Programação Estruturada

Aula 5 - Comandos de seleção

Yuri Malheiros (yuri@ci.ufpb.br)

Exercício

- Vamos fazer um programa para saber se um aluno foi aprovado por média, vai para final ou foi reprovado de acordo com sua nota

Operador condicional

- O operador condicional permite criar uam expressão que produz um determinado valor de acordo com uma condição
- Este operador é formado por dois símbolos `?` e `:`

```
expressao1 ? expressao2 : expressao3
```

Operador condicional

- O operador condicional é um caso único de operador que precisa de 3 operandos
- Por isso, ele é chamado de operador ternário

Operador condicional

`expressao1 ? expressao2 : expressao3`

- A expressão deve ser lida da seguinte forma:
  - Se `expressao1` for verdadeira, então `expressao2` , se não `expressao3`
  - `if expressao1 then expressao2 else expressao3`

Operador condicional

```
int i, j, k;

i = 1;
j = 2;

k = i > j ? i : j;
k = (i >= 0 ? i : 0) + j;
```

Operador condicional

- Usando operadores condicionais podemos escrever programas menores

```c
int main(void) {
    int i=4;
    int j=5;

    if (i>j) {
        printf("%d\n", i);
    } else {
        printf("%d\n", j);
    }

    return 0;
}
```

Operador condicional

- Entretanto, podemos torná-lo mais difícil de entender

```c
int main(void) {
    int i=4;
    int j=5;

    printf("%d\n", i>j ? i : j);

    return 0;
}
```

switch

- Muitas vezes precisamos comparar uma expressão com uma série de valores
- Para isso podemos usar `if` e `else if` para condicionais:

```
if (nota == 5)
    printf("Excelente\n");
else if (nota == 4)
    printf("Bom\n");
else if (nota == 3)
    printf("Regular\n");
else if (nota == 2)
    printf("Ruim\n");
else if (nota == 1)
    printf("Péssimo\n");
else
    printf("Nota inválida\n");
```

switch

- Podemos fazer um programa equivalente usando `switch`

```
    switch (nota) {
        case 5: printf("Excelente\n");
                break;
        case 4: printf("Bom\n");
                break;
        case 3: printf("Regular\n");
                break;
        case 2: printf("Ruim\n");
                break;
        case 1: printf("Péssimo\n");
                break;
        default: printf("Nota inválida\n");
                 break;
    }
```

switch

```
switch (expressao) {
  case expressao_constante : comandos
  ...
  case expressao_constante : comandos
  default : comandos
}
```

- `expressao` deve produzir um valor inteiro
- A `expressao_constante` é uma expressao que não pode conter uma variável
- Se `expressao` for igual ao resultado de `expressao_constante` então os comandos associados serão executados
- Se `expressao` não for igual a nenhum valor especificado, então o `default` é executado

switch

- Apenas uma constante deve vir após a palavra `case`
- Se for necessário executar os mesmos comandos para mais de um valor, podemos agrupá-los:

```
switch (nota) {
    case 5: printf("Excelente\n");
            break;
    case 4: case 3:
            printf("Bom\n");
            break;
    case 2: case 1:
            printf("Ruim\n");
            break;
    default: printf("Nota inválida\n");
             break;
}
```

switch

- Para que serve o `break` ?
- Se retirarmos o `break`, os comandos dos `case`s após o `case` da condição satisfeita também serão executados
    - Isto dificilmente é o que queremos

switch

```c
int nota=5;

switch (nota) {
    case 5: printf("Excelente\n");
    case 4: printf("Bom\n");
    case 3: printf("Regular\n");
    case 2: printf("Ruim\n");
    case 1: printf("Péssimo\n");
    default: printf("Nota inválida\n");
}
```

- Todos os `printf` s serão executados

switch

- Não usar o `break` pode ser útil em alguns casos
- Se eu quiser calcular a porcentagem de alunos que passaram por média
  - Eu preciso contar quantos passaram por média e quantos alunos tenho no total
- Para isso, eu poderia ter um programa assim:

```
switch (nota) {
  case 10: case 9: case 8: case 7:
      passaram++;
  case 6: case 5: case 4: case 3: case 2: case 1: case 0:
      total++;
}
```