Estruturas de repetição:

- while
- for
- do while

As estruturas de repetição (ciclos ou laços)

- Estruturas de repetição são usadas quando uma ou mais instruções devem ser repetidas enquanto uma certa condição estiver sendo satisfeita.

- Em C/C++ existem três estruturas de repetição:
  - **while**
  - **for**
  - **do while**

A estrutura de repetição while (enquanto)

- Uma estrutura de repetição permite ao programador especificar que uma ação deve ser repetida enquanto alguma condição for verdadeira.

- Exemplo (em pseudocódigo):

Enquanto existirem mais itens em minha lista de compras
Comprar próximo item e excluí-lo da minha lista

descreve a repetição que acontece durante uma saída para compras.

- A condição "existirem mais itens em minha lista de compras" pode ser verdadeira ou falsa.
- Se ela for verdadeira, então a ação, "Comprar próximo item e excluí-lo da minha lista" é executada.
- Esta ação será repetidamente executada, enquanto a condição for verdadeira (true).

A estrutura de repetição while

- O(s) comando(s) contidos na estrutura de repetição while constituem o **corpo** do while.
- O corpo da estrutura while pode ser um comando único ou um comando composto.
- Em algum momento, a condição se tornará falsa (false) (no exemplo: quando o último item da lista de compras foi comprado e excluído da mesma) e então a repetição termina.

Erro comum de programação

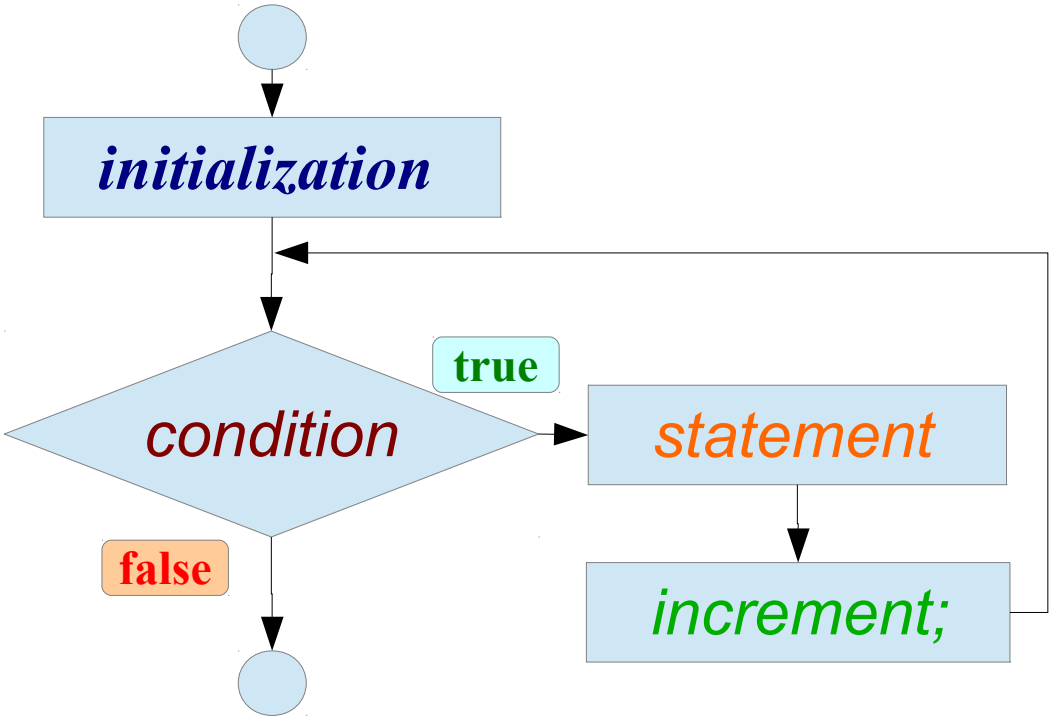
Não fornecer no corpo de uma estrutura while uma ação que faça com que a condição se torne falsa (false).

Em algum momento resultará em um erro chamado “ciclo infinito”, no qual a estrutura de repetição nunca termina de ser executada.

A estrutura de repetição while

- O formato geral da estrutura while:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```



