# Estruturas Condicionais

## Introdução

Vamos reconsiderar um programa bem simples em C. No início ele pergunta ao usuário o raio do círculo e com esta informação calcula a área e o perímetro da figura.

```
#include <stdio.h>
#include <stdlib.h>

/*
 * Este programa pergunta o raio de um círculo.
 * O programa calcula então a área e o perímetro do círculo.
 */
int main(int argc, char* argv[]) {
     // Declarar variáveis
     double pi = 3.141592;
     double raio, area, perimetro;

     // Pedir ao usuário escrever o raio
     printf("Digite o raio: ");
     scanf("%lf", &raio);

     // Cálculos
     area = pi * (raio * raio);
     perimetro = 2.0 * pi * raio;

     // Imprimir resultados
     printf("\n");
     printf("Raio: %f \n", raio);
     printf("Area: %f \n", area);
     printf("Perimetro: %f \n", perimetro);

     return 0;
}
```

*Consulte: EstruturasCondicionais\AreaPerimetro01\AreaPerimetro01.vcproj*

Este é um exemplo típico de um programa que executa o bloco da função `main`, uma sentença após a outra, até finalizar no comando `return`. Programas seqüenciais como este se limitam a algoritmos muito simples.

Neste capítulo, estudaremos construções da linguagem C que permitem estabelecer condições que controlam se um determinado trecho de código é executado ou não. O programa poderia, por exemplo, analisar o valor de uma variável ou o resultado de uma expressão para determinar quais comandos executar em seguida. Por exemplo, se o valor do raio que for lido é negativo, o programa deveria parar com uma mensagem de erro, caso contrário deveria calcular a área e o perímetro do círculo.

## Condições

Antes de iniciar o estudo das estruturas condicionais, vamos entender como C descreve condições e como elas são avaliadas durante a execução do programa.

Em C, uma **condição** é nada mais que qualquer expressão, cujo valor resultante precisa ser necessariamente um número inteiro. Se esse valor for `0` (zero), então a expressão é

  

interpretada como sendo *falsa*. Caso contrário, se a expressão resultar em qualquer outro valor não nulo, então a condição será tratada como *verdadeira*.

> Uma condição é uma expressão cujo resultado deve ser um inteiro (zero ou não zero). Na linguagem C, o conceito *falso*/*verdadeiro* se confunde com *zero*/*não zero*.

## Operadores de Comparação

A maioria das decisões de um programa é baseada na comparação entre dois valores. Para escrever tais condições, existem os operadores de comparação (também chamados de operadores relacionais).

Os operadores de comparação são tratados da mesma forma que os demais operadores aritméticos, tais como `+`, `-`, `/` e `*` (respectivamente, soma, subtração, divisão e multiplicação). Para escrever comparações, valem as mesmas regras usadas para expressões aritméticas.

O resultado de um operador de comparação será **1** se a comparação for verdadeira, ou **0** caso ela seja falsa. Então temos:

| Expressão | Resultado |
|---|---|
| `valor1 < valor2` | Resulta **1** se `valor1` é menor que `valor2`, caso contrário, resulta em **0**. |
| `valor1 <= valor2` | Resulta **1** se `valor1` é menor ou igual que `valor2`, caso contrário, resulta em **0**. |
| `valor1 > valor2` | Resulta **1** se `valor1` é maior que `valor2`, caso contrário, resulta em **0**. |
| `valor1 >= valor2` | Resulta **1** se `valor1` é maior ou igual que `valor2`, caso contrário, resulta em **0**. |
| `valor1 == valor2` | Resulta **1** se `valor1` é exatamente igual que `valor2`, caso contrário, resulta em **0**. |
| `valor1 != valor2` | Resulta **1** se `valor1` é diferente que `valor2`, caso contrário, resulta em **0**. |

### Observações importantes

Note que estes operadores têm o mesmo significado que na notação matemática. No entanto, C adota uma representação um pouco diferente para eles.

Os operadores "menor/maior que ou igual a" (≤ e ≥) são escritos, respectivamente, usando-se os símbolos "`<`" e "`>`", seguidos do símbolo "`=`". O operador "diferente de" (≠), é escrito

  

usando-se o símbolo de exclamação ("`!`"), seguido pelo símbolo igual ("`=`"). Note que os símbolos ≤ , ≥ e ≠ não estão disponíveis em teclados comuns.

A igualdade merece atenção especial. Observe que o operador é formado por dois símbolos de igualdade consecutivos ("`==`").

**Cuidado:** O símbolo "`=`" simples significa atribuição. A troca entre operadores "`=`" e "`==`" não necessariamente será detectada pelo compilador. **Essa é uma fonte de erros corriqueira para os iniciantes nas linguagens C, C++, C#, Java e outras semelhantes**.

**Exemplos de expressões de comparação**

Existe um número ilimitado de possibilidades para escrever expressões de comparação. Vamos analisar alguns exemplos:

`2 < 5` A mais simples delas, simplesmente **compara duas constantes**. Este tipo de expressão é de pouco interesse, pois esta comparação sempre gera o mesmo resultado. Nesse caso, gera `1`.

`a >= 5` **Comparação entre uma constante e o valor de uma variável**. Esta expressão é mais interessante, pois com ela um programa pode conhecer algo sobre o valor da variável.

`a != b` **Comparação entre os valores de duas variáveis**. Pode-se decidir dependendo dos valores das duas variáveis.

`(nota1 + nota2) / 2 >= 5.0`

**Comparação entre o resultado de uma expressão aritmérica e uma constante**. Nesse caso, por exemplo, o lado esquerdo calcula a média de duas provas e o lado direito representa a nota de corte.

