INSTITUTO DE INGENIERÍA ELÉCTRICA

FACULTAD DE INGENIERÍA UDELAR

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Lenguaje C

Sistemas embebidos para tiempo real

NEON
Network of Competence on Internet of Things

Co-funded by the Erasmus+ Programme of the European Union

Clasificación de lenguajes

- Nivel
  - **alto nivel**: **C**, C++, Pascal, FORTRAN
  - bajo nivel: ensamblador, código de máquina.
- Compilación
  - **compilados**: **C**, C++, Pascal, FORTRAN
  - pre-compilados e interpretados: Java (bytecode)
  - interpretados: Phyton, Matlab, Perl, bash,...
- Otros: Tipeo (fuerte, débil), etc.

Primer programa: primer.c

primer.c

```
#include <stdio.h>

void main(void) {
    printf("Hola mundo!!\n");
}
```

`#include <stdio.h>` // archivo de encabezado (stdlib.h, math.h, etc.)

`main()` // main es una función, el punto de entrada al programa

`{}` // paréntesis, son para indicar el comienzo y el fin de un bloque, como el cuerpo de una función

`Printf()` // es el llamado a la función imprimir

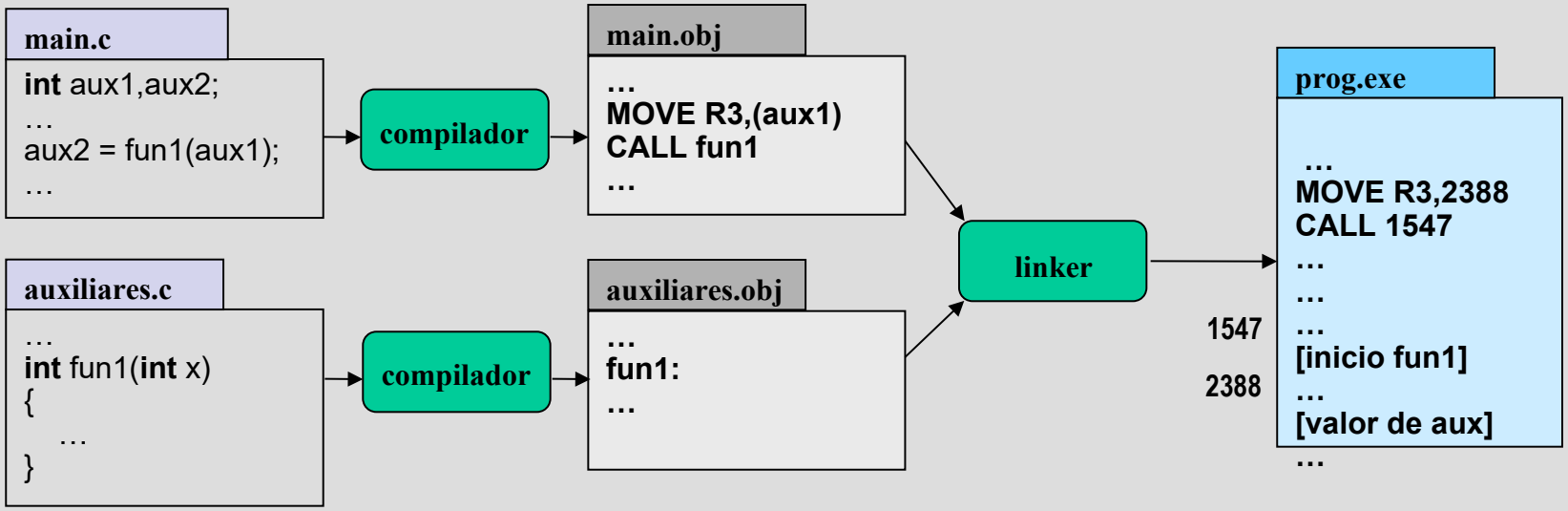
Otro programa: segundo.c

segundo.c

```
#include <stdio.h>
#define  NUM 2
void main(void) {
   int a, b, suma;
   a = NUM;
   b = 2;
   suma = a + b;
   printf("suma vale %d", suma);
}
```

```
#define  NUM 2      // directiva al preprocesador, sustituye NUM por 2
int a, b, suma  // declara las variables enteras: a, b y suma
B = 2;             // define las variables y les asigna valor
suma = a + b;   // realiza la suma
printf();          // imprime el resultado
```

Proceso de compilación



Declaración/Definición

- Declaración: especifica al compilador que un variable o función existe.

```
int i, j;
int aux;
int optimo(int a, int b);
```

- Definición: la variable o función misma.

```
int optimo(int a, int b){
    int aux = a;
    /* sigue... */
    return aux;
}
```

Atributo	Descripción
Tipo	char, int, unsigned int, etc.
Nombre	Identificador para acceder el objeto.
Valor	Datos contenidos en el objeto.
Dirección	La ubicación en memoria donde reside el objeto.
Alcance	El código fuente donde el nombre del objeto es reconocido.
Tiempo de vida	Cuando el objeto está disponible (dependiendo de cuando el objeto es creado y destruido).

