5. Estruturas de Controle

Estruturas de controle permitem controlar a seqüência das ações lógicas de um programa. Basicamente, existem dois tipos de estruturas de controle: estruturas de **repetição** e estruturas de **decisão**. A estrutura de repetição permite que um bloco de instruções seja executado repetidamente uma quantidade controlada de vezes. A estrutura de decisão permite executar um entre dois ou mais blocos de instruções. Neste capítulo estudaremos em detalhe as instruções do C que permitem implementar estas estruturas.

5.1 Condição de controle

Em todas as estruturas, existe pelo menos uma expressão que faz o controle de **qual** bloco de instruções será executado ou **quantas vezes** ele será executado: é o que chamamos de **condição de controle**. Uma condição de controle é uma expressão lógica ou aritmética cujo resultado pode ser considerado verdadeiro ou falso. Conforme vimos na seção 3.5, a linguagem C não possui, entretanto, variáveis ou constantes lógicas, possui somente expressões numéricas, assim quando uma **expressão numérica** se encontra em uma **condição de controle**, ela será considerada **falsa** se seu valor for **igual a zero**, e **verdadeira** se seu valor for **diferente de zero**.

Exemplo: Observe nas condições abaixo, seu valor numérico e seu significado lógico. Considere as variáveis `int i = 0, j = 3;`

condição	valor numérico	significado lógico
(i == 0)	1	verdadeiro
(i > j)	0	falso
(i)	0	falso
(j)	3	verdadeiro

Este fato deve ficar claro pois, nas estruturas que estudaremos neste capítulo, quando for dito que uma condição é **falsa** ou **verdadeira** quer se dizer que seu valor e **igual** a zero ou **diferente** de zero.

5.2 Estrutura `do...while`

Esta é uma estrutura básica de repetição condicional. Permite a execução de um bloco de instruções repetidamente. Sua sintaxe é a seguinte:

Sintaxe:

```
do{
   bloco
}while( condição);
```

onde: *condição* é uma expressão lógica ou numérica.

bloco é um conjunto de instruções.

Esta estrutura faz com que o bloco de instruções seja executado pelo menos uma vez. Após a execução do bloco, a condição é avaliada. Se a condição é **verdadeira** o bloco é executado outra vez, caso contrário a repetição é terminada. Ofluxograma desta estrutura é mostrada na figura 5.1:

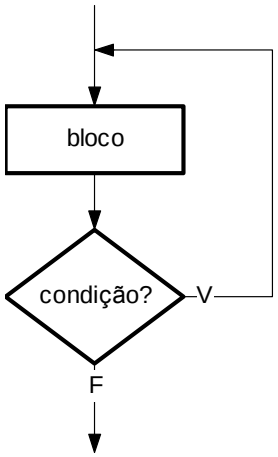

Figura 5.1: *Fluxograma da estrutura* `do...while`.

Exemplo: No trecho abaixo, a leitura de um número é feita dentro de um laço de repetição condicional. A leitura é repetida caso o número lido seja negativo.

```
do{
  puts("Digite um número positivo:");
  scanf("%f",&num);
}while(num <= 0.0);
```

Programa exemplo: No arquivo `e0501.cpp` existe um programa que calcula o fatorial de um número. Este programa ilustra o uso da estrutura `do...while`.

5.3 Estrutura `while`

A estrutura de repetição condicional `while` é semelhante a estrutura `do...while`. Sua sintaxe é a seguinte:

Sintaxe:

```
while( condição){
   bloco
}
```

onde: *condição* é uma expressão lógica ou numérica.

bloco é um conjunto de instruções.

