Programação Estruturada Aula 4 - Introdução

Prof. Luis Carlos Retondaro

Técnico em Telecomunicações
2^o Ano

CEFET/RJ - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

Campus Petrópolis

2017

Sumário

Operadores

- A linguagem C é muito rica em operadores internos;
- No C são definidas 4 classes de operadores:
  1. Operadores aritméticos;
  2. Operadores relacionais;
  3. Operadores lógicos;
  4. Operadores bit a bit;
- Além desses, o C tem operadores especiais para tarefas específicas.

Operador de atribuição

- O operador de atribuição (=) é utilizado para atribuir um valor (constante ou variável) a uma variável;
- Forma geral do operador de atribuição:

```
nome_da_variavel = expressao;
```

- Sendo que a expressão pode ser tão simples como uma constante ou tão complexa quanto seja necessário;
- O destino deve ser uma variável ou um ponteiro.

Operador de atribuição

- **Exemplo:** expressões simples:

```
int num, n;
char letra;
double salario;

num = 10;
letra = 'A';
salario = 2345.54;
n = num;
```

- **Exemplo:** errado de atribuição;

```
int a;
unsigned int b;
float f, g;
char c;

a b = 10;
b = -15
d = 90;
```

Operador de atribuição

- A operação de atribuição também pode aparecer no momento da declaração de uma variável;

```
void main(){
 int a = 10, b = -5; //na mesma linha separados por ,
 float c;
 char d = '4';

 c = a + 1.8;          //atribui o valor 11.8 a c
 b = a + b;            // soma 10 e -5 e atribui 5 a b
}
```

Atenção

Todo comando de declaração de variável ou de atribuição deve terminar com ';'

As atribuições só podem ser feitas após a declaração da variável.

Conversão de tipos em atribuição

- A converão de tipos ocorre quando variáveis de um tipo são misturadas com variáveis de outro tipo;
- Em um comando de atribuição a regra de converão de tipos é simples: o valor do lado direito da atribuição é convertido no tipo do lado esquerdo.

```
#include <stdio.h>
void main(){
        int x = 2, num = 3;
        char ch = 'L';
        float f = 12.45;

        ch = x;
        x = f;
        f = ch;
        printf("**** Conversão ****\n");
        printf("int -> char: %d\n",ch);
        printf("float -> int: %d\n",x);
        printf("char -> float: %f\n",f);
        f = num;
        printf("int -> float: %f\n\n", f);
}
```

Conversão de tipos em atribuição

```c
#include <stdio.h>
void main(){
        int x = 2, num = 3;
        char ch = 'L';
        float f = 12.45;

        ch = x;
        x = f;
        f = ch;
        printf("**** Conversão ****\n");
        printf("int -> char: %d\n",ch);
        printf("float -> int: %d\n",x);
        printf("char -> float: %f\n",f);
        f = num;
        printf("int -> float: %f\n\n", f);
}
```

- Na primeira conversão, Os bits mais significativos da variável inteira x são ignorados, deixando o ch com os 8 bits menos significativos (entre 0 e 256 possuem valores idênticos);
- Na segunda conversão, x recebe a parte inteira de f.

Conversão de tipos em atribuição

```c
#include <stdio.h>

void main(){
        int x = 2, num = 3;
        char ch = 'L';
        float f = 12.45;

        ch = x;
        x = f;
        f = ch;
        printf("**** Conversão ****\n");
        printf("int -> char: %d\n",ch);
        printf("float -> int: %d\n",x);
        printf("char -> float: %f\n",f);
        f = num;
        printf("int -> float: %f\n\n", f);
}
```

- Na terceira conversão, f converte o valor inteiro de 8 bits armazenado em ch, no mesmo valor em formato de ponto flutuante;
- Na quarta conversão, f converte o valor inteiro de 16 bits no formato de ponto flutuante.

