7 Mais Detalhes da Linguagem C

Ronaldo F. Hashimoto e Carlos H. Morimoto

Nesta aula vamos mostrar alguns conceitos e detalhes da linguagem C, como blocos, abreviaturas, atribuições e constantes. Ao final dessa aula você deverá ser capaz de:

- Identificar blocos, situações que necessitam a utilização de chaves em seu programa, e situações onde as chaves podem ser omitidas.
- Simular e utilizar abreviaturas do comando de atribuição (como += e -=).
- Simular e utilizar atribuições múltiplas e no instante da declaração.
- Definir constantes e utilizá-las em seus programas.

Observação: por permitirem um programa mais conciso, alguns programadores acreditam que seja mais fácil ler e entender programas que se utilizam desses detalhes. Você não precisa utilizá-los, mas deve entendê-los. Se quiser utilizá-los, use de seu bom senso e utilize esses detalhes em seu programa sempre que você acreditar que vale a pena. Procure não utilizá-los de forma aleatória, e tente sempre ser consistente e claro na organização de seus programas.

7.1 Definição de Bloco

Os comandos dentro de uma repetição (comando `while`) são colocados entre chaves, como mostra a figura 9a.

Comando Composto (bloco)	Comando Simples

while (<condição>) {
  <comando_1>;
  <comando_2>;
  ...
  <comando_n>;
}
<comando_fora_do_while_1>;
	while (<condição>)
	  <comando_1>;
	
	<comando_fora_do_while_1>;
	...
	<comando_fora_do_while_n>;

(a)	(b)

Figura 9: a) Comando composto definido entre { e }; (b) Comando simples, não é necessário o uso de chaves.

Todos os comandos que estiverem entre as chaves { e } estão dentro da repetição e formam um comando composto.

Agora, se dentro da repetição houver somente um comando, é possível omitir as chaves. Assim, no trecho de programa da figura 9b somente o <comando_1> está dentro da repetição. Os demais comandos estão fora! Note a correta endentação dos comandos.

O mesmo acontece com os comandos `if` e `if-else`. Compare os dois trechos de códigos:

```
num = 1;
soma = 0;
while (num<n) {
  if (n % num == 0) {
    soma = soma + num;
  }
  num = num + 1;
}
if (soma == n) {
  printf ("numero perfeito\n");
}
else {
  printf ("nao eh perfeito\n");
}
```

```
num = 1;
soma = 0;
while (num<n) {
  if (n % num == 0)
    soma = soma + num;
  num = num + 1;
}
if (soma == n)
  printf ("numero perfeito\n");
else
  printf ("nao eh perfeito\n");
```

Estes dois trechos são equivalentes, ou seja, produzem o mesmo resultado. No entanto, no trecho do lado esquerdo, todas chaves foram colocadas; enquanto que no trecho direito, algumas chaves foram omitidas para os comandos simples. Confuso? Basta se lembrar que um comando composto pode ter zero ou mais comandos, ou seja, um comando simples entre chaves nada mais é que um comando composto com apenas um comando.

Note que, uma vez que dentro do comando de repetição (`while`) temos dois comandos:

- um de seleção (`if (n % num == 0)`)
- e outro de atribuição (`num = num + 1`)

o uso das chaves é obrigatório.

EXERCÍCIO: Simule a execução desse programa para n = 120. O que é um número perfeito?

DICA: É muito importante adotar um bom espaçamento entre blocos para que você consiga visualizar claramente a estrutura de seu programa, ou leiaute. Para saber mais dicas sobre leiaute, veja as notas de aula do Prof. Feofiloff em `http://www.ime.usp.br/~pf/algoritmos/aulas/layout.html`.

7.2 Abreviaturas do Comando de Atribuição

Na linguagem C, é possível abreviar alguns comandos de atribuição.

