Curso C

Estructuras/Vectores/Matrices

Vectores o Arrays o Matrices

- Una matriz es una estructura homogénea, compuesta por varios elementos, todos del mismo tipo y almacenados consecutivamente.
- El nombre de la matriz representa su dirección
- Vector o array -> Caso especial de matrices Unidimensionales

  Tipo nombre [tamaño] -> int m[10];
  struct ordenador hacer[50];

  El acceso a los vectores a arrays.

  m[0] -> primera posicion vector m

```
int m[100], k=0;
k=50;
m[k+1]=34;
```

Ojo de no sobrepasar los límites de los vectores.

```
#define N 100
//...
int m[N], k=0, a=0;
//...
if (k>=0 && k<N-1)
  m[k+1]=m[k];
else
  printf("indice fuera de limites");
```

Ejercicio

Programa que introduzca la nota de los alumnos (100) en un array y calcule la nota media de todos.

```
/******************** Nota media del curso ********************/
/* notas.c
 */
#include <stdio.h>

#define N_ALUMNOS 100  // número máximo de alumnos

main()
{
  float notas[N_ALUMNOS]; // matriz notas
  int i = 0;              // índice
  int nalumnos = 0;       // número real de alumnos
  float suma = 0;         // suma total de todas las notas medias

  do
  {
    printf("Número de alumnos: ");
    scanf("%d", &nalumnos);
  }
  while (nalumnos < 1 || nalumnos > N_ALUMNOS);

  // Entrada de datos
  for (i = 0; i < nalumnos; i++)
  {
    printf("Alumno número %3d, nota media: ", i+1);
    scanf("%f", &notas[i]);
  }

  // Sumar las notas
  for (i = 0; i < nalumnos; i++)
    suma += notas[i];

  // Escribir resultados
  printf("\n\nNota media del curso: %5.2f\n", suma / nalumnos);
}
```

Vectores o Arrays o Matrices

Inicializar un array/vector.

Si la variable es GLOBAL, sus elementos se inicializan automaticamente (0, \0, NULL)

Si es LOCAL, no se inicializar, contendran BASURA (lo que hubiese en memoria)

```
float temperatura[6]={10.2F,12.3F,3.4F,14.5F,15.6F,16.7F};
```

Tamaño igual o mayor que numero de valores especificados. Si mayor, resto de valores se inicializan a 0,\0,NULL;

Siempre que se inicializa una matriz al definirla, podemos omitir el tamaño. También cuando se declara como parámetro formal de una función

```
void VisualizaMatriz(int [], int);
```

```
# include <stdio.h>
void VisualizarMatriz(int [], int);

main(){
  int x=[]={10,20,30,40,50};
  VisualizarMatriz(x,5);
}

void VisualizarMatriz(int m[], int n){
  int i=0;
  for (i=0; i<n; i++)
     printf("%d ", m[i]);
}
```

Matrices multidimensionales

```
tipo nombre[numero][numero2][…];

int m[2][3]; -> contiene 6 elementos

int m2[2][3]={
                    {10,20,30},
                    {11,21,31}
                };
```

Los elementos son colocados por filas consecutivas en memoria.

El nombre de la matriz representa su dirección:

m, m[0], &m[0][0] son la misma dirección

Las de dos dimensiones las representamos por tablas

```
int x=3;
m[1][2]=x;
m[0][1]=m[1][2]
```

Ejercicio

Programa que introduzda datos en una matriz m de dos dimensiones y luego de el resultado de sumar las filas de la matriz.

```
/********* Suma de las filas de una matriz bidimensional
*********/
#include <stdio.h>
#define FILAS_MAX 10  // número máximo de filas
#define COLS_MAX  10  // número máximo de columnas
main(){
  float m[FILAS_MAX][COLS_MAX];  // matriz m de dos dimensiones
  float sumafila;      // suma de los elementos de una fila
  int filas, cols;     // filas y columnas de la matriz de
trabajo
  int fila, col;       // fila y columna del elemento accedido

  do  {
    printf("Número de filas de la matriz:    ");
    scanf("%d", &filas);
  }
  while (filas < 1 || filas > FILAS_MAX);

  do  {
    printf("Número de columnas de la matriz: ");
    scanf("%d", &cols);
  }
  while (cols < 1 || cols > COLS_MAX);

  // Entrada de datos
  printf("Introducir los valores de la matriz.\n");
  for (fila = 0; fila < filas; fila++)
    for (col = 0; col < cols; col++)    {
      printf("m[%d][%d] = ", fila, col);
      scanf("%f", &m[fila][col]);
    }

  // Escribir la suma de cada fila
  for (fila = 0; fila < filas; fila++)  {
    sumafila  = 0;
    for (col = 0; col < cols; col++)
      sumafila += m[fila][col];
    printf("Suma de la fila %d = %g\n", fila, sumafila);
  }
printf("\nFin del proceso.\n");
}
```

