Introdução a Técnicas de Programação

Estruturas de repetição (laços)

Estruturas de repetição
Prof. André Campos

Estruturas de repetição

O número de repetições pode ser

- Determinado: sabemos quantas vezes o bloco será executado
- Indeterminado: não sabemos de antemão quantas vezes o bloco será executado. Dependerá de uma condição.

A condição de parada pode ser

- No início do bloco de comandos: a verificação é feita antes da execução do bloco
- No final do bloco de comandos: o bloco é executado pelo menos uma vez

```
do {
    // comandos
} while (condição);
```

```
while (condição) {
    // comandos
}
```

```
for (repetição) {
    // comandos
}
```

Mas antes...

"*Syntax sugar*" (forma reduzida de algumas operações)
Quando as operações matemáticas (+, -, *, /, %) atuam sobre uma variável para alterar a própria variável, elas pode ser reescritas

`i = i + 10` pode ser escrito como `i += 10`
`i = i - 5` pode ser escrito como `i -= 5`
`i = i * a` pode ser escrito como `i *= a`
`i = i / (5 - a)` pode ser escrito como `i /= (5 - a)`

Com operações + e -, quando o valor é apenas 1, há uma forma mais simples

`i = i + 1` pode ser escrito como `i += 1` ou **`i++`** ou **`++i`**
`i = i - 1` pode ser escrito como `i -= 1` ou **`i--`** ou **`--i`**

Operadores de incremento e decremento

Onde os operadores se encontram definem o valor de retorno (antes ou depois da operação)

- `i++` e `i--` retornam o valor da variável primeiro, e depois incrementam o valor
- `++i` e `--i` incrementam o valor primeiro e depois retornam o valor da variável

Exemplo

```
int valor = 5;
int teste;

teste = valor++; // (teste = 5, valor = 6)
teste = ++valor; // (teste = 7, valor = 7)
```

While (enquanto)

Executa um bloco de comando **enquanto** uma condição for verdadeira

Teste da condição é realizado **antes** da execução do bloco

- Se a condição for verdadeira, executa o bloco e volta a testar a condição novamente
- Se a condição for falsa, continua a execução do programa após o bloco de comandos

O que o código faz?

```
#include <iostream>
using namespace std;

int main() {
  int n, fat = 1;

  cin >> n;
  while (n > 0) {
    fat = fat * n;
    n--;
  }
  cout << fat << endl;

  return 0;
}
```

Dicas para entender os laços

Construa uma tabela para ver a evolução das variáveis a cada iteração do laço.

Digamos que o dado de entrada (n) seja 6

fat							
n							

```cpp
#include <iostream>
using namespace std;

int main() {
  int n, fat = 1;

  cin >> n;
  while (n > 0) {
    fat = fat * n;
    n--;
  }
  cout << fat << endl;

  return 0;
}
```

Dicas para entender os laços

Construa uma tabela para ver a evolução das variáveis a cada iteração do laço.

Digamos que o dado de entrada (n) seja 6

fat	1	6	30	120	360	720	720
n	6	5	4	3	2	1	0

Resultado = 720

```cpp
#include <iostream>
using namespace std;

int main() {
  int n, fat = 1;

  cin >> n;
  while (n > 0) {
    fat = fat * n;
    n--;
  }
  cout << fat << endl;

  return 0;
}
```

Mistério 1

O que esse programa calcula?

> Dica: imagine casos de teste e veja como as variáveis estão se modificando em cada caso.

Para n = 3?

s							
i							

Para n = 6?

s							
i							

```
#include <iostream>
using namespace std;

int main() {
  int n, i, s;
  cin >> n;

  i = 1;
  s = 0;
  while ( i <= n ) {
    s += i;
    i++;
  }
  cout << s << endl;

  return 0;
}
```

Mistério 1

O que esse programa calcula?

> Dica: imagine casos de teste e veja como as variáveis estão se modificando em cada caso.

Para n = 3?

s	0	1	3	6			
i	1	2	3	4			

Para n = 6?

s	0	1	3	6	10	15	21
i	1	2	3	4	5	6	7

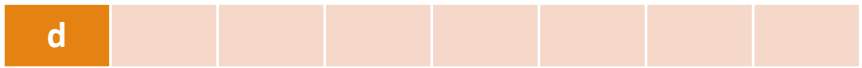
```
#include <iostream>
using namespace std;

int main() {
  int n, i, s;
  cin >> n;

  i = 1;
  s = 0;
  while ( i <= n ) {
    s += i;
    i++;
  }
  cout << s << endl;

  return 0;
}
```

Mistério 2

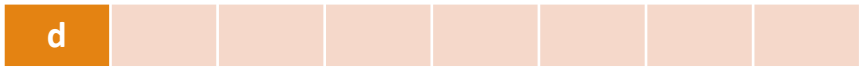
O que esse programa imprime?

