eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

Actividad 1
C para SO
Introducción a los Sistemas Operativos, 2025-2026

Pablo González Nalda

Depto. de Lenguajes y Sistemas Informáticos
EU de Ingeniería de Vitoria-Gasteiz,
UPV/EHU

26 de enero de 2026

Contenidos de la presentación

Contenidos

¿Por qué C?

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72

¿Por qué C?

Y porque C se diseñó para programar Unix, permite un control total del hardware y es un estándar.

Contenidos

Programa “Hola, muy buenas...”

Programa simple:

```
1  /* Programa simple */
   /* Comentario. No pueden anidarse */
   #include <stdio.h>
4  main( ) {
       printf("\nHola, muy buenas...\n");
   }
```

```
gcc -o hola hola.c # compilar en el fichero de salida hola
./hola # ejecutar el fichero hola del directorio actual
```

Variables

Lenguaje fuertemente tipado, **sin** orientación a objetos.

```
1  #include <stdio.h>
   main( ) {
       int entero;     /* entero con signo */
4      char caracter;      /* carácter ASCII */
       float real;     /* real simple precisión */
       entero = 2+2;
7      caracter = 'a';
       real=6.023E23;
       printf("\nResultado: %d\t'%c' ",entero, caracter);
10     printf("\treal %f", real);
   }
```

Constantes

Constantes en el código: `66`

Constantes de preprocesador (sustituir antes de compilar) con `#define`.

Constantes con `const`

```
1  #include <stdio.h>
   #define MAX 50
   main( ) {
4      const int entero=3;
       const float PI=3.1415926;
       printf("\nResultado: %d, otros %d %d",entero, 66, MAX);
7      printf("\treal %f", real);
   }
```

Control de compilación

En el código compilado sólo habrá una de los dos `printf` según esté o no la línea 2.

```
1 #include <stdio.h>
  #define PRUEBA
  main( ) {
4 #ifdef PRUEBA
      printf("Prueba");
  #else
7     printf("No hay prueba");
  #endif
  }
```

Control de compilación

La inclusión o no de la definición PRUEBA se proporciona en el momento de compilación.

```
#include <stdio.h>
main( ) {
#ifdef PRUEBA
    printf("Prueba");
#else
    printf("No hay prueba");
#endif
}
```

```
gcc -o prueba prueba.c -DPRUEBA
```

Dualidad carácter/valor ASCII

Por ejemplo, una variable de un byte (`char`) puede manipularse como caracteres o enteros, ya que los caracteres son valores ASCII.

Una resta entre caracteres es la resta de sus valores ASCII.

Un `char` se puede imprimir como carácter o como entero.

```
#include <stdio.h>
main( ) {
    char a='C',b='f', c='3';
    int x;
    x=a-'A';
    printf("\nDistancia: %d",x);
    printf("\nValor numérico: %d", c-'0'); /* 51-48 */
    a=a+('a'-'A');      /* +32 pasa a minúsculas*/
    b=b-32;          /* Pasa a mayúsculas */
    printf("\n%c \t%c", a,b); /* resultado */
}
```

Operadores matemáticos

Java heredó las mismas operaciones.

```
a=-b;
a=a+b;
a=c-b;
a=c*b;
a=c/b;    Si son enteros, sólo da el cociente de la
    división. Si uno de ellos (por lo menos) es real, da la
     división con todos los decimales posibles
a=c%b;    Sólo se puede usar con enteros, y da el resto de
    la división entera
```

Abreviaturas


```
a=a+1; -> a++; o también ++a; (Hay diferencia)
b=b-1; -> b--; o también --b; (Hay diferencia)
b = b + c; -> b += c;
b = b - c; -> b -= c;
b = b * c; -> b *= c;
b = b / c; -> b /= c;
 /*********** ¡¡Cuidado!! ************/
c=3;
b=c+1; -> b tiene 4 y c tiene 3 (d=c+1;          b=d;)
b=c++; -> b tiene 3 y c tiene 4 (d=c; c=c+1;     b=d;)
b=++c; -> b tiene 4 y c tiene 4 (c=c+1; d=c;     b=d;)
```

