Tema 1. Introducción a C

Algoritmia

Profesor: Andrés Muñoz

Escuela Politécnica

Andrés Muñoz - Tlf: (+34) 968278821
Universidad Católica San Antonio de Murcia - Tlf: (+34) 968 27 88 00 info@ucam.edu - www.ucam.edu

Índice

Tipos, operadores y expresiones

- Tipos básicos

char	Tipo carácter
short	Entero corto
int	Entero
long	Entero largo
float	Real de punto flotante con precisión normal
double	Real de punto flotante de doble precisión

Se puede añadir "unsigned" a los tipos carácter y enteros para indicar la ausencia de signo (únicamente positivos)

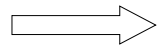
3

Tipos, operadores y expresiones

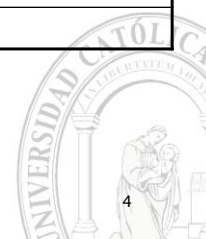
- Operadores aritméticos, relacionales y lógicos

%	Módulo aritmético (sólo para enteros)
+, -, *, /	Operaciones básicas
++, --	Operadores unarios de incremento y decremento
==, !=	Igualdad y desigualdad
>=, <=, <, >	Relaciones de orden

```
int a = 5 , b  = 6, c = 0;
c = ++a + b++;
printf("Salida: %d",c);
```



```
Salida: 12
```



4

Tipos, operadores y expresiones

- Operadores y expresión de asignación

a += b	a = a + b
a -= b	a = a – b
a /= b	a = a / b
a %= b	a = a % b
a *= b	a = a * b

5

Tipos, operadores y expresiones

- Arrays

```
int nombreVariable[10];
```

  - Las cadenas en C son arrays de caracteres terminados con un carácter nulo '\0'

H	O	L	A	\0

  - Las posiciones se enumeran a partir de cero.

H	O	L	A	\0
0	1	2	3	4

```
nombreVariable[0] = 'h';
```

6

Tipos, operadores y expresiones

- Constantes simbólicas

  #define LOWER 0

- Macros
  - Lista de parámetros que son sustituidos por el texto correspondiente en el lugar donde se usan las macros

```
#define max(A, B) ((A) > (B) ? (A) : (B))
```

la línea

```
x= max(p+q, r+s);
```

será sustituida por

```
x = ((p+q) > (r+s) ? (p+q) : (r+s));
```

7

Índice

- Tipos, operadores y expresiones
- Control de flujo
- Funciones y estructura de programas
- Punteros y tablas
- Estructuras y definición de tipos
- Ejemplos

8

Control de flujo

- Proposición *if-else*

```
if (expresión)
        proposición;
else
        proposición;
```

```
if (expresión)
        proposición;
elsif (expresión)
        proposición;
```

```
if (expresión){
        proposición-1;
        proposición-n;
}else{
        proposición-1;
        proposición-n;
}
```

- Proposición *switch*

```
switch (c){
        case '0':
        case '1':
                proposición-1;
                proposición-n
                break;
        default:
                proposición-1;
                break;
}
```

Control de flujo

- Proposición *do-while*

```
do{
        proposición-1;
        (…)
        proposición-n
}while (expresión);
```

- Proposición *while*

```
while (expresión){
        proposición-1;
        (…)
        proposición-n
}
```

Control de flujo

- Proposición *for*

```
for (proposición; condición; proposición){
        proposición-1;
        proposición-2;
        (…)
        proposición-n;
}
```

Sólo se ejecuta al inicio

Condición de salida

Se ejecuta cada iteración y antes de evaluar la condición

Control de flujo

- Proposición *for*

```
for (proposición; condición; proposición){
        proposición-1;
        proposición-2;
        (…)
        proposición-n;
}
```

```
int matriz[10];
for (int i=0,j=10 ; i < 10; i++, j--){
        int t = matriz[i];
        matriz[i] = matriz[j]
        matriz[j] = t;
}
```

Se puede separar con "," la inicialización de variables

Control de flujo

- Proposición *break*
  - Permite que la condición de evaluación no se realice ni al principio (while) ni al final (do-while), sino en algún paso de la estructura iterativa.

```
#define loop_forever for(;;)

loop_forever{
        proposición_1;
        proposición_2;
        (…)
        if (condición) break;
        (…)
        proposición_n-1;
        proposición_n;
}
```

