REVISÃO DE C

DCE792 - AEDs II (Prática)

Atualizado em: 6 de agosto de 2024

Iago Carvalho

Departamento de Ciência da Computação

LINGUAGENS DE PROGRAMAÇÃO (LP)

Os programadores escrevem seus programas em várias LPs, algumas entendidas diretamente pelos computadores, outras requerendo passos intermediários de tradução.

No geral, LPs podem ser divididas em 3 classes

- Linguagens de máquina
- Linguagens assembly
- Linguagens de alto-nível
  - Imperativas
  - Orientadas a objetos
  - Funcionais
  - ...

PROCESSO DE DESENVOLVIMENTO EM C (C++)

O processo de desenvolvimento é composto por 6 fases

1. Criando um programa
2. Pré-processando o programa
3. Compilação
4. Ligação (*linking*)
5. Carga (*loading*)
6. Execução

CRIANDO UM PROGRAMA

Esta fase consiste da edição de um arquivo com um *programa editor* (um editor de textos qualquer).

- Bloco de notas
- Visual studio code
- NetBeans
- ...

Você digita um programa C/C++ (tipicamente conhecido como *programa fonte*) usando o editor, faz as correções necessárias e salva o programa em um dispositivo de memória secundária.

Frequentemente, os nomes de arquivos dos programas fonte C terminam com a extenção **.c** e de C++ com as extensões **.cpp** ou **.cc**

PRÉ-PROCESSANDO O PROGRAMA

Nesta fase ocorre o comando para *compilar* o programa desenvolvido

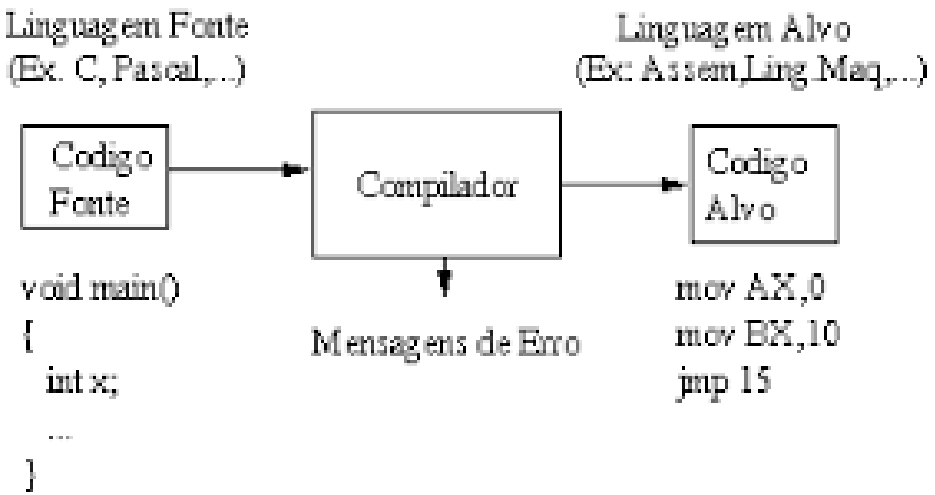
Em um sistema C/C++, um software *pré-processador* executado automaticamente antes que a fase de tradução do compilador inicie (então, chamaremos a fase 2 de pré-processamento e a fase 3 de compilação).

O *pré-processador* obedece a comandos chamados diretivas do pré-processador, que indicam que certas manipulações são realizadas no programa antes da compilação.

Estas manipulações usualmente incluem outros arquivos de texto para serem compilados, e realizam várias substituições de texto.

COMPILAÇÃO

Na fase 3, o compilador traduz o programa C/C++ (*código fonte*, em alto-nível) para um código de *linguagem de máquina* (código objeto, em baixo nível).



LIGAÇÃO

Tipicamente, um programa C/C++ contém referências para funções e dados definidos em outros lugares, tais como nas bibliotecas padrão ou nas bibliotecas privadas de um grupo de programadores trabalhando em um projeto particular.