Cadenas de caracteres

```
char cadena[]="Hola";  // se inserta \0 como fin de cadena
char cadena[]={'a', 'b', 'c', 'd', '\0'};
char cadena[]={97, 98, 99 100, 0};
char cadena[20];
char cadena[40]={0}; // todos los elementos se inician a \0

scanf("%s",cadena);
```

Ojo, no se pone & delante de la variable.

Si introducimos "Hola esto es una cadena", la sentencia anterior solo guardaría "Hola"

Si no se asigna el \0 un printf puede escribir fuera de límites y ocasionar error.

Usar gets (cadena) // lee la linea completa hasta encontrar \n

Leer y escribir cadenas de caracteres

```
char nombre[41];
scanf(“%s”, nombre);
printf(“%s\n”,nombre);
  //Recordar el espacio es
  separador

modificador de formato de
  scanf
%[caracteres]
  -> scanf(“%[a-zA-
  Z]”,nombre);
%[^caracteres]
  -> scanf(“%[^\n]”,nombre);

scanf(“%[^\n]”,nombre)  igual
  que usar la función gests
```

```
/* getchar.c */
#include <stdio.h>
#define LONG_CAD 41

main(){
  // matriz de LONG_CAD caracteres
  unsigned char cadena[LONG_CAD];
  int i = 0, car;
  printf("Introducir un texto: ");
  while ((car = getchar()) != '\n' &&
         i < LONG_CAD-1)  {
    cadena[i] = car;
    i++;
  }
  // Finalizar la cadena con 0
  cadena[i] = 0; //IMPORTANTE

  printf("Texto introducido:  %s\n", cadena);
  printf("Longitud del texto: %d\n", i);
}
```

Funciones: gets y puts

```
#include <stdio.h>
char *gets(char *var);
```

var será el buffer donde queremos que entre el texto.
Terminará en \0
NO introduce el \n

```
char nombre[41];
gets(nombre);
printf(“%s\n”);
```

```
/* gets.c
 */
#include <stdio.h>

main(){
 //para almacenar el valor retornado por gets
  char *c = NULL;
  char texto[40];

  printf("Introducir líneas de texto.\n");
  printf("Para finalizar introducir la marca
EOF\n\n");
  // Leer la primera línea de texto
  c = gets(texto);
  while (c != NULL)  {
    // Operaciones con la línea leída
    // ...
    // Leer otra línea de texto
    c = gets(texto);
  }
}
```

```
/* puts.c */
#include <stdio.h>
main(){
  char *c = NULL; // para almacenar el valor retornado por gets
  char texto[80];

  printf("Introducir una línea de texto:\n");
  c = gets(texto);
  printf("\nEl texto introducido es:\n");
  puts(texto);    // equivalente a: printf("%s\n", texto);
  puts("\nSe escribe por segunda vez:");
  puts(c);
}
```

Problemas - fflush

\n queda en el buffer despues de scanf o getchar

si se vuelve a usar **scanf** o **getchar** **\n** entra en la siguiente lectura.

con gets, si en la cadena a leer hay un **\n** pendiente también entrará y se retornará una cadena vacia.

Usar **fflush** o **gets(dummy)** para limpiar el buffer de entrada antes de usar **gets**.

```
/* gets.c
 */
#include <stdio.h>

main(){
 //para almacenar el valor retornado por gets
  char *c = NULL;
  char texto[40];

  printf("Introducir líneas de texto.\n");
  printf("Para finalizar introducir la marca
EOF\n\n");
  // Leer la primera línea de texto
  c = gets(texto);
  while (c != NULL)  {
    // Operaciones con la línea leída
    // ...
    // Leer otra línea de texto
    c = gets(texto);
  }
}
```

Ejercicio:

Programa que convierta las minúsculas en mayúsculas del texto entrado por el usuario.

```
/*********** Conversión de minúsculas a
mayúsculas ***********/
/* strupr.c */
#include <stdio.h>

#define LONG_MAX 81       // longitud máxima
void MinusculasMayusculas(char str[]);

main(){ // función principal
char cadena[LONG_MAX];
  int i = 0;

  printf ("Introducir una cadena: ");
  gets(cadena);
  MinusculasMayusculas(cadena);
  printf ("%s\n", cadena);
}

/*****************************
Función MinúsculasMayúsculas
*****************************/
// Convierte minúsculas a mayúsculas
void MinusculasMayusculas(char str[]){
  int i = 0, desp = 'a' - 'A';

  for (i = 0; str[i] != '\0'; ++i)
    if (str[i] >= 'a' && str[i] <= 'z')
      str[i] = str[i] - desp;
}
```