Esta estrutura faz com que a condição seja avaliada em primeiro lugar. Se a condição é **verdadeira** o bloco é executado uma vez e a condição é avaliada novamente. Caso a condição seja **falsa** a repetição é terminada sem a execução do bloco. Observe que nesta estrutura, ao contrário da estrutura `do...while`, o bloco de instruções pode não ser executado nenhuma vez, basta que a condição seja inicialmente falsa. O fluxograma desta estrutura é mostrada na figura 5.2:

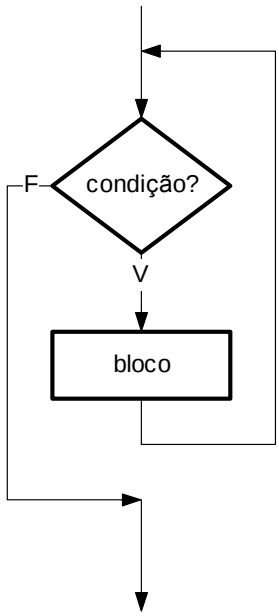

Figura 5.2: *Fluxograma da estrutura* `while`.

Exemplo: No trecho abaixo, calcula-se a precisão (ε) do processador aritmético do PC. A variável `eps` tem seu valor dividido por `2` enquanto o processador conseguir distinguir entre 1 e 1+ε. Após a execução do laço, o valor de `eps` contém a precisão da máquina.

```
eps = 1.0;
while(1.0 + eps > 1.0){
```

```
    eps /= 2.0;
}
```

Programa exemplo: No arquivo `e0502.cpp` existe um programa que calcula a raiz quadrada de um número real positivo usando o método de Newton. Este programa ilustra o uso da estrutura `while`.

5.4 Estrutura `for`

A estrutura `for` é muito semelhante as estruturas de repetição vistas anteriormente, entretanto costuma ser utilizada quando se quer um número determinado de ciclos. A contagem dos ciclos é feita por uma variável chamada de **contador**. A estrutura `for` é, as vezes, chamada de estrutura de **repetição com contador**. Sua sintaxe é a seguinte:

Sintaxe:

```
for(inicialização; condição; incremento){
   bloco
}
```

onde: *inicialização* é uma expressão de inicialização do contador.
condição é uma expressão lógica de controle de repetição.
incremento é uma expressão de incremento do contador.
bloco é um conjunto de instruções a ser executado.

Esta estrutura executa um número determinado de repetições usando um contador de iterações. O contador é inicializado na expressão de *inicialização* **antes** da primeira iteração. Por exemplo: `i = 0;` ou `cont = 20;`. Então o bloco é executado e **depois** de cada iteração, o contador é incrementado de acordo com a expressão de *incremento*. Por exemplo: `i++` ou `cont -= 2`. Então a expressão de condição é avaliada: se a condição for verdadeira, o *bloco* é executado novamente e o ciclo recomeça, se a condição é falsa termina-se o laço. Esta condição é, em geral, uma expressão lógica que determina o ultimo valor do contador. Por exemplo: `i <= 100` ou `cont > 0`.

Exemplo: No trecho abaixo, o contador `i` é inicializado com o valor `1`. O bloco é repetido enquanto a condição `i <= 10` for verdadeira. O contador é incrementado com a instrução `i++`. Esta estrutura, deste modo, imprime os números `1, 2, ..., 9, 10`.

```
for(i=1; i<=10; i++){
```

```
    printf(" %d",i);
}
```

É interessante notar que a mesma estrutura lógica pode ser implementada usando as estruturas `for` ou `do...while`:

Exemplo: As seguintes instruções são plenamente equivalentes:

```
i = 0;                  for(i = 0; i <= 100; i++){
do{                        bloco
   bloco                }
  i++;
}while(i <= 100);
```

Podem existir mais de uma expressão de *inicialização* e de *incremento* na estrutura `for`. Estas expressões devem ser separadas por vírgula (`,`). Mas **não pode** haver mais de uma expressão de *condição*. Por exemplo: `for(i=0, j=10; i<10; i++, j--){...}`

Programa exemplo: No arquivo `e0503.cpp` existe um programa que calcula a amplitude de um conjunto de valores. Este programa exemplifica o uso da estrutura `for...`

5.5 Estrutura de decisão `if...else`

A estrutura `if...else` é a mais simples estrutura de controle do C. Esta estrutura permite executar um entre vários blocos de instruções. O controle de qual bloco será executado será dado por uma *condição* (expressão lógica ou numérica). Esta estrutura pode se apresentar de modos ligeiramente diferentes. Nesta seção vamos apresentar separadamente cada uma das possibilidades de sintaxe.

