INSTITUTO FEDERAL
Sul-rio-grandense
Câmpus Passo Fundo

Ciência da
Computação

ALGORITMOS I

Professores:

Adilso Nunes de Souza

Maikon Cismoski dos Santos

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

ROTEIRO DA AULA

- Conceitos de Variáveis
  - Contador
  - Acumulador
  - Flag
- Estrutura de Repetição
- Comando Enquanto / While
- Comando Repita / Do While
- Comando Break
- Comando Continue
- Exemplo de teste de mesa

CONCEITOS VARIÁVEIS

- Novos conceitos na utilização de variáveis:
  - As variáveis vistas até agora serviram sempre para o armazenamento de valores do programa.
  - Uma variável era definida, colocado um valor nela via teclado (quando era lida) ou via expressão numérica (comando de atribuição). Seu valor era utilizado ou em testes ou em cálculos e, finalmente, quando necessário, era exposto. Todavia, veremos outras formas de utilizações para as variáveis.

CONCEITOS VARIÁVEIS

- Os tipos de variáveis não mudam, ou seja, os tipos “inteiro”, “real”, “caracter”, entre outros são os mesmos; o que muda é o enfoque dado a estas variáveis.
- Elas serão especialmente úteis em programas com repetição.
- Em muitos casos serão os controladores dos comandos de repetição.
- São chamadas de contador, acumulador e flag.

CONTADOR

- Contador:
  - Variável do tipo numérica inteira, que serve para contar quantas vezes algum trecho de programa foi executado (quantas vezes alguma ação foi realizada).
  - Por exemplo: para sabermos quantas vezes foi executado um bloco de comandos, colocamos um comando a fim de incrementar ou decrementar a variável contadora a cada vez que o bloco é executado.
  - Deve ser previamente inicializada.

ACUMULADOR

- Acumulador:
  - Variável que serve para acumular valores, geralmente numéricos. Serve na prática para obtermos uma determinada soma. Por exemplo: para sabermos a soma de 30 números a serem lidos, junto com o trecho de leitura atualizamos a variável acumuladora, que já deve estar inicializada.

FLAG

- Flag:
  - Tradução do Inglês: bandeira, sinal é um tipo de variável para controle do estado de um programa. Usada em testes, serve na prática para sabermos se um laço pode ou não acabar, se uma fórmula foi ou não resolvida (o programador deve saber os motivos, o porquê da fórmula não ser resolvida).
  - Nos possibilita a comunicação entre diversas partes de um programa.