**Comparação entre duas expressões**. Esta é a forma mais geral, na qual o programa decide comparando o resultado de duas expressões.

  

**Exemplos do resultado das expressões de comparação**

```
int a, b;
...

a = (1 < 2);
b = (3 <= 2);
```

Às variáveis `a` e `b` são atribuídos os resultados das expressões de comparação entre os parênteses. Eles são necessários para indicar que a comparação será realizada antes da atribuição.

Após a execução, `a` recebe 1 (pois 1 < 2 é verdadeiro)

e `b` recebe 0 (pois 3 ≤ 2 é falso).

```
int c = 3;
int d = 10;
int e, f;
...
e = (c == d);
f = (c != d);
```

Este exemplo ilustra a comparação entre o valor de duas variáveis `c` e `d`.

Após a execução, `e` recebe 0 (pois 3 = 10 é falso)

e `f` recebe 1 (pois 3 ≠ 10 é verdadeiro).

## Estrutura Condicional `if`

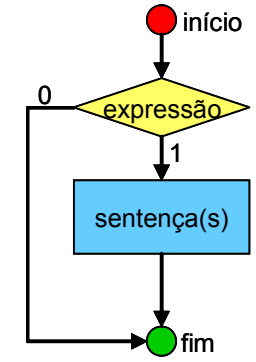


**Figura 1 - Fluxo da estrutura condicional `if`**

**Sintaxe:**

```
if (expressão) {
    sentença;
    sentença;
    ...
}
```

**Figura 2 – Sintaxe da estruturas condicional `if`**

A construção **`if`** executa uma sentença ou bloco de sentenças somente se uma determinada condição for verdadeira.

A Figura 2 ilustra a sintaxe da estrutura condicional **`if`** em C. Ela controla se o próximo bloco de sentenças deve ser executado ou não. O bloco de sentenças é delimitado pelas chaves ("`{`" e "`}`"). O fluxo de execução desta estrutura está ilustrado na Figura 1.

Primeiro, avalia-se a expressão, que representa a condição a ser verificada. Note que esta expressão é obrigatoriamente envolvida por parênteses. Muito possivelmente, essa expressão utilizará um dos operadores de comparação.

  

Se a expressão resultar em um valor diferente de zero (verdadeiro), então o programa executa o bloco de sentenças e depois continua normalmente com próxima sentença logo após o bloco do **`if`**.

Mas se a expressão resultar em zero (falso), então todo o bloco de sentenças é ignorado e a execução segue diretamente para o primeiro comando após o bloco de sentenças do **`if`**.

**Observação 1:** A condição deve ser escrita obrigatoriamente entre parênteses.

**Observação 2:** Note que as sentenças dentro do bloco da construção **`if`** estão deslocadas para a direita, usando tabulações. Isto é uma boa prática de programação, pois a disposição do código refletirá a estrutura lógica do programa.

### Exemplo

Este programa pede para o usuário digitar sua idade. Baseando-se nessa informação, ele imprime se a pessoa pode obter sua carteira de habilitação.

**Código fonte:**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
     // Declarar variáveis
     int idade;

     // Pedir ao usuário para escrever sua idade
     printf("Digite sua idade: ");
     scanf("%d", &idade);

     // Comparar com a idade mínima
     if (idade >= 18) {
          printf("Ja pode obter habilitacao!");
     }

     return 0;
}
```

*Consulte: EstruturasCondicionais\Idade01\Idade01.vcproj*

**Descrição passo a passo:**

```
int idade;
```

Primeiro, declara-se a variável que recebe o valor da idade digitada pelo usuário. A idade é declarada como um número inteiro, mas seu valor inicial não foi atribuído.

```
printf("Digite sua idade: ");
scanf("%d", &idade);
```

Em seguida, o comando `printf` imprime uma mensagem para solicitar a idade do usuário, e `scanf` realiza a leitura de um número inteiro na variável `idade`.

```
if (idade >= 18) {
      printf("Ja pode obter habilitacao!");
}
```

Por fim, o valor da variável `idade` é comparado com o valor `18`. Se o resultado da expressão (`idade >= 18`) for verdadeiro (diferente de zero), então o próximo bloco é executado. Este bloco contém apenas o comando `printf` que imprime a

  

autorização. Caso contrário, a execução salta este bloco e passa para o comando `return 0`, que finaliza o programa.

**Primeiro exemplo de execução:**

```
Digite sua idade: 15
```

Neste caso a autorização não é impressa, pois o valor da idade é menor que 18.

**Segundo exemplo de execução:**

```
Digite sua idade: 25
Ja pode obter habilitacao!
```

**Sintaxe:**

```
if (expressão)
    sentença;
```

**Figura 3 – Sintaxe abreviada da estrutura condicional `if`**

## Variação da estrutura condicional `if`

Em alguns casos, o bloco de uma estrutura **`if`** contém apenas uma única sentença. Neste caso particular, pode-se omitir as chaves que delimitam o bloco. No entanto, recomenda-se evitar esta forma abreviada, pois sem as chaves que delimitam o bloco, é difícil associar visualmente se a sentença pertence ao **`if`** ou ao programa que o envolve. Além disso, se, futuramente, quisermos inserir outras sentenças no bloco e esquecermos as chaves, o código, provavelmente, vai produzir resultados inesperados.

## Expressões sem operadores de comparação

Alguns programadores experientes em C tentam evitar ao máximo o uso de operadores de comparação nas condições de um **`if`**. Por exemplo, para verificar se o valor de uma variável inteira é diferente de zero, poderíamos utilizar nossos conhecimentos deste capítulo para escrever a seguinte expressão em uma estrutura **`if`**:

