3. Linguagem de Programação C

Prof. Renato Tinós

Departamento de Computação e Matemática
(FFCLRP/USP)

Principais Tópicos

3.4. Estruturas de Controle

3.4.1. Comandos

3.4.2. Estruturas de Decisão

- 3.4.2.1. Comando *if-else*
- 3.4.2.2. Comando *switch*

3.4.3. Estruturas de Repetição

- 3.4.3.1. Comando *while*
- 3.4.3.2. Comando *do-while*
- 3.4.3.3. Comando *for*

3.4.1. Comandos

- **Programas C são compostos por funções**
  - Funções são formadas por comandos

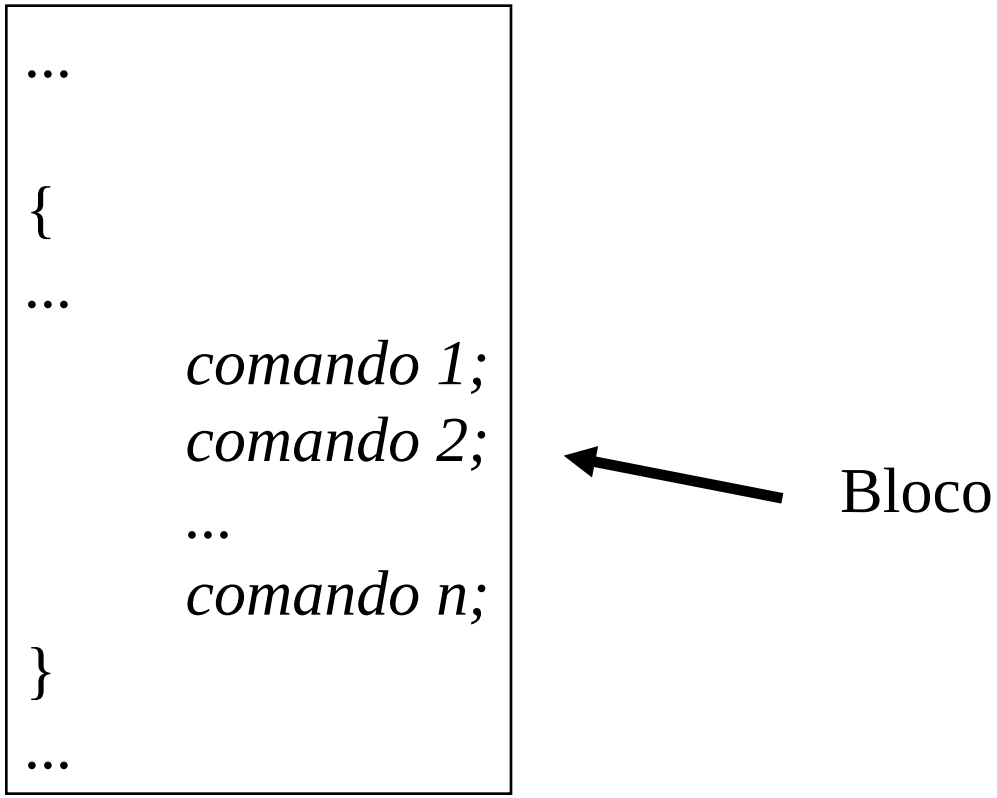
- **Comandos em C podem ser:**
  - Comandos simples
    - Realizam alguma ação
  - Comandos de controle
    - Afetam a maneira como outros comandos são executados

3.4.1. Comandos

- **Maioria dos comandos em C são comandos simples**
- **Expressão seguida de ponto e vírgula (expressão;)**
- **Pode envolver:**
  - Atribuição
  - Chamada de função
  - Incremento (decremento) de variável

3.4.1. Comandos

- **Comandos de controle podem ser aplicados a uma seqüência de comandos simples (bloco)**



