

Aula 04:

Estruturas de Repetição

Prof. Edvard

edvard@unifei.edu.br

Universidade Federal de Itajubá

Atribuição Simplificada

- Muitas vezes, durante a programação, é necessário realizar operações em uma única variável e atribuir o novo valor a ela mesmo.

Por exemplo:

```
// Operadores de atribuição utilizados em conjunto  
// com operadores aritméticos, atualizando o valor  
// de uma única variável  
a = a + 10;  
b = b * 4;  
c = c - 1;  
d = d / 10;
```

- O C oferece uma série de **operadores de atribuição simplificada**, que vem exatamente para tornar as tarefas acima mais simples e com código mais elegante.

Atribuição Simplificada

- Quando formos realizar uma operação em uma única variável, podemos utilizar os seguintes **operadores de atribuição simplificada**:

```
// Operadores de atribuição simplificada  
// Realizam a operação aritmetica e fazem a atribuição  
// em um único passo de escrita
```

```
a += 10; // a = a + 10;  
b *= 4;  // b = b * 4;  
c -= 1;  // c = c - 1;  
d /= 10; // d = d / 10;  
e %= 2;  // e = e % 2;
```

- É possível utilizá-los sempre que formos **adicionar**, **subtrair**, **multiplicar**, **dividir** ou **obter o resto** de um valor, e atribuir o valor obtido à própria variável.

Incremento e Decremento

- Existe ainda uma outra forma de atribuição simplificada, especial para casos onde é necessário se somar um (**incrementar**, ++) ou subtrair uma unidade (**decrementar**, --) de um valor.

- Se quisermos somar 1 sobre uma variável, temos, portanto, **três opções**.

```
a = a + 1; // atribuição simples
a += 1;    // atribuição simplificada
a++;       // incremento
```

- Se quisermos subtrair 1 de uma variável, temos, também, **três opções**.

```
a = a - 1; // atribuição simples
a -= 1;    // atribuição simplificada
a--;       // decremento
```

Estruturas de Repetição

- Você já reparou como uma boa parte das ações que realizamos no dia a dia são **repetitivas**?
 - Quando **giramos a tampa de uma garrafa** de água, por exemplo, repetimos a mesma ação “girar a tampa”, até que ela saia por inteiro.
 - Quando alguém nos diz que para chegar até algum lugar, precisamos **andar 5 quarteirões** em uma determinada direção, o que fazemos realmente é a mesma coisa por cinco vezes: andamos, chegamos à esquina, olhamos para atravessar: se não vier carro, atravessamos e fazemos tudo de novo.

Estruturas de Repetição

- Em **programação**, é a mesma coisa. Em muitas situações, teremos que executar a **mesma tarefa várias vezes**, mudando, por exemplo, apenas o valor de uma ou mais variáveis.

— Considere o exercício proposto:

(35 pontos) Escreva um programa em C que calcule a distância que é percorrida em cada raia de uma pista de atletismo para completar uma volta. Uma pista de atletismo circular é formada por uma quantidade de raias concêntricas (mesmo centro), consecutivas e com a mesma largura. A partir do valor do raio que marca o centro da primeira raia, definido pelo usuário, é possível calcular o valor do raio que marca o centro da segunda raia adicionando o valor da largura, e assim sucessivamente. Escreva as respostas com cinco dígitos após a vírgula (ponto).

- A pista possui cinco raias de 1m de largura.
- A fórmula para cálculo do perímetro de uma circunferência é $P=2*\pi*r$ e o valor de π é 3.14159265.

Estruturas de Repetição

- Uma solução possível é a seguinte:

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    float raio;
```

```
    float d1, d2, d3, d4, d5;
```

```
    float pi = 3.14159265;
```

```
    printf("Entre com o raio ate o centro da primeira raia: ");
```

```
    scanf("%f", &raio);
```

```
    d1 = 2*pi*raio;
```

```
    d2 = 2*pi*(raio + 1);
```

```
    d3 = 2*pi*(raio + 2);
```

```
    d4 = 2*pi*(raio + 3);
```

```
    d5 = 2*pi*(raio + 4);
```

```
    printf("Distancia raia 1: %.5f\n", d1);
```

```
    printf("Distancia raia 2: %.5f\n", d2);
```

```
    printf("Distancia raia 3: %.5f\n", d3);
```

```
    printf("Distancia raia 4: %.5f\n", d4);
```

```
    printf("Distancia raia 5: %.5f\n", d5);
```

```
    return 0;
```

```
}
```

Quanta repetição!

Veja como fazemos sempre a mesma coisa. A única coisa que muda é o incremento do raio (1, 2, 3, etc.).

A mesma coisa, por aqui.

Podemos mudar esse panorama.

Estruturas de Repetição

- Repare na solução utilizando estruturas de repetição:

```
#include <stdio.h>

int main()
{
    float raio, d; // somente declaro uma distância, d
    float pi = 3.14159265;
    int i;

    printf("Entre com o raio ate o centro da primeira raia: ");
    scanf("%f", &raio);

    // Fazemos a mesma coisa cinco vezes, com i variando de 0 até 4, de um em um.
    for(i = 0; i < 5; i++)
    {
        d = 2*pi*(raio + i);
        printf("Distancia raia %d: %.5f\n", (i+1), d);
    }

    return 0;
}
```

Repare como a solução é mais **concisa**, mas não perde em **clareza**.

Repetimos uma operação semelhante por cinco vezes, utilizando um “**laço**” ou “**loop**” de **repetição**.

Estruturas de Repetição

- **Loop** pode ser traduzido livremente como “laço”, ou “laçada”.
- Segundo o dicionário: “É um trecho de programa executado repetidamente,
 - Um **número definido de vezes**, ou;
 - Até que uma **condição seja satisfeita**”.
- O laço deve ser sempre interrompido por alguma interferência, de maneira que não execute indefinidamente.
- Um laço que não tenha uma **condição de parada bem definida** irá gerar um erro em seu programa.
 - Esta é uma das causas mais comuns para o “**travamento**” de aplicativos. Chamamos de “**loop infinito**”.

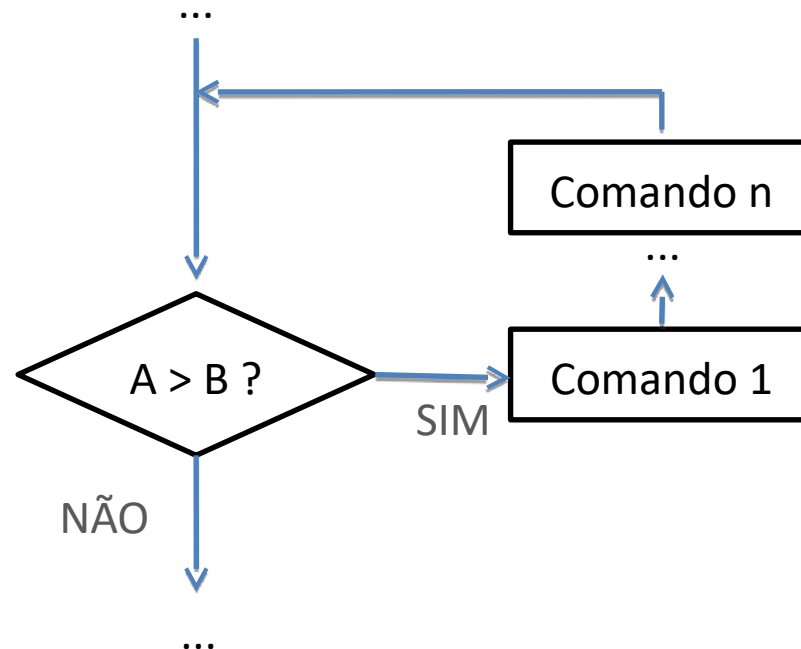
Repetição por Condição

- Em alguns casos, desejamos que um determinado bloco de comandos seja executado **enquanto** uma condição for verdadeira.
- Este comando terá a seguinte configuração básica:

Enquanto condição **faça**
 sequência de comandos
Fim enquanto

Em C, sua forma geral é:

```
while (condição)  
{  
    // Faça alguma coisa  
}
```



Repetição por Condição

- Imagine o seguinte exemplo:

Escreva um programa que imprima na tela todos os números pares menores ou iguais a 100.

```
#include <stdio.h>

int main()
{
    int num = 0;

    while(num < 100)
    {
        num += 2; // Calculo proximo número par
        printf("%d ", num); // Imprimo o número par
    }