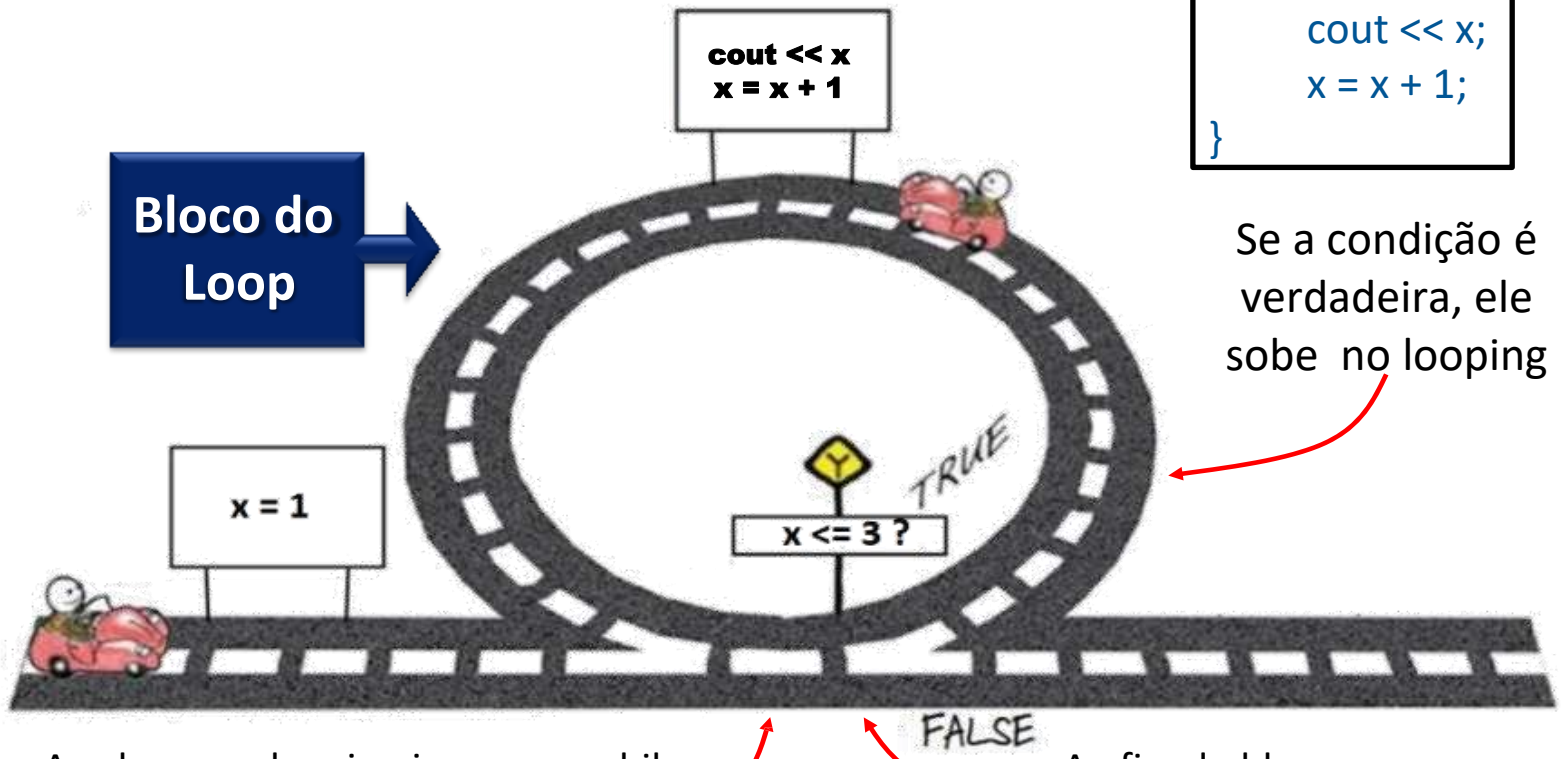
ESTRUTURAS DE REPETIÇÃO

- Como o próprio nome diz, são comandos que repetem sua execução (e de outros comandos relacionados) até um determinado ponto quando encerram. São vulgarmente conhecidos como “laços”, ou seja, quando chegamos ao final de um trecho de programa “envolto” por um laço, retornamos ao início do mesmo, e o executamos novamente.

ESTRUTURAS DE REPETIÇÃO

- Imprimindo números de 1 até 3

```
x = 1
while (x <= 3){
        cout << x;
        x = x + 1;
}
```



