11 Repetições Encaixadas

Ronaldo F. Hashimoto e Carlos H. Morimoto

O conceito de repetições encaixadas nada mais é que uma repetição dentro de uma outra repetição. Ao final dessa aula você deverá ser capaz de:

- Simular programas com repetições encaixadas.
- Utilizar repetições encaixadas em seus programas.

11.1 Repetições Encaixadas

Até o momento, nossos programas foram simples e os problemas não exigiam muitos comandos dentro de uma repetição. Mas os comandos dentro de uma repetição podem se tornar bastante complexos. Nessa aula, vamos ver como alguns problemas exigem a criação de repetição dentro de repetição (ou repetições encaixadas), como por exemplo:

```
1       while (<condição 1>) {
2         while (<condição 2>) {
3           while (<condição 3>) {
4           }
5         }
6       }
```

A seguir vamos apresentar esse recurso através de um problema.

11.2 Exemplo de um Problema que usa Repetições Encaixadas

Problema: Dados dois naturais `n>0` e `m>0`, determinar entre todos os pares de números inteiros `(x,y)` tais que $0 \leq x \leq n$ e $0 \leq y \leq m$, um par para o qual o valor da expressão $x \times y - x^2 + y$ seja máximo e calcular também este máximo. Em caso de empate, imprima somente um valor.

Exemplo: para `n=2` e `m=3`, a saída do programa deve ser $(1, 3)$ com valor máximo igual a `5`.

Solução:

- Tomando o exemplo `n = 2` e `m = 3`;
- Gerar todos os pontos (x, y) tais que $0 \leq x \leq n$ e $0 \leq y \leq m$.
- Para cada `x` fixo, temos que gerar os valores para `y` de `0` a `m=3`. Assim, temos:

  `x=0`

  $(0,0)$ $(0,1)$ $(0,2)$ $(0,3)$ $\Rightarrow$ Para `x=0`, um laço para gerar os valores de `y` de `0` a `3`.

  `x=1`

  $(1,0)$ $(1,1)$ $(1,2)$ $(1,3)$ $\Rightarrow$ Para `x=1`, um laço para gerar os valores de `y` de `0` a `3`.

  `x=2`

  $(2,0)$ $(2,1)$ $(2,2)$ $(2,3)$ $\Rightarrow$ Para `x=2`, um laço para gerar os valores de `y` de `0` a `3`.
- Assim, para cada `x` fixo, temos uma repetição para gerar os valores para `y` de `0` a `3`.

```
/* para um valor fixo de x */
y = 0;
while (y <= m) {
  printf ("%d, %d\n", x, y);
  y = y + 1;
}
```

- Agora é necessário ter uma repetição para gerar os valores de `x` de `0` a `2`.

```
x = 0;
while (x <= n) {
  /* gera um valor para x */
  x = x + 1;
}
```

- Note que a repetição que gera a sequência para `y` deve estar dentro da repetição que gera a sequência para `x`
- Assim, juntando os códigos, temos

```
x = 0;
while (x <= n) {
  /* gera um valor para x */
  /* para um valor fixo de x */
  y = 0;
  while (y <= m) {
    printf ("%d, %d\n", x, y);
    y = y + 1;
  }
  x = x + 1;
}
```

- Agora temos que detectar um par `(x,y)` para o qual o valor da expressão $\texttt{x} \times \texttt{y} - \texttt{x}^2 + \texttt{y}$ seja máximo e calcular também este máximo.
- Para cada par `(x,y)` gerado pelo código anterior, calcular o valor da expressão $\texttt{x} \times \texttt{y} - \texttt{x}^2 + \texttt{y}$.

  `x=0`

  $(0,0) \Rightarrow 0 \quad (0,1) \Rightarrow 1 \quad (0,2) \Rightarrow 2 \quad (0,3) \Rightarrow 3$

  `x=1`

  $(1,0) \Rightarrow -1 \quad (1,1) \Rightarrow 1 \quad (1,2) \Rightarrow 3 \quad (1,3) \Rightarrow 5$

  `x=2`

  $(2,0) \Rightarrow -4 \quad (2,1) \Rightarrow -1 \quad (2,2) \Rightarrow 2 \quad (2,3) \Rightarrow 5$

- Assim, basta ter uma variável `max` que, para cada par gerado, guarda o valor máximo até o presente momento, de forma que:

  `x=0`

  $(0,0) \Rightarrow 0, \texttt{max=0} \quad (0,1) \Rightarrow 1, \texttt{max=1} \quad (0,2) \Rightarrow 2, \texttt{max=2} \quad (0,3) \Rightarrow 3, \texttt{max=3}$

  `x=1`

  $(1,0) \Rightarrow -1, \texttt{max=3} \quad (1,1) \Rightarrow 1, \texttt{max=3} \quad (1,2) \Rightarrow 3, \texttt{max=3} \quad (1,3) \Rightarrow 5, \texttt{max=5}$

  `x=2`

  $(2,0) \Rightarrow -4, \texttt{max=5} \quad (2,1) \Rightarrow -1, \texttt{max=5} \quad (2,2) \Rightarrow 2, \texttt{max=5} \quad (2,3) \Rightarrow 5, \texttt{max=5}$

- Para cada par gerado, calcula-se o valor da expressão $\texttt{v} = \texttt{x} \times \texttt{y} - \texttt{x}^2 + \texttt{y}$ e testa se $\texttt{v} > \texttt{max}$. Em caso afirmativo, o valor de `max` deve ser atualizado para `v` fazendo $\texttt{max} = \texttt{v}$;
- Temos então o seguinte código:

```
1       x = 0;
2       while (x <= n) {
3         /* gera um valor para x */
4         y = 0;
5         while (y <= m) {
6           v = x*y - x*x + y;
7           if (v > max)
8             max = v;
9           y = y + 1;
10        }
11        x = x + 1;
12      }
13      printf ("O valor maximo = %d\n", max);
```

- O código anterior consegue determinar o valor máximo, mas não consegue detectar qual par `(x,y)` tem este valor.
- Para detectar qual par `(x,y)` tem o valor máximo, temos que guardá-lo toda vez que a variável `max` é atualizada:

```
1       x = 0;
2       while (x <= n) {
3         /* gera um valor para x */
4         y = 0;
5         while (y <= m) {
6           v = x*y - x*x + y;
7           if (v > max) {
8             max = v;
9             x_max = x;
10            y_max = y;
11          }
12          y = y + 1;
13        }
14        x = x + 1;
15      }
16      printf ("O valor maximo = %d em x = %d e y = %d\n", max, x_max, y_max);
```

- Falta ainda resolver um problema: quais devem ser os valores iniciais para `max`, `x_max` e `y_max`?
- O problema de iniciarmos `max` com 0 é que se os valores calculados de `v` forem todos negativos, a variável `max` nunca será atualizada.
- O problema de iniciarmos `max` com um número negativo (digamos -1000) é que se os valores calculados de `v` forem todos menores que -1000, a variável `max` nunca será atualizada.
- Uma boa idéia é inicializarmos `max` com um valor que `v` assume. Poderia ser qualquer valor da sequência `0, 1, 2, 3, -1, 1, 3, 5, -4, -1, 2, 5`.
- Assim, podemos escolher um par `(x,y)` que é gerado. Por exemplo, o par $(0, 0)$ e calcular o valor da expressão $\texttt{v} = \texttt{x} \times \texttt{y} - \texttt{x}^2 + \texttt{y}$ para este par. Neste caso, $\texttt{v} = 0$ e colocamos para $(\texttt{x\_max}, \texttt{y\_max})$ o par $(0, 0)$.
- A solução final seria:

```c
# include <stdio.h>

int main () {
  int x, y, n, m, v, x_max, y_max;

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

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

  x = 0; max = x_max = y_max = 0;
  while (x <= n) {
    /* gera um valor para x */
    y = 0;
    while (y <= m) {
      v = x*y - x*x + y;
      if (v > max) {
        max = v;
        x_max = x;
        y_max = y;
      }
      y = y + 1;
    }
    x = x + 1;
  }
  printf ("O valor maximo = %d em x = %d e y = %d\n", max, x_max, y_max);
  return 0;
}
```

11.3 Exercícios Recomendados

1. Dados `n>0` números inteiros positivos, calcular a soma dos que são primos.

   (a) Sua solução deve conter uma repetição com contador para ler `n` números pelo teclado.

   (b) Para cada número lido, seu programa deve testar se ele é primo (usando uma outra repetição com indicador de passagem). Em caso afirmativo, acumular em uma soma.