5.5.1 Decisão de um bloco (`if...`)

A estrutura de decisão de um bloco permite que se execute (ou não) um bloco de instruções conforme o valor de uma condição seja verdadeiro ou falso. O fluxograma desta estrutura é mostrada na figura 5.3.

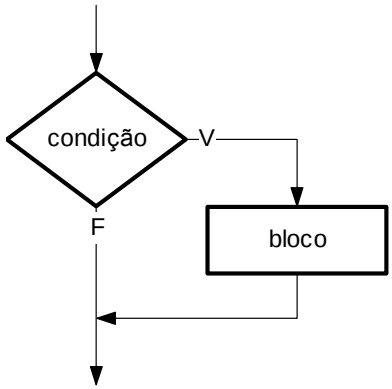

Figura 5.3: *Fluxograma da estrutura de decisão* `if...`

Sintaxe: Decisão com um bloco:

```
if( condição){
   bloco
}
```

onde: *condição* é uma expressão lógica ou numérica.

bloco é um conjunto de instruções.

Se a condição **verdadeira**, o *bloco* é executado. Caso contrário, o bloco não é executado.

Exemplo: No trecho abaixo, se o valor lido for maior que 10, então o seu valor é redefinido como 10. Observe que o bloco constitui-se de um única instrução.

```
printf("Digite o número de repetições: (máximo 10)";
scanf("%d",&iter);
if(iter > 10){
  iter = 10;
}
```

Programa Exemplo: O arquivo `e0504.cpp` mostra um programa que utiliza a estrutura `if...` para emitir um sinal sonoro ao imprimir um número múltiplo de 4.

5.5.2 Decisão de dois blocos (`if...else`)

Também é possível escrever uma estrutura que execute um entre dois blocos de instruções. A figura 5.4 mostra o fluxograma correspondente a esta estrutura de decisão.

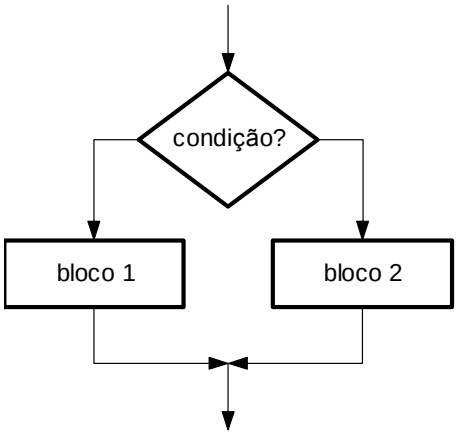

Figura 5.4: *Fluxograma da estrutura de decisão* `if...else`

Sintaxe: Decisão de dois blocos:

```
if( condição){
   bloco 1;
}else{
   bloco 2;
}
```

onde: *condição* é uma expressão lógica ou numérica.

bloco 1 e *bloco 2* são conjuntos de instruções.

Se a `condição` for **verdadeira** o *bloco 1* é executado. Caso contrário, o *bloco 2* é executado.

Exemplo: No trecho abaixo, se o valor de `raiz*raiz` for maior que `num` o valor de `raiz` será atribuído a `max`, caso contrario, será atribuído a `min`.

```
if(raiz*raiz > num){
  max = raiz;
}else{
  min = raiz;
}
```

Programa Exemplo: O arquivo `e0505.cpp` mostra um programa que utiliza a estrutura `if...else` para determinar o tipo de raízes de uma equação de segundo grau.

5.5.3 Decisão de múltiplos blocos (`if...else if...`)

Também é possível escrever uma estrutura que execute um entre múltiplos blocos de instruções. A figura 5.5 mostra o fluxograma correspondente a esta estrutura de decisão.