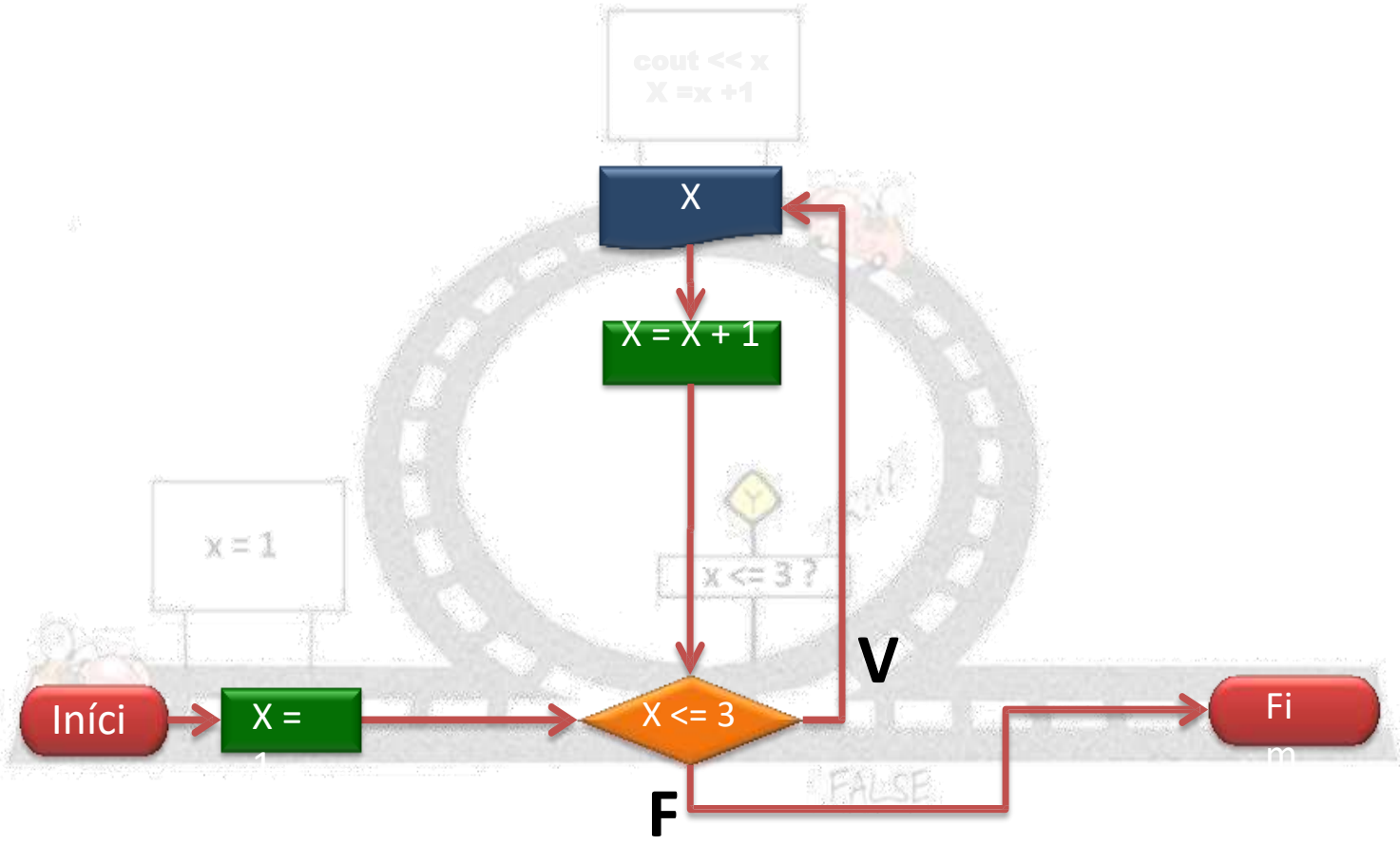
Se a condição é verdadeira, ele sobe no looping

Ao chegar pela primeira vez no while, o computador verifica a condição

Ao fim do bloco, o programa volta a verificar a condição

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

ESTRUTURAS DE REPETIÇÃO



COMANDO ENQUANTO

- Estrutura de controle de fluxo de execução que permite repetir N vezes um trecho do algoritmo, verificando sempre **antes** de cada execução se a condição é verdadeira.
- Sintaxe

