ICMC USP

1.semestre/2011

Introdução à Ciência da Computação

scc-120

Aula : Variáveis em C

Profa. Roseli Romero

mailto: rafrance@icmc.sc.usp.br

Slides cedidos pela profa. Renata Fortes

Variáveis

- variáveis em um programa C estão associadas a posições de memória que armazenam informações
- nomes de variáveis podem ter vários caracteres

  em C, apenas os 31 primeiros caracteres são considerados
- o primeiro caracter tem que ser uma letra ou *underscore* "_"
- o restante do nome pode conter letras, dígitos e sublinhados

exercício

- Escolha a opção que inclui somente nomes válidos para variáveis na linguagem C.

  a). If, a_b_2, H789, _yes

  b). i, j, int, obs

  c). 9xy, a36, x*y, --j

  d). 2_ou_1, \fim, *h, j

  e). Nenhuma das opções anteriores

Variáveis

- Exemplos de nomes de variáveis:

Corretos	Incorretos
Contador	1contador
Teste23	oi!gente
Alto_Paraiso	Alto..Paraíso
__sizeint	_size-int

Variáveis

- Palavras-chave de C não podem ser utilizadas como nome de variáveis: ***int***, ***for***, ***while***, etc...
- C é *case-sensitive*:

  contador ≠ Contador ≠ CONTADOR

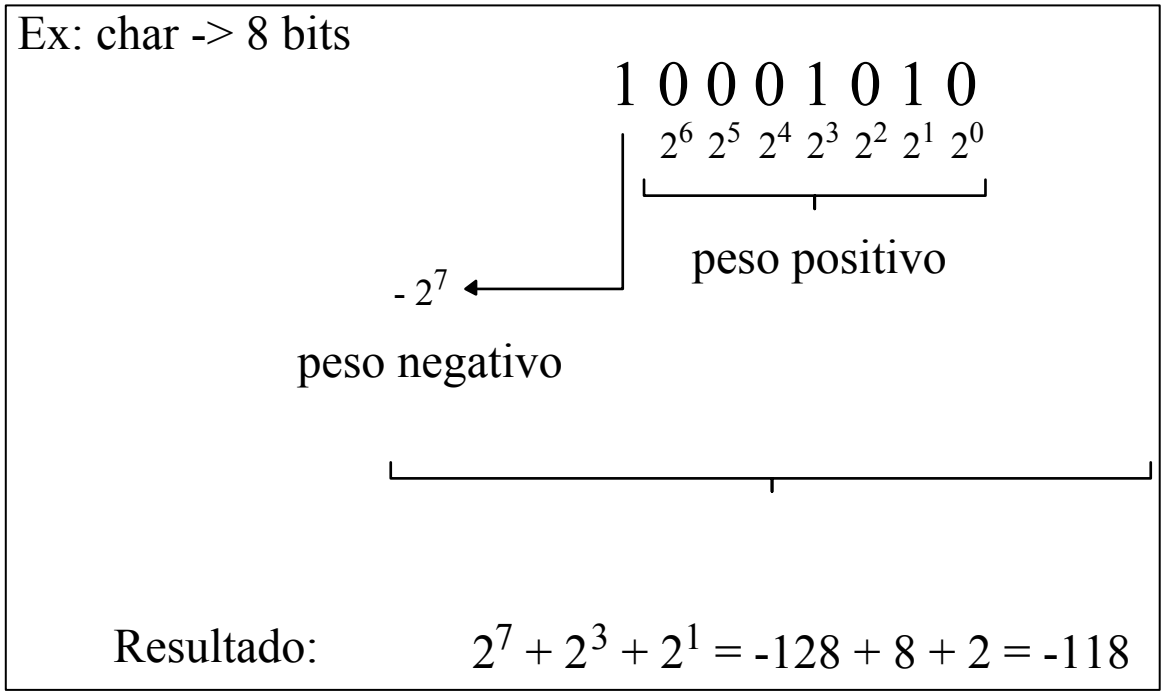
Tipos de Dados

- O *tipo* de uma variável define os valores que ela pode assumir e as operações que podem ser realizadas com ela
- Ex:
  - variáveis tipo *int* recebem apenas valores inteiros
  - variáveis tipo *float* armazenam apenas valores reais

