

Fundamentos de Programação

Vetores

Dainf - UTFPR

Profa. Leyza B. Dorini
Prof. Bogdan T. Nassu

Estruturas de Dados

- Até agora, temos usado os tipos primitivos de dados: `int`, `float`, `double`, etc.
- Sozinhos, estes tipos não representam de forma eficiente conceitos mais abstratos...
 - Uma imagem.
 - Uma gravação de áudio.
 - A lista de compras de um cliente.
 - O estado de cada personagem em um game.
 - O seu perfil no Facebook.
- Estruturas de dados: grupos de valores de tipos primitivos.
 - Algumas têm propósito geral, outras são bem especializadas.
 - A primeira estrutura que veremos é o vetor (array)

Motivação

Inicialmente, iremos discutir um exemplo, o qual vai ilustrar a importância do tópico de hoje: vetores (*arrays*)!



Exemplo 1: calcular a média de quatro notas

A primeira solução é bem simples, e nem chega a explorar laços de repetição.

```
1 float nota1, nota2, nota3, nota4, media;
2
3 printf("Nota do aluno 1: ");
4 scanf("%f", &nota1);
5 printf("Nota do aluno 2: ");
6 scanf("%f", &nota2);
7 printf("Nota do aluno 3: ");
8 scanf("%f", &nota3);
9 printf("Nota do aluno 4: ");
10 scanf("%f", &nota4);
11
12 media = (nota1 + nota2 + nota3 + nota4)/4;
```

E se fosse necessário calcular a média de 100 notas? A solução acima não é facilmente expandida, além de ser suscetível a erros de digitação!

Exemplo 1: calcular a média de quatro notas

Explorando laços de repetição, não precisamos de uma variável para cada nota. Além disso, agora é trivial expandir a solução para calcular a média de 100 notas, concorda?

```
1 float nota, media=0;
2 int i;
3
4 for(i=0; i<4; i++){
5     printf("Nota do aluno %d: ", i+1);
6     scanf("%f", &nota);
7
8     media = media + nota;
9 }
10
11 media = media / 4;
```

Checkpoint

Como você adaptaria os dois programas anteriores para mostrar na tela os alunos que tiveram notas acima da média?



@peanutsspecials

Exemplo 2: determinar alunxs com nota acima da média!

Como a média só será conhecida após somar todos os valores, não é possível adaptar a solução com laços de repetição!

```
1 float nota, media=0;
2 int i;
3
4 for(i=0; i<4; i++){
5     printf("Nota do aluno %d: ", i+1);
6     scanf("%f", &nota);
7
8     media = media + nota;
9 }
10 media = media / 4;
11
12 //como não armazenamos todas as notas, não conseguimos
    mais comparar quais delas estão acima da média
    calculada...
```

Exemplo 2: determinar alunxs com nota acima da média!

Portanto, precisamos guardar todas as notas :-(Consegue imaginar como ficaria a solução para 100 notas?

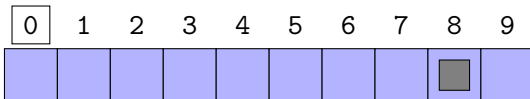
```
1 float nota1, nota2, nota3, nota4, media;
2
3 printf("Nota do aluno 1: ");
4 scanf("%f", &nota1);
5 /*Leitura das notas 2 e 3*/
6 printf("Nota do aluno 4: ");
7 scanf("%f", &nota4);
8
9 media = (nota1 + nota2 + nota3 + nota4)/4;
10
11 if(nota1 > media)
12     printf("Aluno 1 com nota acima da media");
13 /*Comparação das notas 2 e 3*/
14 if(nota4 > media)
15     printf("Aluno 4 com nota acima da media");
```


Para lidar com problemas desse tipo de forma mais efetiva, vamos aprender uma estrutura de dados, chamada vetor (*array*). Ela permite armazenar um conjunto de elementos de tal forma que sua manipulação torna-se mais eficiente em diversos casos.

