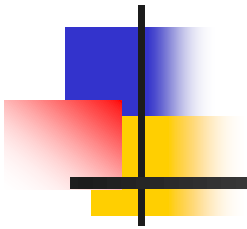
Linguagens de Programação II

Conceitos Básicos de Programação e Algoritmos.
Revisão da Linguagem C.

Dany Sanchez Dominguez

dsdominguez@gmail.com

Modulo 9 – Pavilhão Jorge Amado



Roteiro

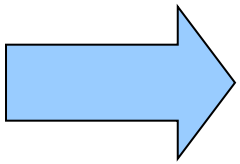
- Problemas computacionais
- Algoritmos e técnicas de representação
- Tipos de erros
- Programação estruturada
  - Sequencial, condicional e repetição
- Básico de Linguagem C
  - Estrutura de um programa
  - Tipos de dados
  - Declaração de variáveis
  - Operadores
  - Estruturas de controle
  - Operações de entrada e saída

PROBLEMAS!!!

- **Como resolver problemas de computação?**
  1. conhecer o problema a ser resolvido (*entender o problema*),
  2. extrair todas as informações ao respeito do problema (*dados e operações*),
  3. descrever claramente os passos para chegar a solução,
  4. organizar os passos segundo uma seqüência lógica que leve a solução (*algoritmo*),
  5. converter esses passos num programa utilizando uma linguagem de programação (*programa*),
  6. Verificar a solução obtida (*testes*).

ALGORITMOS

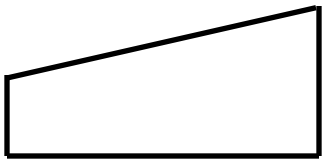
- **Algoritmo**: Conjunto de regras e operações bem-definidas e ordenadas, destinadas à solução de um problema ou de uma classe de problemas, em um número finito de passos.

Algoritmo  ***Caminho de solução para um problema***

- Existem diversos mecanismos para representar algoritmos: fluxogramas e pseudocódigo.

ALGORITMOS

- Símbolos - Fluxograma

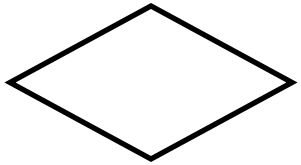
	terminador	Representa a saída ou entrada do ambiente externo
	processo	Representa qualquer tipo de processo (funções, operações)
	linha	Representa o fluxo de dados ou controles
	entrada manual	Representar os dados que sejam fornecidos em tempo de processamento

ALGORITMOS

- Símbolos - Fluxograma

exibição

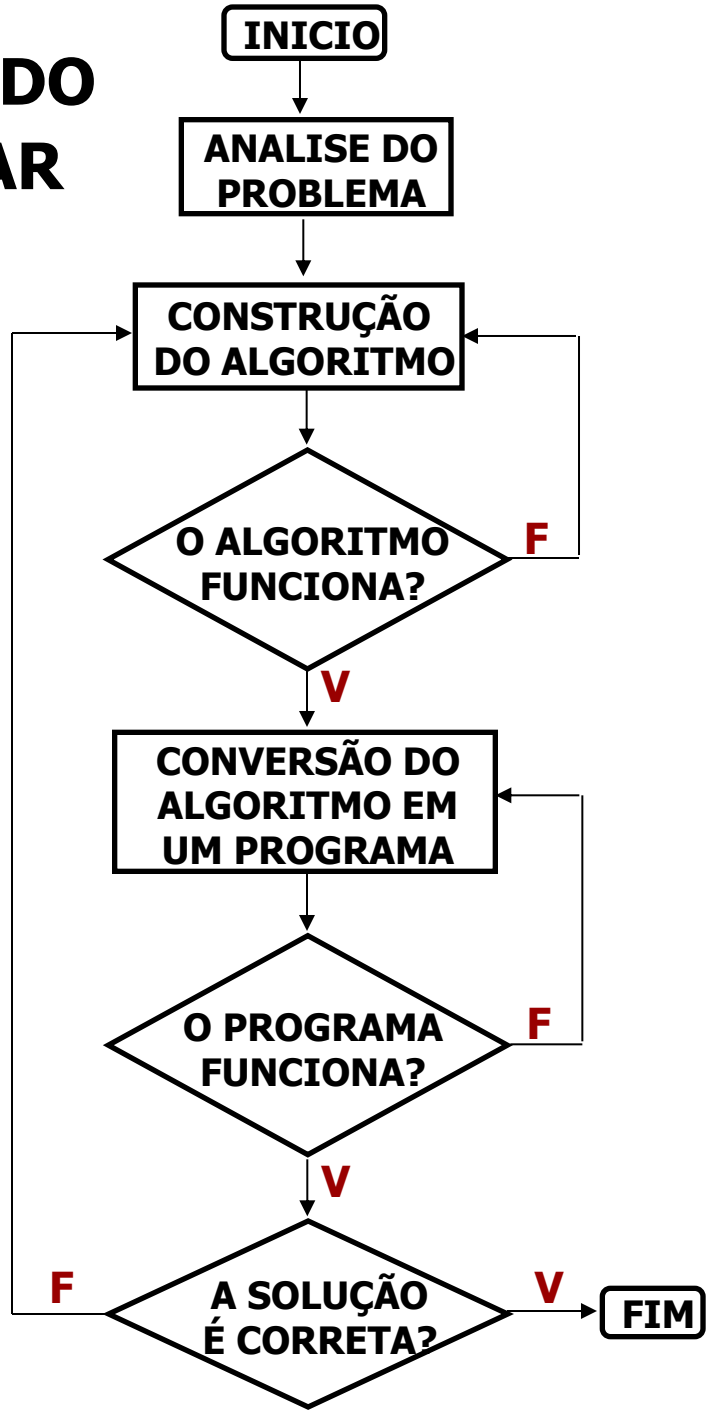
Representa dados que serão mostrados (tela, impressora)



