VETOR PARA ARMAZENAMENTO DE DADOS

Armazenamento de dados

Variáveis, tipos simples e compostos

Durante a execução de um programa, armazenamos os dados necessários ao processamento em **variáveis** que residem, temporariamente, na memória do computador.

Geralmente, as linguagens de programação definem:

Tipos de dados simples

Apenas um dado pode ser armazenado por vez.

Tipos de dados estruturados ou compostos, ou estruturas de dados

Capazes de armazenar mais de um valor na mesma variável, usando mais de uma posição de memória para armazenar os dados da estrutura.

Disponível na grande maioria das linguagens de programação, incluindo a C, o vetor é uma das estruturas de dados mais recorrentes, classificadas como homogêneas, uma vez que armazenam dados do mesmo tipo.

Considere um programa que leia 100 números inteiros e positivos, e mostre o maior deles. Analise o trecho do código, em que as linhas estão numeradas para efeitos de explicação:

```
int main() {

   int num,maior,ind;

   maior = 0;

   for (ind= 1; ind <=110; ind=ind+1) {

      scanf ("%d",&num);

      if (num > maior) {

         maior=num;

      }

   }
```

```
    printf ("O maior dos números lidos e: \n %d", maior);

    return 0;

}
```

1. O uso de variáveis simples resolve esse problema. Podemos usar uma variável do tipo inteiro para armazenar cada número a ser lido e processado, bem como o maior dos números lidos a cada momento (linha 3).
2. A cada valor lido na variável simples e inteira **num (linha 7)**, o conteúdo anterior é descartado, ficando armazenado o último valor lido. A cada laço da iteração (repetição), comparamos o último valor lido com o conteúdo da variável **maior**, simples e inteira, devidamente inicializada com 0 (zero) na linha 4. Caso o valor lido seja superior ao contido na variável **maior**, atribui-se o número lido na variável **num** à variável **maior** (linhas 8 e 9).

Quando os 100 números forem lidos e processados e, portanto, houver o término da repetição, teremos o maior dos valores lidos armazenado na variável maior. Bastará exibi-lo no dispositivo de saída (linha 11) para concluir a solução do problema apresentado.

Existem problemas que não podem ser resolvidos usando apenas variáveis do tipo simples. É o que veremos a seguir.

Exemplo

Faça um programa que leia 100 números inteiros e mostre-os na ordem inversa em que foram lidos. Ao lermos o segundo número na mesma variável, o primeiro número lido será perdido. Assim, não teremos como exibi-los posteriormente, tampouco como usar 100 variáveis simples do tipo inteiro, como anteriormente, pois é necessário exibir os dados na ordem inversa daquela em que foram lidos.

Temos de armazenar todos os dados para, depois, mostrá-los na ordem desejada, conforme esta sequência com apenas 10 números:

Leitura → 10 56 78 90 12 91 23 42 90 58
Exibição → 58 90 42 23 91 12 90 78 56 10

Repare que o primeiro número digitado (lido pelo programa) será o último a ser exibido no dispositivo de saída. Para o caso de 10 números, até poderíamos cogitar a ideia de usar 10 variáveis do tipo simples (inteiro), mas, ainda assim, seria oneroso escrever o código. Como são 100 números, fica inviável usarmos 100 variáveis do tipo simples, pois a manipulação de 100 variáveis seria exaustiva. Imagine, então, se o enunciado pedisse a leitura de 1.000 números?

Tipos de dados estruturados ou compostos

Dados do tipo simples (int, float, double char etc.) são armazenados em uma variável que ocupa uma posição de memória.

Aqui, nós nos valemos da lei da Física: “Dois corpos não podem ocupar o mesmo lugar no espaço.” Logo, concluímos que existem diferenças entre tipos de **dados simples e estruturados**. Observe:

Tipos de dados simples

Podem armazenar apenas um valor por vez. Ao ser lido ou atribuído um novo valor a uma variável simples, seu conteúdo anterior é substituído por ele.

Tipos de dados estruturados ou compostos

Podem armazenar um conjunto de dados, simultaneamente, em memória, e, claro, usam mais de uma posição de memória.

Agregação e alocação dos dados estruturados

A forma como os dados são armazenados varia de estrutura para estrutura e depende de como a linguagem de programação os implementa.

