C++
Básico ao Avançado
Dando mais voltas

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

Escolha

Escolha

- Qual a diferença entre escolha e decisão?

Escolha

- Qual a diferença entre escolha e decisão?
- Problemas da escolha...

Escolha

- Qual a diferença entre escolha e decisão?
- Problemas da escolha...
- Porque então vou usar essa coisa?

Escolha

- Qual a diferença entre escolha e decisão?
- Problemas da escolha...
- Porque então vou usar essa coisa?

```
int A;
//...
if (A == 1)
{
  <comand1A>;
  <comand1B>;
  <comand1C>;
  <comand1D>;
  <comand1E>;
}
else
{
  if (A == 2) <comand2>;
  else
  {
    if (A == 3)
    {
      <comand3>;
      <comand4A>;
      <comand4B>;
    }
    else
    {
      if (A == 4)
      {
        <comand4A>;
        <comand4B>;
      }
      else <comand0>;
    }
  }
}
```

Não entendeu? Sem problemas

Não entendeu? Sem problemas

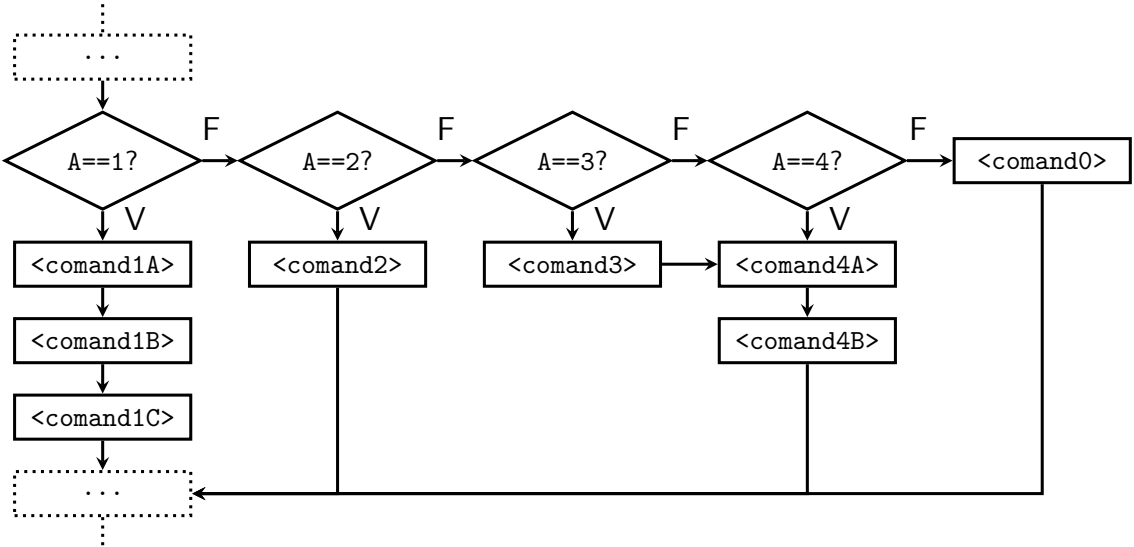

Figura: Fluxograma de esturura de decisão composta

O *switch*

O *switch*

- O que diabos foi aquilo?

O *switch*

- O que diabos foi aquilo??

O *switch*

- O que diabos foi aquilo???

O *switch*

- O que diabos foi aquilo???
- Como faço pra não ter aquilo?

O *switch*

- O que diabos foi aquilo???
- Como faço pra não ter aquilo?
- A estrutura de escolha:

O *switch*

- O que diabos foi aquilo???
- Como faço pra não ter aquilo?
- A estrutura de escolha: o `switch`

O *switch*

- O que diabos foi aquilo???
- Como faço pra não ter aquilo?
- A estrutura de escolha: o `switch`
- Recebe o argumento que precisar

O *switch*

- O que diabos foi aquilo???
- Como faço pra não ter aquilo?
- A estrutura de escolha: o `switch`
- Recebe o argumento que precisar
- Principal desvantagem:

O *switch*

- O que diabos foi aquilo???
- Como faço pra não ter aquilo?
- A estrutura de escolha: o `switch`
- Recebe o argumento que precisar
- Principal desvantagem: é binário

O *switch*

- O que diabos foi aquilo???
- Como faço pra não ter aquilo?
- A estrutura de escolha: o `switch`
- Recebe o argumento que precisar
- Principal desvantagem: é binário(?)

