

Diseño de Sistemas Microprocesados

Fundamentos del lenguaje ANSI C

Profesor:

Juan Vega Martinez



Ventajas del uso del lenguaje ANSI C en sistemas embebidos

- El lenguaje C es universal y por tanto es fácil portar un programa hecho para un microcontrolador a otro modelo de microcontrolador.
- El lenguaje C facilita el trabajo en equipo.
- El lenguaje C permite la modularización o creación de librerías específicas para el manejo de periféricos conocidos como el LCD, puerto RS232, Display 7 segmentos, etc.
- El lenguaje C permite el encapsulado (programas con entradas o argumentos y salidas específicas o retornos).
- El lenguaje C es una herramienta rápida de programación.
- El lenguaje C permite la incorporación de bloques de ensamblador.
- El compilador utiliza todo el set de instrucciones, por lo tanto convierte el código en C a un programa eficiente en lenguaje ensamblador.



Desventajas del uso del lenguaje C en sistemas embebidos

- El lenguaje C ocupa mayor espacio en la memoria FLASH y requiere un mayor consumo de memoria RAM.
- Los buenos compiladores tienen un precio, los ensambladores son gratuitos.
- Los compiladores son programas que pueden tener algunos inconvenientes.
- Los compiladores generan código redundante e innecesario que muchas veces no es necesario ejecutar.
- El lenguaje C no requiere que el programador tenga un conocimiento profundo de la arquitectura del microcontrolador.



Estructura de un programa en C

```
1
2  /** ***** **/
3  //      Curso de Diseño de Sistemas Microprocesados      //
4  //      Programa ejemplo en lenguaje C                      //
5  //                                                         //
6  // File : main.c                                           //
7  /** ***** **/
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```



Estructura de un programa en C

```
1
2  /*******//
3  //      Curso de Diseño de Sistemas Microprocesados      //
4  //      Programa ejemplo en lenguaje C                      //
5  //                                                          //
6  // File : main.c                                           //
7  /*******//
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```

#include

Es una directiva del preprocesador que incluye el contenido del archivo "stdio.h" al proyecto.



Estructura de un programa en C

```
1
2  //*****//
3  //    Curso de Diseño de Sistemas Microprocesados    //
4  //          Programa ejemplo en lenguaje C          //
5  //                                                    //
6  // File : main.c                                     //
7  //*****//
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```

#define

Es una directiva del procesador, que es creado en esta línea para sustituir un texto por un valor de 3.1415



UNIVERSIDAD
TECNOLÓGICA
DEL PERÚ

Directivas de Preprocesamiento

MPLAB XC16 C compiler user guide

Directive	Meaning	Example
#define	Define preprocessor macro	#define SIZE 5 #define FLAG #define add(a,b) ((a)+(b))
#elif	Short for #else #if	see #ifdef
#else	Conditionally include source lines	see #if
#endif	Terminate conditional source inclusion	see #if
#error	Generate an error message	#error Size too big
#if	Include source lines if constant expression true	#if SIZE < 10 c = process(10) #else skip(); #endif
#ifdef	Include source lines if preprocessor symbol defined	#ifdef FLAG do_loop(); #elif SIZE == 5 skip_loop(); #endif
#ifndef	Include source lines if preprocessor symbol not defined	#ifndef FLAG jump(); #endif
#include	Include text file into source	#include <stdio.h> #include "project.h"
#line	Specify line number and file name for listing	#line 3 final
#pragma	Compiler specific options	Refer to Section 19.5 "Pragmas vs. Attributes"
#undef	Undefines preprocessor symbol	#undef FLAG
#warning	Generate a warning message	#warning Length not set



Estructura de un programa en C

```
1
2  /*******//
3  //      Curso de Diseño de Sistemas Microprocesados      //
4  //      Programa ejemplo en lenguaje C                      //
5  //                                                         //
6  // File : main.c                                           //
7  /*******//
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```

Función

La función: `int main (void)` , es la función principal del proyecto.



Estructura de un programa en C

```
1
2  //*****//
3  //    Curso de Diseño de Sistemas Microprocesados    //
4  //          Programa ejemplo en lenguaje C          //
5  //                                                    //
6  // File : main.c                                     //
7  //*****//
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```

Declaración de variables

En el programa existen 2 variables de punto flotante que han sido declaradas.



Estructura de un programa en C

```
1
2  /*******//
3  //      Curso de Diseño de Sistemas Microprocesados      //
4  //      Programa ejemplo en lenguaje C                      //
5  //                                                          //
6  // File : main.c                                           //
7  /*******//
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```

Comentarios

No ocupan memoria, ni se construyen.



Estructura de un programa en C

```
1
2  //*****//
3  //    Curso de Diseño de Sistemas Microprocesados    //
4  //          Programa ejemplo en lenguaje C              //
5  //                                                    //
6  // File : main.c                                     //
7  //*****//
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```

Esta es una línea de asignación de valores.



Estructura de un programa en C

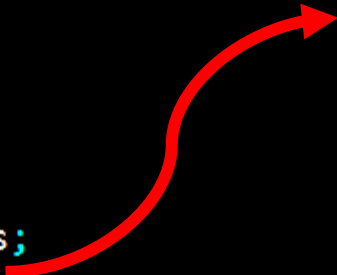
```
1
2  /*******//
3  //    Curso de Diseño de Sistemas Microprocesados    //
4  //          Programa ejemplo en lenguaje C          //
5  //                                                    //
6  // File : main.c                                     //
7  /*******//
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```

Esta también es una línea de
asignación de valores.



