Capítulo 8

Estruturas de controle
no nível de sentença

Tópicos

- Introdução
- Sentenças de seleção
- Sentenças de iteração
- Desvio incondicional
- Comandos protegidos
- Conclusões

Sentenças de controle

- São mecanismos que permitem:
  - **Escolher entre dois ou mais caminhos alternativos**
  - **Executar repetidamente** um ou mais trechos de código do programa
- Quais sentenças de controle vocês conhecem e/ou já utilizaram?

Sentença x Estrutura de Controle

- Detalhe meio bobo mas, formalmente, uma **estrutura de controle** é formada pela *sentença de controle* e a coleção de *sentenças de código* que ela controla.

```c
#include <stdio.h>

int main(void) {
    int idade = 0;

    printf("%s", "Digite sua idade:\n");
    scanf("%i", &idade);

    if (idade == 22) {
        printf("%s", "Dois patinhos na lagora.");
    } else {
        printf("%s", "Não sei sua idade.");
    }
}
```

Sentenças de seleção

- Fornecem um meio de escolher entre caminhos alternativos.

- São divididas em duas grandes categorias:
  - Seleção entre 2 caminhos
  - Seleção entre múltiplos caminhjos

Sentenças de Seleção: 2 caminhos

- Forma geral:

```
if expressão_de_controle
        then sentença(s)
        else sentença(s)
```

- Questões a serem consideradas:
  - Forma e tipo da expressão de controle
  - Especificação das cláusulas **`then`** e **`else`**
  - Como seletores aninhados devem ser especificados/interpretados

A expressão de controle

- Varia de linguagem para linguagem
  - Consulte a documentação da linguagem para saber:
    - Parênteses são obrigatórios? Se podem ser opcionais, em quais situações são obrigatórios?
    - Chaves são obrigatórias? Se podem ser opcionais, em quais situação são obrigatórias?
    - A palavra **then** é obrigatória? A palavra **else** é obrigatória?
    - Indentação obrigatória ou opcional?
- Em geral pode ser aritmética, relacional ou booleana

Qual está correta?

```c
if (idade == 22) {
    printf("%s", "Dois patinhos na lagoa.");
} else {
    printf("%s", "Não são dois patinhos na lagoa.");
}
```

```bash
if [ $idade = 22 ];
then
  echo "Eu acho que sua idade"
  echo "são dois patinhos na lagoa."
  echo $(pwd)
else
  echo $(pwd)
  echo "Não sei bem sua idade."
fi
```

```python
if idade == 22:
    print("Sua idade é:")
    print("Dois patinhos na lagoa.")
else:
    print("Não sei sua idade.")
```

Cuidado com o aninhamento

- A cláusula “else” abaixo refere-se ao primeiro ou ao segundo “if”?

```
if (idade == 22)
    if (sexo == "F")
        resultado = 1;
else
    resultado = 2;
```

Em LISP/Scheme

```
(defun fat (n)
  (if (or (= n 1) (= n 0))
      1
      (* n (fat (- n 1)))))
```

Usos especiais

- Em algumas linguagens funcionais (LISP, ML, F#) as sentenças de seleção podem ser expressões utilizadas para retornar valores!

```
let y = if x > 0 then x
        else 2 * x
```

Sentenças de Seleção: N caminhos

- Permitem a seleção de 1 entre vários caminhos alternativos
- Questões a serem consideradas:
  1. Forma e tipo da sentença de controle
  2. Como especificar os segmentos alternativos?
  3. O fluxo de execução por meio da estrutura pode incluir apenas um segmento selecionável?
  4. Como os valores de cada caso são especificados?
  5. O que fazer com valores não representados nos casos selecionáveis?

Switch-case

- Diversas linguagens como C, C++, Java, JavaScript e outras (Python não!) oferecem a cláusula switch-case:

```
switch (expressão) {
  case expressao_constante1: sentenças1;
  ...
  case expressão_constanten: sentençasn;
  [default: sentençasn+1]
}
```

Switch-case

- Observações:
  1. As expressões de controle são constantes
  2. Os segmentos selecionáveis podem ser sentenças, blocos ou sentenças complexas
  3. **QUALQUER NÚMERO DE SEGMENTOS SELECIONÁVEIS** pode ser executado, não existe uma "saída" automática após a seleção. Isso é **FONTE COMUM DE ERROS**! Para isso, use o "**break**"!
  4. **`default`** serve para especificar a alternativa caso não haja nenhum segmento selecionável adequado para o caso. Se não houver um default e nenhum segmento selecionável "casar", então a estrutura de controle não faz nada
  5. Se diversos valores selecionam o mesmo segmento, não precisa repetir, basta informar somente o último segmento.

