Programação Estruturada

Aula 4 - Comandos de seleção

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- Comandos de seleção permitem um programa selecionar um determinado caminho de execução de acordo com alguma condição
    - Comandos `if` e `switch`
- As condições são escritas através de expressões lógicas

Expressões lógicas

- O comando `if`, por exemplo, precisa testar se uma expressão é verdadeira ou falsa
  - `i < j` - `i` menor que `j`
- C não possui valores booleanos
- Uma expressão lógica retorna `0` se ela for falsa e `1` se for verdadeira

Operadores relacionais

Símbolo	Significado
`<`	menor que
`>`	maior que
`<=`	menor ou igual a
`>=`	maior ou igual a

- Estes operadores podem ser usados para comparar inteiros e floats, inclusive com operandos de tipos misturados
    - `1 < 2.5` resulta no valor 1 e `5.6 < 4` resulta em 0

Operadores relacionais

- A precedência dos operadores relacionais são menores que a dos operadores aritméticos
- `i + j < k - 1` é o mesmo que `(i + j) < (k - 1)`
- Os operadores relacionais tem associatividade à esquerda

Operadores relacionais

- A expressão `i < j < k` é válida, mas não tem o sentido que normalmente esperamos
- Como a associatividade do operador é à esquerda, temos que a expressão é equivalente a:
    - `(i < j) < k`

Operadores de igualdade

Símbolo	Significado
`==`	igual a
`!=`	diferente de

- Os operadores de igualdade tem associatividade à esquerda
- Tem precedência menor que os operadores relacionais
- `i < j == j > k` é equivalente a `(i < j) == (j < k)`

Operadores lógicos

- Podemos criar expressões lógicas mais complexas a partir de expressões simples que são unidas através de operadores lógicos

Símbolo	Significado
`!`	negação
`&&`	and
`||`	or

- Os operadores lógicos tratam `0` como falso e qualquer valor diferente de `0` como verdadeiro

Operadores lógicos

- `!expr` tem valor 1 se `expr` tiver valor `0`
- `expr1 && expr2` tem valor 1 se os valores de `expr1` e `expr2` ambos forem diferentes de `0`
- `expr1 || expr2` tem valor 1 se os valores de `expr1` ou `expr2` (ou os dois) forem diferentes de `0`
- Nos outros casos, esses operadores produzem o valor `0`

Operadores lógicos - curto circuito

- Os operadores `&&` e `||` usam o curto circuito para avaliar seus operandos
- Primeiro eles avaliam o operando do lado esquerdo, se já for possível deduzir o resultado apenas desse operando, então o operando do lado direito não é avaliado
- `(i != 0) && (j / i > 0)`
  - Se `(i != 0)` resultar em `0`, então já sabemos que o valor da expressão vai ser `0` (falso) também

Operadores lógicos - curto circuito

- Cuidado com efeitos colaterais em expressões lógicas
- Por causa do curto circuito, efeitos colaterais podem não acontecer em alguns casos
- `i > 0 && ++j > 0`
- Aparentemente `j` está sendo incrementado quando a expressão é avaliada
- Entretanto, se `i > 0` for falso, `++j > 0` não é avaliado e `j` não é incrementado
- É melhor incrementar `j` separadamente

Operadores lógicos - precedências e associatividades

- `!` tem a mesma precedência do `+` e `–` unários
- `&&` e `||` têm precedência menor que dos operadores relacionais e de igualdade
- `!` tem associatividade à direita
- `&&` e `||` têm associatividade à esquerda

Comando if

- O comando `if` permite que o programa escolha entre duas alternativas de acordo com uma expressão
- O comando `if` tem a seguinte forma:
- `if (expressão) comando`
    - os parênteses são obrigatórios

Comando if

- `if (expressão) comando`
- Ao ser executado, a expressão é avaliada. Se ela for verdadeira (diferente de `0`), o comando é executado

```c
if (x > 10)
    printf("x é maior que 10\n");
```

Comando if

- Para testar se `x` está entre um intervalo de valores, temos:

```
if (0 <= x && x < 10)
```