3.4.1. Comandos

- **Bloco de comandos**
  - Tratado por comandos de controle como se fossem um único comando
  - Também são chamados de comandos compostos
    - C permite a declaração de variáveis antes dos comandos de qualquer bloco
    - Tabulação interior de um bloco faz parte de um bom estilo de programação

3.4.1. Comandos

- **Estruturas de Controle**
  - São comandos que afetam a maneira como outros comandos são executados
    - Podem ser:
      - De decisão
        - » Condicionam a execução de um comando ao valor de uma expressão
      - De repetição
        - » Permite a execução de um comando repetidas vezes

3.4.2. Estruturas de Decisão

- **São estruturas de controle condicionais**
  - Muitas vezes, a avaliação de uma condição define a execução ou não de um comando (execução condicional)
  - A maneira mais fácil de fazer isso é através do comando *If*

```
if (condicao)
        comando;
```

```
if (condicao)
        comando1 ;
else
        comando2;
```

  - Comandos podem ser simples ou compostos (bloco)

3.4.2.1. Comando *if-else*

```
/* Programa: Comandos em bloco*/
# include <stdio.h>

main ( ) {
        int num;

        printf (“Este programa diz se um numero eh par ou impar \n \n”);
        printf (“Entre com um numero inteiro: “);
        scanf(“%d”, &num)
        if (num % 2 == 0){
                printf (“O número %d eh par ”, num);
                printf (“porque o resto de n / 2  é igual a zero \n”);
        }
        else{
              printf (“O número %d eh impar ”, num);
              printf (“porque o resto de n / 2  é diferente de zero \n”);
        }
}
```

3.4.2.1. Comando *if-else*

- **Operadores lógicos podem ser utilizados na equação de condição**
  - Exemplos:

```
...
if ( (a>5) && (b==-1) ) {
    ...
```

```
...
if ( (a>1) || !(b==2) ) {
    ...
```

3.4.2.1. Comando *if-else*

- **Comando *if***
  - Algumas aplicações precisam de estruturas de decisão mais complicadas
    - Por exemplo, quando existem mais que duas possibilidades
    - Uma alternativa é utilizar uma cadeia de comandos *if*

```
if (condicao1)
        comando1;
else if (condicao2)
        comando2;
else if (condicao3)
        comando3;
...
```

3.4.2.1. Comando *if-else*

```
...
int pontosTeste;
char nota;
...
if (pontosTeste >= 90)
      nota = 'A';
else if (pontosTeste >= 80)
       nota = 'B';
else if (pontosTeste >= 70)
      nota = 'C';
else if (pontosTeste >= 60)
      nota = 'D';
else
      nota = 'F';
...
```

- **Apenas um *else***
- **Se um *else if* for verdadeiro, os comandos *else if* e *else* abaixo dele não são avaliados**
- **Outra alternativa: utilizar o comando *switch***

3.4.2.1. Comando *if-else*

Exercício3.4.1. As raízes de uma equação quadrática da forma $ax^2+bx+c=0$ são reais se e somente se o discriminante dado por b^2-4ac for maior ou igual a zero. Escreva um programa em C que leia os valores dos coeficientes *a, b* e *c* e imprime a raiz quadrada do valor do discriminante se ele for maior ou igual a zero (dica: use a função *sqrt()* da biblioteca *math.h*).

Exercício3.4.2. Desenvolva um programa que calcule a média final (m_f) de um aluno quando a nota da prova (p) e do trabalho (t) é inserida pelo usuário (ambas as notas estão no intervalo [0,10]). A média final é dada por:

$$m_f = \begin{cases} 0{,}8p + 0{,}2t & se\ p \geq 5\ e\ t \geq 5 \\ p & se\ p < 5 \\ t & se\ t < 5\ e\ p \geq 5 \end{cases}$$

3.4.2.2. Comando *switch*

- **Executa comandos condicionalmente**
  - Baseado no valor de alguma expressão

```
switch (expressao) {
    case constante-1: comando; break;
    case constante-2: comando; break;
    . . .
    case constante-n: comando; break;
    default: comando; break;
}
```

