

Fundamentos de Programação

Controle de fluxo com break e continue

Dainf - UTFPR

Profa. Leyza B. Dorini
Prof. Bogdan T. Nassu

break e continue

Os comandos `break` e `continue` servem para modificar o fluxo de um *loop* durante a sua execução.

`break`: interrompe imediatamente o loop.

`continue`: inicia imediatamente a próxima iteração do loop, pulando o restante do bloco.

Nenhum dos dois é obrigatório. Tudo o que se pode fazer com estes comandos pode ser obtido por outros meios...

- ... às vezes usar `break` e `continue` é menos trabalhoso!
- ... frequente fazer com `break` e `continue` é menos elegante!

break

Lembrando: usando *flag* para controlar o fluxo

Este programa informa se um dado número *n* é primo. Usamos a *flag* para interromper o *loop* quando um divisor de *n* é encontrado.

```
1      int n, div, eh_primo;
2
3      scanf("%d", &n);
4
5      eh_primo = 1;
6      for (div = 2; div < n && eh_primo; div++)
7          if (n%div == 0)
8              eh_primo = 0;
9
10     if (eh_primo)
11         printf ("Eh primo.\n");
12     else
13         printf ("Nao eh primo.");
```

Usando break para controlar o fluxo

Nesta versão, usamos o `break` para interromper o *loop* quando um divisor de `n` é encontrado. O resto do programa não muda!

```
1     for (div = 2; div < n; div++)
2         if (n%div == 0)
3             {
4                 eh_primo = 0;
5                 break;
6             }
```

As duas soluções não são **exatamente** equivalentes. Se um divisor for encontrado, na versão somente com a *flag*, o valor de `div` vai ser incrementado antes do *loop* terminar. Já com o `break`, o *loop* será interrompido imediatamente, sem incrementar `div` mais uma vez! Então, se quisermos saber qual foi o divisor encontrado, na versão com o `break` é `div`, mas na versão sem o `break` é `div-1`.

Com break × sem break

```
1     for (div = 2; div < n && eh_primo; div++)
2         if (n%div == 0)
3             eh_primo = 0;
4     else
5         printf ("%d nao eh divisor.\n", div);
```

```
1     for (div = 2; div < n; div++)
2         if (n%div == 0)
3         {
4             eh_primo = 0;
5             break;
6         }
7     printf ("%d nao eh divisor.\n", div);
```

A parte abaixo do bloco do if jamais será atingida após o break, então não precisamos “protegê-la” com o else.

Outro exemplo (com o break)

O programa abaixo mostra o menor divisor > 1 comum a $v1$ e $v2$. Note que usamos o valor de div para saber se o *loop* foi até o final.

```
1      int v1, v2, div;
2
3      scanf ("%d %d", &v1, &v2);
4
5      for (div = 2; div <= v1 && div <= v2; div++)
6          if (v1 % div == 0 && v2 % div == 0)
7              break;
8
9      if (div > v1 || div > v2)
10         printf ("Nenhum divisor maior que 1.\n");
11     else
12         printf ("O menor divisor eh %d\n", div);
```

Outro exemplo (sem o break)

Usando uma *flag*, o teste final é mais claro, mas precisamos lembrar que o incremento ocorrerá antes do *loop* ser interrompido!

```
1      int v1, v2, div, achou = 0;
2
3      scanf ("%d %d", &v1, &v2);
4
5      for (div = 2; div <= v1 && div <= v2 && !achou; div++)
6          if (v1 % div == 0 && v2 % div == 0)
7              {
8                  achou = 1;
9                  div--; // Porque logo depois vem um div++
10             }
11
12     if (!achou)
13         printf ("Nenhum divisor maior que 1.\n");
14     else
15         printf ("O menor divisor eh %d\n", div);
```


continue

Usando continue para controlar o fluxo

O `continue` pula todo o restante de um bloco e inicia imediatamente a próxima iteração de um *loop*. Ele **executa** o incremento e o teste da condição!

```
1 // Somando numeros de 0 a N, pulando os divisores de 3.
2
3     int i, total = 0;
4
5     for (i = 0; i < N; i++)
6     {
7         if (i%3 == 0)
8             continue;
9         total += i;
10    }
11
12    printf ("%d\n", total);
```

Com continue × sem continue

Na maior parte dos casos, podemos evitar o uso do continue...

```
1   for (i = 0; i < N; i++)
2   {
3       if (i%3 == 0)
4           continue;
5       total += i;
6   }
```

```
1   for (i = 0; i < N; i++)
2       if (i%3 != 0)
3           total += i;
```

break e continue: como não usar?

Código feio!

As construções abaixo, apesar de funcionarem costumam ser confusas ou inúteis. Evite-as!!! (Mas, como sempre, não existe uma regra absoluta!)

```
1 while (1)
2 {
3     (...)
4     if (aconteceu alguma coisa)
5         break;
6 }
```

```
1 while (alguma condicao)
2 {
3     if (aconteceu alguma coisa)
4         continue;
5     else
6         faz alguma coisa interessante
7 }
```

break e continue em *loops* aninhados

O break interrompe o *loop* interno!

```
1  (...)
2  while (foo)
3  {
4      while (bar)
5      {
6          (...)
7          if (alguma_condicao)
8              break;
9          (...) // Pula esta parte se entrar no if.
10     }
11
12     (...) // O break manda o fluxo para ca.
13 }
14 (...)
```

O `continue` vai para a próxima iteração do *loop* interno!

```
1  (...)
2  while (foo)
3  {
4      while (bar) // O continue volta para ca.
5      {
6          (...)
7          if (alguma_condicao)
8              continue;
9          (...) // Pula esta parte se entrar no if.
10     }
11
12     (...) // O continue nao afeta esta parte.
13 }
14 (...)
```