

Linguagem C

Escopo de Variáveis

- Na linguagem C, as variáveis podem ser declaradas em diversas partes de um programa.
- A região dentro do programa onde o nome de uma variável é visível (ou tem significado) é conhecida como **escopo** da variável.
- A linguagem C define três categorias de escopo:
 - Bloco
 - Parâmetro de função
 - Arquivo

Linguagem C

- **Bloco**: região do programa delimitada por chaves { e }.
- Exemplo: **corpo de uma função**

```
void alarme(int n)
{
    int i; ←
    for (i = 0; i < n; i++)
        printf("\a");

    return;
}
```

Normalmente, as variáveis **locais** são declaradas no **início do bloco**.

- Uma variável local tem **visibilidade apenas dentro do bloco** em que foi declarada.
- Mas, mesmo dentro do bloco, a variável local pode não ser visível, caso exista um **bloco mais interno** que declara uma outra variável de mesmo nome.

Linguagem C

- Escopo de bloco:

```
void simplifica(int x, int y)
{
    int n,d,r;
    n = x;
    d = y;
    while (d != 0)
    {
        r = n % d;
        n = d;
        d = r;
    }
    x = x/n;
    y = y/n;
    printf("%d / %d",x,y);
    return;
}
```

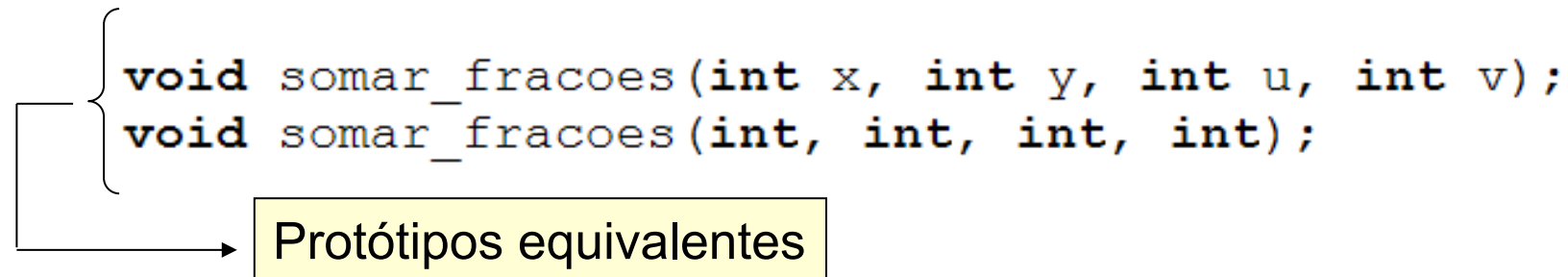
r pode ser usada dentro e fora do **while**.

```
void simplifica(int x, int y)
{
    int n,d;
    n = x;
    d = y;
    while (d != 0)
    {
        int r;
        r = n % d;
        n = d;
        d = r;
    }
    x = x/n;
    y = y/n;
    printf("%d / %d",x,y);
    return;
}
```

r só pode ser usada dentro do **while**.

Linguagem C

- **Escopo parâmetro de função** existe na declaração de protótipos ou no cabeçalho de definição de funções.
- Variáveis declaradas em um protótipo de função têm significado apenas dentro do próprio protótipo.
- Assim, no protótipo de uma função pode-se declarar apenas os tipos dos parâmetros:



```
void somar_fracoas(int x, int y, int u, int v);  
void somar_fracoas(int, int, int, int);
```

Protótipos equivalentes

- Variáveis declaradas como parâmetros no cabeçalho de uma função são consideradas **variáveis locais** à função.

Linguagem C

- Variáveis declaradas **fora** de qualquer função do programa têm **escopo de arquivo**.
- Uma variável com escopo de arquivo tem significado no restante do programa, **a partir da linha em que foi declarada**.
- Variável com escopo de arquivo: **variável global**.

