UNIVERSIDAD DE MURCIA

Introducción a la Programación

Grado en Ingeniería Informática

14. Tablas

Bloque 4. Tablas
14. Tablas
Dr. Isidro Verdú

Tablas / Arrays

Tablas / Arrays

Operaciones frecuentes

- ✓ Acceso directo a un dato
- ✓ Recorrido de todos los elementos, por orden, para aplicar una misma acción
- ✓ Búsqueda de un elemento que cumpla cierta propiedad

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

T_letras = tabla [0,22] de caracteres
letras : T_letras

```
p ← letras[8]

para (i ← 0, i<23, i ← i+1)
  Escribir (letras[i]);

i ← 0;
encon ← false
mientras ( i<23 and (not encon))
  Si (letras[ i ] = p) encon ← true
  i ← i+1;
```

3

Arrays en C/C++

Array (tabla, vector, matriz) :
estructura de datos compuesta para agrupar datos del mismo tipo

```
main()
{
  int v[5];

  v [ 0 ] = 0;
  v [ 1 ] = 9;
  v [ 2 ] = 29;
  v [ 3 ] = 17;
  v [ 4 ] = 33;
}
```

Define un array formado por 5 números enteros.

Los arrays en C siempre tienen índices desde 0 hasta n-1

v[0], v[1], … v[4]

Mete los valores en v

V_0	0
V_1	9
V_2	29
V_3	17
V_4	33

4

Arrays en C/C++

Se puede acceder a los datos así

```
int v[10],a,b;

a=4; b=8;

v [ a ] = b;
v [ ++a ]= 4*b+2;
v [ 2*a-b ] = 12;
v [ 6 ] = v [ a-1 ];
v [ a ] = v [ a+1];
b = v [ a ];

if (v [ a ] > v [ a+1])
    v [ a ]=0;
```

5

Arrays en C/C++

```
#define Tam 5
main()
{
      int a, v [ Tam ];

      v [ 0 ] = 0;
      v [ 1 ] = 9;
      v [ 2 ] = 29;
      v [ 3 ] = 17;
      v [ 4 ] = 33;
}
```

Establecer con un ***define*** el tamaño del array es una buena práctica.

```
#define Tam 5
main()
{
      int v [Tam ],a;
      …
      a=7;
      v [ a ] = 10;
}
```

¡Ojo! C no comprueba indices de arrays fuera de rango. ¡ escribe en otra zona de memoria !

6

Arrays en C/C++

```
#define M 3
#define N 2
main()
{
      int a [ M ] [ N ];

      a[0][0] = 1;   a[0][1] = 7;
      a[1][0] = 8;   a[1][1] = 8;
      a[2][0] = 2;   a[2][1] = 9;
}
```

Esto es un ***array bidimensional*** (una tabla, una matriz)

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{bmatrix}$$

Se puede usar para guardar desde una matriz matemática hasta una imagen.

7

Arrays en C/C++

```
#define M 10
#define N 12
#define P 5
#define L 20

float a [M] [N] [P], coste;
int b [M] [N] [P] [L], i, j, k;

. . .

a [ i ] [ j ] [ k ] = coste*12.0;
b [7] [10] [9] [0] = 20;
```

Y de 3 dimensiones, y de 4, y de…

8

Arrays en C/C++

```
main()
{
        int a[5];
        int b[5] = {2,7,12,0,1};

        . . .
        a[0]=2;   a[1]=7;
        a[2]=12; a[3]=0;
        a[4]=1;
        . . .
}
```

Se puede inicializar un array de esta forma

Pero esta es mejor

9

Arrays en C/C++

```
main()
{
        int a[2][3] = {0,1,2,10,11,12};
        int b[2][3] = { {0,1,2} , {10,11,12} };

        . . .
        sentencias;
        . . .
}
```

Los de dos dimensiones se inicializan de cualquiera de estas dos formas; mejor la segunda.