O código objeto produzido pelo compilador C ou C++ contém, tipicamente, *buracos* por causa dessas partes ausentes. Um ligador (*linker*) liga o código objeto com o código das funções ausentes para produzir um programa executável (sem partes ausentes).

Se um programa é compilado e ligado corretamente, é produzida uma imagem executável.

CARGA

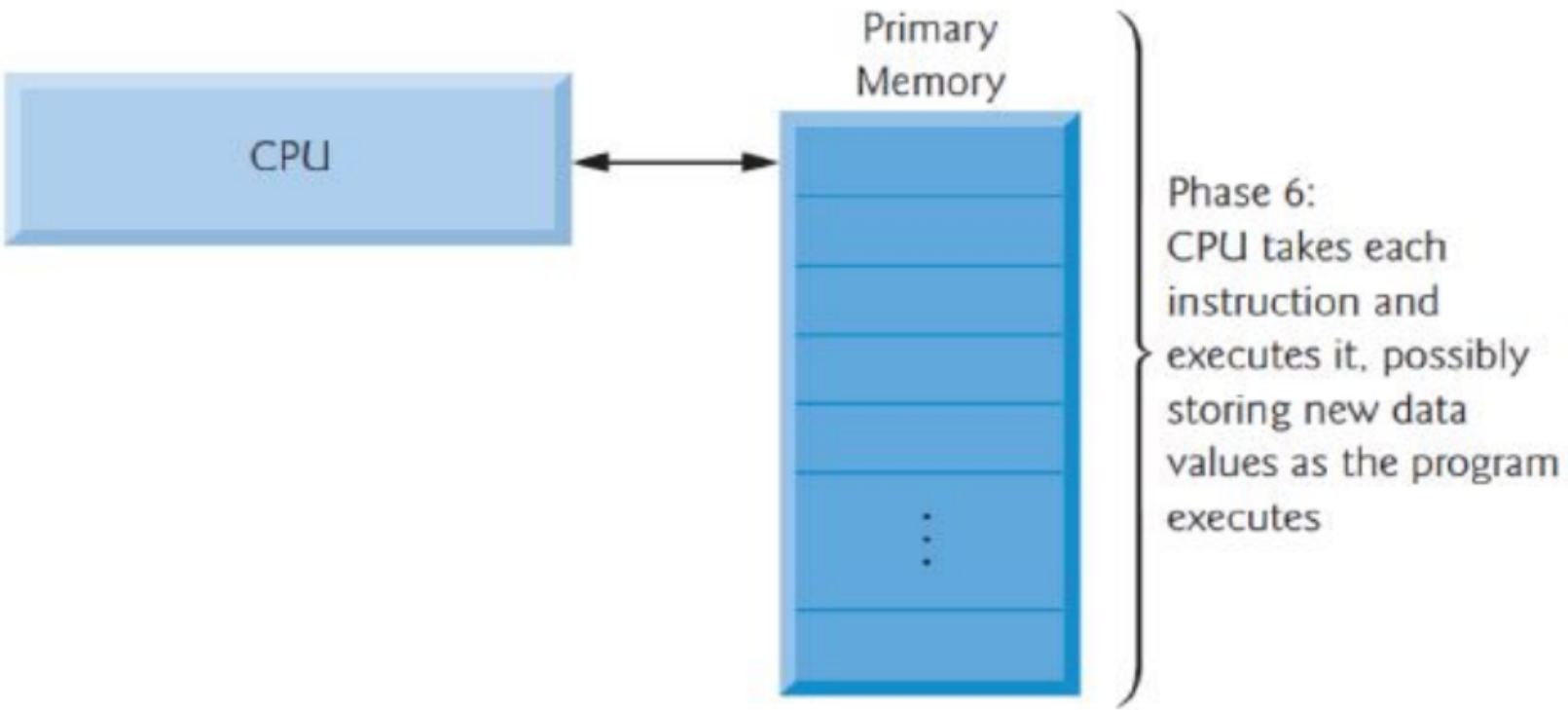
Antes de um programa ser executado, ele deve ser primeiramente colocado na memória (primária). Isto é feito pelo *carregador* (loader), que toma a imagem executável do disco e a transfere para a memória.