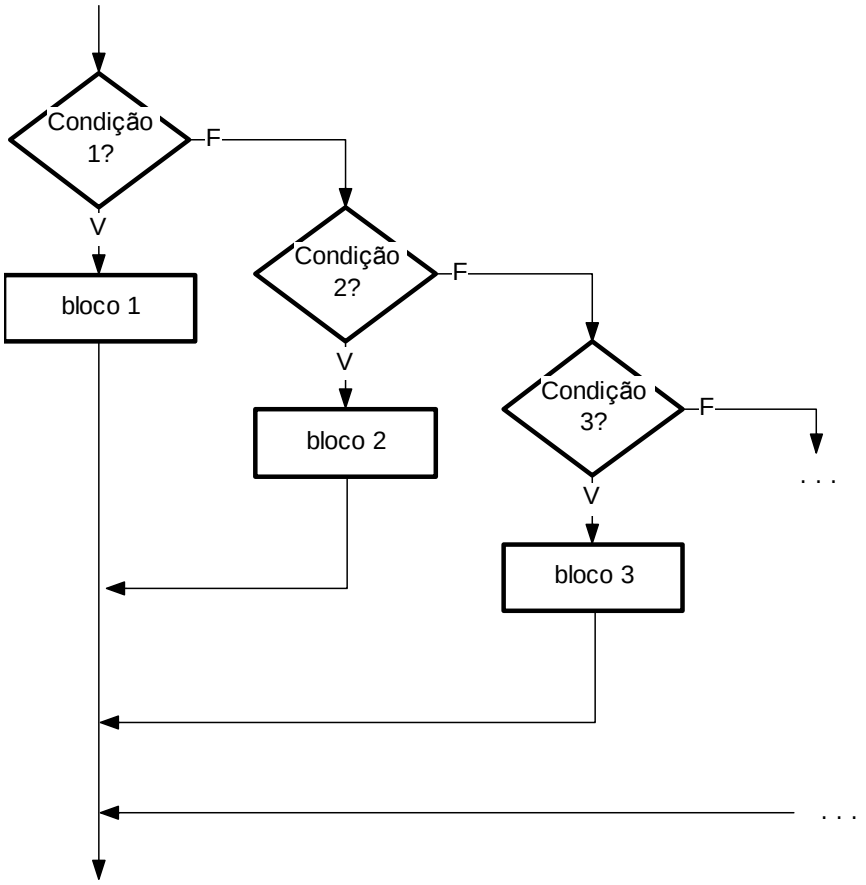

Figura 5.5: *Fluxograma da estrutura de decisão* `if... else if`.

Sintaxe: Decisão de múltiplos blocos:

```
if( condição 1){
   bloco 1;
...
}else if( condição N){
   bloco N;
}else{
   bloco P
}
```

onde: *condição 1, condição 2, ...* são expressões lógicas ou numéricas.

bloco 1 , bloco 2,... são conjuntos de instruções.

Se a condição 1 for **verdadeira** o *bloco 1* é executado. Caso contrario, a condição 2 é avaliada. Se a condição 2 for **verdadeira** o *bloco 2* é executado. Caso contrario, a condição 3 é avaliada e assim sucessivamente. Se nenhuma condição é **verdadeira** *bloco P* é executado. Observe que apenas um dos blocos é executado.

Exemplo: No trecho abaixo, uma determinada ação é executada se o valor de `num` for positivo, negativo ou nulo.

```
if(num > 0){
  a = b;
}else if(num < 0){
  a = b + 1;
}else{
  a = b - 1;
}
```

Programa Exemplo: O arquivo `e0506.cpp` mostra um programa que utiliza a estrutura `if...else if` para determinar se um número é maior, menor ou igual a outro.

5.6 Estrutura `switch...case`

A estrutura `switch...case` é uma estrutura de decisão que permite a execução de um conjunto de instruções a partir pontos diferentes conforme o resultado de uma expressão inteira de controle. O resultado deste expressão é comparado ao valor de cada um dos rótulos, e as instruções são executadas a partir desde rótulo. A figura 5.6 mostra o fluxograma lógico desta estrutura.