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

void obter_fracoas();
void somar_fracoas(int x, int y, int u, int v);
void subtrair_fracoas(int x, int y, int u, int v);
void multiplicar_fracoas(int x, int y, int u, int v);
void dividir_fracoas(int x, int y, int u, int v);
void simplificar_fracao(int x, int y);

int a,b,c,d;

int main(int args, char * arg[])
{
    ...
}
```

Linguagem C

- Uma variável global **perde visibilidade** em uma função, caso haja declaração de variável local de mesmo nome.

Variáveis distintas

```
int a[3] = { 1,2,3 };  
int i = 0;  
void func()  
{  
    int i = 2;  
    printf("Em func: a[%d] = %d\n",i,a[i]);  
}  
  
void main()  
{  
    func1();  
    i = 1;  
    printf("Em main: a[%d] = %d\n",i,a[i]);  
}
```

Linguagem C

Passagem de Parâmetros

- Mecanismo usado para informar os valores a serem processados por uma função.
- A linguagem C define dois mecanismos: **passagem por valor** e **passagem por endereço** (ou **passagem por referência**).
- Normalmente, a passagem de parâmetros a uma função é **por valor**.
- Mas, como os parâmetros de uma função são variáveis **locais**, alguns aspectos devem ser observados.

Linguagem C

- Passagem de parâmetros por valor:

```
void alterar(int x, int y, int z)
{
    printf("Valores recebidos ... %d, %d e %d\n",x,y,z);
    x++;
    y++;
    z++;
    printf("Valores alterados ... %d, %d e %d\n",x,y,z);
}

void main()
{
    int a = 1, b = 2, c = 3;

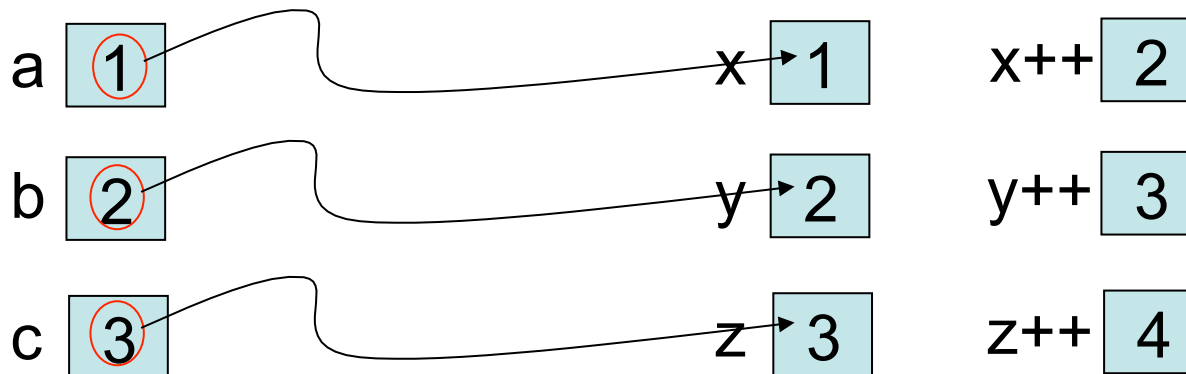
    alterar(a,b,c);
    printf("Valores finais ..... %d, %d e %d\n",a,b,c);
}
```


Linguagem C

- Observe que os valores das variáveis **a**, **b** e **c** não foram modificados na função **alterar**. Por quê?
- O tipo de passagem de parâmetros utilizado é **por valor**. Ou seja, são feitas apenas cópias dos valores das variáveis **a**, **b**, e **c** nas variáveis **x**, **y** e **z**.

Escopo: função **main**

Escopo: função **alterar**



Apenas os conteúdos de **x**, **y** e **z** são alterados.

Linguagem C

- Passagem de parâmetros por referência:

```
void alterar(int *x, int *y, int *z)
{
    printf("Valores recebidos ... %d, %d e %d\n", *x, *y, *z);
    *x++;
    *y++;
    *z++;
    printf("Valores alterados ... %d, %d e %d\n", *x, *y, *z);
}

void main()
{
    int a = 1, b = 2, c = 3;

