

# Programação e Desenvolvimento de Software I

## Repetição

---

Prof. Héctor Azpúrua  
(slides adaptados do Prof. Pedro Olmo)

# Problema I

- Dado o valor da variável  $N$ , determine a soma dos números inteiros de  $1$  a  $N$ 
  - Deseja-se calcular o valor de:  $1 + 2 + 3 + \dots + N$
- **Observação:**
  - Não sabemos, *a priori*, quantos termos serão somados, pois o valor de  $N$  é estabelecido dinamicamente...
- **Como fazer?**
  - Para se calcular esta soma, utiliza-se o comando **while**

O comando **while** permite que um conjunto de instruções seja executado tantas vezes quantas forem necessárias, **enquanto** uma condição for verdadeira

# Problema 1

- Dado o valor da variável N, determine a soma dos números inteiros de 1 a N

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

```
// gcc p4.c -o p4 -lm && ./p4
```

```
int main() {
    int n;
    int i, s;
    printf("Digite o valor de N:");
    scanf("%d", &n);
    s = 0;
    i = 1;
    while (i <= n) {
        s = s + i;
        i++;
    }
    printf("soma=%d\n", s);
    return 0;
}
```

Laço ou loop!



# Comando While

- Quando um programa executa um conjunto de instruções repetidas vezes, diz-se que o programa está realizando um **processamento iterativo**
- Cada execução do conjunto de instruções denomina-se uma iteração. Exemplo de uso do comando **while**:

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

Instante	s	i	(i <= 3)
inicial	0	1	Verdadeiro

# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);  
  
while (i <= n) {  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001		
10FA0002		
10FA0003		
10FA0004		

# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);  
  
while (i <= n) {  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001		
10FA0002		
10FA0003		
10FA0004		


# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);  
  
while (i <= n) {  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	
10FA0002	<b>i</b>	<b>1</b>
10FA0003	<b>s</b>	<b>0</b>
10FA0004		

# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);
```

```
while (i <= n) {  Verdadeiro!  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	<b>3</b>
10FA0002	<b>i</b>	<b>1</b>
10FA0003	<b>s</b>	<b>0</b>
10FA0004		




# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);  
  
while (i <= n) {  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	3
10FA0002	<b>i</b>	<b>2</b>
10FA0003	<b>s</b>	<b>1</b>
10FA0004		

# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);
```

```
while (i <= n) {  Verdadeiro!  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	3
10FA0002	<b>i</b>	2
10FA0003	<b>s</b>	1
10FA0004		


# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);  
  
while (i <= n) {  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	3
10FA0002	<b>i</b>	<b>3</b>
10FA0003	<b>s</b>	<b>3</b>
10FA0004		

# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);
```

```
while (i <= n) {  Verdadeiro!  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	3
10FA0002	<b>i</b>	3
10FA0003	<b>s</b>	3
10FA0004		

# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);  
  
while (i <= n) {  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	3
10FA0002	<b>i</b>	<b>4</b>
10FA0003	<b>s</b>	<b>6</b>
10FA0004		

# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);
```

```
while (i <= n) {  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

← **FALSO!**

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	3
10FA0002	<b>i</b>	<b>4</b>
10FA0003	<b>s</b>	<b>6</b>
10FA0004		

# Comando While

```
int n;  
int i=1, s=0;  
printf("Digite o valor de N:");  
scanf("%d", &n);  
  
while (i <= n) {  
    s = s + i;  
    i++;  
}  
printf("soma=%d\n", s);
```

Endereço	Variável	Conteúdo
10FA0001	<b>n</b>	3
10FA0002	<b>i</b>	<b>4</b>
10FA0003	<b>s</b>	<b>6</b>
10FA0004		

A screenshot of a terminal window with a black background. The text 'soma=6' is displayed in white. The terminal has a standard window title bar with buttons for minimize, maximize, and close.

