Escuela Politécnica Superior

SEMINARIO-TALLER DE SOFTWARE (STI-S)

UNIDAD 4. Programación en C

Sesión 1

Índice

Índice

Introducción a la programación

- El Software esta presente en la mayoría de las tareas de ingeniería. Por ejemplo, diseño, control, simulación y optimización…
- Muchos programas no son visibles (no son ejecutados en el ordenador con pantalla, teclado y ratón) se ejecutan en microprocesadores.
- Estos microprocesadores están integrados en muchos dispositivos y se llaman **sistemas empotrados**.
- La programación es necesaria ya que:
  - La mayoría de los sistemas digitales están programados.
  - El software aporta inteligencia a los dispositivos.
  - Estructura la mente y potencia el pensamiento lógico.
  - …. y además es divertido.

Introducción a la programación

- Términos comunes en programación:
  - **Código fuente**: Lo que escribe el programador.
  - **Compilar**: Generar un ejecutable a partir del código fuente.
  - **Ejecutable**: Programa en código máquina que se ejecuta.
  - **Función**: Parte de código que se encarga de realizar una tarea especifica.
  - **Biblioteca**: Funciones externas que realizan ciertas tareas.
  - **Algoritmo**: Secuencia de acciones para solucionar un problema.

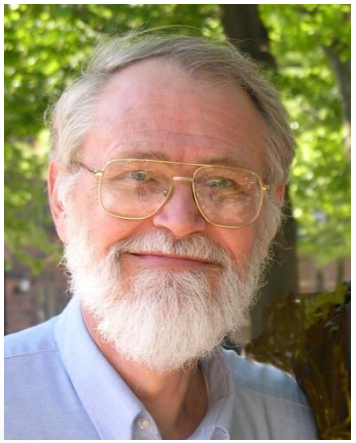
Índice

Introducción al lenguaje de programación C

- Ventajas de C
  - Es pequeño, eficiente y estable.
  - Hay mucho código C escrito.
  - Es la base de muchos otros como son C++, Java, AWK o PHP.
  - C es muy usado en sistemas empotrados.
- Desventajas de C
  - Es cercano a lenguajes de bajo nivel.
  - Poco estructurado: difícil de aprender.

Introducción al lenguaje de programación C

- Algo de historia…
  - Creado por Brian Kernighan y Dennis Ritchie en los laboratorios AT&T.
  - Originariamente se creó para codificar UNIX.
  - Se describió en la primera edición del K&R y esta descripción fue usada como estándar (1978)



Brian Kernighan

Dennis Ritchie

Índice

Tipos Básicos de datos

- Los programas necesitan almacenar datos. Cada dato se guarda en una **variable**.
- El *tipo de valor* que almacena la variable es llamado **tipo de dato**.

- Algunos tipos de datos básicos en C:
  - Entero (*Integer*): **int (5 19 50321)**
  - Coma flotante (*Float*), o número real: **float (1.98 3.1415 1.6E19)**
  - Carácter (*Character*): **char (a b d)**

Tipos Básicos de datos

Definición	Tipo	Tamaño	Desde	Hasta
char	Entero	8	-127	127
unsigned char	Entero	8	0	255
short	Entero	16	-32768	32767
unsigned short	Entero	16	0	65535
int	Entero	32	-2147483648	2147483647
unsigned int	Entero	32	0	4294967295
long	Entero	32/64	-9.223.372.036.854.775.808	9.223.372.036.854.775.808
unsigned long	Entero	32/64	0	18.446.744.073.709.551.615
float	Real	32	$3{,}4 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$
double	Real	64	$1.7 \cdot 10^{-208}$	$1.7 \cdot 10^{208}$

Índice

Variables

- Las variables se usan para almacenar un valor de un dato en un programa.
- Para definir una variable se necesita especificar:
  - El **tipo de dato** que va a almacenar, por ejemplo: `int`, `char`, `unsigned long`, etc.
  - El **nombre** de la variable.
- Es importante tener en cuenta:
  - Una vez que la variable es definida, su tipo de dato correspondiente no puede cambiar.
  - Si se necesita otro tipo de dato habrá que definir otra nueva variable.
  - Los tipos de datos básicos solo permiten almacenar **un valor**. Para almacenar *conjuntos de valores* existen otros tipos como **arrays** o **punteros**.