```
if (variavel != 0) {
    ...
}
```

A mesma condição poderia ser escrita explorando o fato do valor não nulo ser interpretado como verdadeiro:

```
if (variavel) {
    ...
}
```

Note que, se a variável apresentar qualquer valor diferente de zero, o **`if`** concluirá que esse valor é não nulo e, por conseqüência, o interpretará como verdadeiro. O bloco é então executado nas mesmas condições que quando utilizamos a expressão `(variavel != 0)`.

A motivação de lançar mão deste recurso é um tanto questionável. O código fonte é reduzido em alguns caracteres e o programa realizará uma comparação a menos ao executar. Por conseqüência, o programa resultante será algo mais compacto e mais rápido.

Porém, o objetivo das linguagens de programação é escrever algoritmos da forma mais clara possível. É duvidoso se compensa tornar o código mais obscuro para economizar alguma memória no código objeto e não mais que frações ínfimas de segundo de processamento. Ademais, todos os compiladores modernos automaticamente otimizam o código de forma a tornar desnecessárias tais preocupações de eficiência.

  

> Neste curso, adotaremos sempre o código mais claro e legível ao invés do mais otimizado.

## Estrutura Condicional `if...else`

Vamos continuar estudando o exemplo anterior, o qual decide se a idade do usuário lhe permite tirar carteira de habilitação. Desejamos agora imprimir uma mensagem negando a autorização, caso a pessoa seja menor de idade. Basta adicionar uma segunda estrutura condicional **`if`** para verificar o caso oposto (menor que 18 anos).

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
     int idade;

     printf("Digite sua idade: ");
     scanf("%d", &idade);

     // Comparar com a idade mínima
     if (idade >= 18) {
          printf("Você ja pode obter habilitacao!");
     }
     if (idade <= 17) {
          printf("Espere mais alguns anos!");
     }
return 0;
```

*Consulte: EstruturasCondicionais\Idade02\Idade02.vcproj*

Note que, pela lógica do programa, se a condição da expressão do primeiro **`if`** for avaliada como verdadeira, então, obrigatoriamente, a expressão do segundo **`if`** é necessariamente falsa e vice-versa. Ou seja, as duas condições são mutuamente exclusivas e apenas um dos blocos das estruturas condicionais do **`if`** é executado.

Para estas situações, a linguagem oferece a estrutura condicional **`if...else`**, que executa exatamente um dentre dois blocos de sentenças, dependendo do resultado de uma determinada condição.

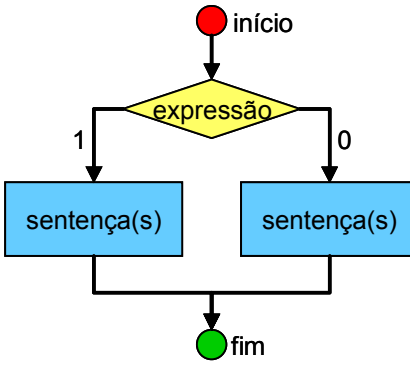


**Figura 4 – Fluxo da estrutura condicional `if...else`**

**Sintaxe:**

```
if (expressão) {
     sentença;
     sentença;
     ...
} else {
     sentença;
     sentença;
     ...
}
```

**Figura 5 – Sintaxe da estrutura condicional `if...else`**

  

A Figura 5 ilustra a sintaxe da estrutura condicional **`if...else`** em C. Os dois blocos de sentenças são delimitados por chaves ("`{`" e "`}`"). O fluxo de execução desta estrutura está ilustrado na Figura 64.

**Sintaxe:**

```
if (expressão)
    sentença;
else
    sentença;
```

**Figura 6 – Sintaxe abreviada da estrutura condicional `if...else`**

Primeiro, a expressão que representa a condição é avaliada. Note que esta expressão é obrigatoriamente envolvida por parênteses. Se o resultado da expressão for um valor diferente de zero (verdadeiro), então apenas o primeiro bloco de sentenças é executado. Caso contrário, se a expressão revelar-se igual a zero (falso), então apenas o segundo bloco de sentenças é executado. Independentemente do resultado da expressão, depois de executar um dos dois blocos de sentenças, o programa continua normalmente com a próxima sentença ou comando, logo após a estrutura **`if`**. Quando o bloco da estrutura **`if...else`** contém apenas uma única sentença, pode-se omitir as chaves que delimitam o bloco. Novamente, recomenda-se evitar esta forma abreviada devido à falta de clareza. Veja a Figura 6.

## Exemplo

O próximo programa informa se uma pessoa pode ou não obter carteira de habilitação. Caso positivo, imprime há quantos anos a pessoa já tem direito à carteira, caso contrário, imprime quantos anos a pessoa ainda precisa esperar.

**Código fonte:**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
     // Declarar variáveis
     int idade;
     int diferenca_tempo;

     // Pedir ao usuário para escrever sua idade
     printf("Digite sua idade: ");
     scanf("%d", &idade);

     // Comparar com a idade mínima
     if (idade >= 18) {
          diferenca_tempo = idade - 18;
          printf("Voce ja pode tirar habilitacao!\n");
          printf("Voce tem direito a habilitacao ha %d ano(s)",
               diferenca_tempo);
     } else {
          diferenca_tempo = 18 - idade;
          printf("Voce precisa esperar mais %d ano(s)!\n",
               diferenca_tempo);
     }

     return 0;
}
```

*Consulte: EstruturasCondicionais\Idade03\Idade03.vcproj*

**Descrição passo a passo:**

```
int idade;
int diferenca_tempo;
```

  

Primeiro, declara-se a variável que recebe o valor da idade digitada pelo usuário. A segunda variável, `diferenca_tempo`, também declarada como um número inteiro, será utilizada somente mais adiante, quando for necessário calcular o tempo em relação aos 18 anos. No entanto, C exige que todas as variáveis sejam declaradas no início do bloco.