Conversão de tipos em atribuição

Resultado da execução do programa de conversão de tipos

```
**** Conversão ****
int -> char: 2
float -> int: 12
char -> float: 2.000000
int -> float: 3.000000
```

```
//imprimindo ch como caractere

**** Conversão ****
int -> char:
float -> int: 12
char -> float: 2.000000
int -> float: 3.000000
```

Conversão de tipos em atribuição

- A regra básica para conversão de *int* → *char*, *int* → *short int*, *long int* → *int* é que a quantidade de bits significativos será ignorada;
- A conversão de *int* → *float* ou *float* → *double*, não aumenta a precisão, apenas muda a forma como o valor é representado;
- A forma de representação de cada tipo pode variar de compilador para compilador (e processadores);
- Então, de forma geral, o melhor é usar variáveis do tipo char para armazenar caracteres, int para inteiro, etc;
- Sempre que for necessário fazer uma conversão sempre converta um tipo por vez. Exemplo: *double* → *int*, primeiro converta *double* → *float* depois *float* → *int*.

Atribuição múltipla

- O C permite que se atribuia o mesmo valor a muitas variáveis, em um único comando, usando atribuições múltiplas;

```
int x, y, z;

x = y = z = 0;
```

- Valores comuns são atribuídos a variáveis diferentes usando esse método.

Sintaxe dos operadores aritméticos

- A linguagem C possui representação para as principais operações aritméticas;
- Os operadores aritméticos podem ser aplicados em qualquer tipo de dado interno permitido em c.

Operador aritmético	**Ação**
-	Subtração
+	Adição
*	Multiplicação
/	Divisão
%	Resto da divisão
++	Incremento
- -	Decremento

Sintaxe dos operadores aritméticos

- Quando o operador $\backslash$ é aplicado a um inteiro ou caractere, qualquer resto é truncado. Exemplo: $5 \backslash 2 = 2$;
- O operador de módulo (%) devolve o resto da divisão, porém ele não pode ser aplicado a tipos de ponto flutuante;
- **Exemplo:** $22\%5 = 2$, porque $(4x5 + 2) = 22$.

Resultado

Resultado do exemplo do operador módulo

```
#include <stdio.h>
 //exemplifica o operador \%
 int main(){
    int x, y;

    x = 5;
    y = 2;
    printf("Divisão: %d\t", x/y);
    printf("Módulo: %d\n\n", x%y);

    x = 1;
    y = 2;
    printf("Divisão: %d\t Módulo: %d\n", x/y, x%y);
    return(0);
 }
```

```
Divisão: 2      Módulo: 1

Divisão: 0       Módulo: 1
```

Incremento e Decremento

- Ambos operadores só possuem um operando;
- O operador de incremento soma 1 ao seu operando;
- O operador de decremento subtrai 1 do seu operando.

```
//semelhança entre operações

x = x+1;
++x;

x = x-1;
x--;
```

- Ambos operadores podem ser usados como prefixo ou sufixo.

Incremento e Decremento

Exemplo do uso do operador ++

```
#include <stdio.h>

int main(){
    int x = 10, y;

    y = ++x;
    printf("y: %d\n",y);

    x = 10;
    y = x++;
    printf("y: %d\n",y);
    return(0);
 }
```

Resultado

```
y: 11
y: 10
```

Precedência dos operadores

Mais alta	++ - -
	- (menos unário)
	* / %
Mais baixa	+ -

- Os operadores com mesmo nível de precedência são avaliados pelo compilador da esquerda para a direita;
- Os parênteses podem ser usados para alterar a ordem de precedência, forçando a operação a ter um nível de precedência maior.