Índice

- Tipos, operadores y expresiones
- Control de flujo
- Funciones y estructura de programas
- Punteros y tablas
- Estructuras y definición de tipos
- Ejemplos

Funciones y estructura de programa

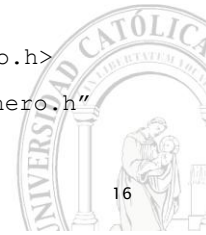
- Las funciones en C desempeñan el papel de las subrutinas o procedimientos en otros lenguajes
  - Agrupan una serie de operaciones para su posterior invocación.
  - Permiten modularizar el código.
  - Simplifican el código, su desarrollo y depuración.
- Al igual que las variables, es necesario declararlas previamente y posteriormente implementarlas.

```
Declaración:
        tipo_devuelto nombre_funcion (argumentos);
Definición:
        tipo_devuelto nombre_funcion (argumentos) {
                sentencias;
        }
```

15

Funciones y estructura de programa

- En un procedimiento el tipo devuelto es *void*.
- Los parámetros se separan mediante comas.
- Muchas veces suele ser interesante agrupar las funciones en una librería o en un fichero
  - Para incluir una librería: `#include <stdio.h>`
  - Para incluir un fichero: `#include "mifichero.h"`



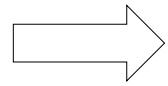
16

Funciones y estructura de programa

- Preprocesador de C
  - Prepara el código antes del compilador.
  - Sustituye el texto de las directivas #define.
  - Incluye los ficheros de las directivas #include.

Un ejercicio (I)

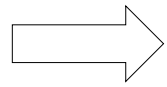
```
#include <stdio.h>
#include <stdlib.h>

#define al_cuadrado(x) (x * x)

int main()
{
 int a = 0;
    int p = 4;

    a = al_cuadrado(p);
    printf("%d \n",a);

    system("PAUSE");
    return 0;
}
```



¿?

Un ejercicio (I)

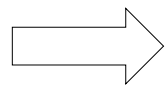
```
#include <stdio.h>
#include <stdlib.h>

#define al_cuadrado(x) (x * x)

int main()
{
    int a = 0;
    int p = 4;

    a = al_cuadrado(p);
    printf("%d \n",a);

    system("PAUSE");
    return 0;
}
```



- 16

Un ejercicio (II)

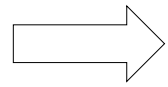
```
#include <stdio.h>
#include <stdlib.h>

#define al_cuadrado(x) (x * x)

int main()
{

    int a = 0;
    int p = 4;

    printf("%d \n",p);
    a = al_cuadrado(++p);
    printf("%d \n",a);

    system("PAUSE");
    return 0;
}
```



¿?

Un ejercicio (II)

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

#define al_cuadrado(x) (x * x)

int main()
{
    int a = 0;
    int p = 4;

    a = al_cuadrado(++p);
    printf("%d %d\n",a , p);

    system("PAUSE");
    return 0;
}
```



➢36 6

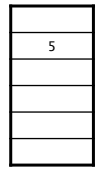
Índice

- Tipos, operadores y expresiones
- Control de flujo
- Funciones y estructura de programas
- Punteros y tablas
- Estructuras y definición de tipos
- Ejemplos

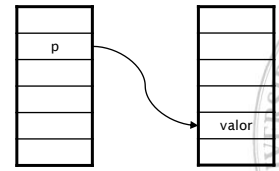
Punteros y tablas

- Una variable “normal” almacena el dato en una dirección de memoria.
- Un puntero almacena una dirección de memoria en otra.

