LINGUAGENS DE PROGRAMAÇÃO II

ESTRUTURAS DE DADOS ESTÁTICAS

Estruturas Estáticas

- Podem ser de dois tipos: homogêneas ou heterogêneas.

- As estruturas homogêneas, como o nome indica, só podem armazenar dados de mesmo tipo. São os vetores e matrizes.

- As estruturas heterogêneas são capazes de armazenar um conjunto de dados de tipos diferentes. São as fichas, ou structs.

VETORES

Vetores

- Em muitas aplicações queremos trabalhar com conjuntos de dados que são semelhantes em tipo.

- Por exemplo o conjunto das alturas dos alunos de uma turma, ou um conjunto de seus nomes. Nestes casos, seria conveniente poder colocar estas informações sob um mesmo conjunto, e poder referenciar cada dado individual deste conjunto por um índice.

- Em programação, este tipo de estrutura de dados é chamada de **vetor** (ou *array*, em inglês) ou, de maneira mais formal estruturas de dados estáticas e homogêneas.

Vetores

- A maneira mais simples de entender um vetor é através da visualização de um **lista**, de elementos com um nome coletivo e um índice de referência aos valores da lista.

N	nota
0	8.4
1	6.9
2	4.5
3	4.6
4	7.2

- Nesta lista, N representa um número de referência e nota é o nome do conjunto. Assim podemos dizer que a 2ª nota é 6.9 ou representar nota[1] = 6.9

Declaração de Vetores

- Um vetor é um conjunto de variáveis de um **mesmo tipo** que possuem um nome identificador e um índice de referência.

- **Sintaxe:**

```
tipo nome[tam];
```

- onde:
  - *tipo* é o **tipo** dos elementos do vetor: int, float, double ...
  - *nome* é o **nome** identificador do vetor. As regras de nomenclatura de vetores são as mesmas usadas em variáveis (seção 2.2.1).
  - *tam* é o tamanho do vetor, isto é, o número de elementos que o vetor pode armazenar.

Declaração de Vetores

- **Exemplos:**

```
int idade[100]; // declara o vetor de nome 'idade',
                // tipo int,
                // com 100 elementos.
float nota[25]; // declara o vetor 'nota',
                // tipo float, de 25 elementos.
char nome[80];  // declara o vetor 'nome',
                // tipo char, com 80 caracteres.
```

- Na declaração de um vetor estamos reservando, em tempo de compilação, espaço de memória para seus elementos. A quantidade de memória (em *bytes*) usada para armazenar um vetor pode ser calculada como:
  - *quantidade de memória = tamanho do tipo * tamanho do vetor*

- Nos exemplos acima, a quantidade de memória utilizada pelos vetores é, respectivamente: 200(2x100), 100(4x25) e 80(80x1) *bytes*.

Referência aos Elementos do Vetor

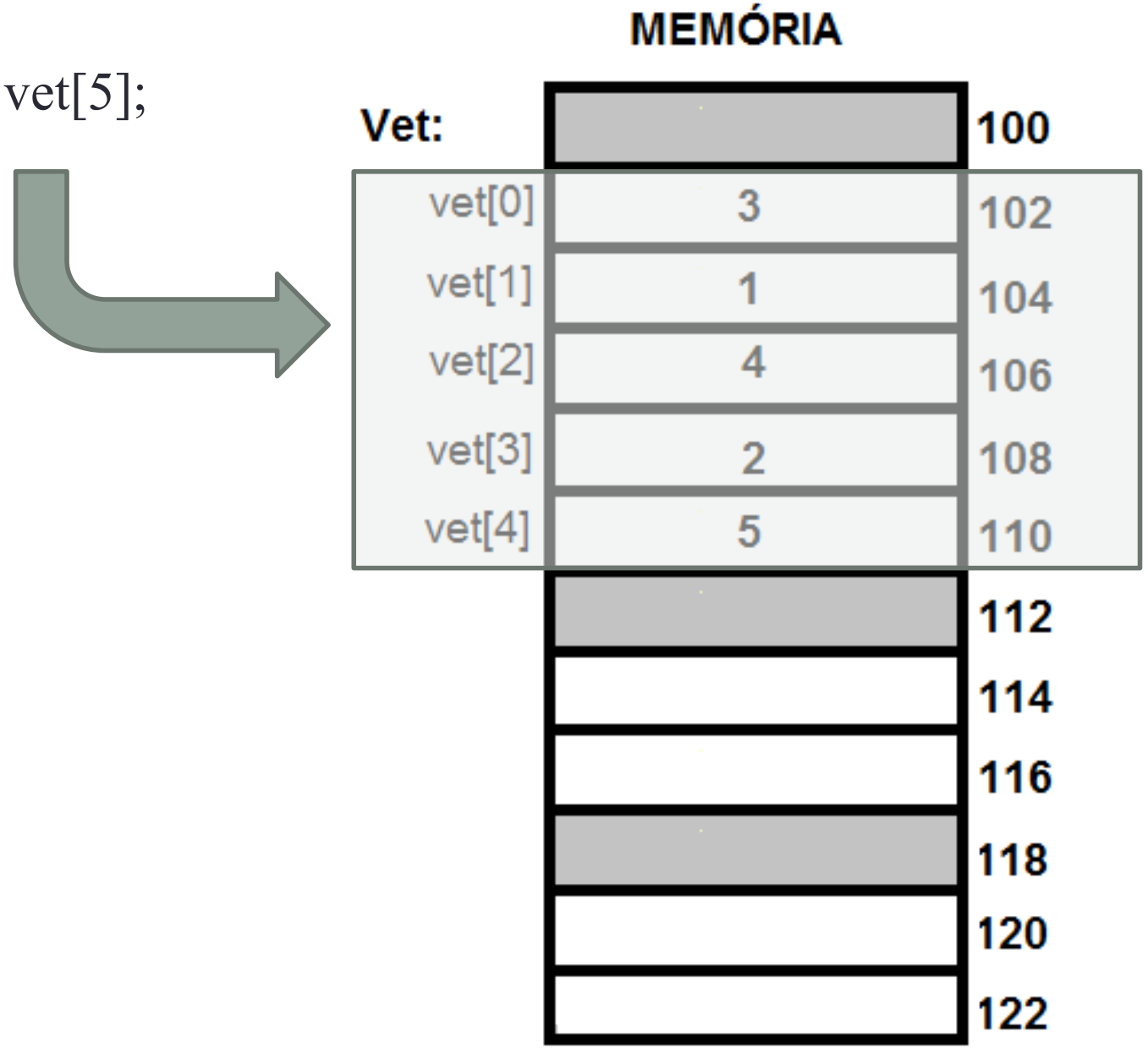
- Cada elemento do vetor é referenciado pelo nome do vetor seguido de um índice inteiro.
- O primeiro elemento do vetor tem índice 0 e o último tem índice tam-1, onde o índice de um vetor deve ser um número inteiro.
- Exemplos:

```
#define MAX 5


main() {
      int i = 7;
      float valor[MAX+1];       // declaração de vetor

      valor[1] = 6.645;
      valor[MAX] = 3.867;
      valor[i] = 7.645;              // errado, mas vai passar
      valor[sqrt(MAX)] = 2.705;      // NÃO é válido!
}
```

