



***tema:* Programação Estruturada**

***disciplina:* ALGORITMOS E
PROGRAMAÇÃO**



ROTEIRO

1 introdução

2 elementos principais

Variáveis e constantes, operadores,
comandos de entrada e saída

3 estruturas de controle

sequência, repetição e controle

4 estruturas de dados

vetores e matrizes

5 modularização


funções e procedimentos

6 conclusão

1

Introdução

Qual o contexto de algoritmos de programação?

- 
- O que são algoritmos?
 - Sequenciamento lógico para tratar dados e destes obter informações ou realizar ações;
 - Como fazer um suco de laranja?
 - 1 - escolher a laranja; 2 - descascar a laranja; 3 - Espremer a laranja
 - Armazenamento de dados em área fixa e prevista.



→ Contexto da linguagem C;

```
#include <stdio.h> //operações de entrada e saída
#include <string.h> //operações com strings
#include <math.h> //operações matemáticas
#include <time.h> //data e hora
#include <stdbool.h> //dados booleanas

int main(){

    return 0;

}
```

2

elementos principais

palavras reservadas, variáveis, constantes,
operadores e identificadores

2.1 VARIÁVEIS E CONSTANTES

```
#define constante 100.2
#include <stdio.h>
#include <stdbool.h>

bool booleanno = true;
int inteiro = 3;
char caractere = 'a';
float real6bits = 22.2;
double real8bits = 1.222;
```

diretivas, (nome) identidades, string de formato



```
int main(){
    printf("constante: %f \n", constante);
    printf("booleanno: %i \n", booleanno);
    printf("inteiro: %i \n", inteiro);
    printf("caractere: %c \n", caractere);
    printf("real6bits: %f \n", real6bits);
    printf("real8bits: %.1f \n", real8bits);
    return 0;
}
```

2.1 VARIÁVEIS E CONSTANTES

```
#define constante 100.2  
bool booleanno = true;  
int inteiro = 3;  
char caractere = 'a';  
float real6bits = 22.2;  
double real8bits = 1.222;
```

```
constante: 100.200000  
booleanno: 1  
inteiro: 3  
caractere: a  
real6bits: 22.200001  
real8bits: 1.2
```


2.2 OPERADORES (Aritméticos)

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

```
int main(){  
    printf("soma: %i \n", 1 + 3 );  
    printf("subtração: %i \n", 1 - 3);  
    printf("resto divisão : %i \n", 1%3);  
    printf("multiplicação inteiro: %i \n", 1*3);  
    printf("divisão inteiro: %i \n", 1/3);  
    printf("multiplicação real: %f \n", 1.1*3.);  
    printf("divisão real: %f \n", 1./3.);  
  
    return 0;  
}
```

```
soma: 4  
subtração: -2  
resto divisão : 1  
multiplicação inteiro: 3  
divisão inteiro: 0  
multiplicação real: 3.300000  
divisão real: 0.333333
```

2.2 OPERADORES (Lógicos)

```
#include <stdio.h>

int i = 1;
int j = 3;

int main(){
    printf("primeiro E: %i \n", (i == 1) && (j == 3));
    printf("segundo E: %i \n", (i == 2) && (j == 3));
    printf("terceiro E: %i \n", (i == 2) && (j == 4));
    printf("primeiro OU: %i \n", (i == 1) || (j == 3));
    printf("segundo OU: %i \n", (i == 2) || (j == 3));
    printf("terceiro OU: %i \n", (i == 2) || (j == 4));
    printf("NEGAÇÃO : %i \n", !(i == 2));
    return 0;
}
```

Operador igualdade e atribuição



```
primeiro E: 1
segundo E: 0
terceiro E: 0
primeiro OU: 1
segundo OU: 1
terceiro OU: 0
NEGAÇÃO : 1
```

2.2 OPERADORES (Relacionais)

```
#include <stdio.h>
int i = 1;
int j = 3;

int main(){
    printf("igualdade: %i \n", (i == i));
    printf("menor que: %i \n", (i < j));
    printf("menor ou igual que: %i \n", (i <= j));
    printf("maior que: %i \n", (i > j));
    printf("maior ou igual que: %i \n", (i >= j));
    printf("diferentes: %i \n", (j != j+1));

    return 0;
}
```

```
igualdade: 1
menor que: 1
menor ou igual que: 1
maior que: 0
maior ou igual que: 0
diferentes: 1
```

