PREPROCESADOR

EL PREPROCESADOR

El preprocesador es por donde pasa un código fuente antes de ser compilado. En otras palabras, el da una versión final del código fuente para ser compilado.

El preprocesamiento ocurre antes de la compilación de un programa. Algunas de las acciones que puede realizar son la inclusión de otros archivos dentro del archivo a compilar, la definición de constantes simbólicas y macros, la compilación condicional del código de un programa y la ejecución condicional de las directivas del preprocesador.

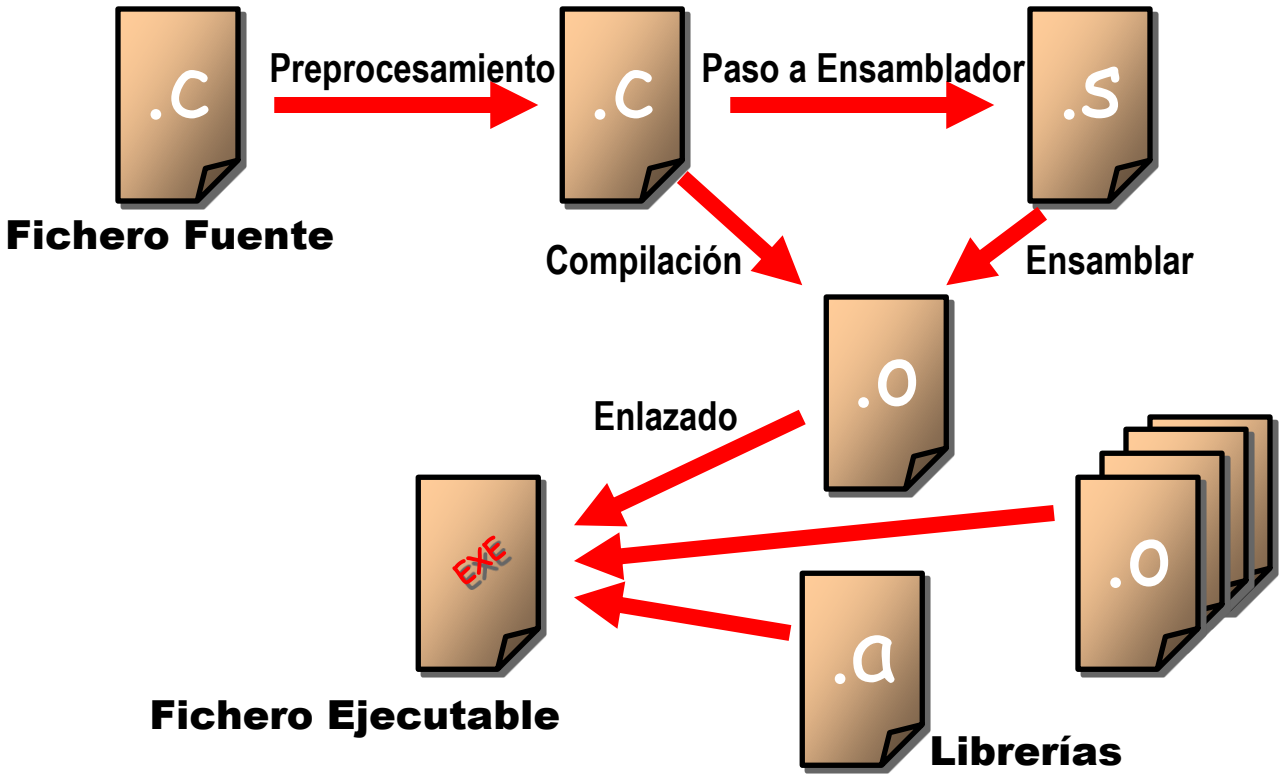
Todas las directivas del preprocesador comienzan con # y, en una misma línea, antes de una directiva solamente pueden aparecer espacios en blanco.