# O comando While

- A solução do problema sempre irá terminar, pois **i** será maior do que **N** em algum momento
- Porém, pode a execução de um programa com processamento iterativo **não terminar?**
- Observe:

```
s = 0;  
i = 0;  
while (i < 3) {  
    i--;  
    s = s + i;  
}
```

Nunca vai terminar!  
Erro de lógica

```
s = 0;  
i = 0;  
while (i > -3) {  
    i--;  
    s = s + i;  
}
```





# O comando While

- **Atenção!**
- Em alguns casos, o loop infinito pode ser desejável.
  - Exemplo: um programa que monitora um reator nuclear deve estar sempre em execução.
- Neste caso, pode-se escrever:

```
while (1) {  
    monitora_reator();  
}
```

# O comando Do-While

- Outra forma de repetir um conjunto de instruções é com o comando **do-while**

```
s = 0;  
i = 1;  
do {  
    s = s + i;  
    i++;  
} while (i <= n);
```

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

- Veja que no comando **while**, a **condição é testada antes da execução** das instruções, ao contrário do comando do-while.
  - O que acontece para N=0?

# Lembrando...

## Nosso primeiro algoritmo

- Problema 2:
  - Suponha que soma (+) e subtração (-) são as únicas operações disponíveis
  - Dados dois números inteiros positivos **A** e **B**, determine o **quociente** e o **resto** da divisão de **A** por **B**
- Para resolver o Problema 1, **precisamos de um algoritmo:**

Sequência finita de **instruções** que, ao ser executada, chega a uma **solução de um problema**

# Algoritmos estruturados

- Para escrever este algoritmo, podemos usar a seguinte ideia:
  - Representar os números  $A$  e  $B$  por retângulos de larguras proporcionais aos seus valores
  - Verificar quantas vezes  $B$  cabe em  $A$

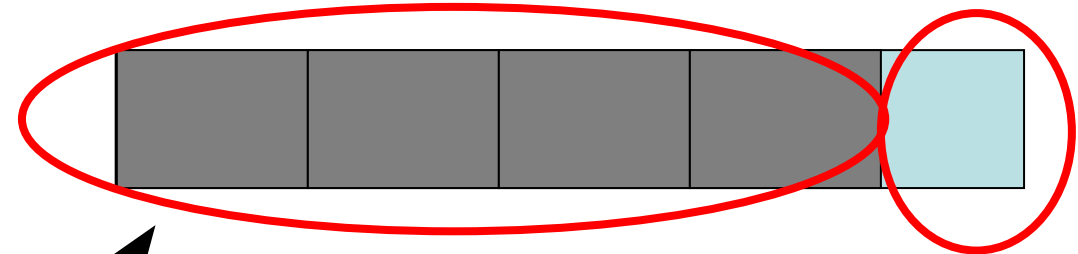
# Algoritmos estruturados

## Problema 2

- Suponha que soma (+) e subtração (-) são as únicas operações disponíveis em C. Dados dois números inteiros positivos  $A$  e  $B$ , determine o **quociente** e o **resto** da divisão de  $A$  por  $B$



$$A = 9, B = 2$$



Quociente de  $A/B$ : nº de vezes que B cabe em A

O que sobra em A é o resto da divisão


# Algoritmos estruturados

## Problema 2

- Pode-se escrever este algoritmo como:

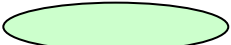
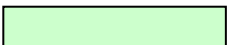
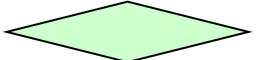



1. Sejam  $A$  e  $B$  os valores dados;
2. Atribuir o valor 0 ao quociente ( $q$ );
3. Enquanto  $B \leq A$ :
4. {
5.     Somar 1 ao valor de  $q$ ;
6.     Subtrair  $B$  do valor de  $A$ ;
7. }
8. Atribuir o valor final de  $A$  ao resto ( $r$ );

