

vetores e alocação dinâmica

- alguns códigos não precisam armazenar o conjunto de valores, mas, em muitas aplicações, necessita-se armazenar o conjunto de valores na memória da computador para depois efetuar os cálculos em cima destes valores
- a forma mais simples de estruturar um conjunto de dados é por meio dos vetores

DEFINIÇÃO DE UM VETOR: `int v[10];`

- ↳ uma declaração diz que `v` é um vetor de inteiros dimensionado com 10 elementos, ou seja, reservamos um espaço de memória contínuo para armazenar 10 valores inteiros
- ↳ se cada `int` ocupa 4 bytes, a declaração reserva um espaço de memória de 40 bytes.
- ↳ o acesso a cada elemento do vetor é feito por meio de uma indexação da variável `v`.
- ↳ a indexação de um vetor varia de 0 a `n-1`, onde `n` representa a dimensão do vetor.

`V[0]` → acessa o primeiro elemento de `v`
`V[1]` → acessa o segundo elemento de `v`
...
`V[9]` → acessa o último elemento de `v`
`V[10]` → está errado (inversão de memória)

- Para exemplificar o uso de vetores, vamos usar o problema de cálculo da média e variância de um conjunto de valores, vamos considerar que desejamos calcular esses valores para um conjunto de dez números reais.
- no exemplo a seguir, os valores são lidos e armazenados no vetor, depois efetuamos os cálculos da média e da variância sobre o conjunto de valores armazenado

```
● ○ ●
1 /*Cálculo da média e da variância de 10 números*/
2
3 #include <stdio.h>
4
5 int main (void){
6     float v[10]; //declara vetor com 10 elementos
7     float med, var; //variáveis para a média e a variância
8     int i; //variável usada como índice do vetor
9
10 }
11
12 //Leitura dos valores
13 for(i = 0; i < 10; i++) //faz índice variar de 0 a 9
14     scanf("%f", &v[i]); //lê cada elemento do vetor
15
16 //cálculo da média
17 med = 0.0f; //inicializa média com zero
18 for(i = 0; i < 10; i++)
19     med = med + v[i]; //acumula soma de elementos
20 med = med / 10; //calcula a média
21
22 //cálculo da variância
23 var = 0.0f; //inicializa com zero
24 for (i = 0; i < 10; i++)
25     var = var+(v[i]-med)*(v[i]-med); //acumula
26 var = var / 10; //calcula a variância
27
28 //exibição do resultado
29 printf("Media = %f Variância = %f \n", med, var);
30 return 0;
```

- 4 devemos observar que passarmos para a função só tem o endereço de cada elemento do vetor ($\&v[i]$), não desejamos o armazenamento dos valores capturados nos elementos do vetor
- 5 existe uma associação entre os vetores e ponteiros, pois existe a declaração: $\text{int } v[10];$
- 6 o símbolo $\&(vetor)$, é uma constante que representa seu endereço inicial; ou seja: o seu indexação aponta para o primeiro elemento do vetor

- 7 podemos somar e subtrair ponteiros, desde que o valor de ponteiro resultante aponte para dentro da área reservada para o vetor

$v + 0 \rightarrow$ aponta para o primeiro elemento do vetor
$v + 1 \rightarrow$ aponta para o segundo elemento do vetor
$v + 2 \rightarrow$ aponta para o terceiro elemento do vetor
...
$v + 9 \rightarrow$ aponta para o décimo elemento do vetor

Portanto, escrever $\&v[i]$ é equivalente a escrever $(v + i)$. E $v[i]$ é equivalente a escrever $*(v + i)$

↳ a forma indexada é mais clara e adequada.

- 8 os vetores também podem ser inicializados na declaração: $\text{int } v[5] = \{5, 10, 15, 20, 25\};$
- 9 ou simplesmente: $\text{int } v[] = \{5, 10, 15, 20, 25\};$

↳ dimensiona o vetor pelo número de elementos inicializados

passagem de vetores

- 1 passar um vetor para uma função consiste em passar o endereço da primeira posição do vetor
- 2 se passarmos um valor de endereço, a função chamada deve ter um parâmetro do tipo ponteiro para armazenar esse valor
- 3 se passarmos para uma função um vetor int, devemos ter um parâmetro do tipo int^* , capaz de armazenar endereços de inteiros
- 4 os elementos do vetor não são copiados para a função, o argumento aponta é apenas o endereço do primeiro elemento
- 5 para exemplificar, vamos modificar o código anterior, usando funções separadas para o cálculo da média e variância

```

1 //cálculo da média e variância de 10 reais (segunda versão)
2
3 #include <stdio.h>
4
5 //função para cálculo da média que recebe um inteiro n e um ponteiro v real
6 float media(int n, float *v){
7     int i;
8     float s = 0.0f;
9     for(i = 0; i < n; i++)
10         s += v[i];
11     return s/n; //Retorna a média calculada dividindo o valor acumulado s pelo número de elementos n.
12 }
13
14 //Função para cálculo da variância que recebe um variável do tipo inteiro, um ponteiro real e uma variável real
15 float variancia(int n, float *v, float real){
16     int i;
17     float s = 0.0f;
18     for(i = 0; i < n; i++)
19         s += (v[i] - real) * (v[i] - real);
20 }
21
22 int main(void){
23     float v[10]; //Declaração de um vetor com 10 caracteres
24     float med, var;
25     int i;
26     //Entrada dos dados
27     for(i = 0; i < 10; i++)
28         scanf("%f", &v[i]);
29
30     med = media(10, v); //Chama a função media e passa seus atributos
31     var = variancia(10, v, med); //Chama a função variância e passa seus atributos
32     printf("Média = %f\n", med);
33     printf("Variância = %f\n", var);
34     return 0;
35 }
```