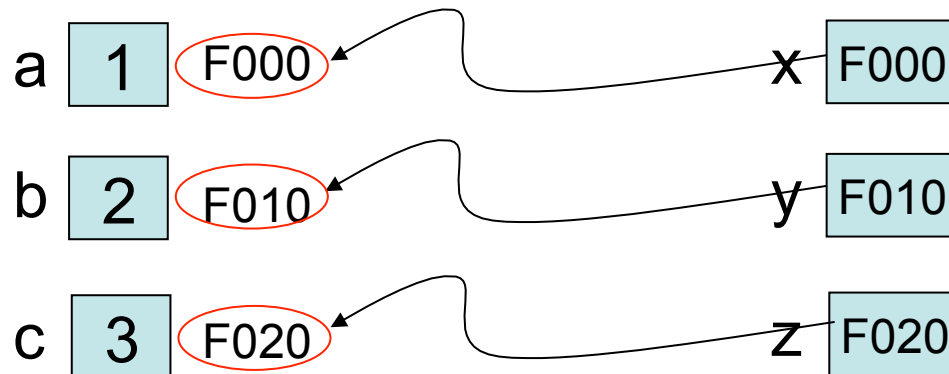
    alterar(&a, &b, &c);
    printf("Valores finais ..... %d, %d e %d\n", a, b, c);
}
```

Linguagem C

- Observe agora que os valores das variáveis **a**, **b** e **c** foram modificados na função **alterar**. Por quê?
- O tipo de passagem de parâmetros utilizado é **por referência**. Ou seja, são passados os endereços das variáveis **a**, **b**, e **c** para os ponteiros **x**, **y** e **z**.

Escopo: função **main**

Escopo: função **alterar**



***x++ = a++** 2

***y++ = b++** 3

***z++ = c++** 4

Altera os
conteúdos
de **a**, **b** e **c**!

Linguagem C

- Considere que o endereço de **x** é **FFF1**.

```
int x = 1;  
int *a;  
a = &x;
```

- Então, teremos:

```
a = FFF1 (endereço de x)  
*a = 1   (pois *a = x = 1)
```

```
&(*a) = &x = FFF1 = a
```

```
&(*a) ≡ a
```

Linguagem C

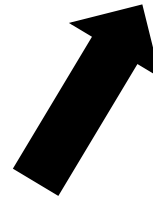
Tipos estruturados

```
struct ponto
{
    float coord_x;
    float coord_y;
};
```



```
struct circulo
{
    float raio;
    struct ponto centro;
};
```

```
struct cilindro
{
    float altura;
    struct circulo base;
};
```



A declaração de variáveis de um tipo estruturado (estruturas) é feita da mesma forma que para um tipo simples.

Linguagem C

- Acesso aos campos de uma estrutura:

```
struct ponto
{
    float coord_x;
    float coord_y;
};
```

```
struct circulo
{
    float raio;
    struct ponto centro;
};
```

```
struct cilindro
{
    float altura;
    struct circulo base;
};
```

```
struct cilindro d;
d.altura = 3.0;
d.base.raio = 5.5;
d.base.centro.coord_x = 1.2;
d.base.centro.coord_y = 3.8;
```

Linguagem C

- O comando **typedef** permite definir um novo nome para um determinado tipo.

```
typedef nome_antigo nome_novo;
```

Exemplos:

```
typedef int inteiro;  
  
inteiro num;
```

```
typedef struct frac  
{  
    int num;  
    int den;  
} frac;
```

```
typedef struct tipo_endereco  
{  
    char rua[50];  
    int numero;  
    char bairro[20];  
    char cidade[30];  
    char sigla_estado[3];  
    long int CEP;  
} TEndereco;
```

Linguagem C

- A definição de um nome para a estrutura simplifica a declaração de variáveis.
- Exemplos:

```
typedef struct ponto
{
    float coord_x;
    float coord_y;
} ponto;
```

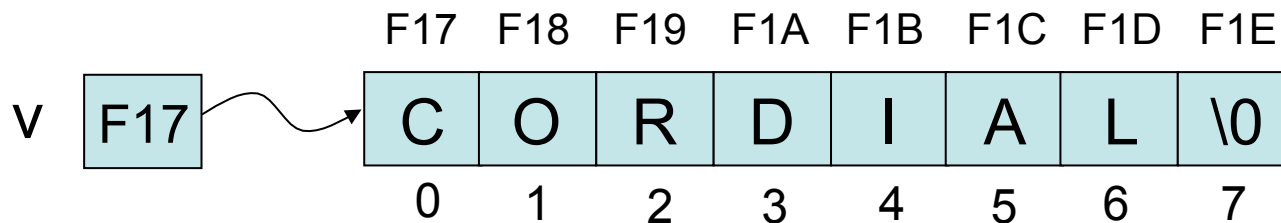
```
typedef struct circulo
{
    float raio;
    ponto centro;
} circulo;
```

```
typedef struct cilindro
{
    float altura;
    circulo base;
} cilindro;
```


Linguagem C

- O nome de um vetor é um ponteiro para sua 1ª posição.

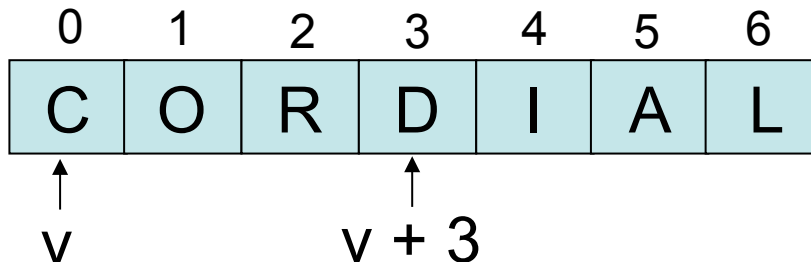
```
char v[8];
```



- Notações equivalentes:

`v[k]`

`*(v + k)`



`v ≡ &v[0]`

`*v ≡ v[0] = 'C'`

`*(v + 3) ≡ v[3] = 'D'`

Linguagem C

- Como o nome de um vetor é um ponteiro, a **passagem de parâmetros** para vetores é sempre **por referência**.