Variables: tipos de variables

- Tipo de variables fundamentales:
  - enteros: **`int`**, **`char`**.
  - flotantes: **`float`**, **`double`**.
- Modificadores (adjetivo):
  - **`short`**, **`long`**, **`unsigned`**, **`signed`**.
  - No todas las combinaciones de tipos y modificadores son válidas.

Tipos de datos enteros

Tipo de dato		Tamaño	Rango
unsigned	char	8 bits	0 a 255
	short int	16 bits	0 a 65,535
	int	16 bits o 32 bits	Idem unsigned short int; Idem unsigned long int.
	long int	32 bits	0 a 4,294,967,295
signed	char	8 bits	-128 a +127
	short int	16 bits	-32,768 a +32,767
	int	16 bits o 32 bits	Idem signed short int; Idem signed long int.

- Atención:

  `int` tamaño del bus del CPU.
  `short int` menor o igual a `int`.
  `long int` mayor o igual a `int`.

Operadores: asignación

- Operadores: =, +=, -=, *=, /=
  - Ejemplos:

```
int i = 1;
i += 2;  //equivalente a: i = i + 2;
i *= 10;     //equivalente a: i = i * 10;
```

Operadores: aritméticos

- Operadores: +, -, *, /, %
  - Ejemplos:

```
int a,b,n;
a = 10;
b = 3;
n = a%b; // resultado: n = 1;
```

- Operadores pos, pre-incremento: ++, --
  - Ejemplos:

```
int i = 10;
i++;// equivalente a: i = i + 1;
```

Operadores: relacionales

- Operadores: ==, !=, <, <=, >, >=
  - Ejemplo:

```
int x, y, b;
x = 10;
y = 3;
b = (x==y);  // resultado: b = 0;
```

  - Observación:
    - No existe el tipo boolean: resultado entero
      - false: `0`
      - true: `!0`
    - `a = b` no es lo mismo que `a == b`

Operadores: lógicos

- Operadores: `&&` (AND), `||` (OR), `!` (NOT).
  - Ejemplos:

```
int x, b;
// sigue: x = ...
b = (x < 0) && (x >10);
```

  - Observación:
    - Expresiones con operadores lógicos pueden ser corto circuitadas (algunos compiladores).
      Por ejemplo: si `x < 0` es falso, toda la expresión será falsa.

Operadores: manipulación de bits

- Permiten la manipulación de bits individuales de una variable.
- Operadores:
  - | (OR), & (AND), ~ (NOT), ^ (XOR), >> (RRA), << (RLA)
  - Ejemplos:

```
mask = (1<<7);        // mask = 0x80.
bits = bits | mask; // bits = bits OR mask.
                    // setea el bit 7 de bits.
```

Operadores lógicos y “bitwise”

Operación	Operador lógico	Operador “bitwise “
AND	&&	&
OR	||	|
XOR	No definido	^
NOT	!	~

- Ejercicio:
  - `(5 || !(3)) && 6`
  - `(5 | ~(3)) & 6`

Operadores: conversión de tipos

- Conversión de tipos: “casting”
- Promoción: no hay pérdida de precisión
  - Ejemplos:

```
int i = 1;
float f = i;
```

- Degradación: hay pérdida de precisión
  - Ejemplos:

```
float f = 4.3;
int i = f;
```

  - Pregunta:
    - ¿se señala un warning?

Control de flujo

- Secuencia
  - `{...}`
- Condicionales
  - `if-else`, `switch-case`
- Iteraciones
  - `while`, `do-while`, `for`

Control de flujo

- Condicionales: `if(cond)-else`,
  - Ejemplos:

```
int x, y;
if (x < 0){
   y = x;
} else {
   y = x*10;
}
```

Control de flujo

- Condicionales: `switch-case`
  - Ejemplo:

```
char c;
//...
if (c == 'a'){
    // sentencias si 'a'
} else if (c == 'b'){
    // sentencias si 'b'
} else if (c == 'c'){
    // sentencias si 'c'
} else {
    // otras letras
}
```

```
char c;
//...
switch ( c ) {
case 'a':
   // sentencias si 'a'
   break;
case 'b':
   // sentencias si 'b'
   break;
case 'c':
   // sentencias si 'a'
   break;
default:
   // otras letras
}
```

