Revisão de Linguagem C

Jun Okamoto Jr.

Referências

- Kernighan, Brian W. and Ritchie, Dennis M. *The C Programming Language (ANSI C)*; Prentice Hall; 2ª Edição; 1988
- Microchip Technology Inc. *MPLAB XC8 C Compiler User's Guide*; 2015

Tipos de variáveis

- Em C, o tipo de variável tem a ver com o seu tamanho e não com o seu conteúdo
- O tamanho do tipo de variável tem a ver com o compilador

Tipos de variáveis em XC8

Tipo	Standard ISO types	Tamanho (bits)	Tipo aritmético
bit		1	unsigned integer
signed char	int8_t	8	signed integer
unsigned char	uint8_t	8	unsigned integer
signed short		16	signed integer
unsigned short		16	unsigned integer
signed int	int16_t	16	signed integer
unsigned int	uint16_t	16	unsigned integer
signed short long	int24_t	24	signed integer
unsigned short long	uint24_t	24	unsigned integer
signed long	int32_t	32	signed integer
unsigned long	uint32_t	32	unsigned integer
float		24 ou 32	real
double		24 ou 32	real

OBS: O tipo default está sublinhado. ISO Types em <stdint.h>.

Tipos de variáveis em XC 8

- Exemplo

```
// C types

char a; // variável de 8-bits sem sinal, pode armazenar valores
        // entre 0 a 255

int b;  // variável de 16-bits com sinal, pode armazenar valores
        // entre -32.768 e 32.767 (em complemento de 2)

long c; // variável de 32-bits com sinal, pode armazenar valores
        // entre -2.147.483.648 e 2.147.483.647 (em complemento de 2)

// ISO standard types

int8_t x;   //variável de 8-bits com sinal, pode armazenar valores
            // entre -128 a 127

uint16_t y; // variável de 16-bits sem sinal, pode armazenar valores
            // entre 0 e 65.535

uint32_t z; // variável de 32-bits com sinal, pode armazenar valores
            // entre 0 e 4.294.967.295
```

Tipos de variáveis em XC8

- Qualificadores

Qualificador	Descrição	Exemplo
const	indica que o objeto é somente para leitura, é armazenado na área de flash	`const int tableDef[] @ 0x100 = {0, 1, 2, 3, 4};`
volatile	indica que não existem garantias de que o objeto retenha seu valor entre sucessivos acessos	`volatile static unsigned int TACTL;`
static	variável é alocada em uma posição fixa da memória, são locais em relação ao escopo onde são declaradas	
persistent	indica que a variável mantém seu valor entre Resets. É armazenada em área separada das outras variáveis	`void test(void){ static persistent int intvar; /* must be static */ // ... }`

Radicais numéricos em C

Radix	Format	Example
binary	`0b` *number* or `0B` *number*	0b10011010
octal	`0` *number*	0763
decimal	*number*	129
hexadecimal	`0x` *number* or `0X` *number*	0x2F

Radicais numéricos e valores armazenados

- Exemplo

```
char x, y, z;    // variáveis de 8-bits sem sinal (0 e 255)

x = 129;         // variável contém 10000001 em binário

y = 0x81;        // variável contém 10000001 em binário

z = 0b10000001;  // variável contém 10000001 em binário


                 // (x == y) é True ou False ?
                 // (y == z) é True ou False ?
                 // (x == z) é True ou False ?
```

Radicais numéricos e valores armazenados

- Exemplo

```
char x, y, z;    // variáveis de 8-bits sem sinal (0 e 255)

x = 129;         // variável contém 10000001 em binário

y = 0x81;        // variável contém 10000001 em binário

z = 0b10000001;  // variável contém 10000001 em binário


                 // (x == y) é True ou False ?
                 // (y == z) é True ou False ?
                 // (x == z) é True ou False ?
```

Todos avaliam como TRUE!

