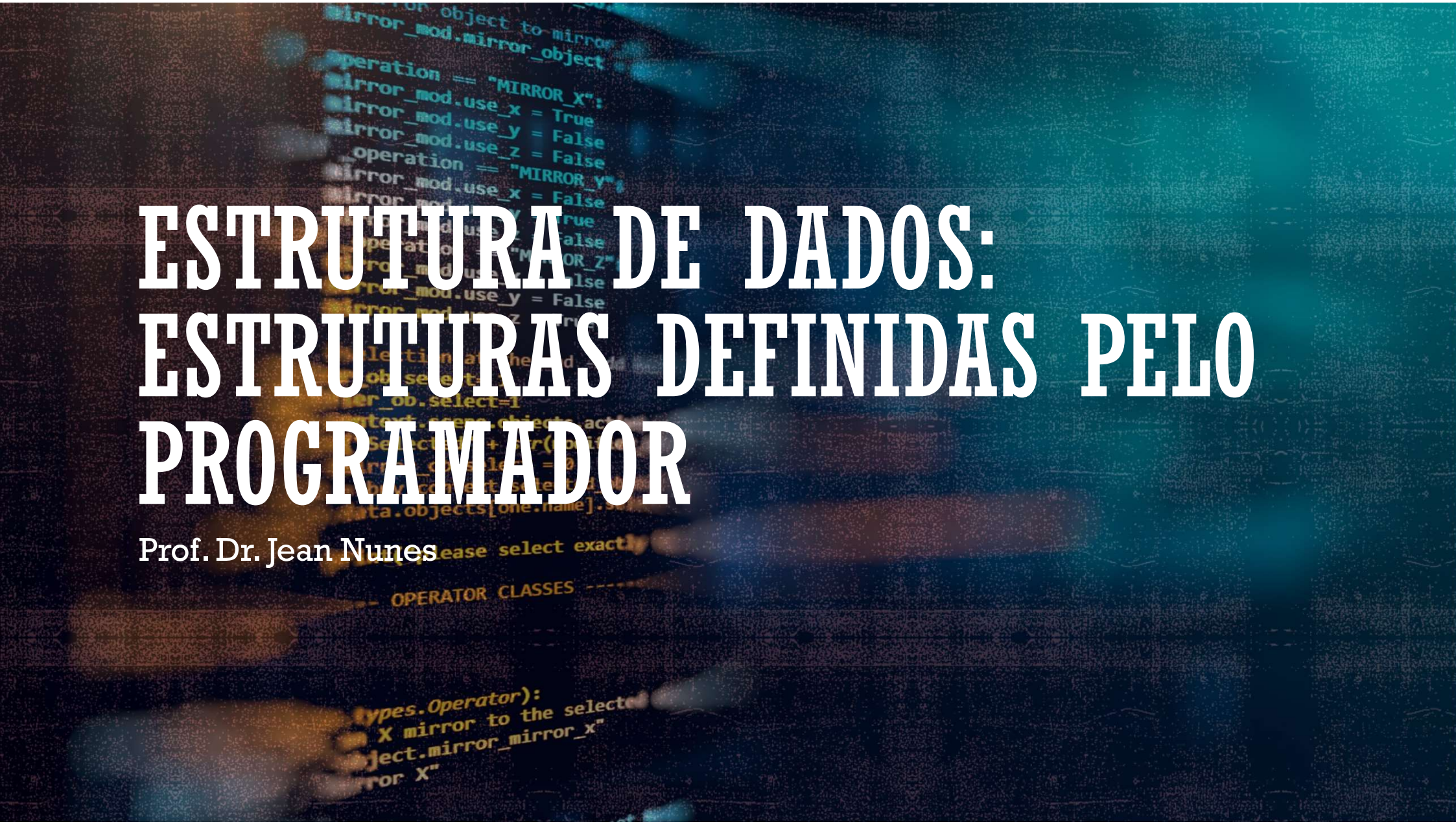


ESTRUTURA DE DADOS: ESTRUTURAS DEFINIDAS PELO PROGRAMADOR

Prof. Dr. Jean Nunes

VARIÁVEIS

Categorias de variáveis:

simples: **int**, **float**, **double** e **char**;

compostas homogêneas (ou seja, do mesmo tipo): definidas por **array**.