Em geral, as linguagens de programação classificam os **dados estruturados** da seguinte forma:

Critérios	Classificação	Descrição	Exemplos
Forma de alocação	Estáticos	Dados declarados e criados no início do programa, e mantidos assim até o final de sua execução.	• Vetor • Registro • Matriz

Critérios	Classificação	Descrição	Exemplos
Tipos de dados	Dinâmicos	Dados criados durante a execução do programa	Ponteiros
	Homogênicos	Todos os dados são do mesmo tipo	• Vetores • Matrizes
	Heterogêneos	Os dados da estrutura podem ser de tipos diferentes.	Registros

Vetores

Conceito

O **vetor** é um tipo de dado estruturado ou composto, estático e homogêneo, disponível na maioria das linguagens de programação, incluindo a C.

Como estrutura de dados, o vetor permite que mais de um dado do mesmo tipo seja armazenado.

Como o vetor organiza os dados que armazena?

O vetor usa posições consecutivas de memória e simula esse funcionamento a partir de índices. Por isso, é uma estrutura **indexada**.

Os números em **verde** representam os índices do vetor, usados para acessar cada um de seus elementos. A posição 0 é a primeira, e a posição 9 é a última, como visto na seguinte imagem:

Representação de um vetor.

Declaração do vetor

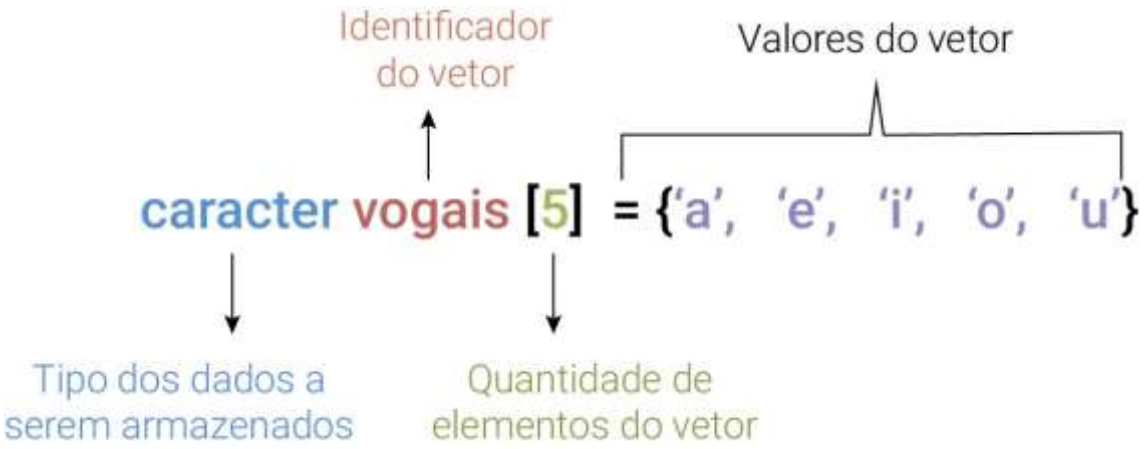
Como qualquer variável, o vetor precisa ser declarado na maioria das linguagens de programação que assim o exigem, como no caso da C. Você deve estar se perguntando:

Como os vetores, em geral, são declarados em Portugol Studio e na linguagem C?

Essa declaração é feita da seguinte maneira:

Declaração do vetor.

Vamos ilustrar a forma de declarar, em Portugol Studio, um vetor de 5 elementos do tipo **caractere**:



Como fica declarado o vetor representado na imagem anterior, em Portugol Studio e na linguagem C? É só observar os seguintes códigos:

// Portugol Studio

inteiro numeros[10]

// Linguagem C

int numeros[10]

Vamos ver outros exemplos de declaração de vetores:

Portugol Studio

inteiro numeros[100] → vetor de 100 elementos do tipo inteiro;
real notas [20] → vetor de 20 elementos do tipo real;
caractere vogais [5] → vetor de 5 elementos do tipo caractere.

Linguagem C

int numeros[100];
float notas [20];
char vogais [5].

