Estruturas de Dados I

1-Conceitos Fundamentais

Lidio Mauro Lima de Campos
lidio@ufpa.br

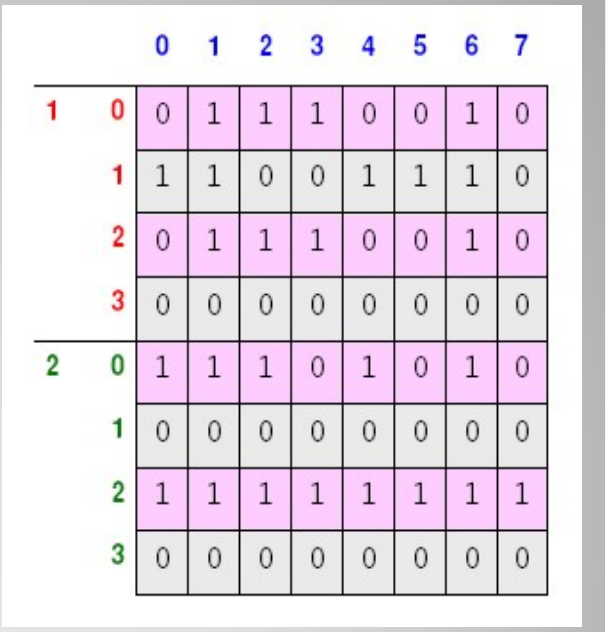
Universidade Federal do Pará – UFPA
Faculdade de Computação

Referências

- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados, Editora Campus* (2004)
- Kernigan, B.W. Ritchie, D.M. A Linguagem de Programação C. Campus.
- Ascencio, Ana Fernades G., de Araújo, Graziela Santos. Estruturas de Dados. Prentice Hall, 2010.

1-Conceitos Fundamentais

- **Bits, Bytes e Palavras**
  - Organização da memória
    - Bit: menor unidade
    - armazena 0 ou 1
  - Byte:
    - seqüência de 8 bits
  - Palavra:
    - seqüência de bytes
    - número de bytes da palavra
  - varia conforme a arquitetura do computador



		0	1	2	3	4	5	6	7
1	0	0	1	1	1	0	0	1	0
	1	1	1	0	0	1	1	1	0
	2	0	1	1	1	0	0	1	0
	3	0	0	0	0	0	0	0	0
2	0	1	1	1	0	1	0	1	0
	1	0	0	0	0	0	0	0	0
	2	1	1	1	1	1	1	1	1
	3	0	0	0	0	0	0	0	0

Expressões

- Em C, uma **expressão** é uma combinação de variáveis, constantes e operadores, que pode ser avaliada computacionalmente.

- O Resultado é chamado **valor da expressão**.

Expressões

- **Variável** – representa um espaço de memória do computador para armazenar um determinado tipo de dado.
  - Em C, todas as variáveis precisam ser explicitamente declaradas (Tipo nome).
  - int a; float b;

Tipos Básicos

Tipo	Tamanho	Menor valor	Maior valor
char	1 byte	-128	+127
unsigned char	1 byte	0	+255
short int (short)	2 bytes	-32.768	+32.767
unsigned short int	2 bytes	0	+65.535
int (*)	4 bytes	-2.147.483.648	+2.147.483.647
long int (long)	4 bytes	-2.147.483.648	+2.147.483.647
unsigned long int	4 bytes	0	+4.294.967.295
float	4 bytes	-10^{38}	$+10^{38}$
double	8 bytes	-10^{308}	$+10^{308}$

(*) depende da máquina, sendo 4 bytes para arquiteturas de 32 bits

Variáveis e Constantes

- **Valor Constante**: armazenado na memória possui um tipo, indicado pela sintaxe da constante.

```
123       /* constante inteira do tipo "int" */
12.45     /* constante real do tipo "double" */
1245e-2   /* constante real do tipo "double" */
12.45F    /* constante real do tipo "float" */
```

```
#define ICMS 0.18
#define MAX 100
#define ERRO "Erro!!!"
```