Os componentes adicionais das bibliotecas compartilhadas, que proveem suporte ao programa, também são carregados.

EXECUÇÃO

Finalmente, o computador, sob o controle de sua CPU, executa o programa, uma instrução por vez.

- A maioria das arquiteturas de computadores atuais podem executar várias instruções em paralelo.



VARIÁVEIS, OPERADORES, COMANDOS DE ENTRADA/SAÍDA

DECLARAÇÃO DE VARIÁVEIS

Toda variável tem tipo, nome, endereço de memória e valor.

Uma variável deve ser declarada com um identificador e um tipo de dado antes de ser usada no programa.

Se já existir um valor armazenado na variável e um novo valor for atribuído a esta variável, esse valor sobrescreve o valor anterior.

Exemplo: int number;

- O tipo int especifica que o valor armazenado é do tipo inteiro (valor inteiro).
- O identificador number é o nome da variável.

Pode-se declarar várias variáveis em uma mesma linha:
int number1, number2, number3, number4;

ENDEREÇOS

Ao declararmos uma variável x, ela será associada a

- Um nome
- Um endereço de memória ou referência
  - Por exemplo, *0xbfd267c4*
- Um valor

```
1  int x = 9;
```

Para acessar o endereço de uma variável, utilizamos **&**

OPERADORES ARITIMÉTICOS

Operação	Operador aritmético	Exemplo	Exemplo em C/C++
Adição	+	$f + 7$	`f + 7`
Subtração	-	$p - c$	`p - c`
Multiplicação	*	bm ou $b \times m$	`b * m`
Divisão	/	x/y ou $x \div y$ ou $\frac{x}{y}$	`x / y`
Módulo	%	$r \bmod s$	`r % s`

OPERADORES DE IGUALDADE E RELACIONAIS

Símbolo	Nome do Operador	Exemplo	Significado
>	Maior que	x > y	x é maior que y?
>=	Maior ou igual	x >= y	x é maior ou igual a y ?
<	Menor que	x < y	x é menor que y?
<=	Menor ou igual	x <= y	x é menor ou igual a y ?
==	Igualdade	x == y	x é igual a y?
!=	Diferente de	x != y	x é diferente de y?

OPERADORES LÓGICOS

Operador	Expressão	Nome	Descrição
!	!p	NÃO (negação)	!p é falso, se p é verd.; !p é verd., se p é falso.
&&	p && q	E (conjunção)	p && q é verdadeiro, se ambos, p e q são verd.; e falso, caso contrário.
||	p || q	OU (disjunção)	p || q é verdadeiro, se p, q ou ambos é verd.; e falso, caso contrário.

SAÍDA NA TELA

Utilizamos a função *printf*

- printf(formato valor/variável);

```
1  printf("%d", 10);
```

SAÍDA NA TELA

Alguns formatos possíveis para o *printf*

"%d": `int` (número inteiro)

"%ld": `long long` (número inteiro)

"%f": `float` (ponto flutuante)

"%lf": `double` (ponto flutuante)

"%c": `char` (caractere)

"%s": `string` (cadeia de caracteres)

CARACTERES ESPECIAIS NA SAÍDA

É possível utilizar alguns caracteres especiais.

Os mais comuns são

- \n: Quebra de linha
- \t: Tabulação (tab)
- \": Aspas duplas
- \': Aspas simples
- \\: Barras duplas

LEITURA DE DADOS

Pode-se ler do teclado utilizando *scanf*

- scanf(formato endereço de memória);

```
1  int x;
2  scanf("%d", &x);
```

LEITURA DE DADOS - ERROS

Porquê os comandos abaixo geram erros?