- Estamos testando se `x` é maior ou igual a `0` e menor que `10`

Comando if

- Para testar se `x` **não** está entre um intervalo de valores, temos:

```
if (x < 0 || x >= 10)
```

- Estamos testando se `x` é menor que `0` ou maior ou igual a `10`

Comando if - comandos compostos

- `if (expressão) comando`
  - Note que aqui só temos um comando se a expressão for verdadeira
- Para usarmos dois ou mais comandos, nós definimos um comando composto (ou bloco de comandos)
- `if (expressão) { comandos }`

Comando if - comandos compostos

```
if (x > 10) { printf("x é maior que 10\n"); printf("outro comando!\n"); }
```

Comando if - comandos compostos

- Para melhorar a leitura:

```
if (x > 10) {
    printf("x é maior que 10\n");
    printf("outro comando!\n");
}
```

Comando else

- O comando `if` pode vir acompanhado do `else`
- `if (expressão) comando else comando`
- O comando após a palavra `else` é executado se a expressão for falsa (resultar em 0)

Comando else

```
if (x > 10)
    printf("x é maior que 10\n");
else
    printf("x não é maior que 10\n");
```

Comando else

- O `else` também pode receber um bloco de comandos

```
if (x > 10) {
    printf("executando if...\n");
    printf("x é maior que 10\n");
}
else {
    printf("executando else...\n");
    printf("x não é maior que 10\n");
}
```

Aninhando ifs

- Não existe restrição de quais comandos podem aparecer em um `if`
- Podemos colocar comandos `if` dentro de um `if`

Aninhando ifs - exercício

- Escreva um programa que recebe três números e exibe qual o maior deles

Aninhando ifs - exercício

```c
int i, j, k, max;

printf("Digite o primeiro valor: ");
scanf("%d", &i);

printf("Digite o segundo valor: ");
scanf("%d", &j);

printf("Digite o terceiro valor: ");
scanf("%d", &k);

if (i > j)
    if (i > k)
        max = i;
    else
        max = k;
else
    if (j > k)
        max = j;
    else
        max = k;

printf("O maior valor é %d\n", max);

return 0;
```

Aninhando ifs

- Alinhar os `if`s e `else`s deixa a leitura mais clara
  - Mas o C não obriga o programador a fazer isso
- Usar chaves também facilita a leitura

```
if (i > j) {
    if (i > k)
        max = i;
    else
        max = k;
}
else {
    if (j > k)
        max = j;
    else
        max = k;
}
```

else if

- É comum precisar testar uma série de condições e parar assim que uma delas for verdadeira
- Por exemplo, testar se `x` é menor que `0`, igual a `0` ou maior que `0`

else if

```
if (x < 0) {
    printf("x é menor que 0\n");
}
else {
    if (n == 0) {
      printf("x é igual a 0\n");
    }
    else {
      printf("x é maior que 0\n");
    }
}
```

else if

- Programadores costumam alinhar esse tipo de condição colocando o `else` e o `if` na mesma linha

```
if (x < 0) {
    printf("x é menor que 0\n");
}
else if (n == 0) {
    printf("x é igual a 0\n");
}
else {
    printf("x é maior que 0\n");
}
```

else if

- De forma genérica, temos que um if em cascata se organiza assim:

```
if (expressão)
    comando
else if (expressão)
    comando
...
else if (expressão)
    comando
else
    comando
```

else if

- No código abaixo, o `else` pertence a que `if`?

```
if (y != 0)
    if (x != 0)
        result = x / y;
else
    printf("Erro, y é igual a 0\n");
```

- O `else` pertence ao `if` mais próximo

else if

- No código abaixo, o `else` pertence a que `if`?

```
if (y != 0)
    if (x != 0)
        result = x / y;
    else
        printf("Erro, y é igual a 0\n");
```

- A indentação não faz diferença

else if

- Para fazer o `else` pertencer ao primeiro `if`, vamos usar as chaves

```
if (y != 0) {
    if (x != 0)
        result = x / y;
} else
    printf("Erro, y é igual a 0\n");
```

else if

- Para não causar confusão, pode-se usar as chaves sempre

```
if (y != 0) {
    if (x != 0) {
        result = x / y;
    }
} else {
    printf("Erro, y é igual a 0\n");
}
```