



# Preprocesador

- Ejecuta antes que el compilador, modifica el fuente y su resultado es lo que ingresa al compilador.
- Al inicio el preprocesador:
  - Elimina todos los comentarios, reemplazándolos por un espacio en blanco
  - Las líneas terminadas con \ se unen a la línea siguiente
  - Se reemplazan las macro predefinidas por el valor que corresponda.



# Preprocesador

- Macros predefinidas, se reemplazan con un literal cadena con el valor que corresponda, algunos casos son:
  - `__FILE__` nombre del archivo (fuente o encabezado)
  - `__LINE__` número de línea
  - `__DATE__` fecha de compilación
  - `__TIME__` hora de compilación



# Constantes y nombres

- Se pueden definir “constantes” con la directiva `define` que en forma general: `#define nombre valor`  
`#define PI 3.14159265358`  
`#define IVA 0.21`
- Lo que hace el preprocesador es reemplazar todas las apariciones del nombre por el valor. Con nuestro ejemplo si en el fuente se tiene  
`circ = PI * diam;`  
Al compilador le llega  
`circ = 3.14159265358 * diam;`
- Si no se da un valor se reemplaza por nada (literalmente lo borra) pero el nombre queda en la tabla de nombres definidos
- La directiva `#undef` permite borrar nombres de dicha tabla



# Compilación Condicional

- Se puede consultar por el valor dado a nombres en un define o bien consultar si un nombre a sido definido
- El objetivo es elegir que partes del fuente llegarán al compilador.

```
#ifdef DEBUG_FULL
    printf("debug full %s\n", infofull);
#elif DEBUG_LEVEL > 2
    printf("debug nivel alto %s\n", infoalta);
#else
    printf("debug nivel bajo %s\n", infobaja);
#endif
```



# Macros

- Es darle forma de “función” a un código que va incrustado.
- De cierta manera era el modo de hacer una función inline cuando no existía esta característica.
- No controla tipos, hace un reemplazo textual, da algunas ventajas (simular sobrecarga de tipos) y varias desventajas (la falta de control puede terminar en una infinidad de errores)
- Hay que tener particular cuidado con la precedencia de los operadores al momento de invocar la macro, en como va a terminar expandiéndose.
- Otro problema es si tenemos efectos laterales



# Macros – Ejemplos

```
#define PORDOS(x) x * 2
```

```
r = PORDOS(4+3);
```

Expande como  $r = 4+3 * 2; //r = 10$

```
#define PORDOS(x) (x) * 2
```

Expande como  $r = (4+3) * 2 //r = 14$

```
#define MASTRES(x) (x) + 3
```

```
r = MASTRES(5) * 2;
```

Expande como  $r = (5) + 3 * 2; //r = 11$

```
#define MASTRES(x) ((x) + 3)
```

Expande como  $r = ((5) + 3) * 2; //r = 16$



# Macros – Ejemplos

```
#define max(a, b) ((a) > (b) ? (a) : (b))  
// Hoy dia muchos no usan mayúsculas para las  
macros
```

```
int x = 5;  
int y = 10;  
int z = max(x, y);  
Expande como int z = ((x) > (y) ? (x) : (y));  
//z = 10
```

```
z = max(x++, y++);  
Expande como z = ((x++) > (y++) ? (x++) : (y++));  
//z = 11 ya que el ? es punto de secuencia  
//En otros casos se puede caer en comportamiento  
//no definido
```



# Inclusiones

- Incluye el contenido de otro archivo en reemplazo de la directiva
- Una vez incluido el preprocesador lo analiza, de este modo se pueden hacer inclusiones “recursivas”
- Hay dos modos básicos:  
`#include <stdbool.h>`  
`#include "archivo.h"`
- El primer caso es para los encabezados de la biblioteca estándar de C (se pueden agregar otras manualmente) que el compilador sabe donde están.
- El segundo caso es para mis propios archivos y se los especifica con un camino (path) típicamente relativo.





# Guardas de inclusión

- Dado que un archivo puede incluir otro se corre el riesgo de incluir un archivo más de una vez y que esto produzca redeclaraciones u otros inconvenientes
- Para evitarlo:
  - `#pragma once` esta directiva colocada en el archivo evita duplicar su inclusión, es práctica y efectiva, pero a pesar de estar muy extendida no es parte del estándar
  - Usar compilación condicional: es parte del estándar, requiere más trabajo, se corre el riesgo de tener guardas duplicados

Si el nombre del archivo es `archivo.h`

```
#ifndef ARCHIVO_H_INCLUDED
#define ARCHIVO_H_INCLUDED
... contenido ...
#endif // ARCHIVO_H_INCLUDED
```



# Definición de Tipos

- En realidad alias de tipos de datos, usando typedef antepuesto a una definición común.

```
double velocidad; //velocidad es un double
typedef double velocidad;
//velocidad es alias un double
```

- No es un verdadero tipo de dato nuevo sino un alias, uno puede hacer

```
typedef double velocidad;
typedef double aceleracion;
```

```
velocidad v1 = 5.2;
aceleracion ac1 = v1 + 2.1;
```

```
/* no compilaría si aceleracion y velocidad fuesen
tipos de datos diferentes */
```



# Uso del typedef

- Tiene dos usos, uno del que nadie duda es una buena práctica: ocultar detalles de la implementación.
  - La función malloc permite pedir memoria en tiempo de ejecución, lleva un parámetro entero que es la cantidad de bytes requeridos. Que tipo de entero depende del objetivo de compilación. Para no tener que modificar los fuentes que definen o usan malloc se hace un typedef al tipo de entero que corresponda y se lo llama size\_t
- El otro uso es controvertido, ahorra escritura pero hace menos clara la lectura. Se trata simplemente de usar un typedef para abreviar.

```
typedef struct nodo *pnodo; //Con este typedef  
pnodo p; //puedo definir p así  
struct nodo *p; //en lugar de este modo
```



# Casting

- Tiene la forma (tipo-de-dato) expresión
- El valor resultante de la expresión se convierte (si es necesario) al tipo de dato entre paréntesis que antecede a la expresión
  - Ejemplo: (**double**) 'c' convierte a 99.0
- La expresión y el tipo de dato deben ser escalares.
- No se puede convertir entre tipos punteros y tipos flotantes



# Licencia

*Esta obra, © de Eduardo Zúñiga, está protegida legalmente bajo una licencia Creative Commons, **Atribución-CompartirDerivadasIgual 4.0 Internacional**.*

*<http://creativecommons.org/licenses/by-sa/4.0/>*

***Se permite: copiar, distribuir y comunicar públicamente la obra; hacer obras derivadas y hacer un uso comercial de la misma.  
Siempre que se cite al autor y se herede la licencia.***

