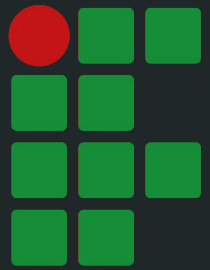


INSTITUTO FEDERAL
Fluminense
Campus Bom Jesus do Itabapoana

Estrutura de Dados

Prof. Anderson Veiga da Silva
anderson.silva@iff.edu.br

Introdução

Os tipos primitivos disponíveis em uma linguagem de programação para representar uma determinada informação são muito simples e, muita das vezes não são suficientes para representar informações mais complexas.

Uma alternativa seria permitir ao programador construir novos tipos de informações, utilizando os tipos primitivos como base, a fim de suprir esta necessidade.

Estruturas de dados podem ser definidas como formas mais eficientes de representar os dados dentro de um programa de computador, utilizando tipos primitivos para construir novos tipos mais complexos e parecidos com o objeto do mundo real a ser representado.

Exemplo

Como representar uma data em um programa de computador?

Uma alternativa seria criar três variáveis inteiras para representar cada informação, porém, seria difícil gerenciar mais de uma data dentro do programa pois várias variáveis seriam necessárias, além de problemas de validação que permitiriam colocar valores não permitidos, como um dia com o valor 32 ou -5.

Estruturas de dados nos permitem modelar novos tipos, além de fornecer meios de validação para os dados.

```
//3 variáveis para representar uma data
int dia, mes, ano;
dia = 44; //Não é possível validar o dia
```

```
//Tipo construído para representar a data
data nascimento;
nascimento.dia(44); //É possível validar o dia
```

Vetores

O vetor é uma estrutura de dados linear utilizada para armazenar um conjunto de valores do mesmo tipo.

O vetor possui um tamanho fixo dividido em células idênticas, onde cada célula pode armazenar um valor referente ao tipo de dados do vetor.

Exemplo de um vetor de inteiros:

1	5	2	7	14

Apesar de não ser um tipo primitivo, as linguagens de programação já fornecem os vetores de forma nativa para uso dentro de um programa.

Índices do vetor

Cada compartimento do vetor possui um endereço próprio, denominado **índice**, através do qual uma posição pode ser referenciada dentro do vetor.

Exemplo de um vetor com seus respectivos índices:

Os índices de um vetor sempre começam pelo **valor 0** (zero), ou seja, a primeira posição do vetor é a de índice 0, a segunda é a de índice 1, e assim se segue.

Criando um vetor

Para declarar um vetor é preciso informar seu tipo, nome e a quantidade de valores a ser armazenada dentro da estrutura, utilizando a seguinte sintaxe:

`tipo nome_vetor[quantidade];`

Exemplos:

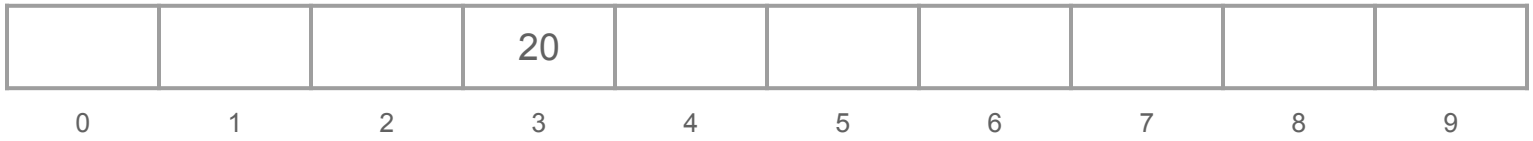
```
string nomes[10];
int numeros[5];
char palavra[20];
```

Inserindo valores de um vetor

Para inserir um valor em uma posição do vetor é preciso informar o índice da posição. Lembrando que o índice zero representa a primeira posição.

Exemplo de inserção de um valor na quarta posição do vetor:

```
int main () {
      int numeros[10];
      numeros[3] = 20;
}
```



Criando um vetor com valores

Para criar um vetor já preenchido com valores, utiliza-se o seguinte comando:

```
int main() {
     int numeros[] = {1, 5, 4, 8, 14, 17};
}
```

Vetor resultante:

1	5	4	8	14	17
0	1	2	3	4	5

Acessando valores de um vetor

Para acessar o valor, basta informar o vetor e o índice da respectiva posição.

Exemplo de impressão do valor de um vetor:

```
#include <iostream>
using namespace std;
int main() {
     int numeros[] = {1, 5, 4, 8, 14, 17};
     cout << numeros[3]; //Será exibido o número 8
}
```

Percorrendo um vetor

Para percorrer um vetor, utiliza-se uma estrutura de repetição. A estrutura mais adequada para para fazer esta varredura é o **`for`**.

Exemplo de impressão de todos os valores de um vetor:

```
#include <iostream>
using namespace std;
int main() {
     int numeros[] = {1, 5, 4, 8, 14, 17, 20, 28, 39, 55};
     for (int i = 0; i < 10; i++) {
          cout << numeros[i] << endl;
     }
}
```

Exemplo

Imagine resolver o cálculo da média entre 3 notas e informar quantas estão acima da média, porém sem usar vetores.

```
#include <iostream>
using namespace std;
int main() {
	int n1, n2, n3, qtd = 0;
	float media;
	cout << "Informe as notas" << endl;
	cin >> n1 >> n2 >> n3;
	media = (n1 + n2 + n3) / 3.0;
	if (n1 > media)
		qtd++;
	if (n2 > media)
		qtd++;
	if (n3 > media)
		qtd++;
	cout << "Média: " << media << "\nNotas acima da média: " << qtd <<endl;
}
```

Seria inviável escrever este algoritmo com uma grande quantidade de notas.

O uso de vetores simplifica este tipo de situação.

Exemplo

A resolução do mesmo problema utilizando vetores:

```
#include <iostream>
using namespace std;
int main() {
	int notas[10], qtd = 0;
	float media = 0;
	for (int i = 0; i < 10; i++) {
		cout << "Informe a nota:" << endl;
		cin >> notas[i];
	}
	for (int i = 0; i < 10; i++)
		media += notas[i];
	media = media / 10.0;
	for (int i = 0; i < 10; i++)
		if (notas[i] > media)
			qtd++;
	cout << "Média: " << media << "\nNotas acima da média: " << qtd <<endl;
}
```

Referências

ASCENCIO, A. F. G., CAMPOS, E. V. **Fundamentos da Programação de Computadores**. 2ª Ed. São Paulo: Pearson/Prentice Hall, 2008.

FORBELLONE, A. L. V., EBERSPACHER, H. F. **Lógica de Programação**: a Construção de Algoritmos e Estruturas de Dados. São Paulo: Makron Books, 2000.