```
int i = 5
```




```
int *p;
p = (int *)malloc(N*sizeof(int));
*p = 5;
```



Punteros y tablas

- Uso elemental de punteros

```
int i, *p;
```

p = &i	Hace que el apuntador ‘p’ apunte a la variable ‘i’
i = *p	Asigna a ‘i’ el valor al que apunta ‘p’
(*p)++	Incrementa en uno el valor entero al que apunta ‘p’
*p += 2	Incrementa en dos el valor entero al que apunta ‘p’

```
int a[10], *pa, *pb;
```

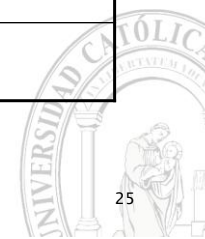
*pa	Accede al entero al que apunta ‘pa’
pa[0]	Es equivalente a la expresión anterior
*(pa+2)	Accede al tercer entero que se encuentra a partir de la dirección apuntada por ‘pa’
pa[2]	Es equivalente a la expresión anterior

Punteros y tablas

- Uso elemental de punteros

```
int a[10], *pa, *pb;
```

pa = &a[4]	¿?
pa++	
pa += 2	
pa –pb	



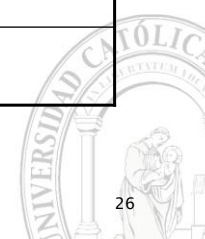
25

Punteros y tablas

- Uso elemental de punteros

```
int a[10], *pa, *pb;
```

pa = &a[4]	Hace que 'pa' apunte al quinto elemento del vector 'a'
pa++	
pa += 2	
pa –pb	



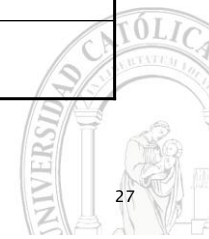
26

Punteros y tablas

- Uso elemental de punteros

```
int a[10], *pa, *pb;
```

pa = &a[4]	Hace que 'pa' apunte al quinto elemento del vector 'a'
pa++	**¿?**
pa += 2	
pa -pb	

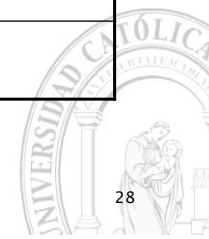


Punteros y tablas

- Uso elemental de punteros

```
int a[10], *pa, *pb;
```

pa = &a[4]	Hace que 'pa' apunte al quinto elemento del vector 'a'
pa++	Hace que 'pa' apunte al entero siguiente al que apuntaba
pa += 2	
pa -pb	

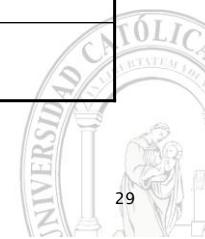


Punteros y tablas

- Uso elemental de punteros

```
int a[10], *pa, *pb;
```

pa = &a[4]	Hace que 'pa' apunte al quinto elemento del vector 'a'
pa++	Hace que 'pa' apunte al entero siguiente al que apuntaba
pa += 2	**¿?**
pa –pb	



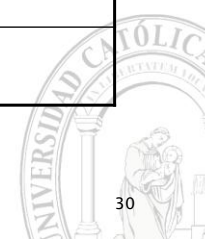
29

Punteros y tablas

- Uso elemental de punteros

```
int a[10], *pa, *pb;
```

pa = &a[4]	Hace que 'pa' apunte al quinto elemento del vector 'a'
pa++	Hace que 'pa' apunte al entero siguiente al que apuntaba
pa += 2	Hace que 'pa' apunte al entero que se encuentra dos posiciones más allá del que apuntaba
pa –pb	



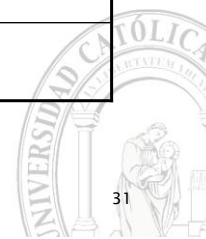
30

Punteros y tablas

- Uso elemental de punteros

```
int a[10], *pa, *pb;
```

pa = &a[4]	Hace que 'pa' apunte al quinto elemento del vector 'a'
pa++	Hace que 'pa' apunte al entero siguiente al que apuntaba
pa += 2	Hace que 'pa' apunte al entero que se encuentra dos posiciones más allá del que apuntaba
pa –pb	**¿?**



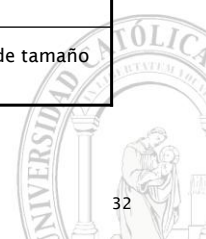
31

Punteros y tablas

- Uso elemental de punteros

