Universidade Federal de Uberlândia
Engenharia Eletrônica e de Telecomunicações

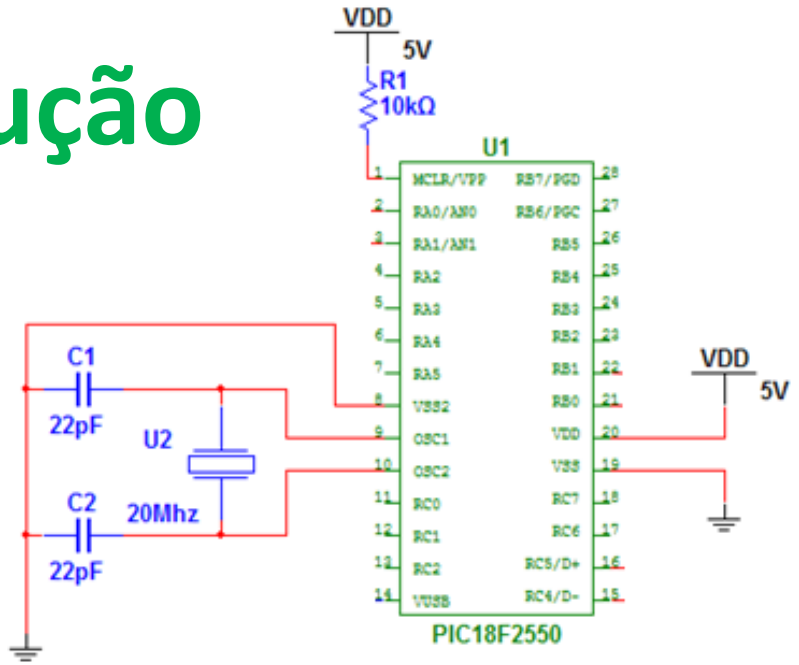
– Microprocessadores –

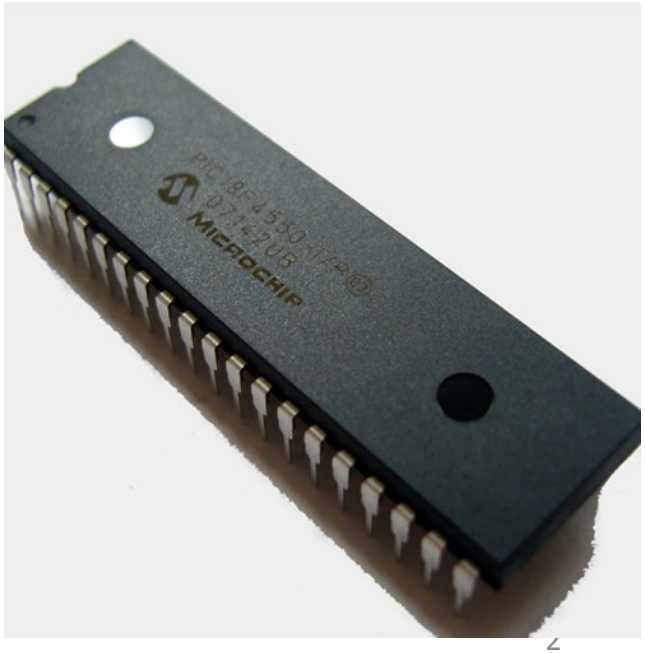
Cap. 3 – Treinamento em C para o compilador CCS

Prof. Alan Petrônio Pinheiro

0 - Introdução

- Por que C ?
- Porque CCS /PCW
  - PCB (série 12 e 16C – 12bits)
  - PCM (séries 14 e 16 – 14bits)
  - PCH (serie 18 – 16bits)
  - PCD (série 24 e 32 – 24bits)
- Ferramentas auxiliares: MPLab