    printf("\n\n"); // pulo linhas

    return 0;
}
```

Irá executar **enquanto** $\text{num} < 100$.
num = 0, 2, 4, 6, ..., 94, 96, 98.
Quando $\text{num} == 100$, ele para o laço.

Hands-On!

- Escreva um programa em C que receba dois números inteiros do usuário. Utilize *printf* e *scanf*. Os números, **a** e **b**, devem ser inseridos em ordem crescente.
- A seguir, **imprima todos os números inteiros entre a e b**, incluindo eles mesmos.
 - Dica, crie uma variável auxiliar, que irá guardar cada número que você deseja imprimir e terá seu valor mudado a cada laço do *while*.

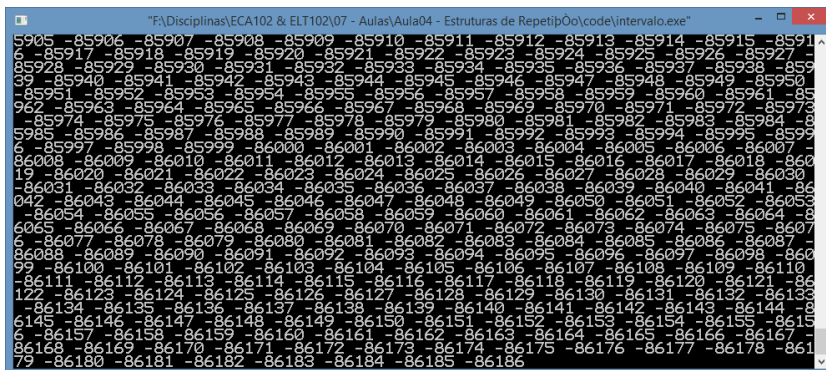
Hands-On!

- Como conseguiríamos, por exemplo, criar uma condição que nunca seria insatisfeita, gerando um **loop infinito**?

```
printf("Entre com dois valores inteiros, em ordem crescente: ");  
scanf("%d %d", &a, &b);
```

```
printf("Os valores inteiros no intervalo sao: \n");  
atual = a;
```

```
while(atual <= b) // se número atual for menor ou igual a b, continue o laço  
{  
    printf("%d ", atual); // imprimo o número atual  
    atual--; // incremento o número  
}
```



```
85905 85906 85907 85908 85909 85910 85911 85912 85913 85914 85915 85916 85917 85918 85919 85920 85921 85922 85923 85924 85925 85926 85927 85928 85929 85930 85931 85932 85933 85934 85935 85936 85937 85938 85939 85940 85941 85942 85943 85944 85945 85946 85947 85948 85949 85950 85951 85952 85953 85954 85955 85956 85957 85958 85959 85960 85961 85962 85963 85964 85965 85966 85967 85968 85969 85970 85971 85972 85973 85974 85975 85976 85977 85978 85979 85980 85981 85982 85983 85984 85985 85986 85987 85988 85989 85990 85991 85992 85993 85994 85995 85996 85997 85998 85999 86000 86001 86002 86003 86004 86005 86006 86007 86008 86009 86010 86011 86012 86013 86014 86015 86016 86017 86018 86019 86020 86021 86022 86023 86024 86025 86026 86027 86028 86029 86030 86031 86032 86033 86034 86035 86036 86037 86038 86039 86040 86041 86042 86043 86044 86045 86046 86047 86048 86049 86050 86051 86052 86053 86054 86055 86056 86057 86058 86059 86060 86061 86062 86063 86064 86065 86066 86067 86068 86069 86070 86071 86072 86073 86074 86075 86076 86077 86078 86079 86080 86081 86082 86083 86084 86085 86086 86087 86088 86089 86090 86091 86092 86093 86094 86095 86096 86097 86098 86099 86100 86101 86102 86103 86104 86105 86106 86107 86108 86109 86110 86111 86112 86113 86114 86115 86116 86117 86118 86119 86120 86121 86122 86123 86124 86125 86126 86127 86128 86129 86130 86131 86132 86133 86134 86135 86136 86137 86138 86139 86140 86141 86142 86143 86144 86145 86146 86147 86148 86149 86150 86151 86152 86153 86154 86155 86156 86157 86158 86159 86160 86161 86162 86163 86164 86165 86166 86167 86168 86169 86170 86171 86172 86173 86174 86175 86176 86177 86178 86179
```

Observe que, com o código acima, a seria decrementado indefinidamente e sempre permaneceria menor do que b. A condição está mal feita e o código errado!

Repetição por Condição

- **Informações Importantes:**

- Um laço infinito é uma sequência de comandos em seu programa que se **repete infinitamente**, gerando erros e “travamento”. Isso acontece sempre que uma condição de parada existe mas nunca é atingida.
- A **condição** estabelecida pode ser qualquer expressão que resulte em uma resposta do tipo “**verdadeiro**” ou “**falso**” e que utilize operadores matemáticos, relacionais e/ou lógicos.
 - Idênticas às condições do **if / else**.
- Não se usa ponto e vírgula após o comando while. Seu início e final são marcados por chaves (**{ e }**).
 - Caso somente um **único comando** seja executado, as chaves são **opcionais**.
Exemplo:

```
while(x < 100)
    x++;
```

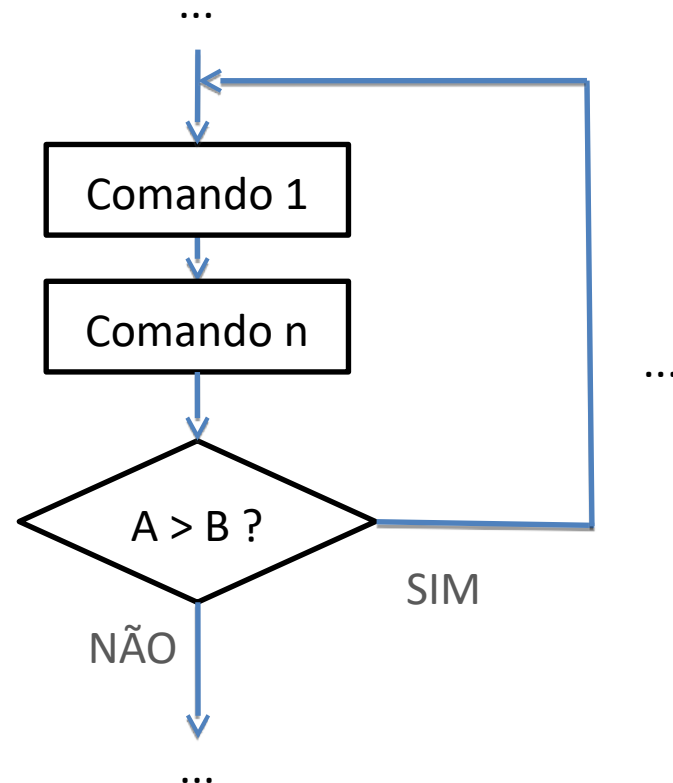
Repetição por Condição

- Existe ainda, uma variação do laço “enquanto”, que é utilizado sempre que o **código a ser repetido precisa ser executado ao menos uma vez**, sem que a condição seja testada.
- A diferença em relação à anterior é que, na segunda, a verificação da **condição somente é realizada no final do bloco**. Assim, o bloco é sempre executado pelo menos uma vez.

Faça
 sequência de comandos
Enquanto condição

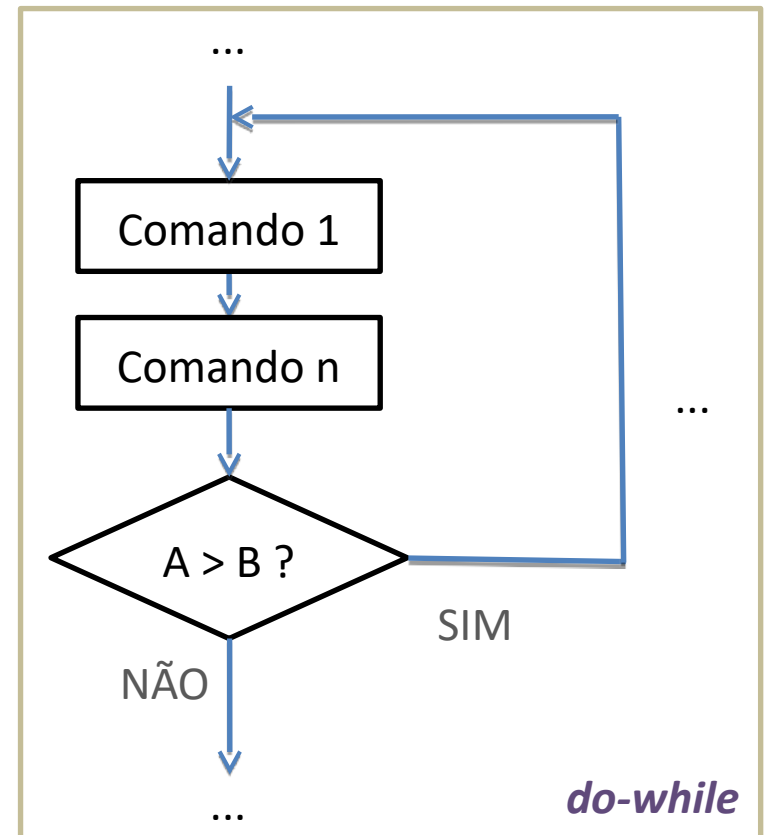
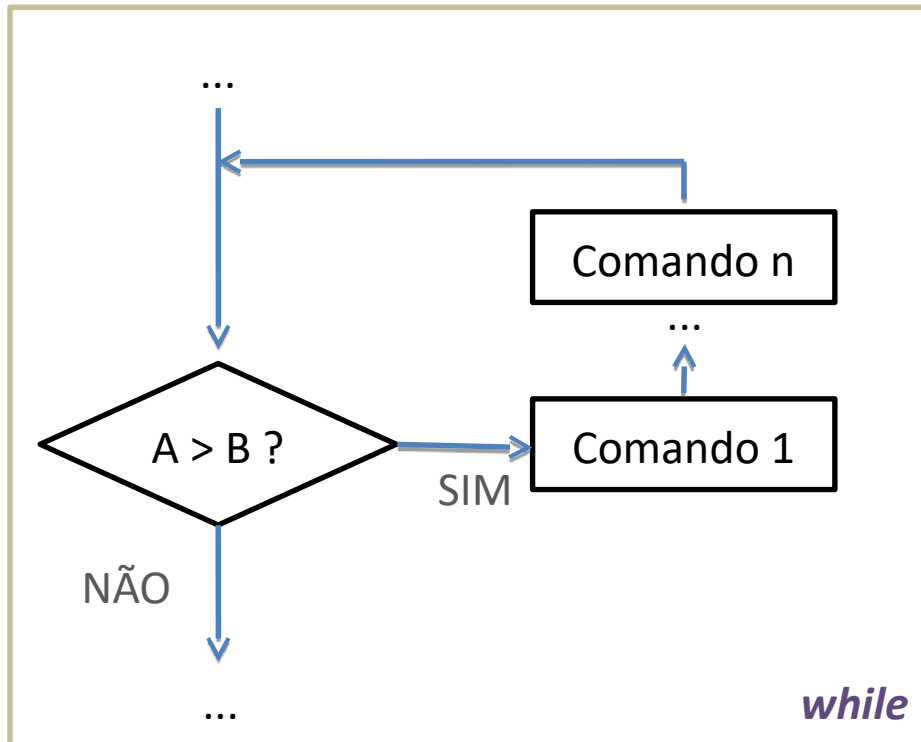
Em C, sua forma geral é:

```
do {  
    // Faça alguma coisa  
} while (condição);
```



Repetição por Condição

- Repare bem na diferença entre os fluxogramas do comando *while*, e do comando *do-while*.
- A única diferença clara é o local onde a condição é testada.
while, no início, *do-while*, no final.



Repetição por Condição

- **Informações Importantes:**

- Continua sendo de inteira responsabilidade do programador, **estabelecer uma condição de término que possa ser atingida durante a execução**. Caso contrário, ficará preso num laço infinito e nunca terminará sua execução.
- Outra diferença entre as duas técnicas de repetição por condição, é a utilização do **ponto e vírgula**.
 - *while* nunca utiliza ponto e vírgula para finalizar o comando.
 - *do-while* precisa terminar **SEMPRE** com ponto e vírgula. Veja:

```
while (a > b)
{
    c = a;
    a = b;
    b = c;
}
```

```
do
{
    c = a;
    a = b;
    b = c;
} while (a > b);
```



Repetição por Condição

- Vamos utilizar o do-while para fazer uma melhoria em outro programa exemplo. Veja o enunciado:

Muitos alunos do IESTI conseguiram bolsas e irão viajar em breve pelo programa Ciência sem Fronteiras. Alguns deles estão indo para os Estados Unidos e gostariam de sua ajuda: eles precisam de um programa que faça a conversão de dólares para reais! Sem sua ajuda eles irão acabar se perdendo em suas finanças e terão que lavar muitos pratos para vir embora. Siga os passos:

- a. Declare duas variáveis do tipo ***float***, uma para o valor em reais e outra para o mesmo valor em dólares;
- b. Peça para que o usuário entre com um valor em dólares (***printf***);
- c. Guarde este valor na variável correspondente (***scanf***);
- d. Faça o cálculo do valor em reais, utilizando a seguinte taxa de câmbio: US\$ 1 = R\$ 2,50.
- e. Imprima na tela o valor calculado em reais (***printf***).

Havia um inconveniente: caso o usuário queira fazer mais de uma conversão sem ter que fechar e abrir o programa novamente, isso não é possível.

Podemos resolver o caso com um do-while. Veja!

Repetição por Condição

```
#include <stdio.h>

int main()
{
    // Declarando variaveis
    float reais, dolares, cambio;

    // Pedimos os dados iniciais ao usuário
    printf("Entre com o valor em dolares: ");
    scanf("%f", &dolares);

    // Realizamos cálculo
    reais = dolares * 2.50;

    // Mostramos resultados
    printf("%f dolares = %f reais\n", dolares, reais);
    system("pause");

    return 0;
}
```

Este é o código original.

Repare que, o que queremos agora é que toda o programa seja repetido, **a partir do primeiro *printf*, até o último**, até que o usuário resolva sair do programa por conta própria.

Como poderíamos proceder?