Estructura de un programa en C

```
1
2  //*****//
3  //    Curso de Diseño de Sistemas Microprocesados    //
4  //          Programa ejemplo en lenguaje C              //
5  //                                                    //
6  // File : main.c                                     //
7  //*****//
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```



Esta línea llama a la function printf e
imprime el valor resultante en
pantalla.



Comentarios en lenguaje C

Se utilizan para documentar la funcionalidad de un programa de y para explicar lo que hace un determinado bloque o línea de código.

Los comentarios son ignorados por el compilador, por lo que puede escribir todo lo que quieras en ellos

Soporta 2 tipos de comentarios:

- 1) Comentarios por línea (**// Texto**)
- 2) Comentarios por bloque (**/* Texto */**)



Comentarios en lenguaje C

1) Comentarios por línea

```
1  //=====
2  // Program:   Hello.c
3  // Author:    R. Ostapiuk
4  //=====
5  #include <stdio.h>
6
7  // Application begins here
8  int main(void)
9  {
10     printf("Hello, world!\n"); // Display "Hello, world!" in terminal
11 }
```




Comentarios en lenguaje C

1) Comentarios por línea

```
1  //=====
2  // Program:   Hello.c
3  // Author:    R. Ostapiuk
4  //=====
5  #include <stdio.h>
6
7  // Application begins here
8  int main(void)
9  {
10     printf("Hello, world!\n"); // Display "Hello, world!" in terminal
11 }
```



Comentarios en lenguaje C

2) Comentarios por bloque

```
1  /*****
2  * Program:   Hello.c
3  * Author:    R. Ostapiuk
4  *****/
5  #include <stdio.h>
6
7  /*****
8  * Function:  main()
9  * (some details about what main does here...)
10 *****/
11 int main(void)
12 {
13     int i;                // Loop Count Variable
14     char *p;              // Pointer to output string
15     /*
16     printf("Hello, world!\n");    // Display "Hello, world!" in terminal
17     */
18     ...
19 }
```



Comentarios en lenguaje C

2) Comentarios por bloque

```
1  /******  
2  * Program:   Hello.c  
3  * Author:    R. Ostapiuk  
4  *****/  
5  #include <stdio.h>  
6  
7  /*****  
8  * Function:  main()  
9  * (some details about what main does here...)  
10 *****/  
11 int main(void)  
12 {  
13     int i;                // Loop Count Variable  
14     char *p;              // Pointer to output string  
15     /*  
16     printf("Hello, world!\n");    // Display "Hello, world!" in terminal  
17     */  
18     ...  
19 }
```



Comentarios en lenguaje C

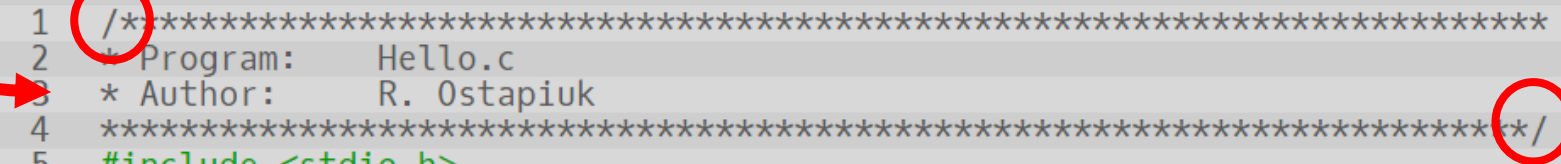
2) Comentarios por bloque

```
1  /******  
2  * Program:   Hello.c  
3  * Author:    R. Ostapiuk  
4  *****/  
5  #include <stdio.h>  
6  
7  /*****  
8  * Function:  main()  
9  * (some details about what main does here...)  
10 *****/  
11 int main(void)  
12 {  
13     int i;                // Loop Count Variable  
14     char *p;              // Pointer to output string  
15     /*  
16     printf("Hello, world!\n");    // Display "Hello, world!" in terminal  
17     */  
18     ...  
19 }
```



Comentarios en lenguaje C

2) Comentarios por bloque



```
1  /******  
2  * Program:   Hello.c  
3  * Author:    R. Ostapiuk  
4  *****/  
5  #include <stdio.h>  
6  
7  /******  
8  * Function:  main()  
9  * (some details about what main does here...)  
10 *****/  
11 int main(void)  
12 {  
13     int i;                // Loop Count Variable  
14     char *p;              // Pointer to output string  
15     /*  
16     printf("Hello, world!\n");    // Display "Hello, world!" in terminal  
17     */  
18     ...  
19 }
```



Comentarios en lenguaje C

2) Comentarios por bloque