Declaração de Variável

- variáveis devem ser explicitamente declaradas

```
int a;      /* declara uma variável do tipo int */
int b;      /* declara uma variável do tipo int */
float c;    /* declara uma variável do tipo float */

int d, e;   /* declara duas variáveis do tipo int */
```

Declaração de Variável

- variáveis só armazenam valores do mesmo tipo com que foram declaradas.

```
int a;      /* declara uma variável do tipo int */
a = 4.3;    /* a armazenará o valor 4 */
```

```
int a = 5, b = 10;     /* declara e inicializa duas variáveis do tipo int */
float c = 5.3;         /* declara e inicializa uma variável do tipo float */
```

Declaração de Variável

- Uma variável deve ter valor definido quando utilizada.

```
int a, b, c;          /* declara e inicializa duas variáveis do tipo int */
a = 2;
c = a + b;            /* ERRO: b contém "lixo" */
```

Operadores e Expressões

- **Operadores**
  - Aritméticos, Atribuição , Incremento e decremento
  - Relacionais e lógicos , Outros
- **Operadores aritméticos (+ , - , * , / , %):**
  - operações são feitas na precisão dos operandos.
  - o operando com tipo de menor expressividade é convertido para o tipo do operando com tipo de maior expressividade.
  - divisão entre inteiros trunca a parte fracionária.

```
int a
double b, c;
a = 3.5;          /* a recebe o valor 3 */
b = a / 2.0;      /* b recebe o valor 1.5 */
c = 1/3 + b;      /* 1/3 retorna 0 pois a operação será sobre inteiros */
                  /* c recebe o valor de b */
```

Operadores e Expressões

- **Operadores aritméticos (cont.):**
  - o operador módulo, **"%"**, aplica-se a inteiros

  precedência dos operadores: *** , / , - , +**

```
x % 2          /* o resultado será 0, se x for par; caso contrário, será 1 */

a + b * c / d  é equivalente a  (a + ((b * c) / d))
```

Operadores e Expressões

- **Operadores de atribuição**

(= , += , -= , *= , /= , %=)

- C trata uma atribuição como uma expressão
- C oferece uma notação compacta para atribuições em que a mesma variável aparece dos dois lados
  - var *op= expr é equivalente a var = var op* ***(expr)***

```
i += 2;          é equivalente a   i = i + 2;
x *= y + 1;      é equivalente a   x = x * (y + 1);
```

Operadores e Expressões

- **Operadores de incremento e decremento (++ , --)**
  - incrementa ou decrementa de uma unidade o valor de uma variável
  - os operadores não se aplicam a expressões
  - o incremento pode ser antes ou depois da variável ser utilizada
  - n++ incrementa n de uma unidade, depois de ser usado
  - ++n incrementa n de uma unidade, antes de ser usado

Operadores e Expressões

- **Operadores de incremento e decremento (++ , --)**

```
n = 5;
x = n++;          /* x recebe 5; n é incrementada para 6 */
x = ++n;          /* n é incrementada para 6; x recebe 6 */
a = 3;
b = a++ * 2;      / b termina com o valor 6 e a com o valor 4 */
```

Operadores e Expressões

- **Operadores relacionais (< , <= , == , >= , > , !=):**
  - o resultado será 0 ou 1 (não há valores booleanos em C).

```
int a, b;
int c = 23;
int d = c + 4;


c < 20          retorna 0
d > c           retorna 1
```

Operadores e Expressões

- **Operadores lógicos (&& , || , !)**
  - a avaliação é da esquerda para a direita.
  - a avaliação para quando o resultado pode ser conhecido.

```
int a, b;
int c = 23;
int d = c + 4;

a = (c < 20) || (d > c);        /* retorna 1 */
                                /* as duas sub-expressões são avaliadas */
b = (c < 20) && (d > c);        /* retorna 0 */
                                /* apenas a primeira sub-expressão é avaliada */
```

