Disciplina: Introdução a Programação

Aula 9: Estruturas de dados homogêneas unidimensionais

Apresentação

Na última aula, vimos a estrutura de repetição **para**, ou **for**, que é muito utilizada quando se sabe, de antemão, o número de iterações a serem realizadas, ou seja, quantas vezes será necessário repetir o trecho definido dentro da estrutura.

Essa estrutura inclui um contador em sua definição que será automaticamente incrementado ou decrementado. Normalmente, esta é a estrutura de repetição escolhida quando precisamos trabalhar com **vetores**, um novo tipo de variável que abordaremos nesta aula.

Bons estudos!

Objetivos

- Definir os vetores e sua funcionalidade;
- Avaliar o comportamento de algoritmos e programas com vetores;
- Escrever códigos com vetores como recurso de armazenamento.

Vetores

Ao longo de todas as aulas que tivemos até o momento, trabalhamos com variáveis. As variáveis são espaços da memória reservados ao armazenamento temporário de valores. Cada espaço pode armazenar somente um valor de determinado tipo por vez.

Fonte: Shutterstock.

Mas, e se você precisar armazenar vários dados, como o nome de 45 alunos de uma turma?

Seria necessário criar 45 variáveis diferentes, uma para cada nome?

Parece um tanto inviável, não é mesmo?

Para casos nos quais vários dados de um mesmo tipo precisam ser armazenados, como no exemplo do nome dos alunos, é possível recorrer a um tipo especial de variável: os vetores.

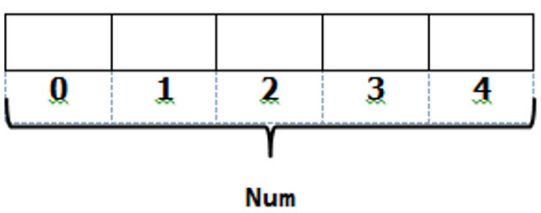
Um vetor é uma variável e, portanto, também armazena dados em caráter temporário. A diferença é que o vetor é capaz de armazenar um conjunto de dados do mesmo tipo, todos sob o mesmo identificador. Trata-se, então, de uma série de elementos de um mesmo tipo, armazenados em espaços contíguos da memória principal, que podem ser individualmente referenciados através de um índice.

Vejamos um exemplo:

Exemplo

Se você precisa armazenar cinco valores do tipo inteiro, ao invés de criar cinco variáveis diferentes – uma para cada valor, você pode criar um vetor para cinco elementos. Esse vetor reserva espaços adjacentes da memória (um ao lado do outro) e todos os valores armazenados poderão ser acessados por um mesmo identificador (nome da variável). O identificador precisa estar acompanhado de um número denominado **índice**, que indica a posição do vetor a ser acessada.

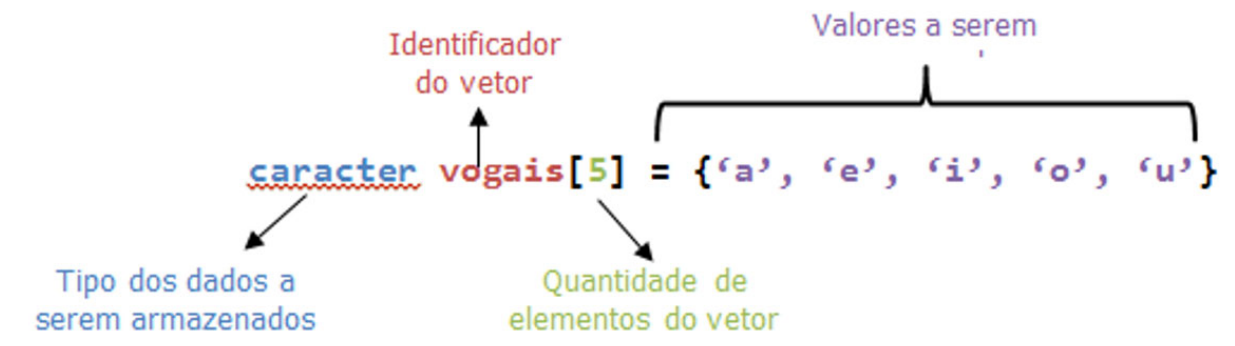
Caso você precise criar o vetor descrito no parágrafo anterior, capaz de armazenar cinco elementos inteiros, e deseje chamá-lo de **Num**, poderíamos representá-lo graficamente da seguinte maneira:



Cada retângulo representa um elemento do vetor e os números de 0 a 4 representam os índices das posições do vetor. A posição 0 é a primeira, e a posição 4 é a última. Cada um desses elementos pode armazenar um valor do tipo inteiro, caso o vetor seja declarado como sendo desse tipo.