enquanto <condição> faca

    <lista de comandos>

    <incremento/decremento do controlador>

fimenquanto

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

COMANDO ENQUANTO

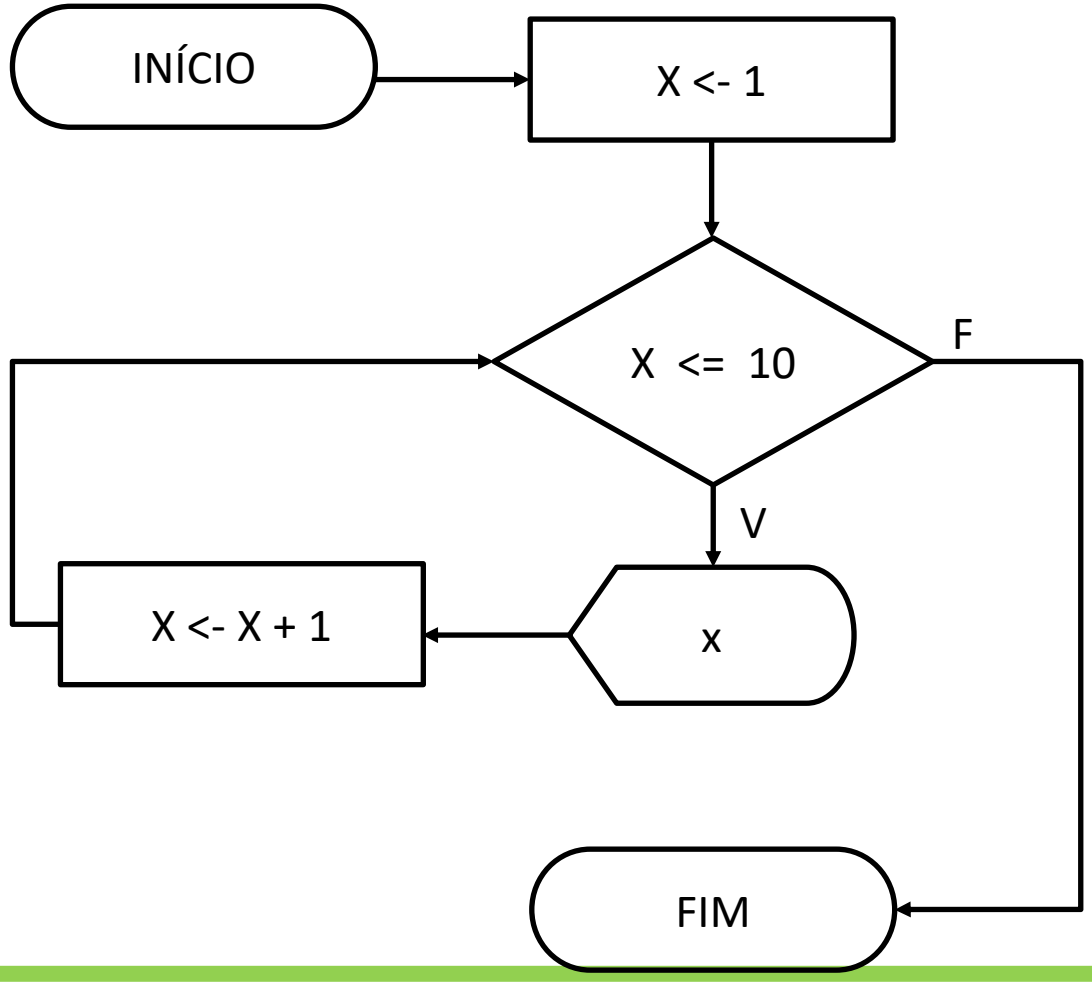
```
algoritmo "laco_enquanto"

var
    x : inteiro
inicio
    x <- 1;
    enquanto (x <= 10) faca
        escreva(x,", ")
        x <- x + 1
    fimenquanto
fimalgoritmo
```

DIAGRAMA DE BLOCOS

- O controle da execução de um comando de repetição é avaliado em um teste condicional, por esta razão utiliza-se o losango para expressarmos este trecho do algoritmo no diagrama de blocos.