O *switch*

```
switch(A)
{
  case 1:
    <comand1A>;
    <comand1B>;
    <comand1C>;
    <comand1D>;
    <comand1E>;
    break;
  case 2:
    <comand2>;
    break;
  case 3:
    <comand3>;
  case 4:
    <comand4A>;
    <comand4B>;
    break;
  default:
    <comand0>;
}
```

Note os detalhes:

O *switch*

```
switch(A)
{
  case 1:
    <comand1A>;
    <comand1B>;
    <comand1C>;
    <comand1D>;
    <comand1E>;
    break;
  case 2:
    <comand2>;
    break;
  case 3:
    <comand3>;
  case 4:
    <comand4A>;
    <comand4B>;
    break;
  default:
    <comand0>;
}
```

Note os detalhes:

- `switch`

O *switch*

```
switch(A)
{
  case 1:
    <comand1A>;
    <comand1B>;
    <comand1C>;
    <comand1D>;
    <comand1E>;
    break;
  case 2:
    <comand2>;
    break;
  case 3:
    <comand3>;
  case 4:
    <comand4A>;
    <comand4B>;
    break;
  default:
    <comand0>;
}
```

Note os detalhes:

- `switch`
- `case`

O *switch*

```
switch(A)
{
  case 1:
    <comand1A>;
    <comand1B>;
    <comand1C>;
    <comand1D>;
    <comand1E>;
    break;
  case 2:
    <comand2>;
    break;
  case 3:
    <comand3>;
  case 4:
    <comand4A>;
    <comand4B>;
    break;
  default:
    <comand0>;
}
```

Note os detalhes:

- switch
- case
- break

O *switch*

```
switch(A)
{
  case 1:
    <comand1A>;
    <comand1B>;
    <comand1C>;
    <comand1D>;
    <comand1E>;
    break;
  case 2:
    <comand2>;
    break;
  case 3:
    <comand3>;
  case 4:
    <comand4A>;
    <comand4B>;
    break;
  default:
    <comand0>;
}
```

Note os detalhes:

- `switch`
- `case`
- `break`
- `default`

Não entendeu? Sem problemas

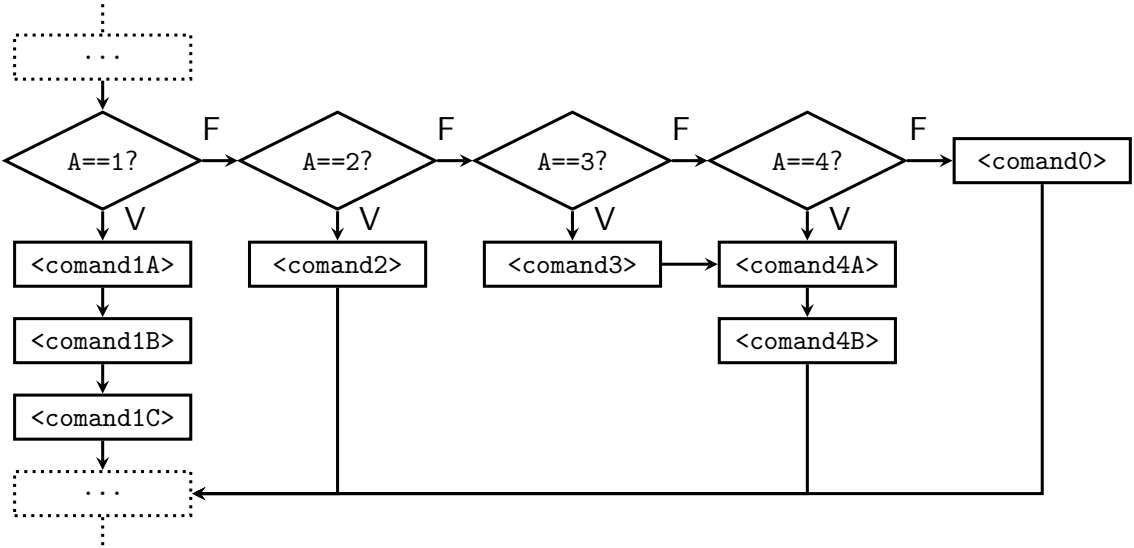

Figura: Fluxograma de esturura de decisão composta

Notas novamente

```
switch(conceito)
{
  case 'A':
  case 'B':
  case 'C':
  case 'D':
    aprovado = true;
    break;
  case 'F':
    aprovado = false;
    break;
}
```

Repetições definidas

Repetições definidas

- Utilizar o `while` pra repetição definida?

Repetições definidas

- Utilizar o `while` pra repetição definida?
- O `while` tem uma estrutura feita para repetição indefinida

Repetições definidas

- Utilizar o `while` pra repetição definida?
- O `while` tem uma estrutura feita para repetição indefinida
- Há uma estrutura feita só para estas situação

Repetições definidas

- Utilizar o `while` pra repetição definida?
- O `while` tem uma estrutura feita para repetição indefinida
- Há uma estrutura feita só para estas situação
- O fluxograma dela é idêntico ao do `while`

Repetições definidas

- Utilizar o `while` pra repetição definida?
- O `while` tem uma estrutura feita para repetição indefinida
- Há uma estrutura feita só para estas situação
- O fluxograma dela é idêntico ao do `while`

```
int i(0);
while(i<10) //Quantas iterações acontecem aqui?
{
  //...
  i++;
}
```

O *for*

O *for*

- Sintaxe simpática

O *for*

- Sintaxe simpática
- Todo programador gosta

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos(?)

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos(?)
- Iteração!

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos(?)
- Iteração! $\neq$ interação...

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos(?)
- Iteração! $\neq$ interação...
- Tem uma variável própria

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos(?)
- Iteração! $\neq$ interação...
- Tem uma variável própria em 95% das vezes...

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos(?)
- Iteração! $\neq$ interação...
- Tem uma variável própria em 95% das vezes...
- `++i` $\neq$ `i++`

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos(?)
- Iteração! $\neq$ interação...
- Tem uma variável própria em 95% das vezes...
- `++i` $\neq$ `i++`

```
for(<inic>; <cond>; <passo>) <comand>;
```

O *for*

- Sintaxe simpática
- Todo programador gosta
- Crucial para sequências de memórias
- Recebe três agrupamentos como argumentos(?)
- Iteração! $\neq$ interação...
- Tem uma variável própria em 95% das vezes...
- `++i` $\neq$ `i++`

```
for(<inic>; <cond>; <passo>) <comand>;
```

```
for(<inic>; <cond>; <passo>)
{
  <comand1>;
  //...
  <comandN>;
}
```

Alguns (muitos) detalhes

Alguns (muitos) detalhes

- A declaração da variável de controle pode ser feita na região de inicaliação

Alguns (muitos) detalhes

- A declaração da variável de controle pode ser feita na região de inicaliação
- Mais de uma variável pode ser criada e utilizada (todas do mesmo tipo)

Alguns (muitos) detalhes

- A declaração da variável de controle pode ser feita na região de inicaliação
- Mais de uma variável pode ser criada e utilizada (todas do mesmo tipo)
- As variáveis de iteração podem ser utilizadas dentro da estrutura de repetição

Alguns (muitos) detalhes

- A declaração da variável de controle pode ser feita na região de inicaliação
- Mais de uma variável pode ser criada e utilizada (todas do mesmo tipo)
- As variáveis de iteração podem ser utilizadas dentro da estrutura de repetição
- A condição de continuação não precisa ser atrelada as variáveis criadas

Alguns (muitos) detalhes

- A declaração da variável de controle pode ser feita na região de inicaliação
- Mais de uma variável pode ser criada e utilizada (todas do mesmo tipo)
- As variáveis de iteração podem ser utilizadas dentro da estrutura de repetição
- A condição de continuação não precisa ser atrelada as variáveis criadas
- O passo de iteração pode alterar todas as variáveis criadas de forma independente

