VARIÁVEIS E DADOS

		Range		
Type-Specifier	**Size**	**Unsigned**	**Signed**	**Digits**
int1	1 bit number	0 to 1	N/A	1/2
int8	8 bit number	0 to 255	-128 to 127	2-3
int16	16 bit number	0 to 65535	-32768 to 32767	4-5
int32	32 bit number	0 to 4294967295	-2147483648 to 2147483647	9-10
float32	32 bit float	-1.5×10^{45} to 3.4×10^{38}		7-8

C Standard Type	Default Type
short	int1
char	unsigned int8
int	int8
long	int16
long long	int32
float	float32

DECLARAÇÃO DE VARIÁVEIS

Tipos de dados	Tamanho em bits	Descrição	exemplo
int	**8**	variáveis inteiras sem sinal, capacidade de 0 a 255	int X, Y=1, VAR1=0;
signed int	**8**	variáveis inteiras com sinal, capacidade de –128 a 127	signed int VAR3 = -10;
long	**16**	variáveis inteiras sem sinal, capacidade de 0 a 65535	long A, B = 0, VAR2;
signed long	**16**	variáveis inteiras com sinal, capac. de –32768 a 32767	signed long GRAU=0;
float	**32**	variáveis reais com sinal. Representação aproximada.	float VAR4, VAR5;
short ou **boolean**	**1**	variáveis lógicas, de um bit, podendo valer 0 ou 1	boolean FLAG1=0, SENSOR; short CHAVE;
int32	**32**	variáveis inteiras sem sinal, podendo valer de 0 a 4294967295	int32 CONTADOR;
signed int32	**32**	variáveis inteiras com sinal, podendo armazenar valores de -2147483648 a 2147483647	signed int32 VAR10;
char	**8**	variáveis que armazenam caracteres em forma de bytes.	char C = 'a', LETRA = ' '; H;
OBS: Os tipos de dados int e long podem assumir outras configurações em outros tipos de processadores. Ex: em um processador de 16 bits, o tipo de dado INT assume 65536 possibilidades diferentes.			

PALAVRAS RESERVADAS DA LINGUAGEM

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

EXEMPLOS DE IDENTIFICADORES DE VARIÁVEIS

variavel	válido
variavel1	válido
1abc	inválido
variável	inválido
_teste_2	válido
return	inválido
_123_abc	válido
erro\2	inválido

OPERADORES

+ (adição)	== (igual)	<= (menor igual)	| (ou binário)	= (atribuição)
- (subtração)	!= (diferente)	&& (e)	>> (rotação binária para direita)	++ (incremento)
* (multiplicação)	> (maior que)	|| (ou)	<< (rotação binária para esquerda)	-- (decremento)
/ (divisão)	< (menor que)	! (negação)	~ (negação binária)	
% (resto divisão)	>= (maior igual)	& (e binário)	^ (ou exclusivo)	

Categoria	Operador	Ação	Exemplos	Obs.

Atribuição e prioridade	**=**	Atribuição (recebe)	int x **=** 5;
			x **=** 20;
			a **=** x + y / 2;
			x=y=z=105;
				A atribuição indica que tudo que está a sua direita será computado, resolvido e a solução obtida será armazenada na área de memória (variável) indicada a sua esquerda.