FASES DE COMPILACIÓN

- Preprocesamiento
  - Sustitución de constantes simbólicas
  - Expansión de macros
  - Procesamiento de directivas

  `#include`, **`#define`**, **`#if`**, **`#ifdef`**, **`#else`**, **`#elif`**, **`#endif`**, **`#ifndef`**, **`#undef`**
- Compilación
  - Análisis sintáctico
  - Análisis semántico
  - Generación de código objeto
- Enlace
  - Enlazar código objeto de librerías y módulos
  - Generar programa ejecutable.

FASES DE COMPILACIÓN



DIRECTIVAS DEL PREPROCESADOR

Directiva	Acción
#define	#define TRUE 1 //define constante TRUE #define MAX(a,b) (((a)>(b))?(a):(b)) // macro
#include	#include <stdio.h> //incluir archivo stdio.h
#if	#if !defined(TRUE) //si no definida constante TRUE
#ifdef	#ifdef SWAP //si definida constante SWAP
#endif	#endif // termina bloque de #if
#ifndef	#ifndef MSDOS //si no definida constante MSDOS

DIRECTIVAS DEL PREPROCESADOR

Directiva	Acción
#error	#error “Programa no compila bajo MSDOS”
#elif	#elif defined(SWAP) //si constante SWAP definida
#undef	#undef SWAP //indefinir constante SWAP
#else	#else //parte else de anterior #if
#line	#line 10000 "DEBUGIT.C“ //cambia núm. de línea
#pragma	#pragma message “Using code version 1.”

DIRECTIVAS DEL PREPROCESADOR

Son expandidas en la fase de preprocesado:

`#define`: Define una nueva constante o macro del preprocesador.

`#include`: Incluye el contenido de otro fichero.

`#ifdef #ifndef`: Preprocesamiento condicionado.

`#endif`: Fin de bloque condicional.

`#error`: Muestra un mensaje de error

CONSTANTES Y MACROS

La directiva #define crea constantes simbólicas (constantes representadas como símbolos) y macros (operaciones definidas como símbolos). El formato de la directiva #define es:

```
#define variable valor
```

Cuando en un archivo apare esta linea todas las subsecuentes apariciones de variable serán de forma automática reemplazadas por valor. Por ejemplo:

```
#define PI 3.14159
```

reemplaza todas las ocurrencias subsiguientes de la constante simbólica PI por la constante numérica 3.14159.

CONSTANTES Y MACROS

Las constantes simbólicas permiten al programador ponerle un nombre a una constante y utilizarlo a lo largo de todo el programa. Si durante el programa requiere de modificación, puede ser modificada una vez en la directiva #define y cuando sea el programa recompilado, todas las ocurrencias de la constante dentro del programa serán modificadas de manera automática.

Observación: todo lo que esté a la derecha del identificador reemplaza a la variable. Es también un error redefinir una constante simbólica con un nuevo valor.

```
#define PI = 3.14159
```

CONSTANTES Y MACROS

Una macro es una operación definida en una directiva de procesador #define. Como en el caso de las constantes simbólicas, antes de compilarse el programa, la macro identificación se reemplaza en el mismo texto por la función.

Las macros pueden ser definidas con o sin argumentos. Una macro sin argumentos se procesa como una constante simbólica.

En una macro con argumentos, los argumentos se incluyen en el texto de reemplazo, y a continuación la macro se expande, es decir, en el programa el texto de reemplazo reemplaza al identificador y a la lista de argumentos.

Ejemplo

```
#define macro(args,...) función
```

CONSTANTES Y MACROS

Veamos la siguiente definición de macro con un argumento, correspondiente al área de un círculo:

```
#define CIRCLE_AREA(x) PI * (x)* (x)
```

Siempre que en el archivo aparezca CIRCLE_AREA(x), el valor de x será sustituido por x en el texto de reemplazo, la constante simbólica PI será reemplazada por su valor (previamente definido), y en el programa la macro se expandirá. Por ejemplo, el enunciado

```
area = CIRCLE_AREA(4);
```

se expande a

```
area = 3.14159 * (4) * (4);
```

Dado que la expresión está sólo formada por constantes, al momento de compilarse se evalúa el valor de la expresión y se asigna a la variable área.

