ("Enter string: ");
scanf ("%s", str);
printf ("String is: %s", str);
```

ARRAYS AS PARAMETERS OF FUNCTIONS

Arrays as parameters of functions

- A function can take an array as parameter
- But a function can't return an array using return
  - return used only with simple data
- A function can modify an array if it is passed to it as a parameter

- Arrays are always passed to functions by reference
  - .. The name of the array is a pointer (memory address of the first element)

- Syntax

Matrices and vectors as parameters : syntax

- Declaration (prototype) and definition (header) – formal parameters
  - Use name of the array, type of data and dimensions
    - same syntax used when you declare the array

```
int myFunction (int matrix[ROW][COL]){
int myFunctionVect (int vector[SIZE]){
```

- Call to the function – actual parameters
  - Use only the name of the array

```
resu = myFunction (myMatrix);
Resu = myFunctionVect(myVector);
```

- Size of the array in the main and in the function has to be the same

Example 1

Find the maximum of a two dimensional matrix (a table)

```
#include <stdio.h>

#define ROW 2
#define COL 3

int findMax (int a[ROW][COL]);
void printMatrix(int a[ROW][COL]);


int main(void) {
  int matrix[ROW][COL];
  int i, j;
  // We assign some values to the matrix elements
 for (i=0; i<ROW; i++)
     for (j=0; j<COL; j++)
       matrix [i][j]=i+j;
```


```
  return 0;
```

```
int findMax(int a[ROW][COL]) {
    int i, j, max;
    max=a[0][0];
    for (i=0; i<ROW; i++)
        for (j=0; j<COL; j++)
            if (max<a[i][j])
               max=a[i][j];
    return max;
}

void printMatrix (int a[ROW][COL]) {
   int i, j;
   for (i=0; i<ROW; i++){
        for (j=0; j<COL; j++)
               printf("%i\t",a[i][j]);
        printf ("\n");
   }
```

- Write a function that takes a table (two dimensions) as parameter and adds up all the elements

Example 2

Function to read a vector

```
#define TAM 5

void getVector(int a[]);

int main(void) {
  int v[TAM];
  printf("Enter the vector elements\n");
  getVector(v);
  return 0;
}

void getVector(int a[]) {
  int i;
  for (i=0; i<TAM; i++)
    scanf("%i", &a[i]);
  return;
```

Omitting the size of the first dimension to work with vectors of different sizes

- You can omit the size of the **first** dimension when declaring a function
  - This code will work with vectors/matrices of different sizes

```
int myFunction (int matrix[][COL]){
int myFunctionVect (int vector[]){
```

- Call to the function remains the same

```
resu = myFunction (myMatrix);
Resu = myFunctionVect(myVector);
```

- **Only the first** dimension: It's not possible to omit the size of the second, third… dimensions
  - We don't see this – but we introduce it in Unit 8

Example 3

Function to read vectors of different sizes

```
void getVector(int a[], int longitud);

int main(void) {
  int v[256];
  printf("Enter the vector elements\n");
  getVector(v, 256);
  return 0;
}

void getVector (int a[], int len){
  // arguments: a--vector to read
  //            len—length of the vector
int i;
  for (i=0; i<len; i++)
    scanf("%i", &a[i]);
```

New version of function getVector can read vectors of any length

Vector a is a parameter, but declared without specifying its length

The actual length is

Example 4

Program that reads two vectors and copies them to a third vector

```
#include <stdio.h>
#define L1 5
#define L2 3

void copyVectors (int v1[], int v2[], int v3[], int len1, int len2);
void getVector(int v[], int vectorLength);
void printVector(int v[], int vectorLength);
```

```
int main(void)
{
  int va[L1], vb[L2], vc[L1+L2];

  printf("Enter values for vector 1\n");
  getVector(va, L1);
  printVector(" Enter values for vector 2\n");
  readVector(vb, L2);

  copyVectors(va,vb,vc, L1, L2);

  printf("The vectors you entered are\n");
  printVector(va, L1);
  printVector(va, L2);
  printf("And the two vectors together are");
  printVector(vc, L1+L2);
```

```
void copyVectors (int v1[], int v2[], int v3[], int len1, int len2){
  int i;
  for (i=0; i< len1+len2; i++){
      if (i<len1)
            v3[i]=v1[i];
      else
            v3[i]=v2[i-t1];
      }
    return;
 }
void getVector(int v[], int vectorLength){
   int i;
   for (i=0; i< vectorLength; i++)
         scanf("%i", &v[i]);
   return;
}

void printVector(int v[], int vectorLength){
    int i;
    for (i=0; i< vectorLength; i++)
```

const keyword in function arguments

- **const** keyword can be used to force the complier to check that a given argument is not changed within the function
  - if by mistake you try to modify a const argument , the compiler will throw an error
- Uses
  - Extra security check when passing parameters by value
    - `int findMinimum (const int n1, const int n2, const int n3){`
  - Extra security check for arrays
    - Arrays are always passed by reference
    - Use const for input data if you want to make sure you don't modify them

```
int sumVector(const int v[LEN]){
```

Example 4 bis

Modify the headers in Example 4 using const as an extra check for input vectors

```
#include <stdio.h>
#define SIZE1 5
#define SIZE2 3

void copyVectors (int const v1[], const int v2[], int v3[],
                                  int len1, int len2);
void getVector(int v[], int vectorLength);
void printVector(int const v[], int vectorLength);
```

UNIT 6.
STRUCTURED DATA TYPES
PART 1: ARRAYS

Programming

Year 2017-2018

Industrial Technology Engineering

Paula de Toledo