	( )	Prioridade	x = 20 * **(**3 + 1**)**;	Em uma expressão, os parênteses indicam o que será executado prioritariamente. No exemplo citado, caso não fosse utilizado, a multiplicação ocorreria antes da soma.
Aritméticos	**+**	Soma	x = a **+** b;	Retorna a soma de dois elementos.
	-	Subtração ou Inversão de sinal	x = a **–** b; x = -x;	Subtração ou inversão de sinal.
	*	Multiplicação	x = a ***** b;	Multiplicação. Assim como a divisão, tem prioridade sobre a soma e a subtração.
	/	Divisão	x = a **/** b;	Divisão. Quando feita com valores inteiros, o resultado também é um inteiro.
	%	Resto de divisão (somente inteiros)	x = a **%** b;	Somente para valores inteiros, retorna o resto da divisão do primeiro termo pelo segundo.
Relacionais	**>**	Maior que	if (a > b) ...; x = a > b;	Retorna (1) se o primeiro operando for maior que o segundo, e (0) caso contrário.
	>=	Maior ou igual a	if (a >= b) ...; x = a >= b;	Retorna (1) se o primeiro operando for maior ou igual ao segundo. Caso contrário, retorna (0).
	<	Menor que	if (a < b) ...; x = a < b;	Retorna (1) se o primeiro operando for menor que o segundo, e (0) caso contrário.
	<=	Menor ou igual a	if (a <= b) ...; x = a <= b;	Retorna (1) se o primeiro operando for menor ou igual ao segundo. Caso contrário, retorna (0).
	==	Igual	if (a == b) ...; x = a == b;	Retorna (1) se o primeiro operando for igual ao segundo, e (0) se for diferente.
	!=	Diferente	if (a != b) ...; x = a != b;	Retorna (1) se o primeiro operando for diferente que o segundo, e (0) se for igual.
Lógicos	&&	E (AND)	if ((a > b) **&&** (a > c)) ...; x = (a **&&** c);	Retorna verdadeiro (1) somente se o primeiro operando for **verdadeiro** e o segundo operando também for **verdadeiro**.
	||	OU (OR)	if ((a > b) **||** (a > c)) ...; x = (a **||** c);	Retorna verdadeiro (1) se pelo menos um dos operandos for **verdadeiro**. Se ambos forem falso (0), o valor retornado será falso (0).
	!	NÃO (NOT)	if (**!**(a > b))...; x = **!**a;	Retorna o valor lógico invertido. verdadeiro = **!**falso falso = **!**verdadeiro
	&	AND (E)	x = a & b;	Operação binária E entre os bits de dois números binários. Se o bit n de

				ambos os valores estiverem ligados, o bit n do resultado também estará ligado. 1010 & 0110 = 0010
	|	OR (OU)	x = a | b;	Operação binária OU entre os bits de dois números binários. Se o bit n do primeiro valor, ou do segundo valor, ou de ambos estiver ligado, o bit n do resultado também estará ligado. 1010 | 0110 = 1110
	^	XOR (OU exclusivo)	x = a ^ b;	Operação binária OU EXCLUSIVO entre os bits de dois números binários. Se o bit n do primeiro valor estiver ligado e o bit n do segundo valor desligado, ou se o bit n do primeiro valor estiver desligado, e do segundo valor ligado, o bit n do resultado estará ligado. 1010 ^ 0110 = 1100
	~	NOT (NÃO)	x = ~a;	Inverte o valor de todos os bits. ~ 0101 = 1010
	>>	RIGHT SHIFT (Deslocamento de bits para direita)	x = a >> 1; x = a >> 3;	Desloca os bits para direita, um determinado número de casas. 0101 >> 1 = 0010 0101 >> 2 = 0001
	<<	LEFT SHIFT (Deslocamento de bits para esquerda)	x = a << 1; x = a << 2;	Desloca os bits para esquerda, um determinado número de casas. 0101 << 1 = 1010 0101 << 2 = 0100

OPERADORES AVANÇADOS

++	Incremento	x++; ++x;	Incremento de 1. Se usado antes da variável, o valor da variável é incrementado ante do uso da mesma. Se aparecer depois, o valor é incrementado após o uso da mesma.
--	Decremento	x--; --x;	Similar ao anterior, realizando o decremento da variável.
+=	Incremento	x += 10;	O exemplo equivale a x = x + 10;
-=	Decremento	x -= 10;	O exemplo equivale a x = x – 10;
***=**	Produto	x *= 10;	O exemplo equivale a x = x * 10;
/=	Divisão	x /= 10;	O exemplo equivale a x = x / 10;
%=	Resto da divisão	x %= 10;	Equivale a x = x % 10;
<<=	Rotação esquerda	x <<= 1;	Equivale a x = x << 1;
>>=	Rotação direita	x >>= 1;	Equivale a x = x >> 1;
Estas operações contractas podem ser utilizadas com quase todos os operadores aqui estudados, trazendo em alguns casos (dependendo do compilador), uma redução no código objeto gerado (arquivo HEX menor)			