Outra forma  
de representar



# Fluxograma

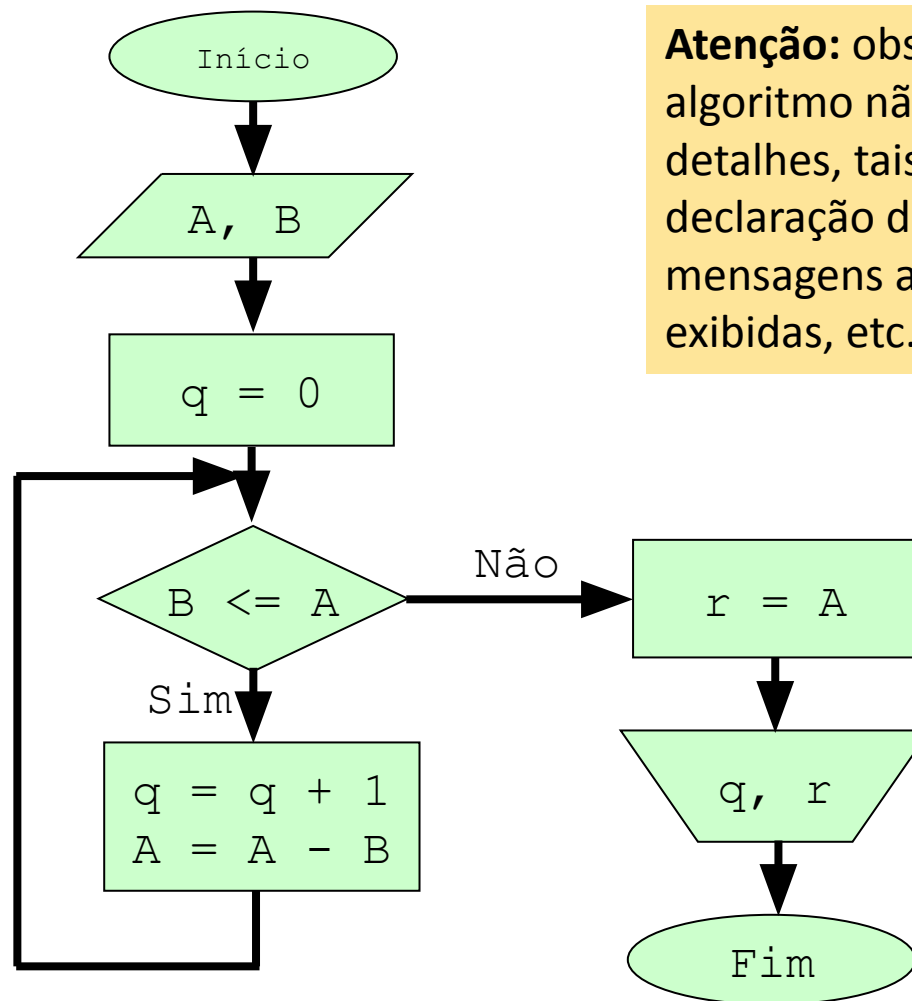
- É conveniente representar algoritmos por meio de fluxogramas (diagrama de blocos)
- Em um fluxograma, as operações possíveis são representadas por meio de figuras:

Figura	Usada para representar
	Início ou fim
	Atribuição
	Condição
	Leitura de dados
	Apresentação de resultados
	Fluxo de execução

# Fluxograma

## Exemplo do Problema 2

- O algoritmo para o Problema 2 pode ser representado pelo seguinte fluxograma
- Não estão atrelados a uma linguagem de programação específica!



**Atenção:** observe que um algoritmo não inclui detalhes, tais como declaração de variáveis, mensagens a serem exibidas, etc.