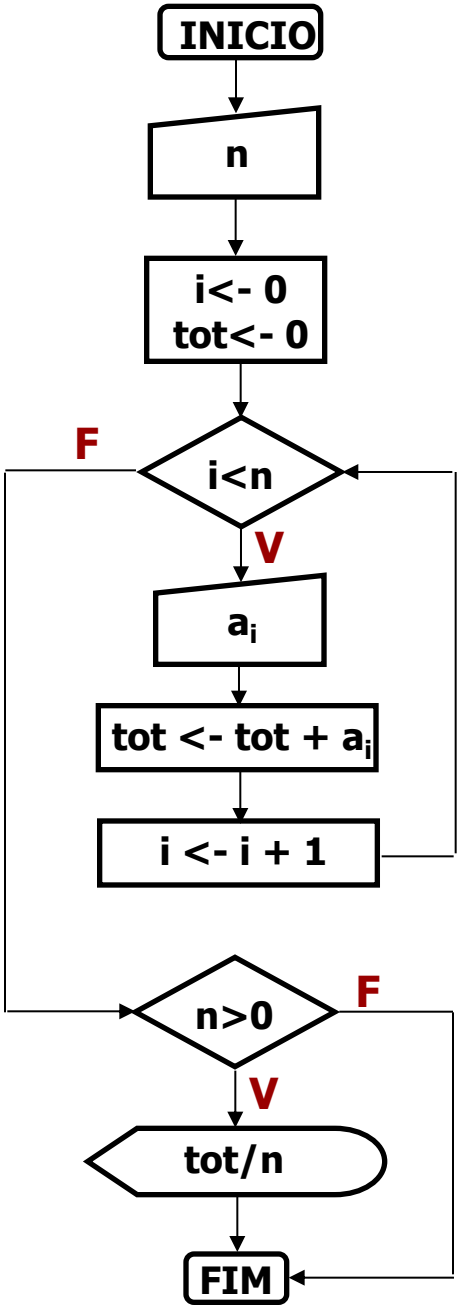
decisão

Representa uma decisão ou desvio (uma entrada; saídas: uma, duas, múltiplas) de acordo com a decisão se tomara apenas uma saída.