Entrada/Salida


```
1  #include <stdio.h>
   void main(){
        int num;
4       char car, nombre[10]; /* Cadena de caracteres */
        printf("Introduce un numero entero");
        scanf("%d", &num); /* enteros */
7       printf(" la variable \"car\": ");
        fflush(stdin); /* Vacía el búfer del teclado */
        scanf("%c", &car); /* caracteres */
10      fflush(stdin);
        printf("\nIntroduce un nombre");
        scanf("%s", nombre); /* cadena */
13      printf("\n\nEl número es %d, \t y el ", num);
        printf("carácter %c.\n", car);
        printf("La cadena es %s", nombre);
16 }
```

Operaciones lógicas con bits


```
#include <stdio.h>
main() {
    char dato, mascara=0x01; /* mascara=1 */
    int i, cont=0;
    scanf("%c", &dato);
    for (i=0; i<8; i++) {
        printf("%d",((dato&mascara)!=0));
        if((dato & mascara)!=0) cont++; /* and de bits */
        mascara = mascara << 1; /* Desplazamiento de */
    }                                   /* bits a la izq */
    printf("\nEn '%c' hay %d unos\n", dato,cont);
}

& and          >> Desplazamiento a la derecha
| or          << Desplazamiento a la izquierda
~ not          Lo mismo con iguales     &=     |=     ^=
^ xor                        <<=    >>=
```

Contenidos

Condicionales


```
1 if (b == 0) {
      a=1;
  }
4 else {
      a=2;
  }
7 /* Equivalente pero cuesta leerlo */
  a = (b==0 ? 1 : 2 );
```

Operaciones condicionales


```
1  <          menor que
   <=      menor o igual que
   ==     igual (dos iguales)
4  !=      distinto de
   >         mayor que
   >=      mayor o igual que
7  ||         uno u otro, o los dos    (or lógico)
   &&     uno y otro     (and lógico)
   !( expresión)  no es cierto, no ocurre ese algo (not lógico
       )
```

Ejemplo de condicionales: `if`


```
/*    if anidados adecuados para switch.C */
#include <stdio.h>
int nota;
void main() {
    printf("Dame tu nota ");
    scanf("%d", &nota);
    if(nota==0||nota==1||nota==2|| nota==3 ||nota==4) {
        printf("\nLo siento, has suspendido \n");
        printf("Si intentas otra, apruebas\n\n");
    }
    else if (nota==5 || nota==6)
        printf("\nUn aprobado \n");
    else if (nota==7 || nota==8)
        printf("\nUn notable, muy bien \n");
    else if (nota==9)
        printf("\nSobresaliente \n");
    else if (nota==10) printf("\nFelicidades, un 10 \n");
    else if (nota==11) {
        printf("\n Menos lobos... \n");
        printf("\n¿Qué nota es ésa? \n");
    } else printf("\n¿Qué nota es ésa? \n");
    getch(); /* para el programa hasta pulsar una tecla*/
}
```

Ejemplo de condicionales: `switch`


```
1  #include <stdio.h>
   int nota;
   void main(){
4          printf("Dame tu nota "); scanf("%d", &nota);
       switch(nota){
       case 0: case 1: case 2: case 3:case 4:
7          printf("\nLo siento, has suspendido \n");
           printf("Si intentas otra vez, apruebas\n\n");
       break;
10     case 5:case 6: printf("\nUn aprobado \n"); break;
       case 7:
       case 8:
13         printf("\nUn notable, muy bien \n");
       break;
       case 10:
16         printf("\nFelicidades, un 10 \n");
       case 9: printf("\nSobresaliente \n"); break;
       case 11:
19         printf("\n Menos lobos... \n");
       default:
           printf("\n¿Qué nota es ésa? \n");
22     } /* fin switch */
       getch(); /* para el programa hasta pulsar tecla*/
   }
```

Ejemplo de iterativas


```
#include <stdio.h>
void main() {
    char sn;
    int n=10;
    do {
        printf("\n¿seguimos?(S/N)");
        fflush(stdin);
        scanf("%c", &sn);
    } while (sn=='s' || sn=='S');

    while (n>0) {
        printf("\t%d,",n);
        n--;
    }
    printf("\t%d.",n);
}
```