2.3 ENTRADA E SAÍDA

```
#include <stdio.h>

int i = 1; float f = 1; char c = 's';

int main(){
    printf("Entre com um valor inteiro para i:");
    scanf("%i", &i);
    printf("%i \n", i);
    printf("Entre com um valor real para f:");
    scanf("%f", &f);
    printf("%f \n", f);
    printf("Entre com um valor caractere para c:");
    scanf(" %c", &c);
    printf("%c \n", c);
    return 0;
}
```

scanf() para variável tipo *char*



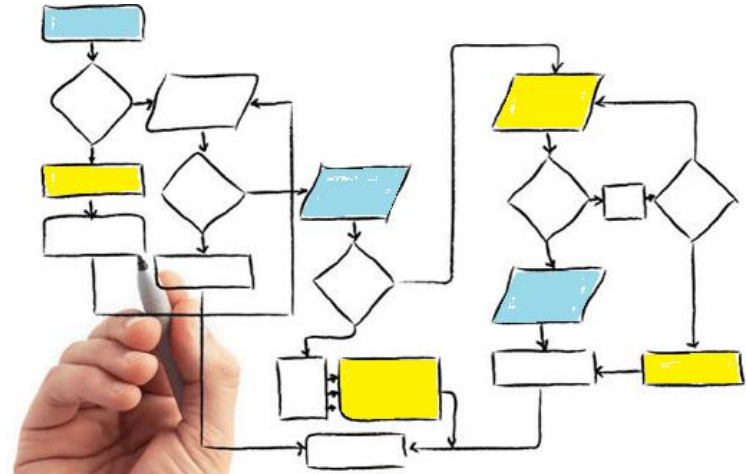
```
Entre com um valor inteiro para i:2
2
Entre com um valor real para f:2
2.000000
Entre com um valor caractere para c:M
M
```

3

estruturas de controle

Como ordenar?

- Resultado da teoria da linguagem de programação;
- Fluxograma;



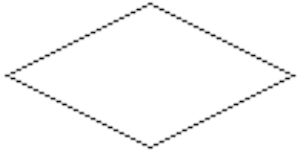
elementos de um fluxograma



Início e fim do algoritmo



Sentido do fluxo



Bloco de decisão



Bloco de entrada ou saída



Bloco de saída



Bloco de entrada manual



Bloco de ação

3.1 SEQUÊNCIA



“ Exibir o valor de uma **variável inteira**, i (iniciando em 1), três vezes e **somar** com 10 a cada **exibição**. Antes de finalizar o algoritmo **exibir** o **resto da divisão** do acumulativo de i por 7.

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

```
int i = 1;
```

```
// int j = 3;
```

```
int main(){
```

```
    printf("i: %i \n", i);
```

```
    i = i + 10;
```

```
    printf("i: %i \n", i);
```

```
    i = i + 10;
```

```
    printf("i: %i \n", i);
```

```
    i = i + 10;
```

```
    // 31/7 = 7*4 + 3
```

```
    printf("resto da divisão: %i \n", i%7);
```

```
    return 0;
```

```
}
```

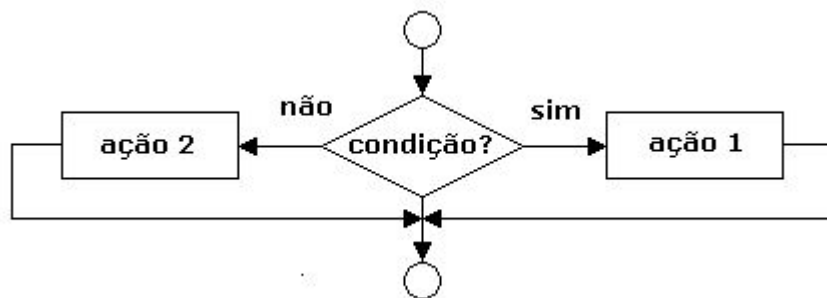
comentário em código



```
i: 1
i: 11
i: 21
resto da divisão: 3
```

3.2 CONTROLE (IF-ELSE)

```
if(){  
    Instruções;  
} else () {  
    Instruções;  
}
```



“ Verifique **se** duas **entradas** de valores do **tipo int**, *i* e *j*, e inseridas pelo usuário, são **iguais**. **Se** forem diferentes, indique qual é **maior**.