Conversão de Tipos

- conversão de tipo é automática na avaliação de uma expressão.
- conversão de tipo pode ser requisitada explicitamente.

```
float f;          /* valor 3 é convertido automaticamente para "float"   */
float f = 3;      /* ou seja, passa a valer 3.0F, antes de ser atribuído a f */


int g, h;         /* 3.5 é convertido (e arredondado) para "int"         */
g = (int) 3.5;    /* antes de ser atribuído à variável g                 */
h = (int) 3.5 % 2 /* e antes de aplicar o operador módulo "%"            */
```

Entrada e Saída

- **Função "printf":**
- possibilita a saída de valores segundo um determinado formato.
- printf(formato, lista de constantes/variáveis/expressões...);

```
printf ("%d %g", 33, 5.3);
```

tem como resultado a impressão da linha:

33 5.3

Entrada e Saída

- **Função "printf":**
- possibilita a saída de valores segundo um determinado formato.
- printf(formato, lista de constantes/variáveis/expressões...);

```
printf ("Inteiro = %d   Real = %g", 33, 5.3);


com saída:

Inteiro = 33   Real = 5.3
```

Entrada e Saída

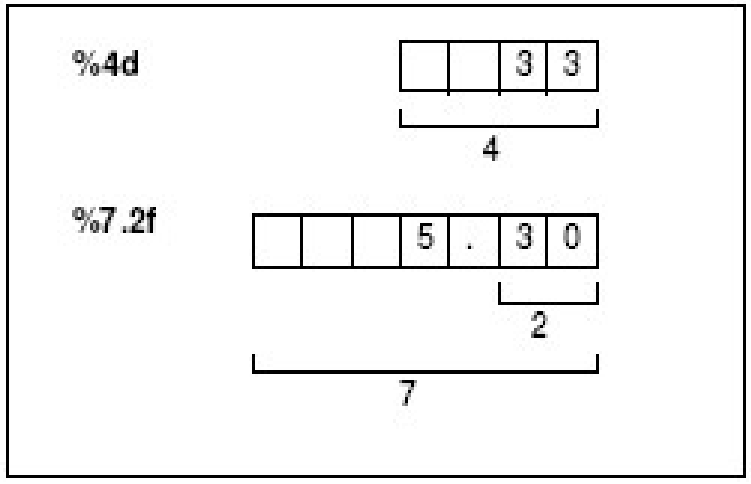
- **Função "printf":**

`%c`	*especifica um char*
`%d`	*especifica um int*
`%u`	*especifica um unsigned int*
`%f`	*especifica um double (ou float)*
`%e`	*especifica um double (ou float) no formato científico*
`%g`	*especifica um double (ou float) no formato mais apropriado (%f ou %e)*
`%s`	*especifica uma cadeia de caracteres*

Entrada e Saída

- **Função "printf":**
- Especificação de caracteres de "escape":
  - \n caractere de nova linha , \t caractere de tabulação

Especificação de tamanho de campo:



Entrada e Saída

- **Função "scanf":**
  - captura valores fornecidos via teclado

```
scanf (formato, lista de endereços das variáveis...);
```

```
int n;
scanf ("%d", &n);
```

valor inteiro digitado pelo usuário é armazenado na variável n

Entrada e Saída

- **Função "scanf":** Especificação de formato

`%c`	*especifica um char*
`%d`	*especifica um int*
`%u`	*especifica um unsigned int*
`%f, %e, %g`	*especificam um float*
`%lf, %le, %lg`	*especificam um double*
`%s`	*especifica uma cadeia de caracteres*

Entrada e Saída

- **Função "scanf":** Especificação de formato

```
scanf ("%d:%d", &h, &m);
```

valores (inteiros) fornecidos devem ser separados pelo caractere dois pontos (:)

Exercício

- 1)Calcular a estatística desvio padrão, σ, de cinco números. A fórmula requerida é:

$$\sigma = \frac{1}{4}\left(\sqrt{\sum_{i=1}^{5}(xi - \bar{X})^2}\right)$$

onde x1, x2, ... , x5 são os cinco valores lidos; x indica a média aritmética e σ conforme indicado.