Erros comuns

```
1  int x;
2  scanf(x);
```

```
1  double valor = 10.0;
2  scanf(valor);
```

LEITURA DE DADOS - ERROS

Erros comuns

```
1  int x;
2  scanf(x);
```

```
1  double valor = 10.0;
2  scanf(valor);
```

scanf deve receber um texto/formato (entre aspas), não um int ou double (seja valor ou variável).

scanf deve receber um endereço de memória, e não um valor.

LEITURA DE DADOS - ERROS

E os códigos a seguir? Também geram erros?

Erros comuns

```c
int x;
scanf("%d", x);
```

```c
double valor = 10.0;
scanf("%lf", valor);
```

scanf deve receber um texto/formato (entre aspas), não um int ou double (seja valor ou variável).

scanf deve receber um endereço de memória, e não um valor.

FLUXOGRAMAS

FLUXOGRAMAS

Os fluxogramas são representações gráficas dos programas.

- São utilizados para nos ajudar a compreender um programa.
- Não estão associados a um linguagem específica.
- Apresentam a lógica do algoritmo e não as instruções da linguagem.

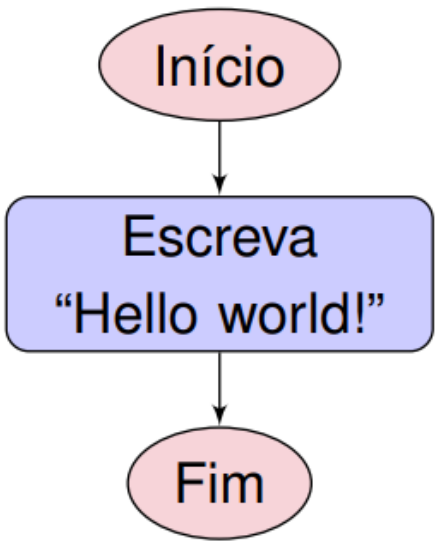
Utilizam diferentes tipos de blocos para indicar os comandos (entradas, saídas, processamentos, decisões, etc)

Além disso, utilizam setas para indicara sequência de execução.

FLUXOGRAMAS

Fluxograma de um código simples

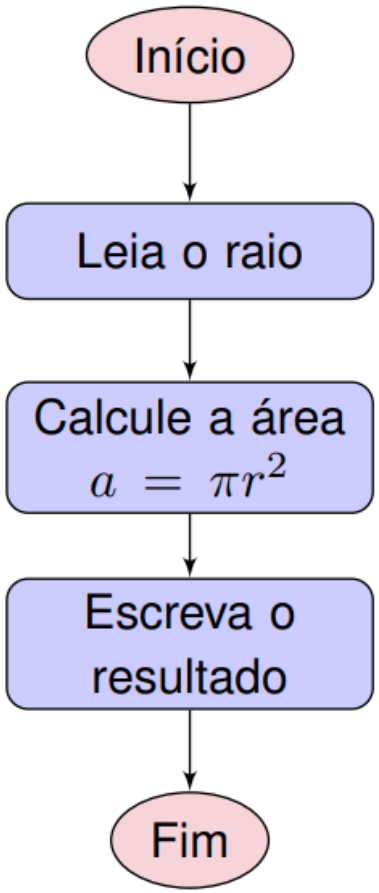
```
1  // Meu Primeiro Programa
2
3  #include <stdio.h>
4
5  int main()
6  {
7      // comentário explicativo
8      printf("Hello world!\n");
9      return 0;
10 }
```



FLUXOGRAMAS

Fluxograma de um código para calcular a área de um círculo

```c
/* Programa que calcula a área de um círculo
 */

#include <stdio.h>

int main()
{
    // declaração da constante Pi
    const double PI = 3.141592;
    double raio;

    printf("Digite o raio do círculo: ");
    scanf("%lf", &raio);

    // calculando e imprimindo a área
    double area = PI * raio * raio;
    printf("\nÁrea do círculo: %lf\n", area);

    return 0;
}
```