A linguagem C permite a criação de novas estruturas a partir dos tipos básicos.

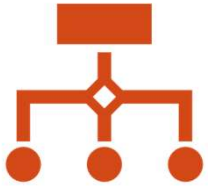
struct

ESTRUTURAS

```
struct nomestruct{
    tipo1 campo1;
    tipo2 campo2;
    ...
    tipoN campoN;
};
```

Uma estrutura pode ser vista como um **novo tipo de dado**, que é formado por composição de variáveis de outros tipos.

ESTRUTURAS



Agrupamento de dados.

Ex.: cadastro de pessoas.

Todas essas informações são da mesma pessoa, logo podemos agrupá-las.

Isso facilita também lidar com dados de outras pessoas no mesmo programa

```
struct cadastro{
      char nome[50];
      int idade;
      char rua[50]
      int numero;
};
```

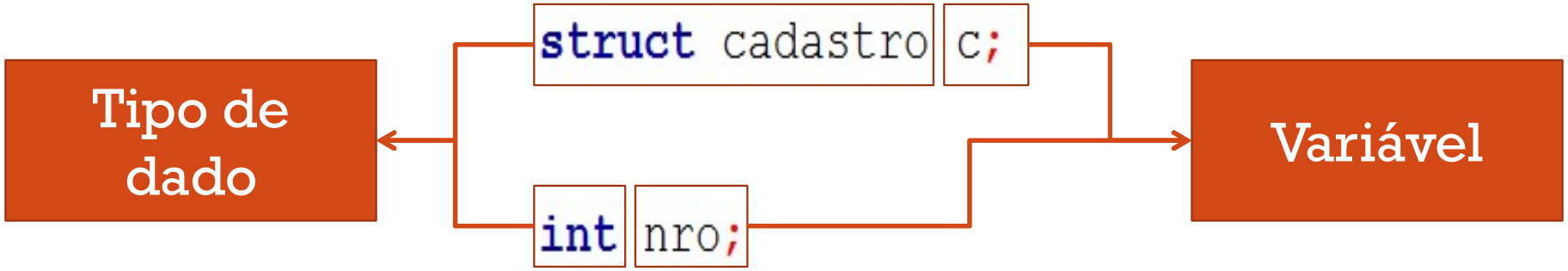
char nome[50]
int idade;
char rua[50];
int numero;

cadastro

ESTRUTURAS - DECLARAÇÃO

```
struct cadastro c;
```

Uma vez definida a estrutura, uma **variável** pode ser declarada de modo similar aos tipos já existentes.



- Por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável.

ESTRUTURAS - DECLARAÇÃO

EXERCÍCIO

Declare uma estrutura capaz de armazenar o número e 3 notas para um dado aluno.

EXERCÍCIO - SOLUÇÃO

```c
struct aluno {
    int num_aluno;
    int nota1, nota2, nota3;
};

struct aluno {
    int num_aluno;
    int nota1;
    int nota2;
    int nota3;
};

struct aluno {
    int num_aluno;
    int nota[3];
};
```

ESTRUTURAS

O uso de estruturas facilita na manipulação dos dados do programa. Imagine declarar 4 cadastros, para 4 pessoas diferentes:

```
char nome1[50], nome2[50], nome3[50], nome4[50];
int idade1, idade2, idade3, idade4;
char rua1[50], rua2[50], rua3[50], rua4[50]
int numero1, numero2, numero3, numero4;
```

ESTRUTURAS

Declarando a estrutura

```
struct cadastro{
    char nome[50];
    int idade;
    char rua[50]
    int numero;
};

//declarando 4 cadastros
struct cadastro c1, c2, c3; c4
```

```
//declarando a variável
struct cadastro c;

//acessando os seus campos
strcpy(c.nome,"João");
scanf("%d",&c.idade);
strcpy(c.rua,"Avenida 1");
c.numero = 1082;
```

Cada variável da estrutura pode ser acessada com o operador ponto “.”

ACESSO ÀS VARIÁVEIS

Como é feito o acesso às variáveis da estrutura?