FUNÇÕES MAIS UTILIZADAS DA LINGUAGEM C PADRÃO CCS

Função	Descrição	Exemplo
output_high()	Ativa um determinado pino do microcontrolador	output_high(PIN_D0); output_high(PIN_C2);
output_low()	Desativa um determinado pino do microcontrolador	output_low(PIN_D0); output_low(PIN_C2);

input()	Busca o estado de um pino	if (input(PIN_A1))
		{
		output_high(PIN_D0);
		x = input(PIN_A4);
		}

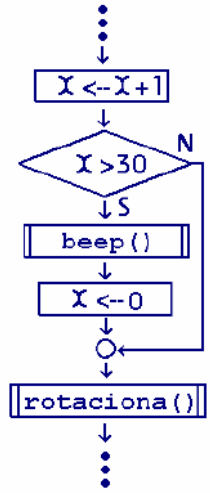
output_a()	Envia um byte para o PORT A	output_a(VAR1); // envia VAR1 para PORTA
output_b()	Envia um byte para o PORT B	output_b(0xff); // liga todos bits de PORTB
output_c()	Envia um byte para o PORT C	output_c(VAR1); // envia VAR1 para PORTC
output_d()	Envia um byte para o PORT D	output_d(0x00); // desliga todos os bits de PORTD
output_e()	Envia um byte para o PORT E	output_e(VAR1); // envia VAR1 para PORTE
input_a()	Busca um byte do PORT A	int VAR1; VAR1 = input_a();
input_b()	Busca um byte do PORT B	int VAR1; VAR1 = input_b();
input_c()	Busca um byte do PORT C	int VAR1; VAR1 = input_c();
input_d()	Busca um byte do PORT D	int VAR1; VAR1 = input_d();
input_e()	Busca um byte do PORT E	int VAR1; VAR1 = input_e();

lcd_init()	Inicializa o LCD.	#use delay (clock=4000000)
		#define use_portb_lcd true
		#include <lcd.c>
		…
		void main()
		{
		…
		lcd_init();
		…
		}

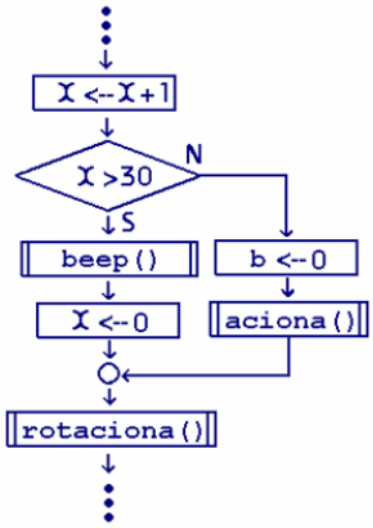
lcd_putc()	Envia uma string (seqüência de caracteres) para o LCD	lcd_putc(“ \f TESTE”);
delay_ms()	Causa um atraso em milésimos de segundo	delay_ms(VAR1); delay_ms(100);
delay_us()	Causa um atraso em milionésimos de segundo	delay_us(10); delay_us(VAR1);
printf()	Cria uma saída formatada, geralmente utilizada para exibir dados das variáveis no LCD	float VAR1; int VAR2; long VAR3; printf(lcd_putc, “\f TESTE %f”, VAR1); printf(lcd_putc,”\fTESTE\n %lu %f”, VAR3, VAR1);

COMANDOS E FUNÇÕES
IF

```
if (condição)
  {
  instruções;
  }
```




```
...
x ++ ;
if (x > 30)
    {
    beep();
    x = 0;
    }
rotaciona();
...
```




```
if (condição)
  {
  intruções;
  }
else
  {
  instruções;
  }
```