Definição

Vetor

Coleção de variáveis do mesmo tipo, armazenadas em posições contíguas de memória e referenciadas por um nome comum.

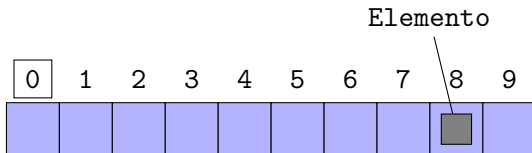


Definição

Vetor

Coleção de variáveis do mesmo tipo, armazenadas em posições contíguas de memória e referenciadas por um nome comum.

- Cada valor armazenado é chamado de *elemento* do vetor.

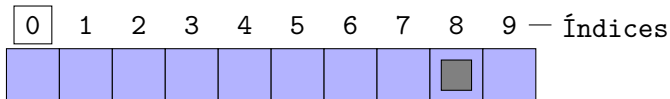


Definição

Vetor

Coleção de variáveis do mesmo tipo, armazenadas em posições contíguas de memória e referenciadas por um nome comum.

- Cada elemento é numerado com um inteiro (*índice*).

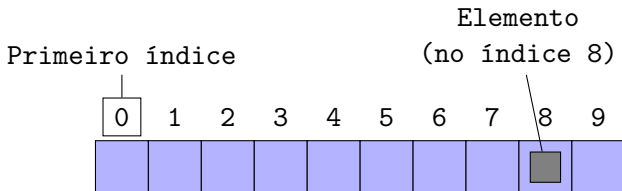


Definição

Vetor

Coleção de variáveis do mesmo tipo, armazenadas em posições contíguas de memória e referenciadas por um nome comum.

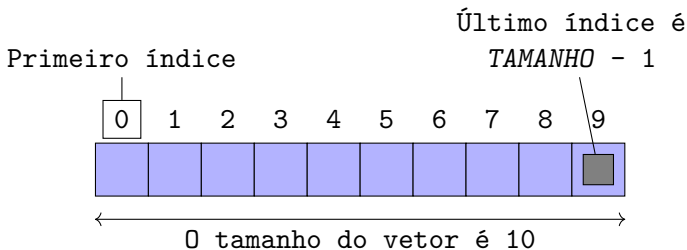
- O acesso aos elementos é realizado por meio do seu índice, que inicia em zero!



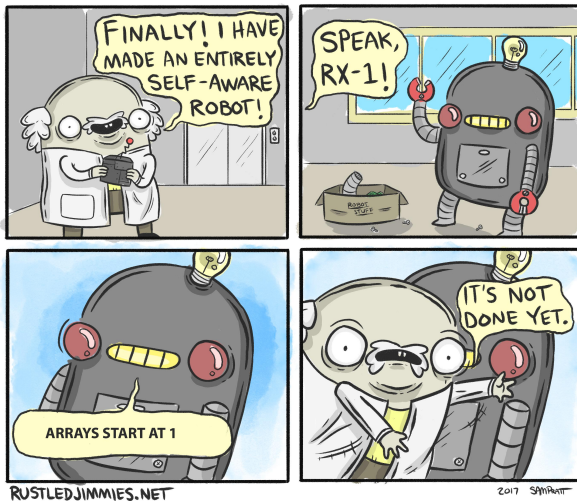
Primeiro índice do vetor é zero!

Importante

A primeira posição de um vetor **tem índice 0!**



Nunca esqueça: o primeiro índice do vetor é zero!



Declarando um vetor

Declarando um vetor

Ao declarar um vetor devemos informar, além do tipo dos elementos que serão armazenados, a sua dimensão, isto é, o número máximo de elementos que poderá ser armazenado no espaço de memória que é reservado para o vetor:

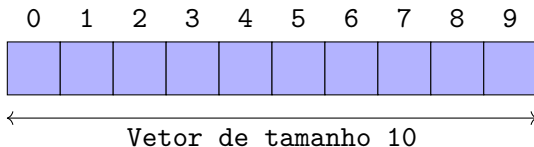
<tipo> identificador [<número de posições>**];**

No C ANSI, o tamanho (número de posições) precisa ser uma constante inteira. Em alguns compiladores, pode ser um parâmetro de função. Mas é bom evitar.