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

```
int i = 1;
```

```
int j = 3;
```

```
int main(){
```

```
    printf("Entre com um valor inteiro para i:");
```

```
    scanf("%i", &i);
```

```
    printf("Entre com um valor inteiro para j:");
```

```
    scanf("%i", &j);
```

```
    if (i== j){
```

```
        printf("São iguais");
```

```
    } else {
```

```
        if (i>j){
```

```
            printf("i maior que j");
```

```
        } else {
```

```
            printf("j maior que i");
```

```
        }
```

```
    }
```

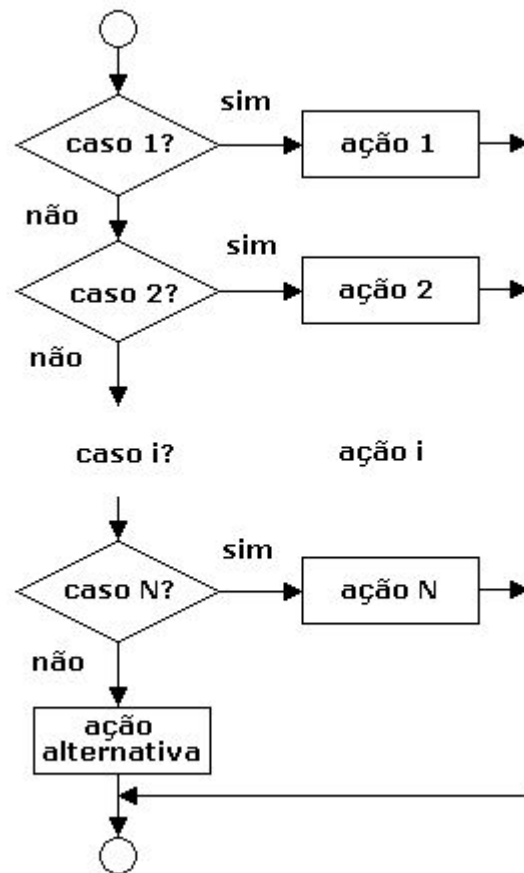
```
    return 0;
```

```
}
```

```
Entre com um valor inteiro para i:5
Entre com um valor inteiro para j:8
j maior que i
```

3.2 CONTROLE (SWITCH-CASE)

```
switch (variável){  
    case constante1:  
        Instruções;  
        break;  
  
    case constante2:  
        Instruções;  
        break;  
  
    default:  
        Instruções;  
}
```



“ Forneça três **opções** para o usuário e identifique qual foi **escolhida**. O menu deve ser:

- 1 - Escolha número um
- 2 - Escolha número dois
- 3 - Escolha número três

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

```
int i ;
```

```
int main(){  
    printf("Entre com um valor do menu \n");  
    printf("1 - Escolha número um \n");  
    printf("2 - Escolha número dois \n");  
    printf("3 - Escolha número três \n");  
    scanf("%i", &i);
```

```
    switch (i){  
        case 1:  
            printf("você escolheu 1");  
            break;  
  
        case 2:  
            printf("você escolheu 2");  
            break;  
  
        case 3:  
            printf("você escolheu 3");  
            break;  
  
        default:  
            printf("você fez uma escolha  
inválida");  
    }  
  
    return 0;  
}
```



```
switch (i){
    case 1:
        printf("você escolheu 1");
        break;

    case 2:
        printf("você escolheu 2");
        break;

    case 3:
        printf("você escolheu 3");
        break;

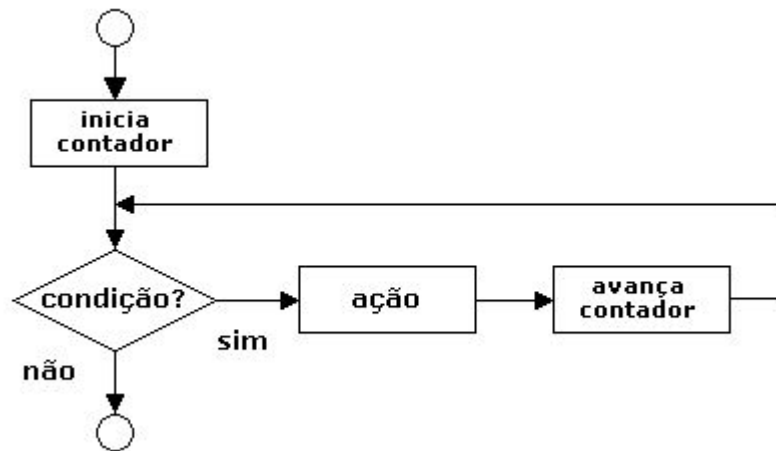
    default:
        printf("você fez uma escolha
inválida");
}
```

```
Entre com um valor do menu
1 - Escolha número um
2 - Escolha número dois
3 - Escolha número três
1
você escolheu 1
```

```
Entre com um valor do menu
1 - Escolha número um
2 - Escolha número dois
3 - Escolha número três
S
você fez uma escolha inválida
```

3.3 REPETIÇÃO (FOR)

```
for(valor_inicial; condição_final; valor_incremento){  
    instruções;  
}
```



“ Criar um **laço** de repetição *for* com **10 interações**. A cada interação verificar **se** o valor resultante é par ou ímpar e **exibir** o número quando par.