```
void ordenar_por_selecao(int x[], int n)
{
    int menor, pos;
    int i, k = 0;
    ...
```

Na definição da função, podemos substituir **int x[]** por **int *x**.

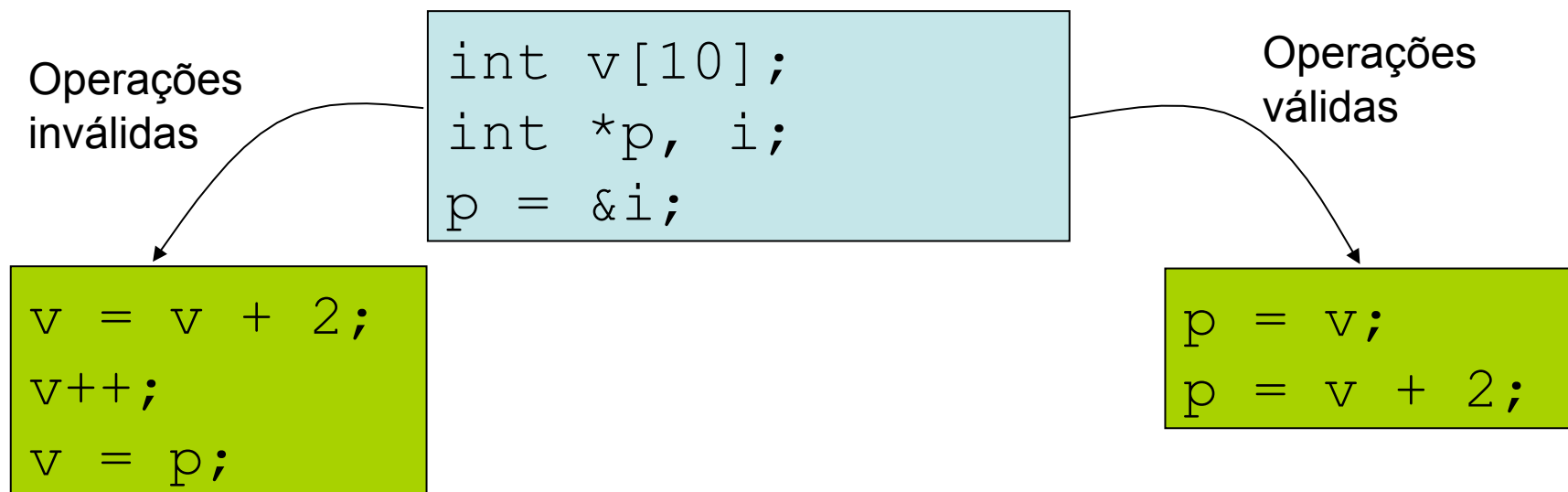
```
// Classificar vetor
ordenar_por_selecao(a, n);
...
```

As alterações realizadas no vetor **x** dentro da função **ordenar_por_selecao** serão também realizadas no vetor **a**.

Linguagem C

- Diferença importante: um ponteiro é uma variável, mas o “nome do vetor” não é uma variável.
- Portanto: não se consegue alterar o endereço que é apontado pelo “nome do vetor”.

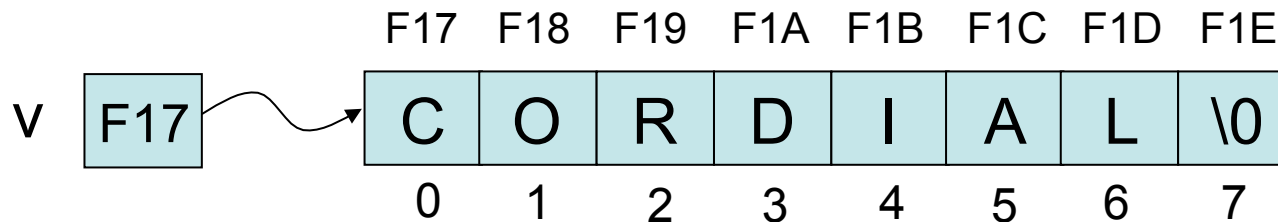
Exemplo:



Linguagem C

- Declaração de vetor:

```
char v[8];
```



- Portanto, uma outra forma de declarar o vetor é:

```
char *v;
```

- A diferença está na **alocação de memória**.
- No primeiro caso, o **compilador** aloca o espaço de memória necessário. No segundo caso, a alocação deverá ser feita em tempo de execução: **alocação dinâmica de memória**.

Linguagem C

- Alocação dinâmica de memória: funções `calloc` ou `malloc`.
- Parâmetros da função `calloc`: número de posições de memória e tamanho em bytes de cada posição.
- Parâmetro da função `malloc`: espaço total em bytes de memória necessário.
- Estas funções retornam um ponteiro do tipo `void` para o início do espaço de memória alocado.
- Exemplo:

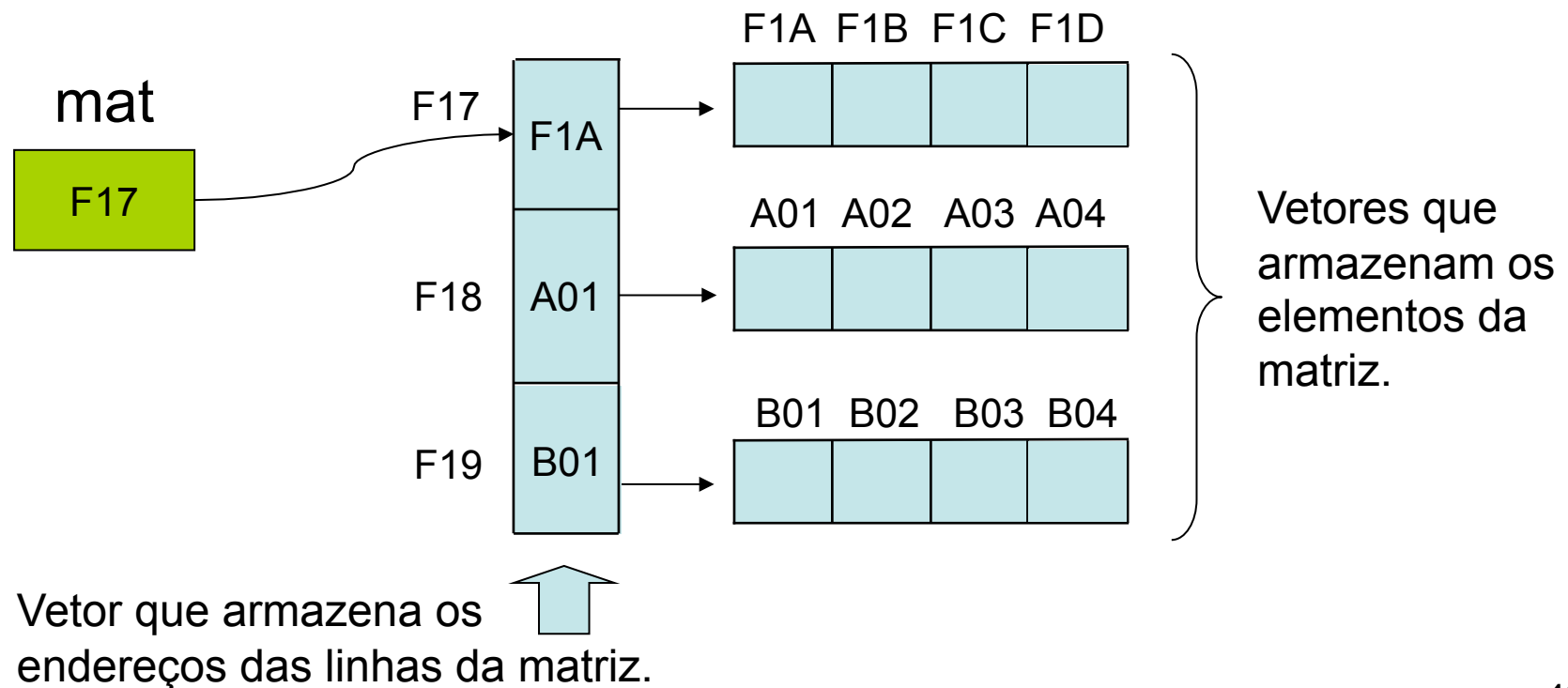
```
int *w;  
w = (int *)calloc(10, sizeof(int));
```

```
int *w;  
w = (int *)malloc(10*sizeof(int));
```

Linguagem C

Alocação de matrizes

- Uma matriz $m \times n$ pode ser imaginada como um vetor de tamanho m , em que cada elemento é um ponteiro para o início de um vetor de tamanho n .
- Exemplo: `int mat[3][4];`



Linguagem C

Para uma matriz de **m** linhas x **n** colunas:

- Inicialmente, alocar as **m** linhas:

```
mat = (int **)calloc(m, sizeof(int *));
```

- Depois, para cada linha (**mat[i]**), alocar um vetor com **n** elementos do tipo **int**.

