Universidad Francisco de Vitoria
UFV Madrid

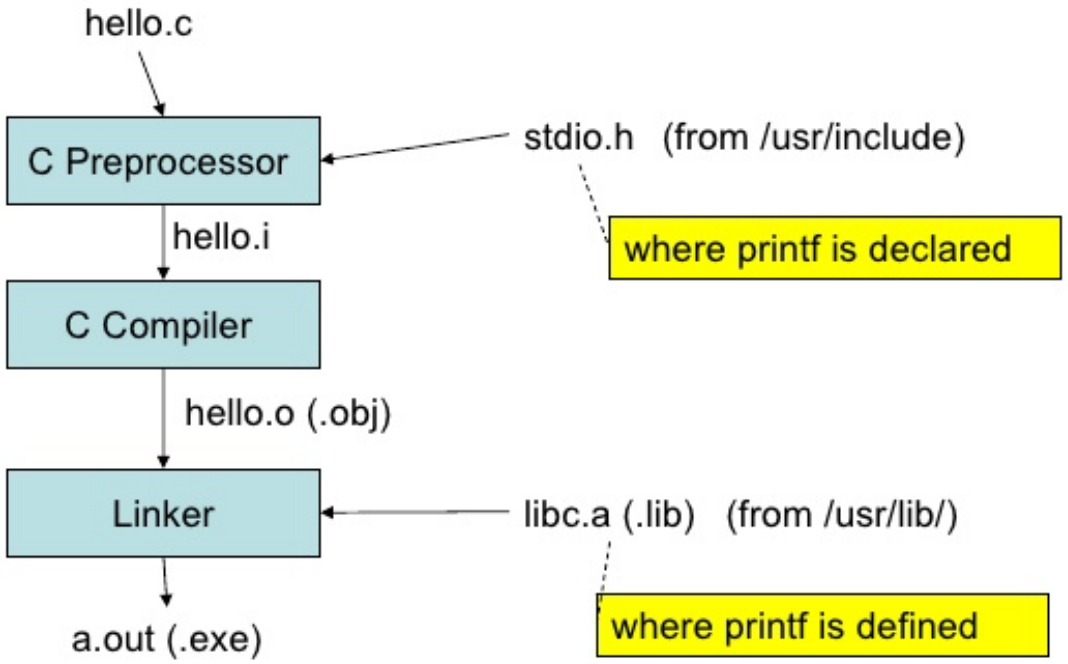
# Elementos básicos de C

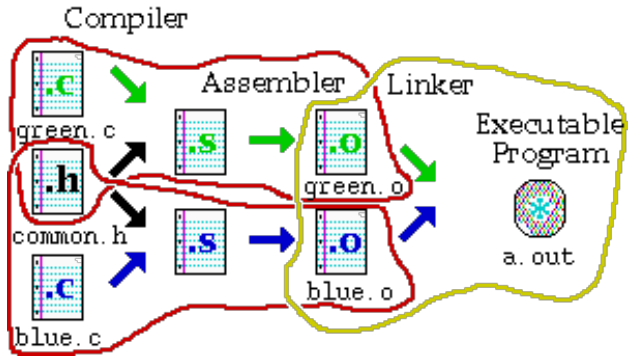
Óscar Marbán (oscar.marban@ufv.es)