DIAGRAMA DE BLOCOS



WHILE

- **Laço while (enquanto):**
  - O laço é executado **enquanto** a <condição> for verdadeira.
  - No Algoritmo:

<variavel> = <valor inicial>

enquanto (<condição parada>)

Bloco de Comandos ou instrução;

<incremento da variável>

fimenquanto;

WHILE

- **Na linguagem C/C++:**

```
<variavel> = <valor inicial>
while (<condição parada>)
{
    Bloco de Comandos ou instruções;
    <incremento da variável>
}
```

WHILE

- **Exemplo:**

```
int x = 1, soma=0;
while (x <= 10)
{
    cout << "Digite o número: ";
    cin >> num;
    fflush(stdin);
    soma = soma + num;
    x++;
}
```

WHILE

- Ao chegar ao comando a condição é avaliada por isso a variável a ser testada deve ser inicializada anteriormente.
- Se for “FALSE”, o bloco de comandos não é executado, e a linha de execução passa para o próximo comando após o delimitador de final de bloco;
- Caso a condição avaliada seja “TRUE”, o bloco de comandos é executado, e ao final deste, a condição é novamente avaliada, seguindo novamente esta sistemática.

WHILE

- A condição pode ser simples ou composta.
- A utilização prática deste comando se dá, pela necessidade de que alguns comandos só se repitam **quando** e **enquanto** uma condição for verdadeira.
- Este comando embute as funcionalidades de um comando condicional, junto com um laço de repetição, podendo a variável de controle ser modificada **somente** nas instruções dentro do laço.

WHILE

- Por já realizar o teste no início, a(s) variável(eis) envolvida(s) na condição devem conter um valor antes do início do comando.
- A variável de controle pode ser um acumulador e não necessariamente um contador, podendo com isso ser adicionado valores diferentes de 1.

WHILE

- Exemplo:

```
int x = 1, soma = 0;
while (soma < 10) {
   cout << "Digite um numero: ";
   cin >> x;
   fflush(stdin);
   soma += x;
   x++;
}
```

- Quantas vezes o laço será executado?

WHILE
CONDIÇÃO DE PARADA

- Um cuidado fundamental que o construtor do algoritmo deve ter é o de certificar-se que a condição torne-se, em algum momento, verdadeira, atendendo assim a condição de parada do laço.

- Evitando que o algoritmo entre em um laço infinito.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

WHILE
CONDIÇÃO DE PARADA

- Exemplos de laços infinitos

```
int main()
{
    int x=1;
    while (x < 3)
    {
        cout << x << endl;
        x--;
    }
    return EXIT_SUCCESS;
}
```

```
int main()
{
    int x=1;
    while (x < 3)
    {
        cout << x << endl;
    }
    return EXIT_SUCCESS;
}
```

COMANDO REPITA CONCEITOS

- Estrutura de repetição com avaliação da condição no final
- Permite que um bloco ou ação seja repetido até que uma determinada condição seja verdadeira
- Sintaxe

repita

<lista de comandos>

<incremento/decremento do controlador>

ate <condição de parada>;