int vet[5];


MEMÓRIA
Vet:
100
vet[0]
3
102
vet[1]
1
104
vet[2]
4
106
vet[3]
2
108
vet[4]
5
110
112
114
116
118
120
122

Inicialização de Vetores

- Assim como podemos inicializar variáveis na declaração (por exemplo: int j = 3;), podemos inicializar vetores.

- Sintaxe:

```
tipo nome[tam] = {lista de valores};
```

  - onde:
    - *lista de valores* é uma lista, separada por vírgulas, dos valores de cada elemento do vetor.

- Exemplos:

```
int dia[7] = {12,30,14,7,13,15,6};
float nota[5] = {8.4,6.9,4.5,4.6,7.2};
char vogal[5] = {'a', 'e', 'i', 'o', 'u'};
```

Inicialização de Vetores

- Também podemos inicializar os elementos do vetor dentro do corpo do programa, enumerando-os um a um.

- Exemplo:

```
int cor_menu[4] = {BLUE,YELLOW,GREEN,GRAY};
    ou
int cor_menu[4];
cor_menu[0] = BLUE;
cor_menu[1] = YELLOW;
cor_menu[2] = GREEN;
cor_menu[3] = GRAY;
```

Limites

- Atenção:
  - Na linguagem C, devemos ter cuidado com os limites de um vetor. Embora o tamanho de um vetor esteja definido na sua declaração, o compilador não faz nenhum teste de verificação de acesso.
  - Por exemplo se declaramos um vetor como int valor[5], teoricamente só tem sentido usarmos os elementos valor[0], ..., valor[4]. Porém, o compilador não acusará erro se usarmos valor[12] em algum lugar do programa. Estes testes de limite devem ser feitos logicamente dentro do programa.

Tamanho Parametrizado

- Na linguagem C não é possível declarar um vetor com tamanho variável.

- **Exemplo:**

```
...
int num;
puts("Quantos números?");
scanf("%d", &num);
float valor[num];          // declaração de vetor (errado!)
...
```

- Mas é possível declarar um vetor com tamanho parametrizado, apenas se usarmos uma constante, declarada com a diretiva #define no cabeçalho do programa.

- Deste modo podemos alterar o número de elementos do vetor antes de qualquer compilação do programa.

Passando Vetores para Funções

- Vetores podem ser usados como argumentos de funções. Vejamos como se declara uma função que recebe um vetor e como se passa um vetor para uma função.

- Na chamada a funções usamos a seguinte sintaxe para passar vetores como parâmetros:

```
nome_da_função(nome_do_vetor)
```

- onde:
  - *nome_da_função* é o nome da função que se está chamando.
  - *nome_do_vetor* é o nome do vetor que queremos passar. Indicamos **apenas** o nome do vetor, **sem** índices.