Exercício

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
int main(void)
{
  float num1,num2,num3,num4,num5;
  float somadesv,desv;
  scanf("%f",&num1); scanf("%f",&num2);
  scanf("%f",&num3);  scanf("%f",&num4);
  scanf("%f",&num5);
  float media=(num1+num2+num3+num4+num5)/5.0;
  somadesv=(pow((num1-media),2)+pow((num2-  media),2)+pow((num3-
media),2)+pow((num4-media),2)+pow((num5-media),2));
  desv=1/4.0*sqrt(somadesv);
  printf ("desv=%f",desv);
  return 0;}
```

2-Controle de Fluxo

- Tópicos
  - Tomada de decisão.
  - Construções com laços.
  - Seleção.

Controle de Fluxo

- Tomada de Decisão
  - função para qualificar a temperatura:
    - se a temperatura for menor do que 20ºC, então está frio.
    - se a temperatura estiver entre 20ºC e 30ºC, então está agradável.
    - se a temperatura for maior do que 30ºC, então está quente.

Controle de Fluxo

- **Comando "if":**
  - comando básico para codificar tomada de Decisão.
  - se expr for verdadeira , executa o bloco de comandos 1.
  - se expr for falsa, executa o bloco de comandos 2.

```
if ( expr )
{ bloco de comandos 1 }
else
{ bloco de comandos 2 }

ou

if ( expr )
{ bloco de comandos }
```

Controle de Fluxo

- **Comando "if":**

```
if ( expr )
{ bloco de comandos 1 }
else
{ bloco de comandos 2}


ou


if ( expr )
{ bloco de comandos }
```

Controle de Fluxo

- **"if's encaixados":**

```
if (expr1)
{BC1;}
else if (expr2)
          {BC2;}
       else if(exp3)
                {BC3;}
        else
              {BC4;}
```

Controle de Fluxo

```
#include <stdio.h>
#include <conio.h>
int main(void)
{ int temp;
  printf("Digite a temperatura:");
  scanf("%d",&temp);
  if (temp<10)
   printf("Temperatura muito fria\n");
  else if (temp<20)
            printf("Temperatura fria\n");
  else if (temp<30)
            printf("Temperatura agradavel\n");
  else
    printf("Temperatura muito quente\n");