Códigos para Texto

- ASCII
  - American Standard Code for Information Interchange
  - Publicado em 1963, revisto em 1967
  - Código com 7-bits para representar texto em computadores e equipamentos de comunicação (telégrafo na época)

ASCII

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Outros códigos

- Windows-1252 →
  - superset de ISO 8859-1
  - Também conhecido como ANSI
- UNICODE
- UTF-8
  - código de largura variável de 1 a 8 bytes
  - Está se tornando padrão hoje em dia

```
                ................
                ................
                 !"#$%&'()*+,-./
                0123456789:;<=>?
                @ABCDEFGHIJKLMNO
                PQRSTUVWXYZ[\]^_
                `abcdefghijklmno
                pqrstuvwxyz{|}~.
                €.‚ƒ„…†‡ˆ‰Š‹Œ.Ž.
                .‘’“”•–—˜™š›œ.žŸ
                 ¡¢£¤¥¦§¨©ª«¬-®¯
                °±²³´µ¶·¸¹º»¼½¾¿
                ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
                ÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
                àáâãäåæçèéêëìíîï
                ðñòóôõö÷øùúûüýþÿ
```

Texto em C

```
char x, y, z;    // variáveis de 8-bits sem sinal (0 e 255)

x = 'A';         // x contém código ASCII para a letra A

y = 0x41;        // y contém código ASCII para a letra A

z = 65;          // z contém código ASCII para a letra A


                 // (x == y) é True ou False ?
                 // (y == z) é True ou False ?
                 // (x == z) é True ou False ?
```

Texto em C

```
char x, y, z;    // variáveis de 8-bits sem sinal (0 e 255)

x = 'A';         // x contém código ASCII para a letra A

y = 0x41;        // y contém código ASCII para a letra A

z = 65;          // z contém código ASCII para a letra A


                 // (x == y) é True ou False ?
                 // (y == z) é True ou False ?
                 // (x == z) é True ou False ?
```

Todos avaliam como TRUE!

Texto em C

```
char z;  // variável de 8-bits (1 byte)

...

z = "A"; // Está correto?
```

Texto em C

```
char z;  // variável de 8-bits (1 byte)

...

z = “A”; // Está correto?
```

NÃO! “A” representa um string e necessita de 2 bytes para ser armazenado em C

Strings em C

- String em C é um vetor de char terminado por zero
- Zero significa

```
0    // em decimal

0x0  // em hexadecimal

0b0  // em binário

‘\0’ // em escape sequence (caracter nulo)
```

Strings em C

- Assim o espaço para armazenar um string deve ser o número de caracteres mais um

```
char myString[9]; // armazena 8 caracteres e um
                  // terminador
```

String em C

- Como criar um string em C

```
sprintf(myString, “vel = %d”, vel);

// o uso é igual ao printf(), mas o string é
//  armazenado num vetor
```

Escape Sequences em C (também C++)

- Representam um caracter ASCII por meio de uma combinação com o caracter ‘\’ seguido por algum caracter

```
\n - New line                \ooo - representação em octal

\r - Carriage Return         \xdd - representação em hexadecimal

\t - Horizontal Tab

\\ - Backslash

\‘ - Single Quotation Mark

\” - Double Quotation Mark
```

Operadores

- Aritméticos: `+ - * / %`
- Lógicos: `> >= < <= == != && ||`

Operadores

- Incremento e decremento: `++` `--`

Operadores

- Incremento e decremento: `++` `--`
  - Pré-incremento

```
y = ++x; // incrementa antes e usa o
         // valor depois
```

Operadores

- Incremento e decremento: `++` `--`
  - Pré-incremento

```
y = ++x; // incrementa antes e usa o
         // valor depois
```

  - Pós-incremento

```
y = x++; // usa o valor primeiro e
         // incrementa depois
```

Operadores

- Incremento e decremento: `++` `--`
  - Pré-incremento

```
y = ++x; // incrementa antes e usa o
         // valor depois
```