Precedência dos operadores

Calcule o resultado das expressões:

```
int x,y,z;
float a;

x=y=10;
z=++x;
x= -x;
y++;
x=x+y-(z- -);
x = 25 % 5;
y = 6 / 4;
a = 6 / 4
y = 3 + 5 * 4 % (2 + 8);
```

Precedência dos operadores

Calcule o resultado das expressões:

```
int x,y,z;

x=y=10;           // x = 10 e y = 10
z=++x;            // z = 11 e x = 11
w = x++;          // w = 11 e x = 12
x=-x;             // x = -12
y++;             // y = 11
x=x+y-(z--);     // x = -12 z = 10
x = 25 % 5;      // x = 0
y = 6 / 4;       // y = 1
a = 6 / 4        // a = 1.5
y = 3 + 5 * 4 % (2 + 8);          //y = 3
```

Operadores relacionais e lógicos

- O termo relacional refere-se as relações que os valores podem ter uns com os outros;
- O termo lógico refere-se as maneiras como essas relações podem ser conectadas;
- A ideia de verdadeiro e falso é a base dos conceitos dos operadores lógicos e relacionais;
- Em C verdadeiro é qualquer valor diferente de zero (0) e falso é zero.

Operadores relacionais e lógicos

Expressões relacionais

- Expressões relacionais são aquelas que realizam uma comparação entre duas expressões e retornam:
  1. 0 (zero), se o resultado for falso;
  2. Qualquer valor diferente de zero, se o resultado for verdadeiro.

Operadores relacionais e lógicos

Operadores relacionais

Operadores relacionais	
Operador	Ação
==	Igualdade
!=	Diferente
>	Maior que
<	Menor que
>=	Maior que ou igual que
<=	Menor que ou igual que

Operadores relacionais e lógicos

Operadores relacionais

- **Exemplos:**
  - **<expressão1> == <expressão2>:** retorna verdadeiro se as duas expressões retornarem o mesmo valor (a == b);
  - **<expressão1> != <expressão2>:** retorna verdadeiro se as duas expressões retornarem valores diferentes (a != b);
  - **<expressão1> > <expressão2>:** retorna verdadeiro se a 1^a expressão retornar um valor maior que a 2^a (a > b);
  - **<expressão1> < <expressão2>:** retorna verdadeiro se a 1^a expressão retornar um valor menor que a 2^a (a < b);

Operadores relacionais e lógicos

Operadores lógicos

Operadores lógicos	
Operador	Ação
&&	AND
||	OR
!	NOT

- Expressões lógicas são aquelas que realizam uma operação lógica (e, ou, não, ...) e retornam verdadeiro ou falso;
- É permitido colocar diversas operações em uma única expressão:

```
int a = 10;
int b = 5;
int c = 9

 a > b && !(a < c) || b <= c
```

Operadores relacionais e lógicos

Tabela verdade dos operadores lógicos

p	q	p && q	p || q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Operadores relacionais e lógicos

- Embora C não tenha um operador lógico or exclusivo (xor), pode-se facilmente criar uma função que execute essa tarefa usando outros operadores lógicos.

```
#include <stdio.h>
int xor(int a, int b){
        return((a || b) && !(a && b));
}

void main(void){
printf("xor(1,0) = %d\n", xor(1,0));
printf("xor(1, 1) = %d\n", xor(1, 1));
printf("xor(0, 1) = %d\n", xor(0, 1));
printf("xor(0, 0) = %d\n", xor(0, 0));

}
```

Operadores relacionais e lógicos

Exemplo operadores lógicos

```
 ...

Se(num >= 0)
  Se(num % 2 == 0)
    printf("Número par não negativo.\n");


//conectando as duas condições com um operador lógico

Se((num>=0) && (num%2 == 0))
  printf("Numero par nao negativo.\n");

//ou...

Se(num>0 && !(num%2))
  printf("Numero par nao negativo.\n");

...
```

Operadores relacionais e lógicos

Precedência relativa dos operadores relacionais e lógicos

Maior	!
	> >= < <=
	== !=
	&&
Menor	||

Exemplo:

```
!0 && 0 || 0

!(0 && 0) || 0
```

Operadores relacionais e lógicos

Precedência - O que será impresso?

```c
#include <stdio.h>
int main(){
  int score = 5;

  printf("%d\n", 5 + 10 * 5 % 6);
  printf("%d\n", 10 / 4);
  printf("%f\n", 10.0 / 4.0);
  printf("%c\n", 'A' + 1);
  printf("%d\n", score + (score == 0));
}
```