 return 0;  }
```

Controle de Fluxo

- Operadores Relacionais

Operador	Significado
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a
<>	diferente de
==	igual a

Atenção em C/C++:
= é o operador de atribuição
== é o operador relacional de igualdade

É um erro de programação utilizar
"=" quando se deve utilizar "=="
(e vice-versa)

Controle de Fluxo

- **Operadores Lógico**s-Um conjunto especial de operadores é necessário para combinar condições simples criando condições compostas: operadores lógicos.

Operador	Significado
não	negação
e	conjunção
ou	disjunção

Controle de Fluxo

- Sejam A e B duas variáveis lógicas, que assumem valores **verdadeiro** (**V**) ou **falso** (**F**).

A	não A
V	F
F	V

A	B	A e B
V	V	V
V	F	F
F	V	F
F	F	F

A	B	A ou B
V	V	V
V	F	V
F	V	V
F	F	F

Controle de Fluxo

- Exercício:
  - Assuma A = 2; B = 3; C = 10
  - não (A > B)
  - não (B < C)
  - A < B e C > B
  - A < B e C < B
  - A > B e C > B
  - A > B e C < B
  - A < B e C > B e A > 0
  - A < B e C > B e A < 0
  - A < B ou C > B
  - A < B ou C < B
  - A > B ou C > B
  - A > B ou C < B
  - A < B ou C > B ou A > 0
  - A < B ou C > B ou A < 0
  - não (A < B e C > B)
  - não (A < B ou C > B)
  - não (A > B) e C > B
  - não (A > B) ou B > C

Controle de Fluxo

- Operadores Lógicos em C

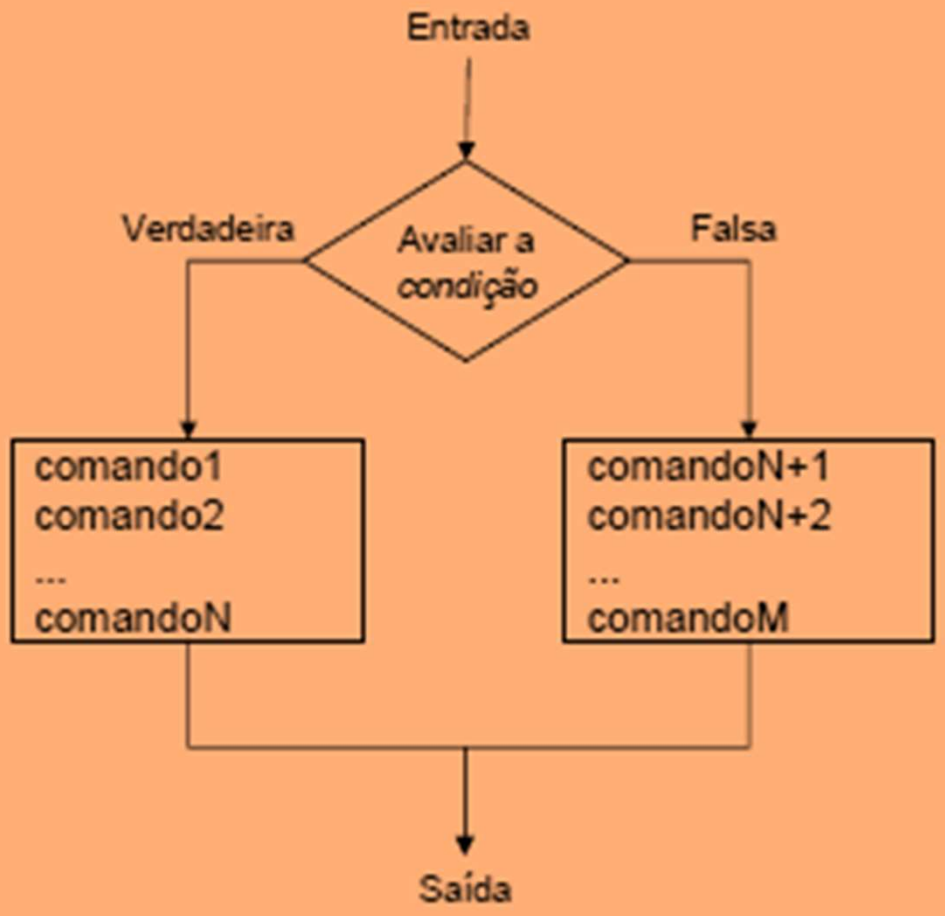
Operador	Significado
!	negação
&&	conjunção
||	disjunção

Controle de Fluxo

Pseudo-código

Se *condição* **Então**
  comando1
  comando2
  ...
  comandoN
Senão
  comandoN+1
  comandoN+2
  ...
  comandoM
Fim Se

Fluxograma



Controle de Fluxo

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int main(void)
{
  float N1,N2;
  printf("Entre com a Nota 1=");
  scanf("%f",&N1);
  printf("Entre com a Nota 2=");
  scanf("%f",&N2);
  float media=(N1+N2)/2.0;
  if(media>=5.0)
   {
     printf("\nMedia=%f",media);
     printf("\nAprovado por Media");
   }
  else
   {
     printf("\nMedia=%f",media);
     printf("\nNao Aprovado por Media");
   } return 0;
}
```

Controle de Fluxo

- Estrutura de bloco:
  - declaração de variáveis:
    - só podem ocorrer no início do corpo da função ou de um bloco
  - escopo de uma variável:
    - uma variável declarada dentro de um bloco é válida no bloco.
    - após o término do bloco, a variável deixa de existir.

```
if ( n > 0 )
   { int i; ... }
...        /* a variável i não existe neste ponto do programa */
```

Controle de Fluxo

2)Codificar um programa em C para ler um valor inteiro e determinar se ele é par ou ímpar.