```
printf("Digite sua idade: ");
scanf("%d", &idade);
```

Em seguida, o comando `printf` imprime a mensagem na tela, e `scanf` realiza a leitura de um número inteiro armazenando-o na variável `idade`.

```
if (idade >= 18) {
  diferenca_tempo = idade - 18;
  printf("Voce ja pode tirar habilitacao!\n");
  printf("Voce tem direito a habilitacao ha %d ano(s)", diferenca_tempo);
} ...
```

A expressão (`idade >= 18`) é avaliada. Caso seja verdadeira, o primeiro bloco da estrutura condicional **`if..else`** é executado. O primeiro bloco calcula quantos anos se passaram desde os 18 anos e armazena esse valor na variável `diferenca_tempo`. Depois, imprime uma mensagem de autorização e, em seguida, imprime uma segunda mensagem informando o tempo decorrido desde os 18 anos.

```
... else {
      diferenca_tempo = 18 - idade;
      printf("Voce precisa esperar mais %d ano(s)!", diferenca_tempo);
}
```

Caso contrário, se a expressão `idade >= 18` for falsa, então o segundo bloco é executado. Ele calcula a diferença entre a idade e 18 anos, e armazena esse valor na variável `diferenca_tempo`. Em seguida, o valor dessa variável é impresso.

**Primeiro exemplo de execução:**

```
Digite sua idade: 25
Voce ja pode tirar habilitacao!
Voce tem direito a habilitacao há 7 ano(s)
```

**Segundo exemplo de execução:**

```
Digite sua idade: 15
Voce precisa esperar mais 3 ano(s)!
```

## Operadores lógicos

No início do capítulo, estudamos os operadores de comparação, também denominados operadores relacionais. Eles permitem descrever condições envolvendo comparação entre valores tais como números, variáveis e resultados de expressões.

Mas como proceder quando desejamos tomar decisões sobre uma combinação de condições? No exemplo da carteira de habilitação, não basta apenas apresentar idade maior ou igual a 18 anos, mas também é necessário ser aprovado nos exames. Este é um exemplo de duas condições que precisam ser verdadeiras simultaneamente.

  

Um outro exemplo é a admissão em um cargo de boa remuneração: a pessoa apresenta um excelente currículo profissional ou conhece alguém influente disposto a indicá-la. Este é um caso onde apenas uma das condições precisa ser verdadeira.

Para se escrever tais condições em C, podemos usar os operadores lógicos, que unem duas expressões em uma condição maior e mais complexa. Lembre-se que C interpreta o valor 0 como falso e os demais valores como verdadeiro. Nos casos abaixo, `cond1` e `cond2` representam expressões cujos resultados são números inteiros.

| Idéia | Expressão | Resultado |
|---|---|---|
| **E (AND)** | `(cond1) && (cond2)` | Resulta **1** se `cond1` e `cond2` forem verdadeiras (não nulas). Se uma das duas for falsa (zero), então resulta em **0**. |
| **OU (OR)** | `(cond1) \|\| (cond2)` | Resulta **1** se `cond1` ou `cond2` forem verdadeiras (não nulas). Sem as duas forem falsas (zero), então resulta em **0**. |

O comportamento destes dois operadores pode ser ilustrado pelas seguintes tabelas:

| && (E) | | cond1 | |
|---|---|---|---|
| | | **falso** | **verdade** |
| **cond2** | **falso** | *falso* | *falso* |
| | **verdade** | *falso* | *verdade* |

| \|\| (OU) | | cond1 | |
|---|---|---|---|
| | | **falso** | **verdade** |
| **cond2** | **falso** | *falso* | *verdade* |
| | **verdade** | *verdade* | *verdade* |

Os operadores lógicos são tratados da mesma forma que os demais operadores aritméticos e de comparação. Lembre que as expressões envolvendo qualquer um deles resultam em 0 ou 1. A linguagem C não distingue a natureza das expressões.

Além dos operadores (tais como `||` e `&&`) existe um terceiro, muito útil, e que pode ser usado para inverter o significado das expressões: é o operador `!`.

| Idéia | Expressão | Resultado |
|---|---|---|
| **NÃO (NOT)** | `! (condicao)` | Resulta **1** (verdadeiro) se `condicao` for falso (zero). Caso contrário, resulta em **0** (falso). |

A tabela abaixo ilustra o comportamento do operador:

| ! (NÃO) | Resultado | |
|---|---|---|
| condicao | **falso** | **verdade** |
| (! condicao) | *verdade* | *Falso* |

**Observação 1:** Ao unir duas condições em uma expressão maior com os operadores `&&` ou `||`, escreve-se cada uma das duas condições entre parênteses. Isto evita ambigüidades para interpretar a expressão composta.

  

**Observação 2:** O compilador C não necessariamente realiza todos os testes de uma expressão quando o resultado for evidente. No caso do operador **&&** (e), se a primeira expressão for falsa, então a condição toda será falsa, independente da segunda expressão. Neste caso, o compilador *poderá* decidir por não avaliar a segunda expressão, já que ela não afetará o resultado.

O operador **||** (ou) tem um comportamento similar: se a primeira expressão for verdadeira, então a condição toda será também verdadeira, sem ser necessário avaliar a segunda expressão.

**Exemplos:**

Suponha duas variáveis, `media` e `freqüência`, que indicam a nota final na disciplina e o número de aulas freqüentadas por um aluno. Uma terceira variável, `aprovado`, armazena o resultado da expressão (que pode ser *verdadeiro* ou *falso*, ou seja, `1` ou `0`).

Professores muito severos programariam a condição da seguinte forma:

```
aprovado = (nota >= 7.0) && (frequencia >= 40)
```

Ou seja, para que à variável `aprovado` seja atribuído o valor 1, será necessário satisfazer tanto `(nota >= 7.0)` como também `(frequencia >= 40)`. O aluno não só precisa de nota maior ou igual a 7.0, como necessita assistir no mínimo 40 aulas!

Já outros professores preferem adotar uma política mais flexível:

```
aprovado = (nota >= 5.0) || (frequencia >= 30)
```

Assim, é necessário satisfazer apenas uma condição: `(nota >= 5.0)` ou `(frequencia >= 30)`. Basta obter uma boa média (mesmo faltando à maioria das aulas), ou apenas assistir a maioria das aulas. Satisfazendo qualquer uma das condições, o aluno estará aprovado.

Em casos ainda mais complicados, alguns professores gostam do seguinte critério de aprovação:

```
aprovado = (nota >= 9.0) || ((nota >= 5.0) && (frequencia >= 30))
```

A condição é a combinação de três expressões. Para se aprovar, o aluno precisa satisfazer a primeira condição `(nota >= 9.0)`, ou a segunda condição `((nota >= 5.0) && (frequencia >= 30))`. Note que a segunda condição, por sua vez, é composta por duas condições ligadas pelo operador lógico **&&** (**E**). Isto significa que o aluno é aprovado com média maior ou igual a nove (independente do número de aulas que assistir), ou precisa de média no mínimo cinco e precisa também assistir pelo menos 30 aulas.

### Exemplo:

Um programa para informar se uma pessoa pode ou não obter a carteira de habilitação. A pessoa precisa ser maior de idade e apresentar média acima da nota mínima nos exames.

**Código fonte:**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
     // Declarar variáveis
     int idade;
     float media;

     // Pedir para escrever idade e a média dos exames
     printf("Digite sua idade: ");
```

  

```
        scanf("%d", &idade);
        printf("Digite sua media nos exames: ");
        scanf("%f", &media);

        // Comparar com a idade mínima
        if ( (idade >= 18) && (media >= 5.0) ) {
            printf("Voce está aprovado!");
        } else {
            printf("Ainda nao aprovado!");
        }

        return 0;
    }
```

*Consulte: EstruturasCondicionais\Idade04\Idade04.vcproj*

**Descrição passo a passo:**

```
int idade;
float media;
```

Começamos declarando duas variáveis, que armazenam os dados digitados pelo usuário. Em seguida, os valores são lidos através dos dois comandos `scanf`.

```
if ( (idade >= 18) && (media >= 5.0) ) ...
```

Neste momento, as condições são avaliadas. Preste atenção na construção desta condição composta por duas expressões `(idade >= 18)` e `(media >= 5.0)`. Cada uma das expressões está envolvida por parênteses. Entre as duas expressões está o operador `&&`, indicando que ambas precisam ser verdadeiras.

Toda a condição está novamente envolvida com um segundo par de parênteses, conforme exigido pela sintaxe da estrutura condicional `if...else`.

```
... {
        printf("Voce esta aprovado!");
} else {
        printf("Ainda nao aprovado!");
}
```

Por fim, caso a condição seja verdadeira, então o primeiro bloco é executado (imprimindo a mensagem de aprovação), caso contrário, o segundo bloco será executado (escrevendo a mensagem de rejeição).

**Primeiro exemplo de execução:**

```
Digite sua idade: 20
Digite sua media nos exames: 3.5
Ainda nao aprovado!
```

**Segundo exemplo de execução:**

```
Digite sua idade: 20
Digite sua media nos exames: 7.5
Voce esta aprovado!
```

## Estruturas condicionais IF...ELSE IF...ELSE

Chegou o momento de evoluir ainda mais um pouco nosso exemplo, utilizando estruturas condicionais quer permitem mais de uma decisão.

  

Vamos continuar estudando o primeiro exemplo deste capítulo, que decide se a idade do usuário permite ou não obter carteira de habilitação. Além de avaliar se a pessoa é maior de idade, agora desejamos verificar se ela tem mais de 65 anos (e precisa fazer exames regulares a cada 3 anos).

A linguagem C oferece a estrutura condicional **`if...else if...else`**, com capacidade de realizar decisões múltiplas. A estrutura oferece vários blocos como alternativas para serem executados, cada qual associado à sua própria expressão condicional.