Switch-case

```c
#include <stdio.h>

int main(void) {
    int idade = 0;
    printf("%s", "Digite sua idade:\n");
    scanf("%i", &idade);
    switch (idade) {
        case 1:
            printf("%s", "Bebê");
        case 2:
        case 3:
        case 4:
        case 5:
            printf("%s", "Criança");
        case 15:
            printf("%s", "Adolescente");
        case 18:
            printf("%s", "Adulto");
    }
}
```

```
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
1
BebêCriançaAdolescenteAdulto
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
2
CriançaAdolescenteAdulto
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
5
CriançaAdolescenteAdulto
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
15
AdolescenteAdulto
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
18
Adulto
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
20

[abrantesasf@cosmos ~]$
```

Switch-case

```c
#include <stdio.h>

int main(void) {
    int idade = 0;
    printf("%s", "Digite sua idade:\n");
    scanf("%i", &idade);
    switch (idade) {
        case 1:
            printf("%s", "Bebê");
            break;
        case 2:
        case 3:
        case 4:
        case 5:
            printf("%s", "Criança");
            break;
        case 15:
            printf("%s", "Adolescente");
            break;
        case 18:
            printf("%s", "Adulto");
            break;
        default:
            printf("%s", "Não sei dizer");
            break;
    }
}
```

```
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
1
Bebê
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
2
Criança
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
5
Criança
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
15
Adolescente
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
18
Adulto
[abrantesasf@cosmos ~]$ ./teste
Digite sua idade:
20
Não sei dizer
[abrantesasf@cosmos ~]$
```

Switch-case

```
switch (x)
  default:
  if (prime(x))
    case 2: case 3: case 5: case 7:
      process_prime(x);
  else
     case 4: case 6: case 8: case 9: case 10:
       process_composite(x);
```

If – Else If – Else

- A seleção de N caminhos pode ser feita com cláusulas if – then – else if – else:

```c
#include <stdio.h>

int main(void) {
    int contagem = 0;
    int resultado1 = 0;
    int resultado2 = 0;
    int resultado3 = 0;
    int resultado4 = 0;

    if (contagem < 10) {
        resultado1 = 1;
    } else {
        if (contagem < 100) {
            resultado2 = 1;
        } else {
            if (contagem < 1000) {
                resultado3 = 1;
            } else {
                resultado4 = 1;
            }
        }
    }
}
```

```c
#include <stdio.h>

int main(void) {
    int contagem = 0;
    scanf("%d", &contagem);
    int resultado1 = 0;
    int resultado2 = 0;
    int resultado3 = 0;
    int resultado4 = 0;

    if (contagem < 10) {
        resultado1 = 1;
    } else if (contagem < 100) {
        resultado2 = 1;
    } else if (contagem < 1000) {
        resultado3 = 1;
    } else {
        resultado4 = 1;
    }

    printf("%d-%d-%d-%d", resultado1, resultado2,
                          resultado3, resultado4);
}
```

Seleção múltipla em LISP/Scheme

- Usa a expressão `COND`:

```
(COND
  (predicado_1 expressão_1)
   …
  (predicado_n expressão_n)
  [(ELSE expresssão_n+1)]
 )
```

  - A cláusula `ELSE` é opcional pois é um sinônimo para "true"
  - Cada par predicado-expressão é um parâmetro
  - O valor retornado é o valor do primeiro par predicado-expressão que for true

Sentenças de Iteração (loops)

- São sentenças cujo objetivo é repetir um bloco de código
  - Obs.: a repetição de um bloco de código também pode ser obtida por **recursão**!
- Duas grandes categorias de loops:
  - For
  - While
- Questões importantes:
  1. Como controlar a iteração?
     - Contador
     - Valor lógico
     - Estrutura de dados
  2. Se while, onde o mecanismo de controle da iteração deve aparecer no loop?

Loops controlados por contador

- Declaramos uma variável que será incrementada (ou decrementada) por um “passo”, de um valor inicial até um valor final

- Verificar a documentação da linguagem!

- Linguagens baseadas em C:

```
for ([expr_1] ; [expr_2] ; [expr_3]) sentenças
```

Loops controlados por contador

```c
int main(void) {
    for (int i = 1; i <= 10; ++i) {
        printf("%d\n", i);
    }
    return 0;
}
```

```c
int i = 1;
for (; i <= 10; ++i) {
    printf("%d\n", i);
}
```

```c
int i = 1;
for (; ; ++i) {
    printf("%d\n", i);
    if (i == 10) {
        break;
    }
}
```

```c
int contador = 1;
while (contador <= 10) {
    printf("%d\n", contador);
    contador++;
}
```

Loops controlados logicamente

- A repetição é controlada por um valor booleano

- Duas grandes categorias:
  - “Pré-teste”
  - “Pós-teste”

Loops controlados logicamente

```c
#include <stdio.h>

int main(void) {
    int continua;
    printf("%s\n", "Digite 0 ou 1:");
    scanf("%d", &continua);
    while (continua) {
        printf("%s\n", "Diversos comandos");
        printf("%s", "Digite 0 ou 1:");
        scanf("%d", &continua);
    };

    return 0;
}
```