ACESSO ÀS VARIÁVEIS

Como nos arrays, uma estrutura pode ser previamente inicializada:

```
struct ponto {
    int x;
    int y;
};

struct ponto p1 = { 220, 110 };
```

```
struct cadastro c;

gets(c.nome);//string
scanf("%d",&c.idade);//int
gets(c.rua);//string
scanf("%d",&c.numero);//int
```

Lemos cada variável independentemente, respeitando seus tipos.

ACESSO ÀS VARIÁVEIS

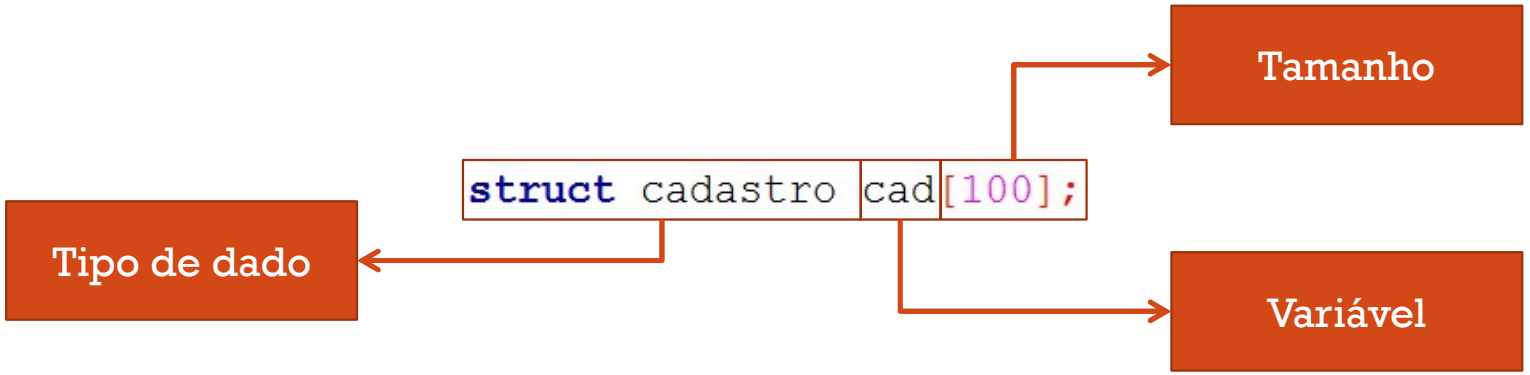
E se quiséssemos ler os valores das variáveis da estrutura?

ESTRUTURAS

E se, ao invés de 4 cadastros, quisermos fazer 100 cadastros?

ARRAY DE ESTRUTURAS

- Sua declaração é similar a declaração de um array de um tipo básico



- Desse modo, declara-se um array de 100 posições, onde cada posição é do tipo **struct cadastro**.

ARRAY DE ESTRUTURAS

ATENÇÃO:

- **struct**: define um "conjunto" de variáveis que podem ser de tipos diferentes;
- **array**: é uma "lista" de elementos de mesmo tipo.

```
struct cadastro{
    char nome[50];
    int idade;
    char rua[50]
    int numero;
};
```

cad[0]	cad[1]	cad[2]	cad[3]
char nome[50]; int idade; char rua[50] int numero;	char nome[50]; int idade; char rua[50] int numero;	char nome[50]; int idade; char rua[50] int numero;	char nome[50]; int idade; char rua[50] int numero;

ARRAY DE ESTRUTURAS

```
int main(){
   struct cadastro c[4];
   int i;
   for(i=0; i<4; i++){
      gets(c[i].nome);
      scanf("%d",&c[i].idade);
      gets(c[i].rua);
      scanf("%d",&c[i].numero);
   }
   system("pause");
   return 0;
}
```

Num array de estruturas, o operador de ponto (**.**) vem depois dos colchetes (**[ ]**) do índice do **array**.