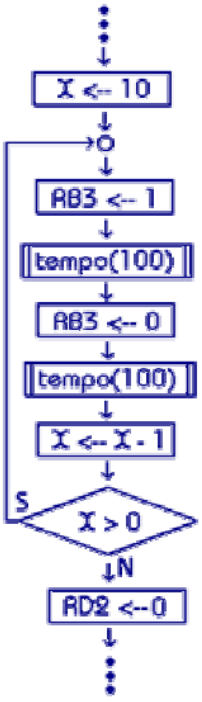
```
...
x ++ ;
if (x > 30)
    {
    beep();
    x = 0;
    }
else
    {
    b = 0;
    aciona();
    }
rotaciona();
...
```

DO-WHILE

```
do {
    ...;
    } while (condição);
```

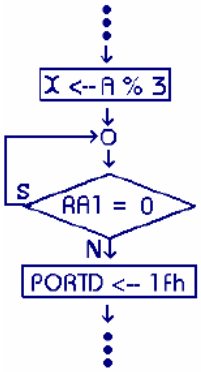
do / while singifica faça / enquanto.

Ao contrário do while, o do/while permite que o bloco seja executado ao menos uma vez.




```
...
x =10;
do {
    output_high(PIN_B3);
    delay_ms(100);
    output_low(PIN_B3);
    delay_ms(100);
    x --;
    } while (x > 0);
output_low(PIN_D2);
...
```

WHILE



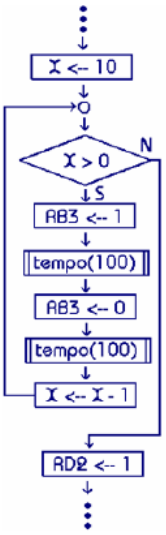

```
...
while(condição);
```

```
x = a % 3;
while(input(PIN_A1)==0);
PORTD = 0x1F;
...
```

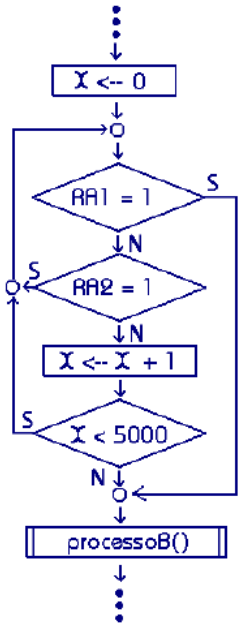
While vazio. Muito útil quando se deseja **"reter"** o programa até que uma condição ocorra.
A condição entre os parênteses indica a condição na qual o sistema ficará retido. Lembre que WHILE significa ENQUANTO. O PONTO e VÍRGULA identifica o laço vazio.

```
while (condição)
    {
    ....;
    ....;
    }

// enquanto a condição for
// verdadeira, serão executadas
// as instruções entre as chaves
// ou a instrução seguinte no caso
// de não existirem as chaves.
```



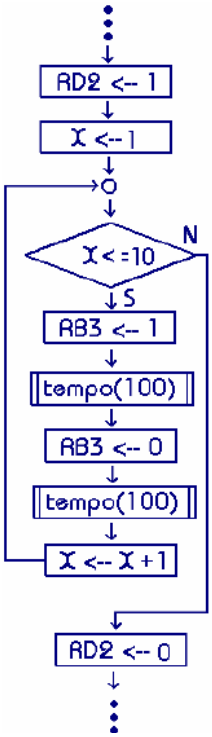

```
...
x =10;
while (x > 0){
                output_high(PIN_B3);
                delay_ms(100);
                output_low(PIN_B3);
                delay_ms(100);
                x --;
                }
output_high(PIN_D2);
...
```




```
...
x = 0;
do{
  if (input(PIN_A1)) break;
  if (input(PIN_A2)) continue;
  x ++;
  } while(x < 5000);
processoB();
...      ou ainda ...
x = 0;
while(!input(PIN_A1))
  {
  if (input(PIN_A2)) continue;
  x ++;
  if (x >= 500) break;
  }
processoB();
...
```

```
while (condição)
    {
    ....;
    if (condição)
      continue; // volta p/ inicio do while
    ....;
    if (condição)
      {
      ...;
      break; // interrompe o while
      }
    }
```