```
1  /******  
2  * Program:   Hello.c  
3  * Author:    R. Ostapiuk  
4  *****/  
5  #include <stdio.h>  
6  
7  /******  
8  * Function:  main()  
9  * (some details about what main does here...)  
10 *****/  
11 int main(void)  
12 {  
13     int i;                // Loop Count Variable  
14     char *p;              // Pointer to output string  
15     /*  
16     printf("Hello, world!\n");    // Display "Hello, world!" in terminal  
17     */  
18     ...  
19 }
```



Comentarios en lenguaje C

2) Comentarios por bloque

```
1  /******  
2  * Program:   Hello.c  
3  * Author:    R. Ostapiuk  
4  *****/  
5  #include <stdio.h>  
6  
7  /******  
8  * Function:  main()  
9  * (some details about what main does here...)  
10 *****/  
11 int main(void)  
12 {  
13     int i;                // Loop Count Variable  
14     char *p;              // Pointer to output string  
15     /*  
16     printf("Hello, world!\n");    // Display "Hello, world!" in terminal  
17     */  
18     ...  
19 }
```




Comentarios en lenguaje C

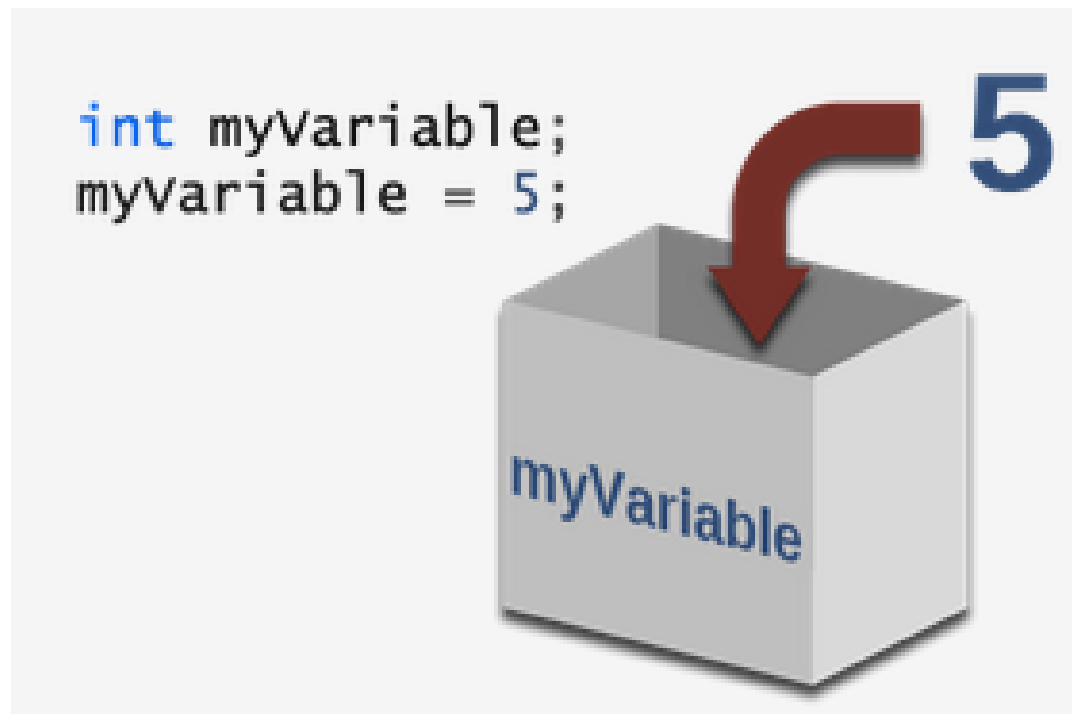
2) Comentarios por bloque

```
1  /******  
2  * Program:   Hello.c  
3  * Author:    R. Ostapiuk  
4  *****/  
5  #include <stdio.h>  
6  
7  /******  
8  * Function:  main()  
9  * (some details about what main does here...)  
10 *****/  
11 int main(void)  
12 {  
13     int i;           // Loop Count Variable  
14     char *p;         // Pointer to output string  
15     /*  
16     printf("Hello, world!\n");    // Display "Hello, world!" in terminal  
17     */  
18     ...  
19 }
```



Variables

Una variable representa una combinación de un tipo de datos y un identificador (Nombre), esto representa una o mas direcciones de memoria usados en un programa.

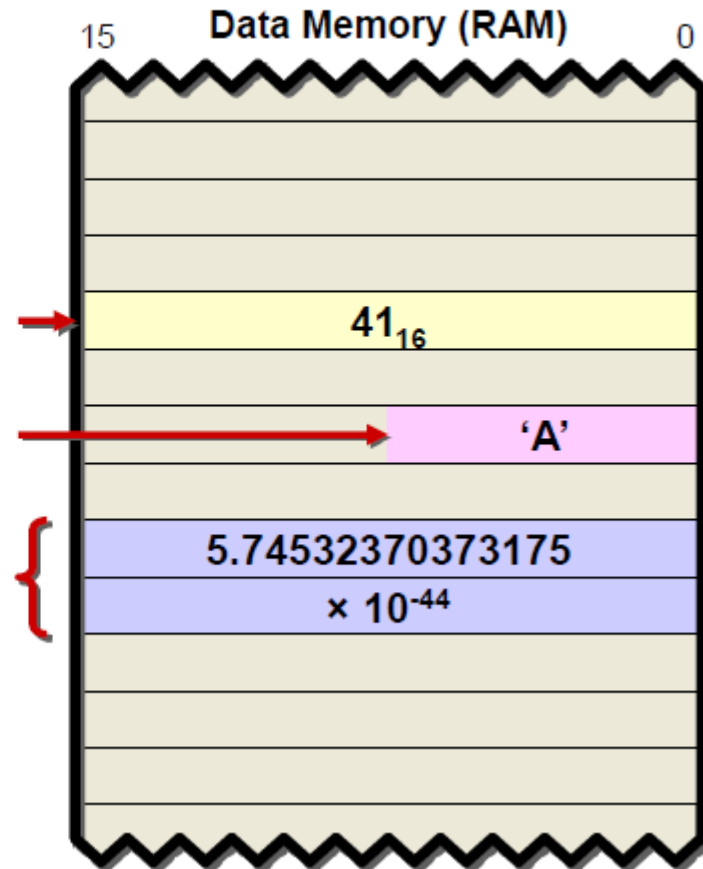




Variables

- Variable declarations consist of a unique identifier (name)...