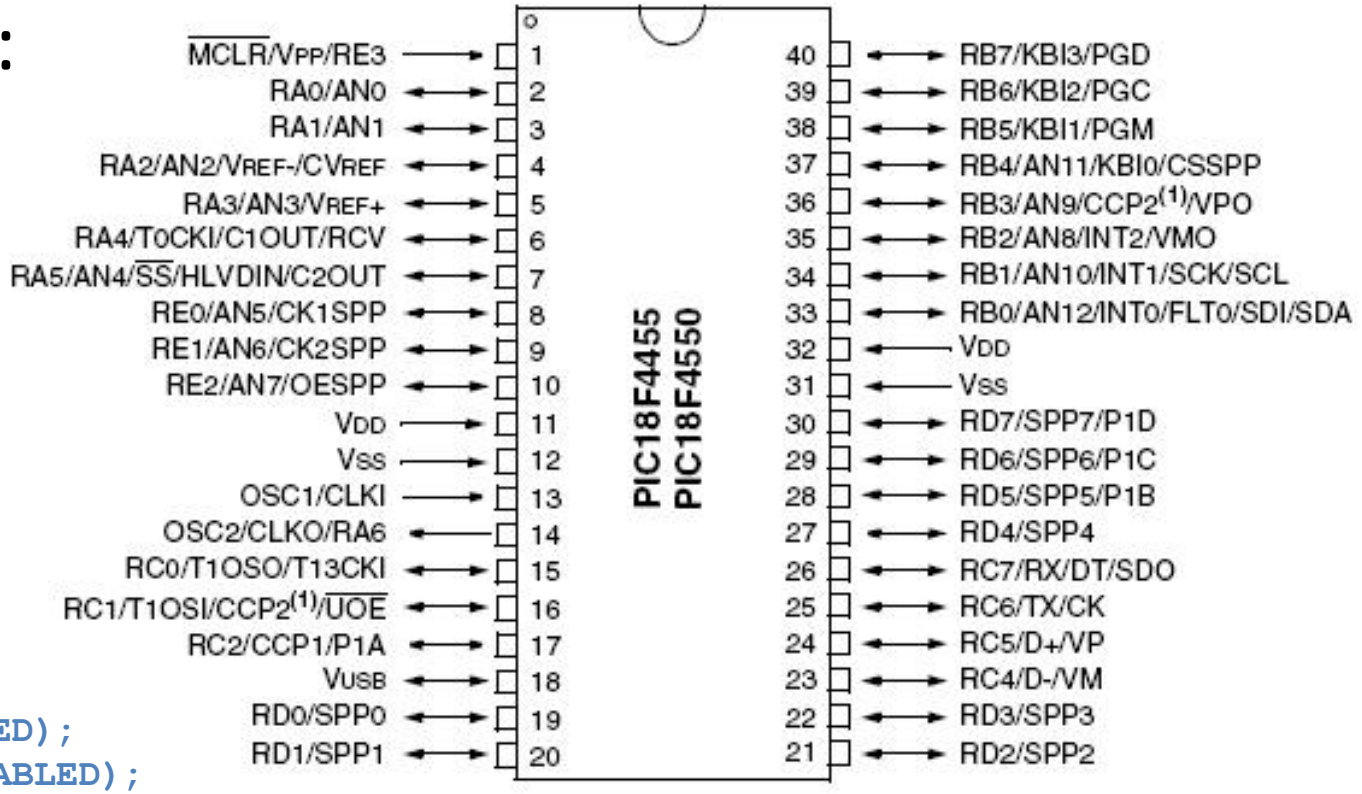




- Exemplo CCS:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
```




```
   setup_adc_ports(NO);
   setup_adc(ADC_OFF);
   setup_psp(PSP_DISABLED);
   setup_spi(SPI_SS_DISABLED);
   setup_wdt(WDT_OFF);
   setup_timer_0(RTCC_INTERNAL);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
   setup_comparator(NC_NC_NC_NC);
   setup_vref(FALSE);