Ejemplo de iterativas: `for`


```
for (cont=1; cont<=10; cont=cont+1)
    printf("\n¡Hola!");
// Para imprimir los múltiplos de 7 menores de 500:
for (cont=7; cont<500; cont=cont+7)
    printf("\n%d",cont);
// Y si queremos una cuenta atrás:
for (cont=10; cont>0; cont=cont-1)
    printf("\n%d",cont);
// O también:
for (cont=10; cont>0; cont--)
    printf("\n%d",cont);
// O también:
cont=10;
while (cont>0)
    printf("\n%d",cont--);
```

El factorial en una línea (digamos que es ilegible):

```
for(i=1,f=1; i<=x; f*=i, i++);
for(i=f=1; i<=x; f*=i++);
```

Contenidos

Tablas, arrays, vectores, matrices, arreglos


```
1  #include <stdio.h>
   #define N 10
   main( ) {
4      int i, v[N], aux;
       for (i=0; i<N; i++) {
           printf("\nDame el %dº valor: ",i+1);
7          scanf("%d", &v[i]);
       }
       aux=v[0];
10     for (i=0; i<N-1; i++)
           v[i]=v[i+1];
       v[N-1]=aux;
13     printf("\nDesplazada a la izquierda: \n");
       for (i=0; i<N; i++) printf("\t%d ", v[i]);
   }
```

Contenidos

Cadenas o *strings*

En C, una cadena de caracteres es una tabla de caracteres ASCII (un byte) terminada en un byte con cero binario (`ASCII 0 o '0\0'`)

```
char cad[40];
cad[0] = 'h';
cad[1] = 'o';
cad[2] = 'l';
cad[3] = 'a';
cad[4] = '\0'; /* cad[4] = 0; */

char cad[40]= {'H','o','l','a','\0'};
char cad[40]= "Hola";
```

En el resto de posiciones hay valores indeterminados, lo que contenía la memoria antes de reservar ese trozo (probablemente ceros, ya que hay memoria de sobra).

Contenidos

Subprogramas


```
#include <stdio.h>
long fact(int n); /* declaración */
main( ) {
    printf("\nEl factorial de 14 es %ld", fact(14));
}
long fact(int n) { /* definición */
  long r=1l; /* constante 1 de tipo long */
  int i;
  for (i=1; i<=n; i++)
    r*= (long) i; /* conversión o cast opcional de int a
         long */
  return r;
}
```

La función devuelve un long a partir del dato de entrada int

Funciones y Booleanos

Los booleanos son enteros. Cualquier valor distinto de cero es *verdadero*.

```
#include <stdio.h>
#define VERD 1
#define FALSO 0

int es_negativo(int x) {
  if (x<0)
    return VERD;
  else
    return FALSO;
}

main( ) {
    if (es_negativo(5))
        printf("5 es negativo");
}
```

Procedimientos


```
#include <stdio.h>
#define N 10
void pedir_tabla(int t[]) {
    int i;
    for (i=0; i<N; i++)
        scanf("%d", &t[i]);
}

void mostrar_tabla(int t[]) {
    int i;
    for (i=0; i<N; i++) printf("\t%d ", t[i]);
}

main( ) {
    int v[N];
    pedir_tabla(v);
    mostrar_tabla(v);
}
```

Manejo de ficheros

E/S estándar

```
printf("formato", lista de expresiones);
scanf ("formato", lista de expresiones);
```

E/S no estándar

```
1 fprintf(fich, "formato", lista de expresiones);
  fscanf (fich, "formato", lista de expresiones);
```

Conversión en memoria

```
1 sprintf(cadena, "formato", lista de expresiones);
```

Manejo de ficheros

Cifrado César (siguiente carácter)

```
#include <stdio.h>
#define EOF (-1)
void main() {
    FILE *entrada, *salida;
    int i;
    char ce, cs;
    entrada= fopen("entrada.txt","r");
    salida = fopen("salida.txt","w");
    while((ce=getc(entrada))!=EOF) {
        cs = ce+1;
        putc(cs,salida);
    }
}
```

Argumentos del programa


```
#include <stdio.h>
void main(int argc, char *argv[]) {
    int i;
    printf("\nPrograma: %s\n",argv[0]);
    for(i=1;i<argc;i++)
        printf("argumento %d: %s\n",i,argv[i]);
}
```

Contenidos

Basado en...