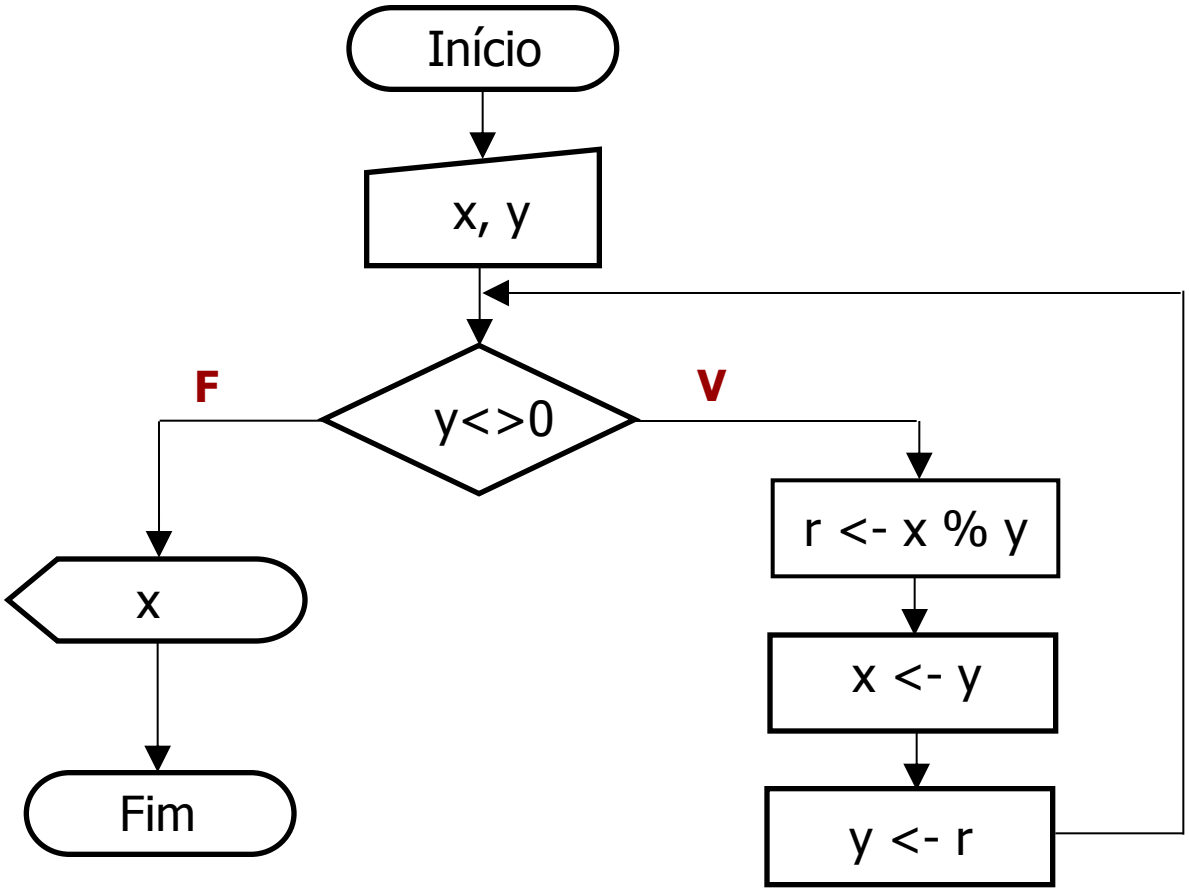
FLUXOGRAMA REPRESENTANDO A ATIVIDADE DE PROGRAMAR



QUE FAZ O ALGORITMO REPRESENTADO PELO SEGUINTE FLUXOGRAMA?



QUE FAZ O ALGORITMO REPRESENTADO PELO SEGUINTE FLUXOGRAMA?



TIPOS DE ERROS

- **Erros de sintaxe ou de compilação:**
  - são provocados pelo uso indevido dos recursos da linguagem ou erros de digitação,
  - são detectados ao compilar o programa,
  - são muito fáceis de diagnosticar e corrigir.
  - Mais comuns: ponto-e-vírgula ao final da linha; fechamento de chaves, parêntesis ou colchetes; erros de digitação em palavras chaves; outros ...

TIPOS DE ERROS

- **Erros de semântica ou de tempo de execução:**
  - envolvem códigos tecnicamente corretos que contém problemas em seu significado,
  - os compiladores não detectam este tipo de erros,
  - são detectados ao executar o programa pois o programa aborta,
  - são fáceis de detectar, em ocasiões corrigi-los pode ser complicado.
  - Mais comuns: uso incorreto da memória; incompatibilidade nas instruções de entrada; acesso a recursos inexistentes, ...

TIPOS DE ERROS

- **Erros de lógica:**
  - envolvem códigos sintática e semanticamente corretos,
  - entretanto o código não executa da maneira desejada,
  - são os mas difíceis de diagnosticar e corrigir,
  - geralmente são provocados por uso incorreto de operadores (ex. igualdade == vs atribuição =), uso incorreto de recursos da linguagem ou erros na compreensão do problema.

Exemplo

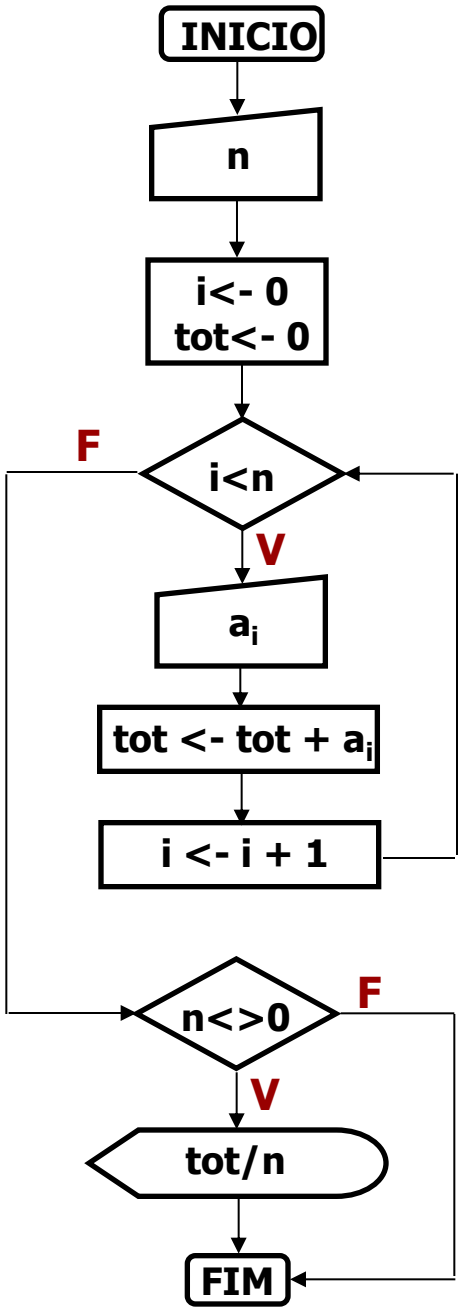
- O seguinte programa funciona?