CONDICIONAIS (IF, IF-ELSE, SWITCH)

COMANDO *IF*

Operador condicional

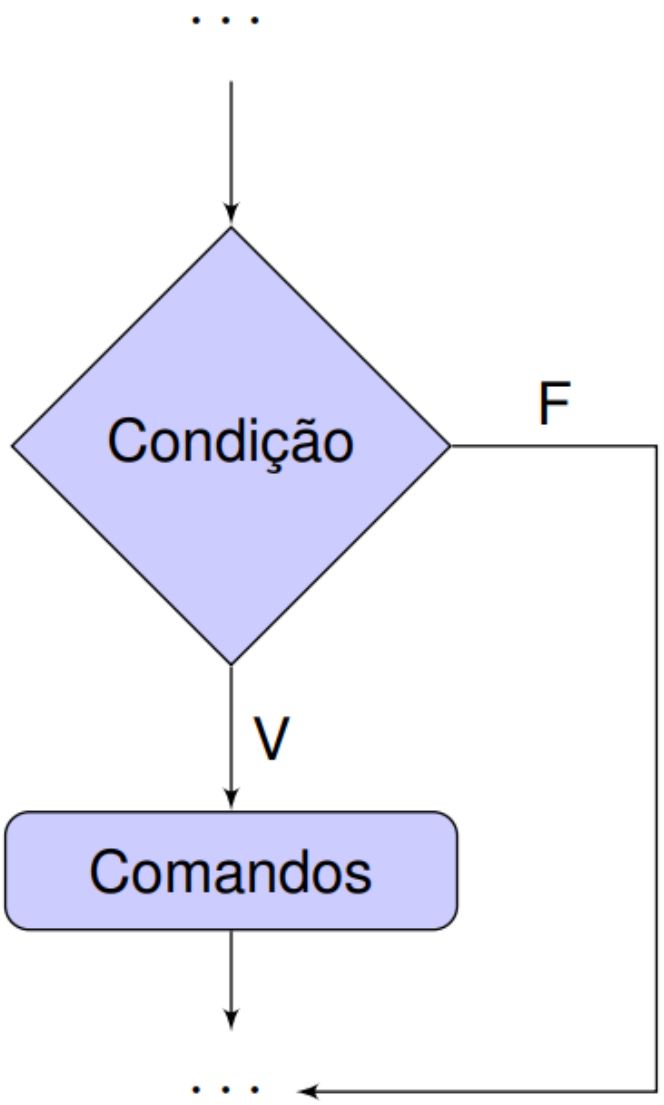
- Equivalente a *se*

```
1  if ( <expressão_de_teste> )
2      instrução_única;
```

ou

```
1  if ( <expressão_de_teste> )
2  {
3      instrução1;
4      instrução2;
5      instrução3;
6      ...
7  }
```

FLUXOGRAMA DO *IF*



COMANDO *IF*

```c
// Programa que verifica se um no. é par ou impar
#include <stdio.h>

int main()
{
    int numero; //variável para armazenar o número
    printf("Digite um numero inteiro: ");
    scanf("%d", &numero);

    // testa se o número é par
    if (numero % 2 == 0) {
        printf("\nO número %d é par.\n", numero);
    }

    // testa se o número é impar
    if (numero % 2 != 0) {
        printf("\nO número %d é impar\n.", numero);
    }

    return 0;
}
```

