UNIDAD 4

DIRECTIVAS DEL PREPROCESADOR DE "C/C++"

Para aumentar su capacidad el Lenguaje "C" se basa en gran medida en el **Preprocesador**.

Las líneas que se inician con **#** en la primera columna se llaman líneas de control y se comunican con el preprocesador. La sintaxis de las líneas de control es independiente del resto del lenguaje "C". El efecto de una línea de control abarca desde su lugar en el archivo hasta el final del mismo.

#include

La directiva **#include** ocasiona que el compilador incluya un archivo fuente. El archivo fuente a leer debe estar entre comillas (" ") o entre ángulos (< >).

FORMA GENERAL

```
#include  <nombre de archivo>

Ejemplo:

#include "lista.h"
#include <stdio.h>

Ejemplo:

/* Archivo "lista.h"  DEFINICION DE LA ESTRUCTURA */

typedef char DATA;

struct lista {
          DATA           d;
          struct lista   *siguiente;
          };

typedef struct lista ELEMENTO;
typedef ELEMENTO     *ENLACE;
```

Esto ocasiona que el compilador remplace las líneas anteriores con una copia del archivo referido.

Cuando el nombre del archivo se encierra entre comillas, el archivo primero se busca en el directorio actual y luego en los lugares estándar.

Cuando el archivo va entre ángulos, el preprocesador solo lo busca en los lugares estándar, por ejemplo, en el directorio include.

Dichos archivos reciben el nombre de archivos de cabecera (head) y es por ello que se usa la extensión (.h).

Si el archivo se encuentra en una trayectoria específica, se puede indicar toda la trayectoria.

#define

Definición de **macros.**

Esta directiva define un identificador y una cadena que será substituida por el identificador, cada vez que se encuentre en el archivo fuente. El estandar ANSI, hace la siguiente definición.

identificador **macronombre.**

Proceso de remplazo **macrosubstitución.**

FORMA GENERAL

```
#define  identtificador  cadena
```

Note que la cadena no termina con (;) puede haber cualquier número de espacios entre el **identificador** y la cadena. Una vez iniciada la cadena ésta debe terminar con un salto de línea. Si la cadena es muy larga y se desea pasar de línea ésto se puede hacer usando un (\) y luego se continúa en la siguiente línea.
Como práctica el **identificador** se escribe con letras MAYUSCULAS.

Ejemplo:

```
#define MENSAJE "Este es un \
                 mensaje de bienvenida"
```

Otros ejemplos:

```
#define FALSO 0
```

Cada vez que encuentra el identificador FALSO, lo substituye por 0.

Una vez definido un macronombre se puede usar para hacer otras definiciones.

```
#define UNO  1
#define DOS  UNO+UNO
#define TRES UNO+DOS
```

Macros con paréntesis.

FORMA GENERAL.

#define **identificador**(lista de argumentos) cadena

No existe espacio entre el identificador y el paréntesis de la lista de argumentos.
La lista de argumentos es opcional y consiste de una secuencia de argumentos, separados por (,).

```
#define CUBO(x) ((x)*(x)*(x))
```

al hacer la siguiente declaración y asignación, resulta:

```
int n, y;

n = CUBO(y);

Al remplazar, resulta:

n = ((y)*(y)*(y));
```

El uso de paréntesis se puede justificar si hacemos la siguiente asignación:

```
n = CUBO(l+y);

Se remplaza por:

n = ((l+y)*(l+y)*(l+y));

Otro ejemolo;

#define SUMA ((a)+(b))
```

al hacer la siguiente declaración y asignación, resulta:

```
int i, j, s;
s = SUMA(i, j);
```

Al remplazar, resulta:

```
s = ((i)+(j));
```

Algunas reglas al usar macros.