Basado en: A TUTORIAL ON POINTERS AND ARRAYS IN C

by Ted Jensen

`http://pweb.netcom.com/~tjensen/ptr/cpoint.htm`

Punteros

Los punteros o apuntadores (*pointers*) son variables que contienen una posición de memoria, normalmente de otra variable de un tipo de datos determinado.

```
int k,j;
k=2; j=7;

int *ptr;          /* p es un puntero a enteros */
// Es decir, contiene una dirección de memoria, y el
    lenguaje sólo permite que apunte a enteros
ptr=NULL;
ptr=&k; /* ptr contiene la dirección de memoria de k */
j=*ptr; /* j recibe lo apuntado por ptr, un entero con el
    valor 2 */
*ptr=7; /* en ese entero se introduce el 7 */

printf("j %d @ %p\n", j, (void *)&j);
printf("k %d @ %p\n", k, (void *)&k);
printf("ptr %p @ %p\n", ptr, (void*)&ptr);
printf(" Apuntado por ptr es %d\n", *ptr);
```

Bytes de un entero con punteros

Orden de los bytes de un entero:

```c
#include <stdio.h>
int main() {
    int n=0x05060708; // en hexadecimal, 4 bytes
    char *p;
    p = (char *) &n; /* &n apunta a un entero y se
        convierte a ptr a carácter*/
    for (int b=0;b<4;b++) {
        printf("\nbyte %d \t%p\n", *p, p);
        p++; // se suma 1 porque es un puntero a char
    }
    /* lel mismo for en dos líneas */
    for (int b=0;b<4;b++)
        printf("\nbyte %d \t%p\n", *p, p++);
    return 0;
}
```

```
1942  vi p.c
1943  gcc -o p p.c
1944  ./p
```

Variables por valor y por referencia

Las variables simples, si no se usan punteros, se pasan por valor:

```
int suma (int a, int b) return a+b;
s= suma(c,d);
```

Las variables compuestas siempre se pasan por referencia por ser su nombre un puntero al primer elemento:

```
1 void desplaza (int a[]);
  int t1[10];
  desplaza(t1);
```

Variables por valor y por referencia (2)

Para pasar por referencia las variables simples hay que usar punteros:

```
void  intercambia (int * a, int * b) {
    int x;
    x=*a;
    *a=*b;
    *b=x;
}

// Y en el programa principal:

int c,d;
intercambia(&c, &d);
```

Variables por valor y por referencia (3)

Una función puede devolver un puntero, que es un dato simple. Hay que tener en cuenta que si se crea una cadena dentro de la función se reserva en la pila, y al salir de la función se puede perder.

```
1  #include <stdio.h>
   char * pidecadena (char * cad) {
       scanf("%s",cad);
4      return cad;
   }
   void main(){
7      char c[10];
       printf("\nDime una cadena: ");
       printf("Me has dado: \"%s\"\n", pidecadena(c));
10 }
```

Core

Core es un volcado de memoria (*core dump*) cuando un programa por ejemplo accede a memoria que no es suya.

En Unix se escribe un fichero `core` con el contenido de la memoria del programa para depurar. En Windows se escribe un fichero de extensión `.dmp`

Contenidos

Punteros y tablas (arrays)

Puntero para recorrer una tabla

```
int tabla[] = {1,23,17,4,-5,100};
int *ptr;
int i;
ptr = &tabla[0];  /*1*/
ptr = tabla;      /*2*/
printf("\n\n");
for (i = 0; i < 6; i++) {
    /*A*/    printf("tabla[%d] = %d",i,tabla[i]);
    /*B*/    printf("ptr + %d = %d\n",i, *(ptr + i)); //
        tabla+i
}
```

Dos formas de llenar un *array*

Llenamos un *array* reservado en el montículo en el puntero `p` accediendo por el nombre del array `p` dándole el valor de la posición más 10, y con un puntero `q` que al incrementarse con `q++` se le añade 4 por ser puntero a entero. Con `q` se le da `x+20`.

```
#include <stdio.h>
#include <stdlib.h> // man malloc nos dice su librería
int main() {
    int *p, *q;
    p=malloc(10*sizeof(int));
    q=p;
    for (int x=0;x<10;x++) {
        p[x]= (x+10);
        printf("#%d: %d  \t%d  \n", x, p[x], *q);
        *q  = (x+20);
        printf("#%d: %d  \t%d  \n", x, p[x], *q);
        printf("%p %p\n", &p[x], q++);
    }
    return 0;
}
```