```

#include <stdio.h>

int main()
{
    // Declarando variaveis
    float reais, dolares, cambio;
    int repete;

    do
    {
        // Limpa a tela do console
        system("cls");

        // Pedimos os dados iniciais ao usuário
        printf("Entre com o valor em dolares: ");
        scanf("%f", &dolares);

        // Realizamos cálculo
        reais = dolares * 2.50;

        // Mostramos resultados
        printf("%.2f dolares = %.2f reais\n\n", dolares, reais);

        printf("Digite 1 para continuar ou 0 para sair: ");
        scanf("%d", &repete);

    } while (repete != 0);

    return 0;
}

```

- 1) Acrescentamos uma nova variável chamada **“repete”**.
- 2) Acrescentamos uma estrutura do tipo **do-while** em volta de todo o código que precisamos que seja repetido.
- 3) Acrescentamos um comando para **limpar a tela**, sempre que formos fazer uma nova conversão (opcional).
- 4) Perguntamos ao usuário se ele prefere **continuar** ou **sair do programa**.
- 5) Se ele digitar 0, saímos do laço. Caso contrário, repetimos tudo novamente.

Hands-On!

- Escreva um novo programa em C que faça a soma de todos os números múltiplos de três no intervalo entre 0 e 100.
 - Utilize uma estrutura de repetição do tipo **do-while** para completar a tarefa.
 - Não esqueça de declarar uma variável para manter o **número atual** e outra para manter a **soma** de todos os números até o atual.
 - Quando o número atual for **maior que 100**, termine seu laço.

Verificando a conta...

$$S_n = \frac{(a_1 + a_n)n}{2} = (3 + 99) \cdot 33 / 2 = \mathbf{1683}$$

Repetição com Contador

- Percebemos que, tanto no ***while*** quanto no ***do-while***, não estamos necessariamente preocupado com a quantidade de vezes em que o código será repetida.
 - O coração dessas estruturas é uma **condição**.
- Existe um outro tipo de estrutura que realiza uma “repetição controlada por **contador**” ou “repetição com **variável de controle**”.
 - Ela é utilizada quando precisamos repetir um bloco de comandos uma **quantidade preestabelecida de vezes**.
 - Para tal, contamos com o auxílio de uma variável que conta as repetições: um **contador**..

Repetição com Contador

- Sua fórmula geral é a seguinte:

Para contador de X até Y, com passo Z, faça
sequência de comandos

Fim Para

Em C, sua forma geral é:

```
int i = 0;  
...  
for(i = X; i < Y, i++)  
{  
    // faça alguma coisa  
}
```

O contador utilizado será sempre uma variável inteira. Neste caso, i.

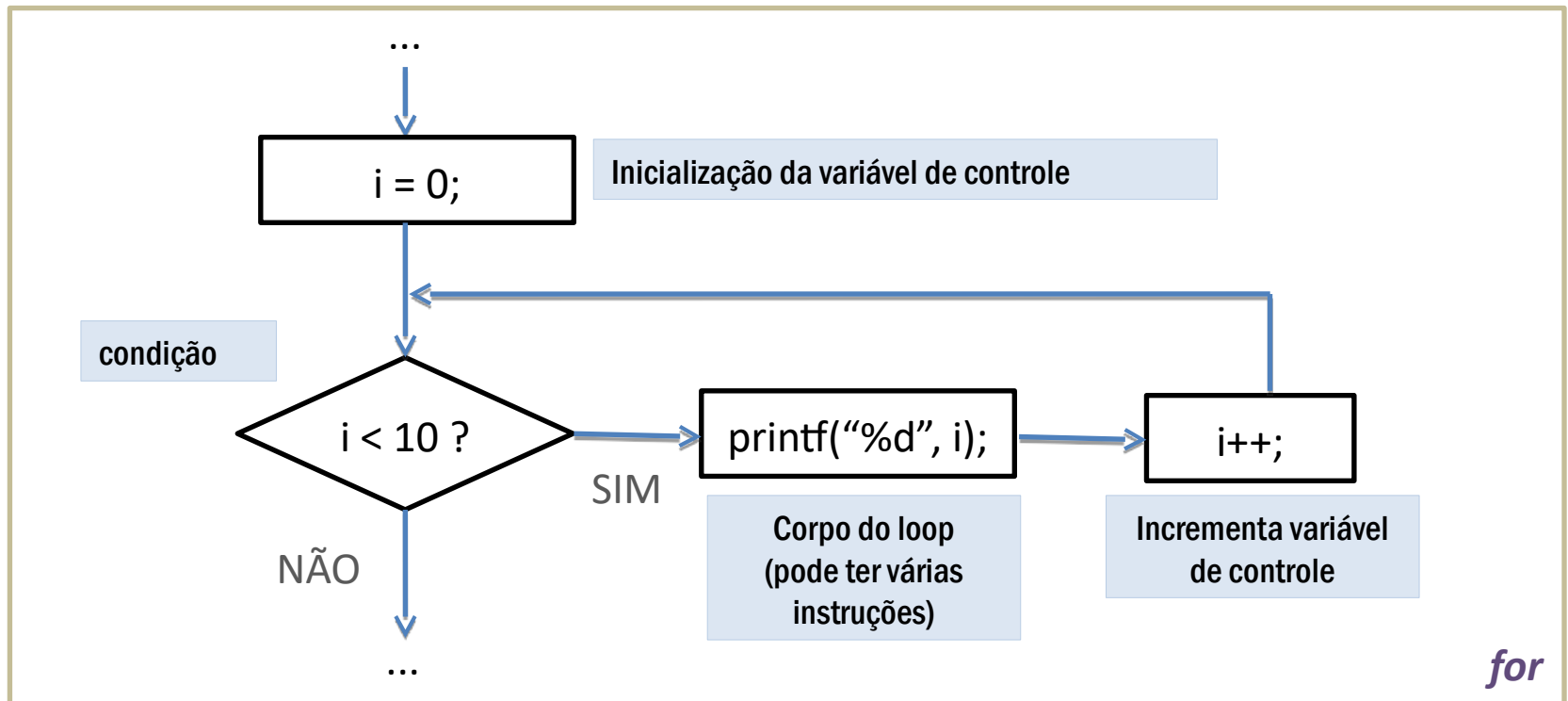
O contador i será incrementado de um em um, começando em X, até Y-1. Quando for igual a Y, o laço é quebrado e o programa prossegue sua execução.

Repetição com Contador

- E como ficaria o fluxograma?

Imagine o seguinte **exemplo**:

```
// imprime todos os inteiros de 0 a 9
for(i = 0; i < 10; i++)
    printf("%d", i);
```



for

Repetição com Contador

- Vamos fazer, agora, um programa que soma todos os números entre 0 e 9, calcule seus triplos e some. Ao final, o programa deve imprimir o valor total da soma:

```
#include <stdio.h>

int main()
{
    int num, soma;

    // Inicializa soma
    soma = 0;

    // inicia em 0 e tem que ser menor que 10.
    for(num = 0; num < 10; num++)
    {
        soma += num*3; // calcula triplo de cada num
    }

    printf("O valor total da soma eh %d", soma);

    return 0;
}
```

Repetição com Contador

- É possível modificar o valor do contador também de outras maneiras. Por exemplo, ao invés de incrementar a partir de 0, podemos decrementar a partir de 9. Veja:

```
// inicia em 9 e tem que ser maior ou igual a 0.  
for(num = 9; num >= 0; num--)  
{  
    soma += num*3; // calcula triplo de cada num  
}
```

- Mais uma vez é importante salientar que o bloco de comandos deve ser limitado por chaves. As chaves somente são opcionais quando um único comando será repetido pelo laço for.
- **IMPORTANTE:** A inicialização, teste e incremento da variável são separados SEMPRE por **PONTO E VÍRGULA**.

Repetição com Contador

- **Outro exemplo:** P.A. iniciando em 2 e terminando em 100, com razão igual a 2. Qual o resultado da soma de todos os seus itens?

```
#include <stdio.h>

int main()
{
    int num, soma;

    // Inicializa soma
    soma = 0;

    // inicia em 2 e tem que ser maior ou igual a 0.
    for(num = 2; num <= 100; num += 2)
    {
        soma += num; // calcula triplo de cada num
    }

    printf("O valor total da soma da PA eh %d", soma);

    return 0;
}
```

O mais comum é sempre fazer o contador “caminhar” de um em um. No entanto, pode ser necessário que o passo seja diferente (dois, no exemplo), e a estrutura *for* nos dá essa liberdade.

Hands-On!

- Faça um programa que escreva na tela, em ordem alfabética, todas as letras do alfabeto, começando em 'a' e terminando em 'z'.
 - Declare uma variável de controle (contador);
 - Inicialize-a com o valor referente a 'a' na tabela ASCII.
 - Incremente sempre em uma unidade.
 - Continue enquanto o contador for menor ou igual ao valor de 'z' na tabela ASCII.

```
Lista com as letras do alfabeto:  
a b c d e f g h i j k l m n o p q r s t u v w x y z  
  
Process returned 0 (0x0)   execution time : 0.009 s  
Press any key to continue.
```

Break e Continue

- **break:**

Ao utilizar o comando break, **o laço se interrompe e para imediatamente de se repetir**, e o programa continua sua execução normal, a partir do primeiro comando após o loop.

- **continue:**

Ao utilizar o comando continue, **a iteração atual do laço é interrompida**, e o laço começa novamente, a partir da próxima repetição.

- Se utilizada dentro do for, o contador continua sendo incrementado.

break

```
// inicia em 9 e tem que ser maior ou igual a 0.  
for(letra = 'a'; letra <= 'z'; letra++)  
{  
    if(letra == 'd')  
        break;  
    printf("%c ", letra);  
}
```

```
Lista com as letras do alfabeto:  
a b c
```

Repetição para imediatamente

continue