   // TODO: USER CODE!!
}
```

1 – Tipos de dados e variáveis

- **int1** (1 bit)
- **boolean** (1 bit)
- **char** (8 bits – 0 a 255)
- **int** (8 bits – 0 a 255)
- **int8** (8 bits – 0 a 255)
- **byte** (8 bits)
- **int16** (16 bits – 0 a 65535)
- **int32** (32 bits – 0 a 4.294.967.295)
- **float** (32 bits – $3,48.10^{-38}$ a $3,4.10^{+38}$)
- **void** (remete a "sem tipo de dado")

Type-Specifier	Size	Range		
		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
int48	48 bit number	0 to 281474976710655	-140737488355328 to 140737488355327	14-15
int64	64 bit number	N/A	-9223372036854775808 to 9223372036854775807	18-19
float32	32 bit float	-1.5×10^{45} to 3.4×10^{38}		7-8

- Modificadores de acesso:
  - `signed`
  - `unsigned`
  - `short`
  - `long`
  - Exemplos:
    - `signed int` = -128 a +127
    - `signed char` = -128 a +127
    - `short int` = 1bit = int1 = boolean
    - `long int` = 16 bits = int16

- Relembrando: variáveis globais e locais

```
#include <18F4550.h>
#include <stdio.h>
#include <stdlib.h>
#use delay(clock=20000000)
#use RS232 (baud=19200, xmit=PIN_B2, rcv=PIN_B1)
int somatorio;
void soma(int valor)
{
   int conta;
   somatorio = somatorio + valor;
   printf("A soma de 0");
   for(conta=1; conta<(valor+1); conta++)
      printf("+ %u", conta);
   printf("é igual a %u\r\n", somatorio);
}
void main()
{
   int conta;
...
   somatorio = 0;
   for(conta=1; conta<20; conta++)
      soma(conta);
}
```

- Representação de valores:

```
int valor;
valor = 50;              //sis. decimal
valor = 0x50;            //sis. hexade.
valor = 050;             //sis. octal
valor = 0b01010101;      //sis. binário
```

- Outras atribuições:

```
val1 = 50;          //inteiro
val2 = -5;          //inteiro negativo
val3 = 54.32;       //ponto flutuante (real)
val4 = 'a';         //caractere aspas simples
val5 = "teste";     //string com aspas duplas
```

2 – Diretivas de compilação

- **`#include`**
  - Exemplo:

    `#include "18F4550.h"` ou `#include <18F4550.h>`
  - Indica que anexe ao programa o arquivo 18F4550.h onde são feitas muitas declarações e definições e também as bibliotecas.
  - Bibliotecas mais usuais:
    - `stdio.h`
    - `stdlib.h`

- **#use**
  - Exemplo:

```
#use delay(clock=4000000)
#use rs232(baud=19200, xmit=PIN_B2, rcv=PIN_B3)
```

  - Diretiva interna do compilador usada para configurar determinados aspectos de execução do µC.

- **#define**
  - Exemplo:

```
#define numero_linhas  10
...
int matriz [numero_linhas];
```

- **#fuses**
  - Exemplo:

    ```
    #fuses NOWDT, BROWNOUT, NOLVP
    ```

  - Indica o estado dos "fusíveis" que indica em que modo o µC deve operar e quais recursos devem ou não estar disponíveis. No exemplo **NOWDT** indica watchdog desligado, **BROWNOUT** indica resete por brown-out ligado e **NOLVP** programação por baia tensão desligada.
  - Dica: para ver quais "fuses" são válidas para um determinado µC e quais seus significados, vá no PCW no menu **View > Valid Fuses**

- **#asm** e **#endasm**
  - Exemplo:

```
#asm
        movf   a,w
        addwf  b,w
        ; comandos aqui
#endasm
```

  - Em meio a um bloco de código C pode ser aberto um trecho de código assembly e depois fechado

- **#device**
  - Define o nome do processador usado. Exemplo:

```
#device PIC18F4550 ADC=8
```

  - Especifica o pic18f4550 e indica que ele deve retornar 8 bits de conversão a partir da função read_adc();

3 – Comandos de controle

- IF