Declarando um vetor

Exemplo

```
float notas[10];
```



Um vetor com N valores de um tipo que ocupa B bytes na memória ocupa $N*B$ bytes na memória. Exemplo:

```
int foo [50]; // 50*4 = 200 bytes.
```

Checkpoint

Faça um programa que declare dois vetores de inteiros (um de 5 posições e outro de 7 posições) e um vetor do tipo float de tamanho 10.

```
1 int main(){  
2  
3     int v1[5], v2[7];  
4     float v3[10];  
5  
6     return 0;  
7 }
```

Como são do mesmo tipo, podem ser declarados assim (da mesma forma que fazemos para variáveis).

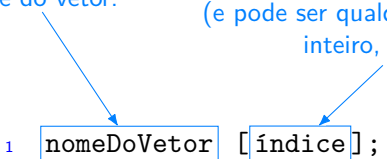
A declaração do vetor deve ser sempre acompanhada do seu tamanho entre [].

Acessando os elementos

Acessando os elementos um vetor

A sintaxe para acessar um determinado elemento de um vetor é:

Nome do vetor. Índice do elemento que deseja acessar (entre colchetes). Precisa ser um valor inteiro (e pode ser qualquer expressão com resultado inteiro, incluindo variáveis).



```
1 nomeDoVetor [índice];
```

Cada valor de um vetor se comporta como se fosse uma variável. Além disso, lembre-se que o valor inicial de todas as posições é indeterminado.

Cuidado com acesso a índices fora do vetor

Atenção!

Tentar acessar índices fora de um vetor implica em comportamento indefinido! O compilador **NÃO AVISA** quando isto acontece.

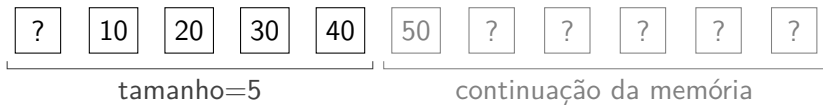
Possíveis consequências (em ordem de severidade):

- Um erro durante a execução.
- Um erro estranho na saída do programa.
- Nenhum erro detectável (até ser tarde demais).
- Um bug que pode ser explorado por invasores!!!

Depois assista ao vídeo sobre *buffer overflow*!

Cuidado com acesso a índices fora do vetor

Portanto, cuidado para não acessar posições inválidas!!!



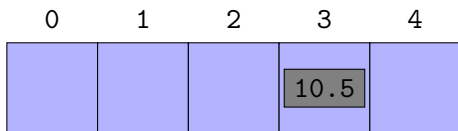
```
1 int main(){
2     int v[5], i;
3
4     for(i=1; i<=5; i++){ //note que i vai de 1 até 5
5         v[i] = i*10;
6     }
7 }
```

De novo: para entender melhor o que acontece, veja o vídeo sobre *buffer overflow* disponibilizado pelos professores.

Armazenando valores em um vetor

Armazenando valores em um vetor

Por exemplo, suponha um vetor chamado `preco`, do tipo `float` e de tamanho 5. Como armazenar o valor 10.5 no índice 3?



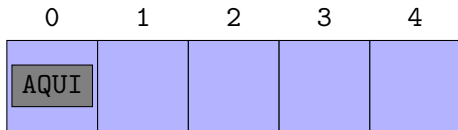
```
1  int main(){
2
3      float preco[5];
4
5      preco[3] = 10.5;
6
7      return 0;
8  }
```

Nome do vetor.

Índice do elemento
que deseja acessar
(entre colchetes).