EXERCÍCIO

Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos.

```
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};
```

EXERCÍCIO - SOLUÇÃO

Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos

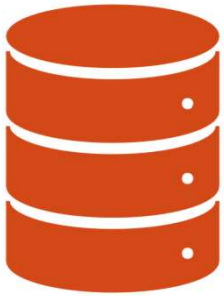
```
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};


int main(){
    struct aluno a[10];
    int i;
    for(i=0;i<10;i++){
        scanf("%d",&a[i].num_aluno);
        scanf("%f",&a[i].nota1);
        scanf("%f",&a[i].nota2);
        scanf("%f",&a[i].nota3);
        a[i].media = (a[i].nota1 + a[i].nota2 + a[i].nota3)/3.0
    }
}
```

ATRIBUIÇÃO ENTRE ESTRUTURAS

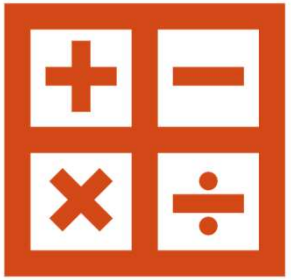
- Atribuições entre estruturas só podem ser feitas quando as estruturas possuem o mesmo nome!

```
struct cadastro c1,c2;
c1 = c2; //CORRETO

struct cadastro c1;
struct ficha c2;
c1 = c2; //ERRADO!! TIPOS DIFERENTES
```

ATRIBUIÇÃO ENTRE ESTRUTURAS



```
struct cadastro c[10];
c[1] = c[2]; //CORRETO
```



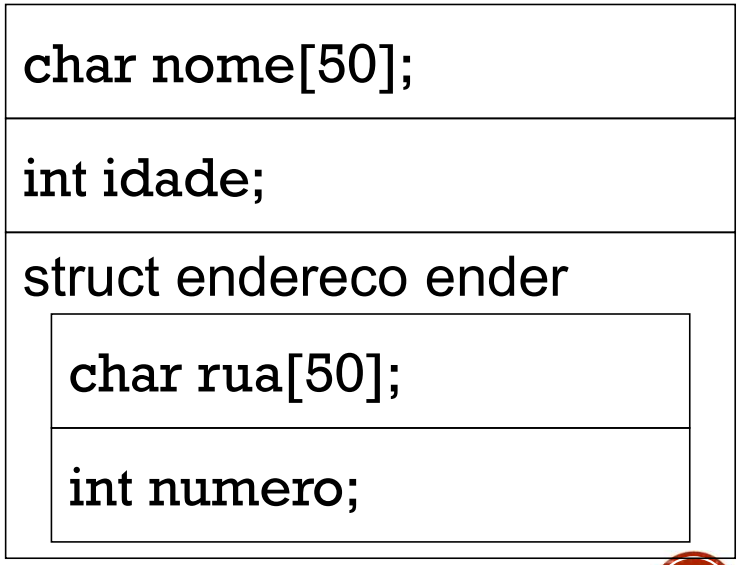
Em arrays, a atribuição entre diferentes elementos do array é válida!

Note que nesse caso, os tipos dos diferentes elementos do array são sempre IGUAIS.

ESTRUTURAS DE ESTRUTURAS

Sendo uma estrutura um tipo de dado, podemos declarar uma estrutura que utilize outra estrutura previamente definida:

```
struct endereco{
     char rua[50]
     int numero;
};
struct cadastro{
     char nome[50];
     int idade;
     struct endereco ender;
};
```



cadastro

ESTRUTURAS DE ESTRUTURAS

Nesse caso, o acesso aos dados do **endereço** do cadastro é feito utilizando novamente o operador ponto “.”.

```
struct cadastro c;

//leitura
gets(c.nome);
scanf("%d",&c.idade);
gets(c.ender.rua);
scanf("%d",& c.ender.numero);

//atribuição
strcpy(c.nome,"João");
c.idade = 34;
strcpy(c.ender.rua,"Avenida 1");
c.ender.numero = 131;
```