7.2.1 Incremento e Decremento

É normal encontramos comandos de repetição onde é necessário o incremento ou decremento de uma variável, em geral, relacionado à condição de parada da repetição. No exemplo anterior, tivemos a atribuição

```
num = num + 1;
```

que significa aumentar de um o conteúdo da variável `num`. Na linguagem C é possível abreviar este comando fazendo:

```
num++;
```

Exemplos de outras abreviaturas:

Descrição	Exemplo	Comando Abreviado
Incremento de um	`i = i + 1`	`i++`
Decremento de um	`n = n - 1`	`n--`
Incremento por uma variável	`soma = soma + num`	`soma += num`
Decremento por uma vairável	`soma = soma - num`	`soma -= num`
	`mult = mult * num`	`mult *= num`
	`divd = divd / num`	`divd /= num`
	`rest = rest % num`	`rest %= num`

Usando estas abreviaturas, o trecho de programa anterior poderia ser escrito como:

```
num = 1;
soma = 0;
while (num<n) {
  if (n % num == 0)
    soma = soma + num;
  num = num + 1;
}
if (soma == n)
  printf ("numero perfeito\n");
else
  printf ("nao eh perfeito\n");
```

```
num = 1;
soma = 0;
while (num<n) {
  if (n % num == 0)
    soma += num;
  num++;
}
if (soma == n)
  printf ("numero perfeito\n");
else
  printf ("nao eh perfeito\n");
```

7.2.2 Atribuição Múltipla

Em certas situações, podemos ter atribuições de um mesmo valor para várias variáveis. No exemplo abaixo, o valor 0 (zero) pode ser atribuido às variáveis `i`, `soma` e `count` usando as seguintes formas:

```
i = 0;
soma = 0;
count = 0;
```

```
i = soma = count = 0;
```

Para entender o que ocorre no caso de atribuições múltiplas mostradas na coluda da direita, basta observar que o operador de atribuição `=` "devolve" o valor atribuido a uma variável, e sua precedência é da direita para esquerda. Assim, a atribuição `count = 0;` "devolve" o valor zero para ser atribuido a `soma`, que por sua vez "devolve" o mesmo valor para ser atribuido para `i`.

7.2.3 Atribuição na Declaração de Variáveis

Em alguns programas é necessário inicializar algumas variáveis. Por exemplo, no trecho de programa anterior, inicializar as variáveis `soma` e `num`.

É possível fazer estas inicializações destas variáveis na declaração das mesmas. Por exemplo:

```c
# include <stdio.h>

int main () {
  int num, soma, n;

  printf ("Entre com n > 0: ");
  scanf ("%d", &n);

  num = 1;
  soma = 0;
  while (num<n) {
    if (n % num == 0)
      soma += num;
    num++;
  }
  if (soma == n)
    printf ("numero perfeito\n");
  else
    printf ("nao eh perfeito\n");

  return 0;
}
```

```c
# include <stdio.h>

int main () {
  int num = 1, soma = 0, n;

  printf ("Entre com n > 0: ");
  scanf ("%d", &n);

  while (num<n) {
    if (n % num == 0)
      soma += num;
    num++;
  }
  if (soma == n)
    printf ("numero perfeito\n");
  else
    printf ("nao eh perfeito\n");

  return 0;
}
```

Note que no lado esquerdo, a inicialização das variáveis ocorre nos comandos de atribuição imediatamente anterior ao comando de repetição; enquanto que no lado direito, a inicialização das variáveis é feita na declaração das mesmas.

7.3 Definição de Constantes

É possível definir constantes na linguagem C. Uma constante em um programa é um valor que não muda durante a sua execução. Diferente de uma variável, que durante a execução do programa pode assumir diferentes valores.

Vamos começar dando um exemplo. Suponha que no seu programa, você queira definir uma constante de nome `UM` que seja o número inteiro um e uma outra constante de nome `ZERO` que seja o número inteiro zero.

Aí, é possível fazer a seguinte atribuição:

```c
num  = UM;
soma = ZERO;
```

Para definir uma constante, você deve fazer:

```c
# define <NOME_DA_CONSTANTE> <VALOR>
```

Assim, para nosso exemplo, para definir `UM` e `ZERO` devemos fazer

```c
# define UM 1
# define ZERO 0
```

Um programa completo com as constantes pode ser visto na figura 10.

```c
# include <stdio.h>

# define UM 1
# define ZERO 0

int main () {
  int num = UM, soma, n;
  printf ("Entre com n > 0: ");
  scanf ("%d", &n);

  soma = ZERO;
  while (num<n) {
    if (n % num == ZERO)
      soma += num;
    num++;
  }
  if (soma == n)
    printf ("numero perfeito\n");
  else
    printf ("nao eh perfeito\n");

  return 0;
}
```

Figura 10: Programa que utiliza constantes.