Operadores relacionais e lógicos

Precedência - O que será impresso?

```c
#include <stdio.h>
int main(){
  int score = 5;

  printf("%d\n", 5 + 10 * 5 % 6);
  printf("%d\n", 10 / 4);
  printf("%f\n", 10.0 / 4.0);
  printf("%c\n", 'A' + 1);
  printf("%d\n", score + (score == 0));
}
```

```
7
2
2.500000
B
5
```

O operador vírgula

- O operador vírgula (,) é usado para encadear diversas expressões;

```
x = (y=3, y+1);
```

- Primeiro atribui o valor 3 a y, em seguida, atribui o valor 4 a x;
- Os parênteses são necessários porque o operador de vírgula tem uma precedência menor que o de atribuição;
- A vírgula provoca uma sequência de operações;
- Esse operador tem o mesmo significado da palavra e em português, como faça isso e isso e isso...

Outros operadores

- operadores de ponteiros: & e *;
- operadores de estruturas e uniões: $\cdot$ e $\rightarrow$;
- Operador ? : substitui certas centenças da forma if-then-else;
- Parênteses como operadores: servem para aumentar a precedência das operações dentro deles;
- Colchetes como operadores: Realizam indexação de matrizes.

Resumo das precedências

Maior	() [] →
	! ++ - - -(tipo) * &
	* / %
	+ -
	< <= > >=
	== !=
	&&
	||
	?:
	= += -= *= etc
Menor	,

Uso de cast

- O programdor pode forçar uma expressão a ser de um determinado tipo usando um cast;
- Forma genérica:

```
(tipo) expressão;
```

- Onde tipo é qualquer tipo de dados válido em C;
- **Exemplo:** Para garantir que a expressão $x \setminus 2$ seja um float:

```
(float)x\2;
```

Uso de cast

Exemplo:

```
#include <stdio.h>

void main(void){
  int i;

  for(i=1; i<=100; i++)
    printf("%d / 2 é: %f\n", i, (float)i/2);
}
```

```
 1 / 2 é: 0.500000
2 / 2 é: 1.000000
3 / 2 é: 1.500000
4 / 2 é: 2.000000
5 / 2 é: 2.500000
6 / 2 é: 3.000000
...
95 / 2 é: 47.500000
96 / 2 é: 48.000000
97 / 2 é: 48.500000
98 / 2 é: 49.000000
99 / 2 é: 49.500000
100 / 2 é: 50.000000
```

Uso de cast

Exemplo:

```
#include <stdio.h>

void main(void){
  int i;

  for(i=1; i<=100; i++)
    printf("%d / 2 é: %f\n", i, (float)i/2);
}
```

- Sem o cast (float), teria sido efetuado somente uma divisão inteira;
- O cast garante exibir a parte fracionária do resultado.

O uso de espaçamento e parênteses

- O programador pode usar tabulações e espaços em expressões em C para torná-las mais legíveis;

```
// A expressão
 x=10/y+(127/x);

//idêntica a
 x = 10/y + (127/x);
```

- Parênteses redundantes ou adicionais não causam erros ou diminuem a velocidade da execução das expressões;

- Usa-se parênteses para esclarecer a ordem de avaliação para o próprio programador e os demais.

```
 x=y/2-34*temp%127;

 x = (y/2) - ((34*temp) %127);
```

Simplificando expressões

- Existe uma variante no comando de atribuição, às vezes, chamado de C reduzido, que simplifica a codificação de um certo tipo de atribuição;

```
 x = x + 10;

//identico a
 x +=10;
```

- Essa abreviação existe para todos os operadores binários em C. Forma geral:

```
var = var operador expressão;

//idêntico a
var operador= expressão;
```

Referências

1. C Completo e Total, Herbert Schidt; Pearson Makron Books; 3a. Ed., 1997.
2. Linguagem C. DAMAS, Luis. 10a. Edição. LTC, 2014.