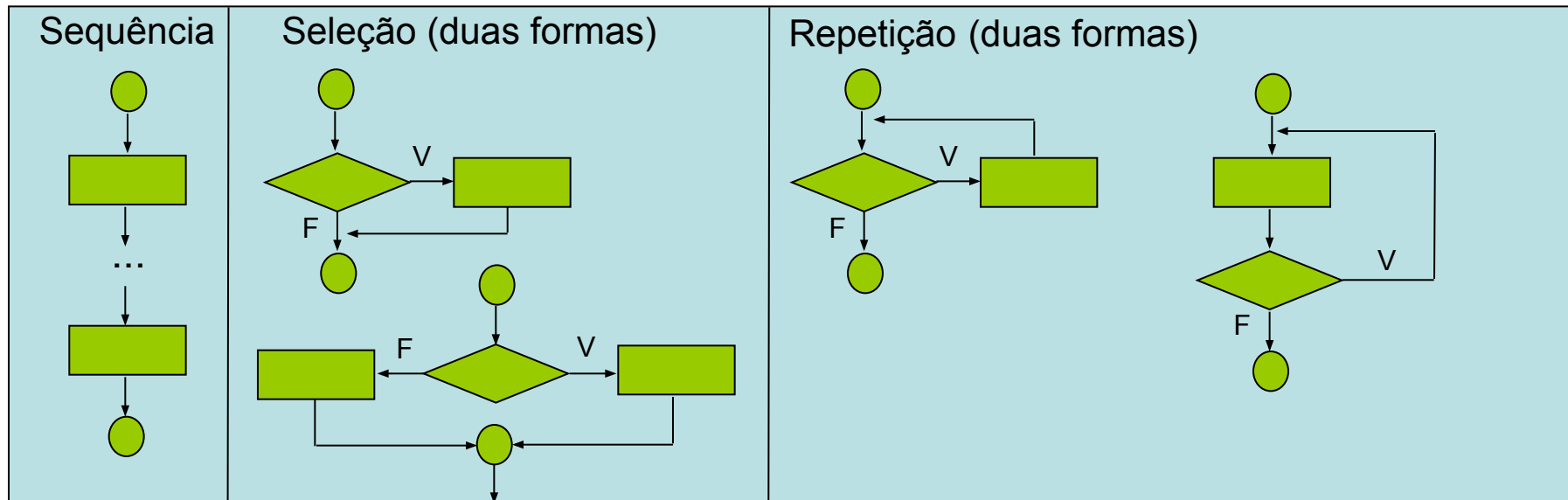


# Programação Estruturada

- Um algoritmo tem sempre um **único bloco início** e deve conter, pelo menos, **um bloco fim**
  - **A execução segue setas**
- Em geral, construir um algoritmo é mais difícil do que codificar em uma linguagem
- Porém, a construção de algoritmos pode se beneficiar de técnicas, como a **Programação Estruturada**

# Programação Estruturada

- Na **Programação Estruturada**, usa-se apenas três estruturas: **sequência**, **seleção** e **repetição**
  - Cada estrutura tem apenas um único ponto de entrada e um único ponto de saída (círculos)



# Programação Estruturada

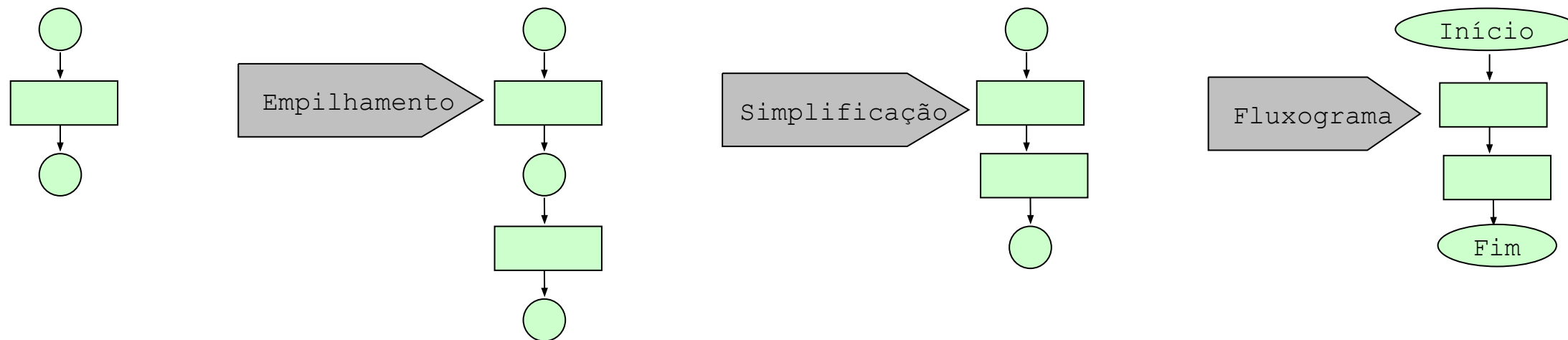
- Atenção!
  - Os retângulos indicam qualquer ação, incluindo leitura de dados ou exibição de resultados
- Construir programas estruturados corresponde a combinar estas estruturas de duas maneiras:
  - **Regra do empilhamento:** o ponto de saída de uma estrutura pode ser conectado ao ponto de entrada de outra estrutura
  - **Regra do aninhamento:** um retângulo de uma estrutura pode ser substituído por uma outra estrutura qualquer