```
for (i = 0; i < m; i++)  
    mat[i] = (int *)calloc(n, sizeof(int));
```

Linguagem C

- Matriz tridimensional:

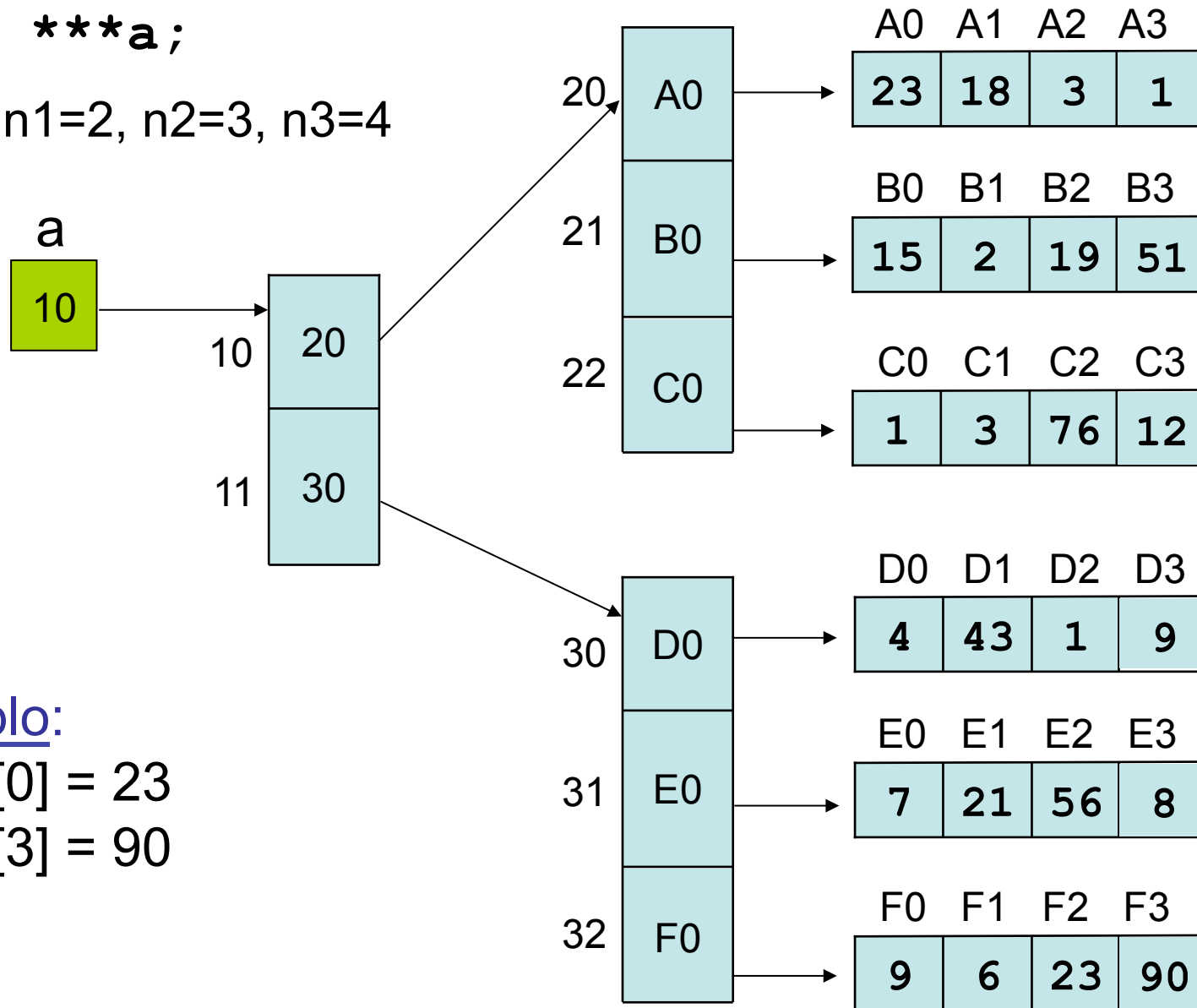
```
float ***a;

a = (float ***)calloc(n1,sizeof(float **));
for (i = 0; i < n1; i++)
{
    a[i] = (float **)calloc(n2,sizeof(float *));
    for (j = 0; j < n2; j++)
        a[i][j] = (float *)calloc(n3,sizeof(float));
}
```


Linguagem C

```
float ***a;
```

Sejam: $n_1=2$, $n_2=3$, $n_3=4$



Exemplo:

`a[0][0][0] = 23`

`a[1][2][3] = 90`

Linguagem C

Arquivos de dados

```
// Programa p32.c
#include <stdio.h>
#include <stdlib.h>

int main(int args, char * arg[])
{
    int i,j,nl,nc;
    int **ma,**mb,**ms;
    FILE *arq;

    arq = fopen("dados32.txt", "r");
    if (arq == NULL)
    {
        printf("Erro ao abrir o arquivo de dados\n");
        system("pause");
        return 1;
    }

    // Leitura das dimensoes a partir do arquivo
    fscanf(arq, "%d %d", &nl, &nc);