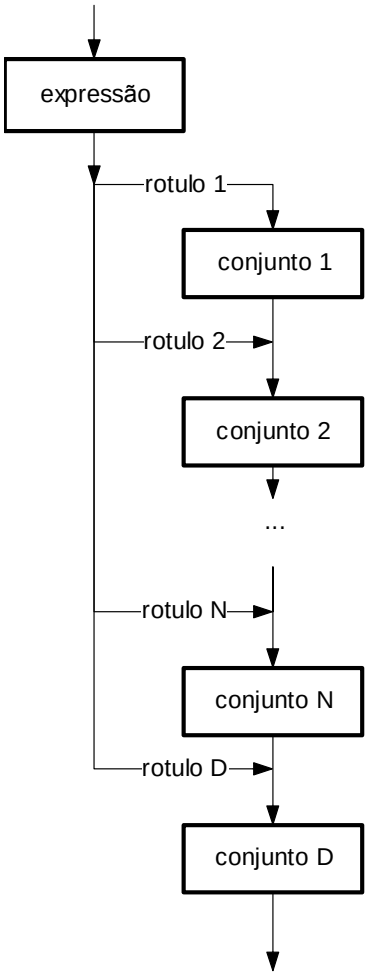

Figura 5.6: *Fluxograma da estrutura* `switch... case`.

Sintaxe: Esta estrutura possui a seguinte sintaxe:

```
switch(expressão){
case rótulo_1:
   conjunto_1
case rótulo_2:
   conjunto_2
...
case rótulo_n:
   conjunto n
[default:
   conjunto d]
}
```

onde:

expressão é uma expressão inteira.

rótulo_1, *rótulo_2*, ... *rótulo_n* e *rótulo_d* são constantes inteiras.

conjunto 1, *conjunto 2*, ..., *conjunto n* e *conjunto d* são conjuntos de instruções.

O valor de *expressão* é avaliado e o fluxo lógico será desviado para o conjunto cujo *rótulo* é igual ao resultado da expressão e todas as instruções **abaixo** deste rótulo serão executadas. Caso o resultado da expressão for diferente de todos os valores dos rótulos então *conjunto d* é executado. Os rótulos devem ser expressões constantes inteiras **diferentes** entre si. O rótulo `default` é opcional.

Esta estrutura é particularmente útil quando se tem um conjunto de instruções que se deve executar em ordem, porém se pode começar em pontos diferentes.

Exemplo: O trecho abaixo ilustra o uso da instrução `switch` em um menu de seleção. Neste exemplo, o programa iniciará o processo de usinagem de uma peça em um ponto qualquer dependendo do valor lido.

```
int seleção;
puts("Digite estagio de usinagem:");
scanf("%d",&selecao);
switch(seleção){
case 1:
  // desbaste grosso...
case 2:
  // desbaste fino...
case 3:
  // acabamento...
case 4:
  // polimento...
}
```

Programa Exemplo: O arquivo `e0507.cpp` mostra um programa que utiliza a estrutura `switch` para determinar o valor de um lanche.

5.7 Interrupção e desvio: `break, continue, goto, exit()`

As instruções vistas anteriormente podem sofrer **desvios** e **interrupções** em sua seqüência lógica normal através do uso certas instruções. As instruções que veremos a seguir devem ser usadas com muita parcimônia, pois fogem da lógica estruturada tem a tendência de tornar um programa incompreensível.

5.7.1 A instrução `break`.

Esta instrução serve para terminar a execução das instruções de um laço de repetição (`for`, `do...while`, `while`) ou para terminar um conjunto `switch...case`.

Quando em um laço de repetição, esta instrução força a interrupção do laço independentemente da condição de controle.