- 6 usaremos também os operadores de atribuição $+=$ para acumular os valores
- 7 como se passado para a função o endereço do primeiro elemento do vetor, podemos alterar os valores dos elementos do vetor dentro da função

```

1 //Incrementa elementos de um vetor
2
3 #include <stdio.h>
4
5 //declaração da função incr_vetor que recebe os seguintes atributos: um inteiro n e um ponteiro do tipo inteiro chamada v
6 void incr_vetor(int n, int *v){
7     int i;
8     for (i = 0; i < n; i++)
9         v[i]++;
10 }
11
12 int main (void){
13     int a[3] = {1, 3, 5}; //declara e inicializa um vetor
14     incr_vetor(3, a);
15     printf("%d %d %d\n", a[0], a[1], a[2]); //imprime as posições e a saída é: 2 4 6
16
17     return 0;
18 }

```

alocação de memória

- dimensionar um vetor nos obriga a reservar quanto espaço necessário (tínhamos que prever um número máximo de elementos).
- ↳ essa tri-dimensionamento é um fator limitante
- ↳ se dimensionar um vetor muito alto, ocorre um desperdício de memória
- a linguagem C oferece meios de requisitar espaços de memória em tempo de execução

uso da memória

- existem três maneiras de reservar espaço de memória para o armazenamento de informações:

VARIÁVEIS GLOBAIS (E ESTÁTICAS): espaço reservado existe enquanto o programa estiver sendo executado.

VARIÁVEIS LOCAIS: o espaço só existe apenas enquanto a função que declarou a variável está sendo executada, sendo liberado para outros usos quando a execução da função termina.

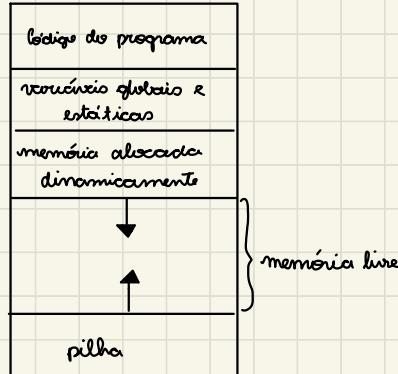
↳ as variáveis globais e locais podem ser simples ou vetores.

ALOCAÇÕES DINÂMICAS: requisitar ao sistema, em tempo de execução, um espaço de determinado tamanho. Esse espaço alocado dinamicamente permanece reservado até que seja explicitamente liberado pelo programa.

↳ por isso, podemos alocar dinamicamente um espaço de memória em uma função e acioná-la em outra.

↳ se o programa não liberar um espaço alocado, ele será automaticamente liberado quando a execução do programa terminar.

A DISTRIBUIÇÃO SERIA:



funções da biblioteca padrão

- existem funções da biblioteca stdlib que permitem alocar e liberar memória dinamicamente

FUNÇÃO MALLOC: recebe como parâmetro o número de bytes que se deseja alocar e retorna o endereço inicial da área de memória alocada.

- int * v;

v = malloc(10*4); → alocação dinâmica de um vetor de inteiros com 10 elementos

para ficarmos independentes de compiladores e máquinas, usamos o operador sizeof()

- v = malloc(10 * sizeof(int));

malloc retorna um ponteiro genérico, para um tipo qualquer, representado por void*, que pode ser convertido automaticamente para o tipo apropriado na atribuição

mas é comum fazer a conversão explicitamente:

- v = (int*) malloc(10 * sizeof(int));

se não houver espaço livre suficiente para realizar a alocação, a função retorna um endereço nulo (NULL)

```
1 v = (int*) malloc(10*sizeof(int));
2 if(v==NULL){
3     printf("Memória Insuficiente.\n");
4     exit(1); //aborda o programa e retorna 1 para o sistema operacional
5 }
```

para liberar um espaço de memória alocado dinamicamente, usamos a função free, que recebe como parâmetro o ponteiro da memória a ser liberada:

free(v);

```
1 //cálculo da média e da variância de n reais
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main (void){
7     int i, n;
8     float *v;
9     float med, var;
10
11    //Leitura do número de valores
12    scanf("%d", &n);
13    //alocação dinâmica
14    v = (float*) malloc(n*sizeof(float));
15
16    if(v==NULL){
17        printf("Memória insuficiente.\n");
18        return 1;
19    }
20
21    //Leitura dos valores
22    for (i = 0; i < n; i++)
23        scanf("%f", &v[i]);
24    med = media(n, v);
25    var = variância(n, v, med);
26    printf("Media = %f Variancia = %f \n", med, var);
27
28    free(v);
29    return 0;
30}
```

- a função `malloc()` aloca um bloco contíguo de memória do tamanho especificado, enquanto a função `calloc()` aloca um bloco contíguo de memória e inicializa todos os bytes para 0.

FUNÇÃO REALLOC: é usada para redimensionar uma área de memória já alocada dinamicamente

- `void * realloc(void * ptr, size_t size);`

↳ é útil quando precisamos alocar mais memória para um vetor ou matriz dinâmica, mas não quer perder os dados armazenados nela ou para liberar espaço desnecessário, se o novo tamanho for menor que o antigo.