COMANDO *IF-ELSE*

Operador condicional

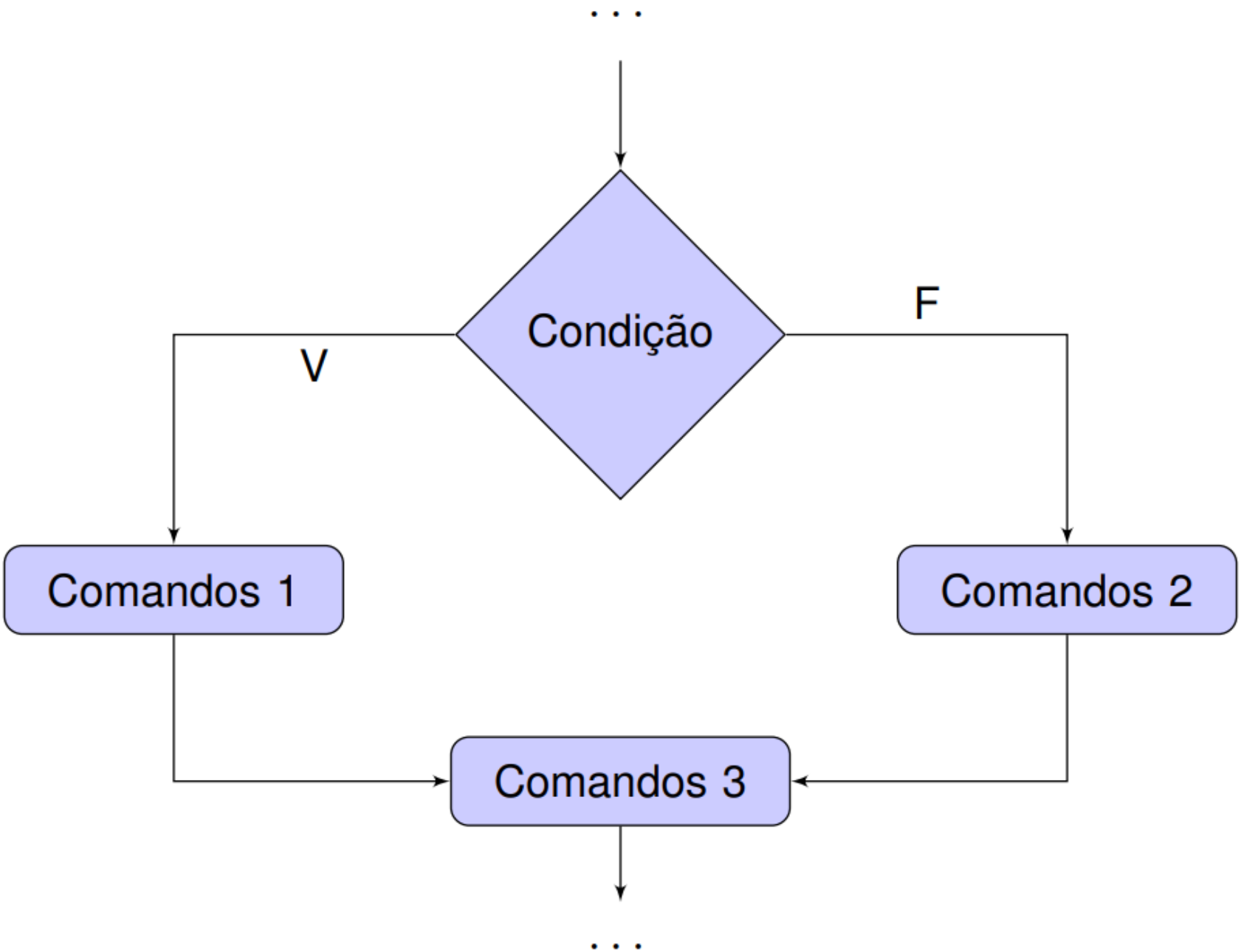
- Equivalente a *se-então*

```
1  if ( <expressão_de_teste> )
2     instrução_única_V;
3  else
4     instrução_única_F;
```