Assim como as variáveis de tipo simples, as do tipo vetor também podem ser inicializadas durante sua declaração, conforme observamos a seguir:

// Portugol Studio

inteiro numeros[5] = {10,12,14,16,18}

// Linguagem C

int numeros[5] = {10,12,14,16,18};

Existem linguagens de programação que possibilitam a definição do índice inicial e final, mas, em Portugol Studio e na linguagem C, definimos apenas a **quantidade de elementos do vetor**. O primeiro elemento ocupa a posição 0 (zero), e o último, a posição (N-1), na qual N é a quantidade de elementos

definida para o vetor. A tentativa de acessar um elemento que esteja em uma posição inferior a 0 (zero) e superior a N-1 resultará em erro.

Acesso aos elementos do vetor

Exemplos

Os **elementos de um vetor** são acessados, atribuídos, lidos ou exibidos, elemento por elemento.

Considere um vetor de 10 elementos do tipo inteiro, com os seguintes comandos, numerados de 1 a 9, tanto em Portugol Studio quanto em linguagem C:

// Portugol Studio

inteiro vet[10]

1 vet [0] = 5

2 vet [1] = 6

3 vet [3] = 8

4 vet [4] = 12

5 vet [5] = 18

6 vet [6] = 22

7 vet [7] = 34

8 vet [8] = 78

9 vet [9] = 45

// Linguagem C

int vet[10];

1 vet [0] = 5;

2 vet [1] = 6;

3 vet [3] = 8;

4 vet [4] = 12;

5 vet [5] = 18;

6 vet [6] = 22;

7 vet [7] = 34;

8 vet [8] = 78;

9 vet [9] = 45;

Ao final da execução de cada comando associado às linhas numeradas de 1 a 9, teremos o vetor preenchido:

0	1	2	3	4	5	6	7	8	9
5	6	??	8	12	18	22	34	78	45

Vetor int vet[10]

Por que isso ocorreu?

1. Não houve inicialização dos elementos do vetor;
2. Não foi atribuído nenhum valor à posição 2 do vetor, que fica com valor indefinido.

Considerando o vetor apresentado, vamos ver o que vai acontecer com os comandos equivalentes em Portugol Studio e na linguagem C.

Comando para exibir um elemento do vetor

O comando a seguir exibe no dispositivo de saída o elemento que ocupa a posição (índice) 1 do vetor. O elemento da posição 1 é o 6:

// Portugol Studio

escreva ("Posicao 1 do vetor:", vet[1])

// Linguagem C

printf("%d Posicao 1 do vetor:", vet[1]);

Comando para atribuir valor a um elemento do vetor

O comando a seguir armazena o valor 90 no elemento de índice 4 do vetor. Esse elemento, anterior ao comando, é 12 e é substituído pelo 90:

// Portugol Studio

vet[4]=90

// Linguagem C

vet[4]=90

Como ficará o vetor? Veremos a seguir:

0	1	2	3	4	5	6	7	8	9
5	6	??	8	90	18	22	34	78	45

Vetor int vet[10]

O comando a seguir armazena o valor 56 no elemento de índice 2 do vetor. Esse elemento, anterior ao comando, é desconhecido, mas substituído pelo 56:

// Portugol Studio

vet[2]=56

// Linguagem C

vet[2]=56

Como ficará o vetor? Vejamos:

0	1	2	3	4	5	6	7	8	9
5	6	56	8	90	18	22	99	78	45

Vetor int vet[10]

Comando para ler um dado do dispositivo de entrada e armazenar em posição do vetor

Lendo o dado de entrada direto para uma posição do vetor, fica da seguinte forma:

// Portugol Studio

leia (vet[7])

// Linguagem C

scanf ("%d", &vet[7]);

O comando anterior armazena o valor lido pelo dispositivo de entrada na posição (índice 7) do vetor. O elemento de índice 7, anterior ao comando, é 34, que é substituído pelo valor lido.

Se o usuário digitar o número 99, como ficará o vetor?

