Programação Estruturada

Aula 3 - Expressões

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- Expressões são fórmulas que mostram como computar um valor
- Variáveis e constantes são as expressões mais simples
    - Variáveis são computadas quando o programa é executado
    - Constantes são valores que não mudam

Introdução

- Expressões mais complexas aplicam operadores a operandos
  - Os operandos também são expressões
- Na expressão `a+(b*c)`
  - `+` é aplicado aos operandos `a` e `(b*c)`
  - `*` é aplicado aos operandos `b` e `c`

Introdução

- Operadores são as ferramentas básicas para construir expressões
- Operadores aritméticos: adição, subtração, multiplicação e divisão
- Operadores relacionais: realizam comparações do tipo "a maior que b"
- Operadores lógicos: constroem condições do tipo: "a maior que b **e** a menor que 10"
- Entre outros...

Operadores aritméticos

- C possuir os seguintes operadores aritméticos binários (que se aplicam a dois operandos):
  - `+` - adição
  - `-` - subtração
  - `*` - multiplicação
  - `/` - divisão
  - `%` - resto da divisão

Operadores aritméticos

```c
#include <stdio.h>

int main(void) {
    int a, b, c, d, e, f;

    a = 1+2;
    b = 2-1;
    c = 2*3;
    d = 4/2;
    e = 3/2;
    f = 3%2;

    printf("%d\n", a);
    printf("%d\n", b);
    printf("%d\n", c);
    printf("%d\n", d);
    printf("%d\n", e);
    printf("%d\n", f);

    return 0;
}
```

Operadores aritméticos

- Na divisão, quando os dois operadores são inteiros, o resultado vai ser inteiro
- Então, `3/2` resulta em 1, ao invés de 1.5

Operadores aritméticos

```c
#include <stdio.h>

int main(void) {
    float g, h;

    g = 3/2;
    h = 3.0/2.0;

    printf("%f\n", g);
    printf("%f\n", h);

    return 0;
}
```

- `3/2` continua resultando em 1, mas como temos uma variável do tipo float, o resultado será `1.000000`
- `3.0/2.0` tem floats como operandos, então o seu resultado é `1.500000`

Operadores aritméticos

- C também possui operadores aritméticos unários
  - `-` inverte o sinal de um valor
  - `+` mantém o sinal de um valor

Operadores aritméticos

```c
int main(void) {
    int a, b, c, d, e, f;
    float g, h;

    a = -1;
    b = -2+1;
    c = -2-1;
    d = -3*2;
    e = -4/2;
    f = -3/2;
    g = -3/2;
    h = -3.0/2.0;

    printf("%d\n", a);
    printf("%d\n", b);
    printf("%d\n", c);
    printf("%d\n", d);
    printf("%d\n", e);
    printf("%d\n", f);
    printf("%f\n", g);
    printf("%f\n", h);

    return 0;
}
```

Precedência e associatividade

- Quando uma expressão contém mais de um operador sua interpretação pode não ser clara
- `i + j * k`
  - soma `i` e `j` e depois multiplica por `k`?
  - multiplica `j` e `k` de pois soma `i`?

Precedência e associatividade

- Para não confundir, você pode usar parênteses
- soma `i` e `j` e depois multiplica por `k`?
  - `(i + j) * k`
- multiplica `j` e `k` de pois soma `i`?
  - `i + (j * k)`

Precedência e associatividade

- Mesmo sem os parênteses, a linguagem executa sem ambiguidade as expressões
- As regras para execução são dadas pela precedência dos operadores

Precedência	Operador
Alta	`+` `-` (unários)
	`*` `/` `%`
Baixa	`+` `-` (binários)

- Operadores em uma mesma linha tem precedências iguais

Precedência e associatividade

- Quando dois ou mais operadores aparecem na expressão, o compilador inicia com os que tem precedência mais alta e vai descendo para os de precedência mais baixa
- `i + j * k` é equivalente a `i + (j * k)`
- `-i * -j` é equivalente a `(-i) * (-j)`

Precedência e associatividade

- Usar a precedência de operadores não é suficiente para executar as operações
- Existem casos que operadores possuem a mesma precedência
- Neste caso entra a associatividade dos operadores

Precedência e associatividade

- Os operadores aritméticos binários tem associatividade à esquerda
  - Eles são agrupados da esquerda para direita
- `i + j - k` é equivalente a `(i + j) - k`
- `i * j / k` é equivalente a `(i * j) / k`

Precedência e associatividade

- Os operadores aritméticos unários tem associatividade à direita
  - Eles são agrupados da direita para esquerda
- `- + i` é equivalente a `- (+i)`

Operador de atribuição

- Após computar o valor de uma expressão, é comum guardarmos ele em uma variável
- `=` é o operador de atribuição

```
int i, j, k;

i = 5;
j = i;
k = 10 * i + j;
```

- `k` recebe o valor 55

Operador de atribuição

- Se atribuirmos um float a uma variável inteira ou vice-versa, os valores são convertidos

```
int i;
float;

i = 72.99;
f = 136;
```