Tipos de dados básicos em C

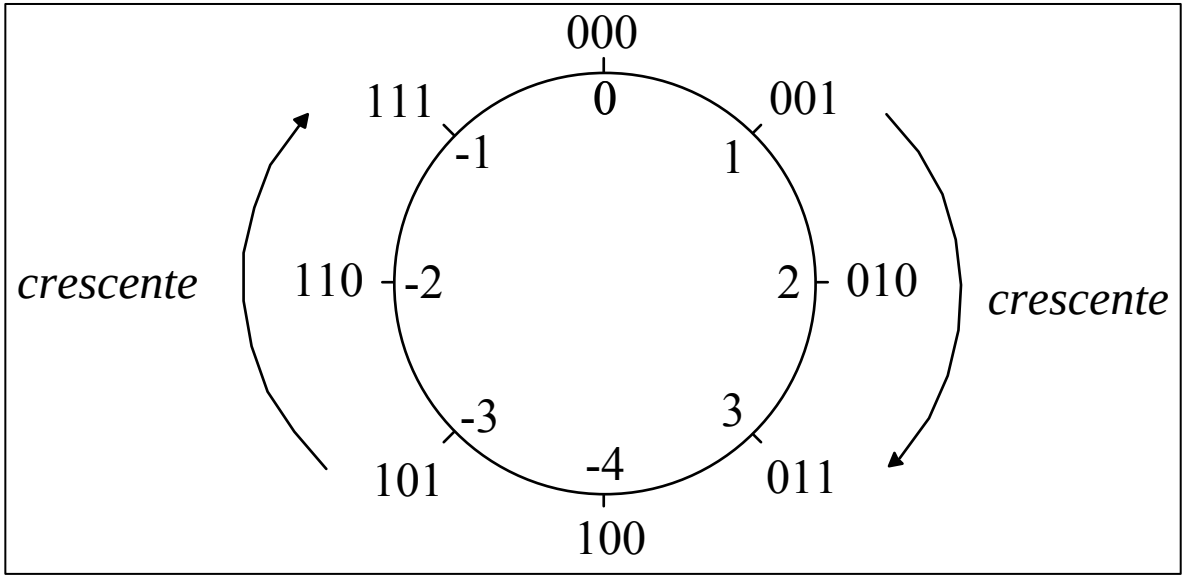
- char : um byte que armazena o código de um caracter do conjunto de caracteres local
- int : um inteiro cujo tamanho depende do processador, tipicamente 16 ou 32 bits
- float : um número real com precisão simples
- double : um número real com precisão dupla

Representação de Inteiros



Complemento de 2

Código Binário - Complemento de 2



Modificadores de Tipos

- modificadores alteram algumas características dos tipos básicos para adequa-los a necessidades específicas
- Modificadores:
  - signed: indica número com sinal (inteiros e caracteres)
  - unsigned: número apenas positivo (inteiros e caracteres)
  - long: aumenta abrangência (inteiros e reais)
  - short: reduz a abrangência (inteiros)

Abrangência dos Dados: 16 bits

Tipo	Tamanho(bytes)	Abrangência		
char	1	-128	a	127
unsigned char	1	0	a	255
int	2	-32768	a	32767
unsigned int	2	0	a	65535
short int	2	-32768	a	32767
long int	4	-2.147.483.648	a	2.147.483.647
unsigned long	4	0	a	4.294.967.295
float	4	$-3{,}4 \cdot 10^{38}$	a	$3{,}4 \cdot 10^{38}$
double	8	$-1{,}7 \cdot 10^{308}$	a	$1{,}7 \cdot 10^{-308}$
long double	10	$-3{,}4 \cdot 10^{4932}$	a	$3{,}4 \cdot 10^{4932}$

Abrangência dos Dados: 32 bits

Tipo	Tamanho(bytes)			Abrangência
char	1	-128	a	127
unsigned char	1	0	a	255
int	4	-2.147.483.648	a	2.147.483.647
unsigned int	4	0	a	4.294.967.295
short int	2	-32768	a	32767
long int	4	-2.147.483.648	a	2.147.483.647
unsigned long	4	0	a	4.294.967.295
float	4	$3,4 \cdot 10^{-38}$	a	$3,4 \cdot 10^{38}$
double	8	$1,7 \cdot 10^{-308}$	a	$1,7 \cdot 10^{-308}$
long double	10	$3,4 \cdot 10^{-4932}$	a	$3,4 \cdot 10^{4932}$