  - Pós-incremento

```
y = x++; // usa o valor primeiro e
         // incrementa depois
```


Operadores

- Incremento e decremento: `++` `--`
  - Pré-incremento

```
y = ++x; // incrementa antes e usa o
         // valor depois
```

  - Pós-incremento

```
y = x++; // usa o valor primeiro e
         // incrementa depois
```

Operadores

- Incremento e decremento: `++` `--`
  - Pré-incremento

```
y = ++x; // incrementa antes e usa o
         // valor depois
```

  - Pós-incremento

```
y = x++; // usa o valor primeiro e
         // incrementa depois
```

Operadores

- Incremento e decremento: `++` `--`
  - Pré-incremento

```
y = ++x; // incrementa antes e usa o
         // valor depois
```

  - Pós-incremento

```
y = x++; // usa o valor primeiro e
         // incrementa depois
```

Operadores

- Exemplo de pós e pré-incremento

```
char x, y;        // variáveis de 8-bits sem sinal (0 e 255)
char myString[8]; // string contendo "PMR2415", não importa como
                  // neste exemplo
int i;            // contador

...

i = 2;

x = myString[i++];   // x = ?, i antes ?, i depois ?

y = myString[++i];   // y = ?, i antes ?, i depois ?
```

Operadores

- Exemplo de pós e pré-incremento

```
char x, y;        // variáveis de 8-bits sem sinal (0 e 255)
char myString[8]; // string contendo "PMR2415", não importa como
                  // neste exemplo
int i;            // contador

...

i = 2;

x = myString[i++];   // x = ?, i antes ?, i depois ?

y = myString[++i];   // y = ?, i antes ?, i depois ?
```

x = 'R',
i antes = 2,
i depois = 3

Operadores

- Exemplo de pós e pré-incremento

```
char x, y;         // variáveis de 8-bits sem sinal (0 e 255)
char myString[8]; // string contendo "PMR2415", não importa como
                   // neste exemplo
int i;             // contador

...

i = 2;

x = myString[i++];   // x = ?, i antes ?, i depois ?

y = myString[++i];   // y = ?, i antes ?, i depois ?
```

```
x = 'R',
i antes = 2,
i depois = 3
```

```
y = '4',
i antes = 3,
i depois = 4
```

Operadores

- De Bits:
  - `&` AND bit a bit
  - `|` OR bit a bit
  - `^` XOR bit a bit
  - `<<` desloca bits para a esquerda
  - `>>` desloca bits para a direita
  - `~` complemento de 1

Operadores

- De Bits:
  - `&` AND bit a bit
  - `|` OR bit a bit
  - `^` XOR bit a bit
  - `<<` desloca bits para a esquerda
  - `>>` desloca bits para a direita
  - `~` complemento de 1

- Não Confundir:
  - `&` com `&&` (AND Lógico)
  - `|` com `||` (OR Lógico)

Operadores

- Exemplos de operação AND e OR binários

```
x =                0b10011101;
y =                0b01001000;
z = x & y; //      ----------
           //  z = 0b00001000
```

```
x =                0b10011101;
y =                0b01010000;
z = x | y; //      ----------
           //  z = 0b11011101
```

Operadores

- Exemplos de operação AND e OR binários

```
x =                0b10011101;
y =                0b01001000;
z = x & y; //      ----------
           //  z = 0b00001000
```

x	y	&
0	0	0
0	1	0
1	0	0
1	1	1

```
x =                0b10011101;
y =                0b01010000;
z = x | y; //      ----------
           //  z = 0b11011101
```