Dica: imagine casos de teste e veja como as variáveis estão se modificando em cada caso.

Para n = 3?



Para n = 6?




```
#include <iostream>
using namespace std;

int main() {
  int n, d;
  cin >> n;

  d = 1;
  while ( d <= n ) {
    if ( n % d == 0 ) {
      cout << d << endl;
    }
    d++;
  }

  return 0;
}
```

Mistério 3 (*hard*)

O que esse programa imprime?

> Dica: imagine casos de teste e veja como as variáveis estão se modificando em cada caso.

Para n = 3?

i							
j							

Para n = 6?

i							
j							

```
#include <iostream>
using namespace std;

int main() {
    int n, i, j;
    cin >> n;

    i = 0;
    j = 1;
    while ( i <= n ) {
        cout << i << endl;
        j += i;
        i = j - i;
    }

  return 0;
}
```

Do...while (faça enquanto)

Executa um bloco de comando **enquanto** uma condição for verdadeira

Teste da condição é realizado **depois** da execução do bloco

- Executa o bloco e depois testa a condição
- Se a condição for verdadeira, executa o bloco e volta a testar a condição novamente
- Se a condição for falsa, continua a execução do programa após o bloco de comandos

O que o código faz?

```
#include <iostream>
using namespace std;

int main() {
 int n, sum = 0;
 cin >> n;

 do {
   sum += n;
   n--;
 } while (n != 0);

 cout << sum << endl;
 return 0;
}
```

Laços infinitos

O que ocorreria com o programa anterior se o valor de entrada fosse negativo?

Por quê?

```
#include <iostream>
using namespace std;

int main() {
 int n, sum = 0;
 cin >> n;

 do {
   sum += n;
   n--;
 } while (n != 0);

 cout << sum << endl;
 return 0;
}
```

Laços infinitos

O que ocorreria com o programa anterior se o valor de entrada fosse negativo?

Por quê?

Condição de parada: garantia de término do programa

Verificar se, a cada iteração, as alterações das variáveis no laço aproxima a expressão de uma condição verdadeira.

```
#include <iostream>
using namespace std;

int main() {
 int n, sum = 0;
 cin >> n;

 do {
   sum += n;
   n--;
 } while (n != 0);

 cout << sum << endl;
 return 0;
}
```

Como evitar laços infinitos

Muitas vezes, é necessário checar casos específicos.

Casos de testes: explorar dados de entrada

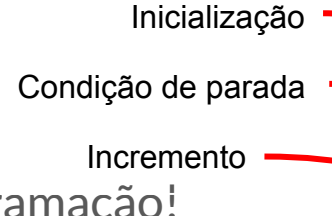
```cpp
#include <iostream>
using namespace std;

int main() {
  int n, sum = 0;
  cin >> n;

  while (n > 0) {
    sum += n;
    n--;
  };

  cout << sum << endl;
  return 0;
}
```

For (para...)

Normalmente usado em repetições em que sabemos o número de iterações a realizar.

Repita N vezes

Padrão comum em programação!




```cpp
#include <iostream>
using namespace std;

int main() {
  int n;
  cin >> n;

  // repete n vezes
  int i = 0;
  while (i < n) {
    // ...
    i++;
  };

  return 0;
}
```

For (para...)

Normalmente usado em repetições em que sabemos o número de iterações a realizar.

Repita N vezes

Padrão comum em programação!

Similar ao padrão com o `while`, só que mais enxuto!!!

```
#include <iostream>
using namespace std;

int main() {
  int n;
  cin >> n;
  // repete n vezes
  for (int i = 0; i < n; i++)
{
    // ...
  };

  return 0;
}
```

Inicialização

Condição de parada

Incremento

For (para...)

```
#include <iostream>
using namespace std;

void main() {
     int n;
     cin >> n;

     // repete n vezes
     int i = 0;
     while (i < n) {
          // ...
          i++;
     };
}
```

```
#include <iostream>
using namespace std;

void main() {
  int n;
     cin >> n;

  // repete n vezes
  for (int i = 0; i < n; i++) {
    // ...
  };
}
```

Inicialização

- Executa uma única vez
- Executa antes do bloco

For (para...)

```
#include <iostream>
using namespace std;

void main() {
    int n;
    cin >> n;

    // repete n vezes
    int i = 0;
    while (i < n) {
        // ...
        i++;
    };
}
```

```
#include <iostream>
using namespace std;

void main() {
  int n;
    cin >> n;

  // repete n vezes
  for (int i = 0; i < n; i++) {
    // ...
  };
}
```

Condição de parada

- Executa a cada iteração
- Executa antes do bloco

For (para...)

```
#include <iostream>
using namespace std;

void main() {
    int n;
    cin >> n;

    // repete n vezes
    int i = 0;
    while (i < n) {
        // ...
        i++;
    };
}
```

```
#include <iostream>
using namespace std;

void main() {
  int n;
    cin >> n;

  // repete n vezes
  for (int i = 0; i < n; i++) {
    // ...
  };
}
```

Incremento

- Executa a cada iteração
- Executa depois do bloco

For (para...)