# Formato de un programa

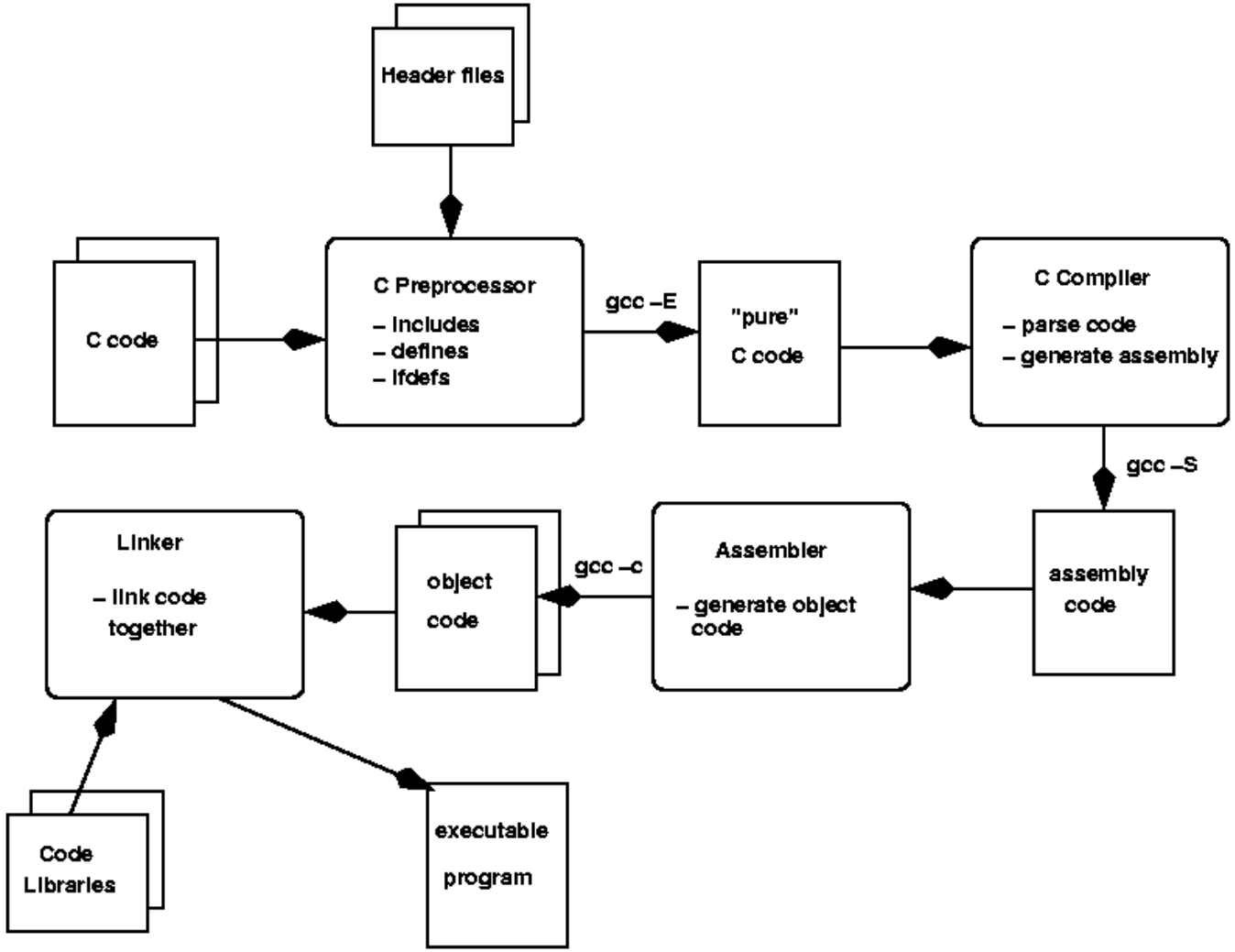
```
1  llamadas a bibliotecas (#include)
2  declaración de funciones (prototipos de funciones)
3  declaración de variables globales
4
5  main() {
6      declaración de variables locales
7      sentencias
8  }
9  definición de funciones
```

# Pasos de compilación y ejecución



# Pasos de compilación y ejecución



# Identificadores

- Un identificador es una secuencia de letras y dígitos (‘_’ se considera letra)
- Deben empezar por una letra (‘_’ no es recomendable. Lo usan las las bibliotecas)
- C distingue entre mayúsculas y minúsculas (CASE SENSITIVE). El identificador *id* es diferente de *ID*
- Por convención las variables en minúsculas y las constantes en mayúsculas
- Las palabras reservadas if, else . . . no pueden usarse como nombres de variables

# Palabras reservadas

| | | | |
|---|---|---|---|
| `auto` | `double` | `int` | `struct` |
| `break` | `else` | `long` | `switch` |
| `case` | `enum` | `register` | `typedef` |
| `char` | `extern` | `return` | `union` |
| `const` | `float` | `short` | `unsigned` |
| `continue` | `for` | `signed` | `void` |
| `default` | `goto` | `sizeof` | `volatile` |
| `do` | `if` | `static` | `while` |

CUANDO ESCRIBÍ ESTE CÓDIGO,
SÓLO DIOS Y YO SABÍAMOS
CÓMO Y PARA QUÉ LO HICE

AHORA, SÓLO DIOS LO SABE

# Comentarios

- Los comentarios en C pueden ocupar varias líneas y se encuentran delimitados entre /* y */

```
int main()
{
  /* Esto es un comentario de varias
     líneas.*/
  return(0);
}
```

*"Documentation is a love letter that you write to your future self." - Damian Conway* 💌

Buscar *funny source code comments*

Code never lies,
comments sometimes do.