- `i` recebe o valor 72
- `j` recebe o valor 136.0

Operador de atribuição

- A atribuição é um operador, ou seja, ela produz um resultado
- `i=10` produz o valor 10
- O valor produzido é o valor da variável, assim `i=72.99` produz 72, não 72.99

Operador de atribuição

- Com isso, a seguinte expressão é válida:
  - `i = j = k = 0;`
  - Que é equivalente a: `i = (j = (k = 0));`

Operador de atribuição

- Qual o valor de `f` no programa abaixo?

```
int i;
float f;

f = i = 33.3;
```

Operador de atribuição

- Qual o valor de `k` no programa abaixo?

```
int i, j, k;

i = 1;
k = 1 + (j = i);
```

- Evite usar atribuição dentro de uma expressão, ela deixa o programa difícil de ler

Operador de atribuição - efeito colateral

- Normalmente, nós não esperamos que um operador modifique seus operandos
  - `i + j` não modifica `i` nem `j`
- Um operador que modifica seus operandos tem **efeito colateral**
  - Eles estão fazendo algo mais além de computar valores
- O operador de atribuição tem o efeito colateral de modificar o seu operando à esquerda
  - O operador da esquerda precisa ser uma variável

Operador de atribuição - atribuição composta

- Atribuições que usam o valor antigo de uma variável para computar um novo valor para ela são comuns
  - Por exemplo, `i = i + 2;`
- A atribuição composta permite realizar essa atribuição de uma maneira mais curta
  - `i += 2;`, isto é o mesmo que `i = i + 2;`
- Também temos os operadores `-=` `*=` `/=` `%=`

Operadores de incremento e decremento

- Outra operação muito comum em variáveis é adicionar 1 (incrementar) ou subtrair 1 (decrementar)
  - `i = i + 1;`
  - `i = i - 1;`
- Podemos usar atribuição composta:
  - `i += 1;`
  - `i -= 1;`

Operadores de incremento e decremento

- Para incremento e decremento, podemos diminuir ainda mais a atribuição:
  - `i++;` (incremento)
  - `i--;` (decremento)

```c
int main(void) {
    int i;

    i = 1;
    i++;

    printf("%d\n", i);

    return 0;
}
```

- O programa exibe o valor 2

Operadores de incremento e decremento

- Os operadores `++` e `--` podem ser usados pós-fixados ou pré-fixados
  - `i++` (operador pós-fixado)
  - `++i` (operador pré-fixado)

```c
int main(void) {
    int i;

    i = 1;
    ++i;

    printf("%d\n", i);

    return 0;
}
```

- O programa também exibe o valor 2

Operadores de incremento e decremento

- `i++` produz o valor `i` e atribui o valor `i+1` a variável `i`

```c
int main(void) {
    int i, j;

    i = 1;
    j = i++;

    printf("i=%d\n", i);
    printf("j=%d\n", j);

    return 0;
}
```

- `i` tem valor 2 e `j` tem valor 1

Operadores de incremento e decremento

- `++i` atribui o valor `i+1` a variável `i` e produz o valor `i` (que agora já está somado de 1)

```c
int main(void) {
    int i, j;

    i = 1;
    j = ++i;

    printf("i=%d\n", i);
    printf("j=%d\n", j);

    return 0;
}
```

- `i` tem valor 2 e `j` tem valor 2

Avaliando expressões

Precedência	Nome	Símbolos	Associativ.
1	Incremento e decremento (pós)	`++` `--`	Esquerda
2	Incremento e decremento (pré)	`++` `--`	Direita
2	Mais e menos unário	`+` `-`	Direita
3	Multiplicativo	`*` `/` `%`	Esquerda
4	Aditivo	`+` `-`	Esquerda
5	Atribuição	`=` `*=` `/=` `%=` `+=` `-=`	Direita

Avaliando expressões

- Vamos avaliar uma expressão complexa:
  - `a = b += c++ - d + --e / -f`
- Vamos identificar o operador com maior precedência e colocar parênteses em volta ele e seus operandos

Avaliando expressões

- `++` pós-fixado tem a maior precedência:
  - `a = b += (c++) - d + --e / -f`
- Repetindo o processo, vamos encontrar o próximo operador com maior precedência e colocar parênteses em volta dele e dos seus operandos

Avaliando expressões

- `--` pré-fixado e `-` unário têm a segunda maior precedência:
  - `a = b += (c++) - d + (--e) / (-f)`
- Continuamos até que a expressão esteja com parênteses para todos os operandos

Avaliando expressões

- O operador `/` tem precedência 3:
  - `a = b += (c++) - d + ((--e) / (-f))`

Avaliando expressões

- Os próximos operadores na tabela de precedência são o `+` e o `-`
- Eles são adjacentes na expressão
- Como suas associatividades são à esquerda, então agrupamos a partir da esquerda
  - `a = b += ((c++) - d) + ((--e) / (-f))`
  - `a = b += (((c++) - d) + ((--e) / (-f)))`

Avaliando expressões

- Falta apenas os operadores de atribuição
- Eles também são adjacentes, mas possuem associatividades à direita
  - `a = (b += (((c++) - d) + ((--e) / (-f))))`
  - `(a = (b += (((c++) - d) + ((--e) / (-f)))))`