# Programação Estruturada

- Estas regras podem ser aplicadas quantas vezes forem necessárias e em qualquer ordem
- Na construção de fluxogramas, pode-se substituir o **primeiro ponto de entrada** e os **últimos pontos de saída** por ovais (início e fim)
- Os demais pontos de entrada e saída podem ser removidos

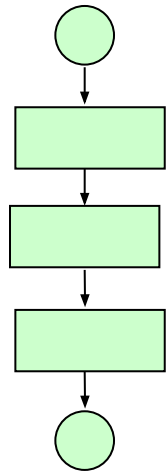
# Programação Estruturada

## ■ Exemplo: aplicação da regra de empilhamento

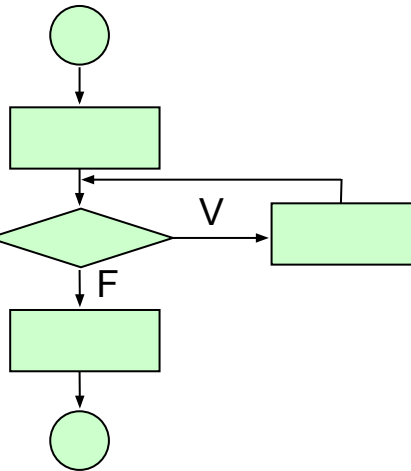


# Programação Estruturada

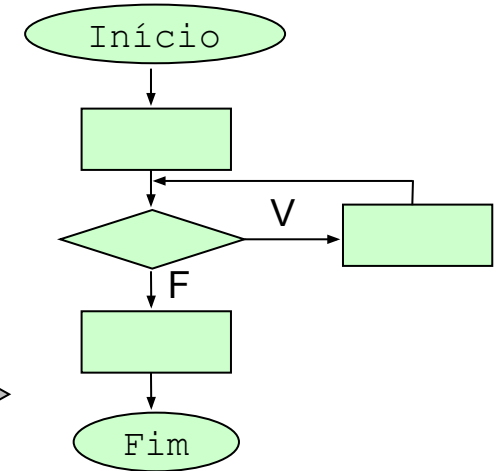
- Suponha que soma (+) e subtração (-) são as únicas operações disponíveis em C. Dados dois números inteiros positivos A e B, determine o quociente e o resto da divisão de A por B.



Aninhamento e Simplificação



Fluxograma



# Problema 3

- Problema:
  - Dados dois números **inteiros A e B**, determinar o **máximo divisor comum (MDC)** destes dois números
- Como calcular o MDC entre dois números **A e B**, representado por **mdc(A,B)**?
  - **Método das divisões sucessivas:** efetua-se várias divisões até chegar em uma divisão exata.

# Problema 3

- Suponha que se deseja calcular **mdc(48,30)**
  - Divide-se o número maior pelo menor:
    - $48/30 = 1$  (resto **18**)
  - Divide-se o divisor anterior pelo resto anterior e, assim sucessivamente:
    - $30/18 = 1$  (resto **12**)
    - $18/12 = 1$  (resto **6**)
    - $12/6 = 2$  (resto **0** – divisão exata)
  - **MDC = 6**