```
int warp_factor;  
char first_letter;  
float length;
```





Variables

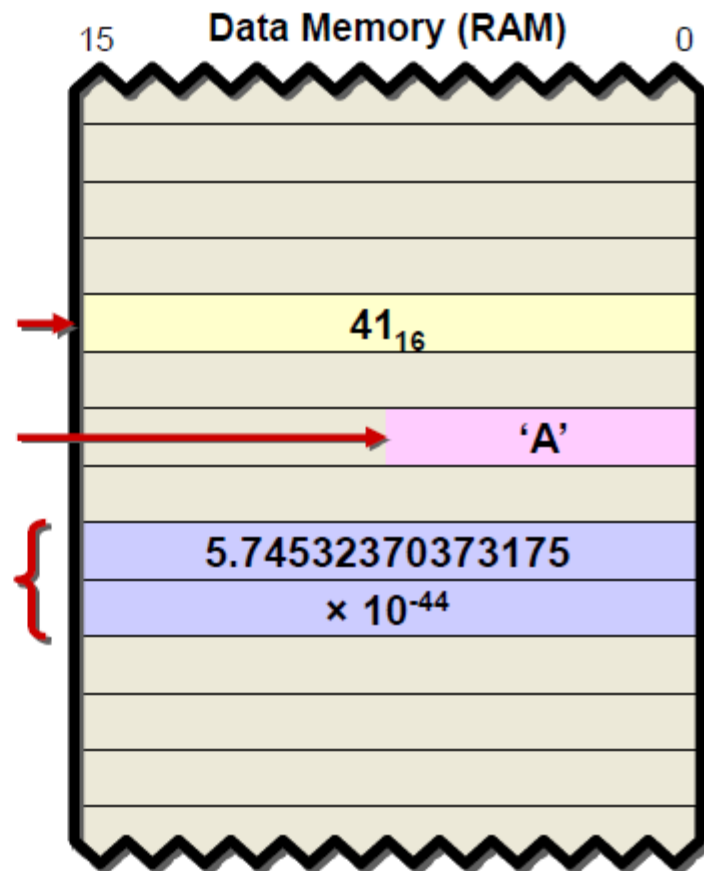
■ ...and a data type

- Determines size
- Determines how values are interpreted

`int` warp_factor;

`char` first_letter;

`float` length;





Variables

Example

```
#include <stdio.h>

#define PI 3.14159

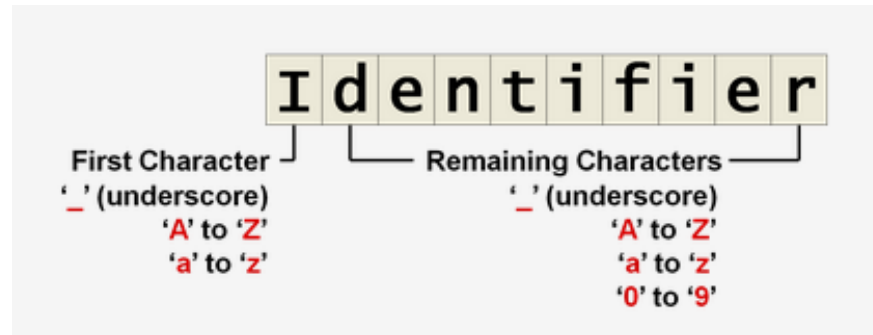
Data Types → int main(void)
{
    → float radius, area; ← Variable Declarations