```
if (condicao)
{
   comandos;
}
else
{
   comandos;
}
```

- SWITCH

```
switch(variavel)
{
  0: output_low(PIN_B0);
     break;
  1: output_high(PIN_B0);
     break;
  default:
     output_high(PIN_B1);
}
```

- FOR

```
for (inicialia; condicao; incremento)
{
    comando;
    comando;
}
```

Exemplo:

```
for (x=0, y=110; x<=100 || y < 1; x++, y--)
{
    printf("%u, %u", x, y);
}
```

- WHILE

```
while (condicao)
{
    comandos;
}
```

- DO WHILE

```
do
{
    comandos;
} while (condicao)
```

4 – Criando funções

- Forma geral:

```
tipo_dado  nome_funcao (parametros)
{
   //comandos
}
```

– Exemplos:

```
float divide (float  a, float b)
 {
      If (b)
          return a/b;
      else
          return 0;
 }
```

```
void divide (float *a, float *b)
{
     If(*b)
          *a=*a/*b
     else
          *a=0;
}
```

- Observação: inserir função antes de main ou declarar antes

```
#include <18f4550.h>

int multiplica (int a, int b)
 {
  return a+b;
 }

void main()
{
  int resultado = 0;
  setup_adc_ports(NO_ANALOGS|VSS_VDD);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(SPI_SS_DISABLED);
  setup_wdt(WDT_OFF);

  resultado = multiplica(2,3);
}
```

```
#include <18f4550.h>

int multiplica (int, int);

void main()
{
  int resultado = 0;
  setup_adc_ports(NO_ANALOGS|VSS_VDD);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(SPI_SS_DISABLED);
  setup_wdt(WDT_OFF);

  resultado = multiplica(2,3);
}
int multiplica (int a, int b)
{
   return a+b;
}
```

- "Janelas" Assembly em funções:

```
int soma (int, int);
void main()
{
   int resultado = 0;
   setup_adc_ports(NO_ANALOGS|VSS_VDD);
   setup_adc(ADC_OFF);
   setup_psp(PSP_DISABLED);
   setup_spi(SPI_SS_DISABLED);
   setup_wdt(WDT_OFF);

   resultado = soma(2,3);
}
int soma (int a, int b)
{
    #asm
      movf  a,w
      addwf b,w
      movwf _RETURN_
    #endasm
}
```

- Passando matrizes para funções:

```
void taxa_inflacao (float *matriz, float correcao)
{
  int contador =0;
  for (contador=0; contador<10; contador ++)
      matriz[contador]=matriz[contador]*correcao;
}

void main()
{
    float precos[10];
     float taxa_inflacao = 1.05;
     corrige_inflacao(precos, taxa_inflacao );
}
```

5 – Operações matemáticas e lógicas

- Operações matemáticas:

Operador	Descrição
+	(Adição)
-	(Subtração)
*	(Multiplicação)
/	(Divisão)
%	(Resto/Módulo)

  - Observação: ++ e – (não podem ser usadas em dados ‘complexos’)
  - Exemplo:

```
5 % 2; /* o resultado é 1, pois 5/2 é 2 na parte
  inteira e tem resto 1 */
```

- Operadores relacionais

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

- Lógicos booleanos

Operador	Ação
&&	AND (E)
||	OR (OU)
!	NOT (NÃO)

- Lógicos bit a bit

```
int a,b;
b = 82;
a = b & 0b00000001;
a = ~ a;
a = a >> 1;
```

Operador	Ação
&	AND
|	OR
^	XOR (OR exclusivo)
~	NOT
>>	Deslocamento de bits a direita
<<	Deslocamento de bits a esquerda

- Resumo de expressões:

Expressão Original	Expressão Equivalente
x=x+k;	x+=k;
x=x-k;	x-=k;
x=x*k;	x*=k;
x=x/k;	x/=k;
x=x>>k;	x>>=k;
x=x<<k;	x<<=k;
x=x&k;	x&=k;
etc...	

- Conversões em tipos de dados
  - Casting:

```
int x = 11;
float resul;
resul = x/2;                //resul = 5
resul = (float) x/2;        //resul = 5.5
```

  - Outro exemplo:

```
int a, b;
float c;
a = b = 2;
c = 10 + ( (float) a/b); //sem casting c seria inteiro
```

6 – Ponteiros, matrizes e estruturas

- Ponteiros:

```
int *var_ponteiro;     /*declaracao de um ponteiro que aponta para inteiro */
int var_inteira = 10;
var_ponteiro= &var_inteira; /* atribui a var_ponteiro o endereco de var_inteira*/
*var_ponteiro= 20; /* atribui o valor 20 ao endereço apontado por var_ponteiro.
                      Assim por consequencia var_inteira também passa a valer 20*/
```

- Vetores e matrizes:

```
int valores [3]={0,1,2};     /*vetor com 3 (de 0 a 2) inteiros incializados
char nome [5]={'a','l','a','n','\o'};/*vetor com 4 (de 0 a 3) caracteres
    e um de final
int16 saldo [6];     /*vetor com 6 valores (de 0 a 5) de 16btis cada sem inicalizacao
int dados [2][3];     /*matriz de 2 linhas e 3 colunas de inteiros
```

- STRUCT

```
struct NOME_DA_ESTRUTURA
{
    tipo variavel1;
    tipo variavel2;
};
```

Exemplo:

```
struct cadastro
{
   char nome [30];
   char endereco [80];
   int idade;
};
struct cadastro cliente1;
cliente1.nome = "Alan";
cliente1.idade = 20;
```

7 – Funções do CCS

7.1 - Manipulação de bit e byte

- **BIT_CLEAR():** apaga um bit de uma variável

```
int teste = 0x2502;
bit_clear(teste, 1); //teste=0x2500
```

- **MAKE8():** lê um dos bytes que compõem uma variável de 16 ou 32 bits

```
int teste = 0x2502;
int x = make8 (teste, 1); //x=0x25
```

- **BIT_SET():** seta um bit de uma variável
- Outras funções de bit/byte: **SHIFT_RIGHT(), SHIFT_LEFT(), ROTATE_RIGHT(), ROTATE_LEFT(), BIT_TEST(), SWAP(), MAKE16(), MAKE32() ...**

7.2 - Entrada e saída

- **OUTPUT_LOW():** coloca pino em 0

```
output_low(PIN_A0); //pin_a0 definido
em arquivos de cabeçalho
```

- **OUTPUT_HIGH():** coloca pino em 1
- **OUTPUT_FLOAT():** coloca pino em alta impedância

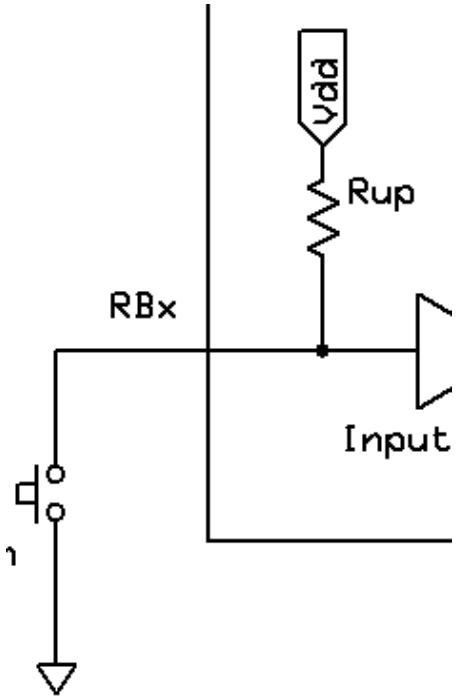
```
output_flow(PIN_A0); //coloca pin A0
como entrada
```

- **OUTPUT_BIT()**

```
output_bit( PIN_B0, 0);
// mesmo que output_low(pin_B0);
output_bit( PIN_B0,input( PIN_B1 ) );
// faz pin_B0 com mesmo nível de B1
```

- **OUTPUT_X():** escreve um byte qualquer na porta X (onde X pode ser A, B, C, ... I, K) do microcontrolador