onde

initialization inicializa a variável de controle do ciclo

condition é a condição de continuação do ciclo

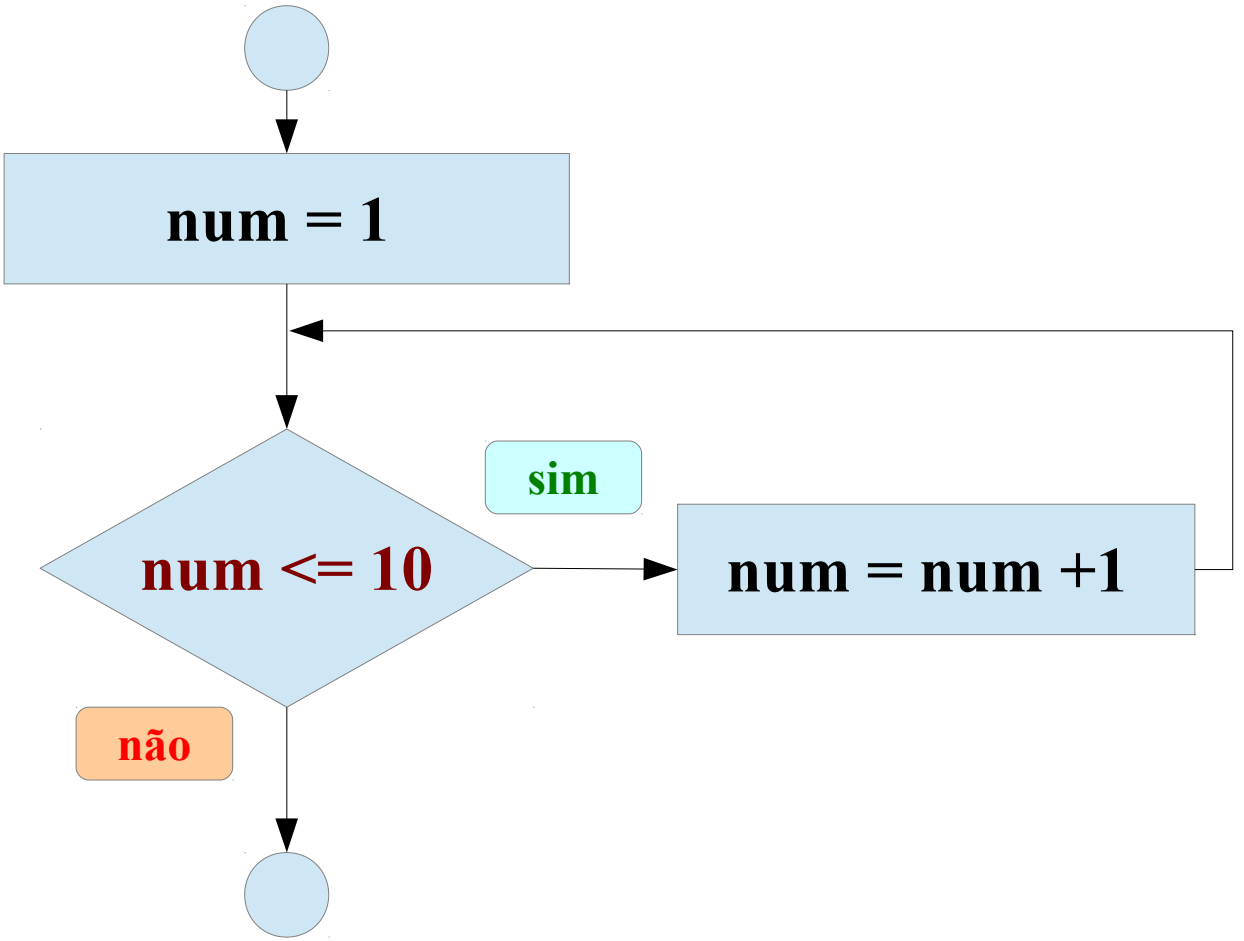
increment incrementa a variável de controle.

A estrutura de repetição while

Exemplo 1: Imprimir os números de 0 a 10.

A variável **int** num sera inicializada com valor 0.

Quando a estrutura de repetição while a seguir terminar de ser executada, num conterá a resposta desejada:



Exemplo 1: Imprimir os números de 0 a 10.