```
int a[10], *pa, *pb;
```

pa = &a[4]	Hace que 'pa' apunte al quinto elemento del vector 'a'
pa++	Hace que 'pa' apunte al entero siguiente al que apuntaba
pa += 2	Hace que 'pa' apunte al entero que se encuentra dos posiciones más allá del que apuntaba
pa –pb	Calcula la diferencia (en número de posiciones de tamaño entero) entre los apuntadores 'pa' y 'pb'

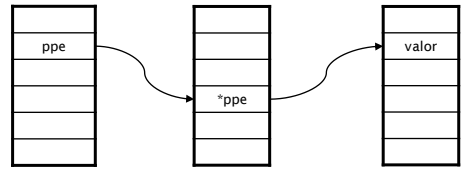


32

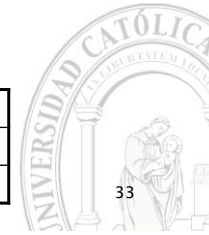
Punteros y tablas

- Apuntadores a apuntadores
  - Utilizando la notación (*) es posible construir apuntadores a apuntadores, apuntadores …

```
int **ppe
```



ppe	Es un puntero a un puntero a entero
*ppe	Es un puntero a entero
**ppe	Es un entero

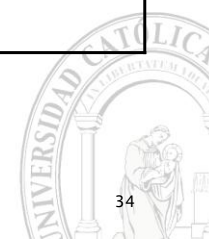


Punteros y tablas

- Uso elemental de punteros

```
int **ppe, *pe;
```

pe = *ppe	**¿?**
ppe = &pe	
ppe[3]	
ppe[2][3]	

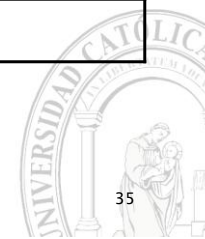


Punteros y tablas

- Uso elemental de punteros

```
int **ppe, *pe;
```

pe = *ppe	Asigna a 'pe' el valor del puntero a entero al que apunta 'ppe'
ppe = &pe	
ppe[3]	
ppe[2][3]	



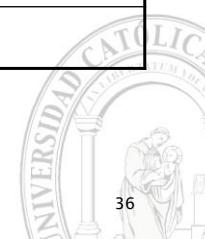
35

Punteros y tablas

- Uso elemental de punteros

```
int **ppe, *pe;
```

pe = *ppe	Asigna a 'pe' el valor del puntero a entero al que apunta 'ppe'
ppe = &pe	**¿?**
ppe[3]	
ppe[2][3]	



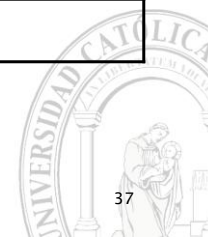
36

Punteros y tablas

- Uso elemental de punteros

```
int **ppe, *pe;
```

pe = *ppe	Asigna a 'pe' el valor del puntero a entero al que apunta 'ppe'
ppe = &pe	Asigna a 'ppe' la dirección de 'pe'
ppe[3]	
ppe[2][3]	



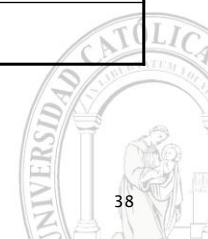
37

Punteros y tablas

- Uso elemental de punteros

```
int **ppe, *pe;
```

pe = *ppe	Asigna a 'pe' el valor del puntero a entero al que apunta 'ppe'
ppe = &pe	Asigna a 'ppe' la dirección de 'pe'
ppe[3]	**¿?**
ppe[2][3]	



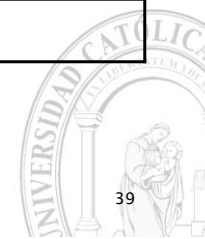
38

Punteros y tablas

- Uso elemental de punteros

```
int **ppe, *pe;
```

pe = *ppe	Asigna a 'pe' el valor del puntero a entero al que apunta 'ppe'
ppe = &pe	Asigna a 'ppe' la dirección de 'pe'
ppe[3]	Es un puntero a entero
ppe[2][3]	



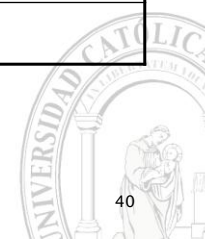
39

Punteros y tablas

- Uso elemental de punteros