Exemplo: No trecho abaixo um laço de repetição lê valores para o cálculo de uma média. O laço possui uma condição de controle sempre verdadeira o que, a principio, é um erro: laço infinito. Porém, a saída do laço se dá pela instrução `break` que é executada quando um valor negativo é lido.

```
puts("digite valores:");
do{
  puts("valor:");
  scanf("%f",&val);
  if(val < 0.0){
    break;     // saída do laço
  }
  num++;
  soma += val;
}while(1);     // sempre verdadeiro
printf("média: %f",soma/num);
```

Exemplo: No exemplo acima, o uso da instrução `break` poderia ter sido evitado, como segue:

```
puts("digite valores:");
do{
  puts("valor:");
  scanf("%f",&val);
  if(val >= 0.0){
    num++;
```

```
    soma += val;
  }
}while(val >= 0.0);
printf("média: %f",soma/num);
```

O outro uso da instrução `break`, em estruturas `switch...case`, serve para separar os conjuntos de instruções em cada `case`.

Exemplo: Estrutura `switch...case` com a instrução `break`:

```
int tipo;
puts("Selecione o sabor de sua pizza:");
puts("Muzzarela Calabreza Alho&Oleo:");
tipo = getch();
switch(tipo){
case ´M´:
  // prepara pizza muzzarela...
case ´C´:
  // prepara pizza calabreza...
case ´A´:
  // prepara pizza Alho&Oleo...
default:
  puts("Opcao incorreta");
}
```

Programa Exemplo: O arquivo `e0508.cpp` mostra um programa que utiliza a estrutura `switch` com a instrução `break` para simular um piano no teclado do computador.

5.7.2 A instrução `continue`.

Esta instrução opera de modo semelhante a instrução `break` dentro de um laço de repetição. Quando executada, ela pula as instruções de um laço de repetição sem sair do laço. Isto é, a instrução força a avaliação da condição de conttrole do laço.

Exemplo: No trecho abaixo revemos um laço de repetição lê valores para o cálculo de uma média. Se `(val < 0.0)` então o programa salta diretamente para a condição de controle, sem executar o resto das instruções.

```
puts("digite valores:");
do{
  puts("valor:");
  scanf("%f",&val);
  if(val < 0.0){      // se val é negativo...
    continue;         // ...salta para...
  }
  num++;              // se (val < 0.0) estas instruções
  soma += val;        // não são executadas!
}while(val >= 0.0);   // ...fim do laço
printf("média: %f",soma/num);
```

5.7.3 A instrução `goto`.

Esta instrução é chamada de desvio de fluxo. A instrução desvia o programa para um rótulo (posição identificada) no programa. São raros os casos onde a instrução goto é necessária, no entanto, há certas circunstâncias, onde usada com prudência, ela pode ser útil.

Sintaxe: A sintaxe da instrução goto é a seguinte:

```
goto rótulo;
...
rótulo:
...
```

onde *rótulo* é um identificador válido.

Exemplo: No trecho abaixo revemos um laço de repetição lê valores para o cálculo de uma média. Foram usadas duas instruções `goto`.

```
puts("digite valores:");
inicio:               // rótulo
  puts("valor:");
  scanf("%f",&val);
  if(val < 0.0){      // se val é negativo...
    goto fim;         // ...salta para fim
  }
  num++;              // se (val < 0.0) estas instruções
  soma += val;        // não são executadas!
```

```
goto inicio;          // salta para inicio
fim:                  // rótulo
printf("média: %f",soma/num);
```

5.7.4 A função `exit()`.

Esta função (não instrução) `exit()` , da biblioteca `stdlib.h`, é uma função que termina a execução de um programa. Normalmente um programa é terminado quando se executa a última sua instrução, porém pode-se terminar a execução do programa a qualquer momento com o uso desta função.

A função `exit()` tem a seguinte declaração: `void exit(int status)`. Onde o argumento da função é um valor inteiro que será passado para o Sistema Operacional: (variável de sistema `errorlevel` no DOS).

Exemplo: No trecho abaixo revemos um laço de repetição lê valores para o cálculo de uma média. Foi usado a função exit para terminar a execução do programa.

```
puts("digite valores:");
do{
  puts("valor:");
  scanf("%f",&val);
  if(val < 0.0){                  // se val é negativo...
    printf("média: %f",soma/num);  // imprime resultado
    exit(0);                      // termina programa
  }
  num++;  soma += val;
}while(1);
```