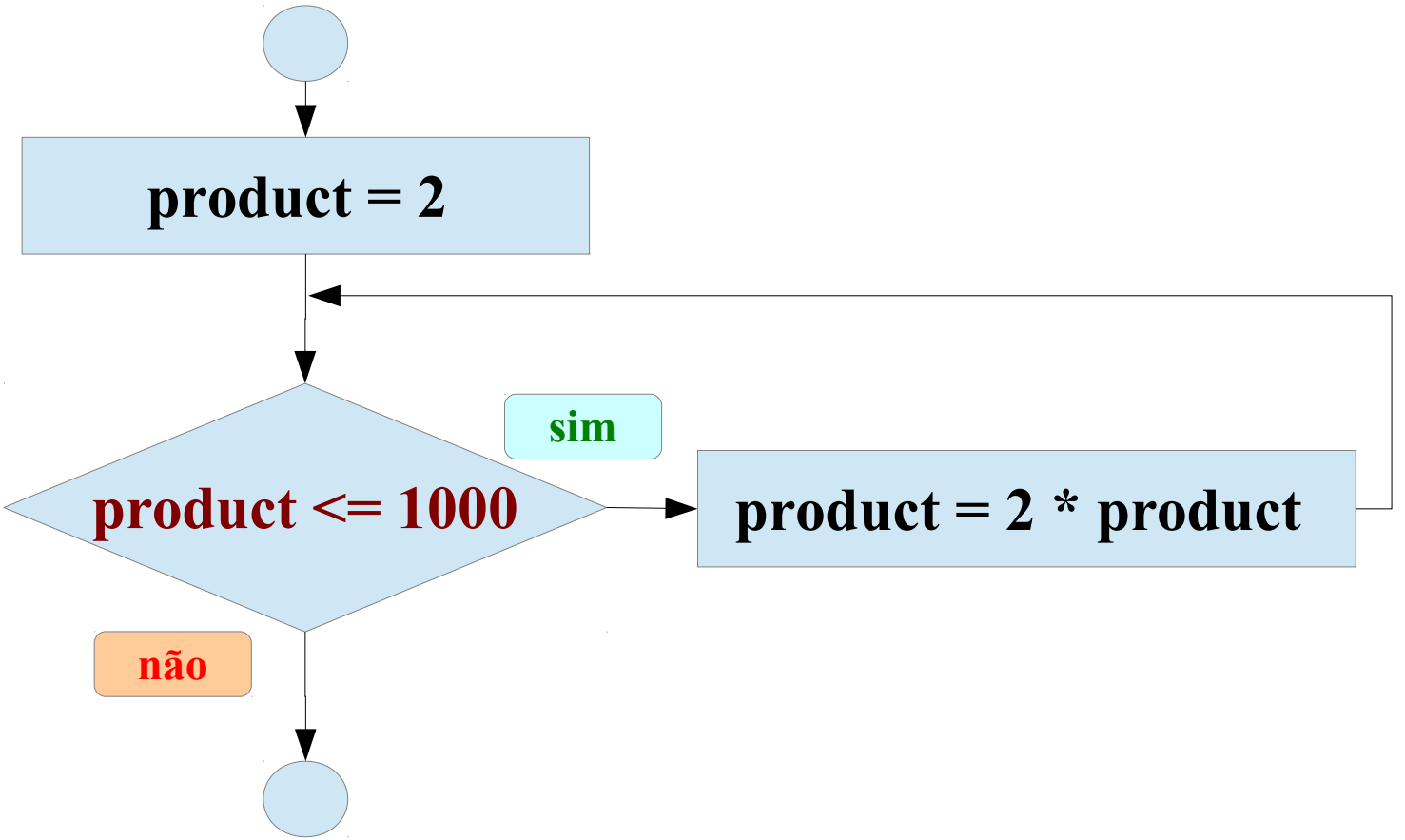
```c
// Imprime números de 0 a 10
#include <stdio.h> // permite ao programa realizar entrada e saída

int main() // a função main inicia a execução do programa
{
    int num = 0;

    printf("\n Numeros de 0 a 10  \n");
    printf(" ( usando WHILE )  \n");

    while ( num <= 10 )
    {
      printf("\n num = %i ", num);
      num = num + 1;
    }

   return 0; // indica que o programa terminou com sucesso
} // fim da função main
```

A estrutura de repetição while

Exemplo 2: Achar a primeira potência de 2 maior do que 1000.

A variável **int** product sera inicializada com valor 2.

Quando a estrutura de repetição while a seguir terminar de ser executada, product conterá a resposta desejada:



Exemplo 2: Achar a primeira potência de 2 maior do que 1000.

```c
// Achar a primeira potência de 2 maior do que 1000
#include <stdio.h> // permite ao programa realizar entrada e saída

int main() // a função main inicia a execução do programa
{
    int product = 2;

    printf("\n A primeira potência de 2 maior do que 1000 \n");
    printf(" ( usando WHILE )  \n");

    while ( product <= 1000 )
        product = 2 * product;

    printf("\n Product = %i \n", product);

   return 0; // indica que o programa terminou com sucesso
} // fim da função main
```

Formulando algoritmos: repetição controlada por contador

Exemplo 3:

- Considere o seguinte problema:
  - Uma turma de dez estudantes resolve um teste.
  - As notas são números inteiros no intervalo de 0 a 100.
  - Determine a média das notas da turma.
  - A média é igual à soma das notas dividida pelo número de estudantes.
- O algoritmo para resolver este problema deve receber como entrada cada uma das notas, executar o cálculo da média e imprimir o resultado.
- Usamos uma repetição controlada por contador para fornecer como entrada as notas, uma de cada vez.
- Esta técnica usa uma variável chamada de contador, para controlar o número de vezes que um conjunto de comandos será executado.

Formulando algoritmos: repetição controlada por contador

- Neste exemplo, a repetição termina quando o contador exceder o número de repetições predefinido, no caso 10.
- A repetição controlada por contador é chamada de repetição **definida**, porque o número de repetições é conhecido antes de o ciclo começar a ser executado.
- No nosso exemplo:
  - total é uma variável usada para acumular a soma de uma série de valores.
  - contador é uma variável usada para contar - neste caso, contar o número de notas lidas.
- As variáveis que são usadas para armazenar algum tipo de dados devem ser normalmente **inicializadas** com zero antes de serem usadas em um programa.
- Caso contrário, a soma incluirá o valor armazenado anteriormente na posição de memória do total

Formulando algoritmos: repetição controlada por contador

- As ações que devem ser executadas e a ordem em que estas ações devem ser executadas para nosso exemplo (em pseudocódigo):

Inicialize total com zero
Inicialize contador de notas com um

Enquanto o contador de notas for menor do que ou igual a dez
    Receba como entrada a próxima nota
    Some a nota ao total
    Some um ao contador de notas

Atribua à média da turma ao total dividido por dez

Imprima a média da turma

Exemplo 3:

```
 4    int main ()
 5    {
 6      int gradeCounter; // número de notas digitadas
 7      float total;      // soma das notas
 8      float grade;      // uma nota
 9      float average;    // média das notas
10
11      // Inicialização
12      total = 0;
13      gradeCounter = 1;
14
15      // fase de processamento
16      while ( gradeCounter <= 10 )
17      {
18       printf ("\n Digite a %i nota: ", gradeCounter);
19       scanf("%f", &grade);
20
21       total = total + grade;
22       gradeCounter = gradeCounter + 1;
23      }
24      // fase de término
25      average = total / 10;
26      printf("\n A nota media da turma = %.2f \n", average);
27
28      return 0;
29    }
```

Formulando algoritmos: repetição controlada por contador

Erro comum de programação

- Não atribuir os valores iniciais corretos.
- Este é um exemplo de erro de lógica.
- No exemplo, se as variáveis **contador** e/ou **total** não forem inicializados, os resultados do programa serão incorretos.