Operadores

- Exemplos de operação AND e OR binários

```
x =                0b10011101;
y =                0b01001000;
z = x & y; //      ----------
           //  z = 0b00001000
```

x	y	&
0	0	0
0	1	0
1	0	0
1	1	1

```
x =                0b10011101;
y =                0b01010000;
z = x | y; //      ----------
           //  z = 0b11011101
```

x	y	|
0	0	0
0	1	1
1	0	1
1	1	1

Operadores

- Exemplos de operadores de bits para deslocamento de bits

```
x =                  0b100111101;
y = x << 3;   // y = 0b111101000  (deslocamento de 3 bits para
                                   esquerda)

x =                  0b100111101;
y = y >> 4;   // y = 0b000011001  (deslocamento de 4 bits para
                                   direita)
```

Operadores

- Atribuição: `=` *`op=`*

  onde *`op`* = {`+ - * / % << >> & | ^`}

- *`expr1 op= expr2`* é equivalente a

  `expr1` = (*`expr1`*) *`op`* (*`expr2`*)

Operadores

- Exemplos de atribuição

```
i = 2;              // ok
i += 2;             // i = i + 2
x *= y + 1;         // x = x * (y + 1) e não x = x * y + 1
```

Operadores

- AND e OR como máscaras

```
x &= 0b01110010;     // equivalente a x = x & 0b01110010
y |= 0b01110010;     // equivalente a y = y | 0b01110010
```

Operadores

- AND e OR como máscaras

```
x &= 0b01110010;     // equivalente a x = x & 0b01110010
y |= 0b01110010;     // equivalente a y = y | 0b01110010
```

x	y	&
0	0	0
0	1	0
1	0	0
1	1	1

Operadores

- AND e OR como máscaras

```
x &= 0b01110010;     // equivalente a x = x & 0b01110010
y |= 0b01110010;     // equivalente a y = y | 0b01110010
```

x	y	&
0	0	0
0	1	0
1	0	0
1	1	1

coloca zeros nos bits 0, 2, 3 e 7

Operadores

- AND e OR como máscaras

```
x &= 0b01110010;     // equivalente a x = x & 0b01110010
y |= 0b01110010;     // equivalente a y = y | 0b01110010
```

x	y	|
0	0	0
0	1	1
1	0	1
1	1	1

Operadores

- AND e OR como máscaras

```
x &= 0b01110010;     // equivalente a x = x & 0b01110010
y |= 0b01110010;     // equivalente a y = y | 0b01110010
```

x	y	|
0	0	0
0	1	1
1	0	1
1	1	1

coloca uns nos bits 1, 4, 5 e 6

Operadores

- Expressões condicionais:

expr1 ? expr2 : expr3

- Exemplo

```
if (a > b)
  z = a;
else
  z = b;


z = (a > b) ? a : b; // z = max(a, b)
```

Operadores

- Precedência e associatividade

Operadores	Associatividade
( ) [ ] -> .	esquerda para direita
! ~ ++ -- + - * & (type) sizeoff	direita para esquerda
* / %	esquerda para direita
+ -	esquerda para direita
<< >>	esquerda para direita
< <= > >=	esquerda para direita
== !=	esquerda para direita
&	esquerda para direita
^	esquerda para direita
|	esquerda para direita
&&	esquerda para direita
||	esquerda para direita
?:	direita para esquerda
= += -= *= /= %= &= ^= |= <<= >>=	direita para esquerda
,	esquerda para direita

Ponteiros

- Ponteiros são variáveis que armazenam endereços de variáveis
- Declaração e uso:

```
char *p;  // p é um ponteiro para uma variável
          // do tipo char
char c;   // c é uma variável do tipo char

p = &c;   // o operador & dá o endereço de c

c = *p;   // o operador * dá o conteúdo apontado
          // por p
```

Ponteiros

- Exemplo

```
int x = 1, y = 2, z[10];
int *ip;                // ip é um ponteiro para int
                        // a declaração int *ip é mnemônica e
                        // significa que a expressão *ip é um int
int *iq;                // outro ponteiro para int

ip = &x;                // ip aponta para x (contém o endereço de x)
y = *ip;                // y agora é 1
*ip = 0                 // x agora é 0
ip = &z[0]              // ip aponta agora para z[0]
iq = ip;                // iq aponta para a mesma coisa que ip
                        // (ponteiros são variáveis)
```