    //calculate area of circle
    radius = 12.0;
    area = PI * radius * radius; ← Variables in use
    printf("Area = %f", area); ← in use
}
```



Identificador

Un identificador es un nombre dado a un elemento de programa como una [variable](#), [funcion](#), ó arreglo([array](#)). Este nombre puede ser usado para referirse al elemento del programa sin conocimiento de su especifica direccion de memoria.



En cuanto a los identificadores, es sensible en cuanto a mayúsculas y minúsculas y jamás pueden comenzar con un dígito o llevar espacios en blanco. Ejm:

Int Radio \neq Int RadiO \neq Int RADIO \neq Int radio \neq Int RaDio



Keywords

Son palabras reservadas por el lenguaje C los cuales tienen un significado especial, no deben ser usados como nombre de identificadores

ANSI C Keywords

auto
break
case
char
const
continue
default
do

double
else
enum
extern
float
for
goto
if

int
long
register
return
short
signed
sizeof
static

struct
switch
typedef
union
unsigned
void
volatile
while



Tipos de datos calificadores

Type	Description	Bits
<code>char</code>	single character	8
<code>int</code>	integer	16
<code>float</code>	single precision floating point number	32
<code>double</code>	double precision floating point number	64

The size of an `int` varies from compiler to compiler.

- MPLAB-XC16 `int` is 16-bits
- MPLAB-XC8 `int` is 16-bits
- MPLAB XC32 `int` is 16-bits
- CCS PCB, PCM & PCH `int` is 8-bits



Tipos de datos calificadores

Calificadores : unsigned, signed, short and long

Qualified Type	Min	Max	Size (Bits)
unsigned char	0	255	8
char, signed char	-128	127	8
unsigned short <i>int</i>	0	65535	16
short <i>int</i> , signed short <i>int</i>	-32768	32767	16
unsigned <i>int</i>	0	65535	16
<i>int</i> , signed <i>int</i>	-32768	32767	16
unsigned long <i>int</i>	0	$2^{32}-1$	32
long <i>int</i> , signed long <i>int</i>	-2^{31}	$2^{31}-1$	32
unsigned long long <i>int</i>	0	$2^{64}-1$	64
long long <i>int</i> , signed long long <i>int</i>	-2^{63}	$2^{63}-1$	64

La palabra ***int*** es opcional cuando esta se modifica (mostrado en cursive)

unsigned int = unsigned



Tipos de datos calificadores

Qualified Type	Absolute Min	Absolute Max	Bits
<code>float</code>	$\pm \sim 10^{-44.85}$	$\pm \sim 10^{38.53}$	32
<code>double</code> ⁽¹⁾	$\pm \sim 10^{-44.85}$	$\pm \sim 10^{38.53}$	32
<code>long double</code>	$\pm \sim 10^{-323.3}$	$\pm \sim 10^{308.3}$	64

XC16: ⁽¹⁾`double` is equivalent to `long double` if –
`fno-short-double` is used

XC16 Uses the IEEE-754 Floating Point Format



Declaración de las variables

Syntax

One declaration on a line

```
type identifier;
```

One declaration on a line with an initial value

```
type identifier = InitialValue;
```

Multiple declarations of the same type on a line

```
type identifier1, identifier2, identifier3;
```

Multiple declarations of the same type on a line with initial values

```
type identifier1 = Value1, identifier2 = Value2;
```



Declaración de las variables

```
unsigned int x;  
unsigned y = 12;  
int a, b, c;  
long int myVar = 0x12345678;  
long z;  
char first = 'a', second, third = 'c';  
float big_number = 6.02e+23;
```

Tipo de dato

identificador

Valor inicial



Declaración de variables


Las variables por defecto se almacenan en la memoria SRAM. Por esta razón, es recomendable no crear más variables de las que realmente se necesitan para no malgastar los recursos de memoria.

```
unsigned char a = 5;           //Variable global (puede ser usada en todo el programa
float pi = 3.1415;           //por las diferentes funciones.
void main( )
{
    char b = -8;              //Variable local (solo puede ser usada en el ámbito de la
                              // función main())
    unsigned int k = 12000;

}
void sumar( )
{
    unsigned char var;
    var = a +10;              //Correcto. a es una variable global.
    var = b +90;              //Error b no existe para esta función
}
```



Declaración de variables

 **main.h**