1.- **Paréntesis anidados y comas**. La lista de argumentos, puede contener paréntesis

anidados balanceados y comas.

```
#define MERR(x, str) MUESTAERR("error", x, str)

definiendo:

MERR(2, "Presione ENTER luego ESC");

Al hacerse la expanción, se muestra:

("error", 2, "Presione ENTER luego ESC");
```

2.- **Uniendo cadenas con ##**. Se pueden unir dos cadenas separándolas con **##** (espacios a ambos lados son opcionales). El preprocesador quita los espacios y ## uniendo las dos cadenas en una nueva. Esto se puede usar para construir identificadores.

```
#define VAR(i,j) (i##j)

asignando:

int x;
VAR(x, 6);
```

Al hacer la expansión resulta x6.

3.- **Convirtiendo a cadena con #**. Anteponiendo el símbolo **#** a un argumento, este es convertido a cadena.

```
#define MOSTRAR(bandera) printf(#bandera "= %d\n", bandera)

Con un fragmento de código

int valor_alto = 1024;
MOSTRAR(valor_alto);

esto se convierte en:

int valor_alto = 1024;
printf("valor_alto " "= %d\n", valor_alto);

lo que es equivalente a:

int valor_alto = 1024;
printf("valor_alto = %d\n", valor_alto);
```

4.- **Algunas confusiones**. La similitud entre una llamada a función y una llamada a **macro** ocasiona confusión.

Una llamada a **macro** no verifica el tipo, lo que puede dar lugar a errores que no ocasionan llamadas de atención (warning).

Una llamada a macro puede ocasionar efectos laterales, especialmente cuando un argumento es evaluado más de una vez.

Ejemplo:

Considere la siguiente función:

```
cubo(x)
int x;
{
     return(x*x*x);
}

#define CUBO(x) ((x)+(x)*(x))
......
int b=0, a=3;
b = cubo(a++);
Al terminar la operación b = 27 y a = 4

a = 3;
b = CUBO(a++);
Al expandir la macro resulta:
b = ((a++)*(a++)*(a++));
Al terminar la operación b = 27 y a = 6
```

#undef

Elimina una definición anterior del **macronombre** que le sigue. Su propósito es asignar los macronombres solo a aquellas secciones de código que los necesiten.

FORMA GENERAL

```
#undef macronombre

#define LON 100
#define ANCHO 200

char a[LON][ANCHO];

#undef LON

/* aquí LON ya no está definido */
```

```
#define LON 50

int b[LON][ANCHO];
```

DIRECTIVAS DE COMPILACION CONDICIONAL.

Estas directivas nos permiten compilar selectivamente parte del código fuente de un programa, lo que se conoce como compilación condicional.

Se utiliza mucho en programas comerciales para mantener compatibilidad entre versiones actuales y anteriores de los mismos.

#if, #else #endif.

FORMA GENERAL

```
     #if expresión constante
          secuencia de sentencias
     #else
          secuencia de sentencias
     #endif
```

Ejemplo copiado del archivo stdio.h.

```
#if __STDC__
#define _Cdecl
#else
#define _Cdecl  cdecl
#endif
```

NOTA: __STDC__ Es un identificador global, esta macro es definida como la constante **1** si se compila con ANSI, de otra forma es indefinido.

#elif.

Esta directiva quiere decir IF-ELSE-IF estableciendo una escala de este tipo para opciones de compilación múltiples.

FORMA GENERAL

```
     #if expresión constante1
          secuencia de sentencias
     #elif expresión constante2
     .
     .
     .
```

```
#elif expresión constanteN
     secuencia de sentencias
#endif
```

Por ejemplo el siguiente fragmento de código define la moneda usada de acuerdo al país.

```
#define USA 1
#define INGLATERRA 2
#define FRANCIA 3


#define PAIS_ACTIVO USA



#if PAIS_ACTIVO == USA
     char moneda[] = "dolar";
#elif PAIS_ACTIVO == INGLATERRA
     char moneda[] = "libra";
#else
     char moneda[] = "franco";
#endif
```

#ifdef, #ifndef.

Se pueden usar estas directivas de compilación, las cuales significan:

```
#ifdef          si definido.
#ifndef         si no definido.
```

Junto con estas directivas se puede usar **#else** y **#enfif** pero NO **#elif**.