Ponteiros

- O operador * mais o ponteiro podem ser usados no lugar da variável

```
int x, y;
int *ip;                // ip é um ponteiro para int

ip = &x;                // ip aponta para a variável x
*ip = *ip + 10;         // incrementa x de 10
y = *ip + 1;            // y contém o valor de x mais 1
*ip += 1;               // o valor de x é incrementado de 1
++*ip;                  // equivalente à linha acima
(*ip)++;                // equivalente à linha acima
                        // os ()s são necessários, sem eles a
                        // expressão incrementaria ip e não x
```

Ponteiros

- Como ponteiros são variáveis, operações aritméticas com ponteiros são válidas

```
int a[10];
char b[10];

int *ip;                // ip é um ponteiro para int
char *iq;               // iq é um ponteiro para char

ip = &a[0];             // ip aponta para a[0]
++ip;                   // ip aponta para a[1]

iq = &b[4] ;            // iq aponta para b[4]
iq -= 2                 // iq aponta para b[2]
```

Ponteiros e Arrays

- Nomes de arrays são ponteiros, mas não são variáveis

```
int i = 1, x, a[10];
int *ip;                // ip é um ponteiro para int

ip = &a[0];             // ip aponta para a[0]
ip = a;                 // é o mesmo que acima
ip++;                   // ip aponta para a[1]

x = a[i];               // x contém o valor de a[1]
x = *(a + i);           // é o mesmo que acima
x = *(pa + i);          // é o mesmo que acima

a = ip;                 // NÃO É VÁLIDO: nome de array
                        // não é variável
a++;                    // também NÃO É VÁLIDO
```

Funções

- Passagem de parâmetros
  - Em C, todos os parâmetros são passados “por valor” (não modifica o valor da variável passada como parâmetro)
  - Para se modificar o valor de um parâmetro passa-se um ponteiro para a variável

Funções

- Exemplo de passagem de parâmetro

```
/* power: base elevada à n-ésima potência; n >= 0 */
int power(int base, int n) {
  int p;
  for (p = 1; n > 0; --n) p = p * base;
  return p;
}
```

Funções

- Exemplo de passagem de parâmetro

```
/* power: base elevada à n-ésima potência; n >= 0 */
int power(int base, int n) {
  int p;
  for (p = 1; n > 0; --n) p = p * base;
  return p;
}
```

n é usado como variável na função, mas seu valor original não é modificado

Funções

- Exemplo de alteração de valor de parâmetro

```
/* strcpy: copia string t para string s */
void strcpy(char *s, char *t) {
  while ((*s = *t) != '\0') {
    s++;
    t++;
  }
}
```

Struct

- Define grupo de variáveis

```
struct [structure tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

- Exemplo (ver `always.h`)

```
struct eight_bits {
  unsigned char bit0 :1;
  unsigned char bit1 :1;
  unsigned char bit2 :1;
  unsigned char bit3 :1;
  unsigned char bit4 :1;
  unsigned char bit5 :1;
  unsigned char bit6 :1;
  unsigned char bit7 :1;
};
```

número de bits

- Uso

```
#include "always.h"

// declaração
struct eight_bits part;
bit LED;

// uso
LED = part.bit3;
```

Union

- Permite armazenar diferentes tipos de variáveis na mesma posição de memória

```
union [union tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

- Exemplo (ver `always.h`)

```
union chartype {
 unsigned char byte;
 struct eight_bits part;
};
```

- Uso

```
#include "always.h"

// declaração
union chartype lineSensor;
bit sensor1;

// uso
lineSensor.byte = sensorLine_read();
sensor1 = lineSensor.part.bit0;
```

Preprocessador C

- Include

```
#include <xc.h>
#include <stdio.h>
#include "always.h"
#include "delay.h"
```