CONSTANTES Y MACROS

Cuando el argumento del macro es una expresión, los paréntesis alrededor de cada x en el texto de remplazo obligan al orden apropiado de evaluación. Por ejemplo el enunciado

```
area = CIRCLE_AREA(c +2);
```

se expande a

```
area = 3.14159 * (c+2) * (c+2);
```

que se evalúa de forma correcta, porque los paréntesis obligan al orden correcto de evaluación. Si se omiten los paréntesis, la expansión del macro es

```
area = 3.14159 * c + 2 * c + 2;
```

lo que se evalúa de forma incorrecta como

```
area = (3.14159 * c )+ (2 * c) + 2;
```

CONSTANTES Y MACROS

```
#define PI  3.14159
#define NUM_ELEM  5
#define AREA(rad) PI*(rad)*(rad)
#define MAX(a,b)  ((a)>(b)?(a):(b))
int main()
{
  int   i;
  float vec[NUM_ELEM];
  for(i=0;i<NUM_ELEM;i++)
      vec[i]=MAX(i*5.2,AREA(i));
}
```

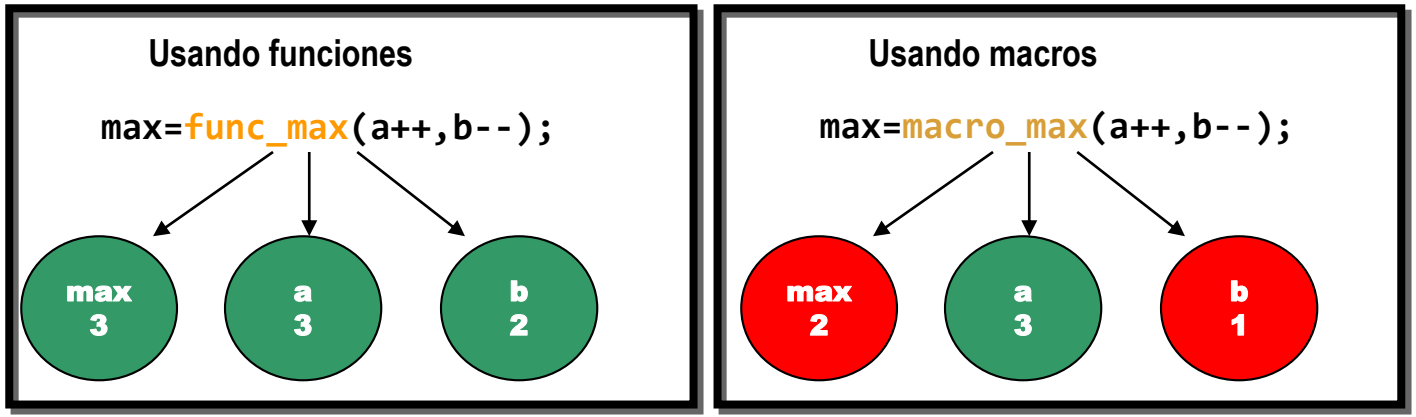
CONSTANTES Y MACROS

Tras la fase de preprocesamiento

```
int main()
{
  int   i;
  float vec[5];
  for(i=0;i<5;i++)
     vec[i]=((i*5.2)>(3.14159*i*i) ? (i*5.2) : (3.14*i*i));
}
```

MACROS vs FUNCIONES

```
int func_max(int a, int b){
  return (a>b ? a : b);
}
#define macro_max(a,b) ((a)>(b)?(a):(b))
int a=2,b=3,max;
```



#undef

Las constantes simbólicas y las macros pueden ser descartadas utilizando la directiva de preprocesador #undef.

La directiva #undef “elimina” la definición de una constante simbólica o de un nombre de macro. El alcance de una constante simbólica o de una macro cubre desde su definición hasta que, mediante #undef, queda eliminada su definición, o si no, hasta el final del archivo.

Una vez eliminada la definición, puede volverse a definir su nombre utilizando #define.