COMANDO REPITA CONCEITOS

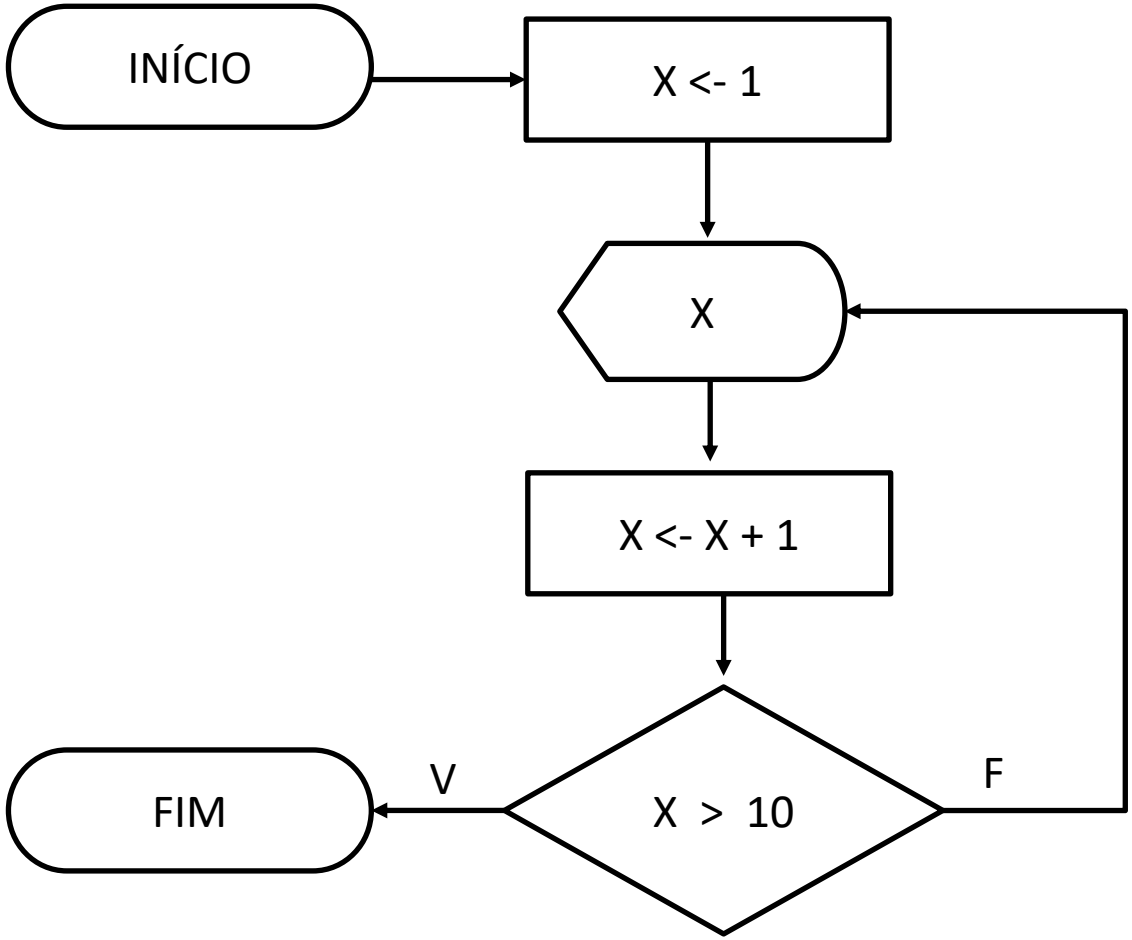
- Analisando a sintaxe do comando observa-se que o bloco de comandos é executado pelo menos uma vez, independente da validade da condição.
- Isso ocorre porque a inspeção da condição é feita após a execução das instruções internas, sendo esta a principal característica deste modelo de repetição.
- Será repetido até que a condição de parada seja verdadeira, quando é interrompido.

COMANDO REPITA EXEMPLO

```
1  algoritmo "repeticao"
2  var
3      x : inteiro
4  inicio
5      x <- 1
6      repita
7          escreva (x, ", ")
8          x <- x + 1
9      ate (x > 10)
10 fimalgoritmo
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

COMANDO REPITA
DIAGRAMA DE BLOCOS



COMANDO REPITA CONTROLADOR

- Muitas vezes durante a construção de algoritmos com este tipo de estrutura não é possível identificar quantas vezes o “laço” será executado, pois depende da ação do usuário.
- A variável controladora deste tipo de estrutura pode ser definida de qualquer tipo de dado e também poderá ser um contador, acumulador ou mesmo uma flag.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

COMANDO REPITA EXEMPLO

- Quantas vezes será executado este “laço”?