Alguns (muitos) detalhes

- A declaração da variável de controle pode ser feita na região de inicaliação
- Mais de uma variável pode ser criada e utilizada (todas do mesmo tipo)
- As variáveis de iteração podem ser utilizadas dentro da estrutura de repetição
- A condição de continuação não precisa ser atrelada as variáveis criadas
- O passo de iteração pode alterar todas as variáveis criadas de forma independente
- Não é necessário inicializar a variável de iteração se esta já vem com um valor estabelecido (ou seja, que não é lixo de memória)

Alguns (muitos) detalhes

- A declaração da variável de controle pode ser feita na região de inicaliação
- Mais de uma variável pode ser criada e utilizada (todas do mesmo tipo)
- As variáveis de iteração podem ser utilizadas dentro da estrutura de repetição
- A condição de continuação não precisa ser atrelada as variáveis criadas
- O passo de iteração pode alterar todas as variáveis criadas de forma independente
- Não é necessário inicializar a variável de iteração se esta já vem com um valor estabelecido (ou seja, que não é lixo de memória)
- O passo de iteração não precisa alterar somente as variáveis de iteração

Alguns (muitos) detalhes

- A declaração da variável de controle pode ser feita na região de inicaliação
- Mais de uma variável pode ser criada e utilizada (todas do mesmo tipo)
- As variáveis de iteração podem ser utilizadas dentro da estrutura de repetição
- A condição de continuação não precisa ser atrelada as variáveis criadas
- O passo de iteração pode alterar todas as variáveis criadas de forma independente
- Não é necessário inicializar a variável de iteração se esta já vem com um valor estabelecido (ou seja, que não é lixo de memória)
- O passo de iteração não precisa alterar somente as variáveis de iteração
- As variáveis declaradas na região de inicialização são locais para o comando (ou bloco) da repetição

```cpp
int acc(0);
for (int i(0); i < 10; i++) acc += i; //Soma todos os números de 0 a 9

int A(13);
int B(-13);
for (int i(A); i > 0; i--)  B += 1;

for (; A > B; B++) ; //Note como o ponto-e-vírgula está sozinho, ou seja, o comando é vazio, porém B é incrementado

int sum(0);
for (int i(0), j(0); i < 10; i++) for (int k(0); k < 10; k++, j++) sum += i * j; //Aninhamento

long prod(1);
for (A = 25, B = 50; A < B; A++, B -= 2)
{
  prod *= B;
  prod /= A;
}
```

Saídas bruscas

Saídas bruscas

- Saídas bruscas são úteis

Saídas bruscas

- Saídas bruscas são úteis
- Existe um controle de fluxo para qubra de repetição

Saídas bruscas

- Saídas bruscas são úteis
- Existe um controle de fluxo para qubra de repetição
- Existe um controle de fluxo para pular iteração

Saídas bruscas

- Saídas bruscas são úteis
- Existe um controle de fluxo para qubra de repetição
- Existe um controle de fluxo para pular iteração
- Associação a `if`

Saídas bruscas

- Saídas bruscas são úteis
- Existe um controle de fluxo para qubra de repetição
- Existe um controle de fluxo para pular iteração
- Associação a `if`
- Se não existir `if`, existe um problema

Saídas bruscas

- Saídas bruscas são úteis
- Existe um controle de fluxo para qubra de repetição
- Existe um controle de fluxo para pular iteração
- Associação a `if`
- Se não existir `if`, existe um problema
- Você já viu um deles

Saídas bruscas

- Saídas bruscas são úteis
- Existe um controle de fluxo para qubra de repetição
- Existe um controle de fluxo para pular iteração
- Associação a `if`
- Se não existir `if`, existe um problema
- Você já viu um deles

```
continue; //Normalmente são utilizados com um if

break;
```

```
1  int A(100), B(5);
2  //...
3  while (A>-50)
4  {
5    if(B==0)
6    {
7      B -=1;
8      continue;
9    }
10   A/=B;              //Note como a necessidade do else é suprida
11   if(A==B) break;
12   A+=B-=5;
13 }
```

Vamos testar!