Constantes

• Constantes são valores fixos que não podem ser modificados pelo programa

Tipo		Exemplos
char	->	‘a’ ‘\n’ ‘9’
int	->	123 1 1000 -23
long int	->	35000L -45L
short int	->	10 -12 90
unsigned int	->	1000U 234U 4365U
float	->	123.45F 3.1415e-10F
double	->	123.45 -0.91254

Constantes char

Barra invertida

- \a bip
- \b backspace
- \n newline
- \t tab horizontal
- \’ apóstrofe
- \” aspa
- \\ backslash
- \f form feed

Declaração de Variáveis

- A declaração de uma variável segue o modelo:

  TIPO_VARIÁVEL lista_de_variaveis;

- Ex:

```
int x, y, z;
float f;
unsigned int u;
long double df;
char c = 'A';                    /* variavel definida e iniciada */
char s[ ] = "vetor de caracteres";
```

Atribuição de Variáveis

- Forma geral:

  variavel = expressão ou constante

Múltiplas atribuições

  – C permite a atribuição de mais de uma variável em um mesmo comando:

```
x = y = z = 0;
```

Conversões de Tipos na Atribuição

- Quando uma variável de um tipo é atribuida a uma de outro tipo, o compilador automaticamente converte o tipo da variável a direita de "=" para o tipo da variável a esquerda de "="

- Ex:

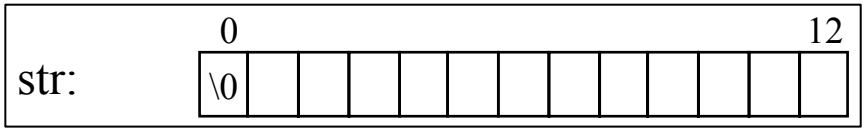
```
int x; char ch; float f;
ch = x;      /* ch recebe 8 bits menos significativos de x */
x = f;       /* x recebe parte inteira de f */
f = ch;      /* f recebe valor 8 bits convertido para real */
f = x;       /* idem para inteiro x */
```

Strings

- strings são sequências de caracteres adjacentes na memória. O caracter '\0' (= valor inteiro 0) indica o fim da sequência

Ex: `char str[13];`

  – define um string de nome "str" e reserva para ele um espaço de 13 (12 + '\0') bytes na memória

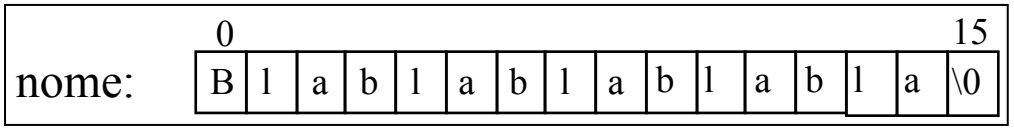


Strings

Ex:

```
char nome[16] = “Blablablablabla”;
```

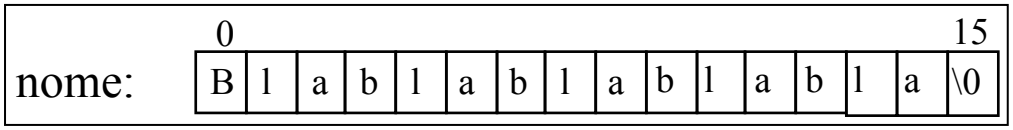
- define um string de nome “nome”, reserva para ele um espaço de memória de 16 (15 + ‘\0’) bytes e inicia-o com o texto indicado



Strings

- os caracteres podem ser individualmente acessados por indexação:
- Ex:

```
nome[0] = 'B';
nome[10] = 'l'
```



Exercícios

- O trecho de programa a seguir é

```
main( )
{
    char condicao;
    condicao = 'D';
    int i = 1;
}
```

a). Válido na linguagem C

b). Não válido na linguagem C

- Em C, "t" e 't' representam a mesma constante.

a). Verdadeiro

b). Falso

Operações com *Strings*

- atribuição: não se pode atribuir um string a outro string:

```
str = name; /* erro */
```

- o header "string.h" contém uma série de funções para manipulação de strings

strlen(str) **retorna o tamanho de *str***
strcpy(dest, fonte) **copia *fonte* em *dest***
strcat(dest, fonte) **concatena *fonte* no fim de *dest***

Operações com *Strings*

- Ex:

char fonte[] = “Bom”;
char dest[] = “ dia!”;

strlen(fonte) => retorna 3
strlen(dest) => retorna 5
strcat(fonte, dest) => “Bom dia!”
strcpy(dest, fonte) => “Bom”