Control de flujo

- Iteración: `while(cond)`, `do-while(cond)`
  - Ejemplos:

```
int cond;
// cond = ...; Se determina si entra a la iterac.
while (cond){
  // sentencias a repetir y modificación de cond
}

int cond; // siempre se entra una vez a la iterac.
do {
  // sentencias a repetir y modificación de cond
while (cond);
```

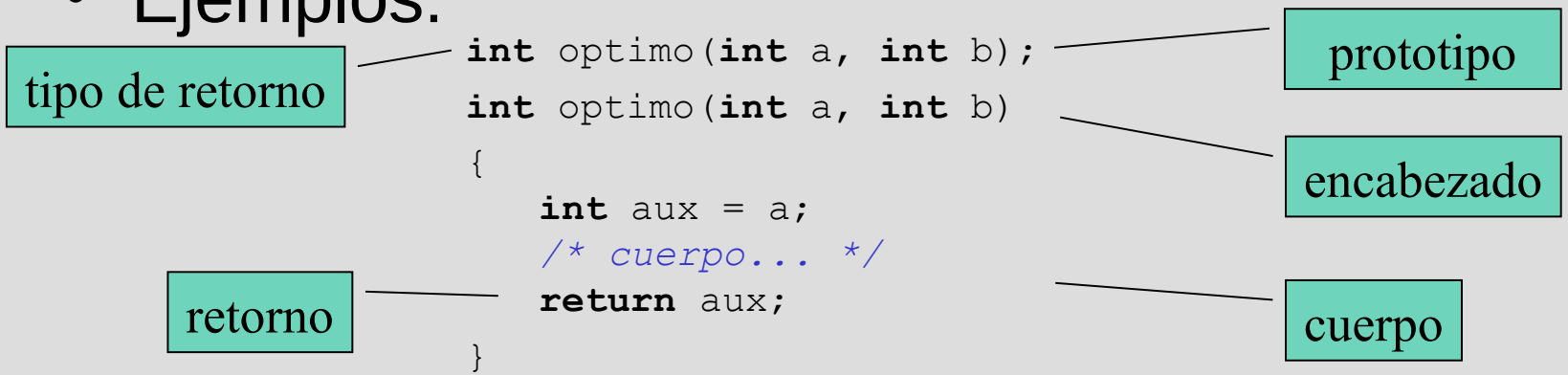
Control de flujo

- Iteración: `for(ini;cond;modif)`
  - Ejemplos:

```
int i;
// cond = ...; Se determina si entra a la iterac.
for(i = 0; i<MAX;i++){
  // sentencias a repetir MAX veces.
  // i pude ser utilizado como índice.
}
for(;;){
  // bucle infinito.
}
```

Funciones

- Declaración y definición
  - declaración: prototipo (interfaz), función abstracta.
  - definición: la función misma.
- Ejemplos:



Funciones

- Prototipo, definición y llamada
  - Ejemplos:

```
int optimo(int a, int b);

main(){
        /* codigo... */
        c = optimo(a,b);
}

int optimo(int a, int b)
{
        /* cuerpo... */
        return aux;
}
```

prototipo

llamada

definición

Arreglos

- Arreglo: colección de elementos del mismo tipo (i.e. `char`, `int`, etc.)
  - ubicados en bloques contiguos de memoria
  - referenciados con nombre único e índice.
  - Ejemplos:

```
#define TAM  10
int ai1[] = {1, 2, 3, 4};

int ai2[TAM];
ai2[0] = 1;

char frase[] = "alarma";
```

Punteros

- Puntero: es una variable
  - guarda la dirección de memoria de otra variable (específica)
  - cuando se declara no guarda espacio para la variable.
- Operadores
  - * (referencia): accede al contenido de la dirección de memoria guardado por el puntero.
  - & (desreferencia): obtiene la dirección de memoria de una variable.
  - Ejemplos:

```
int i = 10;
int* pi;  // equivalente a int *pi
          // pi es una var. int* (puntero a int) o *pi es int
pi = &i;  // pi contiene la dirección de i
*pi = 10; // escribe 10 en la dirección guardada por pi
```

Relación entre arreglos y punteros

- El nombre de un arreglo es un puntero al inicio del bloque de memoria del arreglo

```
int ai[] = {1, 2, 3, 4, 5};
int* pi;

pi = ai;     // equivalente a pi = &ai[0];
```