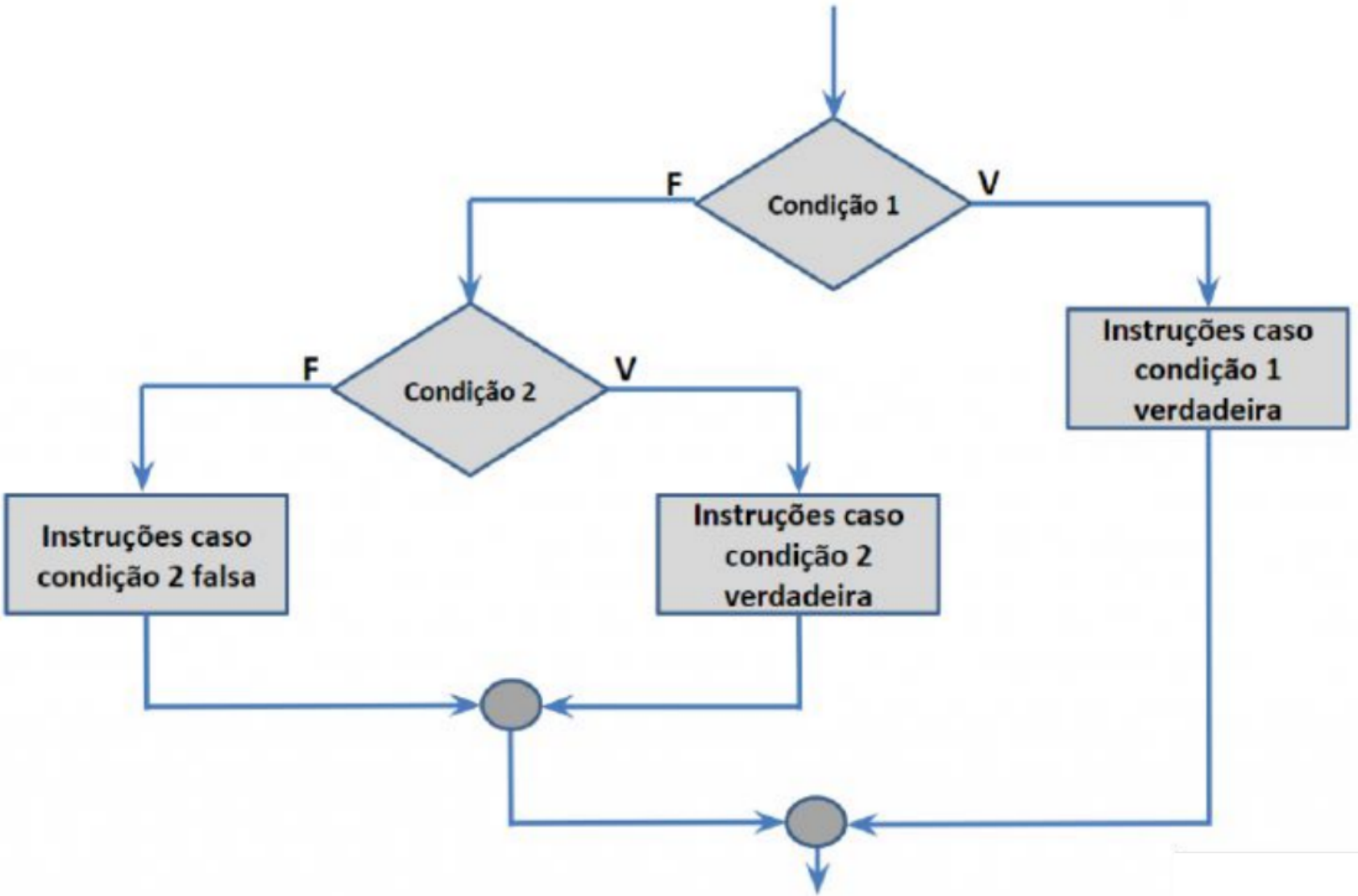
ou

```
1  if ( <expressão_de_teste> )
2  {
3     instrução_V1;
4       ...
5     instrução_Vn;
6  }
7  else
8  {
9     instrução_F1;
10      ...
11    instrução_Fn;
12 }
```