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

int main(void)
{
  int N;
  printf("\nEntre com um numero;");
  scanf("%d",&N);
  if (N % 2 == 0)
    printf("Numero %d e par",N);
 else
  printf("Numero %d e impar",N);

}
```

Controle de Fluxo

Laços permitem que uma sequência de comandos seja executada repetidas vezes.

- Tipos de Laços
  - Condicionais
    - Com teste no início (tipo “enquanto”)
    - Com teste no final (tipo “repita...”)
  - Contados
    - Com teste no início (tipo “para i 1,2,...,N”)

Controle de Fluxo

- Laços Condicionais/Teste no Início

- **Enquanto** condição **Faça**
  comando1
  comando2
  ...
  comandoN
  Fim Enquanto

Fluxograma

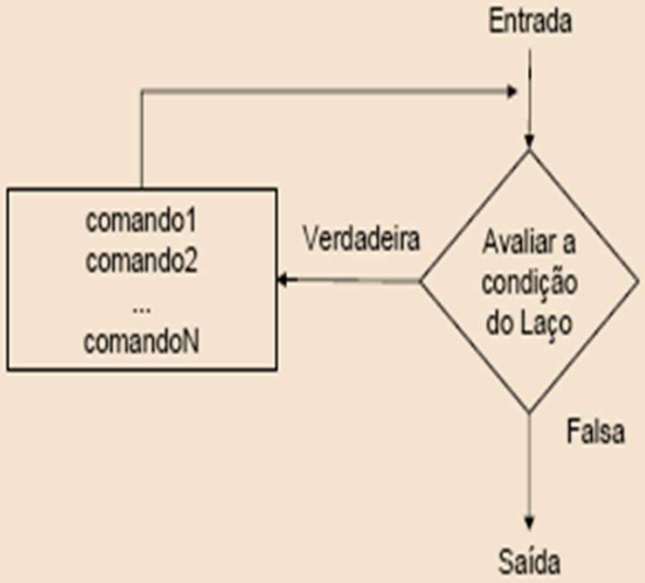
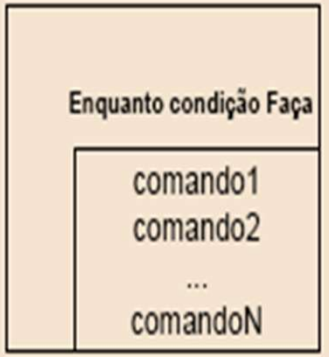


Diagrama de Nassi-Shneiderman



Controle de Fluxo

- Laços Condicionais/Teste no Início em C
- **while(**condição**)**
  - comando1;
- **while(**condição**)**

```
{ comando1;
comando2;
...
comandoN;
}
```

- Assim, se houver mais de um comando dentro de um comando **while**, eles devem ser colocados entre chaves **{** e **}**

Controle de Fluxo

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

int main(void)
{
 int contador;
 float soma,nota,media;
 contador = 1;
 soma = 0.0;
 while(contador <= 10)
 {
   printf("Nota do Aluno  %d=",contador);
   scanf("%f",&nota);
   soma = soma + nota;
   contador = contador + 1;
 }
 media = (soma/ 10.0);
 printf("Media da Classe=%f",media);
 return 0;
}
```

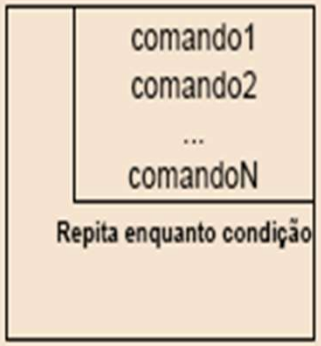
Controle de Fluxo

- Laços Condicionais/teste no fim.

