3. Controle de fluxo

W. Celes e J. L. Rangel

A linguagem C provê as construções fundamentais de controle de fluxo necessárias para programas bem estruturados: agrupamentos de comandos, tomadas de decisão (`if-else`), laços com teste de encerramento no início (`while`, `for`) ou no fim (`do-while`), e seleção de um dentre um conjunto de possíveis casos (`switch`).

3.1. Decisões com `if`

`if` é o comando de decisão básico em C. Sua forma pode ser:

```
if (expr) {
    bloco de comandos 1
    ...
}
```

ou

```
if ( expr ) {
    bloco de comandos 1
    ...
}
else {
    bloco de comandos 2
    ...
}
```

Se *expr* produzir um valor diferente de 0 (verdadeiro), o *bloco de comandos 1* será executado. A inclusão do `else` requisita a execução do *bloco de comandos 2* se a expressão produzir o valor 0 (falso). Cada bloco de comandos deve ser delimitado por uma chave aberta e uma chave fechada. Se dentro de um bloco tivermos apenas um comando a ser executado, as chaves podem ser omitidas (na verdade, deixamos de ter um bloco):

```
if ( expr )
    comando1;
else
    comando2;
```

A indentação (recuo de linha) dos comandos é fundamental para uma maior clareza do código. O estilo de indentação varia a gosto do programador. Além da forma ilustrada acima, outro estilo bastante utilizado por programadores C é:

```
if ( expr )
{
    bloco de comandos 1
    ...
}
else
{
    bloco de comandos 2
    ...
}
```

Podemos aninhar comandos `if`. Um exemplo simples é ilustrado a seguir:

```
#include <stdio.h>

int main (void)
{
   int a, b;
   printf("Insira dois numeros inteiros:");
   scanf("%d%d",&a,&b);
   if (a%2 == 0)
      if (b%2 == 0)
         printf("Voce inseriu dois numeros pares!\n");
   return 0;
}
```

Primeiramente, notamos que não foi necessário criar blocos ({...}) porque a cada `if` está associado apenas um comando. Ao primeiro, associamos o segundo comando `if`, e ao segundo `if` associamos o comando que chama a função `printf`. Assim, o segundo `if` só será avaliado se o primeiro valor fornecido for par, e a mensagem só será impressa se o segundo valor fornecido também for par. Outra construção para este mesmo exemplo simples pode ser:

```
int main (void)
{
   int a, b;
   printf("Digite dois numeros inteiros:");
   scanf("%d%d",&a,&b);
   if ((a%2 == 0) && (b%2 == 0))
      printf ( "Voce digitou dois numeros pares!\n");
   return 0;
}
```

produzindo resultados idênticos.

Devemos, todavia, ter cuidado com o aninhamento de comandos `if-else`. Para ilustrar, consideremos o exemplo abaixo.

```
/* temperatura (versao 1 - incorreta) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf("Digite a temperatura: ");
   scanf("%d", &temp);
   if (temp < 30)
      if (temp > 20)
         printf(" Temperatura agradavel \n");
   else
      printf(" Temperatura muito quente \n");
   return 0;
}
```

A idéia deste programa era imprimir a mensagem `Temperatura agradável` se fosse fornecido um valor entre `20` e `30`, e imprimir a mensagem `Temperatura muito quente` se fosse fornecido um valor maior que `30`. No entanto, vamos analisar o caso de

ser fornecido o valor `5` para `temp`. Observando o código do programa, podemos pensar que nenhuma mensagem seria fornecida, pois o primeiro `if` daria resultado verdadeiro e então seria avaliado o segundo `if`. Neste caso, teríamos um resultado falso e como, *aparentemente*, não há um comando `else` associado, nada seria impresso. Puro engano. A indentação utilizada pode nos levar a erros de interpretação. O resultado para o valor 5 seria a mensagem `Temperatura muito quente`. Isto é, o programa está INCORRETO.

Em C, um `else` está associado ao último `if` que não tiver seu próprio `else`. Para os casos em que a associação entre `if` e `else` não está clara, recomendamos a criação explícita de blocos, mesmo contendo um único comando. Reescrevendo o programa, podemos obter o efeito desejado.

```
/* temperatura  (versao 2) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf ( "Digite a temperatura: " );
   scanf ( "%d", &temp );
   if ( temp < 30 )
   {
      if ( temp > 20 )
         printf ( " Temperatura agradavel \n" );
   }
   else
      printf ( " Temperatura muito quente \n" );
   return 0;
}
```