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

```
int i = 0;
```

```
int main(){  
    for(i; i < 10; i++){  
        if(i%2 == 0){  
            printf("o valor %i é par \n", i);  
        }  
    }  
    return 0;  
}
```

operador de incremento

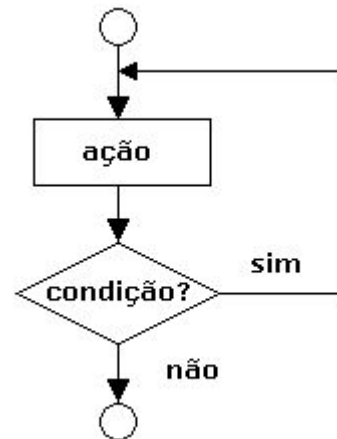
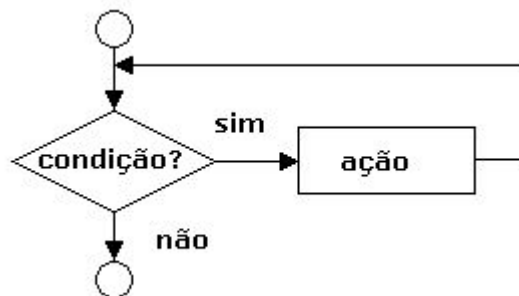


```
o valor 2 é par  
o valor 4 é par  
o valor 6 é par  
o valor 8 é par
```

3.3 REPETIÇÃO (WHILE E DO-WHILE)

```
while (condição){  
    Instrução;  
}
```

```
do {  
    Instrução;  
} while (condição)
```



“ Criar um **laço** de repetição *while* enquanto a variável **j (int)** iniciando em 13 **tiver o resto da divisão por 7 diferente de 0** e **decrementar** j em 1 a cada interação.

operador de decremento



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

```
int j = 13;
```

```
int main(){  
    while(j%7 != 0){
```

```
        //j = j - 1
```

```
        j--;
```

```
    }
```

```
    printf("valor de j ao sair do laço: %i", j);
```

```
}
```

```
valor de j ao sair do laço: 7
```

4

estruturas de dados

Armazenamento apenas em variáveis unitárias?



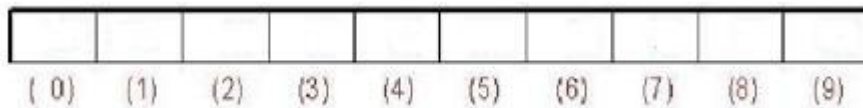
→ E se quiséssemos salvar as notas dos alunos de uma turma?

4.1 VETORES

→ Variável composta homogênea

```
#include <stdio.h>

int i = 0;
int v[10];
int main(){
    for (i; i < 10; i++){
        Instrução;
    }
}
```



“ Criar um **laço** de repetição *for* com **15 interações** e **preencha** um **vetor de inteiros** de tamanho 5. **Atribua** o valor da interação (i) ao vetor na posição i. **Exiba** o vetor.

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

```
int i = 0;  
int v[5];  
int main(){  
    for (i; i< 15; i++){  
        v[i] = i;  
    }  
    for (i=0; i< 5; i++){  
        printf("vetor: %i \n", v[i]);  
    }  
}
```

```
vetor: 0  
vetor: 1  
vetor: 2  
vetor: 3  
vetor: 4
```

→ E se quiséssemos salvar as notas de cada turma de uma escola?

4.2 MATRIZES

→ Variável composta por vetores

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

```
int m[3][3], i, j;
```

```
int main(){
```

```
    for (i=1; i<= 3; i++){
```

```
        for (j=1; j<= 3; j++){
```

```
            Instrução;
```

```
        }
```

```
    }
```

```
}
```

MATRIZ 3X3

COLUNA 1

COLUNA 2

COLUNA 3

LINHA 1	1,1	1,2	1,3
LINHA 2	2,1	2,2	2,3
LINHA 3	3,1	3,2	3,3

“ Crie uma matriz identidade 3x3
com linguagem C.