MACROS PREDEFINIDAS

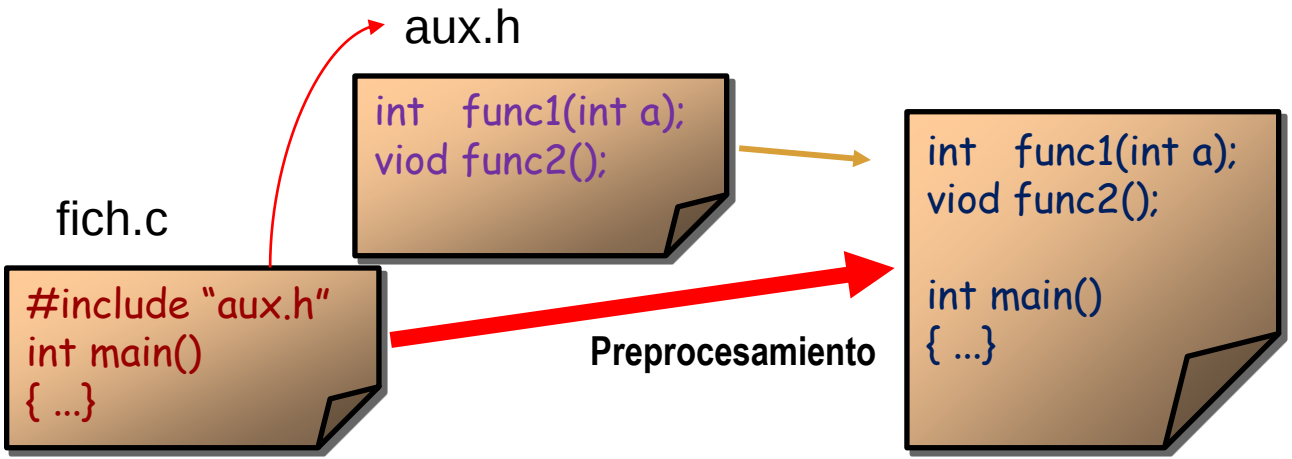
`__DATE__`	Da fecha actual en formato "MMM DD YYYY"
`__TIME__`	Da hora actual en formato "HH:MM:SS"
`__FILE__`	Da nombre de archivo actual
`__LINE__`	Da número de línea actual

INCLUSIÓN DE ARCHIVOS

Los prototipos de las funciones usadas por varios ficheros fuente se suelen definir en fichero de cabecera.

`#include <stdio.h>` Cabeceras del sistema.

`#include "mis_func.h"` Ficheros de cabecera locales.



SENTENCIAS CONDICIONALES

La compilación condicional le permite al programador controlar la ejecución de las directivas de preprocesador, y la compilación del código del programa.

Cada una de las directivas de preprocesador evalúa una expresión entera constante. Las expresiones de cambio de tipo, las expresiones sizeof, y las constantes de enumeración no pueden ser evaluadas en directivas de preprocesador.

El constructor de preprocesador condicional es muy parecido a la estructura de selección if.

SENTENCIAS CONDICIONALES

```
#ifdef variable
<bloque de sentencias>
...
#endif
#ifndef variable
<bloque de sentencias>
...
#endif
```

SENTENCIAS CONDICIONALES

Veamos el siguiente código de preprocesador:

```
#if !defined(NULL)
#define NULL 0
#endif
```

Estas directivas determinan si NULL ha sido definido. La expresión defined(NULL) se evalúa a 1, si NULL está definido y 0 de lo contrario. Si el resultado es 0, !defined(NULL) se evalúa a 1 y NULL queda definido. De lo contrario la directiva #define es pasada por alto.

SENTENCIAS CONDICIONALES

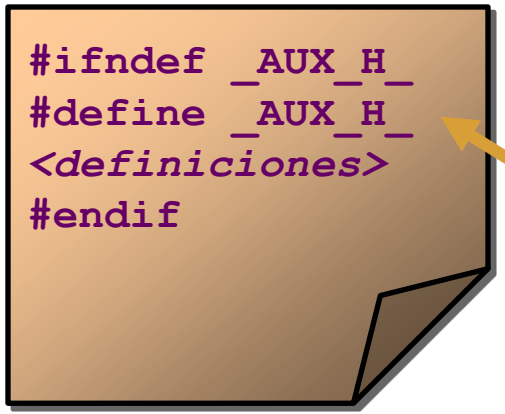
Las directivas `#ifdef` y `#ifndef` son abreviaturas correspondientes a `#ifdefined(nombre)` y `#if !defined(nombre)`.