- Aritmética de punteros: útil para recorridas
  - Ejemplos (continuación):

```
*pi = 10;    // equivalente a ai[0] = 10;
 pi++;
*pi = 12;    // equivalente a ai[1] = 12;
```

String: cadena de caracteres

- No está soportado directamente por C.
- Strings:
  - arreglo de caracteres
  - deben terminar en carácter nulo (`NULL`), que indica fin de `string`.
- Ejemplos:

```
char* frase = {'H', 'o', 'l', 'a', '\0'}; // equivalente a:
char  frase[] = "Hola";    // compilador agrega caract. '\0'
```

- La biblioteca `string.h` provee funciones para manipulación de cadenas:
  - Ejemplos:

```
char *strcpy(char *dest, const char *src);
int strcmp(const char *s1, const char *s2);
int atoi(const char *nptr);
```

Variables

- local vs. global (visibilidad o alcance)
  - local: definida dentro de un bloque (típicamente una función), alcance = propio bloque (función).
  - global: definida fuera de toda función, accesible desde todas las funciones del archivo.
- auto vs. static (vida y almacenamiento)
  - auto: dura mientras dure la función, guardada en el stack o también registros (locales por omisión).
  - static: dura siempre, guardada en un lugar fijo de memoria (globales o locales declaradas static).

Variables

- Cualificadores: `const`, `volatile`.
  - `const` - guarda la variable en un área constante de memoria y hace que la misma sea inmodificable.
  - `volatile` - indica al compilador que este valor puede ser modificado por fuera del control del programa (evita optimizaciones). Esto, por ejemplo, puede ocurrir con variables modificadas dentro de una rutina de atención a interrupción o que son accedidas por dispositivos memory-mapped I/O, como puertos de comunicaciones.

Volatile

```
int exit = false;

void main (void){
   …
     While (!exit) {
        … // Se hacen cosas que no modifican exit
   }
}

#pragma ...
void interrupt_handler(void) {
   exit = true;
}
```

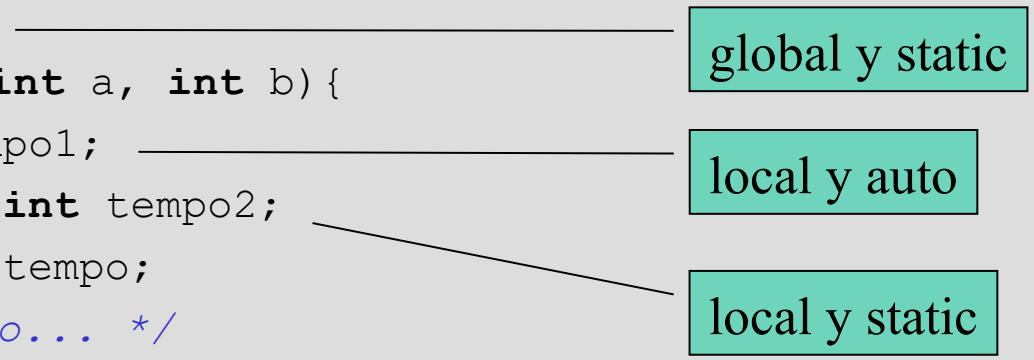
Volatile

```
volatile int exit = false;

void main (void){
   …
     While (!exit) {
        … // Se hacen cosas que no modifican exit
   }
}

#pragma ...
void interrupt_handler(void) {
   exit = true;
}
```

Variables

- Ejemplos:

```
int var1;
void func(int a, int b){
    int tempo1;
    static int tempo2;
    var1 = tempo;
   /* código... */
}
```



- Observaciones:
  - global siempre son static.
  - local puede ser auto (por omisión) o static.

Variables

main.c

```
int fun1();
int fun2();
int base;
void main(void) {
    int var = 1;
    local += fun1();
}
int fun1(){
    int var = 2;
    return (base+var);
}
```

global y static

local y auto

local y auto
diferente local de main

funciones.c

```
extern int base;
int fun2(){
    return base++;
}
```

extern: accede a variable
global de otro archivo

Variables: tipo de almacenamiento

- `extern` - hace que la variable especificada acceda a la variable del mismo nombre de otro archivo (da acceso global de archivo).
- `static` - hace que la variable o función tenga alcance solamente de archivo.
  - Ejemplos:

```
extern int varglobal;
static int funcion();
```

Modularización

- Evitar uso de variables globales
- Utilizar funciones de acceso a variables privadas (protección)
- Separar entre archivo.c y archivo.h