3.4.2.2. Comando *switch*

```
. . .
int mes;
. . .
switch (mes) {
    case 1:  printf("January"); break;
    case 2: printf("February"); break;
    case 3: printf("March"); break;
    case 4: printf("April"); break;
    case 5: printf("May"); break;
    case 6: printf("June"); break;
    case 7: printf("July"); break;
    case 8: printf("August"); break;
    case 9: printf("September"); break;
    case 10: printf("October"); break;
    case 11: printf("November"); break;
    case 12: printf("December"); break;
}
. . .
```

3.4.2.2. Comando *switch*

- **Equivalente a um comando *if-else***

```
. . .
int mes;
. . .
if (mes == 1) {
     printf("January");
} else if (mes == 2) {
    printf("February");
  . . .
```

3.4.2.2. Comando *switch*

- **Qual comando usar? *if-else* ou *switch*?**
  - Decisão do programador
    - Baseada em facilidade de leitura e outros fatores
- **Cada comando *case* deve ser único**
- **Valor fornecido para cada comando *case***
  - Deve ser do mesmo tipo do valor retornado pela expressão do comando *switch*
  - Deve ser do tipo *char* ou *int*

3.4.2.2. Comando *switch*

- **Observações**
  - Comando *break*
    - Faz com que o programa saia de dentro do comando *switch* e continue no próximo comando
    - Sem ele, todos os comandos *case* seriam executados
  - Em algumas situações, pode ser interessante executar mais de um comando *case*

3.4.2.2. Comando *switch*

```
 . . .
int mes, ano;
int numDias;
 . . .
switch (mes) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        numDias = 31;
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        numDias = 30;
        break;
    case 2:
      if ( ((ano % 4 == 0) && !(ano % 100 == 0))
            || (ano % 400 == 0) )
          numDias = 29;
       else
          numDias = 28;
       break;
}
 . . .
```

3.4.2.2. Comando *switch*

- **Comando *default* pode ser usado para lidar com os valores que não foram definidos explicitamente nos comandos *case***

- **Cláusula *default* é selecionada quando nenhum dos rótulos casa com o valor da expressão**
  - É opcional (mas é uma boa prática de programação)
    - *default*:
      - Pode incluir uma mensagem de erro
        - » Ex.: “valor de rótulo não esperado”

3.4.2.2. Comando *switch*

```
. . .
int mes;
. . .
switch (mes) {
    case 1:  printf("January"); break;
    case 2:  printf("February"); break;
    case 3:  printf("March"); break;
    case 4:  printf("April"); break;
    case 5:  printf("May"); break;
    case 6:  printf("June"); break;
    case 7:  printf("July"); break;
    case 8:  printf("August"); break;
    case 9:  printf("September"); break;
    case 10: printf("October"); break;
    case 11: printf("November"); break;
    case 12: printf("December"); break;
   default: printf(”Opa, este nao eh um mes valido!");
}
. . .
```

3.4.3. Comandos de Repetição

- **Permitem a execução de partes de um programa mais de uma vez**
- **São comandos iterativos da linguagem C**
  - Comando *while*
  - Comando *do while*
  - Comando *for*