# Algoritmo

## Problema 3

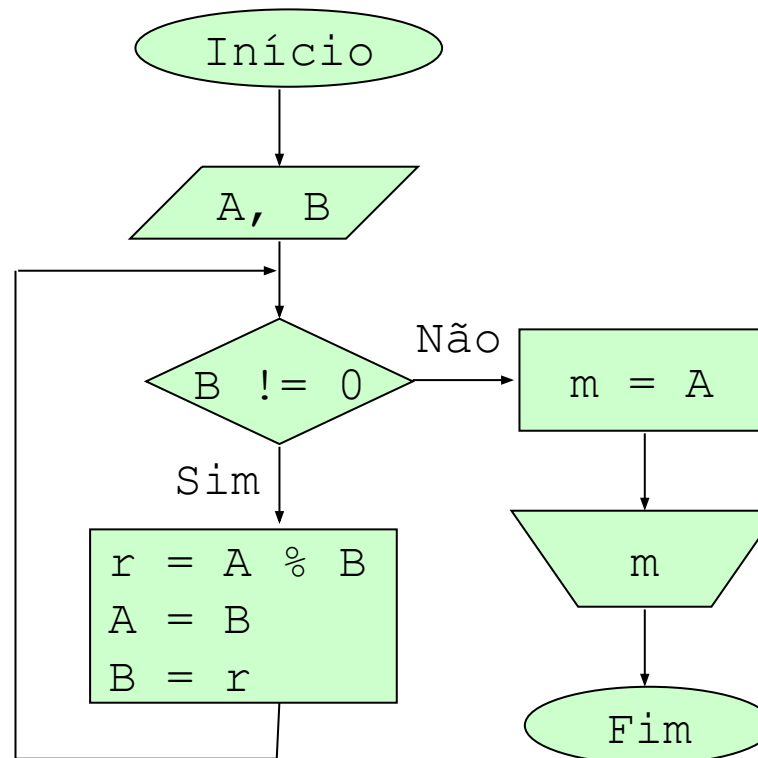
- Um **algoritmo** para este problema pode ser escrito como:

```
1. enquanto B for diferente de zero:
2. {
3.     r = resto da divisão de A por B;
4.     A = B;
5.     B = r;
6. }
7. mdc = A;
```

# Fluxograma

## Problema 3


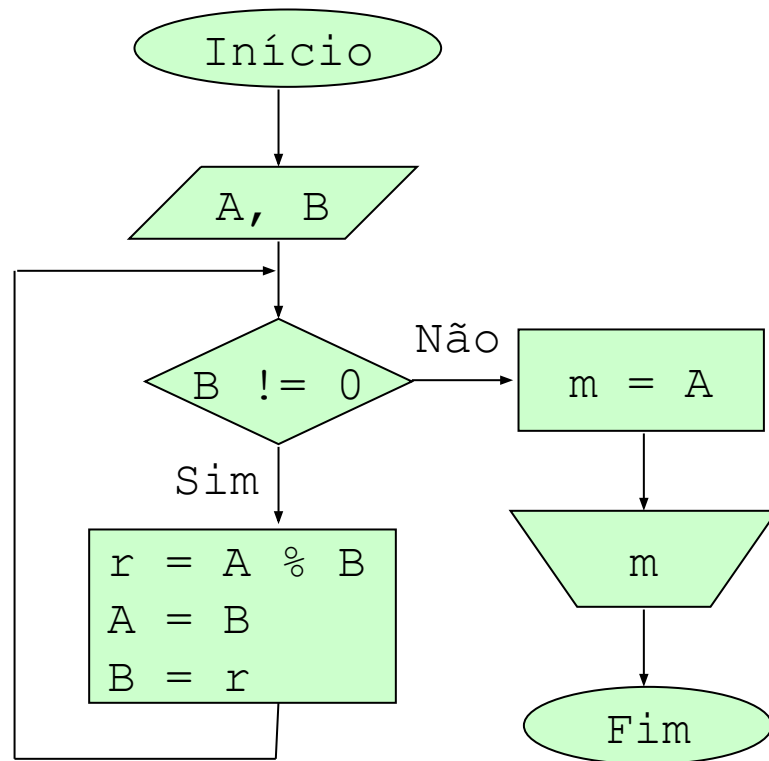
- Um algoritmo para este problema pode ser representado pelo seguinte **fluxograma**:



# Código

## Problema 3

- O código para resolver este problema pode ser escrito como:



```
int MDC(int a, int b) {  
    int r;  
    while (b != 0) {  
        r = a % b;  
        a = b;  
        b = r;  
    }  
    return a;  
}
```

# Problema 4

- Problema:
  - Escrever um programa para ler dois inteiros do teclado, calcular o MDC entre eles e, caso o usuário deseje, **repetir o processo**