```
int **ppe, *pe;
```

pe = *ppe	Asigna a 'pe' el valor del puntero a entero al que apunta 'ppe'
ppe = &pe	Asigna a 'ppe' la dirección de 'pe'
ppe[3]	Es un puntero a entero
ppe[2][3]	***¿?***



40

Punteros y tablas

- Uso elemental de punteros

```
int **ppe, *pe;
```

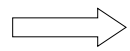
pe = *ppe	Asigna a 'pe' el valor del puntero a entero al que apunta 'ppe'
ppe = &pe	Asigna a 'ppe' la dirección de 'pe'
ppe[3]	Es un puntero a entero
ppe[2][3]	Es un entero

41

Punteros y tablas

- Argumentos en funciones
  - En C todos los parámetros se pasan por valor (excepto los arrays)

```
void swap (int x, int y){
        int temp;

        temp = x;
        x = y;
        y = temp;
}
(…)
main(){
        int a = 5, b= 3;
        swap (a ,b);
        printf("%d %d",a,b);
}
```



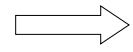
- ¿?

42

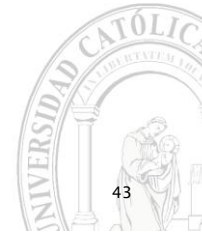
Punteros y tablas

- Argumentos en funciones
  - En C todos los parámetros se pasan por valor (excepto los arrays)

```
void swap (int x, int y){
        int temp;

        temp = x;
        x = y;
        y = temp;
}
(…)
main(){
        int a = 5, b= 3;
        swap (a ,b);
        printf("%d %d",a,b);
}
```



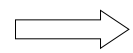
➢ 5 3



Punteros y tablas

- Argumentos en funciones
  - Se puede obviar esta restricción utilizando apuntadores

```
void swap (int *px, int *py){
        int temp;

        temp = *px;
        *px = *py;
        *py = temp;
}
(…)
main(){
        int a = 5, b= 3;
        swap (&a ,&b);
        printf("%d %d",a,b);
}
```

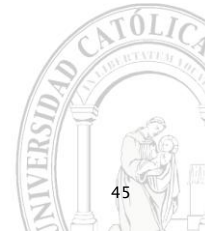


➢ 3 5

Punteros y tablas

- Punteros a funciones

```
float radianes_a_gr (float radianes){
        return (radianes * 180.0 / 3.1415);
}

void convierte (float v[], int n, float(*fun)(float)){
        int i;
        for (i = 0; i < n; i++)
                v[i] = (*fun)(v[i]);
}

main(){
        float (*fun1)(float);
        float vector[50];

        fun1 = radianes_a_gr; //No hay paréntesis

        convierte(vector, 50, fun1)
        printf("%d %d",a,b);
}
```

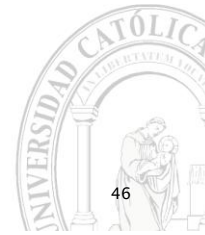


45

Punteros y tablas

- Matrices dinámicas
  - Es posible crear matrices de 'n' dimensiones mediante el uso de punteros

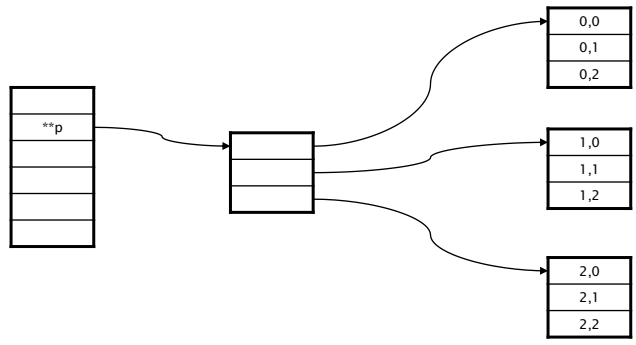
```
void matrizDinamica(int dimension){
        int i, **p;
    p = (int **)malloc(sizeof(int*)*dimension);

    for (i = 0; i < dimension ; i++){
        p[i] = (int *)malloc(sizeof(int)*dimension);
    }

    p[0][0] = 20;
    printf("%d\n",p[0][0]);
}
```

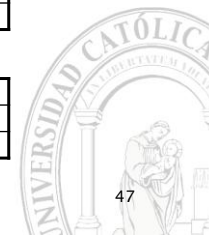


46

Punteros y tablas

- Matrices dinámicas
  - Es posible crear matrices de 'n' dimensiones mediante el uso de punteros





47

Índice

- Tipos, operadores y expresiones
- Control de flujo
- Funciones y estructura de programas
- Punteros y tablas
- Estructuras y definición de tipos
- Ejemplos

48

Estructuras

- Semejantes a los registros de Pascal
- Se declaran con la palabra clave *struct*.