Os vetores são estruturas de dados homogêneas, o que significa que são capazes de armazenar um conjunto de valores de um mesmo tipo.

Para declarar um vetor no Portugol Studio, é preciso informar seu tipo, seu identificador, seu tamanho e, opcionalmente, os valores que ele irá armazenar. Observe o exemplo, no qual é criado um vetor denominado **vogais**, capaz de armazenar cinco elementos do tipo caracter:



Atenção

Tanto no Portugol Studio quanto no C++, a definição do número máximo de elementos do vetor deve ser estar entre colchetes [ ]; o conjunto de elementos a serem armazenados no vetor deve estar definido entre chaves { }.

A declaração de vetores no C++ segue a mesma sintaxe do Portugol Studio. É preciso informar seu tipo, seu identificador, seu tamanho e, opcionalmente, os dados que nele serão armazenados. Observe o exemplo de criação do vetor **vogais**:

```
char vogais[5] = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’};
```

Conforme dito anteriormente, não é obrigatório armazenar valores no vetor no momento de sua criação. Caso queira somente criar o vetor, pare a definição antes do sinal de atribuição. Veja:

*** Portugol Studio**

```
caracter vogais[5] // Criação do vetor vogais, para 5 caracteres
inteiro num[10] // Criação do vetor num, para 10 números inteiros
real notas[35] // Criação do vetor notas, para 35 números reais
```

*** C++**

```
char vogais[5]; // Criação do vetor vogais, para 5 caracteres
int num[10]; // Criação do vetor num, para 10 números inteiros
float notas[35]; // Criação do vetor notas, para 35 números reais
```

É muito importante que você saiba que, para acessar as diferentes posições do vetor para armazenamento ou recuperação de dados, é imprescindível informar seu índice. Veja os exemplos a seguir:

*** Portugol Studio**

```
//Armazena a letra “E” na segunda posição do vetor vogais
vogais[1] = ‘E’
//Armazena o número 35 na quinta posição do vetor num
num[6] = 35
//Armazena o número 8.5 na vigésima posição do vetor notas
notas[19] = 8.5
```

*** C++**

```
//Exibe a primeira posição do vetor vogais
cout <<vogais[0];
//Armazena em RESULT a soma dos dois primeiros valores do vetor num
RESULT = num[0]+num[1];
//Armazena o valor informado via teclado na quarta posição do vetor notas
cin >>notas[3];
```

Dica

Lembre-se de que tanto no Portugol Studio quanto no C++, a primeira posição do vetor é identificada pelo índice **0**. Assim, um vetor de 15 elementos varia da posição 0 à posição 14.

Atividade

1 - Leia os enunciados a seguir e crie os algoritmos que solucionem os problemas propostos. Siga a sintaxe do Portugol Studio:

a) Dois amigos programadores estão entediados e decidiram criar um jogo simples. A ideia é que o primeiro jogador preencha um vetor de 10 números inteiros. Em seguida, o segundo jogador tem três tentativas para tentar adivinhar um dos números digitados. Se ele acertar, recebe a mensagem “Parabéns! Este número está na posição XX do vetor! Você usou YY tentativas...”; onde XX deve ser substituído pela posição do vetor onde se encontra o número, e YY deve ser substituído pelo número de vezes que o usuário tentou acertar o número. Se errar em todas as tentativas, a mensagem a ser exibida é “Que pena! Você não acertou...”.

b) Um dado é lançado 20 vezes e, a cada lançamento, a face sorteada é lançada em um vetor. Ao final dos sorteios, informe quantas vezes cada face ímpar foi sorteada.

c) Um casal de amigos está brincando de par ou ímpar. A cada uma das 10 jogadas que farão, armazene em um vetor o número escolhido pelo jogador A (que escolheu par), e em um segundo vetor, o número escolhido pelo jogador B (que escolheu ímpar). Também a cada jogada, informe o jogador vencedor. Ao final, informe quantas vezes cada jogador venceu.

d) Você está tentando métodos bastante rudimentares de criptografia para que seus dados fiquem seguros. Em um deles, você recebe os elementos de um vetor de 20 letra e as transfere para outro vetor no qual as letras estarão armazenadas na ordem inversa em que foram recebidas. Observe:

VETOR 1:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

VETOR 2:

T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

2 - Com base nos enunciados da atividade 1 – (a), (b), (c) e (d) –, escreva os programas que resolvam os problemas sugeridos.