FOR




```
...
output_high(PIN_D2);
for (x = 1; x <= 10; x ++)
        {
        output_high(PIN_B3);
        delay_ms(100);
        output_low(PIN_B3);
        delay_ms(100);
        }
output_low(PIN_D2);
...
```

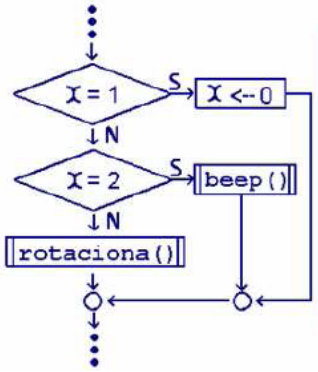
```
for (inicializ ; condição ; incrm)
    ...;
    }
```

O for geralmente é utilizado para se repetir um determinado bloco baseado na contagem (incremetal ou decremental) de uma variável de tipo inteiro (int, long, int32, etc...)

Inicializ : comando a ser executado antes da primeira interação do laço.
Condição : situação para continuar o laço.
Incrm: incremento ou decremento da variável de controle.

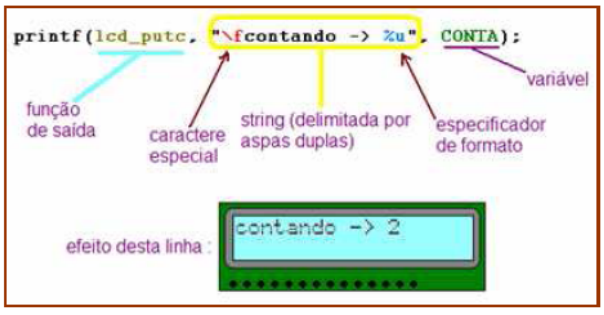
SWITCH

```
switch (variável)
  {
  case   VALOR1 : instruções;
                  break;
  case   VALOR2 : instruções;
                  break;
  default       : instruções;
  }
```




```
...
switch(x)
  {
  case 1 : x = 0;
           break;
  case 2 : beep();
           break;
  default : rotaciona();
  }
...
```

PRINTF



Caracter especial	Função
\f	Limpar display
\n	Pular linha
\\	Barra invertida
\0	Null

Especificador	Tipo
%u	int ou short
%i ou %d	signed int

%lu	long ou int32
%li ou %ld	signed long
%X	int em hexadec.
%f	Float
%c	Caractere
%s	String
%e	float (not.cientf.)
%lx	long hex
%%	símbolo %
%3u	int (3 casas)
%03u	int (3 digitos c/ zeros à esq.)
%1.2f	float (2 casas dec.)

Exemplos de uso :

```
int vlr;
....
printf(lcd_putc,"\f Valor: %u", vlr);
```

```
float a, b; int c;
...
printf(lcd_putc,"\fCont: %u \nX:%1.2f Y:%1.2f", c, a, b);
```

DESCRIÇÃO DAS PRINCIPAIS FUNÇÕES EMBUTIDAS NO COMPILADOR PCW (3.4 ou sup.)

FUNÇÕES DE I/O VIA RS232	
getc() ou getchar();	Busca caractere via porta serial
putc() ou putchar();	Envia caractere via porta serial
fgetc();	Busca caractere em um dispositivo
gets();	Envia uma string pela porta serial
puts();	Envia seqüência de caracteres via porta serial
fgets();	Busca uma seqüência de caracteres via porta serial
fputc();	Envia um caractere a um dispositivo
fputs();	Envia uma seqüência de caracteres a um dispositivo
printf();	Imprime uma seqüência formatada de texto em um dispositivo
kbhit();	Verifica se há caractere disponível na entrada serial
fprintf();	Saída formatada para um dispositivo
set_uart_speed();	Determina velocidade da porta serial
perror();	Imprime uma mensagem de erro no dispositivo padrão de saída
assert();	Usado para depuração
SPI (I/O 2 fios)	
setup_spi();	Inicializa SPI
spi_read();	Lê da interface serial
spi_write();	Grava na interface serial
spi_data_is_in();	Retorna "verdadeiro" se existem dados recebidos pela SPI