Esta regra de associação do `else` propicia a construção do tipo `else-if`, sem que se tenha o comando `elseif` explicitamente na gramática da linguagem. Na verdade, em C, construímos estruturas `else-if` com `if`'s aninhados. O exemplo abaixo é válido e funciona como esperado.

```
/* temperatura  (versao 3) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf("Digite a temperatura: ");
   scanf("%d", &temp);

   if (temp < 10)
      printf("Temperatura muito fria \n");
   else if (temp < 20)
      printf(" Temperatura fria \n");
   else if (temp < 30)
      printf("Temperatura agradavel \n");
   else
      printf("Temperatura muito quente \n");
   return 0;
}
```

Estruturas de bloco

Observamos que uma função C é composta por estruturas de blocos. Cada chave aberta e fechada em C representa um bloco. As declarações de variáveis só podem ocorrer no início do corpo da função ou no início de um bloco, isto é, devem seguir uma chave aberta. Uma variável declarada dentro de um bloco é válida apenas dentro do bloco. Após o término do bloco, a variável deixa de existir. Por exemplo:

```
...
if ( n > 0 )
{
   int i;
   ...
}
...           /* a variável i não existe neste ponto do programa */
```

A variável `i`, definida dentro do bloco do `if`, só existe dentro deste bloco. É uma boa prática de programação declarar as varáveis o mais próximo possível dos seus usos.

Operador condicional

C possui também um chamado *operador condicional*. Trata-se de um operador que substitui construções do tipo:

```
...
if ( a > b )
   maximo = a;
else
   maximo = b;
...
```

Sua forma geral é:

condição `?` *expressão1* `:` *expressão2*`;`

se a *condição* for verdadeira, a *expressão1* é avaliada; caso contrário, avalia-se a *expressão2*.

O comando:

```
maximo = a > b ? a : b ;
```

substitui a construção com `if-else` mostrada acima.

3.2. Construções com laços

É muito comum, em programas computacionais, termos procedimentos iterativos, isto é, procedimentos que devem ser executados em vários passos. Como exemplo, vamos considerar o cálculo do valor do fatorial de um número inteiro não negativo. Por definição:

$$n\,!= n \times (n-1) \times (n-2)...3 \times 2 \times 1, \textit{ onde } 0\,!= 1$$

Para calcular o fatorial de um número através de um programa de computador, utilizamos tipicamente um processo iterativo, em que o valor da variável varia de `1` a `n`.

A linguagem C oferece diversas construções possíveis para a realização de laços iterativos. O primeiro a ser apresentado é o comando `while`. Sua forma geral é:

```
while (expr)
{
   bloco de comandos
   ...
}
```

Enquanto *expr* for avaliada em verdadeiro, o bloco de comandos é executado repetidamente. Se *expr* for avaliada em falso, o bloco de comando não é executado e a execução do programa prossegue. Uma possível implementação do cálculo do fatorial usando `while` é mostrada a seguir.

```
/* Fatorial */

#include <stdio.h>

int main (void)
{
   int i;
   int n;
   int f = 1;

   printf("Digite um número inteiro nao negativo:");
   scanf("%d", &n);

   /* calcula fatorial */
   i = 1;
   while (i <= n)
   {
      f *= i;
      i++;
   }

   printf(" Fatorial = %d \n", f);
   return 0;
}
```

Uma segunda forma de construção de laços em C, mais compacta e amplamente utilizada, é através de laços `for`. A forma geral do `for` é:

```
for (expr_inicial; expr_booleana; expr_de_incremento)
{
   bloco de comandos
   ...
}
```

A ordem de avaliação desta construção é ilustrada a seguir:

```
expr_inicial;
while (expr_booleana)
{
    bloco de comandos
    ...
    expr_de_incremento
}
```

A seguir, ilustramos a utilização do comando `for` no programa para cálculo do fatorial.

```
/* Fatorial (versao 2) */

#include <stdio.h>

int main (void)
{
   int i;
   int n;
   int f = 1;

   printf("Digite um número inteiro nao negativo:");
   scanf("%d", &n);

   /* calcula fatorial */
   for (i = 1; i <= n; i++)
   {
      f *= i;
   }
   printf(" Fatorial = %d \n", f);
   return 0;
}
```

Observamos que as chaves que seguem o comando `for`, neste caso, são desnecessárias, já que o corpo do bloco é composto por um único comando.

Tanto a construção com `while` como a construção com `for` avaliam a expressão booleana que caracteriza o teste de encerramento no início do laço. Assim, se esta expressão tiver valor igual a zero (falso), quando for avaliada pela primeira vez, os comandos do corpo do bloco não serão executados nem uma vez.