```
unsigned int a;  
unsigned int b;  
unsigned int c;
```

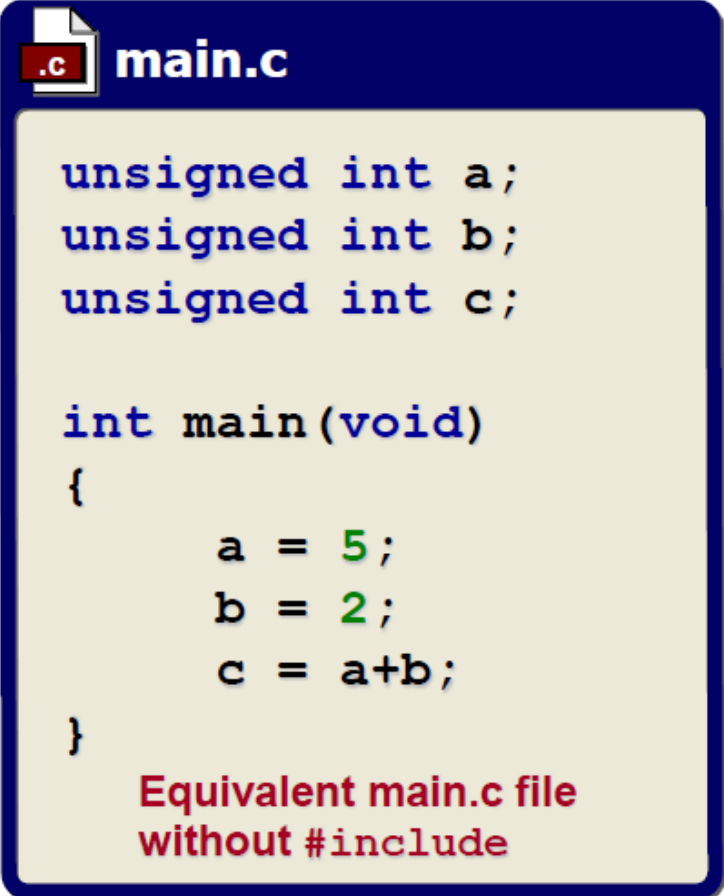
The contents of main.h
are *effectively* pasted into
main.c starting at the
#include directive's line

 **main.c**



Declaración de variables

Después de que el preprocesador corre, esta es la forma en la compilador vería el archivo main.c. El contenido de la archivo de cabecera no son en realidad copiados en el archivo fuente principal, pero se comportará como si fueron copiados.



```
main.c

unsigned int a;
unsigned int b;
unsigned int c;

int main(void)
{
    a = 5;
    b = 2;
    c = a+b;
}
```

**Equivalent main.c file
without #include**



Constantes

Muchas veces, es necesario crear variables cuyo contenido se mantenga estático. Por ejemplo, puede recordar las tablas en lenguaje ensamblador en donde se fijaba un mensaje que se podía presentar en la pantalla LCD o en los Displays 7 segmentos. Dicho mensaje se almacenaba en la memoria FLASH.

Las variables constantes se almacenan en FLASH y por tanto su contenido no se puede actualizar. Para declarar una variable constante se hace uso del operador `const`.

```
const unsigned char data = 10;
const temp = 36.5;
const char curso[ ] = {'u' , 'P' , 'R' , 'O' , 'C' , 'E' , 'S' , 'A' , 'D' , 'O' , 'S'};
void main ()
{
    TRISB = 0x00;
    TRISD = 0x00
    PORTB = data;
    PORTD = curso[4];
}
```



El comando del preprocesador **#define**

El comando del pre procesador **#define** es una macro que define un valor constante que puede ser reemplazado por un “Alias” en todo el programa. El abuso de este tipo de comando no genera ningún gasto en memoria del sistema embebido.

La ventaja del uso del **#define** es que permite mejorar la legibilidad del programa.

Ejemplo:

```
#define ES_PAR      0  
#define ES_IMPAR   1
```



Operadores aritméticos en C

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Residuo
++	Autoincremento
--	Autodecremento



Ejemplo de operaciones aritméticas

```
char a=8, b=-3;
char c;
void main( )
{
    c = a + b;           //c = 5
    c = a - b;           // c = 11
    c = a * b;           // c = -24
    c = a / b;           //c = -2
    b = 3;
    c = a / b;           // c = 2
    c = a % b;           // c = 2
    c++;                 // c = 3
    c = a++;             // c = 8, a = 9
    c = ++a;             // c = 10, a = 10
    c = a--;             // c = 10, a = 9
    c = --a;             // c = 8, a = 8
}
```



UNIVERSIDAD
TECNOLÓGICA
DEL PERÚ

DEV C++

The screenshot shows the Dev-C++ 5.7.1 IDE interface. The title bar reads "PrimerPrograma - [PrimerPrograma.dev] - Dev-C++ 5.7.1". The menu bar includes File, Edit, Search, View, Project, Execute, Tools, CVS, Window, and Help. The toolbar contains various icons for file operations, compilation, and debugging. The compiler is set to "TDM-GCC 4.8.1 64-bit Release". The left sidebar shows the project structure with "PrimerPrograma" and "main.c". The main editor window displays the following C code:

```
1
2 //*****//
3 //  Curso de Diseño de Sistemas Microprocesados  //
4 //      Programa ejemplo en lenguaje C              //
5 //                                                    //
6 // File : main.c                                    //
7 //*****//
8 #include <stdio.h>
9 #include <stdlib.h>
10
11 #define PI 3.14159
12
13 int main(void)
14 {
15     float radius, area;
16
17     //Calculate area of circle
18     radius = 12.0;
19     area = PI * radius * radius;
20     printf("Area = %f", area);
21 }
22
```

The status bar at the bottom shows "Line: 26 Col: 1 Sel: 0 Lines: 39 Length: 820 Insert Modified". The command line at the bottom reads: "Command: mingw32-make.exe -f "C:\Users\Juan\Documents\JUAN\UTP CLASES\DISEÑODESISTEMAS MICROPROCESADOS\DEV\PrimerPrograma\Makefile.win" all".



Operadores de campo de bit

Operador	Operación
&	AND
	OR
^	XOR
~	NOT
<<	Corrimiento izquierda
>>	Corrimiento derecha



Ejemplo de operaciones binarias