- Define

```
#define INPUT 1
#define OUTPUT 0
#define TRUE 1
#define FALSE 0
#define HIGH 1
#define LOW 0
#define hi 1
#define lo 0
#define OFF 0
#define ON 1
```

- If not defined

```
#ifndef LCD8X2_H
#define LCD8X2_H
.
.
.
#endif
```

- Pragma

```
#pragma config PWRTE = OFF
#pragma config MCLRE = ON
#pragma config CP = OFF
#pragma config CPD = OFF
#pragma config BOREN =OFF
```

Preprocessador C

- Include

```
#include <xc.h>
#include <stdio.h>
#include "always.h"
#include "delay.h"
```

Inclui do diretório do path

- Define

```
#define INPUT 1
#define OUTPUT 0
#define TRUE 1
#define FALSE 0
#define HIGH 1
#define LOW 0
#define hi 1
#define lo 0
#define OFF 0
#define ON 1
```

- If not defined

```
#ifndef LCD8X2_H
#define LCD8X2_H
.
.
.
#endif
```

- Pragma

```
#pragma config PWRTE = OFF
#pragma config MCLRE = ON
#pragma config CP = OFF
#pragma config CPD = OFF
#pragma config BOREN =OFF
```

Preprocessador C

- Include

```
#include <xc.h>
#include <stdio.h>
#include "always.h"
#include "delay.h"
```

Inclui do diretório do projeto

- Define

```
#define INPUT 1
#define OUTPUT 0
#define TRUE 1
#define FALSE 0
#define HIGH 1
#define LOW 0
#define hi 1
#define lo 0
#define OFF 0
#define ON 1
```

- If not defined

```
#ifndef LCD8X2_H
#define LCD8X2_H
.
.
.
#endif
```

- Pragma

```
#pragma config PWRTE = OFF
#pragma config MCLRE = ON
#pragma config CP = OFF
#pragma config CPD = OFF
#pragma config BOREN =OFF
```

Preprocessador C

- Include

```
#include <xc.h>
#include <stdio.h>
#include "always.h"
#include "delay.h"
```

- Define

```
#define INPUT 1
#define OUTPUT 0
#define TRUE 1
#define FALSE 0
#define HIGH 1
#define LOW 0
#define hi 1
#define lo 0
#define OFF 0
#define ON 1
```

- If not defined

```
#ifndef LCD8X2_H
#define LCD8X2_H
.
.
.
#endif
```

- Pragma

```
#pragma config PWRTE = OFF
#pragma config MCLRE = ON
#pragma config CP = OFF
#pragma config CPD = OFF
#pragma config BOREN =OFF
```

Exercícios

-
```
long *p1;
long x[8];
p1 = x;
```

  - Endereços de 16-bits, dados de 8-bits
  - Tipo long de 32-bits
  - Se `p1` contém o valor `0xb1fe`, `p1 + 4` conterá que valor?

Exercícios

-
```
x = 'A';   // código ASCII 0x41
y = 'B';   // código ASCII 0x42
c = x + y; // qual é o valor de c
           // em decimal?
```

-
```
int x, y;
x = 'A'; // qual é o conteúdo de x
         // em binário?
y = (x >> 2) << 10; // qual é
                    // valor de y?
```

Exercícios

```
x = 0x25;
y = x & 0b11111011;  // qual é o valor de y?
y = x && 0b11111011; // qual é o valor de y?
```

```
char y[10]; int x;
x = 'A';
sprintf(y, "%d", ++x); // qual é o
                       // valor de y?
sprintf(y, "%d", x++); // qual é o
                       // valor de y?
// Qual é o valor de x após a execução das
// linhas anteriores?
```

Exercícios

```
char *p1; int x, y;
char a[10];
x = 10;
sprintf(a, "%d", x);
p1 = a;
y = *p1 + 10; // qual é o valor de y?
y = *(++p1) + 10; // y?
y = ++*p1 + 10;   // y?
```