```
struct empleado{
        char nombre[40];
        char direccion[50];
        unsigned long dni;
        double salario;
} persona;
```

- Las estructuras se pueden inicializar en su declaración

```
struct empleado gerente = {"Juan","Calle",11,20};
```

49

Estructuras

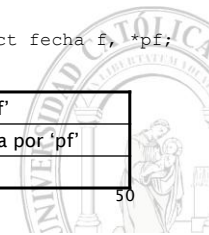
- Las estructuras se pueden anidar

```
struct fecha{
        int dia;
        int mes;
        int anyo;
};
```

```
struct socio_club{
        char nombre[40];
        char direccion[50];
        struct fecha nacimiento;
} t;
```

- Acceso a los datos `t.nacimiento.dia`
- Apuntadores a estructuras `struct fecha f, *pf;`

pf = &f	Hace que el pauntado 'pf' apunte a la estructura 'f'
(*pf).dia	Accede al elemento 'dia' de la estructura apuntada por 'pf'
pf->	Equivalente a la expresión anterior



50

Definición de tipos

- C permite la creación de sinónimos de tipos.
- Palabra reserva *typedef*

```
typedef struct persona{
        char nombre[40];
        char direccion[50];
} PERSONA, *P_PERSONA;

(…)

PERSONA gerente;
P_PERSONA cliente;

(…)
gerente.direccion = cliente->direccion;
```

Índice

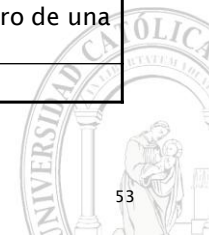
- Tipos, operadores y expresiones
- Control de flujo
- Funciones y estructura de programas
- Punteros y tablas
- Estructuras y definición de tipos
- Ejemplos

Ejemplos

- El programa más simple:

```
main(){
        printf("hello, world\n");
}
```

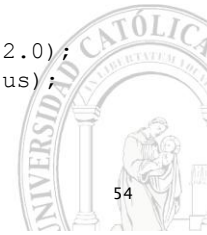
main	Función principal del programa
printf	Función para imprimir por pantalla
\n	Secuencia que indica un salto de línea dentro de una cadena de caracteres entrecomillada
{ … }	Apertura y cierre de bloque



Ejemplos

- Un programa más "complejo"

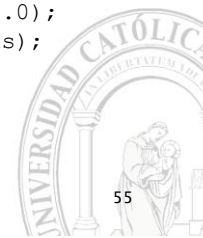
```
main(int argc, char *argv[]){
        int lower, upper, step;
        float fahr, celsius;

        lower = 0;
        upper = 300;
        step = 20;

        while (fahr <= upper){
                celsius = (5.0 /9.0) * (fahr - 32.0);
                printf("%f \t %f \n", fahr,celsius);
                fahr = fahr + step;
        }
}
```



Ejemplos

```
main(){
        int lower, upper, step;
        float fahr, celsius;

        lower = 0;
        upper = 300;
        step = 20;

        while (fahr <= upper){
                celsius = (5.0 /9.0) * (fahr - 32.0);
                printf("%f \t %f \n", fahr,celsius);
                fahr = fahr + step;
        }
}
```

Podemos declarar variables de un mismo tipo separándolas por comas



Ejemplos

```
main(){
        int lower, upper, step;
        float fahr, celsius;

        lower = 0;
        upper = 300;
        step = 20;

        while (fahr <= upper){
                celsius = (5.0 /9.0) * (fahr - 32.0);
                printf("%f \t %f \n", fahr,celsius);
                fahr = fahr + step;
        }
}
```

Toda sentencia termina en ";"