Contenidos

Estructuras


```
struct ficha {
    char ap[20];
    char nom[20];
    int edad;
    float altura;
};
struct ficha yo; // reserva los 48 bytes
// es una variable global

int main(void) {
    strcpy(yo.ap,"G.N.");
    strcpy(yo.nom,"P.");
    printf("\n%s, ",yo.ap);
    printf("%s\n",yo.nom);
    return 0;
}
```

Estructuras y punteros


```
struct ficha *p;
p = &yo; // no reserva, ya tenemos la del código anterior
(*p).altura = 1.85;
p->altura = 1.85; // sintaxis más clara

void imprime(struct ficha *p)
{
    printf("\n%s, ", p->ap);
    printf("%s ", p->nom);
    printf("%d\n", p->altura);
}
```

Estructuras y `malloc`

En este caso sí que reservamos memoria para una ficha en el montículo *(heap)*, averiguando su tamaño con `sizeof` y convirtiendo el tipo del puntero devuelto por `malloc` para poder hacer la asignación.

```
1 struct ficha *p;
  p = (struct ficha *) malloc(sizeof(struct ficha));
```

Usando `typedef` con estructuras y `malloc`

Simplificamos si definimos un tipo usando `typedef`:

```
1  typedef struct {
       char ap[20];
       char nom[20];
4      int edad;
       float altura;
   } Ficha;
7  Ficha *p;
   p = (Ficha *) malloc(sizeof(Ficha));
```

Uso alternativo de `typedef` con estructuras y `malloc`

Creamos un tipo de datos que es puntero a ficha, y reservamos o ubicamos memoria con `malloc`

```
1  struct {
       char ap[20];
       char nom[20];
4      int edad;
       float altura;
   } ficha;
7
   // FICHA es un tipo de datos "puntero a la estructura ficha
       "
   typedef struct ficha *FICHA;
10
   FICHA p; // p es un puntero a estructura

13 p = malloc(sizeof(*FICHA));
```

Tablas de estructuras

Con lo anterior podemos crear un vector de **10 punteros** a ficha y crear para cada uno una ficha.

```
#define TAMTABLA 10
FICHA *t; // struct ficha **t;
t = malloc(sizeof(FICHA)*TAMTABLA);
for (i=0; i<TAMTABLA; i++) {
    t[i]=malloc(sizeof(*FICHA));
    llenaficha(t[i]);
}
```

Punteros a funciones

Punteros a funciones

Creamos cuatro funciones:

```
#include <stdio.h>
2
void f1() {
    printf("a\n");
5 }
void f2() {
    printf("b\n");
8 }
void f3() {
    printf("c\n");
11 }
void f4() {
    printf("d\n");
14 }
```

Podríamos ejecutar una escribiendo `(*f1) ()` que es lo mismo que `f1()`

Punteros a funciones

Y definimos un vector de punteros a código. El nombre de la función es un puntero a la posición de memoria donde empieza el código (binario, lenguaje máquina) de la función.

```
1  int main() {
       // tabla de punteros a 4 funciones
       void ( *t[4] ) (void);
4      int i;
       t[0]=f1;
       t[1]=f2;
7      t[2]=f3;
       t[3]=f4;

10     // def alternativa de la tabla
       // void (*t[])(void) = {f1, f2, f3, f4};

13     for (i=0; i<=3; i++)
           ( *t[i] ) (); // ejecutamos las funciones
   }
```

Ciclo infinito:

```
#define TRUE 1
while(1)
while(TRUE)
for(;;)
```

De un programa que ejecuta un bucle infinito (un *demonio* por ejemplo) se puede salir cuando el programa ejecute la *llamada al sistema exit(n)* que devuelve el valor `n` al programa que lo arrancó, normalmente el SO.

También puede terminar si recibe una *señal* que le *mate*.

Contenidos

¿Más preguntas?

¿Más preguntas?

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

Actividad 1
C para SO
Introducción a los Sistemas Operativos, 2025-2026

Pablo González Nalda

Depto. de Lenguajes y Sistemas Informáticos
EU de Ingeniería de Vitoria-Gasteiz,
UPV/EHU

26 de enero de 2026