Variables

- El nombre de la variable nos permite hacer referencia al valor almacenado. Se recomienda poner nombres identificativos.
- Se pueden usar letras y dígitos.
- Suelen escribirse en letras minúsculas.
- Mayúsculas y minúsculas son diferentes, C es "*case sensitive*". "*variable*" es diferente a "*Variable*".

- Ejemplos de definición de variables:
  - `int var1;`
  - `char miVariable, mivariable, MiVariable;`
  - `unsigned char var2;`

Variables

- Es muy importante inicializar las variable asignándolas un valor inicial.
- La asignación se realiza utilizando el símbolo '=' y puede realizarse:
  - Cuando la variable es declarada:

```
int var1 = 5;
```

  - En cualquier parte del código:

```
int var1;
var1 = 5;
```

- El **valor** de la variable puede cambiar durante la ejecución.
- Es posible definir mas de una variable en una sola línea de código:

```
int var1=5, var2, var3, var4=6;
```

Índice

Operadores Aritméticos

- Una vez que se han definido un conjunto de variables se pueden trabajar con ellas.

Operadores Aritméticos	
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo

```
int var1 = 5;
int var2 = var1 – 3;
int var3;
int var4 = 10;
int var5 = var1 * var4;
var3 = var5 % var2;
```

Operadores Lógicos

Operadores lógicos	
==	Igual
!=	Distinto
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
&&	Y (AND)
||	O (OR)
!	Negación (NOT)

- Resultado lógico: VERDADERO (TRUE) o FALSO (FALSE).
- En C no existen variables lógicas:
  - FALSE se representa como 0
  - VERDADERO se representa como 1 (o cualquier otro valor ≠ 0)
- El valor de las funciones lógicas se expresan con **tablas de verdad**

A	B	A && B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

A	B	A || B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Otros Operadores

Otros operadores	
<varNombre>++	Usar e incrementar
<varNombre>--	Usar y decrementar
++<varNombre>	Incrementar y usar
--<varNombre>	Decrementar y usar

Otros operadores			
+=	Sumar y asignar	`var1 += 3`	`var1 = var1 + 3`
-=	Restar y asignar	`var1 -= 3`	`var1 = var1 - 3`
*=	Multiplicar y asignar	`var1 *= 3`	`var1 = var1 * 3`
/=	Dividir y asignar	`var1 /= 3`	`var1 = var1 / 3`

Índice

Arrays Unidimensionales

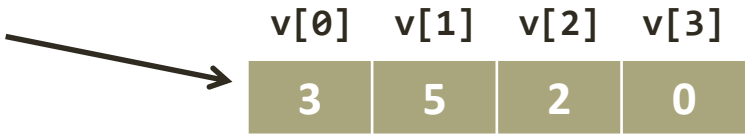
- Almacena un conjunto de datos del mismo tipo en posiciones de memoria consecutivas.
- Se accede a sus elementos por el índice y se pueden manejar como variables.
- El primer elemento siempre tiene el índice 0.