Entrada e saída de strings

- A função *gets(s)* lê um string do teclado e coloca-o em *s*
- Ex:

```
#include <stdio.h>
void main () {
    char nome[80];
    printf (”Entre nome aluno: ");
    gets (nome);
    printf ("\O nome eh: %s",nome);
}
```

Exercícios

1. Programar a função *int verifica_digito(char c)*, que retorna 1 se *c* é um dígito (entre ‘0’ e ‘9’) e 0 caso contrário.
2. Implementar *int tamanho(char s[ ])* que retorna o tamanho do *string* s.
3. Fazer um programa que procura a ocorrência de um caracter *c* em um string *s* e imprime “Achou!” caso *c* apareça em *s* e “Nope!” caso contrário.

Strlen( )

```
int strlen(char s[])
{
  int i;

  for (i = 0; s[i] != 0; i++);
  return(i);
}
```

Escopo de Variáveis

- Escopo define onde e quando uma variável pode ser usada em um programa
- variável declarada fora das funções tem escopo de arquivo:

```
#include <stdio.h>
int i;                  /* variavel global */
                        /* visivel em todo arquivo */
void incr_i() { i++;}
...
void main() { printf("%d", i);}
```

Escopo de Variáveis

- Escopo de bloco: é visível apenas no interior do bloco

```
...

if (teste == TRUE) {
        int i;
          i = i+1;
          ...
        }
else { i = i - 1; /* erro: i não definida */
        ...
}
...
```

Escopo de Variáveis

- Escopo de função: variável declarada na lista de parâmetros da função ou definida dentro da função
- Ex:...

```
int minha_fun (int x, int y) {
int i, j;
/* x,y,i e j visíveis apenas dentro da função */
...
}
x = i+j;   /* erro: x,i e j não definidos */
```

Expressões

- expressões são compostas por:
  - operandos: a, b, x, Meu_dado, 2, ...
  - operadores: +, -, %, ...
  - pontuação: ( ), { }, “,”
- Ex:

  x

  14

  x + y

  (x + y)*z + w - v

Expressões

- expressões retornam um valor:

```
x = 5 + 4 /* retorna 9 */
```

  – esta expressão atribui 9 a x e retorna 9 como resultado da expressão

```
((x = 5 + 4) == 9) /* retorna true */
```

  – na expressão acima, além de atribuir 9 a *x*, o valor retornado é utilizado em uma comparação

Expressões

- a ordem em que uma expressão é avaliada depende da prioridade dos operadores e da pontuação
- expressões podem aparecer em diversos pontos de um programa
  - comandos /* x = y; */
  - parâmetros de funções /* sqrt(x + y); */
  - condições de teste /* if (x == y) */

Operadores

Unários:

+ : mais unário ou positivo	/* + x; */
- : menos unário ou negação	/* - x; */
! : NOT ou negação lógica	/* ! x; */
&: endereço	/* &x; */
*: conteúdo (ponteiros)	/* (*x); */
++: pré ou pós incremento	/* ++x ou x++ */
-- : pré ou pós decremento	/* -- x ou x -- */

Operadores

Binários:

+: adição de dois números	/* x + y */
- : subtração de dois números	/* x - y */
* : multiplicação de dois números	/* x * y */
/ : quociente de dois números	/* x / y */
%: resto da divisão:	/* x % y */

Operadores bit a bit

Operações bit-a-bit (vetores)

<<:	desloca à esquerda	/* x << 2 */
>>:	desloca à direita	/* x >>2 */
^ :	ou exclusivo	/* x ^ 0xF0 */
& :	E bit-a-bit	/* x & 0x07 */
| :	OU bit-a-bit	/* x | 0x80 */
~ :	Complementa bit-a-bit	/* ~ x */

Operações bit a bit

- Ex:

char x = 0xD5;

x = x & 0x0F

máscara de 0's

x	1	1	0	1	0	1	0	1
0x0F	0	0	0	0	1	1	1	1
x & 0x0F	0	0	0	0	0	1	0	1

Operações bit a bit

Ex:

x = x | 0x0F;	x	1	1	0	1	0	1	0	1
máscara de 1's	0x0F	0	0	0	0	1	1	1	1
	x | 0x0F	0	1	0	1	1	1	1	1