*- Ron Jeffries*

```
# you may think that this function
# is obsolete, and doesnt seem to do
# anything. and you would be correct.
# but when we remove this funtion
# for some reason the whole program
# crashes and we cant figure out why,
# so here it will stay.
```

# Tipos de datos

| Data Type | Range | Bytes | Format Specifiers |
|---|---|---|---|
| char | -128 to 127 | 1 | %c |
| unsigned char | 0 to 255 | 1 | %c |
| short signed int | -32,768 to 32,767 | 2 | %d |
| short unsigned int | 0 to 65,535 | 2 | %u |
| signed int | -32768 to 32767 | 4 | %d |
| unsigned int | 0 to 65535 | 4 | %u |
| long signed int | -2147483648 to 2147483647 | 4 | %ld |
| long unsigned int | 0 to 4294967295 | 4 | %lu |
| float | -3.4e38 to 3.4e38 | 4 | %f |
| double | -1.7e308 to 1.7e308 | 8 | %lf |
| long double | -1.7e4932 to 1.7e4932 | 8 | %Lf |
| **Note: The sizes in this figure are for 32 bit compiler** | | | |

**Los rangos dependen del compilador**

# Expresiones constantes

- int
  - Base decimal: 230, -17
  - Base hexadecimal: 0x3A0 (comienza por cero-x)
  - Base octal: 0210 (comienza por cero)
- long
  - A una expresión de tipo long se le añade un L al final: 200L
- double, float
  - Notación decimal: 2.56, -3.14
  - Notación científica: 2.45E-4, -3E10

# Expresiones constantes

- char
  - se define entre comillas simples
  - ASCII: ’A’, ‘a’
  - Comillas: ‘\”’, ‘\”, ‘\\’
  - Especiales: ‘\b’, ’n’, ‘r’, ’t’, ‘\0’, ‘\a’

# Declaración de variables

- *modificador_tipo tipo_dato nombre_var [= valor_ini]*
- Declaración simple
  - char c;
  - unsigned int i;
- Declaración múltiple
  - char c,d;
  - unsigned int i,j,k;
- Declaración y asignación
  - char c=’A’;
  - unsigned int i=133,j=1229;

# Declaración de constantes

- Como directiva del preprocesado
  - #define PI 3.14
- Como constante
  - const float PI = 3.14;

# Operadores aritméticos

- Asignación: =
- Los operadores aritméticos son:
  - Suma (+)
  - Resta (-)
  - Multiplicación (*)
  - División (/)
  - Módulo o resto de la división entera (%)

# Operadores aritméticos

- División entera vs división real
  - Depende de los operandos
    - 4 / 3 --> 1 entero
    - 4.0 / 3 --> 1.333 real
    - 4 / 3.0 --> 1.333 real
    - 4.0 / 3.0 --> 1.333 real

# Operadores aritméticos

- Pre/post-incrementos
  - ++ y -- representan operaciones de incremento y decremento, respectivamente
  - a++;  /* similar a a=a+1 */
  - Ejemplos:
  - a=3; b=a++;  /* a=4, b=3 */
  - a=3;  b=++a;  /* a=4, b=4 */
  - a=3; b=a--;  /* a=2, b=3 */

# Operadores aritméticos

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int a=10;
  int b;

  b=a++;
  printf("a=%d\nb=%d\n", a, b);
}
```

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int a=10;
  int b;

  b=++a;
  printf("a=%d\nb=%d\n", a, b);
}
```

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int a=10;
  int b=5;