3.4.3.1. Comando while

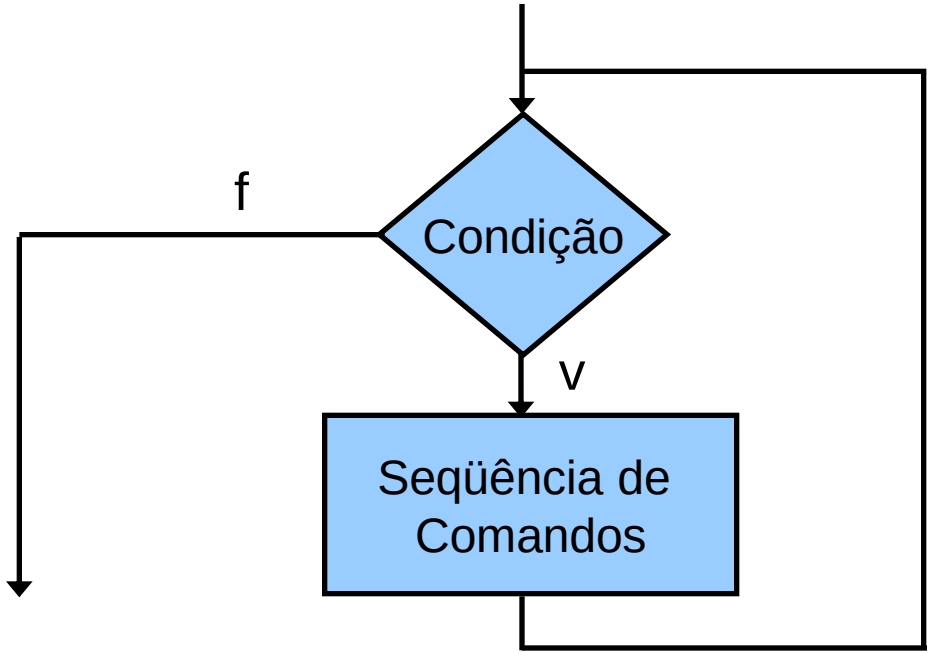
- **Estrutura de repetição mais simples**
- **Executa um comando repetidamente até uma condição se tornar falsa**

```
while (expressão condicional)
        comando;
```

➢ Comandos podem ser simples ou compostos (bloco)

3.4.3.1. Comando while

- **Teste condicional é executado antes de cada ciclo do *loop***
  - Se o primeiro teste resultar no valor 0 (Falso), o corpo do loop não é executado



3.4.3.1. Comando while

Exercício 3.4.3. O que o programa abaixo imprime na tela?

```
/* Programa: Comando while*/
# include <stdio.h>

main() {
    int i=0, a=0, n=5;

    while ( i<=n )
    {
        a+=i;
        i=i+1;
    }
    printf ("a=%d \n",a);
}
```

Exercício 3.4.4. Faça o fluxograma do programa acima?

3.4.3.1. Comando while

- **Utilizado onde existe uma condição de teste que possa ser aplicada antes da execução do corpo**

- **Vários problemas de programação não se encaixam na estrutura do comando *while***
  - As vezes o teste seria mais natural em algum lugar no meio do loop
    - Ex.: leitura de dados do usuário até o recebimento de um valor especial (sentinela)

3.4.3.1. Comando while

- ***Loop* baseado em sentinela**
  - Ler um valor
  - Se o valor é igual ao sentinela, sair do *loop*
  - Senão, executar o processamento requerido por este valor

– Como sair do *loop* quando um dado valor for lido?
  - Problema do *loop* e meio
    – Estratégia 1: Por um comando *break* após a leitura
    – Estratégia 2: Copiar parte do código para fora do loop