    // Alocar memoria para as matrizes
    ma = (int **)calloc(nl,sizeof(int *));
    for (i = 0; i < nl; i++)
        ma[i] = (int *)calloc(nc,sizeof(int));
    mb = (int **)calloc(nl,sizeof(int *));
    for (i = 0; i < nl; i++)
        mb[i] = (int *)calloc(nc,sizeof(int));
```

Linguagem C

- Possíveis modos de abertura de um arquivo:

Modo	Significado
"r"	Abre um arquivo existente para leitura de dados; se o arquivo não existir, irá ocorrer um erro.
"w"	Abre um novo arquivo para gravação de dados; se o arquivo já existir, a gravação irá sobrescrever os dados existentes.
"a"	Abre um arquivo para operações de anexação de dados; se o arquivo não existir, será criado um novo arquivo.

Linguagem C

- Imagine que o arquivo `dados32.txt` contenha os dados dispostos como a seguir:

3	5					← Dimensões das matrizes.
1	3	-2	6	8	} ma	← Linhas das matrizes.
3	-2	0	9	1		
7	2	-4	3	7		
2	7	-3	8	8	} mb	
3	0	-1	9	1		
5	-5	2	0	6		

- No programa, estes dados são lidos pela função `fscanf`. Inicialmente:

```
fscanf(arq, "%d %d", &nl, &nc);
```

Linguagem C

- Depois, a leitura das duas matrizes de **nl** linhas e **nc** colunas é feita, linha por linha, pelas instruções:

```
for (i = 0; i < nl; i++)  
    for (j = 0; j < nc; j++)  
        fscanf(arq, "%d", &ma[i][j]);
```

```
for (i = 0; i < nl; i++)  
    for (j = 0; j < nc; j++)  
        fscanf(arq, "%d", &mb[i][j]);
```

Linguagem C

- Se os dados no arquivo estivessem dispostos como:

3	5										
1	3	-2	6	8	2	7	-3	8	8		
3	-2	0	9	1	3	0	-1	9	1		
7	2	-4	3	7	5	-5	2	0	6		
ma					mb						

a leitura das matrizes seria:

```
for (i = 0; i < nl; i++)
{
    for (j = 0; j < nc; j++)
        fscanf(arq, "%d", &ma[i][j]);
    for (j = 0; j < nc; j++)
        fscanf(arq, "%d", &mb[i][j]);
}
```

Linguagem C

```
void novo_aluno()
{
    int num;
    float n1,n2;
    char nome[20];
    FILE *arq;

    arq = fopen("dados35.txt","a"); ←
    if (arq == NULL)
    {
        printf("Erro ao abrir arquivo\n");
        return;
    }
    printf("\n");
    printf("Digite os dados do novo aluno:\n");
    printf("  Numero ... ");
    scanf("%d",&num);
    printf("  Nome ..... ");
    fflush(stdin);
    gets(nome);
    printf("  Nota 1 ... ");
    scanf("%f",&n1);
    printf("  Nota 2 ... ");
    scanf("%f",&n2);
    fprintf(arq,"%d,%s,%.1f,%.1f\n",num,nome,n1,n2); ←
    fclose(arq);
}
```

Note que, nesta função, o arquivo é aberto com "a", ou seja, no modo **"append"** (que permite acrescentar novos dados ao arquivo).

Aqui, os novos dados são acrescentados ao arquivo.

Linguagem C

- A gravação dos dados no arquivo é feita pela função `fprintf`:

```
fprintf(arq, "%d\n", n);  
for (i = 0; i < n; i++)  
{  
    for (j = 0; j < n; j++)  
        fprintf(arq, "%2d ", D[i][j]);  
    fprintf(arq, "\n");  
}
```

- As funções `fscanf` e `fprintf` são generalizações das funções `scanf` e `printf` (que só podem ser usadas para os dispositivos padrões: `stdin` e `stdout`).

<code>scanf(...)</code>	→	<code>fscanf(stdin, ...)</code>
<code>printf(...)</code>	→	<code>fprintf(stdout, ...)</code>