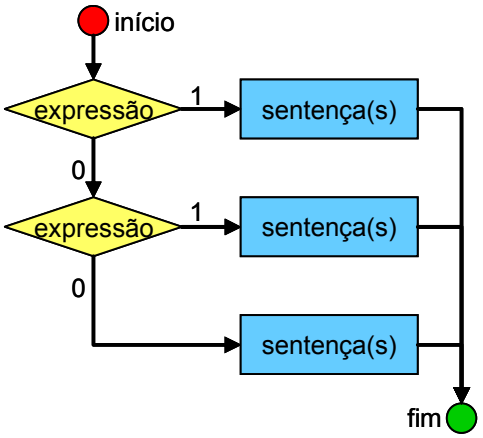


**Figura 7 – Fluxo da estrutura condicional** `if...else if...else`

**Sintaxe:**

```
if (expressão) {
    sentenças;
    ...
} else if (expressão) {
    sentenças;
    ...
} else {
    sentenças;
    ...
}
```

**Figura 8 – Sintaxe da estrutura condicional** `if...else if...else`

A Figura 8 mostra a sintaxe desta estrutura. Ela executa somente o primeiro o bloco se a primeira expressão é satisfeita, ignorando os demais blocos. Depois, a execução do programa continua normalmente com a primeira sentença após toda esta estrutura. As expressões são avaliadas em ordem, de cima para baixo. Se uma expressão resultar em zero (falso), então a próxima expressão é avaliada. Quando a expressão for diferente de zero (verdadeiro), então apenas o bloco correspondente a ela é executado, sendo as demais expressões ignoradas. A Figura 7 ilustra o fluxo de execução desta estrutura.

**Observação 1:** A estrutura pode apresentar tantos blocos **`else if`** quantos forem necessários para descrever a lógica de um algoritmo. Tanto a Figura 7 como a Figura 8 mostram um caso particular com apenas 2 condições.

**Observação 2:** O último bloco, associado com o **`else`**, é executado quando nenhuma das outras condições for verdadeira. Este bloco é opcional e pode ser omitido.

### Exemplo:

Um programa para informar o período de renovação dos exames da carteira de habilitação. Menores de idade não possuem carteira. Até 65 anos, os prazos são de 5 em 5 anos; depois, o exame precisa de renovação a cada 3 anos.

**Código fonte:**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
     // Declarar variáveis
     int idade;
```

  

```
        // Pedir ao usuário para escrever sua idade
        printf("Digite sua idade: ");
        scanf("%d", &idade);

        // Comparar com a idade mínima
        if ( (idade >= 0) && (idade < 18) ) {
             printf("Nao possui carteira de habilitacao.\n");
        } else if ( (idade >= 18) && (idade < 65) ) {
             printf("Renove exames a cada 5 anos.\n");
        } else if (idade >= 65) {
             printf("Renove exames a cada 3 anos.\n");
        }
        return 0;
}
```

*Consulte: EstruturasCondicionais\Idade05\Idade05.vcproj*

**Descrição passo a passo:**

```
int idade;
```

Começamos declarando a variável inteira chamada `idade` que armazena o valor digitado pelo usuário.

```
if ( (idade >= 0) && (idade < 18) ) {
      printf("Nao possui habilitacao.\n");
} ...
```

A primeira condição é avaliada. Ela é formada por duas expressões, unidas pelo operador **&&** (e). Ambas precisam ser satisfeitas para se executar este bloco. Se `idade` for maior ou igual a zero e menor que 18, o bloco é executado imprimindo a mensagem. As demais condições serão ignoradas.

```
... else if ( (idade >= 18) && (idade < 65) ) {
      printf("Renove exames a cada 5 anos.\n");
}
```

Se a primeira condição for falsa, então a segunda será avaliada. Ela também é formada por duas expressões, unidas pelo operador **&&** (e). Se `idade` for maior ou igual a 18 e menor que 65, o bloco é executado imprimindo a mensagem.

```
...else if (idade >= 65) {
      printf("Renove exames a cada 3 anos.\n");
}
```

Se a primeira e a segunda condição forem falsas, então a terceira será avaliada. Se a `idade` for maior ou igual a 65, o bloco é executado, imprimindo a mensagem.

**Observação:**

Vamos analisar novamente a condições dos primeiros dois blocos:

```
if ( (idade >= 0) && (idade < 18) ) ...
... else if ( (idade >= 18) && (idade < 65) ) ...
```

Note que se a idade for menor que 18, então a condição do primeiro bloco será satisfeita, e as demais condições serão ignoradas. Por este motivo, a expressão `(idade >= 18)` do segundo bloco é desnecessária, pois se a execução chegar ao segundo bloco, isto significa que `(idade < 18)` não pode ser verdade.

  

**Primeiro exemplo de execução:**

```
Digite sua idade: 15
Nao possui carteira de habilitacao.
```

**Segundo exemplo de execução:**

```
Digite sua idade: 25
Renove exames a cada 5 anos.
```

**Terceiro exemplo de execução:**

```
Digite sua idade: 75
Renove exames a cada 3 anos.
```

## SWITCH

A estrutura condicional **`switch`** possui uma lógica de funcionamento muito diferente de todas as variantes das estruturas **`if`** que estudamos até agora. Ela não avalia diversas condições para determinar qual opção executar, tal como era o caso das estruturas **`if`**.

O **`switch`** é uma construção de múltiplas possibilidades de decisão. Ele compara o resultado de uma expressão com uma série de valores constantes. A sintaxe desta estrutura está na Figura 10.

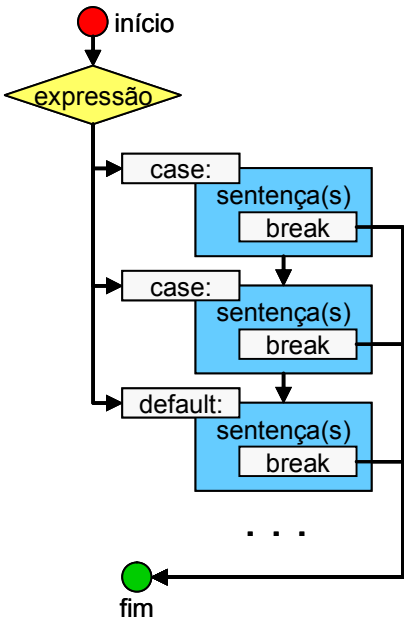


**Figura 9 – Fluxo da estrutura condicional `switch`**

**Sintaxe:**

