8. Estruturas de repetição 2

DIM0321

2015.1

Sumário

1. Laços `for`
2. Operadores de incremento
3. Elementos avançados
4. Desvios de fluxo

1 Laços `for`

2 Operadores de incremento

3 Elementos avançados

4 Desvios de fluxo

Descrição

Sintaxe

```
1 for (comando_init; condicao; comando_last) comando;
2
3 for (comando_init; condicao; comando_last) {
4      comando1;
5      comando2;
6 }
```

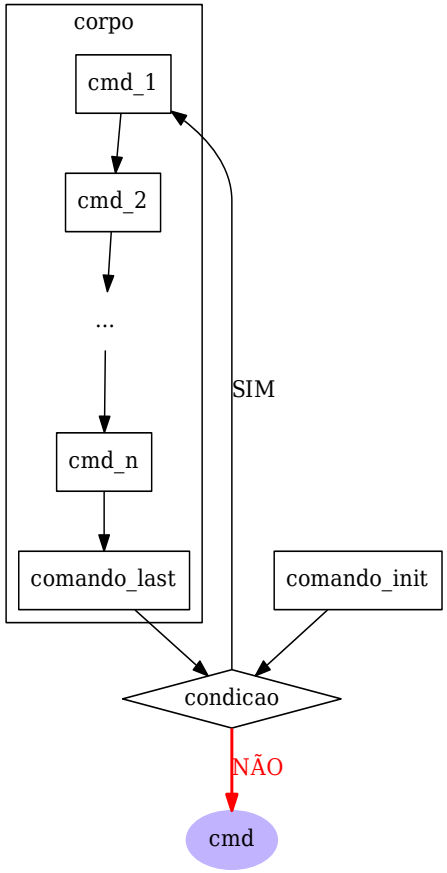
Observação

- `comando_init` descreva comandos de inicialização do laço
- `condicao` é a condição de execução
- `comando_last` é o comando executado apos cada execução do corpo do laço

Semântica informal

1. Executar `comando_init`. Ir a 2
2. Avaliar `condicao`.
   - Se for `verdadeira` (i.e. `!= 0`), ir a `body`.
   - Senão, executar o primeiro comando apos o laço.
3. Executar o corpo. Ir a 4.
4. Executar `comando_last`. Voltar a 2.

Fluxograma


corpo
cmd_1
cmd_2
...
cmd_n
comando_last
comando_init
SIM
condicao
NÃO
cmd

Exemplos

O que fazem os trechos de código abaixo ?

```
1 for (i = 0; i < 10; i++)
2       printf("%d\n", i);
```

Exemplo

```
1 for (x = 0; x < 100; x += 10)
2       printf("%d\n", x);
```

Exemplo

```
1 for (y = 0; y < 1000; y += y)
2       printf("%d\n", y);
```

Selecionar estruturas

- Os laços `while`, `do...while` e `for` são equivalentes em termos de poder expressivo.
- A escolha se faz de acordo com o entendimento do código.
  - `while`
    - ⋆ o bloco não deve ser necessariamente executado
    - ⋆ a condição deve ser testada antes do bloco
  - `do...while`
    - ⋆ o bloco deve ser executado pelo menos uma vez
    - ⋆ a condição deve ser testada depois do bloco
  - `for`
    - ⋆ conhecemos o número de vezes que o bloco será executado

1 Laços `for`

2 Operadores de incremento

3 Elementos avançados

4 Desvios de fluxo

Operadores de atribuição compostos

Simplificações

Expressão	Curta
`x = x + 2`	`x += 2`
`x = x - 2`	`x -= 2`
`x = x / 2`	`x /= 2`
`x = x * 2`	`x *= 2`

Em resumo

- 4 operadores : `+=`, `-=`, `/=`, `*=`

Operadores de in/de-crementos

Visão sintética

Expressão	Curta	Mais curta
`x = x + 1`	`x += 1`	`x++` ou `++x`
`x = x - 1`	`x -= 1`	`x--` ou `--x`

Forma prefixa

A expressão `++var`

1. incrementa o valor da variável `var`
2. retorna esse novo valor

Forma pos-fixa

A expressão `var++`

1. retorna o valor de `var`
2. incrementa o valor da variável

Exemplo

```c
#include <stdio.h>

int main(void)
{
    int tempo = 100;
    int velocidade = 30;

    velocidade = velocidade + 10;  /* velocidade vale 40 */
    velocidade += 15;              /* velocidade vale 55 */
    velocidade++;                  /* velocidade vale 56 */

    tempo = tempo - 5;             /* tempo vale 95 */
    tempo -= 10;                   /* tempo vale 85 */
    tempo--;                       /* tempo vale 84 */

    return 0;
}
```

Padrão de código: o contador

Objetivo

Contar um número de vezes que um laço é executado

```
1 int i = 0;
2
3 while (c) {
4       /* Do something */
5       i++;
6 }
```

```
1 for (i = 0; c; i++) {
2       /* Do something */
3 }
```