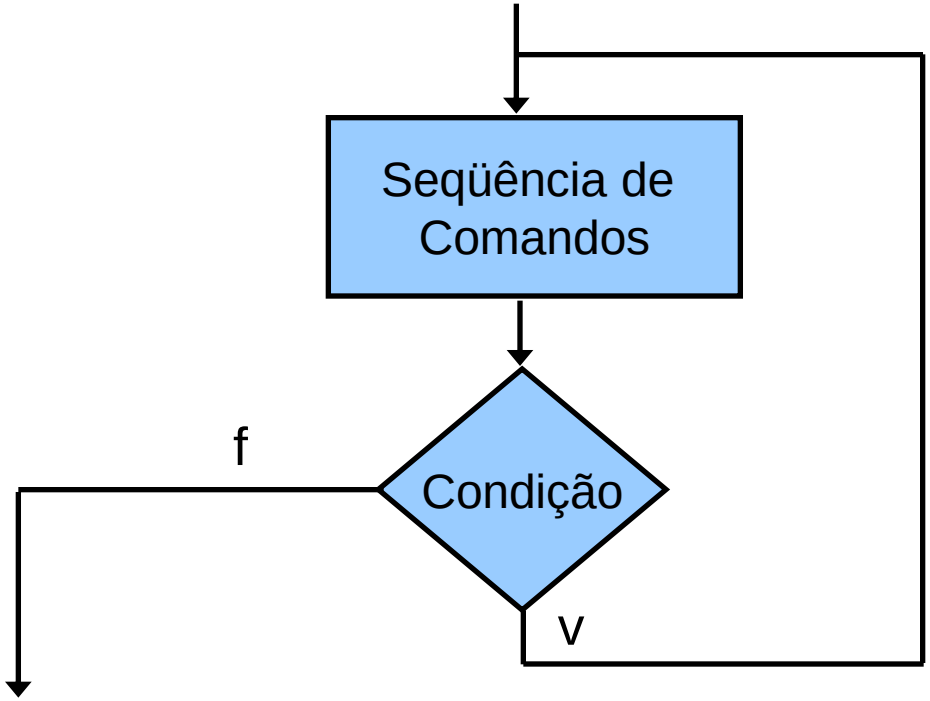
3.4.3.2. Comando do-while

- **Executa um comando repetidamente até uma condição se tornar falsa**

```
do {
    comandos;
} while (expressao condicional);
```

- **Semelhante ao comando *while***
  - Exceto que a expressão é avaliada no final

3.4.3.2. Comando do-while



3.4.3.2. Comando do-while

- **Exemplo**

```
. . .
int c=1;
. . .
do {
            . . .
            scanf (“%d”, &c);
} while (c != -1);
. . .
```

3.4.3.3. Comando for

- **O comando *for* é um comando de repetição determinado pelas expressões:**
  - inicial
  - teste
  - passo

```
for (inicial; teste; passo) {
        comandos;
}
```

3.4.3.3. Comando for

- **Inicial**
  - Expressão que indica como o *loop* do comando *for* deve ser inicializado
  - Executado uma única vez
    - No início do *loop*
  - Define o valor inicial da variável de indexação (contador)
    - Ex.: for (i = 0; .... for (i = -7; ...