```
char a=0b1111 0000, b=0b00110011;
char c;
void main( )
{
    c = a & b;           // c = 00110000 (AND)
    c = a | b;           // c = 11110011 (OR)
    c = a ^ b;           // c = 11000011 (XOR)
    c = ~ b;             // c = 11001100 (NOT)
    a = 0b10101100;
    c = a << 2;           // c = 1011 0000
    a = 0b10101100;
    c = a >> 3;           // c = 0001 0101
}
```




Operadores lógicos y relacionales

Operador	Operación
==	Igual a
!=	Diferente de
>	Mayor que
<	Menor que
>=	Mayor o igual a
<=	Menor o igual a
&&	Y
	O
!	Negación



Ejemplo de operaciones lógicas

```
char a= 15, b=15;
char c;
void main(void)
{
    c = a == b;           // c = 1 (Verdadero)
    c = a != b;           // c = 0 (Falso)
    c = a > b;             // c = 0 (Falso)
    c = a < b;             // c = 0 (Falso)
    c = a >= b;            // c = 1 (Verdadero)
    c = a <= b;            // c = 1 (Verdadero)
    c = a && b;             // c = 1 (Verdadero)
    c = a || b;            // c = 1 (Verdadero)
    c = !a;                // c = 0
    a = 0;
    c = !a;                // c = 1
}
```



Sentencias de selección if

if (Expresión)

Instrucción 1;

else

Instrucción 2;

Condición verdadera porque el argumento es mayor a 0.

Caso contrario la condición es falsa porque el argumento es 0.

if (Expresión)

{

Instrucción 1;

Instrucción 2;

}

else

{

Instrucción 3;

Instrucción 4;

}

Se utiliza la apertura y cierre de la llave cada vez que hay más de una sentencia dentro del condicional



Ejemplos de condicionales compuestos

```
if ( (a>b) && (a >c) )      //Si a es mayor que b y a es mayor que c  
{  
  
}
```

```
if ( (a != b) || (a !=c) )  //Si a es diferente de b o a es diferente de c  
{  
  
}
```

```
if ( a == b)                //Si a es igual a b  
{  
  
}
```



Errores comunes con la sentencia if

```
unsigned char a = 5, b = 7, c;
```

```
void main()
```

```
{
```

```
    if ( a = b)
```

```
    {
```

```
        c = 10;
```

```
    }
```

```
    while(1);
```

```
}
```

Error $a = b$, es decir $a = 7$ y como es una asignación válida el argumento de if es 1 y por tanto la condición es considerada verdadera por lo cual $c = 10$.

```
unsigned char a = 5, b=7,c;
```

```
void main()
```

```
{
```

```
    if( a > 4 & b > 6)
```

```
    {
```

```
        c = 10;
```

```
    }
```

```
    while(1);
```

```
}
```

No afecta el resultado ya que $a > 4 = 1$ y $b > 6 = 1$. La operación AND entre ambos valores da 1 y por tanto la condición es considerada verdadera y $c = 10$. Sin embargo, es importante tener en cuenta que se está realizando la operación AND y no la operación lógica.



Sentencias de selección switch()

Para resolver el problema de ineficiencia debido a múltiples condiciones posibles para una variable se puede hacer uso de la sentencia switch() que únicamente evalúa la opción correcta y ejecuta las instrucciones especificadas para dicha condición. La sentencia tiene la siguiente estructura:

```
switch (value)
{
    case condición 1:
        Instrucciones;
        break;
    case condición 2:
        Instrucciones;
        break;
    .....
    .....
    case condición N:
        Instrucciones;
        break;
    default:
        Instrucciones;
}
```



Ejemplo de uso del switch()

```
unsigned char resul;  
void main()  
{  
    switch(a)  
    {  
        case 0:  
            resul = 10;  
            break;  
        case 1:  
            resul = 11;  
            break;  
        .....  
        .....  
        case 200:  
            resul = 245;  
            break;  
        default:  
            resul = 0;  
    }  
}
```



Sentencias iterativas: bucle for

for (*expresión inicial; condición de la expresión, incremento de la expresión*)

```
{  
    Instrucciones  
}
```

Ejemplos:

```
unsigned char i;  
for (i = 0; i<10; i++)  
{  
    k = i;  
}  
//k = 9, i = 10
```

```
unsigned char i;  
for (i = 0; i<=10; i++)  
{  
    k = i;  
}  
//k = 10, i = 11
```

Bucle infinito:

```
for ( ; ; )  
{  
    Instrucciones  
}
```




Sentencias iterativas: bucle while()

```
while (condición)
{
    Instrucciones
}
```

Ejemplo:

```
int p = 10;
while (p<20)
{
    Instrucciones
}
```

Bucle infinito:

```
while ( 1 )
{
    Instrucciones
}
```



Sentencias iterativas: bucle do while()

Aquí al menos las instrucciones se ejecutan una sola vez si es que la condición dentro del while() es falsa desde un inicio.