```
switch (expressão) {
   case valor1:
      sentenças;
      ...
   case valor2:
      sentenças;
      ...
   case valor3:
      sentenças;
      ...
   default:
      sentenças;
      ...
}
```

**Figura 10 – Sintaxe da estrutura condicional `switch`**

Primeiro, o comando **`switch`** avalia a expressão. Em seguida, começa a comparar o resultado da expressão com o valor de cada uma das possibilidades marcadas pela palavra reservada **`case`**, na sequência dada. Cada possibilidade deve ser um número inteiro (tipo *`int`*, *`long`*, *`short`* e variantes) ou um caractere (tipo *`char`* e variantes).

Quando encontra uma das possibilidades, listadas logo após a palavra **`case`**, que seja idêntica ao valor da expressão, o programa começa a executar as sentenças correspondentes a partir desse ponto. Note que o programa vai executar todas as sentenças a partir desse ponto, na sequência dada. Inclusive passando pelos próximos casos que vêm a seguir, isto

  

é, após terminar com as sentenças relativas a este caso, o programa continua executando as sentenças dos demais casos, na seqüência dada. Ou seja, tudo se passa como se o primeiro caso onde o teste resultou em verdadeiro define um ponto de entrada inicial para que o programa comece a executar todas as sentenças presentes a partir deste ponto. Porém, veja logo a seguir no texto, como controlar melhor este fluxo de execuções.

O bloco iniciado pela palavra **`default`** é opcional e ele é executado se o valor da expressão não for encontrado em nenhum dos outros casos, ou se o comando **`switch`** não for abandonado antes da execução chegar ao **`default`**.

Ao contrário das estruturas **`if`**, onde cada condição possuía seu próprio bloco de execução, o **`switch`** consiste de um único grande bloco que une o código para todas as opções. Os casos servem para sinalizar diversos pontos de entrada onde a execução pode começar.

O fluxo de execução continua seqüencialmente de cima para baixo, testando os casos. Quando encontra um caso, i.e. uma palavra **`case`** cujo valor casa com o valor associado calculado para a expressão, o bloco correspondente a esse **`case`** é executado. A não ser que o fluxo de execução nesse bloco encontre um comando para interromper a execução do **`switch`** como um todo, a execução continua com os demais blocos.

Os comandos **`break`** e **`return`** são as duas formas mais comuns para forçar a execução a sair do bloco **`switch`**. O fato de que a execução segue diretamente para o próximo caso abre a possibilidade de que um certo valor da expressão cause a execução de vários blocos. Para manter a complexidade da lógica de execução dentro de limites aceitáveis, cada caso deveria ser finalizado com um comando **`break`**. A boa prática de programação também recomenda que o ultimo comando do caso **`default`** seja, igualmente, um **`break`**.

### Exemplo:

Queremos um programa que determina o preço com desconto para entradas de cinema. Estudantes recebem 50% de desconto, aposentados, 30%. Os demais clientes pagam o valor integral.

**Código fonte:**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
     // Declarar variáveis
     float preco;
     char categoria;
     float preco_final;

     // Pedir o preço do ingresso
     printf("Digite o preco do ingresso: ");
     scanf("%f", &preco);
     // Pedir a categoria
     printf("Selecione:\n");
     printf("E - estudante,\nA - Aposentado,\nN - normal\n");
     printf("Digite a categoria do cliente (E/A/N): ");
     scanf("%c", &categoria);

     switch (categoria) {
          case 'e':
          case 'E':
               preco_final = preco * 0.50f;
               printf("Com desconto estudante: %f\n",
```

  

```
                              preco_final);
                    break;
               case 'a':
               case 'A':
                    preco_final = preco * 0.70f;
                    printf("Com desconto aposentado: %f\n",
                              preco_final);
                    break;
               case 'n':
               case 'N':
                    printf("Preço sem desconto: %f\n", preco);
                    break;
               default:
                    printf("Categoria invalida!\n");
                    break;
          }
          return 0;
     }
```

*Consulte: EstruturasCondicionais\Cinema\Cinema.vcproj*

**Descrição passo a passo:**

```
float preco;
char categoria;
float preco_final;
```

Primeiro, declaram-se as variáveis. A variável `preco` conterá o valor do ingresso, solicitado ao usuário. A variável `preco_final` armazenará o valor do ingresso com desconto. O valor da variável `categoria` também será informado pelo usuário.

```
printf("Digite o preco do ingresso: ");
scanf("%f", &preco);
```

Solicita que o usuário digite o valor do preço do ingresso. Armazena o número digitado na variável `preco`.

```
printf("Selecione:\n");
printf("E - estudante,\nA - Aposentado,\nN - normal\n");
printf("Digite a categoria do cliente (E/A/N): ");
scanf(" %c", &categoria);
```

Solicita que o usuário digite "E", "A" ou "N" para escolher se o ingresso é para estudante, para aposentado ou um ingresso normal. Observe com atenção que é necessário o espaço em branco antes do `%c` em `scanf` para ler a próxima letra digitada pelo usuário. Se usássemos `%c` sem o espaço, o programa leria o caractere "`\n`" gerado quando o usuário pressionou ENTER na pergunta anterior, relativa ao preço.

```
switch (categoria) {
```

Este comando avalia o conteúdo da variável `categoria`. Em seguida, começa a comparar com os casos do **`switch`**.

```
     case 'e':
     case 'E':
          preco_final = preco * 0.50f;
          printf(...);
          break;
```

  

Se o valor da variável `categoria` for a letra “e” (minúsculo), então a execução passa para logo após o comando `case 'e':`. Caso o valor seja “E” (maiúsculo), a execução passa para `case 'E':`. Como não existem comandos entre esses dois casos, ambos indicam o mesmo ponto de início de execução. Desta forma, o programa aceita que o usuário digite tanto “e” minúsculo como “E” maiúsculo para selecionar a categoria “estudante”.