Formulando algoritmos: repetição controlada por sinalizador

Exemplo 4:

Considere o seguinte problema:

- Desenvolva um programa que calcule a média da turma e que processe um número arbitrário de notas cada vez que o programa é executado.

- No primeiro exemplo o número de notas (10) era conhecido com antecedência.
- Neste exemplo, nenhuma indicação é dada de quantas notas serão digitadas.
- O programa deve processar um número **arbitrário** de notas.

Formulando algoritmos: repetição controlada por sinalizador

- Um modo de resolver este problema é usar um valor especial, chamado de sinalizador.
- Um sinalizador é um valor artificial para indicar o fim de entrada de dados.

- O usuário vai digitar todas as notas que deseja e então ele vai digitar o valor de sinalizador para indicar o fim de entrada de dados.

Formulando algoritmos: repetição controlada por sinalizador

- A repetição controlada por sinalizador é frequentemente chamada de repetição **indefinida**, porque o número de repetições não é conhecido antes do ciclo começar a ser executado.
- Naturalmente, o valor de **sinalizador** deve ser escolhido de forma que não possa ser confundido com um valor aceitável fornecido como entrada.
- Como as notas de um teste normalmente são números não-negativos, **-1** é um valor de sinalizador aceitável para este problema.

Formulando algoritmos: repetição controlada por sinalizador

Erro comum de programação

- Escolher um valor de sinalizador que é também um valor de dados válido – é um erro de lógica.

Observação de engenharia de software

- Muitos programas podem ser logicamente divididos em três fases:
  - uma fase de inicialização, que inicializa as variáveis do programa;
  - uma fase de processamento, que recebe como entrada valores de dados e ajusta as variáveis do programa de acordo;
  - e uma fase de finalização, que calcula e imprime os resultados finais.

Formulando algoritmos: repetição controlada por sinalizador

- Algoritmo em pseudocódigo que usa repetição controlada por sinalizador para resolver o problema da média da turma:

Inicializar total com zero
Inicializar contador com zero
Receba como entrada a primeira nota (possivelmente o sinalizador)
Enquanto o usuário ainda não digitou o sinalizador
    Some esta nota ao total corrente
    Some um ao contador de notas
    Receba como entrada a próxima nota (possivelmente o sinalizador)
Se o contador não for igual a zero
    Inicialize a média com o total dividido pelo contador
    Imprima a média
Senão
    Imprima "Nenhuma nota foi fornecida"

```
 7    int gradeCounter; // número de notas digitadas
 8    float total;      // soma das notas
 9    float grade;      // uma nota
10    float average;    // média das notas
11
12   // fase de inicialização
13    total = 0;
14    gradeCounter = 0;
15
16   // fase de processamento
17    printf ( "\n Digite a nota ou (-1) para finalizar: ");
18    scanf("%f" , &grade);
19
20    while ( grade != -1 )
21    {
22      total = total + grade;
23      gradeCounter = gradeCounter + 1;
24      printf ( "\n Digite a nota ou (-1) para finalizar: ");
25      scanf(" %f" , &grade);
26    }
27
28   // fase de término
29    if ( gradeCounter != 0 )
30    {
31      average = total / gradeCounter;
32      printf("\n A media da turma =  %.2f \n", average);
33    }
34    else
35      printf("\n Nenhuma nota foi fornecida! \n");
```

Operadores de atribuição

Dica de desempenho

- Os programadores podem escrever programas um pouco mais rápidos e os compiladores podem compilar programas um pouco mais rapidamente quando forem usados os operadores de atribuição “abreviados”.

Operadores de atribuição

- C/C++ oferece vários operadores de atribuição para abreviar as expressões de atribuição.
- Por exemplo, o comando

  c = c + 3;

pode ser abreviado com o operador "atribuição com adição" **+=** como

  c += 3;

- O operador **+** soma o valor da expressão à direita do operador ao valor da variável à esquerda do operador e armazena o resultado na variável à esquerda do operador.

- Qualquer comando da forma

variável = variável operador expressão;

onde operador é um dos operadores binários +, -, *, /, ou %, pode ser escrito na forma

variável operador = expressão;

Operadores de incremento e decremento

- C/C++ também fornece
  - o operador unário de incremento ++
  - e o operador unário de decremento --

- Se um operador de incremento ou decremento é colocado antes de uma variável, é chamado de operador de pré-incremento ou pré-decremento, respectivamente.