Cadenas de caracteres

- Comprobación de caracteres

  char car = getchar();

isalnum (car)	Alfanumerico
isalpha (car)	Alfabético
iscntrl (car)	Carácter control
isdigit (car)	Dígito
isgraph (car)	Imprimible?
islower (car)	Minusculas
isupper (car)	Mayusculas
ispunct (car)	Carácter puntuacion
isspace (car)	Espacio
isxdigit (car)	Digito hexadecimal

Cadenas de caracteres

- Búsqueda de números
  - Cuando leemos numeros como cadena de caracteres, hay que convertir dicha cadena en su equivalente numerico
    - "-23", cadena de 3 elementos.
  - Funciones "stdlib.h"
    - atoi (cadena) convierte a int
    - atol (cadena) convierte a long int
    - atof (cadena) convierte a float

Cadenas de caracteres

- Funciones manejo cadena de caracteres

strlen (cadena)	Longitud cadena
strcat (cad1, cad2)	Concatena 2 cadenas
strncat(cad1,cad2,n)	Concatena n caracteres
strcmp(cad1,cad2)	Compara cadenas
strncmp(cad1,cad2,n)	Compara n caracteres
strchr(cad, car)	Busca carácter en cadena
strrchr(cad,car)	Idem pero empezando por final
strstr(cad1,cad2)	Busca subcadena
strtod(cad)	Convierte cadena a float
strtol(cad)	Convierte a long
strcpy(cad1,cad2)	Copiar cadenas
tolower(cad), toupper(cad)	Convierte a minusculas/mayusculas
sprintf	

Tipos enumerados

- enum etiqueta {lista enumerada};
  - Ejemplo:

```
enum codigo_color {negro, marron, verde};
Main(){
enum codigo_color micolor;
micolor = marron;
switch(micolor){
 case negro:
        break;
}
}
```

- Funcionan como constantes (int), es equivalente a los #define negro 0
- Podemos asignar valores:
  - Enum numeros{primero=20,segundo=50};

Definición tipos de datos

- typedef tipo nuevo_tipo
  - Ej: typedef char LETRA
  - En main definiriamos LETRA carácter; en vez de char carácter;

  - Ej. typedef enum estado_luces{rojo,verde, off};
  - int comprobarLuces(enum estado_luces condicion);

- NOTA: typedef no crea nuevos tipos, simplemente crea alias a tipos existentes.

- typedef enum {FALSE, TRUE} bool;

Estructuras

- Las estructuras pretenden aglutinar toda la información de una entidad - un ente.
  - Ejemplo. Un cheque bancario debe contener: nombre (char *), la cantidad (float) y el numero de cheque (int)

- struct identificador_estructura{
  tipo id1;
  tipo id2;
  …
  } variable_estructura;

Estructuras

```
struct ficha{
   char nombre[40];
   char direccion[40];
   long telefono;
};

Uso:
struct ficha mificha;
mificha.telefono=666335666;

Posibilidad:
typedef struct ficha TFICHA;
TFICHA mificha;
```

Notas sobre: estructuras anidadas / inicialización / asignación entre estructuras / Funciones devuelven estructuras

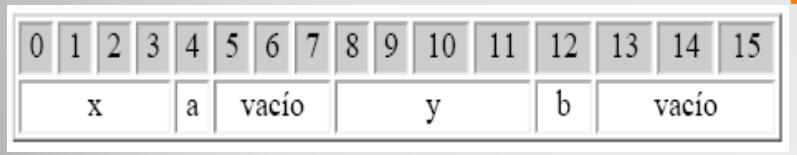
Inicialización de Estructuras

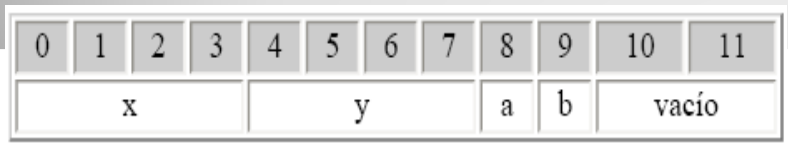
```
struct A {
   int i;
   int j;
   int k;
};

struct B {
   int x;
   struct C {
      char c;
      char d;
   } y;
   int z;
};

A ejemploA = {10, 20, 30};
B ejemploB = {10, {'a', 'b'}, 20};
```

Ojo al inicializar estructuras Anidadas (ejemploB)!!!!!!