Após calcular o preço final e imprimir uma mensagem, aparece o comando `break`. Este comando interrompe a execução do `switch` e desvia para o próximo comando logo após o `switch`. Ele é necessário, pois sem ele, a execução prosseguiria do caso de categoria “estudante” para o caso de categoria “aposentado”.

```
        case 'a':
        case 'A':
                preco_final = preco * 0.70f;
                printf(...);
                break;
```

Os casos marcam o ponto de início de execução para as letras “a” ou “A”. Calcula-se o preço com desconto para aposentados. O `break` impede que a execução avance para a categoria “normais”.

```
        case 'n':
        case 'N':
                printf(...);
                break;
```

Os casos marcam o ponto de início de execução para as letras “n” ou “N”. Como de costume, o `break` impede o avanço da execução.

```
        default:
                printf("Categoria invalida!\n");
                break;
}
```

Se expressão do **`switch`** não coincidiu com nenhum dos casos anteriores, então o usuário digitou uma letra inválida para selecionar a categoria de desconto. Para estas situações o caso **`default`** imprime uma mensagem de erro.

O **`break`** na última linha do **`switch`** é opcional, mas mantém o código sistemático e organizado.

**Exemplo de execução:**

```
Digite o preco do ingresso: 5.00
Selecione:
E - estudante,
A - Aposentado,
N - normal
Digite a categoria do cliente (E/A/N): e
Com desconto estudante: 2.500000
```

Outro exemplo:

```
Digite o preco do ingresso: 5.00
Selecione:
E - estudante,
A - Aposentado,
```

  

```
N - normal
Digite a categoria do cliente (E/A/N): n
Preço sem desconto: 5.000000
```

Mais um exemplo:

```
Digite o preco do ingresso: 5.00
Selecione:
E - estudante,
A - Aposentado,
N - normal
Digite a categoria do cliente (E/A/N): x
Categoria invalida!
```

## Casos de Uso

Talvez você esteja se questionando quando usar cada uma das estruturas condicionais. Portanto, aqui estão algumas dicas:

`if`

- Quando um bloco deve ser executando apenas se uma condição for verdadeira.
- Combinando com outros comandos que aprenderemos mais tarde (como `return` e `break`), pode finalizar o programa sob determinadas condições (por exemplo, detecção de um erro).

`if...else`

- Quando uma condição implica a execução de um ou outro bloco.
- Sempre quando houver duas condições mutuamente exclusivas.
- Exemplo: Verificar um dado digitado pelo usuário e aceitá-lo ou imprimir uma mensagem de erro.

`if...else if... else if...`

- Testar intervalos de valores, ou valores que são contínuos ou podem variar num intervalo não finito.
- Verificar várias condições, entre as quais existe uma prioridade para atender somente a primeira condição que for verdadeira.
- Tratar diversas condições (não necessariamente mutuamente exclusivas).

`switch (…)`

- Para testar uma expressão que gere valores discretos.
- Em situações nas quais deve ser realizado um mesmo processamento para várias condições.
- Quando o número de possibilidade for razoavelmente grande e finito.

Não existe uma regra específica para se escolher uma estrutura ou outra. Em alguns casos, poder-se-ia utilizar qualquer uma destas estruturas. O exemplo abaixo imprime as palavras “um”, “dois” ou “muitos” dependendo do valor digitado pelo usuário.

  

**Exemplo:**

Programa que classifica um número digitado pelo usuário em três categorias: “um”, “dois” ou “muitos”. Ele realiza esta operação com cada uma das estruturas condicionais apresentados neste capítulo.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
     // Declarar variáveis
     int valor;

     // Pedir ao usuário para escrever um valor
     printf("Digite um valor ( >= 1): ");
     scanf("%d", &valor);

     // Usando apenas IF
     printf("Apenas usando IF: ");
     if (valor == 1) {
          printf("um");
     }
     if (valor == 2) {
          printf("dois");
     }
     if (valor >= 3) {
          printf("muitos");
     }
     printf("\n");
     // Usando apenas IF...ELSE
     printf("Apenas usando IF...ELSE: ");
     if (valor == 1) {
          printf("um");
     } else {
          // Aqui fazemos um if dentro do bloco do if anterior
          if (valor == 2) {
               printf("dois");
          }
          if (valor >= 3) {
               printf("muitos");
          }
     }
     printf("\n");
     // Usando apenas IF...ELSE IF... ELSE
     printf("Apenas usando IF...ELSE IF...ELSE: ");
     if (valor == 1) {
          printf("um");
     } else if (valor == 2) {
          printf("dois");
     } else {
          printf("muitos");
     }
     printf("\n");
     // Usando apenas SWITCH
     printf("Apenas usando SWITCH: ");
     switch (valor) {
          case 1:
               printf("um");
               break;
          case 2:
               printf("dois");
               break;
```

  

```
            default:
                printf("muitos");
                break;
        }
        printf("\n");

        return 0;
}
```

*Consulte: EstruturasCondicionais\Comparativo\Comparativo.vcproj*

  