```
do
{
    Instrucciones
}while(condición);
```

Ejemplo:

```
unsigned char a=2,k=0;
void main()
{
    k = 5;
    do
    {
        a ++;
    }while(k<4);
}
//a = 3
```



La sentencia break

Esta sentencia se utiliza para finalizar la iteración cuando ocurre cierta condición en su interior.

Ejemplo: Calcular el factorial de 5.

```
unsigned char fact = 1;
void main()
{
    for(i = 1; i<10; i++)
    {
        if(i<6) fact = fact*i;
        else    break;
    }
    while(1);
}
```



La sentencia continue

continue es una sentencia que se utiliza dentro de los bucles iterativos para controlar el flujo de estos.

Esta instrucción normalmente se utiliza dentro de un condicional if para generar la siguiente iteración.

Ejemplo: Hacer un programa que sume los números pares del 1 al 10.

```
unsigned char suma;
void main()
{
    unsigned char i = 0;
    for( i = 0; i<=10; i++)
    {
        if(i%2 != 0) continue;
        suma+ = i;
    }
    while(1);
}
```



Printf(“controlString”, arg1,arg2... argN)

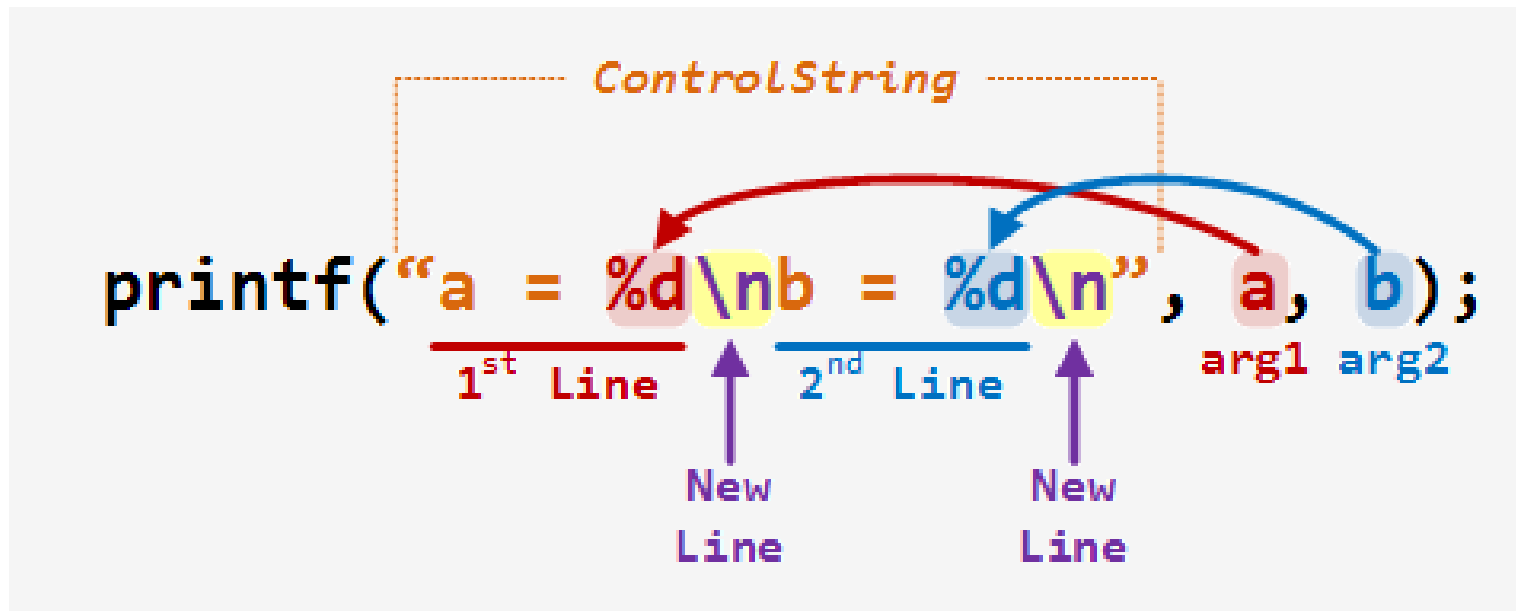
Imprime un texto en la pantalla PC o un terminal serial UART.

Este requiere de una altos recursos de memoria

```
1  {  
2      int a = 5;  
3      int b = 10;  
4      printf("a = %d\nb = %d\n", a, b);  
5  }
```



Printf("controlString", arg1,arg2... argN)





Printf(“controlString”, arg1,arg2... argN)

Formato de datos

Format Specifier	Meaning
%c	Single character
%s	String (all characters until '\0')
%d	Signed decimal integer
%o	Unsigned octal integer
%u	Unsigned decimal integer
%x	Unsigned hexadecimal integer with lower case digits (e.g. 1a5e)
%X	Same as %x but with upper case digits (e.g. 1A5E)
%f	Signed decimal value (floating point)
%e	Signed decimal value with exponent (e.g. 1.26e-5)
%E	Same as %e but uses upper case E for exponent (e.g. 1.26E-5)
%g	Same as %e or %f, depending on size and precision of value
%G	Same as %g but will use capital E for exponent