FLUXOGRAMA DO *IF-ELSE*



OUTRO FLUXOGRAMA DO *IF-ELSE*



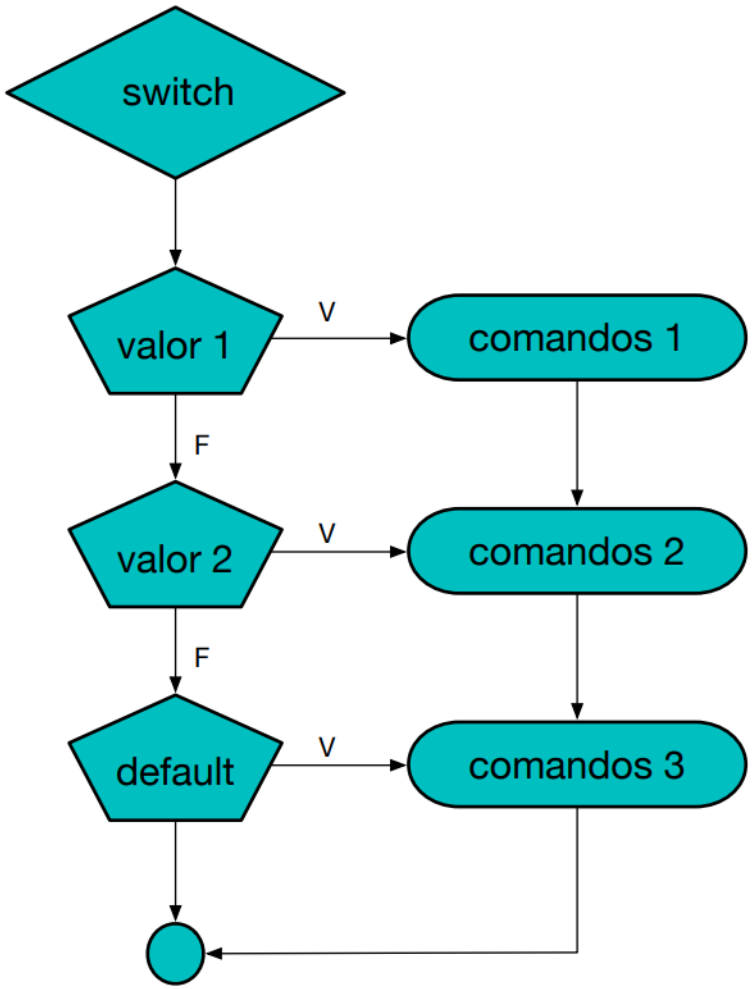
COMANDO *IF-ELSE*

```c
#include <stdio.h>

int main()
{
    int numero; //variável para armazenar o número

    printf("Digite um numero inteiro: ");
    scanf("%d", &numero);

    // se o número for par...
    if (numero % 2 == 0)
        printf("\nO número %d é par.\n", numero);

    // caso contrário
    else
        printf("\nO número %d é ímpar.\n", numero);

    return 0;
}
```

COMANDO *SWITCH*

```
1  switch (op) {
2      case valor1:
3          comandos1;
4          break;
5      case valor2:
6          comandos2;
7          ...
8      default:
9          comandosN;
10 }
```

FLUXOGRAMA DO COMANDO *SWITCH*



COMANDO *SWITCH*

```c
//Verifica se uma letra é vogal ou consoante

int main()
{
    char letra;
    printf("Digite uma letra: ");
    scanf("%d", &letra);

    switch (letra) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            printf("Vogal\n");
            break;
        default:
            printf("Consoante\n");
    }
    return 0;
}
```

OBSERVAÇÕES SOBRE O COMANDO *SWITCH*

1. O *switch* só permite comparar expressões com constantes.
2. Se precisarmos comparar com variáveis ou verificar faixas de valores, devemos usar o comando *if*.
3. Se não usarmos o comando *break* em cada case o programa continuará até o fim do bloco.

PRÓXIMA AULA:

FUNÇÕES
PONTEIROS
DIRETIVAS DE COMPILAÇÃO
BIBLIOTECAS C