FORMA GENERAL para **#ifdef** Si se ha definido el macronombre previamente, se compila secuencia de sentencias1.

```
#ifdef macronombre
     secuencia de sentencias1
#else
     secuencia de sentencias2
#endif
```

FORMA GENERAL para **#ifndef** Si no se ha definido el macronombre previamente, se compila secuencia de sentencias1.

```
#ifndef macronombre
     secuencia de sentencias1
```

```
#else
     secuencia de sentencias2
#endif
```

NOTA: El **#else** puede omitirse.

Ejemplo:

```
#define PRIMERO

#ifdef PRIMERO
     f(a, b)   /* Primera versión */
     {
     .
     .
     .
     }

#else
     f(a, b)  /* segunda versión */
     {
     .
     .
     .
     }
#endif
```

#error

Forza al compilador a parar la compilación puede incluir un mensaje.

FORMA GENERAL

#error mensaje de error

Normalmente esta directiva se usa con una directiva condicional que detecta alguna condición no deseada de compilación.

Ejemplo:
Supongamos que definimos:

```
#define FALSO 3
```

el cual deberá tener un valor de CERO. Entonces, podemos escribir:

```
#if FALSO != 0
#error FALSO deberá ser definido como 0
#endif
```

#line

Esta directiva nos permite dar números a las líneas de un programa para referencia y reporte de errores

FORMA GENERAL

```
#line número "nombre de archivo"
```

```
Donde:

número:              Es cualquier número positivo.

nombre de archivo:   Nombre de archivo fuente.
```

Si una vez definido un archivo, en las subsecuentes declaraciones de #line se puede omitir el nombre del archivo.
#line cambia el contenido de __LINE__ y __FILE__ los que son identificadores globales predefinidos.

Donde:

__LINE__ Es un identificador global, llamado también constante manifiesta. Este macro proporciona el número de la línea corriente del archivo fuente actual que se esta procesando como una constante decimal. La primer línea del archivo fuente es la **1**. Mediante **#line** se puede cambiar.

__FILE__ Este macro nos da el nombre del archivo fuente que se está procesando, lo da en forma de cadena.

Consideremos el siguiente ejemplo:

```
/* Archivo de cabecera prog10.h */

#define PRINCIPIO  0

#if PRINCIPIO == 0
#define INI  3
#else
#define INI  5
#endif
```

El archivo fuente resulta:

```
/* Programa 70 uso de #line */
#include "prog10.h"
#line 4 "prog10.c"

main()
{
     int a, b = INI;

     printf("\n");
     printf("Línea %d de %s", __LINE__, __FILE__);

     printf("\n");
     printf("Línea %d de %s", __LINE__, __FILE__);

     printf("\n");
     printf("Línea %d de %s", __LINE__, __FILE__);
     printf("\na = b+b*b = %d", b+b*b);
}
```

Al usar el programa CPP.EXE el cual crea un archivo de documentación en línea, con la extensión (.I) éste nos numera las líneas del programa de acuerdo a lo indicado por **#line**, resulta:

```
prog10.c 1:
prog10.c 2:
prog10.h 1:
prog10.h 2:
prog10.h 3:
prog10.h 4:
prog10.h 5:
prog10.h 6:
prog10.h 7:
prog10.h 8:
prog10.h 9:
prog10.h 10:
prog10.c 3:
prog10.c 4:
prog10.c 5: main()
prog10.c 6: {
prog10.c 7: int a, b = 3 ;
prog10.c 8:
prog10.c 9: printf("\n");
prog10.c 10: printf("Línea %d de %s", 10, "prog10.c");
```

```
prog10.c 11:
prog10.c 12: printf("\n");
prog10.c 13: printf("Línea %d de %s", 13, "prog10.c");
prog10.c 14:
prog10.c 15: printf("\n");
prog10.c 16: printf("Línea %d de %s", 16, "prog10.c");
prog10.c 17: printf("\na = b+b*b = %d", b+b*b);
prog10.c 18: }
prog10.c 19:
```

Al correr el programa, se obtiene:

```
Línea 10 de PROG10.C
Línea 13 de PROG10.C
Línea 16 de PROG10.C
a = b+b*b = 12
```

#pragma

Esta directiva es definida de acuerdo a la implementación del paquete, nos permite darle varias instrucciones al compilador.

