Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

Resumo de conceitos básicos de C

Material preparado pela profa
Silvana Maria Affonso de Lara

2º semestre de 2010

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

Variáveis

- Exemplos de nomes de variáveis:

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

- 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 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

Modificadores de Tipos

- modificadores alteram algumas características dos tipos básicos para adequá-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 de 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 de 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 do Tipo Char

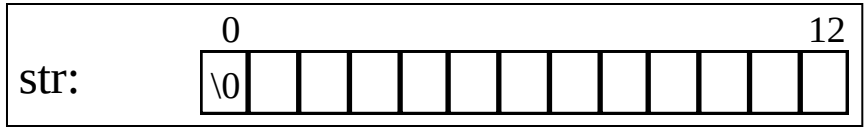
- Barra invertida
- \a bip
- \b backspace
- \n newline
- \t tab horizontal
- \’ apóstrofe
- \” aspa
- \\ backslash
- \f form feed

Strings

- Strings são seqüências de caracteres adjacentes na memória.
- O caracter '\0' (= valor inteiro 0) indica o fim da seqüência

  Ex: `char str[13];`

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



Operações com Strings

- atribuição: não se pode atribuir uma string a outra 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"
```

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 */

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;

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)

Precedência dos Operadores

Maior precedência

```
( ) [ ] ->
! ~ ++ -- . -(unário) (cast) *(unário) sizeof
* / %
+ -
<< >>
<<= >>=
== !=
&
^
|
&&
||
?
= += -= *= /=
```

Menor precedência

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 1.0. Com o *cast* temos o resultado correto.

O Comando IF

if (*expressão é verdadeira*)
    execute comando ou bloco de comandos ;
else /* se expressão é falsa */
    execute comando ou bloco de comandos ;

Ex:

```
if ( count > 9 )
        count = 0;
else
        count++;
```

EXEMPLO IF – ELSE

```
#include <stdio.h>
void main ( )
{
        int num;
        printf ("Digite um numero: ");
        scanf ("%d", &num);
        if (num == 10)
        {
                printf ("\n\n Voce acertou!\n");
                printf ("O numero e igual a 10.\n");
        }
        else
        {
                printf ("\n\n Voce errou!\n");
                printf ("O numero e diferente de 10.\n");
        }
}
```

Aninhamento de IF

É possível aninhar construções do tipo if-else em diversos níveis

```
if (cond1)                  /* if1 */
  if (cond2)                /* if2 */
    comando if2;
  else                      /* else2 */
    comando else2;
else                        /* else1 */
  if (cond3)                /* if3 */
    if (cond4)              /* if4 */
      comando if4;
    else                    /* else4 */
      comando else4;
  else                      /* else3 */
    comando else3;
```

Exemplo ELSE-IF

```
#include <stdio.h>
void main ()
{
        int num;

        printf ("Digite um numero: ");
        scanf ("%d",&num);
        if (num > 10)
                printf ("\n\n O numero e maior que 10");
        else if (num == 10)
        {
                printf ("\n\n Voce acertou!\n");
                printf ("O numero e igual a 10.");
        }
        else if (num < 10)
             printf ("\n\n O numero e menor que 10");
}
```

O Operador "?"

- Uma expressão como:

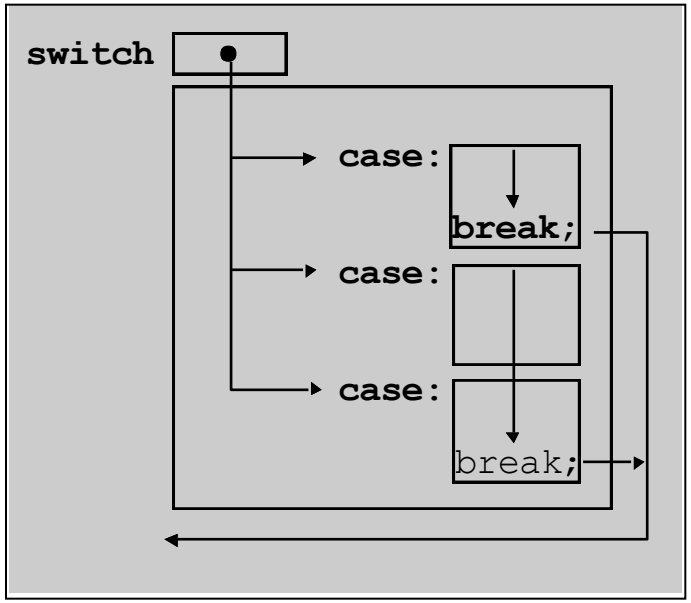
```
if (a > 0)
        b = -150;
else
        b = 150;
```

pode ser simplificada usando-se o operador **?** da seguinte maneira:

```
b = a > 0 ? -150 : 150;
```

O comando Switch

```
switch ( valor ) {
    case valor1:
                    comandos1;
                    break;
    case valork:
                    comandosk;
                    break;
    default:
            comandos_default;
            break;
}
```



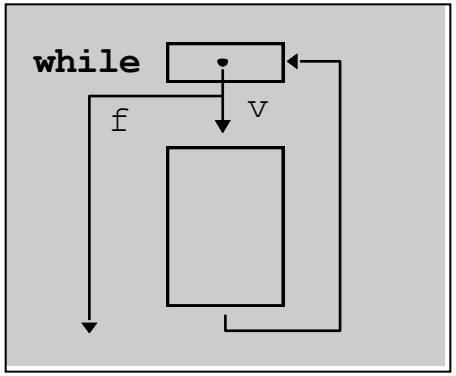
O comando **switch** é próprio para se testar uma variável em relação a diversos valores pré-estabelecidos.

EXEMPLO SWITCH

```
switch( char_in ) {
   case '.': printf("Ponto.\n" );
                  break;
   case ',': printf( "Virgula.\n" );
                  break;
   case ':': printf( "Dois pontos.\n" );
                  break;
   case ';': printf( "Ponto e virgula.\n");
                  break;
   default : printf( "Nao eh pontuacao.\n" );
}
```

O Comando While

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



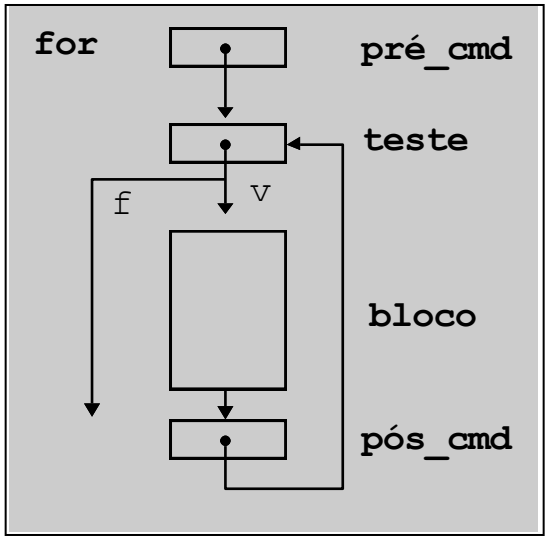
- 1º avalia condição
- se condição é verdadeira, executa comandos do bloco
- ao término do bloco, volta a avaliar condição
- repete o processo até que condição seja falsa

O Comando For

for (pré_cmd; teste; pós_cmd) {
  comandos;
}

- em termos de while, equivale a:

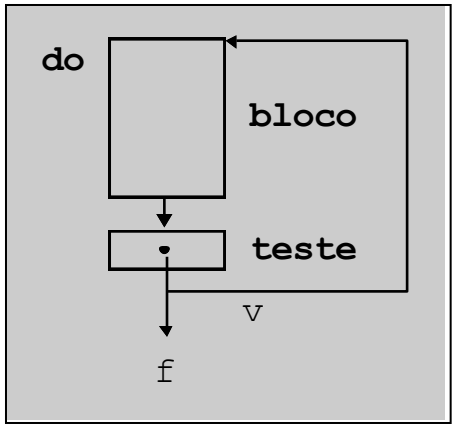
  pré_cmd;
  while (teste) {
    comandos;
    pós_cmd;
  }



O Comando Do-While

- do-while é utilizado sempre que o bloco de comandos deve ser executado ao menos uma vez

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



O Comando Break

- o comando *break* permite interromper a execução de um laço ou de um *switch*
- Ex:

```
main () {
int i, j;
for (i = 0; i < 4; i++)
 for (j = 0; j < 2; j++)
  if (i == 1) break;
  else printf("i: %d j: %d\n", i, j);
}
```

```
i: 0 j: 0
i: 0 j: 1
i: 2 j: 0
i: 2 j: 1
i: 3 j: 0
i: 3 j: 1
```

O Comando Continue

- o comando *continue* leva a execução do próximo passo de uma iteração. Os comandos que sucedem continue no bloco não são executados

- Ex:

```
main() {
int i;
  for (i = 0; i < 5; i++)
        if (i == 1) continue;
        else printf("i: %d \n", i);
}
```

```
i: 0
i: 2
i: 3
i: 4
```

Exercício

- Escreva um programa em C para ler 8 dígitos binários (tem que conferir se é mesmo 0 ou 1), e apresenta no final o número convertido para decimal.

- Exemplo:
- 00010011 → Este número em decimal é 19.
- 00001010 → Este número em decimal é 10.