Modularización

- archivo.c - implementación del módulo
  - definición de variables globales, locales y de acceso archivo (static)
  - definición de funciones públicas y privadas (acceso archivo: static)
  - incluir su propio encabezado para verificación (incluir archivo.h en archivo.c)
- archivo.h - interfaz pública del módulo
  - declaración variables externas (globales)
  - funciones públicas (no declaradas static en .c)
  - Utilización de “Guarda”

Módularización

- “Guarda” para archivo.h:
  - Para evitar múltiples includes.

```
#ifndef SRC_ARCHIVO_H_
#define SRC_ARCHIVO_H_
… // acá va la interfaz pública de archivo.h
#endif
```

Módulo

funciones.h

```
#pragma once
extern int varglobal;
int fun1();
```

declaración (externa)

En funciones **extern** por omisión, no es necesario

funciones.c

```
#include "funciones.h"
int varglobal = 10;
static int fun2();
int fun1(){
    int local = 2;
    return (global+local);
}
static int fun2(){
    int local = 2;
    return (global+local);
}
```

definición (global)

main.c

```
#include "funciones.h"
main(){
    int i = varglobal;
    i = fun1();
}
```

uso de variable global externa

uso de función de otro modulo

Deberes: modularizar

main.c

```
#define N 10
int buf[N];

void set_default_channel(int ch);
void load_default_channel();
void set_channel(int ch);
int read_channel();

int default_channel;
int current_channel;

int main (void){
    int i;
    int channel;
    set_default_channel(1);
    set_channel(2);
    for(i=1;i<N;i++){
      buf[i] = read_channel();
    }
}

//sigue
```

```
//cont

void set_default_channel(int ch){
    default_channel = ch;
}

void load_default_channel(){
    current_channel = default_channel;
}

void set_channel(int ch){
    current_channel = ch;
}

int read_channel(){
    current_channel = default_channel;
}
```

Estructuras

- Colección de items de diferente tipo.

```
struct [etiqueta] {
  tipo1 miembro1;
  tipo2 miembro2;
  ...,
} [variable];
```

Estructuras: ejemplos

a)

```
struct complex  {
    float real;
    float imag
} c1, c2;
struct complex  c3;
 c1.real = 1.0;
 c1.imag = 3.1416;
```

b)

```
typedef struct complex {
    float real;
    float imag;
} complex_t;
complex_t        c1, c2;
c1.real = 1.0;
c1.imag = 3.1416;
```

c)

```
char* str = “este es el mensaje”;
typedef struct {
    char type;
    char payload[TAM];
} message_t;
message_t  m;
m.type = ‘0’;
strcpy(m.payload,str);
```

Uniones

- Declaración y uso igual a las estructuras.
  - se utiliza un solo miembro por vez
  - los miembros comparten la memoria

```
union [etiqueta] {
  tipo1 miembro1;
  tipo2 miembro2;
  ...,
} [variable];
```

Uniones: ejemplos

```
union numero  {
    int i;
    float f;
} n;
int i1;
n.i = 1;
n.f = 1.0;
i1 = n.i;
```

```
typedef struct {
  unsigned char tipo;
  union {
        int entero;
        float flotante;
  } contenido;
} numero;
numero n;
//...
if (n.tipo == ENTERO)
    n.contenido = 1;
else
    n.contenido = 1.0;
```

```
typedef struct {
    unsigned char type;
    union {
      char cmd;
      char data[TAM];
    } payload;
} message_t;
message_t msg1;
msg1.type = CMD;
msg1.payload.cmd = RADIO_ON;
//...
msg1.type = DATA;
msg1.payload.data = read_value;
```

Enumeraciones

- Enums:
  - Conjunto de nombres asociados a enteros constantes
  - Alternativa a **#define** pero automático

Ejemplos:

```
enum e_months {JAN=1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV,DEC};
typedef enum e_months month_t;
month_t currentmonth;
currentmonth = JUN;   /* equivalente a currentmonth = 6; */
printf("%d\n", currentmonth);
```

Bibliografía

- “First Steps with Embedded Systems”, ByteCraft Limited
  - http://www.bytecraft.com/Byte_Craft_Limited_Publishing
  - Libro en pdf gratis.
- "Lenguaje Programacion C”, Kernighan & Ritchie,
  - Pearson Education, 1991.
  - ISBN: 9688802050
- Wikibooks:
  - C Programming (Wikibook).
    - http://en.wikibooks.org/wiki/C_Programming
  - A Little C Primer (Wikibook)
    - http://en.wikibooks.org/wiki/A_Little_C_Primer
  - Programación en C
    - http://es.wikibooks.org/wiki/Programación_en_C