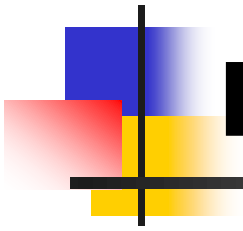
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a, b, c;
    a = 4;
    b = 6;
    c = a + b;
    printf ( "A media entre 4 e 6 é : %d \n", c);
}
```

- Porém, ele está correto?

Identifique os erros no código e classifique-os.

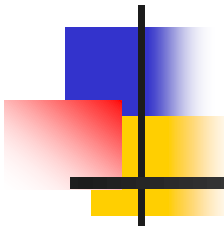
```
int main()
  int n, i, tot; a[10];

  printf("Digite n: ");
  scanf("%f", n);

  tot = 0;
  for(i=0;i<n;i++);{
    printf("Digite a_%d: ", i);
    scanf("%d", &a[i]);
    tot += a[i];
  }

  if n!=0
    printf("Resultado: %.2f\n", tot/n);

  system("PAUSE");
  return 0;
}
```





PROGRAMAÇÃO ESTRUTURADA

- Paradigma de programação que foi adotado como standar para o desenvolvimento de aplicações na década dos 70,
- Baseado em estruturas,
- Permite a modularização (abstração de processos),
- Permitiu criar sistemas mais grandes e complexos, humanizando a tarefa de manutenção de sistemas.



PROGRAMAÇÃO ESTRUTURADA

- Suportada em três estruturas básicas:
  - estrutura seqüencial
  - estrutura condicional (desvios condicionados)
  - estrutura de repetição
- **Elimina** o uso de desvios incondicionados (`goto`)
- Em 1966 Bohm e Jacopini demonstraram que qualquer problema pode ser resolvido com essas três estruturas sem qualquer instrução `goto`.

PROGRAMAÇÃO ESTRUTURADA

- **Estruturas básicas:**
  - Estrutura Seqüencial

```
Programa Média
Início
   leia A
   leia B
   C = (A + B) / 2
   escreva C
Fim
```

PROGRAMAÇÃO ESTRUTURADA

- **Estruturas básicas:**
  - Estrutura Condicional

```
Programa Aprovação
Início
   leia NOTA1
   leia NOTA2
   MEDIA = (NOTA1 + NOTA2)/2
   se (MEDIA >= 7) então
      escreva "ALUNO APROVADO"
   senão
      escreva "ALUNO REPROVADO"
   fim_se
Fim
```

PROGRAMAÇÃO ESTRUTURADA

- **Estruturas básicas:**
  - Estrutura de Repetição
  - Existem três tipos de repetição
    - incondicional
    - condicional com teste no inicio
    - condicional com teste no final
  - Também as podemos classificar como repetição controlada por contador e repetição controlada por sentinela.

PROGRAMAÇÃO ESTRUTURADA

- **Estruturas básicas:**
  - Repetição incondicional

```
Programa Conta
Início
   para CONTADOR de 1 até 5 faça:
      escreva CONTADOR
   fim_para
   escreva "Terminou"
Fim
```

PROGRAMAÇÃO ESTRUTURADA

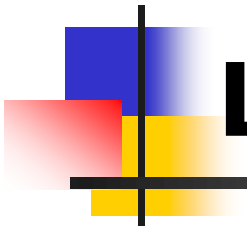
- **Estruturas básicas:**
  - Repetição condicional com teste no inicio

```
Programa Calcula_Dobro
Início
   leia NUM
   enquanto (NUM > 0) faça:
      DOBRO = 2*NUM
      escreva DOBRO
      leia NUM
   fim_enquanto
Fim
```

PROGRAMAÇÃO ESTRUTURADA

- **Estruturas básicas:**
  - Repetição condicional com teste no final

```
Programa Calcula_Dobro
Início
   leia NUM
   faça:
      DOBRO = 2 * NUM
      escreva DOBRO
      leia NUM
   enquanto (NUM > 0)
Fim
```



LINGUAGEM C - BÁSICO

- **Porque escolhemos a linguagem C para as disciplinas de LP1 e LP2?**
- **Vantagens ...**
- **Desvantagens ...**

LINGUAGEM C - BÁSICO

- **Um programa:**

```
#include <stdio.h>
#include <stdlib.h>
... outros includes....
... diretivas de pre-processamento ...

int main(void){

      ... declaração de variáveis...

      ... comandos e blocos...
}
```

LINGUAGEM C - BÁSICO

- Linhas terminam em ponto-e-vírgula,
- MAIÚSCULAS/minúsculas fazem a diferença,
- Existem 32 palavras reservadas ou comandos,
- Todo programa deve ter uma função `main()` onde começa a execução.

LINGUAGEM C - BÁSICO

- **Tipos de Dados Básicos**
  - `char`, representa o conjunto de caracteres ASCII
  - `int`, representa o conjunto dos números inteiros
  - `float`, representa o conjunto dos números reais (precisão simples)
  - `double`, representa o conjunto dos números reais (precisão dupla)
  - `void`, tipo de dados especial, representa vazio ou nulo, (funções sem retorno, ponteiros genéricos).

LINGUAGEM C - BÁSICO

- **Declaração de variáveis**

```
int A;
int B = 0;
float C, D;
char a;
```

- **Modificadores dos tipos de dados básicos**
  - unsigned
  - short
  - long

LINGUAGEM C - BÁSICO

- **Overflow?**
- **Underflow?**
- **Tipo Boleano:**
  - O tipo boleano é usado em expressões lógicas,
  - Tem significado verdadeiro ou falso,
  - Qualquer valor diferente de zero é verdadeiro,
  - O valor zero significa falso,
  - Padrão: 1-Verdadeiro, 0-Falso

LINGUAGEM C - BÁSICO

- **Vetores unidimensionais:**
  - Representa uma lista ou conjunto de elementos similares,
  - coleção de variáveis do mesmo tipo que é referenciado por um nome comum,
  - Exemplos:

```
int c[5];
float b[100];
char s[25];
```

  - Inicialização:

```
int c[5] = {1,2,3,4,5};
char s[4] = {'U','E','S','C'};
```

LINGUAGEM C - BÁSICO

- **Vetores bidimensionais:**
  - Representa uma tabela de elementos similares ordenados em linhas e colunas,
  - Exemplos:

```
float n[5][3];
int a[2][3];
char ch[10][5];
```

  - Inicialização:

```
int a[2][3] = {{1,2,3},{4,5,6}};
```

LINGUAGEM C - BÁSICO

- **Tipos de dados indexados ...**
  - Os subscritos em C variam entre 0 e N-1,
  - A linguagem C não fornece verificação automática de subscritos,
  - O estouro de memória poder provocar erros em tempo de execução ou resultados inesperados na execução do programa.

LINGUAGEM C - BÁSICO

- **Tipos de dados String**
  - A linguagem C não possui o tipo de dado String, apenas char (1 único caractere),
  - O C manipula cadeias de caracteres através de vetores unidimensionais de caracteres,
  - Toda String termina com o caractere especial '\0'
  - Declaração: `char nome[5];`
  - Inicialização: `char nome[5]={'U','E','S','C','\0'};`

LINGUAGEM C - BÁSICO

- **Tipos de dados String ...**
  - Aspas simples indicam caracteres,
  - Aspas duplas indicam strings,
  - ‘a’ vs “a”?
- Caracteres especiais:
  - `\n - nova linha`
  - `\” - aspas duplas`
  - `\’ - aspas simples`
  - `\\ - a própria barra`
  - `\b – backspace`
  - `\t - tabulação`

LINGUAGEM C - BÁSICO

- **Escopo de variáveis**
  - *variáveis locais*: são declaradas dentro de funções ou blocos, tem escopo local
  - *variáveis globais*: são declaradas foras de todas as funções, tem escopo global
  - *parâmetros formais*: são declaradas no cabeçalho da função, tem escopo local no bloco da função.

LINGUAGEM C - BÁSICO

- **Operadores**
  - Aritméticos:
    - + adição
    - - subtração
    - * multiplicação
    - / divisão
    - % resto
    - ++ incremento
    - -- decremento

LINGUAGEM C - BÁSICO

- **Operadores ...**
  - Lógicos:
    - > maior que
    - < menor que
    - >= maior ou igual que
    - <= menor ou igual que
    - == igualdade
    - != desigualdade
    - ! negação

LINGUAGEM C - BÁSICO

- **Operadores**
  - Relacionais:
    - &&, e
    - ||, ou
  - Atribuição
    - = de atribuição
  - Compostos
    - +=, -=, *=, /=, %=

LINGUAGEM C - BÁSICO

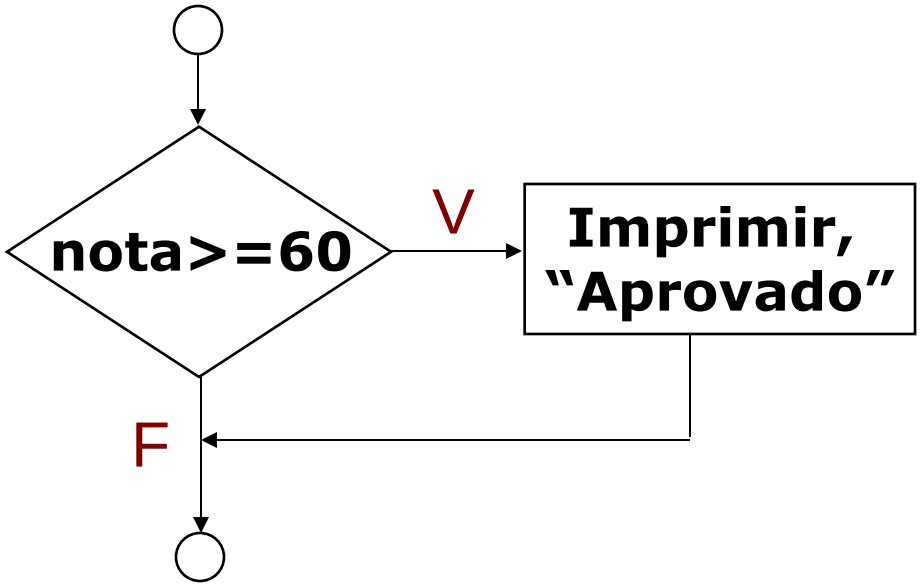
- **Estruturas de controle**
  - Em C existem 7 estruturas de controle:
  - A estrutura seqüencial
  - Três estruturas seleção (condicionais)
    - seleção simples
    - seleção dupla
    - seleção múltipla
  - Três estruturas de repetição
    - incondicional ou por contador
    - condicional com teste no inicio
    - condicional com teste no final

LINGUAGEM C - BÁSICO

- **Estrutura de seleção simples**
  - seleciona ou ignora um bloco de instruções

if(condição)
{instruções}