- Definición:

  tipo nombre[tamaño];

- Ejemplos:

```
int a[10];
int v[4] = {3, 5, 2, 0};
int v[] = {5,-2,2};
```



Arrays Multidimensionales

- Es un array de arrays, similar a una matriz.
  - Las matrices tienen dos índices (fila y columna).
- Se accede a sus elementos a través de sus dos índices.

```
m[2][3] = 5
```

- Definición:

tipo matriz [n_filas] [n_columnas];

- Ejemplos:

```
int m[3][4] ;
```

m_{00}	m_{01}	m_{02}	m_{03}
m_{10}	m_{11}	m_{12}	m_{13}
m_{20}	m_{21}	m_{22}	m_{23}

Índice

Concepto de Estructura

- Las ***matrices*** agrupan datos del **mismo tipo**.
- Si se necesita agrupar **tipos diferentes** la solución es utilizar ***estructuras***.
- Las ***estructuras*** son agrupaciones de variables que nos permiten manejar datos complejos..
  - Es una variable compuesta de variables.
  - Puede contener variables de varios tipos.
  - Cada variable tiene un nombre.
- Ejemplos:
  - Rádar de tráfico (matrícula - velocidad)
  - Sensor temperatura (temperatura - hora)
  - Televisión (canal - nombre - frecuencia)

Declaración de una estructura

- Se declaran con **struct:**

```
struct nombre_estructura {
     tipo nombre_var_1;
     tipo nombre_var_2;
     . . .
     tipo nombre_var_n;
} ;
```

- Ejemplo:

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

void main() {
     fecha fecha_nacimiento ;
     // . . .
}
```

Manejo de estructuras

- Las ***variables*** de una estructura se llaman ***miembros.***
  - Se acceden con el operador ***“.”***
  - Se manejan como cualquier variable.
- Ejemplo:

```
struct s_persona {
      int edad;
      float peso;
} ;
typedef struct s_persona persona;

int main() {
      persona p;
      p.edad = 20;
      p.peso = 83.3;
      printf("% d", p.edad);
      printf("% f ", p.peso);
}
```

Índice

Printf( )

- `printf( )` se utiliza para imprimir por pantalla.
- En el ejemplo anterior, `printf("Hola mundo\n");` se imprime por pantalla la cadena de texto "Hola mundo".
- El símbolo '`\n`' es usado para hacer un salto de línea.
- Si se quiere imprimir el valor de una variable:

```
printf( <tipoDato>, <nombreVariable> )
```

- Donde tipoDato es el tipo de dato de la variable <nombreVariable>:
  - `%s` -> proviene de String y es una cadena de caracteres.
  - `%d, %i` -> int (entero)
  - `%f` -> float
  - `%u` -> unsigned int

Scanf( )

- **`scanf( )`** se utiliza para capturar una secuencia de caracteres escrita por teclado.

- Lee caracteres hasta que se pulsa <Enter>

  `scanf ( <tipoDato>, <nombreVarDestino> )`

- Donde tipoDato es el tipo de dato de la variable <nombreVarDestino>:
  - **`%s`** -> proviene de String y es una cadena de caracteres.
  - **`%d, %i`** -> int (entero)
  - **`%f`** -> float
  - **`%u`** -> unsigned int

Índice

Escribiendo tu primer programa

- Primeramente, abrir un editor de texto (puede ser **vi** o **gedit**) y guarda el fichero como "*miPrograma.c*"
- Escribe lo siguiente:

```
#include <stdio.h>
int main(){
   printf("Hola mundo\n");
   return 0;
}
```

- Este programa utiliza una libreria externa llamada **stdio.h**
- El programa empieza en *main()*
- *printf* imprime una cadena de texto.
- *return* finaliza la ejecución

Escribiendo tu primer programa

- Una vez que salves el archivo, abre un terminal y posiciónate en el directorio que contiene el archivo *miPrograma.c*.
- Compila el programa con el comando *gcc* (GNU Compiler Collection):

```
gcc miPrograma.c –o miPrograma
```

- Este comando compila el programa *miPrograma.c* y genera el fichero ejecutable llamado *miPrograma* solamente si el código es correcto. Si el código contiene errores estos se mostraran por pantalla y no se genera el fichero ejecutable *miPrograma*.

- Para ejecutar el programa escribir:

```
./miPrograma
```