  ++a

- Se um operador de incremento ou decremento é colocado depois de uma variável, é chamado de operador de pós-incremento ou de pós-decremento, respectivamente.

  a++

Operadores de incremento e decremento

- Pré-incrementar (pré-decrementar) uma variável faz com que a variável seja incrementada (decrementada) por 1, sendo o novo valor da variável usado na expressão em que ela aparece.

- Pós-incrementar (pós-decrementar) uma variável faz com que o valor atual da variável seja primeiro usado na expressão em que ela aparece, sendo então, após, o valor da variável incrementado (decrementado) por 1.

Operadores de incremento e decremento

Exemplo 5:

- O programa a seguir demonstra a diferença entre a versão de pré-incremento e a versão de pós-incremento do operador ++.
- Pós-incrementar a variável c faz com que a mesma seja incrementada depois de ser usada no comando de impressão.
- Pré-incrementar a variável c faz com que a mesma seja incrementada antes de ser usada no comando de impressão.

- O programa exibe o valor de c antes e depois que o operador ++ é usado.
- O operador de decremento (--) funciona de maneira similar.

Exemplo 5: Operadores de incremento e decremento

```c
#include <stdio.h>

int main()
{
 int c;

 c=5;
 printf (" Pos-incremento:\n");
 printf (" c = %i \n", c);       // imprime 5
 printf (" c++ = %i \n", c++); // imprime 5 e então pós-incrementa
 printf (" c = % i", c );        // imprime 6

 c=5;
 printf ("\n\n Pre-incremento: \n");
 printf (" c = %i \n", c);       // imprime 5
 printf (" ++c = %i \n", ++c ); // pré-incrementa e então imprime 6
 printf (" c = %i \n", c );      // imprime 6

 return 0; // término normal
}
```

Operadores de incremento e decremento

- Note que, quando incrementarmos ou decrementarmos uma variável sozinha em um comando, as formas de pré-incremento e pós-incremento têm o mesmo efeito (o mesmo vale para pré-decremento e pós-decremento).

- Só quando uma variável aparece no contexto de uma expressão maior é que pré-incrementá-la e pós-incrementá-la têm efeitos diferentes (e, semelhantemente, para pré-decrementar e pós-decrementar).

Erro comum de programação

- Tentar usar o operador de incremento ou decremento em uma expressão que não seja um simples nome de variável, por exemplo, escrever ++ (x + 1), é um erro de sintaxe.

Exemplo 6: Repetição controlada por contador (com incremento)

```
 4   int main()
 5   {
 6    int counter;
 7
 8    //=====================================================
 9    counter = 1;
10    printf("\n WHILE com incremento no corpo:");
11    while ( counter <= 10 )
12     {
13       printf(" \n %i ", counter);
14       ++counter;
15     }
16
17    //=====================================================
18    counter = 0;
19    printf("\n\n WHILE com incremento na condição: ");
20    while ( ++counter <= 10 )
21       printf(" \n %i ", counter);
22
23   return 0;
24   }
```

Operadores de incremento e decremento

- Podemos escrever:

```
while ( ++counter <= 10 )
   printf(" \n %i ", counter);
```

- Este código economiza um comando, porque o incremento é feito diretamente na condição do while (antes da condição ser testada).
- Além disso, este código elimina as chaves em torno do corpo do while porque o while agora contém só um comando.

Erro comum de programação

- Como os valores em ponto flutuante podem ser aproximados, controlar ciclos com variáveis de ponto flutuante pode resultar em valores imprecisos do contador e testes de término inexatos.

Exemplos (execícios)

Utilizando estrutura **while**

Exemplo (execício) 7:

Achar a soma dos números inteiros impares no intervalo de 0 a 100

Exemplo (execício) 8:

Calcular a média de números negativos fornecidos pelo usuário.

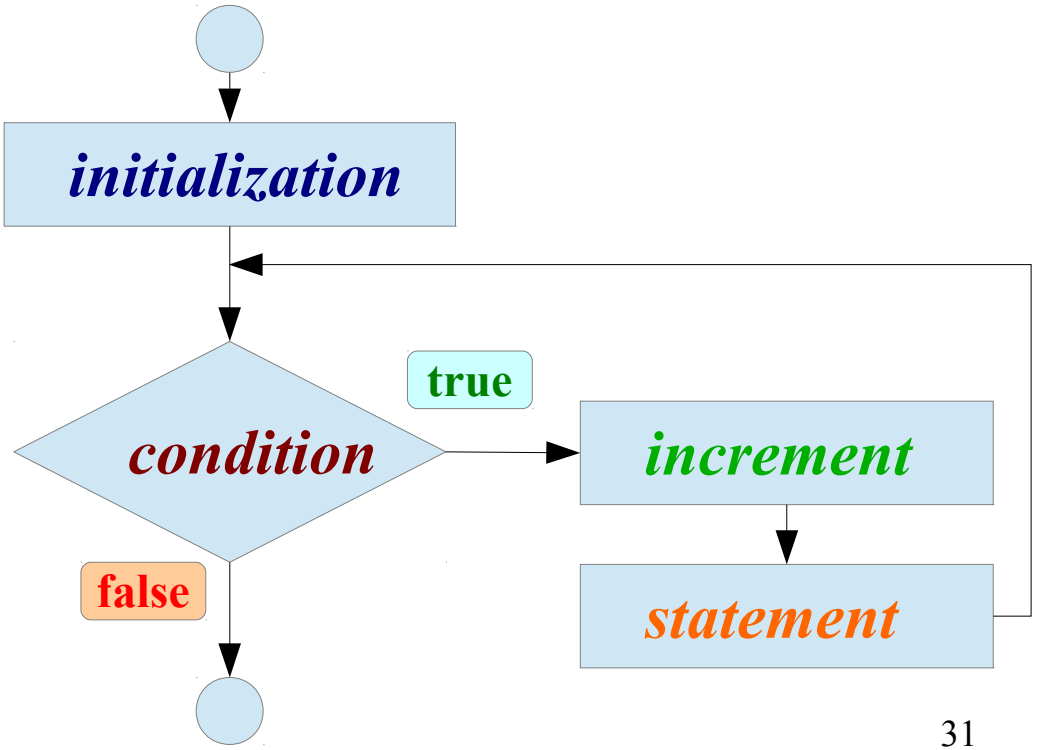
A quantidade de números a serem processados é aleatória.