Tamaño de Estructuras (sizeof)

```
struct A {
   int x;
   char a;
   int y;
   char b;
};
```

```
struct B {
   int x;
   int y;
   char a;
   char b;
};
```

Tamaño int: 4
Tamaño char: 1
Tamaño struct A: 16
Tamaño struct B: 12

No 10 ???

Debido al efecto byte-align o alineación de bytes.

Para mejorar rendimiento procesador, no se accede a todas las posiciones de memoria. Para microprocesadores de 32 bits (4 bytes), es mejor si se accede a múltiplos de 4. El compilador alinea las variables con esas posiciones

Uniones

- Una union es una región de almacenamiento compartida por dos o mas miembros, generalmente de tipos diferentes.
  - Permite manipular diferentes tipos de datos usando la misma zona de memoria.
  - Misma sintaxis que para las estructuras, cambiando la palabra struct por union.
  - Tamaño de memoria el del tipo de dato que más ocupe.

Uniones

```
typedef union{
   float a;
   int b;
} tunion;

main(){
   tunion var1;
   var1.a=10.5;
   printf("%f\n",var1.a);
   printf("%d\n",var1.b);
}
```

Salida programa:
10.5
1093140480

Ver representación coma flotante

```
union unEjemplo {
        int A;
        char B;
        double C;
} UnionEjemplo;

int main() {
  UnionEjemplo.A = 100;
  cout << UnionEjemplo.A << endl;
  UnionEjemplo.B = 'a';
  cout << UnionEjemplo.B << endl;
  UnionEjemplo.C = 10.32;
  cout << UnionEjemplo.C << endl;
  cout << &UnionEjemplo.A << endl;
  cout << (void*)&UnionEjemplo.B << endl;
  cout << &UnionEjemplo.C << endl;
  cout << sizeof(unEjemplo) << endl;
  cout << sizeof(unEjemplo::A) << endl;
  cout << sizeof(unEjemplo::B) << endl;
  cout << sizeof(unEjemplo::C) << endl;
}
```

- Suponiendo que int ocupa dos bytes, char un byte y double 4 bytes, la forma en que se almacena la información en la unión del ejemplo es la siguiente:

```
[BYTE1] [BYTE2] [BYTE3] [BYTE4]
[<----A----->]
[<-B->]
[<-----------C------------>]
```

Uniones

Tipo especial de estructuras que permiten almacenar elementos de diferentes tipos en las mismas posiciones de memoria, aunque no simultáneamente.

union [<tipo unión>] {
[<tipo> <nombre variable>
[, <nombre variable>, ...]] ;
} [<variable_unión>
[,<variable union>...]] ;

Otro ejemplo: ¿Cómo puedo acceder a x,y,z?

```
struct stCoor3D {
    int X, Y, Z;
};

union unCoor3D {
    struct stCoor3D N;
    int Coor[3];
} Punto;
```

Campos de bits

consiste en empaquetar los campos de la estructura en el interior de enteros, usando bloques o conjuntos de bits para cada campo.

```
struct
[<nombre estructura>] {
unsigned <tipo_entero>
<identificador>:<núm_de_
bits>;
.
} [<lista_variables>];
```

```
struct mapaBits {
        unsigned char bit0:1;
        unsigned char bit1:1;
        unsigned char bit2:1;
        unsigned char bit3:1;
        unsigned char bit4:1;
        unsigned char bit5:1;
        unsigned char bit6:1;
        unsigned char bit7:1;
};
```

```
struct mapaBits2 {
        unsigned short int campo1:3;
        unsigned short int campo2:4;
        unsigned short int campo3:2;
        unsigned short int campo4:1;
        unsigned short int campo5:6;
};
```

```
struct mapaBits3 {
        unsigned char campo1:5;
        unsigned char campo2:5;
};
```

- En el primer caso se divide un valor **char** sin signo en ocho campos de un bit cada uno:

7	6	5	4	3	2	1	0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

- Segundo caso, entero 16 bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
campo5						campo4	campo3	campo2					campo1		

- Tercer caso: No es posible empaquetar campo2 dentro del mismo char de campo1

unsigned char								unsigned char							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
			campo2								campo1				

Bits sobrantes no se usan

Estas estructuras se usan en programas, cuando se relacionan con ciertos dispositivos físicos. Por ejemplo: configurar puerto serie en MS-DOS se usa una estructura empaquetada en un unsigned char, que indica los bits de datos, de parada, la paridad, etc