Armazenando valores em um vetor

E se fosse necessário armazenar um valor lido do teclado **na primeira posição**?



```
1 int main(){  
2  
3     float preco[5];  
4  
5     scanf(" %f ", &preco[0]);  
6  
7     return 0;  
8 }
```

Especificador de
formato compatível com
o tipo do vetor.

Endereço da posição do
vetor que deseja acessar.

Armazenando valores em um vetor

E para ler do teclado valores em todas as posições?


```
1  int main(){
2      float preco[5];
3
4      scanf("%f", &preco[0]);
5      scanf("%f", &preco[1]);
6      scanf("%f", &preco[2]);
7      scanf("%f", &preco[3]);
8      scanf("%f", &preco[4]);
9
10     return 0;
11 }
```

Observe que a única alteração é o índice acessado. Portanto, podemos explorar o uso de estruturas de repetição.

Armazenando valores em um vetor

Com uma estrutura de repetição, o programa fica mais robusto.

```
1  int main(){
2      float preco[5];
3      int i;
4
5      for(i=0; i<5; i++){
6          scanf("%f", &preco[i]);
7      }
8
9      return 0;
10 }
```



Acessa o índice *i*,
com *i* variando
de zero até 4.

Armazenando valores em um vetor

Caso seja necessário atribuir valores já na declaração, é possível. Se não for informado, o tamanho é definido automaticamente. A sintaxe é:

```
1 float v[] = {4, 5, 7, 8, 2};
```

0	1	2	3	4
4	5	7	8	2

Atenção

A inicialização desta forma só é possível no momento da declaração, ou seja, **não é** possível fazer:

```
v[] = {4, 5, 7, 8, 2};
```

Armazenando valores em um vetor

Para facilitar testes, podemos atribuir valores aleatórios! **Tem material só sobre esse assunto!** Depois estude-o com atenção!

```
1 int main(){
2     float preco[5];
3     int i;
4
5     srand(time(NULL));
6
7     for(i=0; i<5; i++){
8         preco[i] = rand()%10;
9     }
10
11     return 0;
12 }
```

Inicialização da semente
(senão o programa vai gerar
sempre os mesmos valores).

Atribui valores entre 0 e 9.

Imprimindo valores em um vetor

A lógica é exatamente a mesma da leitura em todas as posições!!

```
1  int main(){
2      float preco[5];
3      int i;
4
5      for(i=0; i<5; i++){
6          scanf("%f", &preco[i]);
7      }
8
9      for(i=0; i<5; i++){
10         printf("%f ", preco[i]);
11     }
12
13     return 0;
14 }
```

Acessa o índice *i*,
com *i* variando
de zero até 4.

Exemplo

```
1 #define BUFLEN 8
2
3 int main ()
4 {
5     int foo[BUFLEN], bar[8], i;
6
7     for (i = 0; i < BUFLEN; i++)
8         foo[i] = BUFLEN-i-1;
9
10    for (i = 0; i < BUFLEN; i++)
11        foo[foo[i]] = 1;
12
13    for (i = 0; i < BUFLEN/2; i++)
14        bar[i] *= (i + foo[BUFLEN-i-1]);
15
16    return 0;
17 }
```

No C ANSI, o tamanho precisa ser uma constante inteira.

Podemos indexar o vetor com qualquer inteiro, inclusive com um valor que está no próprio vetor.

Atenção

Ao usar vetores, procure soluções eficientes e robustas (que possam ser facilmente expandidas para dimensões maiores).



Boa prática: uso de constantes (macros) para definir o tamanho do vetor

Como o tamanho é usado sempre que precisamos percorrer o vetor todo, uma boa prática consiste em utilizar uma contante para definir tal tamanho.

Usando constantes para definir o tamanho de vetores

```
1  int main(){
2      float preco[10]:
3      int i;
4
5      for(i=0; i<10; i++){
6          scanf("%f", &preco[i]);
7      }
8
9      return 0;
10 }
```

Note que o tamanho do vetor também determina a variação do laço de repetição.