A estrutura de repetição for

- A estrutura de repetição for trata de todos os detalhes da repetição controlada por contador.
- O formato geral da estrutura for:

```
for ( initialization; condition; increment)
{
 statement;
}
```

- initialization inicializa a variável de controle do ciclo
- *condition* é a condição de continuação do ciclo
- *increment* incrementa a variável de controle.



A estrutura de repetição for

- Na maioria dos casos, a estrutura for pode ser representada com uma estrutura while equivalente:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

A estrutura de repetição for

- As três expressões na estrutura **for** são opcionais.
- Se a expressão condition é omitida, a condição de continuação do ciclo é sempre verdadeira, criando deste modo um ciclo infinito.
- A parte de inicialization pode ser omitida se a variável de controle for inicializada em outro lugar do programa.
- A expressão increment pode ser omitida se o incremento for calculado por comandos no corpo da estrutura for, ou se nenhum incremento for necessário.

A estrutura de repetição for

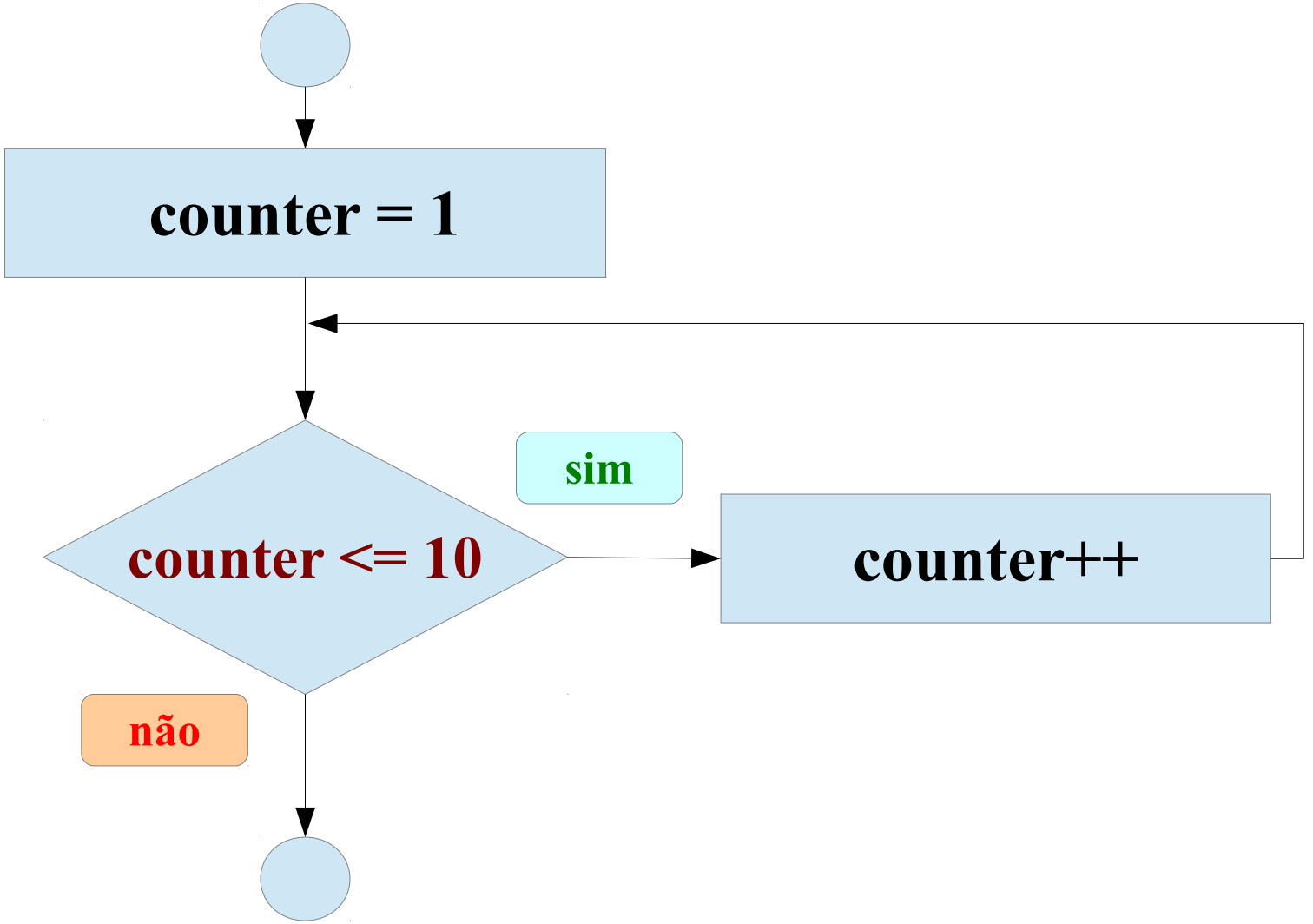
- A expressão de incremento na estrutura for atua como um comando isolado no fim do corpo do for.
- Portanto, as expressões
  - counter = counter + 1
  - counter += 1
  - ++counter
  - counter++