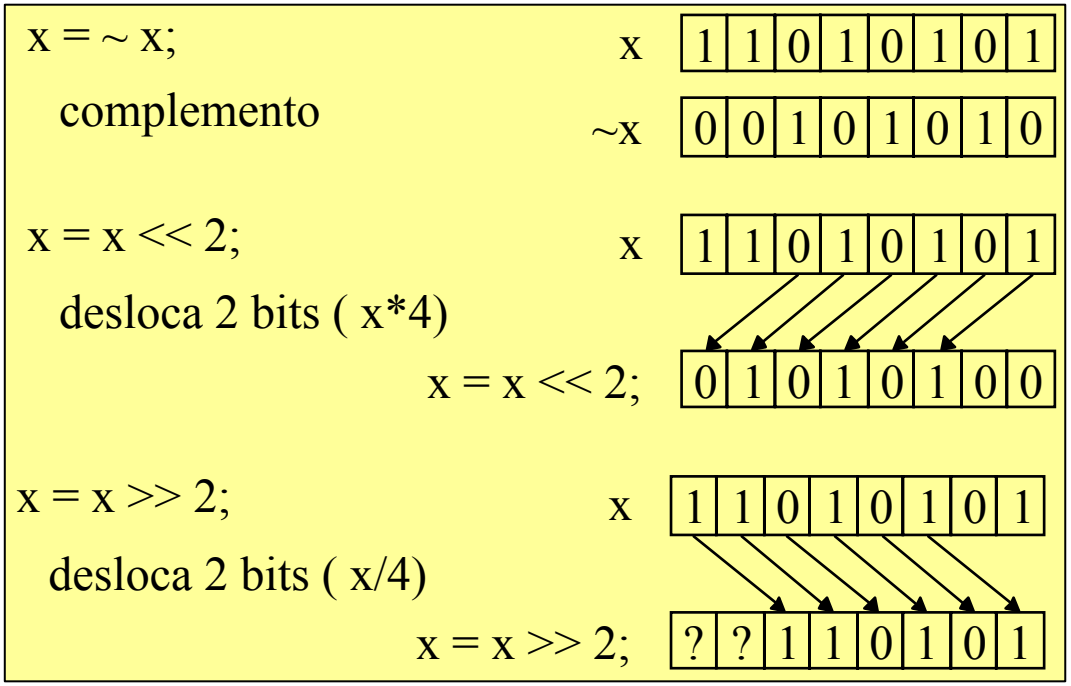
ou

x = x ^ 0x0F;	x	1	1	0	1	0	1	0	1
inversão controlada	0x0F	0	0	0	0	1	1	1	1
	x ^ 0x0F	1	1	0	1	1	0	1	0

ou exclusivo

Operações bit a bit

- Ex:



Atribuição

= : atribui	x = y;
+= : soma e atribui	x += y; <=> x = x + y;
-= : subtrai e atribui	x -= y; <=> x = x - y;
*= : multiplica e atribui	x *= y; <=> x = x * y;
/= : divide e atribui quociente	x /= y; <=> x = x / y;
%= : divide e atribui resto	x %= y; <=> x = x % y;
&= : E bit-a-bit e atribui	x &= y; <=> x = x & y;
|= : OU bit-a-bit e atribui	x |= y; <=> x = x | y;
<<= : shift left e atribui	x <<= y; <=> x = x<<y;

...

Atribuição

- Exemplos:

```
x = 10;
y = 5;

x += y;    /* x = 15 */
x -= 10;   /* x = 5 */
x *= y;    /* x = 35 */
```

Exercícios

1- Diga o resultado das variáveis x, y e z depois da seguinte seqüência de operações:

```
int x, y, z;
x=y=10;
z=++x;
x=-x;
y++;
x=x+y-(z--);
```

a). x = 11, y = 11, z = 11
b). x = -11, y = 11, z = 10
c). x = -10, y = 11, z = 10
d). x = -10, y = 10, z = 10
e). Nenhuma das opções anteriores

2- Diga o resultado das variáveis x, y e z depois da seguinte sequência de operações:

```
int x,y;
int a = 14, b = 3;
float z;
x = a/b;
y = a%b;
z = y/x;
```

a). x = 4.66666, y = 2, z = 0.4286
b). x = 5, y =2, z= 0.4
c). x = 5, y = 2, z = 0.
d). x = 4, y = 2, z = 0.5
e). x = 4, y =2, z = 0.
f). Nenhuma das opções anteriores