```
if (nota >= 60)
  printf("Aprovado\n");
```

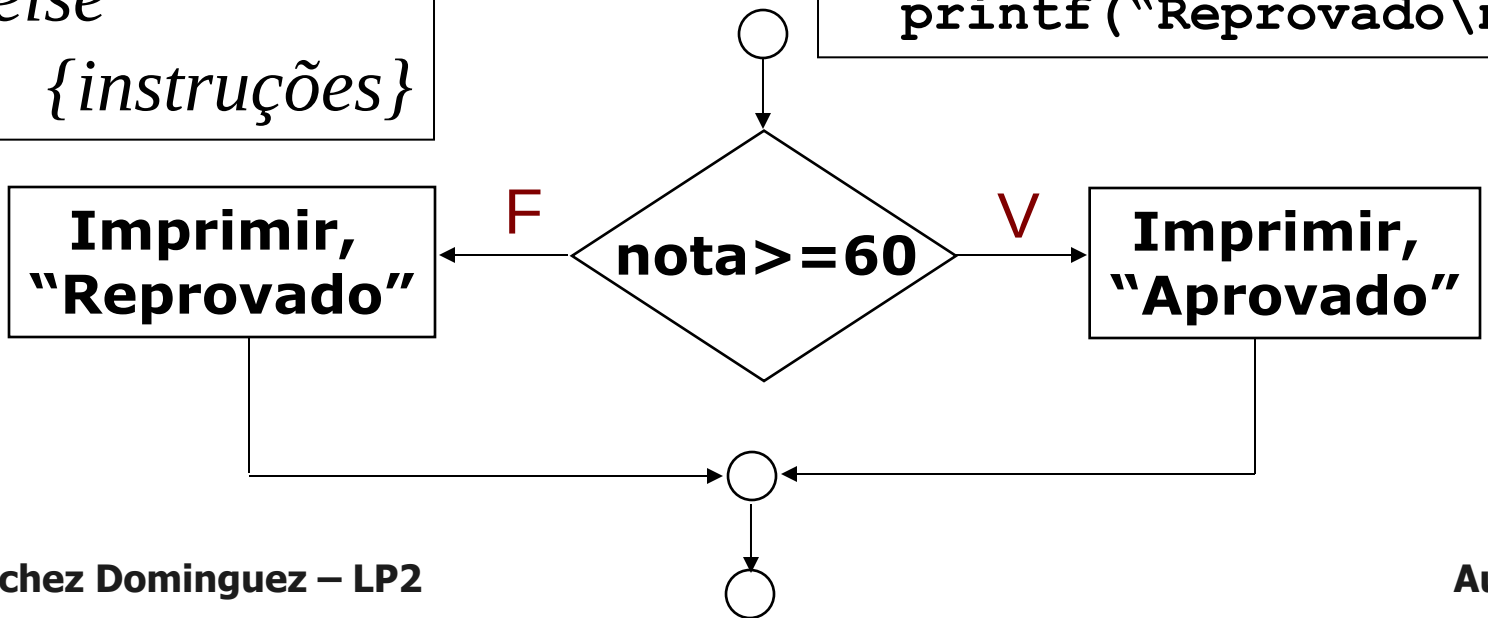


LINGUAGEM C - BÁSICO

- **Estrutura de seleção dupla**
  - seleciona entre dois blocos de instruções diferentes

```
if(condição)
  {instruções}
else
  {instruções}
```

```
if (nota >= 60)
  printf("Aprovado\n");
else
  printf("Reprovado\n");
```



LINGUAGEM C - BÁSICO

- **Estrutura de seleção múltipla**
  - seleciona entre vários blocos de instruções