ENTRADA E SAÍDA DIGITAL	
output_low();	Desativa uma saída
output_high();	Ativa uma saída
output_float();	Habilita o terceiro estado do pino (coletor aberto)
output_bit();	Envia o valor de um bit para um pino
input();	Lê o valor de um pino
output_a();	Envia um byte para o PORTA
output_b();	Envia um byte para o PORTB
output_c();	Envia um byte para o PORTC
output_d();	Envia um byte para o PORTD
output_e();	Envia um byte para o PORTE
input_a();	Lê um byte do PORTA
input_b();	Lê um byte do PORTB
input_c();	Lê um byte do PORTC
input_d();	Lê um byte do PORTD
input_e();	Lê um byte do PORTE
port_b_pullups();	Ativa os PULL-Ups de entrada do portb
set_tris_a();	Define a direção para os pinos do PORTA
set_tris_b();	Define a direção para os pinos do PORTB
set_tris_c();	Define a direção para os pinos do PORTC
set_tris_d();	Define a direção para os pinos do PORTD
set_tris_e();	Define a direção para os pinos do PORTE
PWM	
setup_ccpX();	Define o modo de operação dos pinos de PWM
set_pwmX_duty();	Determina o valor do PWM, de 0 a 1023
INTERFACE DE PORTA PARALELA ESCRAVA (PORTD)	
setup_psp()	Ativa a porta paralela escrava
psp_input_full()	Verifica o funcionamento do recurso de porta paralela escrava
psp_output_full()	Verifica o funcionamento do recurso de porta paralela escrava
psp_overflow()	Verifica o funcionamento do recurso de porta paralela escrava
I2C	
i2c_start()	Inicia interface I2C
i2c_stop()	Para interface I2C
i2c_read()	Lê byte da interface I2C
i2c_write()	Grava byte na interface I2C
i2c_poll()	Verifica buffer da interface
PROCESSOR	
sleep()	Entra em modo SLEEP
reset_cpu()	Reinicia (reseta) o microcontrolador
restart_cause()	Retorna a causa do último reset
disable_interrupts()	Desativa interrupções
ext_int_edge()	Configura comportamento da interrupção por borda
read_bank()	Lê o valor de um registrador em um determinado banco
write_bank()	Grava uma informação em uma posição de memória
label_address()	Endereço ROM representado por um rótulo
goto_address()	Desvia a execução para um endereço ROM
getenv()	Retorna o valor de uma variável de ambiente
BIT/BYTE	
shift_right()	Rola dados para direita.
shift_left()	Rola dados para esquerda.
rotate_right()	Rotaciona dados para direita.
rotate_left()	Rotaciona dados para esquerda.
bit_clear()	Limpa um bit de uma variável
bit_set()	Ativa um bit de uma variável
bit_test()	Testa um bit de uma variável
swap()	Troca os nibbles de uma variável de 8 bits
make8()	Extrai um byte de uma variável
make16()	Extrai uma Word de uma variável
make32()	Extrai um valor de 32 bits de uma variável
ANALOG	
setup_comparator()	Configura o comparador
setup_adc_ports()	Configura portas usadas pelo conversor AD
setup_adc()	Configura o AD
set_adc_channel()	Determina o canal a ser utilizado
read_adc()	Lê valor do canal AD ativado

MATEMÁTICAS	
abs()	Retorna valor absoluto
acos()	Arco cosseno
asin()	Arco seno
atan()	Arco tangente
ceil()	Arredonda acima um float para número inteiro.
cos()	Cosseno
exp()	Calcula função E de um número.
floor()	Arredodna abaixo um float para número inteiro.
labs()	Calcula o valor absoluto de um long
sinh()	Seno hiperbólico
log()	Logaritmo natural
log10()	Logaritmo base 10
pow()	Potência
sin()	Seno
cosh()	Cosseno hiperbólico
tanh()	Tangente hiperbólica
fabs()	Valor absoluto para um float
fmod()	Resto da divisão de ponto flutuante
atan2()	Arco tangente
frexp()	Quebra um float
ldexp()	
modf()	Quebra um float em inteiro e decimal
sqrt()	Raiz quadrada
tan()	Tangente
div()	Divisão retornando quociente e resto
ldiv()	Divisão de um long retornando quociente e resto
VOLTAGE REF	
setup_vref()	Estabelece tensão de refer. dos comparadores