```
// inicia em 9 e tem que ser maior ou igual a 0.  
for(letra = 'a'; letra <= 'z'; letra++)  
{  
    if(letra == 'd')  
        continue;  
    printf("%c ", letra);  
}
```

```
Lista com as letras do alfabeto:  
a b c e f g h i j k l m n o p q r s t u v w x y z
```

Iteração para imediatamente, mas repetição continua. Apenas d não é impresso.

Estruturas de Repetição

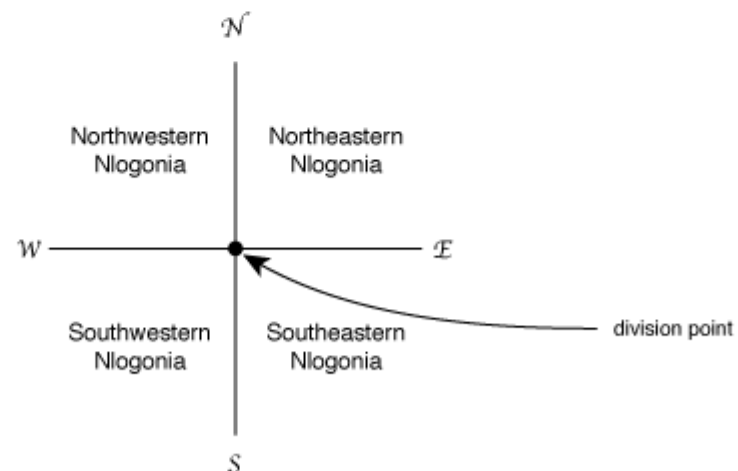
- Agora já sabemos tudo sobre os tipos básicos de C, suas operações e capacidades.
- Além disso, acabamos de conhecer os **cinco elementos básicos** mais importantes para qualquer linguagem de programação:
 - Se, então, senão (*if-else*);
 - Escolha (*switch*);
 - Enquanto (*while*);
 - Faça-Enquanto (*do-while*);
 - E, finalmente, Para (*for*).

Divisão da Nlogônia

Por Ricardo Anido  Brasil**Timelimit: 1**

Depois de séculos de escaramuças entre os quatro povos habitantes da Nlogônia, e de dezenas de anos de negociações envolvendo diplomatas, políticos e as forças armadas de todas as partes interessadas, com a intermediação da ONU, OTAN, G7 e SBC, foi finalmente decidida e aceita por todos a maneira de dividir o país em quatro territórios independentes.

Ficou decidido que um ponto, denominado ponto divisor, cujas coordenadas foram estabelecidas nas negociações, definiria a divisão do país, da seguinte maneira. Duas linhas, ambas contendo o ponto divisor, uma na direção norte-sul e uma na direção leste-oeste, seriam traçadas no mapa, dividindo o país em quatro novos países. Iniciando no quadrante mais ao norte e mais ao oeste, em sentido horário, os novos países seriam chamados de Nlogônia do Noroeste, Nlogônia do Nordeste, Nlogônia do Sudeste e Nlogônia do Sudoeste.



A ONU determinou que fosse disponibilizada uma página na Internet para que os habitantes pudessem consultar em qual dos novos países suas residências estão, e você foi contratado para ajudar a implementar o sistema.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um inteiro K indicando o número de consultas que serão realizadas ($0 < K \leq 10^3$). A segunda linha de um caso de teste contém dois números inteiros N e M representando as coordenadas do ponto divisor ($-10^4 < N, M < 10^4$). Cada uma das K linhas seguintes contém dois inteiros X e Y representando as coordenadas de uma residência ($-10^4 \leq X, Y \leq 10^4$). Em todas as coordenadas dadas, o primeiro valor corresponde à direção leste-oeste, e o segundo valor corresponde à direção norte-sul.

O final da entrada é indicado por uma linha que contém apenas o número zero.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma linha contendo:

- a palavra `divisa` se a residência encontra-se em cima de uma das linhas divisórias (norte-sul ou leste-oeste);
- `NO` se a residência encontra-se na Nlogônia do Noroeste;
- `NE` se a residência encontra-se na Nlogônia do Nordeste;
- `SE` se a residência encontra-se na Nlogônia do Sudeste;
- `SO` se a residência encontra-se na Nlogônia do Sudoeste.

Exemplo de Entrada	Exemplo de Saída
3	NE
2 1	divisa
10 10	NO
-10 1	divisa
0 33	NE
4	SO
-1000 -1000	SE
-1000 -1000	
0 0	
-2000 -10000	
-999 -1001	
0	

Dúvidas?

- Usem o fórum no SIGAA para eventuais dúvidas.
- Também podem me mandar e-mail com questões.
- Compareçam na conferência agendada para conversarmos sobre a disciplina.
- Façam a lista de estrutura de repetição e enviem no SIGAA.
- Bom trabalho!