Passando Vetores para Funções

- Na declaração de funções que recebem vetores, usamos a sintaxe:

```
tipo_função nome_função(tipo_vetor nome_vetor[]){
...
}
```

- onde:
  - *tipo_função* é o tipo de retorno da função.
  - *nome_função* é o nome da função.
  - *tipo_vetor* é o tipo de elementos do vetor.
  - *nome_vetor* é o nome do vetor. Observe que depois do nome do vetor temos um índice vazio [] para indicar que estamos recebendo um vetor.

Passando Vetores para Funções

- **Exemplos:**
- Na declaração da função:

```
float media(float vetor[],float N){
        ...
}
```

- Na chamada da função:

```
void main(){
float valor[MAX];              // declaração do vetor
...
med = media(valor, n);         // passagem do vetor
...
}
```

Passando Vetores para Funções

- Ao contrário das variáveis comuns, o conteúdo de um vetor pode ser modificado pela função chamada.
- Isto ocorre porque a passagem de vetores para funções é feita de modo especial dito passagem por endereço.
- Portanto devemos ter cuidado ao manipularmos os elementos de um vetor dentro de uma função para não modifica-los por descuido.

MATRIZES

Matrizes

- Vetores podem ter mais de uma dimensão. Podemos ter vetores de duas, três, ou mais dimensões e iremos chama-los de matrizes.

- **Exemplo:** Uma matriz bidimensional 5x3 pode ser visualizada através de uma tabela.

nota	0	1	2
0	8.4	7.4	5.7
1	6.9	2.7	4.9
2	4.5	6.4	8.6
3	4.6	8.9	6.3
4	7.2	3.6	7.7

- Nesta tabela representamos as notas de 5 alunos em 3 matérias. Nota é o nome do conjunto e podemos dizer que a nota do 3º aluno na 2ª prova é 6.4 ou representar nota[2,1] = 6.4

Declaração e Inicialização de Matrizes

- A declaração e inicialização de matrizes é feita de modo semelhante aos vetores.

- **Sintaxe:**

```
tipo
nome[tam_1][tam_2]...[tam_N]={{lista},{lista},...{lista}};
```

- onde:
  - tipo é o tipo dos elementos do vetor.
  - nome é o nome do vetor.
  - [tam_1][tam_2]...[tam_N] é o tamanho de cada dimensão do vetor.
  - {{lista},{lista},...{lista}} são as listas de elementos.

Declaração e Inicialização de Matrizes

- **Exemplo:**

```
float nota[5][3] = {{8.4,7.4,5.7}, {6.9,2.7,4.9},
     {4.5,6.4,8.6}, {4.6,8.9,6.3}, {7.2,3.6,7.7}};
int tabela[2][3][2] = {{{10,15}, {20,25}, {30,35}},
                       {{40,45}, {50,55}, {60,65}}};
```

- nota é um vetor **duas** dimensões ([][]), composto por 5 vetores de 3 elementos cada.

- tabela é um vetor de três dimensões ([][][]), composto de 2 vetores de 3 sub-vetores de 2 elementos cada.

Passagem de Matrizes para Funções

- A sintaxe é semelhante a passagem de vetores : chamamos a função e passamos o nome da matriz, sem índices. A única mudança ocorre na declaração de funções que recebem matrizes:

- **Sintaxe:** Na *declaração* de funções que recebem vetores:

```
tipo_f função(tipo_v
vetor[tam_1][tam_2]...[tam_n]){
...
}
```

- Observe que depois do nome do vetor temos os índices com contendo os tamanhos de cada dimensão do vetor.

Passagem de Matrizes para Funções

- **Exemplo:**
- Na declaração da função:

```
int max(int vetor[5][7],int N, int M){
}
```

- Na chamada da função:

```
void main(){
  int valor[5][7];              // declaração do vetor
  ...
  med = media(valor, n, m); // passagem do vetor
  ...
}
```

Observações

- Do mesmo modo que vetores, as matrizes podem ter seus elementos modificados pela função chamada.

- Os índices das matrizes, também começam em 0. Por exemplo: mat[0][0], é o primeiro elemento da matriz.

- Usando a analogia da tabela e para matrizes de duas dimensões, podemos entender o primeiro índice da matriz como o número da linha da tabela e o segundo índice como o número da coluna.

Exemplo:

1. Leia um vetor de 20 posições e mostre-o. Em seguida, troque o primeiro elemento com o último, o segundo com o penúltimo, o terceiro com o antepenúltimo, e assim sucessivamente. Mostre o novo vetor depois da troca.

```
#include <stdio.h>

main() {
        int vet[20];
        int i, aux;

        for (i=0; i<20; i++) {
                printf("\nentre com o %do elemento do vetor: ",i+1);
                //scanf("%d", &vet[i]);  //era para ler, mas fiquei com
                vet[i] = i+1;            // preguiça de digitar...
                printf("%d, ", vet[i]);
        }

        for (i=0; i<10; i++) {
                aux = vet[i];
                vet[i] = vet[19-i];
                vet[19-i] = aux;
        }

        printf("\nO vetor invertido: \n");
        for (i=0; i<20; i++)
                printf("%d, ", vet[i]);

}
```