0	1	2
10	11	12

10

Arrays en C/C++

```
#define Tam 5
main()
{
   int  v [Tam ];

   . . .

   printf (“Introduce 5 datos separados por espacios : ”);
   scanf(“%d %d %d %d %d”, &v[0], &v[1],&v[2],&v[3],&v[4]);
}
```

De este modo puedo cargar el array ***v*** de datos del teclado

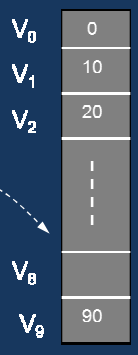
11

Arrays e iteraciones

```
#define Tam 10
main()
{
      int v[Tam], i;

      for (i=0; i<Tam; i++)
            v [ i ] = i * 10;
}
```

Mete los valores en el array.



12

Arrays en C/C++

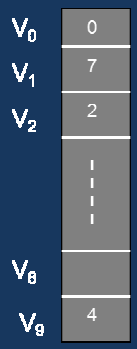
Mete valores aleatorios [0..10] en el array

Y saca en pantalla los datos y la media de los que son pares

```
#define Tam 10
main()
{
     int v[Tam], i, n;  float media;

     media=0; n=0; srand(time(NULL));
     for (i=0; i<Tam; i++)
        v [ i ] = rand() % 11;
     ......
     ......
     for (i=0; i<Tam; i++)
       if (v[i]%2==0)
        {
          media=media+v[i];
          n++;
        }

    media=media/n;
    printf("media %.2f",media);
}
```



13

Arrays en C/C++

```
#define M 1000
#define N 2000

main()
{
      int a [ M ] [ N ], i, j;

      for (i=0; i< M; i++)
            for (j=0; j< N; j++)
                  a [ i ] [ j ] = i* j* 10;
}
```

Recorre un array 2D

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{pmatrix}$$

Se puede usar para guardar desde una matriz matemática hasta una imagen.

14

Carga de Arrays

```
#include <stdio.h>
#define Tam 10
main()
{
        int i, v [ Tam ];

        for (i=0; i< Tam; i++)
        {
                printf (“Introduce %d :”, i);
                scanf(“%d”, &v[ i ] );
        }
}
```

Esta iteración carga el array ***v*** de datos de usuario

15

Carga de Arrays

```
#include <stdio.h>
#include <stdlib.h>
#define Tam 1000
main()
{
        int i; float v[ Tam ];

        srand(time(NULL));
        for (i=0; i< Tam; i++)
          v[i]=rand()/(float)RAND_MAX;
}
```

¡ Y ésta de valores aleatorios entre 0 y 1 !

¡ Y ésta un entero entre 5 y 15 !

```
v[i] = (rand() % 11)+5;
```

Y ésta un real entre 5 y 15

```
v[i]= ( (rand()/(float)RAND_MAX)*10 ) +5;
```

16

Arrays e iteraciones (while)

```
for (i=0; i< Tam; i++)
{
        printf (“Introduce %d :”, i);
        scanf(“%d”, &v[ i ] );
}
```

Por supuesto puedo usar **`while`** en vez de **`for, especialmente en búsquedas`**

```
i=0; flag=true;
while (i< Tam && flag)
{
        if (v[ i ]== valor ) flag=false;
        else i++;
}
```

Y **`do while`**

```
i=0; flag=true;
do
{
        if (v[ i ]== valor ) flag=false;
        else i++;
} while (i< Tam && flag)
```

17

Paso de arrays entre funciones

18

Parámetros: Arrays

A una función se le pasa con normalidad ***una casilla*** de un array; paso por valor.

```
#include <stdio.h>

float media (int a, int b);

main()
{
  int a[4]={1,3,4,5};
  printf("Media: %f\n", media(a[0],a[1]) );
}

float media (int v1, int v2)
{
  float m;
  m=(v1+v2)/2.0;
  return m;
}
```

Pero si paso el array completo la cosa cambia:

¡ Los arrays se pasan siempre por referencia ! . . /. .