FORMA GENERAL

#pragma nombre

Donde **nombre** es el de la sentencia que se quiera.

TurboC define 2 sentencias:

* **warn**
* **inline**

* **warn** Permite que TurboC sobreescriba opciones de mensajes de advertencia (warnings).

FORMATO GENERAL

#pragma warn especificaciones

Donde:

especificaciones. Es una de las varias opciones de mensajes de error.

+xxx Activa los mensajes de warnings
-yyy Desactiva los mensajes de warnings.
,zzz Restaura el valor de los warnings.

Esta opción se usa con el TCC.EXE.

* **inline**. Le dice al TurboC que el programa contiene código ensambaldor en línea, es equivalente a la opción en línea **-B**

FORMA GENERAL

```
#pragma inline
```

Se coloca al principio del archivo para que el compilador arranque de nuevo con la opción **-B**. Si no se usa esta opción el compilador arranca de nuevo al encontrar el comando **asm** el objeto de la directiva o la opción **-B** es ahorrar tiempo de compilación.

MACROS PREDEFINIDOS.

```
__LINE__
__FILE__
__DATE__
__TIME__
__STDC__
```

* La macro __DATE__ contiene una cadena de la forma mes/día/año, que es la fecha de la traducción del archivo fuente a código objeto.
* La macro __TIME__ contiene una cadena de la forma hora:minuto:segundo, la hora de la traducción del archivo fuente a código objeto.

Están definidas otras macro de las cuales el compilador solo define una, basandose en el modelo de memoria que se usa durante la compilación.

```
__TYNY__
__SMALL__
__COMPACT__
__MEDIUM__
__LARGE__
__HUGE__
```

Otras macro definidas son:

__CDECL__ Se define si se usa la conversión de llamada estándar, esto es, si no se usa la macro __PASCAL__, la que esté en uso se define como **1**, la otra queda indefinida.

__MSDOS__ Se define con el valor **1** en todas las situaciones cuando se usa la versión de TurboC para MS-DOS.

__TURBOC__ Contiene el número de la versión de TurboC en hexadecimal.

Veamos un ejemplo de uso de estas **macros** predefinidas.

```
/* Archivo de cabecera "prog11.h */

#if __TINY__
#define MODULO ("TINY")
#elif __SMALL__
#define MODULO ("SMALL")
#elif __COMPACT__
#define MODULO ("COMPACT")
#elif __MEDIUM__
#define MODULO ("MEDIUM")
#elif __LARGE__
#define MODULO ("LARGE")
#elif __HUGE__
#define MODULO ("HUGE")
#else
#define MODULO ("No definido")
#endif
```

El programa fuente nos resulta:

```
/* Programa 11 uso de macros predefinidas */
#include "prog11.h"
#line 4 "prog11.c"

main()
{
     printf("\n");
     printf("Línea %d de %s", __LINE__, __FILE__);

     printf("\n");
     printf("Línea %d de %s", __LINE__, __FILE__);

     printf("\n");
     printf("Línea %d de %s", __LINE__, __FILE__);
     printf("\nLa versión de TurboC es 0x%x", __TURBOC__);
     printf("\nEl modulo de memoria usado es %s", MODULO);
     printf("\nEl día de compilacion es %s", __DATE__);
     printf("\nLa hora de compilación es %s", __TIME__);
}
```

Al correr el programa PROG11.EXE nos da:

```
Línea 8 de PROG11.C
Línea 11 de PROG11.C
Línea 14 de PROG11.C
La versión de TurboC es 0x18d
El modulo de memoria usado es SMALL
El día de compilacion es Dec 08 1990
La hora de compilación es 18:08:46
```