Programação Estruturada

Aula 7 - Comandos de repetição

Yuri Malheiros (yuri@ci.ufpb.br)

for

- O `for` é o comando de repetição mais versátil
- Ele normalmente é utilizado para laços contados, mas ele pode ir além

```
for (expressao1; expressao2; expressao3) comando
```

for

```
for (i=10; i>0; i--)
    printf("%d\n", i);
```

- Quando o `for` é iniciado, a `expressao1` é executada
  - Ou seja, `i=10`
- Em seguida, `expressao2` é avaliada, ela é a expressão de controle
- Se `expressao2` for verdadeira, então o corpo do laço é executado
- Por fim, `expressao3` é executada
- Uma nova iteração do laço recomeça e `expressao2` é testada novamente

for

- O laço abaixo

```
for (i=10; i>0; i--)
    printf("%d\n", i);
```

- Pode ser substituído por:

```
i=10;

while (i>0) {
    printf("%d\n", i);
    i--;
}
```

for

- Existem alguns padrões que aparecem com frequência nos programas
- Contar de 0 até n-1
  - `for (i=0; i<n; i++)`
- Contar de 0 até n
  - `for (i=0; i<=n; i++)`
- Contar de 1 até n
  - `for (i=1; i<=n; i++)`

for

- As expressões no `for` não são obrigatórias, todas elas podem ser omitidas

```
i=10;

for (; i>0; i--)
    printf("%d\n", i);
```

for

- As expressões no `for` não são obrigatórias, todas elas podem ser omitidas

```
for (i=10; i>0;)
    printf("%d\n", i--);
```

for

- As expressões no `for` não são obrigatórias, todas elas podem ser omitidas

```
i = 10;
for (; i>0;)
    printf("%d\n", i--);
```

- Ficou semelhante ao `while`

for

- Omitindo todas as expressões, temos um loop infinito

```
for (;;)
    printf("loop infinito!\n");
```

for

- Muitas vezes, a variável que manipulamos nas expressões do `for` só é utilizada no próprio `for`
- Por isso, é possível declarar e inicializar a variável no `for`
- `for (int i=0; i<n; i++)`
- Essa variável só existe dentro do `for` e não pode ser acessada fora dele

Operador vírgula

- Em alguns casos, precisamos de mais de uma inicialização ou mais de um incremento
- Podemos fazer isso usando o operador `,`
- `expressao1 , expressao2`

Operador vírgula

- `expressao1 , expressao2`
- Primeiro, a `expressao1` é avaliada
- Segundo, a `expressao2` é avaliada e seu valor é retornado
- Por isso, se `expressao1` não tiver um efeito colateral, ela não serve para nada

Operador vírgula

```c
int i=1;
int j=5;
int k;

k = (++i, i+j);

printf("%d\n", k);
```

Operador vírgula

- A precedência do operador `,` é a menor de todas
- `i=1, j=2, k=i+j` é equivalente a `((i=1), (j=2)), (k=(i+j))`

for - exemplo

- Vamos criar um programa que soma os valores de 1 até 10

```
int soma = 0;
int i;

for (i=1; i<=10; i++)
    soma += i;

printf("%d\n", soma);
```

for - exemplo

- Vamos criar um programa que soma os valores de 1 até 10

```
int soma;
int i;

for (soma=0, i=1; i<=10; i++)
    soma += i;

printf("%d\n", soma);
```

Saindo de um laço

- Usando o comando `break` podemos interromper a execução de um laço a qualquer momento
- Vamos criar um programa que verifica se um número é primo

Saindo de um laço

```c
int n;
int d;

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

for (d=2; d<n; d++) {
    if (n%d == 0) {
        break;
    }
}

if (d < n) {
    printf("%d não é primo\n", n);
} else {
    printf("%d é primo\n", n);
}
```

Saindo de um laço - exemplo

- Vamos fazer um programa que calcula o cubo de um número
- Ao exibir o resultado, o programa deve perguntar novamente ao usuário para ele digitar um novo número
- Se o usuário digitar 0 o programa deve ser finalizado

Saindo de um laço

- O comando `continue` termina a execução da iteração atual do laço e vai para a próxima iteração
    - O `break` termina a execução completa do laço

Saindo de um laço

```
int i;

for (i=1; i<=10; i++) {
    if (i%2 != 0)
        continue;

    printf("%d\n", i);
}

return 0;
```