Usando constantes para definir o tamanho de vetores

Ao usar uma constante, evitamos inconsistências como a ilustrada abaixo:

```
1  int main(){
2      float preco[5];
3      int i;
4
5      for(i=0; i<10; i++){
6          scanf("%f", &preco[i]);
7      }
8
9      return 0;
10 }
```

Note que o tamanho do vetor foi alterado, mas o limite da variação do laço de repetição não.

Desta forma, posições indevidas da memória serão acessadas.

Usando constantes para definir o tamanho de vetores

```
1  #define N 10
2
3  int main(){
4      float preco[N];
5      int i;
6
7      for(i=0; i<N; i++){
8          scanf("%f", &preco[i]);
9      }
10
11     for(i=0; i<N; i++){
12         printf("%f", preco[i]);
13     }
14
15     return 0;
```

Ao alterar o valor da constante...

... automaticamente o restante do programa fica com valores consistentes.

Observações importantes

Vetor de tamanho dinâmico

Na linguagem C, **não é correto** criar um vetor de tamanho “dinâmico” da seguinte forma:

```
1  int main(){
2      int n;
3
4      printf("Qual o tamanho do vetor?");
5      scanf("%d", &n);
6
7      int v[n];    ///// NÃÃÃÃÃÃÃO....
8
9      //"Roda", mas é não é a forma correta...
10 }
```

Logo iremos aprender a fazer isso da forma correta: alocação dinâmica!

Professora, mas tá rodando...



Lembre-se: não é porque roda que “está certo”!

Vetores com tamanho pré-definido

- Por enquanto, vamos usar vetores com tamanho pré-definido, ou seja, o tamanho não pode ser mudado durante a execução.
- Conforme já mencionado, no C ANSI o tamanho precisa ser uma constante inteira. Em alguns compiladores, pode ser um parâmetro de função. Mas é bom evitar.
- Não precisamos usar todas as posições de um vetor:
 - Podemos declarar um vetor e usar uma variável para controlar quantos elementos realmente estamos usando.
 - Ao fazer isso, tome cuidado para não acessar posições indevidas.

Veja o exemplo na próxima página.

Exemplo: armazenar n (≤ 100) notas

Se você quiser ler n valores, em que n é determinado pelo usuário em tempo de execução, a estratégia é:

```
1  #define MAX_N 100
2  int main () {
3      float nota [MAX_N]; // Vetor definido com tamanho constante.
4      int n, i;
5
6      // Leitura de um n válido, i.e., positivo e <= MAX_N.
7      printf("Numero de alunos: ");
8      scanf ("%d", &n);
9      while ((n < 0) || (n > MAX_N))
10         scanf ("%d", &n);
11
12     // O loop vai só até n, não até MAX_N.
13     for (i = 0; i < n; i++) {
14         printf ("Nota do aluno %d: ", i+1);
15         scanf ("%f", &nota[i]);
16     }
17     return 0;
18 }
```

Atribuições inválidas

Cuidado!!!

É errado fazer:

- `int v[5], aux = v;` // neste caso, o vetor `v` **NÃO** será copiado para a variável `aux`!!
- `int v = 10;` // neste caso, o vetor `v` **NÃO** receberá 10 em todas as posições!!

O identificador do vetor na verdade é um ponteiro para a primeira posição de memória alocada. Por esta razão:

- Você não vai comparar dois vetores diretamente fazendo `if(vetor1 == vetor2)`.
- Os índices entre colchetes são um “deslocamento” a partir da primeira posição.

Finalizando...

- “Para quem tem um martelo, todo problema se parece com um prego.”
- Você não precisa (nem deve) usar vetores em todo e qualquer programa. Muitas vezes, existe uma solução mais simples e mais barata computacionalmente que não depende de vetores!
- Quando usar vetores, então? Por exemplo,
 - Quando precisamos armazenar todos os valores.
 - Quando não armazenar os valores implica em refazer muitos cálculos.

Agora é contigo: faça os exercícios da lista!