```
output_A (0x255);
output_E (0b10101010);
```



- **INPUT():** lê um pino do microcontrolador

```
valor = input(PIN_A7);
```

- **INPUT_X():** lê 1 byte completo de uma determinada porta X do micro

```
valor = input_A();
```

- **PORT_X_PULLUPS():** ativa ou desativa as resistências pullups internas das portas X (onde X pode ser A, B, D, E ou I)

```
port_a_pullups(true); //liga pullaps
```

- **SET_TRIS_X():** configura a direção dos pinos de uma porta X (onde X pode ser A, B, C, ... I, K)

```
set_tris_D(0b00001111);  /*pinos RD0 a RD3 como entradas e RD4 a RD7 como saídas*/
```

- ***Observação***: uso da diretiva fast_io. Exemplo:

```
#use fast_io(A);
```

Configura o compilador para gerar código que execute I/O sem programação da direção dos registradores de saída.

7.3 - Leituras analógicas

- **SETUP_COMPARATOR():** configura funcionamento dos comparadores internos

```
setup_comparator(A0_A3_A1_A2);
/* neste caso o primeiro comparador tem as duas primeiras
   entradas (+ e –) definidas nos pinos A0 e A3 e o
   segundo comparador tem as entradas (+ e –) definidas
   em A1 e A2 */
```

Outros modos possíveis de trabalho são:

- A0_A3_A1_A2
- A0_A2_A1_A2
- NC_NC_A1_A2
- NC_NC_NC_NC
- A0_VR_A1_VR
- A3_VR_A2_VR
- A0_A2_A1_A2_OUT_ON_A3_A4
- A3_A2_A1_A2

- **SETUP_VREF():** configura a tensão de referência interna do conversor AD

```
setup_vref(VREF_LOW | VREF_A2 |
2);
```

Outros modos

– FALSE: referência desligada

– VREF_LOW: faz VDD = VALOR/24

– VREF_HIGH : faz VDD = VALOR*VDD/32 + VDD/4

– any may be or'ed with VREF_A2.

- **SETUP_ADC():** configura o ADC interno

```
setup_adc(ADC_OFF);   //desliga o conversor
```

Onde temos as opções:

- `ADC_OFF`
- `ADC_CLOCK_INTERNAL`
- `ADC_CLOCK_DIV_32`

- **SETUP_ADC_PORTS():** configura as entradas analógicas do conversor

```
setup_adc_ports(ANALOG_RA3_REF);   /*todas analógicas e RA3 é a referência */
```

Onde temos as opções:

- `ALL_ANALOG`
- `NO_ANALOGS`
- `ANALOG_RA3_REF`
- `RA0_RA1_RA3_ANALOG (…)`

- **SETUP_ADC_CHANNEL():** seleciona canal de entrada para módulo AD interno

```
set_adc_channel(1);
delay_us(10);
valor = read_adc();
```

Observação: aguardar pelo menos 10u seg antes de efetuar a conversão AD e garantir a carga do sample and hold

- **READ_ADC():** efetua uma conversão AD

7.4 – Atraso de tempo

- **DELAY_US():** atrasa *n* microsegundos

  `delay_us(n)`

- **DELAY_MS():** atrasa *n* milisegundos

  `delay_ms(n)`

- **DELAY_CYCLES():** atrasa *n* ciclos de máquina

  `delay_cycles(n)`

7.5 - Controle do processador

- **SLEEP():** coloca PIC em modo sleep

```
sleep();
```

- **RESET_CPU():** reinicia o processador
- **ENABLE_INTERRUPTS():** habilita uma interrupção

```
enable_interrupts(GLOBAL |
INT_TIMER0 | INT_RB);
```

- **DISABLE_INTERRUPTS():** desabilita uma interrupção

```
disable(GLOBAL | INT_TIMER0 |
INT_RB);
```

- **EXT_INT_EDGE():** seleciona a borda de sensibilidade de interrupção externa