Para `i` variando de `0` até `n-1`

```
for (int i = 0; i < n; i++) {
  // ...
};
```

Para `i` variando de `1` até `n`

```
for (int i = 1; i <= n; i++) {
  // ...
};
```

Para `i` variando de `n` até `1`

```
for (int i = n; i > 0; i--) {
  // ...
};
```

Para `i` variando de `10` até `100` de 2 em 2

```
for (int i = 10; i <= 100; i+=2){
  // ...
};
```

Padrões

Contador

```
int count = 0;
for (int i = 0; i < n; i++) {
  // ...
  if (condição) {
    count++;
  }
};
```

Conta quantos vezes uma determinada condição é verdadeira
Ex: quantos divisores, quantas letras...

Acumulador

```
int acc = 0;
for (int i = 0; i < n; i++) {
  // ...
  int val = expressão;
  acc += val; // ou outras
oper.
};
```

Acumulador
Acumula os valores de uma sequência.
Exs: somatório, produtório...

Interrupções nos laços

Há dois comandos que alteram o fluxo dentro de um laço

- break: termina o laço (segue para o próximo comando após ele)
- continue: volta para o início do laço
  - No caso do for, o incremento será executado, seguido do teste de condição de parada

```
for (int i = 0; i < 100; i++) {
  // ...
  if (condição) {
    break;
  }
};
// executa qdo i=100 ou após break
```

```
for (int i = 0; i < 100; i++) {
  if (condição) {
    continue;
  }
  // ...
};
```

Laços aninhados

Qualquer instrução pode ser inserida no bloco do laço... inclusive outros laços

Permite construções mais complexas

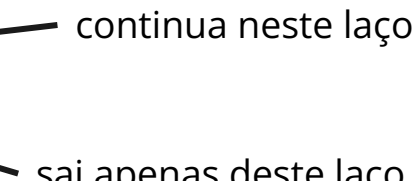
```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < i; j++) {
    // ...
    // ...
    // ...
  }
}
```

Laços aninhados

Qualquer instrução pode ser inserida no bloco do laço... inclusive outros laços

Permite construções mais complexas

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < i; j++) {
    if (condição) {
      break;
    }
  }
}
```



Um break (ou continue) funciona no laço em que ele se encontra

Práticas

1. Dado um número inteiro positivo N, imprimir as primeiras N potências de 2 (lembre-se que a 1ª potência de 2 é $2^0 = 1$.
2. Dado um número inteiro positivo N e dois números naturais não nulos I e J, imprimir em ordem crescente os N primeiros naturais múltiplos de I ou de J (ou de ambos).
3. Dizemos que um número natural é triangular se é o produto de três números naturais consecutivos. Por exemplo, 120 é triangular, pois 4 x 5 x 6 = 120. Dado um número N, imprimir na tela “TRIANGULAR” caso seja triangular ou “NÃO TRIANGULAR” caso não seja.
4. Dado um valor positivo N e uma sequência de N inteiros, imprimir quantos números da sequência são positivos e quantos são não-positivos (um número é não-positivo se é negativo ou se é igual a 0).

Práticas

5. Dado um valor inteiro N e uma sequência de N valores inteiros, imprimir a média dos valores da sequência
6. Dados dois valores inteiros N e V e uma sequência de N valores inteiros, imprimir quantas vezes V aparece na sequência.
7. Dado um valor inteiro N, imprimir todos os números primos menores ou igual a N.
8. Dado um inteiro N, imprimir as figuras a seguir com lados de tamanho N (caracteres)

Práticas

9. Um número inteiro é chamado de 'perfeito' se ele é igual a soma de seus divisores. Por exemplo, 6 é 'perfeito', pois 6 = 1 + 2 + 3. Dado um número inteiro positivo N, imprimir todos os números perfeitos menores que N.
10. Dado um número inteiro positivo n, verificar se o primeiro e o último dígito deste número são iguais. Caso sejam iguais imprima 1 (um) caso não seja imprima 0 (zero).
11. Dados dois números inteiros positivos A e B, representando a fração A/B, imprimir a fração irredutível de A/B.
12. Dados dois números inteiros positivos, determinar o máximo divisor comum entre eles. Dica: utilize o algoritmo de Euclides para cálculo do MDC.