```
switch(expressão){
  case constante1:
      seqüência de comandos
      break;
  case constante2:
      seqüência de comandos
      break;
 case constante3:
      seqüência de comandos
      break;
  .
  .
  .
  default:
       seqüência de comandos
}
```

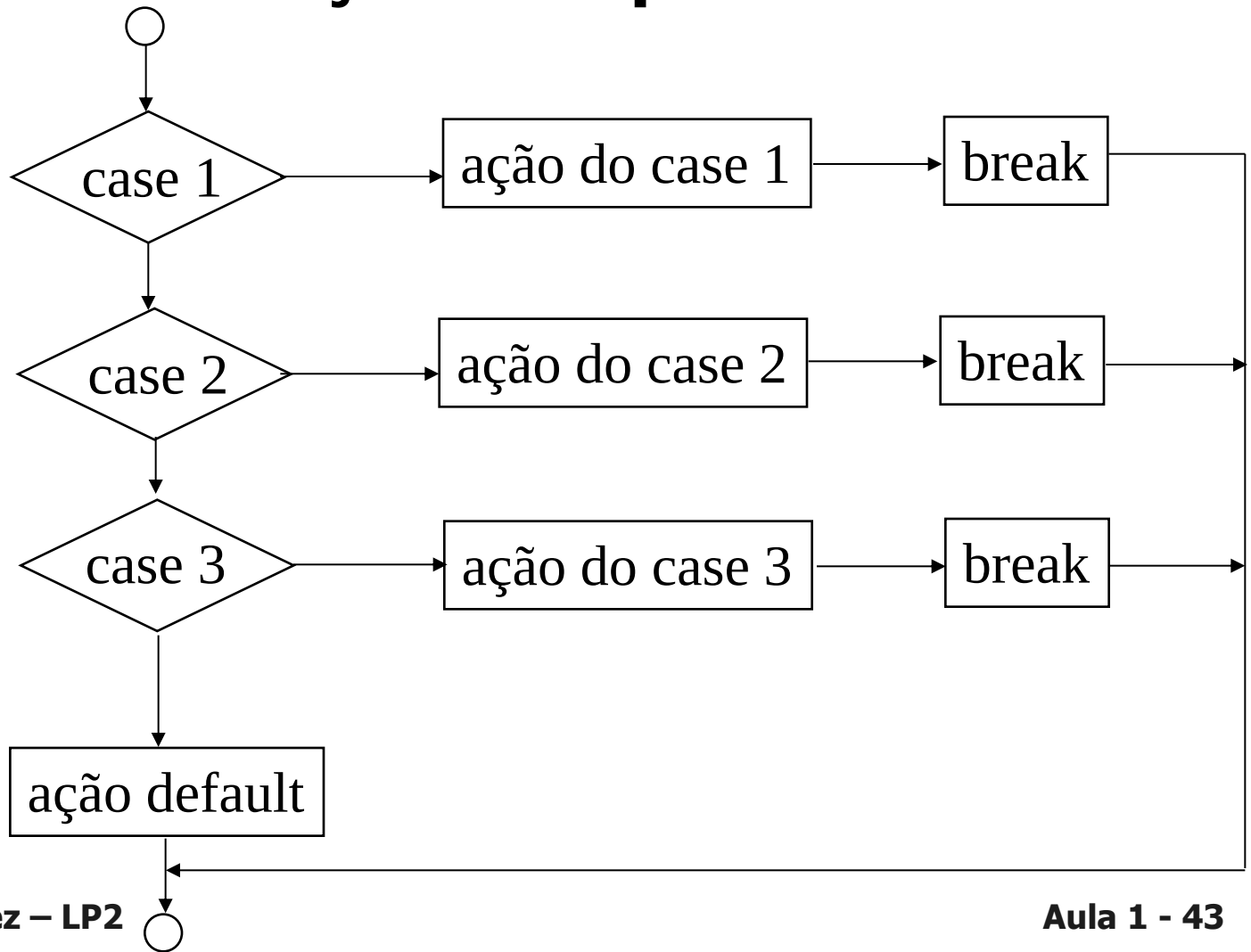
LINGUAGEM C - BÁSICO

- **Estrutura de seleção múltipla ...**

```
switch(nota){
  case 'A': case 'a':
    Ca++;
    break;
  case 'B': case 'b':
    Cb++;
    break;
  case 'C': case 'c':
    Cc++;
    break;
  case 'D': case 'd':
    Cd++;
    break;
  case 'E': case 'e':
    Ce++;
    break;
  default:
    printf("Nota incorreta.\n");
    break;
}
```

LINGUAGEM C - BÁSICO

- **Estrutura de seleção múltipla ...**

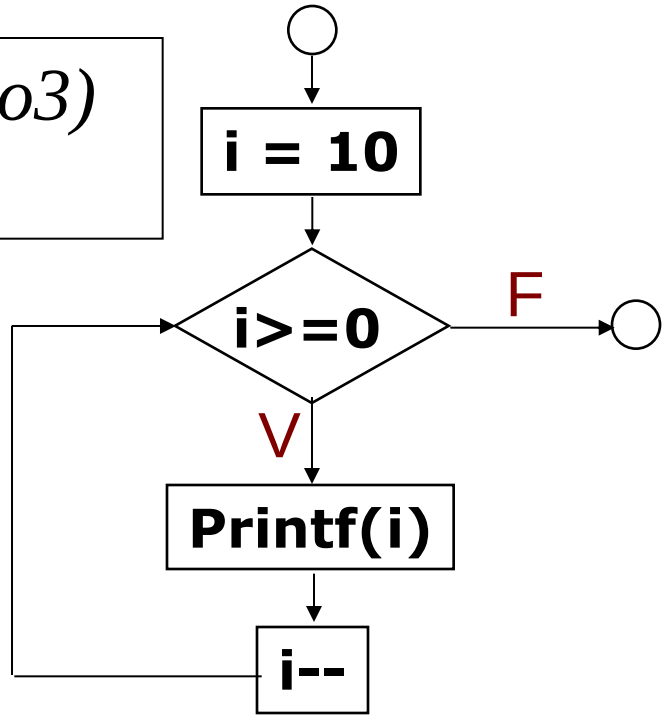


LINGUAGEM C - BÁSICO

- **Estrutura de repetição incondicional (por contador)**
  - permite ao programador repetir uma ação uma quantidade de vezes determinada

for (expressão1; expressão2; expressão3)
{bloco de instruções}

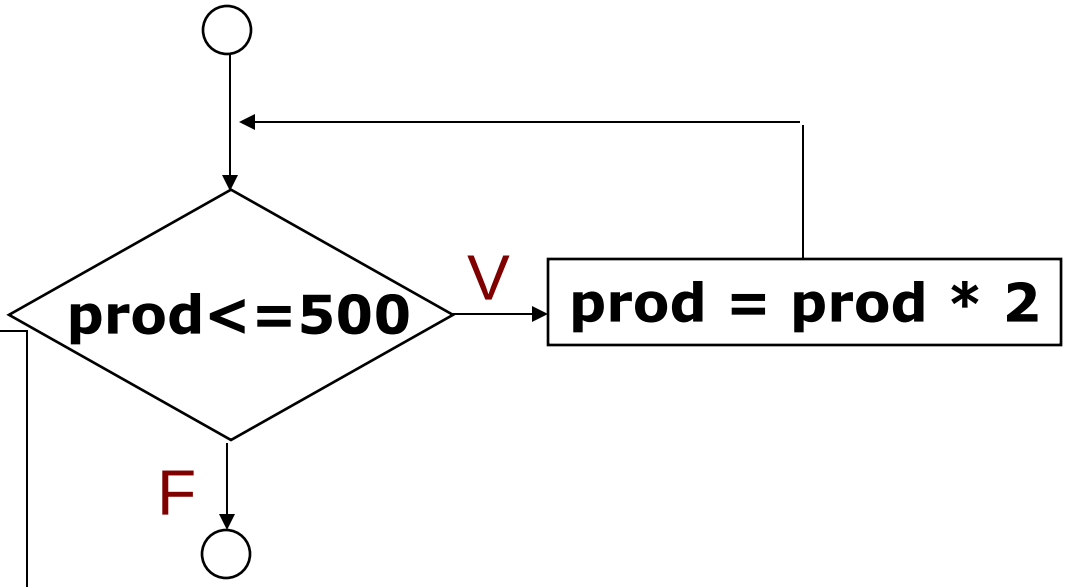
```
for(i=10; i>=0; i--)
  printf("%d\n", i);
```



LINGUAGEM C - BÁSICO

- **Estrutura de repetição com teste ao inicio**
  - permite ao programador especificar que uma ação deve ser repetida enquanto uma determinada condição for verdadeira