C provê outro comando para construção de laços cujo teste de encerramento é avaliado no final. Esta construção é o `do-while`, cuja forma geral é:

```
do
{
   bloco de comandos
} while (expr_booleana);
```

Um exemplo do uso desta construção é mostrado abaixo, onde validamos a inserção do usuário, isto é, o programa repetidamente requisita a inserção de um número enquanto o usuário inserir um inteiro negativo (cujo fatorial não está definido).

```
/* Fatorial (versao 3) */

#include <stdio.h>

int main (void)
{
   int i;
   int n;
   int f = 1;

   /* requisita valor do usuário */
   do
   {
      printf("Digite um valor inteiro nao negativo:");
      scanf ("%d", &n);
   } while (n<0);

   /* calcula fatorial */
   for (i = 1; i <= n; i++)
      f *= i;

   printf(" Fatorial = %d\n", f);
   return 0;
}
```

Interrupções com `break` e `continue`

A linguagem C oferece ainda duas formas para a interrupção antecipada de um determinado laço. O comando `break`, quando utilizado dentro de um laço, interrompe e termina a execução do mesmo. A execução prossegue com os comandos subseqüentes ao bloco. O código abaixo ilustra o efeito de sua utilização.

```
#include <stdio.h>

int main (void)
{
   int i;
   for (i = 0; i < 10; i++)
   {
      if (i == 5)
         break;
      printf("%d  ", i);
   }
   printf("fim\n");
   return 0;
}
```

A saída deste programa, se executado, será:

```
0  1  2  3  4   fim
```

pois, quando `i` tiver o valor `5`, o laço será interrompido e finalizado pelo comando `break`, passando o controle para o próximo comando após o laço, no caso uma chamada final de `printf`.

O comando `continue` também interrompe a execução dos comandos de um laço. A diferença básica em relação ao comando `break` é que o laço não é automaticamente finalizado. O comando `continue` interrompe a execução de um laço passando para a próxima iteração. Assim, o código:

```
#include <stdio.h>

int main (void)
{
   int i;
   for (i = 0; i < 10; i++ )
   {
      if (i == 5) continue;
      printf("%d  ", i);
   }
   printf("fim\n");
   return 0;
}
```

gera a saída:

```
0  1  2  3  4  6  7  8  9  fim
```

Devemos ter cuidado com a utilização do comando `continue` nos laços `while`. O programa:

```
/* INCORRETO */

#include <stdio.h>

int main (void)
{
   int i = 0;
   while (i < 10)
   {
      if (i == 5) continue;
      printf("%d  ", i);
      i++;
   }
   printf("fim\n");
   return 0;
}
```

é um programa INCORRETO, pois o laço criado não tem fim – a execução do programa não termina. Isto porque a variável `i` nunca terá valor superior a `5`, e o teste será sempre verdadeiro. O que ocorre é que o comando `continue` "pula" os demais comandos do laço quando `i` vale `5`, inclusive o comando que incrementa a variável `i`.

3.3. Seleção

Além da construção `else-if`, C provê um comando (`switch`) para selecionar um dentre um conjunto de possíveis casos. Sua forma geral é:

```
switch ( expr )
{
   case op1:
      ...      /* comandos executados se expr == op1  */
   break;
   case op2:
      ...      /* comandos executados se expr == op2  */
   break;
   case op3:
      ...      /* comandos executados se expr == op3  */
   break;
   default:
      ...      /* executados se expr for diferente de todos  */
   break;
}
```

op_i deve ser um número inteiro ou uma constante caractere. Se *expr* resultar no valor op_i, os comandos que se seguem ao caso op_i são executados, até que se encontre um `break`. Se o comando `break` for omitido, a execução do caso continua com os comandos do caso seguinte. O caso `default` (nenhum dos outros) pode aparecer em qualquer posição, mas normalmente é colocado por último. Para exemplificar, mostramos a seguir um programa que implementa uma calculadora convencional que efetua as quatro operações básicas. Este programa usa constantes caracteres, que serão discutidas em detalhe quando apresentarmos cadeias de caracteres em C. O importante aqui é entender conceitualmente a construção `switch`.

```
/* calculadora de quatro operações */

#include <stdio.h>

int main (void)
{
   float num1, num2;
   char op;

   printf("Digite: numero op numero\n");
   scanf ("%f %c %f", &num1, &op, &num2);
   switch (op)
   {
      case '+':
         printf(" = %f\n", num1+num2);
      break;
      case '-':
         printf(" = %f\n", num1-num2);
      break;
      case '*':
         printf(" = %f\n", num1*num2);
      break;
      case '/':
         printf(" = %f\n", num1/num2);
      break;
      default:
         printf("Operador invalido!\n");
      break;
   }
   return 0;
}
```