```
1  algoritmo "repeticao2"
2  var
3      x, soma : real
4  inicio
5      soma <- 0
6      repita
7          escreva ("Digite um valor qualquer: ")
8          leia (x)
9          soma <- soma + x
10     ate (soma > 50)
11     escreva ("Soma: ", soma:5:2)
12 fimalgoritmo
```

DO WHILE

- **Laço do while (faça enquanto):**
  - É uma variação do laço while, diferenciando pelo teste da condição ser no final.
  - O laço é executado no mínimo uma vez.
  - No Algoritmo:

<variavel> = <valor inicial>

repita

Bloco de Comandos ou instrução;

<incremento da variável>

ate (<condição parada>)

DO WHILE

- **Na linguagem C/C++:**

```
<variavel> = <valor inicial>
do
{
    Bloco de Comandos ou instruções;
    <incremento da variável>
} while (<condição parada>);
```

DO WHILE EXEMPLO

- **Exemplo:**

```
int x = 1, soma=0;
do
{
    cout << "Digite o número: ";
    cin >> num;
    fflush(stdin);
    soma = soma + num;
    x++;
} while (x <= 10);
```

DO WHILE EXEMPLO

```
char sexo;

do
{
     cout << "Informe o sexo M ou F\n";
     cin >> sexo;
     fflush(stdin);
}while( (sexo!='M') && (sexo!='F') );
```

DO WHILE EXEMPLO

```
int opMenu, n1, n2, resultado;
do
{
    system("cls"); //limpa a tela do Prompt de Comando

    cout << "###Menu###\n";
    cout << "0- Sair\n";
    cout << "1- Efetuar a soma de dois números\n";
    cin >> opMenu;
    fflush(stdin);

    if(opMenu==1)
    {
        cout << "Informe o primeiro número: ";
        cin >> n1;
        cout << "Informe o segundo número: ";
        cin >> n2;
        fflush(stdin);

        resultado = n1 + n2;
        cout << "Resultado: " << resultado << endl;

        system("pause"); //Pressione qualquer tecla para continuar. . .
    }
    else if(opMenu!=1 && opMenu!=0)
    {
        cout << "Opção do menu inválida!\n";
        system("pause"); //Pressione qualquer tecla para continuar. . .
    }

}while(opMenu != 0);
```

BREAK

- A instrução break é usada sempre que desejamos interromper a execução de uma ou mais rotinas, geralmente dentro de um laço.

BREAK

- Exemplo:

```
int x = 1, soma = 0;
while (soma < 10) {
    cout << "Digite um número positivo: ";
    cin >> x;
     fflush(stdin);
     if(x < 0)
                    break;
    soma += x;
    x++;
}
```

CONTINUE

- A instrução continue é usada para passar para a próxima iteração, abandonando completamente a execução do código dentro da execução atual do laço;

CONTINUE

- Exemplo:

```
num = 0;
while(num <= 20)
{
  if((num == 7) or (num == 13))
  {
    num++;
    cout << "\n";
    continue;
  }
  cout << num << " - ";
  num++;
}
```

TESTE DE MESA

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x;
7      x = 1;
8      while (x < 3)
9      {
10         cout << x << endl;
11         x++;
12     }
13     return EXIT_SUCCESS;
14 }
15
```

Linha	x	(x < 3)
6	?	?
7	1	?
8		TRUE
10	{1}	
11	2	
8		TRUE
10	{2}	
11	3	
8		FALSE

REFERÊNCIAS

- FORBELLONE, André Luiz Villar. Lógica de programação: a construção de algoritmos e estruturas de dados. 3 ed. São Paulo: Prentice Hall, 2005.
- VILARIN, Gilvan. Algoritmos Programação para Iniciantes. Editora Ciência Moderna. Rio de Janeiro, 2004.
- MORAES, Paulo Sérgio. Curso Básico de Lógica de Programação. Centro de Computação – Unicamp, 2000.
- STEINMETZ, Ernesto H. R.; FONTES, Roberto Duarte Cartilha Lógica de Programação. Editora IFB, Brasília - DF, 2013.