STANDARD	
atoi()	Transforma ASCII em int
atoi32()	Transforma ASCII em int32
atol()	Transforma ASCII em long
atof()	Transforma ASCII em float
tolower()	Transforma letras maiúsculas em minúsculas
toupper()	Transforma letras minúsculas em maiúsculas
isalnum()	Verifica se uma string é numérica
isalpha()	Verifica se uma string é alfabética
isamoung()	Verifica se um caractere pertence a uma string
isdigit()	Verifica se é número
islower()	Verifica se é letra minúscula
isspace()	Verifica se é espaço
isupper()	Verifica se é letra maiúscula
isxdigit()	Verifica se é dígito hexadecimal
strlen()	Retorna comprimento de uma string
strcpy()	Copia uma string
strncpy()	Copia com limite de caracteres
strcmp()	Compara strings
stricmp()	Compara strings ignorando maiúscula/minúscula
strncmp()	Compara com limite de caracteres
strcat()	Concatena strings
strstr()	Procura por uma string dentro de outra
strchr()	Procura caracteres em uma string
strrchr()	Procura caracteres em uma string, de traz para frente.
strtok()	Aponta para próximo caractere após separador em uma string
strspn()	Conta caracteres iniciais em strings
strcspn()	Conta caracteres iniciais em strings
strpbrk()	Procura primeiro caracter comum em strings
strlwr()	Converte uma string em minúsculas
sprintf()	Imprime (printf) em uma string
isgraph()	Testa se é caractere gráfico
iscntrl()	Testa se é caractere de controle

isprint()	Testa se é imprimível
strtod()	Extrai um float de uma string
strtol()	Extrai um inteiro de uma string
strtoul()	Idem
strncat()	Concatena com limite de caracteres
strcoll()	Compara caracteres em uma string
strxfrm()	Compara caracteres em uma string

TIMERS

setup_timer_x()	Configura funcionamento dos TIMERS
set_timer_x()	Inicializa o TIMER
get_timer_x()	Busca valor de um TIMER
setup_counters()	Configura contador
setup_wdt()	Configura o Watch Dog Timer
restart_wdt()	Reinicia o Watch Dog Timer

DELAYS

delay_us()	Causa um atraso (tempo) em milhonésimos de segundos
delay_ms()	Causa um atraso (tempo) em milésimos de segundos
delay_cycles()	Causa um atraso em número de ciclos (clock / 4)

STANDARD C

memset()	Copiar um conjunto de dados na memória
memcpy()	Copiar um conjunto de dados na memória
offsetof()	Retorna valor de deslocamento de dados na memória
offsetofbit()	Retorna valor de deslocamento de dados na memóira
malloc()	Aloca dinamicamente uma área de memória
calloc()	Aloca dinamicamente uma área de memória
free()	Libera memória alocada por malloc ou calloc
realloc()	Realoca memória
memmove()	Copiar um conjunto de dados na memória
memcmp()	
memchr()	

EEPROM

read_eeprom()	Ler um byte da EEPROM
write_eeprom()	Gravar um byte na EEPROM
read_program_eeprom()	Ler área da ROM de programa
write_program_eeprom()	Gravar algo na área de ROM de programa (flash)
read_calibration()	Função exclusiva para PIC14000 – lê dado da memória de calibração
write_program_memory()	Grava uma seqüência de bytes na memória de programa
read_program_memory()	Lê uma seqüência de bytes da memória de programa
write_external_memory()	Grava em uma memória externa. Pode depender de implementação.
erase_program_memory()	Apaga uma área da memória flashROM
setup_external_memory()	Configura forma de utilização de memória externa.

STANDARD C SPECIAL

rand()	Geração de número aleatório
srand()	Define o valor máximo para geração de número aleatório