$$\mathbf{I}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

matriz identidade de ordem 3

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

```
int m[3][3], i, j;
```

```
int main(){
```

```
    for (i=0; i< 3; i++){
```

```
        for (j=0; j< 3; j++){
```

```
            if (i == j){
```

```
                m[i][j] = 1;
```

```
            } else {
```

```
                m[i][j] = 0;
```

```
            }
```

```
        }
```

```
    }
```

```
    for (i=0; i< 3; i++){
```

```
        for (j=0; j< 3; j++){
```

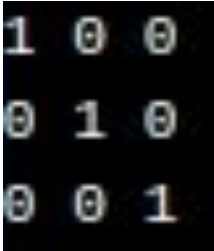
```
            printf("%i ", m[i][j]);
```

```
        }
```

```
        printf("\n");
```

```
    }
```

```
}
```




1	0	0
0	1	0
0	0	1

5

modularização

Como diminuir o código de um programa?

- 
- Modulos funcionais utilizados pela *main()*;
 - Diminuir código;
 - Diferentes fases de detalhamento;
 - Teste e reutilização.

5.1 PROCEDIMENTOS

operador de acumulação



→ Sem retorno de valor

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

```
int i = 1;
```

```
void Procedimento(){  
    i += 10;  
    printf("valor i no procedimento: %i \n", i);  
}
```

```
int main(){  
    printf("valor i antes do procedimento: %i \n", i);  
    Procedimento();  
    printf("valor i depois do procedimento: %i \n", i);  
    return 0;  
}
```

```
valor i antes do procedimento: 1  
valor i no procedimento: 11  
valor i depois do procedimento: 11
```



5.2 FUNÇÕES

- possui retorno de dados
- variáveis locais e globais
- nome é variável de expressão

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

```
int i = 1;
```

```
int Funcao(int i){  
    i += 10;  
    printf("valor i na função: %i \n", i);  
    return i;  
}
```

```
int main(){  
    printf("valor i antes da função: %i \n", i);  
    int iFunc = Funcao(i);  
    printf("valor i depois da função: %i \n", i);  
    printf("valor iFunc: %i \n", iFunc);  
    return 0;  
}
```

```
valor i antes da função: 1  
valor i na função: 11  
valor i depois da função: 1  
valor iFunc: 11
```

→ podem ser recursivas

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

```
int i = 4;
```

```
int FuncaoRecursiva(int i){  
    if (i == 0){  
        return 0;  
    }  
    i += FuncaoRecursiva(i-1);  
    return i;  
}
```

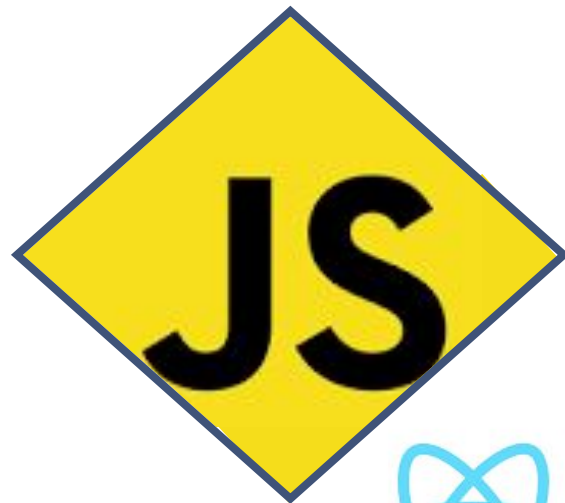
```
int main(){  
    int iFunc = FuncaoRecursiva(i);  
    printf("valor iFunc: %i \n", iFunc);  
    return 0;  
}
```

A black rectangular box with white text that reads "valor iFunc: 10". This represents the output of the recursive function call shown in the code to the left.

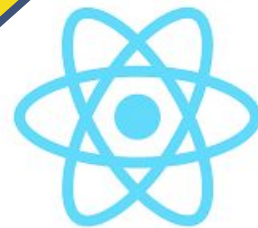
6

conclusão

contexto de possíveis aplicações



TensorFlow



aplicações



```
import numpy as np
```

```
I = np.identity(3)  
I
```

```
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```



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

```
int m[3][3], i, j;  
int main(){  
    for (i=0; i< 3; i++){  
        for (j=0; j< 3; j++){  
            if (i == j){  
                m[i][j] = 1;  
            } else {  
                m[i][j] = 0;  
            }  
        }  
    }  
    for (i=0; i< 3; i++){  
        for (j=0; j< 3; j++){  
            printf("%d ", m[i][j]);  
        }  
        printf("\n");  
    }  
}
```



OBRIGADA!

QUESTIONAMENTOS?