  b=a++ + ++b;
  printf("a=%d\nb=%d\n", a, b);
}
```

# Operadores aritméticos

- Asignación: =
  - a=b+3;
- Existen otras variantes de asignación:
  - a+=3;  /* Equivalente a a=a+3 */
  - a*=c+d;/* Equivalente a a=a*(c+d) */
  - a/=a+1;/* Equivalente a a=a/(a+1) */

# Operadores relacionales

- Los operadores de comparación en C son
  - Igual: ==
  - Distinto: !=
  - Mayor: >
  - Mayor o igual: >=
  - Menor: <
  - Menor o igual: <=
- El resultado de un operador de comparación es un valor entero (0 es falso y distinto de 0 verdadero)
  - a=3>7 /* a vale 0 (falso) */

# Operadores lógicos

- Sobre expresiones lógicas (enteros) se definen los siguientes operadores lógicos
  - And lógico: &&
  - Or lógico: ||
  - Negación lógica: !
- Ejemplo
  - a=(3>2 || 5==4) && !1 /* Falso */
- C tiene un modelo de evaluación perezoso
  - a=3>2 || w==4 /* w==4 no se evalúa */

| A | B | A&&B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| A | B | A\|\|B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| A | !A |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Precedencia de operadores

| *Operator* | *Description* | *Associativity* |
|---|---|---|
| `()`<br>`[]`<br>`.`<br>`->`<br>`++ --` | Parentheses (function call) (see Note 1)<br>Brackets (array subscript)<br>Member selection via object name<br>Member selection via pointer<br>Postfix increment/decrement | left-to-right |
| `++ --`<br>`+ -`<br>`! ~`<br>`(type)`<br>`*`<br>`&`<br>`sizeof` | Prefix increment/decrement<br>Unary plus/minus<br>Logical negation/bitwise complement<br>Cast (change *type*)<br>Dereference<br>Address<br>Determine size in bytes | right-to-left |
| `* / %` | Multiplication/division/modulus | left-to-right |
| `+ -` | Addition/subtraction | left-to-right |
| `<< >>` | Bitwise shift left, Bitwise shift right | left-to-right |
| `< <=`<br>`> >=` | Relational less than/less than or equal to<br>Relational greater than/greater than or equal to | left-to-right |
| `== !=` | Relational is equal to/is not equal to | left-to-right |
| `&` | Bitwise AND | left-to-right |
| `^` | Bitwise exclusive OR | left-to-right |
| `\|` | Bitwise inclusive OR | left-to-right |
| `&&` | Logical AND | left-to-right |
| `\|\|` | Logical OR | left-to-right |
| `?:` | Ternary conditional | right-to-left |
| `=`<br>`+= -=`<br>`*= /=`<br>`%= &=`<br>`^= \|=`<br>`<<= >>=` | Assignment<br>Addition/subtraction assignment<br>Multiplication/division assignment<br>Modulus/bitwise AND assignment<br>Bitwise exclusive/inclusive OR assignment<br>Bitwise shift left/right assignment | right-to-left |
| `,` | Comma (separate expressions) | left-to-right |

# Funciones de E/S

- Las funcionalidades de e/s en C no pertenecen a las palabras reservadas del lenguaje. Son funciones de librería *stdio.h*
  - Entrada: scanf()
  - Salida: printf()

# *printf()*

- printf(format, exp1, exp2, exp3,..., expn);

- format : Es la cadena de texto de formato de salida de los datos

- expi :  Es la expresión a incluir dentro del formato

# Ejemplo *printf()*

- Ejemplo:
  - int a=3;
  - float x=23.0;
  - char c=‘A’;
  - printf(“Hola mundo!!\n”);
  - printf(“Un entero %d\n”,a);
  - printf(“Un real %f \ny un char %c\n”,x,c);

# *printf()*

| Formato | Expresión | Resultado |
|---|---|---|
| %d %i | entero | entero decimal con signo |
| %u | entero | entero decimal sin signo |
| %o | entero | entero octal sin signo |
| %x %X | entero | entero hexadecimal sin signo |
| %f | real | real en notación punto |
| %e %E %g %G | real | real en notación científica |
| %c | carácter | carácter |
| %p | puntero | dirección de memoria |
| %s | string | cadena de caracteres |
| %ld %lu ... | entero largo | entero largo (distintos formatos) |

# *printf()*

- Otras opciones de formato
  - Precisión (número de decimales)
  - Justificación (izquierda o derecha)
  - Caracteres especiales (% o \)
- Ver página del manual
- man printf

# *scanf()*

- scanf(format, dir1, dir2, dir3,..., dirn);

- format : Es la cadena de texto de formato de entrada de los datos

- diri :  Es la dirección donde se almacena el resultado

# *scanf()*

- Ejemplo
  - int a,*pa;
  - float x;
  - char c;
  - scanf(“%d”,&a); /* Lee un entero y lo almacena en a */
  - scanf(“%f %c”,&x,&c); /* Lee x y c */

# Ejemplo

```c
#include <stdio.h>

main() {
    int a;
    float x, y;
    const float PI = 3.14;

    a = 14;
    printf("El valor de a es: %d\n", a);
    printf("Introduzca un numero entero: ");
    scanf("%d", &a);
    printf("El nuevo valor de a es: %d\n", a);
    printf("El valor de a * 2 es: %d\n", 2 * a);
    printf("Introduzca un numero decimal: ");
    scanf("%f", &x);
    y = 3.2;
    printf("El valor de x + y es: %8.2f\n", x + y);
}
```