Exemplo

Assunto

Reescrever `fatorial` com laço `for`.

```c
#include <stdio.h>

int main(void)
{
    int n;
    int i;
    int fact;

    scanf("%i", &n);

    for(i = n, fact = 1; i >= 2; fact *= i, i--);
    printf("Fact(n) = %i\n", fact);

    return(0);
}
```

1 Laços `for`

2 Operadores de incremento

3 Elementos avançados

4 Desvios de fluxo

A vírgula ,

Elementos

- Separador de sequência
- A vírgula de separação nas declarações é diferente!

```
1 int x, y;
2
3 for (i = 0; i < 10; x += i, i++);
```

Semântica

- Avaliação da sequência de expressão esquerda à direita
- O valor retornado é o valor da última expressão

Exemplos

Versão 1

```c
#include <string.h>

void reverse (char s[])
{
     int i, j;
     for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {
          char tmp = s[i];
          s[i] = s[j];
          s[j] = tmp;
     }
}
```

Versão 2

```c
#include <string.h>

void reverse (char s[])
{
     int i, j;
     char c;
     for (i = 0, j = strlen(s) - 1; i < j; i++, j--)
          c = s[i], s[i] = s[j], s[j] = c;
}
```

Versão 3

Inicializações e atualizações

```c
#include <stdio.h>
int main(void)
{
      int i = 0, k;

      for (; i < 10; i++) printf("%d\n", i);

      for (int j = 0; ; j++) {
           printf("%d\n", j);
           if (j == 10) break;
      }

      k = 0;
      for (;;) {
           printf("%d\n", k);
           k++;
           if (k == 10) break;
      }

}
```

1 Laços `for`

2 Operadores de incremento

3 Elementos avançados

4 Desvios de fluxo

Elementos

Observação

A saída dos comandos de laços é dependente do teste.

Então

Porém, podemos modificar como corre a saída do ciclo:

- `break` termina imediatamente a execução do laço e o fluxo passa a linha seguinte ao bloco de comandos
- `continue` o fluxo passa imediatamente ao início do bloco de comandos
- `goto` pula até um ponto nomeado do código.

Break

Descrição

- O comando `break` permite sair mais cedo dos laços `for`, `do` e `while` bem como do `switch`.
- A saída é **imediata**.
- O fluxo de execução continua com o próximo comando após o laço ou o `switch`.

Continue

- O comando `continua` causa a execução imediata do próximo passo de execução de um laço
  - Nos laços `while` e `do`, a parte do teste (a condição) é executada
  - Nos laços `for`, o controle passa à parte da etapa de incrementação do laço

```c
#include <stdio.h>

int main(void) {
  int x;

  for (i = 0; i < 50; i++) {
    if (x % 2 == 0) { continue; }
    printf("%i\n", i);
  }
  return(0);
}
```

Goto & labels

- Um label permite nomear uma linha de código
- O `goto` permite pular até um label dado e continuar a execução do código a partir desse label
- O uso de `goto` deve ser raro porque atrapalha o entendimento do código

E.Dijsktra

Go-to statement considered harmful.
– E. W. Dijkstra [Dij68]

Exemplo

- goto pode ser visto como um break superpoderoso

Exemplo

```
1  for (...)
2       for (..) {
3            ...
4            if (disaster)
5                 goto error;
6       }
7  ...
8  error:
9  // limpar tudo aqui
10 // sair com elegância
```

Exemplo (equivalência)

switch .. break

```c
#include <stdio.h>
int main(void)
{
     unsigned int digits_less5, puncts;
     unsigned others;
     int c;
     digits_less5 = 0;
     puncts = 0;
     others = 0;
     while ((c = getchar()) != EOF) {
          switch (c) {
          case '0': case '1': case '2':
          case '3': case '4': case '5':
               digits_less5++;
               break;
          case '!': case '?': case ':':
          case ';': case '.': case ',':
               puncts++;
               break;
          default:
               others++;
               break;
          }
     }
}
```

switch .. goto .. continue

```c
#include <stdio.h>
int main(void)
{
     unsigned int digits_less5, puncts;
     unsigned others;
     int c;
     digits_less5 = 0;
     puncts = 0;
     others = 0;
     while ((c = getchar()) != EOF) {
          switch (c) {
          case '0': case '1': case '2':
          case '3': case '4': case '5':
               digits_less5++;
               goto next;
          case '!': case '?': case ':':
          case ';': case '.': case ',':
               puncts++;
               goto next;
          default:
               others++;
          }
     next: continue;
     }
}
```

Referências

Precisões

[KR88] Seções 3.5 – 3.8

[Bac13] 5.3, 5.5 – 5.8


André Backes, *Linguagem C completa e descomplicada*, Elsevier, 2013.


Edsger W. Dijkstra, *Letters to the editor: Go to statement considered harmful*, Commun. ACM **11** (1968), no. 3, 147–148.


Brian W. Kernighan and Dennis M. Ritchie, *The (ANSI) C Programming Language*, 2nd ed., Prentice Hall Professional Technical Reference, 1988.

Perguntas ?

http://dimap.ufrn.br/~richard/dim0321