são todas equivalentes à parte de incremento da estrutura for.

Os dois ponto-e-vírgulas na estrutura **for** são obrigatórios.

- O "incremento" de uma estrutura for pode ser negativo (sendo, de fato, um decremento e o ciclo, na verdade, conta para trás).

O fluxograma da estrutura for
(é semelhante ao da estrutura while)



Exemplo 9: Repetição controlada por contador com a estrutura for

```c
// Repetição controlada por contador com a estrutura for
#include <stdio.h>

 int main ()
{
 int counter;
// inicialização, condição de repetição e incremento
// estão todas incluidas no cabeçalho da estrutura for.
 printf("\n Estrutura FOR:");
 for ( counter = 1; counter <= 10; counter++ )
   printf(" \n %i ", counter);

 return 0;
}
```

Exemplos de uso da estrutura for

- Variável de controle varia de 1 até 100 com incrementos de 1:

```
for (  i = 1; i <= 100; i++ )
```

- Variável de controle varia de 100 até 1 com incrementos de -1 (decrementos de 1):

```
for (  i = 100; i > 1; i--)
```

- Variável de controle varia de 7 até 77 com incrementos de 7:

```
for (  i=7; i<=77; i += 7 )
```

Exemplos (execícios)

Utilizando estrutura **for**

Exemplo (execício) 10:

Mostrar números múltiplos de 3 no intervalo de 0 a 30

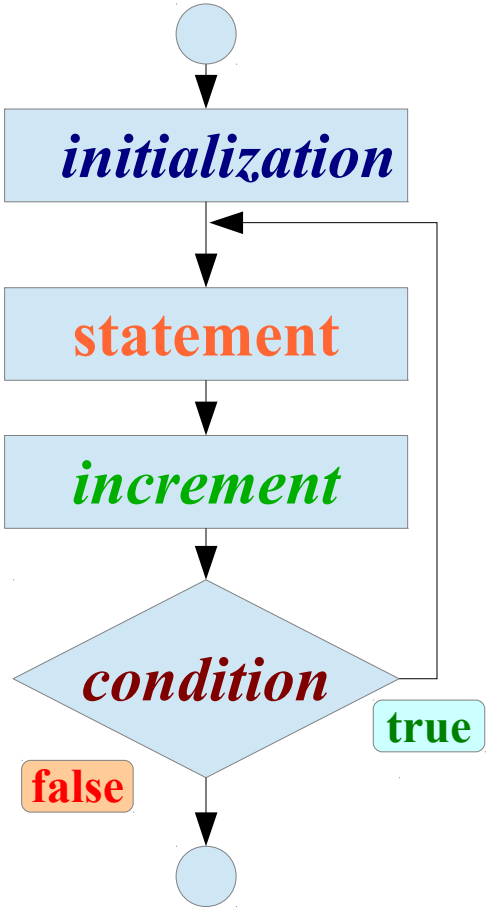
Exemplo (execício) 11:

Achar a soma dos números múltiplos de 5 no intervalo de -10 a 20

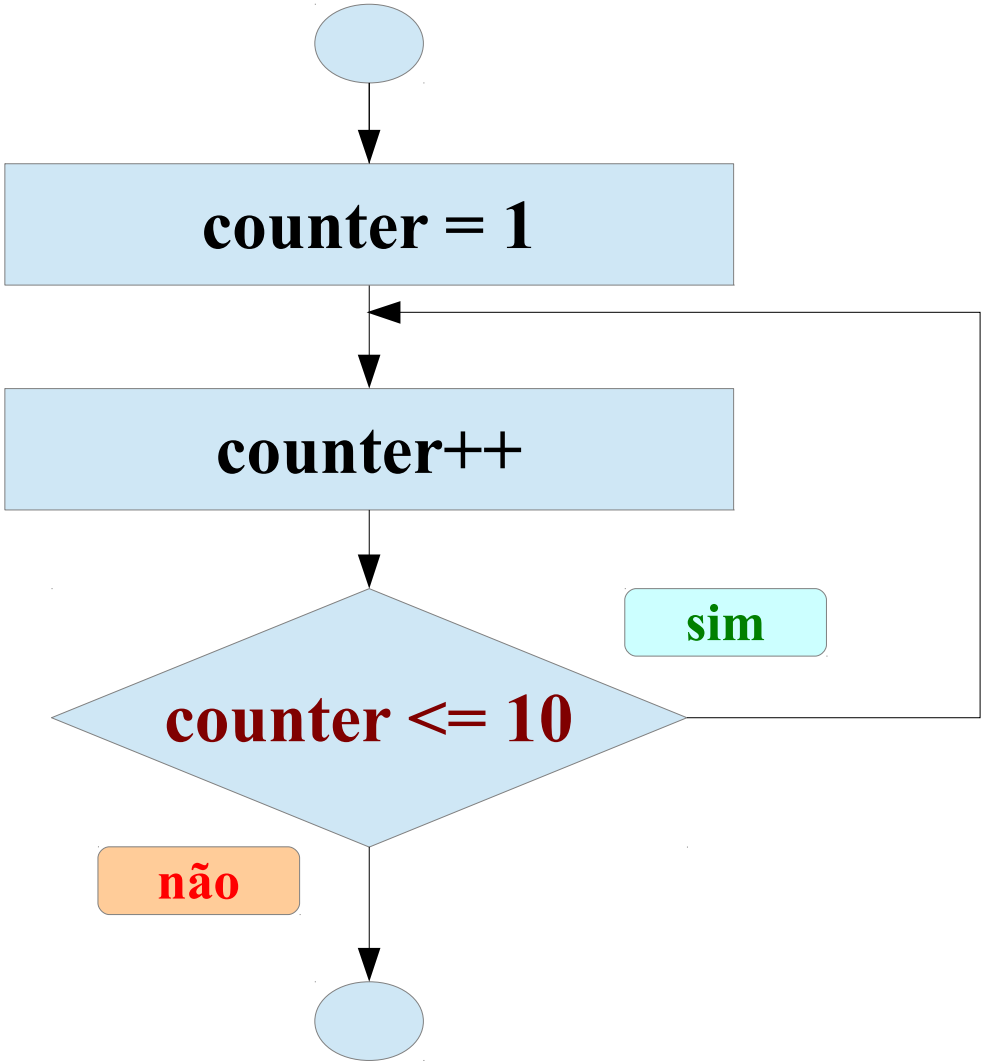
A estrutura de repetição do/while

- A estrutura de repetição do/while é semelhante à estrutura while.
- Na estrutura while, a condição de continuação do ciclo é testada no princípio do ciclo, antes do corpo do ciclo ser executado.
- A estrutura do/while testa a condição de continuação do ciclo depois do corpo do ciclo ser executado.
- Assim, o corpo do ciclo será executado pelo menos uma vez.
- Quando um do/while termina, a execução continua com o comando depois da cláusula while.
- formato:

```
initialization;
do
 {
    statement;
    increment;
 } while (condition);
```



Fluxograma da estrutura do/while



- Este fluxograma torna claro que a condição de continuação do ciclo é executada somente depois dos comandos do ciclo serem executados pelo menos uma vez.

Exemplo 12: A estrutura de repetição do/while

- O programa a seguir usa uma estrutura de repetição do/while para imprimir os números de 1 até 10.
- A variável de controle counter poderia ser pré-incrementada no teste de continuação do ciclo.

```
 2  #include <stdio.h>
 3
 4   int main ()
 5  {
 6   int counter;
 7
 8   counter = 1;
 9   printf("\n Estrutura DO WHILE: ");
10   do {
11     printf(" \n %i ", counter);
12     counter++;
13   } while ( counter <= 10 );
14
15   return 0;
16  }
```

Exemplos (execícios)

Utilizando estrutura **do/while**

Exemplo (execício) 13:

Calcular a média de 5 números fornecidos pelo usuário.

Os comandos break e continue

- Os comandos break e continue alteram o fluxo de controle.

- O comando break, quando executado em uma estrutura while, for, do/while ou switch, provoca a saída imediata da estrutura.

- A execução do programa continua com o primeiro comando depois da estrutura.

- Os usos mais comuns do comando break são sair antecipadamente de um ciclo ou pular o restante de uma estrutura switch.

Os comandos break e continue

- O comando continue, quando executado em uma estrutura while, for ou do/while pula os comandos restantes no corpo dessa estrutura e prossegue com a próxima repetição do ciclo.

- Em estruturas while e do/while o teste de continuação do ciclo é feito logo depois do comando continue ser executado.

- Na estrutura for, a expressão de incremento é executada e então é feito o teste de continuação do ciclo.

Exemplos (execícios):

Obs.: utilize uma das estruturas de repetição estudadas – **for, while** ou **do/while.**

14. Mostrar os números inteiros pares e múltiplos de 3 no intervalo de 1 até N e calcular a soma deles.

N – é um numero inteiro positivo fornecido pelo usuário.

15. O programa recebe as notas de uma turma de 15 alunos.

- As notas são números no intervalo de 0 a 10.
- Considera-se aprovado quem tiver a nota igual ou superior a 6.
- Calcular a nota média dos alunos aprovados.
- Calcular a quantidade dos estudantes reprovados.
- Incluir o controle de dados de entrada: deverão ser aceitos somente os números no intervalo de 0 ate 10, qualquer número fora desse intervalo deve ser desconsiderado.