Preenchimento e exibição do vetor

Você se lembra da situação hipotética proposta no início desta aula, na qual você precisava receber e armazenar o nome dos 45 alunos de uma turma?

Vamos ver como ficaria um algoritmo capaz de resolver esse problema utilizando um vetor?

*** Portugol Studio**

```
1 programa
2 {
3  funcao inicio()
4  {
5   inteiro indice
6   cadeia nomes[45]
7   para (indice=0;indice<=44;indice++)
8   {
9    escreva("Informe o nome do ",indice+1,"o. aluno: ")
10    leia(nomes[indice])
11   }
12   para (indice=0;indice<=44;indice++)
13   {
14    escreva(indice+1,"o. aluno: ",nomes[indice],"\n")
15   }
16  }
17 }
```

Vamos às explicações do que fazem as linhas do algoritmo:

Clique nos botões para ver as informações.

Linha 5

É responsável por criar a variável **indice**, que irá controlar a posição do vetor a ser acessada. É comum que os programadores chamem essa variável de **i**, **ind**, **posicao** ou **pos**.

Linha 6

Nessa linha é criado o vetor **nomes**, com capacidade para armazenar 45 elementos do tipo cadeia.

Linha 7

Inicia-se a estrutura repetitiva para preenchimento do vetor. Foi escolhida a estrutura **para** porque sabemos, de antemão, a quantidade de repetições a ser realizada. Como queremos o nome de 45 alunos, o trecho dentro do laço terá de ser repetido 45 vezes. Observe que a variável **indice** começa em 0 e vai até 44. Você saberia dizer por que ela não vai de 1 a 45? Acertou se disse que é por causa das posições do vetor! Lembra que a primeira posição do vetor é a posição 0? E a última, neste caso, é a posição 44? Então, a variável utilizada no laço para armazenamento de dados no vetor será a mesma que controlará o índice do mesmo e, por isso, precisa variar neste intervalo.

Linha 9

É exibida uma mensagem para o usuário informando que ele deve digitar o nome do primeiro aluno, do segundo aluno, do terceiro aluno, e assim por diante. Veja que, para que a primeira mensagem não solicitasse que o usuário informasse o nome do “0o.” aluno, o que obviamente estaria errado, foi preciso que a mensagem utilizasse a variável **indice+1**; já que a mesma inicia em 0 quando a estrutura repetitiva é criada.

Linha 10

São armazenados no vetor os nomes informados pelo usuário via teclado. Veja que o nome do vetor está acompanhado da variável **indice**, e não de um valor numérico fixo. Precisamos que seja assim porque desejamos armazenar os nomes sequencialmente, da posição 0 até a posição 44, “pulando” de uma em uma.

Linha 12

O laço iniciado na linha 12 é responsável pela exibição de todos os nomes armazenados no vetor.

Agora, imagine que você deseja receber 15 números inteiros quaisquer, armazená-los em um vetor e exibir os números armazenados na ordem inversa em que foram recebidos. Veja como ficaria o programa em C++ para resolver essa situação:

*** Programa em C++**

```
1 #include
2 using namespace std;
3 int main()
4 {
5  int indice, numeros[15];
6  for (indice=0;indice<=14;indice++)
7  {
8   cout <<"Informe o "<<indice+1<<"o. número: ";
9   cin >numeros[indice];
10  }
11  for (indice=14;indice>=0;indice--)
12  {
13   cout <<indice+1<<"o. número: "<<numeros[indice]<<"\n";
14  }
15 }
```

Observe que este programa tem a sintaxe e a estrutura bastante semelhantes ao que vimos no algoritmo no Portugol Studio, não é verdade?