```
ext_int_edge(0 , L_TO_H);
/* seleciona a borda de subida da interrupção 0 */
```

Outras opções são: H_TO_L e L_TO_H. O primeiro argumento seleciona uma das fontes externas de interrupção.

- **READ_BANK ou WRITE_BANK():** leitura/escrita de uma posição de memória RAM

```
x = read_bank(1,2) /*lê a posição de memória de programa no banco 1 com 2 deslocamentos
```

7.6 – Matemáticas: #include <math.h>

- **ABS():** retorna valor absoluto de um número

```
valor = abs(-10);   //10
```

Outras opções: LABS(), FABS(), MODF(), FMOD()

- **SIN(),ASIN(),SINH() OU COS(),ACOS(),COSH()**

```
valor = sin(3.14);
```

- **TAN(), ATAN(), TANH()**

```
valor = tan(2.0);
```

- **CEIL() e FLOOR:** arredonda para número inteiro

```
valor = ceil(9.89); //10
valor = floor(9.89); //9
```

- **EXP(), LDEXP(), LOG(), POW(), SQRT():** potenciação

```
valor = exp(2); //faz e^x
valor = ldexp(2,4); //faz 2x2^exp = 2x2^4
valor = log(10); //log natural de 10 = 2.3
valor = log10(10); //1
valor = pow(3, 2); //faz 3^2 = 9
valor = sqrt(4); //4^1/2 = 2
```

7.7 - Manipulação RAM e EEPROM/flash interna:

- **MEMCPY():** copia n bytes de posição de memória para outra

```
memcpy (&destino, &origem, sizeof(destino));
```

- **MEMSET():** inicializa uma ou mais posições de memória

```
memset(&estrutura, 10, sizeof(estrutura));
```

- **READ_EEPROM():** lê by de um endereço na EEPROM/flash

```
int dado;
dado = read_eeprom(3); // lê quarto endereço
```

- **WRITE_EEPROM():** escreve dado em endereço EEPROM

```
write_eeprom(3, 5); //valor 5 endereço 3
```

- **WRITE_PROGRAM_EEPROM() e READ_PROGRAM_EEPROM():** escreve dado em memória programa

7.8 - Manipulação de caracteres:

- **TOLOWER(), TOUPPER:** converte maiúsculo/minúsculo:

```
switch (TOUPPER(GETC()))
{
      case 'A': break;
      case 'B': break;
}
```

- **ATOI(), ATOL(), ATOI32(), ATOF():** converte texto em números. Exemplo:

```
char texto = {"123"};
int valor;
valor = atoi(texto);
```

- **ISAMOUNG():** verifica se caractere pertence a string

```
confere=isamoung(s,string)
```

- **ISALNUM(), ISALPHA(), ISDIGIT(), ISLOWER(), ISSPACE(), ISUPPER(), ISXDIGIT(), ISCNTRL(), ISGRAPH(), ISPRINT(), ISPUNCT():** confere a natureza dos caracteres:

```
boolean confere;
confere = isalnum(x);//X é 0-9, 'A'..'Z',  ou 'a'..'z`?
confere = isalpha(x); // X é 'A'..'Z' ou 'a'..'z`?
confere = isdigit(x); //X é '0'..'9`?
confere = islower(x); //X é 'a'..'z' ?
confere = isupper(x); //X é 'A'..'Z'?
confere = isspace(x); //X é tecla espaço?
confere = isxdigit(x);//X é '0`-'9', 'A`-'F', ou 'a`-
'f?
```

- **STRLEN():** número de caracteres de uma string

```
char texto = {"UFU PATOS"};
int tam;
tam = strlen(texto);
```

- **STRCPY() e STRNCPY ():** copia conteúdo de uma string para outra

```
char nome [4];
strcpy (nome, "UFU");
```

- **STRLWR():** converte em minúsculo caracteres de string
- **STRTOK():** retorna ponteiro para primeira ocorrência de uma palavra em uma string