Un constructor de preprocesador condicional de varias partes puede ser probado utilizando las directivas

`#elif` (el equivalente de `else if` en una estructura if) y

`#else` (el equivalente de `else` en una estructura if).

DEPURACIÓN



Evita la redefinicion de funciones y variables

```
#include “aux.h”
int main()
{
...
}
```

NÚMEROS DE LÍNEA

La directiva del preprocesador #line hace que las líneas de código subsecuentes se renumeren, iniciándose con el valor entero constante especificado. La directiva:

```
#line 100
```

Inicia la numeración de líneas a partir de 100 empezando con la siguiente línea del código fuente.
El la directiva #line puede incluirse un nombre de archivo. La directiva

```
#line 100 “file1.c”
```

NÚMEROS DE LÍNEA

Indica que las líneas están numeradas a partir de 100 empezando desde la siguiente línea de código fuente, y el nombre del archivo para fines de cualquier mensaje de compilador es “file1.c”. Esta directiva normalmente se utiliza para ayudar a preparar los mensajes producidos debido a errores de sintaxis y a advertencias del compilador. En el archivo fuente no aparecen los números de línea.

CONSTANTES SIMBÓLICAS PREDEFINIDAS

A continuación se puede ver un ejemplo del uso de __FILE__ y __LINE__

// depurando macros para poder ver de dónde proviene un mensaje.

```
#define CADENADONDE "[archivo %s, línea %d] "
#define ARGUMENTOSDONDE __FILE__,__LINE__
printf (CADENADONDE ": mira, x=%d\n", ARGUMENTOSDONDE, x);
```

EJERCICIOS

1. Escriba una directiva de preprocesador para que lleve a cabo cada una de las siguientes:

   a. Definir la constante simbólica YES para que tenga el valor 1

   b. Definir la constante simbólica NO para que tenga el valor 0

   c. Renumerar las líneas restantes del archivo empezando con el número de línea 3000.

   d. Si está definida la constante simbólica TRUE, elimine su definición, y vuélvala a definir como 1. No utilice #ifdef.

EJERCICIOS

e. Si está definida la constante simbólica TRUE, elimine su definición, y vuélvala a definir como 1. Utilice #ifdef.

f. Si la constante simbólica TRUE no es igual a 0, defina la constante simbólica FALSE como 0. De lo contrario defina FALSE como 1.

g. Defina la macro SQUARE_VOLUMEN que calcula el volumen de un cuadrado. La macro toma un argumento.

EJERCICIOS

2. Escriba un programa que defina una macro con un argumento para calcula el volumen de una esfera. El programa deberá calcular el volumen para esferas de radios de 1 a 10, e imprimir los resultados en formato tabular. La fórmula del volumen de una esfera es: `4/3 * PI * r3` donde `PI` es `3.14159`.

3. Escriba un programa que produzca la siguiente salida: `La suma de x e y es 13.` El programa deberá definir la macro SUM con dos argumentos, x e y, y utilizar SUM para producir la salida.

4. Escriba un programa que utilice la macro MININUM2 para determinar el más pequeño de dos valores numéricos. Introduzca los valores desde el teclado.

EJERCICIOS

5. Escriba un programa que utilice al macro MININUM3 para determinar el más pequeño de tres valores numéricos. La macro debe utilizar la macro MININUM2.

6. Escriba un programa que utilice la macro PRINT para imprimir un valor de cadena.

7. Escriba un programa que utilice la macro PRINTARRAY para imprimir un arreglo de enteros. La macro deberá recibir como argumentos el arreglo y el número de elementos en el arreglo.

8. Escriba un programa que utilice la macro SUMARRAY para sumar los valores de un arreglo numérico. La macro deberá recibir como argumentos el arreglo y el número de elementos en el arreglo.