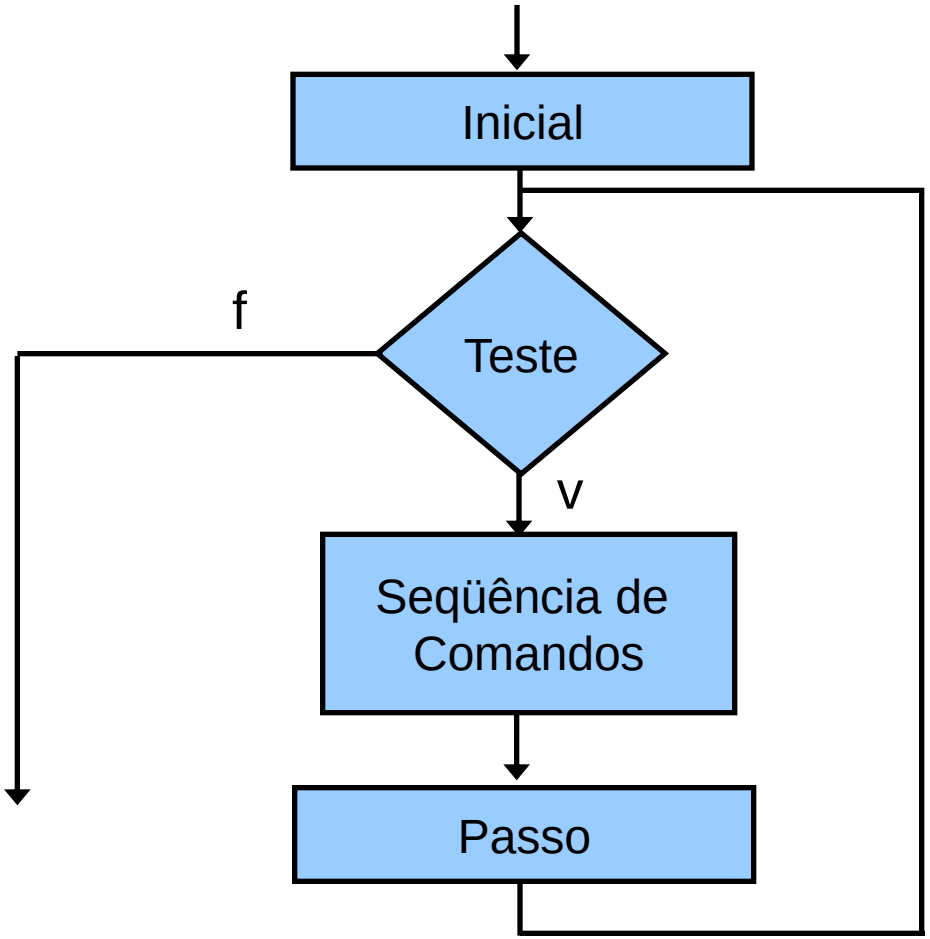
3.4.3.3. Comando for

- **Teste**
  - Expressão que indica quando o *loop* do comando *for* deve continuar
  - Funciona como a condição de teste do *while*
  - Expressão é avaliada no topo de cada interação do *loop*
    - Expressão Booleana com resultados VERDADEIRO ou FALSO
      - Quando o resultado da avaliação é *FALSO*, o *loop* termina
      - Enquanto teste é VERDADEIRO, o *loop* continua
    - Ex.: for (i = 0; i < n ; i++){

3.4.3.3. Comando for

- **Passo**
  - Expressão chamada a cada interação do *loop* para atualizar o valor do contador
  - Indica quanto muda o valor da variável de indexação de um ciclo para outro
  - Especificações de passo mais comuns
    - *index++, index--, index +=n*

3.4.3.3. Comando for



3.4.3.3. Comando for

```
for (inicial; teste; passo)
{
        comandos;
}
```

é equivalente a:

```
inicial;
while (teste) {
        comandos;
        passo;
}
```

3.4.3.3. Comando for

Exercício 3.4.5. O que o programa abaixo imprime na tela?

```
/* Programa: Comando for*/
# include <stdio.h>


main ( ) {
        int t;
        for (t = 5; t >= 0; t --) {
                printf ("%d \n", t);
        }
        printf ("fim! \n");
}
```

Exercício 3.4.6. Faça o fluxograma do programa acima?

3.4.3.3. Comando for

- **Expressões inicial, teste e passo são opcionais**
  - Os separadores (;) devem aparecer
  - Falta de valor inicial ⇒ não é feita nenhuma inicialização do indexador
  - Falta de condição de teste ⇒ ela é assumida como sempre VERDADEIRA
  - Falta de passo ⇒ indexador não é alterado entre ciclos do loop
    - Mas pode ser alterado dentro do ciclo

Exercícios

Exercício 3.4.7. Reescreva os trechos de programas abaixo usando o comando for

```
...
x = 14;
while ( x >= 3){
   printf (“%d\n”, x);
   x - = 5;
}
...
```

```
...
y = 70;
while ( y <= 90){
   printf (“%d\n”, y);
   y += 5;
}
...
```

Exercícios

Exercício 3.4.8. Escreva um programa em C que leia os valores de *n* números reais e imprima os valores da soma e da média aritmética destes números.

Exercício 3.4.9. Escreva um programa em C que imprima os *L* primeiros termos da seqüência de Fibonacci.