3- Quais os valores de a, b e c após a execução do código abaixo?

```
int a = 10, b = 20, c;
c = a+++b;
```

a). a = 11, b = 20, c =30
b). a = 10 , b = 21, c = 31
c). a = 11, b = 20, c = 31
d). a = 10, b = 21, c = 30
e). Nenhuma das opções anteriores

4- Qual o valor das variáveis v, x, y e z após a execução do seguinte
trecho de código

```
int v = 0, x = 1, y = 2, z = 3;
v += x+y;
x *= y = z + 1;
z %= v + v + v;
v += x += y += 2;
```

a). v=11, x=8, y=6, z=3
b). v=0, x=1, y=2, z=3
c). v=10, x=7, y=6, z=3
d). v=13, x=10, y=6, z=3
e). Nenhuma das opções anteriores

Operadores Relacionais

- Aplicados a variáveis que obedeçam a uma relação de ordem, retornam 1 (true) ou 0 (false)

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

Operadores Lógicos

- Operam com valores lógicos e retornam um valor lógico verdadeiro (1) ou falso (0)

Operador	Função	Exemplo
&&	AND (E)	(c >=‘0’ && c<=‘9’)
||	OR (OU)	(a==‘F’ || b!=32)
!	NOT (NÃO)	(!var)

Tabela Verdade

a	b	!a	!b	a && b	a || b
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

Exemplo: No trecho de programa abaixo o **if** será executado, pois o resultado da expressão lógica é verdadeiro:

```
int i = 5, j =7;
if ( (i > 3) && ( j <= 7)  && ( i != j) )  j++;
```

V AND V AND V = V

Exemplo: implementação do operador ou exclusivo (XOR)

```
#include <stdio.h>
int xor(int a, int b);
void main(void)
{
   printf("%d", xor(1, 0));
   printf("%d", xor(1, 0));
   printf("%d", xor(1, 0));
   printf("%d", xor(1, 0));
}
xor(int a, int b)
{
     return((a || b) && !(a && b));
}
```

Exercícios

1- A operação lógica

(-5 || 0)&&(3 >= 2)&&(1 != 0)||(3 < 0) é:

a). Verdadeira

b). Falsa

c). Inválida, pois sua sintaxe está errada.

d). Nem Verdadeira nem Falsa

e). Nenhuma das opções anteriores

Precedência dos Operadores

Maior precedência

( ) [ ] ->

! ~ ++ -- . -(unário) (cast) *(unário) &(unário) sizeof

* / %

+ -

<< >>

<<= >>=

== !=

&

^

|

&&

||

?

= += -= *= /=

Menor precedência

Encadeando expressões:
o operador virgula “,”

O operador , determina uma lista de expressões que devem ser executadas seqüencialmente. O valor retornado por uma expressão com o operador , é sempre dado pela expressão mais à direita.

Exemplo:

```
x = (y=2, y+3);
```

o valor 2 vai ser atribuído a y, se somará 3 a y e o retorno (5) será atribuído à variável x . Pode-se encadear quantos operadores “,” forem necessários.

Modeladores (Casts)

Um modelador é aplicado a uma expressão. Ele força a mesma a ser de um tipo especificado. Sua forma geral é:

(tipo) expressão

```
#include <stdio.h>
 void main ()
 {
      int num;
      float f;
      num = 10;
      f = (float)num/7;
      printf ("%f", f);
}
```

Se não tivéssemos usado o modelador no exemplo ao lado, o C faria uma divisão inteira entre 10 e 7. O resultado seria 1 (um) e este seria depois convertido para **float** mas continuaria a ser 10. Com o *cast* temos o resultado correto.

Exercícios

1. Verificar o código ASCII dos caracteres ‘0’, ‘9’, ‘a’ e ‘A’. (dica: ler um *char* e imprimir como *int*)
2. Ler um caracter do teclado e verificar se é um caracter de pontuação: ‘,’ *ou* ‘.’ *ou* ‘;’ *ou* ‘!’ *ou* ‘?’
3. Verificar se um caracter lido do teclado é maiúsculo ou minúsculo (entre ‘a’ e ‘z’ é minúsculo)
4. Ler um string do teclado com a função *gets(s)* e imprimir o número de ocorrências do caracter ‘a’
5. Fazer *maiuscula(s)*: transformar todas as letras minúsculas em maiúsculas em *s*.

Fim da aula