19

Parámetros: Arrays

Hace la media de un vector

Al pasarle **a** *estoy pasando la dirección de memoria de inicio del array. Esto es paso por referencia*

Esta función además de hacer la media, pone el array a cero, para que main() lo reciba así.

```
#include <stdio.h>
#define MAX 4

float media (float v[MAX]);

main()
{
  float a[MAX]={1,3,4,5};
  printf("Media: %f\n", media(a) );
}

float media (float v[MAX] )
{
  int i; float m=0;
  for (i=0;i<MAX;i++)
    {m=m+v[ i ]; v[ i ]=0;}
  m=m/MAX;
  return m;
}
```

20

Parámetros: Arrays

Esta forma es más elegante. La función queda lista para ser utilizada en cualquier otro programa.

Está permitido no dimensionar el array en el parámetro. Tomará la dimensión que reciba.

(solo para unidimensionales)

Se pasa el tamaño como parámetro, para poder controlar el array.

```
#include <stdio.h>

float media (float v[ ], int lon);

main()
{
   float a[4]={1,3,4,5};
   printf("Media: %f\n", media(a,4) );
}

float media (float v[ ], int lon )
{       int i; float m=0;

        for (i=0; i < lon; i++)
                {m=m+v[ i ]; v[ i ]=0;}
        m=m/lon;
        return m;
}
```

21

Parámetros: Arrays

En esta otra forma un valor de control indica que es el fin del array.

Ya no hace falta pasar el tamaño como parámetro

*Itera hasta encontrar la marca **-1** de fin de array.*

Es necesario que haya un valor que solo sirva para hacer de marca.

```
#include <stdio.h>

float media (float a[ ]);

main()
{
   float a[5]={1,3,4,5,-1};
   printf("Media: %f\n", media(a) );
}

float media (float a[ ])
{       int i=0; float m=0;

        while(a[ i ] != -1)
           {m=m+a[ i ]; a[ i ]=0;i++}
        m=m/lon;
        return m;
}
```

22

Variables globales y locales

23

Variables locales y globales

```
#include <stdio.h>

funcion1 (int a)
{
  int i,j;
  sentencias…
}

funcion2 ()
{
  int i,j,k;
  sentencias…
}

main()
{
  int a,i,j;
  sentencias…
}
```

Las variables declaradas dentro de una función solo son ***visibles*** en esa función.

Además, cuando la función acaba, sus valores se pierden *(salvo que sean parámetros por referencia o que sean devueltos por la funcion)*

Son variables locales

(a la función en la que están)

Puedo poner variables I,j,k en varias funciones, y sus valores no se mezclan.

24

Variables locales y globales

Se pueden definir **variables globales**, que son visibles en todo el programa y no pierden sus valores durante toda la ejecución.

Es recomendable que sean las mínimas

Dificultan la lectura del código de la función

Las funciones dejan de ser exportables a otros programas si usan variables globales.

A veces son muy útiles y prácticas

```
#include <stdio.h>

int mes, h;
float cantidad;

funcion1 (int a)
{  int i,j;
   sentencias…
   mes=…
}
funcion2 ()
{  int i,j,k;
   i=….* mes
   sentencias…
}
main()
{   int a,b,j;
    mes=…
    sentencias…
}
```

25

Variables locales y globales

¡Ojo! Si se definen variables locales con el mismo identificador que variables globales, el compilador no da errores, pero el programa no hará lo que quieres.

```
#include <stdio.h>

int i , j;

funcion1 (int a)
{    int i , k;
   sentencias…
   for ( i=0 …
}

main()
{   int a,b,j;
    i=10;
    funcion1(b);
    …
}
```

26

Orden de un programa C/C++

27

Primero includes y defines

Luego prototipos (declaraciones)
También #include "declaraciones.h"

Variables globales

main()

declaraciones de variables

Cuerpo de la función

Definición de funciones

Este es el orden más correcto en un programa

```
#include <stdio.h>
#define PI 3.1416

float media (int a, int b);
int suma (int a, int b);
void empezar (void);

int mes, h;
float cantidad;

main()
{
  int a,b;
  scanf("%d %d",&a,&b);
  printf("Media: %f\n", media(a,b));
}

float media (int a, int b)
{return ((a+b)/2.0);}

int suma (int a, int b)
{return (a+b);}

…
….
```

28


- - -
29