Atente somente para a possibilidade de declarar a variável **indice** e o vetor **numeros** juntos, como vemos na linha 15. Isso é permitido porque ambos são do mesmo tipo. Veja, também, que a segunda estrutura repetitiva, iniciada na linha 11, define um valor inicial para a variável **indice** maior do que o valor final, além de definir o contador como **indice--**, ou seja, realizando uma operação de decremento.

Você entende a razão pela qual a definição foi feita dessa forma?

Lembra que o enunciado pedia que os números, após recebidos, fossem exibidos na ordem inversa àquela do recebimento?

Por esse motivo, precisamos definir a variável **indice** iniciando em 14, que é a última posição do vetor, e sendo decrementada até 0, que é a primeira posição do vetor.

Muito bem, agora que você sabe como os vetores funcionam e já viu alguns exemplos de sua utilização nos algoritmos e nos programas em C++.

Atividade

3 - Seu professor precisa de ajuda para corrigir as provas objetivas de seus alunos. Ele deseja que você escreva um programa em C++ que receba as respostas das provas de seus alunos e informe quantos acertos cada um deles têm. A prova consta de 15 questões, com alternativas que vão de A a E. Para ajudá-lo, você vai criar um vetor chamado **gabarito**, que irá conter as respostas da prova. As respostas de cada aluno serão armazenadas no vetor **prova**. Para cada turma, cuja quantidade de alunos será informada pelo professor, receba o conjunto de respostas de cada aluno, compare-as com o gabarito e informe quantos acertos ele teve.

Notas

comércio dos outros [1]

Assim, o governante mantém seu grupo familiar ampliado e seu pessoal militar com os proventos de seu próprio comércio e da exploração do comércio dos outros. Sua posição especial com respeito à propriedade da terra é, frequentemente, resultado e não causa da dominação política pela qual pode explorar oportunidades econômicas disponíveis.

Entretanto, o governo patrimonial com base nos recursos pessoais do governante e na administração do grupo familiar não pode normalmente dar conta dos problemas que surgem quando vastos territórios extrapatrimoniais ficam sujeitos a esse governo.

deveres dos dependentes [2]

Embora, em determinados casos, possa ser difícil distinguir entre a obediência pessoal de um dependente e os deveres públicos de um súdito político, está claro que a expansão do governo patrimonial tende a retirar o dependente e o súdito político do controle direto do governante.

aventureiro [3]

Essa característica aventureira é responsável pela extraordinária “plasticidade” do português, fazendo com que os primeiros colonos se fizessem instrumentos passivos, sobretudo se aclimatando facilmente, aceitando o que lhes sugeria o ambiente, sem cuidar de impor normas fixas e indeléveis.

Esse dominar ajustando-se às condições de nova terra acaba por deixar suas marcas na história da sociedade brasileira, e isto fica representado, talvez de forma mais intensa, na constituição das cidades, as quais, segundo a bela expressão do autor, foram como que semeadas, e não construídas por "ladrilhadores" com seus planejamentos e critérios rígidos, como as cidades da América hispânica.

Sergio Buarque [4]

Esse é, em linhas gerais, o raciocínio seguido dor Sergio Buarque na construção do que se pode considerar um dos eixos de argumentação de *Raízes do Brasil*.

Desde a apresentação do legado ibérico, associado à cultura da personalidade, até a construção da ideia de cordialidade, passando pelo ruralismo e pela aventura como elemento orquestrador da colonização, Sergio Buarque está descrevendo o "tradicionalismo" peculiar à sociedade brasileira, noção que toma a base do conceito de "patrimonialismo" de Max Weber, como podemos ver nas descrições anteriores.

Referências

MANZANO, J. A. N. G., OLIVEIRA, J. F. **Algoritmos**: lógica para desenvolvimento de programação de computadores. 28.ed. São Paulo: Érica, 2016.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados com aplicações em Java**. 2.ed. São Paulo: Prentice Hall, 2005.

Próxima aula

- Matrizes;
- Comportamento de algoritmos e programas com matrizes;
- Códigos com matrizes como recurso de armazenamento.

Explore mais

Um recurso muito utilizado em programação é a criação de funções, que são blocos de código que recebem um nome e, quando chamados, executam uma tarefa específica, como somar dois números ou calcular a raiz quadrada de outro.

Compreenda melhor esse tema, com o desafio <https://studio.code.org/s/course4/stage/12/puzzle/1 > bastante interessante. Divirta-se enquanto aprende esse novo conceito!