Loops controlados logicamente

```c
#include <stdio.h>

int main(void) {
    int continua;
    printf("%s\n", "Digite 0 ou 1:");
    scanf("%d", &continua);
    do {
        printf("%s\n", "Diversos comandos");
        printf("%s", "Digite 0 ou 1:");
        scanf("%d", &continua);
    } while (continua);

    return 0;
}
```

Controle via continue e break

- Pode ser necessário tomar uma decisão de continuar ou parar o loop antes do local apropriado (início ou fim). Para isso:
  - continue
  - break

```
for (int i = 1; i <= 10; ++i) {
    if (i > 7) {
        break;
    } else if (i%2 == 1) {
        printf("%d\n", i);
    } else {
        continue;
    }
}
```

Loops baseados em estruturas de dados

- A quantidade de elementos na estrutura de dados controla a iteração do loop

- O mecanismo de controle é uma chamada a uma função "***iterator***" que retorna o próximo elemento. Se não houver um próximo elemento o loop termina

```
cores = ["amarelo", "azul", "branco", "verde", "vermelho"]
for i in cores:
    print(i)
```

```
List<String> names =  new  List<String>();
names.Add("Bob");
names.Add("Carol");
names.Add("Alice");
. . .
foreach  (String name  in  names)
  Console.WriteLine(name);
```

Desvio incondicional

- Uma sentença de desvio incondicional transfere o controle da execução para uma posição específica do programa. Em geral é feito pelo comando **goto**.
- Vantagens: poder, flexibilidade
- Desvantagens: perigoso, dificuldade de legibilidade, custo de manutenção
- Algumas linguagens suportam (C, C++, C#) e outras não suportam (Java, Python, Ruby)
- Usar? Briga de cachorro grande!
  - Edsger Dijkstra: NÃO
  - Donald Knuth: SIM

Desvio incondicional

```c
int main(void) {
    int contador = 1;
    inicio:
    if (contador <= 10) {
        printf("%d\n", contador);
        contador++;
        goto inicio;
    }
    return 0;
}
```

Comandos Protegidos

- Criados por Dijkstra para suportar uma nova metodologia de programação: **verificar a corretude do programa durante seu desenvolvimento**, e não ao verificar ou testar programas complexos
- Idéia básica:
  - Um segmento selecionável é independente de qualquer outra parte da sentença
  - Todas as expressões são avaliadas (a ordem não é importante e não deve ser especificada) e, se mais de uma expressão for verdadeira, uma das sentenças pode ser escolhida de maneira não determinística (a corretude do programa não depende de qual sentença é escolhida)

Comandos Protegidos

- Forma if:

```
if <Expressão Booleana> -> <sentença>
[] <Expressão Booleana> -> <sentença>
 ...
[] <Expressão Booleana> -> <sentença>
fi
```

- Semântica
  - Avalia todas as expressões, em qualquer ordem
  - Se mais de uma for verdade, escolha uma delas de modo não determinístico
  - Se nenhuma for verdade, retorna um erro

Comandos Protegidos

```
if (x >= y)
  max = x;
else
  max = y;
```

```
if   x >= y -> max := x
[] y >= x -> max := y
fi
```

Comandos Protegidos

- For do:

  do <Booleano> -> <sentença>
  [] <Booleano> -> <sentença>
  ...
  [] <Booleano> -> <sentença>
  od

- Semântica: para cada iteração
  - Avalia todas as expressões booleanas
  - Se mais de uma for verdadeira, escolha uma delas de forma não determinística e reinicia o loop
  - Se nenhuma for verdadeira, sai do loop

Comandos Protegidos

```
q1, q2, q3, q4 = 3, 1, 2, 0
lista = [q1, q2, q3, q4]
lista.sort()
q1, q2, q3, q4 = lista[0], lista[1], lista[2], lista[3]
```

```
do q1 > q2 -> temp := q1; q1 := q2; q2 := temp;
[] q2 > q3 -> temp := q2; q2 := q3; q3 := temp;
[] q3 > q4 -> temp := q3; q3 := q4; q4 := temp;
od
```

Comandos Progegidos: Por quê

- Conexão entre sentenças de controle e verificação do programa é íntima
- Verificação é impossível com sentenças `goto`
- Verificação é possível com seleção e loops lógicos
- Verificação é relativamente simples com comandos protegidos

Conclusões

- Loops são, quase sempre, “intercambiáveis”, ou seja, o que pode ser feito com FOR pode ser feito com WHILE ou com DO WHILE
  - E também com algoritmos recursivos
- Seleção e loops while com pré-teste resolvem quase todas as situações. Outras escolhas são um balanço entre facilidade e legibilidade
- Linguagens funcionais e lógicas utilizam estruturas de controle diferentes
- Leia a documentação da linguagem!