# Problema 4

```
int MDC(int a, int b) {  
    int r;  
    while (b != 0) {  
        r = a % b;  
        a = b;  
        b = r;  
    }  
    return a;  
}
```

Após exibir o valor do **mdc** o programa exibe a mensagem: Continua? (S/N)

Espera-se que o usuário digite **S** ou **N**, caracteres que serão lidos pela função **getchar**

O loop sera executado por enquanto o usuário digite '**S**'

```
int main() {  
    int a, b, m;  
    char c;  
    do {  
        printf("Digite os valores de A e B: ");  
        scanf("%d %d", &a, &b);  
        m = MDC(a, b);  
        printf("MDC(%d,%d) = %d\n", a, b, m);  
        printf("Continua? (S/N): ");  
        do {  
            c = getchar();  
            if (c != 'S' && c != 'N' && c != '\n')  
                printf("So as letras 'S' ou 'N' sao permitidas\n");  
        } while (c != EOF && (c != 'S' && c != 'N'));  
        while (c == 'S');  
        printf("\n");  
    } while (1);  
    return 0;  
}
```

# Problema 4

- Para evitar a comparação com letras maiúsculas e minúsculas de forma separada, pode-se usar a função **toupper**:

```
#include <ctype.h>

do {
    c = toupper(getchar());
    if (c != 'S' && c != 'N' && c != '\n')
        printf("So as letras 'S' ou 'N' sao permitidas\n");
} while (c != EOF && (c != 'S' && c != 'N'));
```

Verifica se o valor de seu parâmetro corresponde ao código ASCII de uma letra minúscula:

- **Caso afirmativo:** retorna o código da letra maiúscula correspondente
- **Caso negativo:** retorna o próprio valor do parâmetro

# Problema 5

## ■ Problema:

- Escreva um programa que permita ao usuário escolher dentre as seguintes opções:
  1. Exibir o conteúdo da pasta;
  2. Exibir a hora do sistema;
  3. Exibir a data do sistema;
  4. Terminar a execução do programa.

# Problema 5

```
int main() {  
    char optei;  
    int erro;  
    do {  
        system("clear");  
        printf("A. Exibir o conteudo da pasta\n");  
        printf("B. Exibir a hora do sistema\n");  
        printf("C. Exibir a data do sistema\n");  
        printf("X. Terminar execucao\n");  
        printf("Escolha: ");
```

```
    }  
    do {  
        erro = 0;  
        optei = toupper(getchar());  
  
        if (optei == 'A') {  
            system("ls");  
        } else if (optei == 'B') {  
            system("date '+%T'");  
        } else if (optei == 'C') {  
            system("date '+%d-%m-%y'");  
        } else if (optei == 'X') {  
            // sair  
        } else {  
            printf("Opcao nao permitida\n");  
            erro = 1;  
        }  
    } while (erro == 1);  
    printf("\n");  
    system("read -p \"Pressione enter para continuar\"");  
} while (optei != 'X');  
return 0;  
}
```



# Função **system**

- O programa desenvolvido para o Problema 5 mostra diversas possibilidades de uso da **função system**:
  - `system("clear")`
    - Limpar a tela de execução
  - `system("ls")`
    - Exibir o conteúdo da pasta em uso
  - `system("date '+%T'")`
    - Exibir o horário atual
  - `system("date +%d-%m-%y")`
    - Exibir a data de hoje

# Função **system**

- Os parâmetros possíveis para a função **system** dependem do **sistema operacional** sob o qual os programas serão executados.
  - Os parâmetros: “ls”, “date”, “read -p”, etc. correspondem a **comandos do Linux**
  - Os parâmetros “CLS”, “DIR”, “TIME”, “DATE”, “PAUSE”, correspondem a comandos do **sistema DOS (Windows)**

# Perguntas?

- E-mail:
  - [hector@dcc.ufmg.br](mailto:hector@dcc.ufmg.br)
- Material da disciplina:
  - <https://pedroolmo.github.io/teaching/pdsI.html>
- Github:
  - <https://github.com/h3ct0r>



**Héctor Azpúrua**  
h3ct0r