```
while (condição){
  bloco de instruções
  }
```



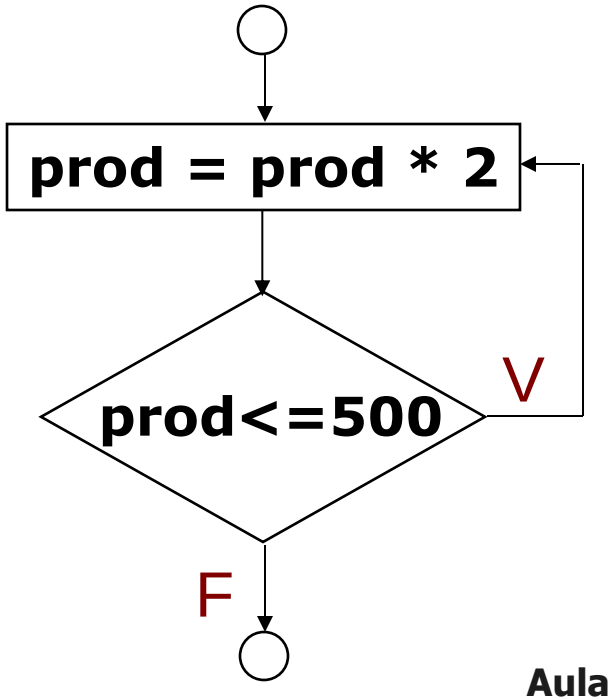

```
prod = 2;
while (prod <= 500){
   prod = prod * 2;
   }
```

LINGUAGEM C - BÁSICO

- **Estrutura de repetição com teste no final**
  - permite ao programador especificar que uma ação deve ser repetida enquanto uma determinada condição for verdadeira

```
do{
  bloco de instruções
}while(condição)
```

```
prod = 2;
do{
   prod = prod * 2;
   }while (prod <= 500)
```



LINGUAGEM C - BÁSICO

- **Comandos de desvio**
  - A linguagem C tem quatro comandos que realizam desvio incondicional: **`return`**, **`break`**, **`continue`** e **`exit`**.
  - **`return`**, é usado para retornar o valor de uma função.
  - **`break`**, força a terminação imediata do bloco.
  - **`continue`**, força a próxima iteração de um laço.
  - **`exit(-1)`**, é utilizada para terminar um programa.

LINGUAGEM C - BÁSICO

- **Operações de Entrada/Saída (E/S)**
  - `printf()`, envia informação a tela do computador.
  - `scanf()`, leitura de dados via teclado.

- Função `printf()`:

printf ("como vai exibir os dados", variáveis);

- %c caractere
- %d inteiros
- %ld inteiros longos
- %f float
- %lf double
- %s "string"

LINGUAGEM C - BÁSICO

- Função `printf()`...

```
int a=456;
float b =45.234;
char c='a';
printf("Inteiro %d Flutuante %f Caractere %c \n" , a, b, c );
```

Saída: **Inteiro 456 Flutuante 45.324 Caractere a**

```
char NOME[10] = "UESC";
printf ( "uma string: %s \n " , NOME );
```

Saída: **uma string: UESC**

```
float VALOR = 456.234;
printf ( " %5.1f \n " , VALOR );
```

Saída: **456.2**

LINGUAGEM C - BÁSICO

- Função `scanf()`:
  - similar ao printf(), porém usa o nome das variáveis (onde os dados serão lidos) com um e-comercial (&)

```
int OPCAO;
float NOTA1;
scanf ("%d", &OPCAO);
scanf ("%f", &NOTA1);
```

  - **Não é apropriado para ler strings (quebra nos espaços)**
  - Para leitura de strings utilize a função `gets();`

```
char NOME[50];
gets(NOME);
```

EXERCICIOS