Repita
comando1
comando2
...
comandoN
Enquanto condição

Fluxograma

Diagrama de Nassi-Shneiderman



Controle de Fluxo

Laços Condicionais/teste no fim.

```
int main(void)
{ int contador,N;  float soma,nota,media;   contador=1;
  soma=0.0;
  printf("Numero de Alunos:");
  scanf("%d",&N);
    while(contador<=N)
    {do
      {
       printf ("Nota do aluno%d=",contador);
       scanf("%f",&nota);
      }while (nota < 0 || nota > 10);
     soma = soma + nota;
     contador = contador + 1;
  }
  media=soma/N;
  printf("media=%f",media);   getch();   return 0;
}
```

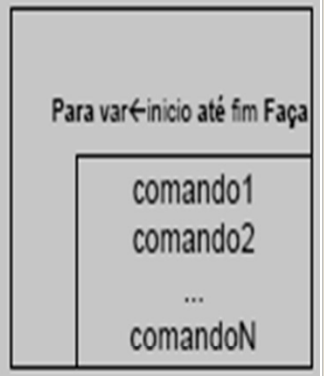
Controle de Fluxo

```
for(i=inicio; i>= fim; i- -)
{ comando1;
  comando2;
  ...
  comandoN;
}
for(i=inicio; i>= fim; i- -)
  comando1;
```

Para var←inicio **até** fim **Faça**
  comando1
  comando2
  ...
  comandoN
Fim Para

Fluxograma

Diagrama de Nassi-Shneiderman



Controle de Fluxo

- **Construções com laços**
  - Exemplo:
    - Elabore um programa para calcular o fatorial de um número inteiro N. Por definição:
      - fatorial de um número inteiro não negativo:
        - n!=(n)*(n-1)*(n-2)*.....3*2*1
        - Onde 0!=1

Controle de Fluxo

- Comando "for":
  - forma compacta para exprimir laços.

```
for (expressão_inicial; expressão_booleana; expressão_de_incremento)
{
  bloco de comandos
}
```

– equivalente a:

```
expressão_inicial;
while ( expressão_booleana )
{
  bloco de comandos
  ...
  expressão_de_incremento
}
```

Controle de Fluxo

```
#include <stdio.h>
int main (void)
{
  int k;
  int n;
  long int f = 1;
  printf("Digite um numero inteiro nao negativo:");
  scanf("%d", &n);

  /* calcula fatorial */
  k = 1;
  while (k <= n)
  {
    f = f * k;                    /* a expressão “f = f * i” é equivalente a “f *= k”    */
    k = k + 1;                    /* a expressão “k = k + 1” é equivalente a “k++”       */
  }
  printf(" Fatorial = %d \n", f);
  return 0;
}
```

Controle de Fluxo

```
#include <stdio.h>

int main (void)
{
  int k;
  int n;
  int f = 1;

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

  /* calcula fatorial */
  for (k = 1; k <= n; k=k+1) {   /* a expressão "k = k + 1" é equivalente a "i++"      */
    f = f * k;                   /* a expressão "f = f * k" é equivalente a "f *= k"    */
  }
  printf(" Fatorial = %d \n", f);
  return 0;
}
```

Controle de Fluxo

- Construções com Laços
  - Comando "do-while":
  - teste de encerramento é avaliado no final

```
do
{
  bloco de comandos
} while (expr);
```

Controle de Fluxo

```
#include <stdio.h>
int main (void)
{
  int k;
  int n;
  int f = 1;
  /* requisita valor do usuário até um número não negativo ser informado */
  do
  {  printf("Digite um valor inteiro nao negativo:");
     scanf ("%d", &n);
  } while (n<0);
  /* calcula fatorial */
  for (k = 1; k <= n; k++)
     f *= k;
  printf(" Fatorial = %d\n", f);
  return 0;
}
```