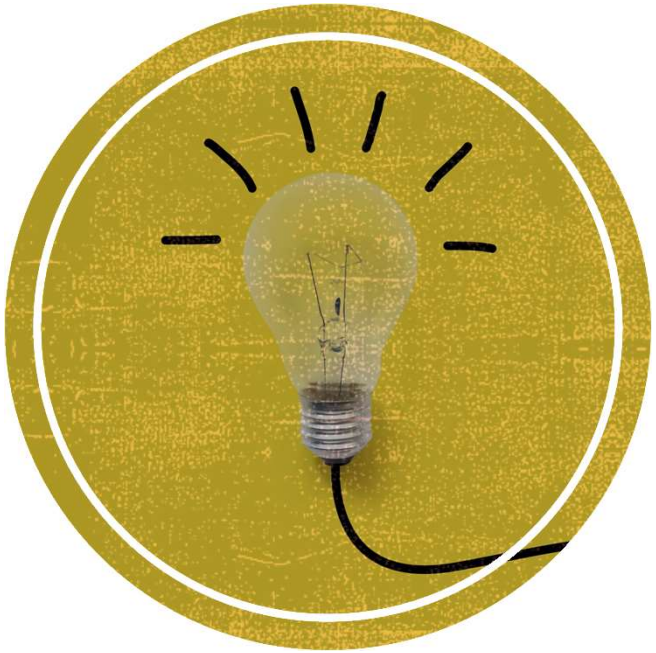
```
struct ponto {
    int x, y;
};

struct retangulo {
    struct ponto inicio, fim;
};

struct retangulo r = {{10,20},{30,40}};
```

ESTRUTURAS DE ESTRUTURAS

Inicialização de uma estrutura de estruturas

TYPEDEF

- A linguagem C permite que o programador defina os seus próprios tipos com base em outros tipos de dados existentes.



```
typedef tipo_existente novo_nome;
```

TYPEDEF

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

typedef int inteiro;

int main(){
    int x = 10;
    inteiro y = 20;
    y = y + x;
    printf("Soma = %d\n",y);

    return 0;
}
```

O comando **typedef** não cria um novo tipo chamado **inteiro**. Ele apenas cria um sinônimo (**inteiro**) para o tipo **int**

TYPEDEF

```c
struct cadastro{
    char nome[300];
    int idade;
};
// redefinindo o tipo struct cadastro
typedef struct cadastro CadAlunos;

int main(){
    struct cadastro aluno1;
    CadAlunos aluno2;

    return 0;
}
```

É muito utilizado para definir nomes mais simples para estrutura, evitando carregar a palavra **struct** sempre que referenciamos a estrutura

1. CONTROLE DE NOTAS

I. Escreva um programa que usa uma estrutura capaz de armazenar as notas das atividades práticas e a nota final dos alunos de uma disciplina a partir da seguinte fórmula expressa pela Equação abaixo:

$$AV_1 = \left(\left(\frac{AP_{1ªmn} + AP_{2ªmn} + AP_{3ªmn} + AP_{4ªmn}}{4}\right) * 0{,}4\right) + (NP * 0{,}6))$$

AP – Atividade Prática: Subscrito "1ªmn": "primeira maior nota". Os demais subscritos seguem a ordem representada pelo número.

II. Faça o cadastro de 10 alunos e mostre o resultado.

2. SIMULAÇÃO DE EMPRÉSTIMO

I. Escreva um programa que usa uma estrutura capaz de armazenar dados de solicitações de empréstimos de clientes de um banco. O programa deve ler os seguintes dados: **nome do cliente, salário do cliente, o valor do empréstimo e o número de meses para quitação.**

- O programa deve calcular o valor da parcela do empréstimo requerido.
- Se o valor da parcela for maior que 20% do salário, o empréstimo será **reprovado**, caso contrário, o empréstimo será **aprovado**.

II. Simule a solicitação de 10 empréstimos. No final, imprima a quantidade de empréstimos concedidos e a quantidade de empréstimos não concedidos.

3. PEDIDOS

O cardápio de uma lanchonete é o seguinte:

Índice	Especificação	Código	Preço
1	Cachorro-quente	100	R$ 5.20
2	Bauru simples	101	R$ 6.30
3	Bauru com ovo	102	R$ 6.50
4	Hambúrguer	103	R$ 7.20
5	Cheeseburguer	104	R$ 8.30
6	Refrigerante	105	R$ 5.00

I. Escreva um programa que usa estruturas capaz de armazenar os pedidos dos clientes.

II. Simule 10 pedidos. No final, imprima o faturamento do dia.