0	1	2	3	4	5	6	7	8	9
5	6	56	8	90	18	22	99	78	45

Vetor int vet[10]

Lendo o dado do dispositivo de entrada em uma variável e, depois, atribuindo o conteúdo dessa variável a uma posição do vetor, fica da seguinte forma:

// Portugol Studio

leia (num)

vet [9] = num

// Linguagem C

scanf ("%d:", &num);

vet[9] = num;

O comando anterior armazena, em uma variável, o valor lido pela digitação do usuário no dispositivo de entrada. Na sequência, armazena o conteúdo dessa variável na posição (índice 9) do vetor. O elemento de índice 9, anterior ao comando, é 45, que é substituído pelo valor lido.

Se o usuário digitar o número 122, como ficará o vetor?

0	1	2	3	4	5	6	7	8	9
5	6	56	8	90	18	22	99	78	122

Vetor int vet[10]

Os comandos a seguir resultarão em erro, pois acessam posições (índices do vetor) que não são válidas:

```
// Portugol Studio
escreva (vet[10])
vet[10] = 901
leia (vet[10])

// Linguagem C
printf("%d Posicao 10 do vetor:", vet[10]);
vet[10] = 901;
scanf ("%d", &vet[10]);
```

Embora existam 10 elementos no vetor, nas linguagens Portugol Studio e C, o primeiro elemento ocupa a posição (índice) 0 (zero), e o último elemento ocupa a posição N-1, onde N é a quantidade de elementos do vetor. Assim, ao tentar acessar a posição 10 do vetor, haverá um erro na execução do programa, pois o último elemento é o da posição 9.

Cadeia de caracteres

Neste vídeo, vamos explorar a manipulação e o processamento de strings em C. Vamos aprender sobre a declaração, inicialização e manipulação de strings, incluindo funções e técnicas avançadas para trabalhar com texto.

Conceito e declaração

Muitas linguagens possuem uma **cadeia de caracteres** (*string*), mas a linguagem C não dispõe de um tipo de dado específico para armazenar essa cadeia, como o nome de uma pessoa.

Por isso, se quisermos armazenar nomes de pessoas, de lugares, enfim, qualquer cadeia de caracteres, teremos de usar um vetor de caracteres.

Uma cadeia de caracteres pode ser definida como um vetor contendo caracteres ou símbolos.

Na linguagem C, uma cadeia de caracteres pode ser tratada como um vetor de elementos do tipo **char** e com o marcador "**\0**". Se quisermos armazenar uma cadeia com 10 caracteres, deveremos declará-la com 11 elementos, pois o 11º será o caractere de controle, que é "**\0**".

Vamos ver um exemplo de representação de um vetor de caracteres, armazenando a cadeia "**INTRODUCAO!**":

0	1	2	3	4	5	6	7	8	9	10
I	N	T	R	O	D	U	C	A	O	\0

Representação de um vetor de caracteres.

Como é feita a declaração da cadeia de caracteres? Da seguinte forma:

char ‹Nome› [Tamanho+1]

Exemplo

Declaração para o vetor representado (com conteúdo = "INTRODUCAO"):
char nome [11]; //

Inicialização

Para **inicializar** uma cadeia de caracteres, basta:

1. Atribuir cada dado a uma posição, como vimos no vetor;
2. Usar funções para manipulação de caracteres, contidas na biblioteca padrão *string.h*.

Vamos mostrar aqui, com algumas variações, apenas a primeira opção com o que já sabemos de vetores:

```
#include <stdio.h>
int main() {
   char cadeia[10];
   cadeia[0]= "a";
   cadeia[1]= "l";
   cadeia[2]= "g";
   cadeia[3]= "o";
   cadeia[4]= "r";
```

```
    cadeia[5]= "i";
    cadeia[6]= "t";
    cadeia[7]= "m";
    cadeia[8]= "o";
    cadeia[9]= "s";
}
```

Ou podemos fazer desta maneira:

```
#include <stdio.h>
int main() {
    char cadeia[10] = {"a", "l", "g", "o", "r", "i", "t", "m", "o", "s"};
}
```

Leitura

Podemos **ler uma cadeia**, caractere por caractere, como fazemos com qualquer vetor, lendo elemento por elemento. Contudo, é mais simples lermos a cadeia inteira, usando a formatação **“%s”**:

```
scanf("%s",cadeia)
```

Note que não usamos &. O comando anterior seria equivalente ao trecho de código:

```
for (ind=0;ind<5;ind++) {
    scanf ("%c",&nome[ind]);
    getch();
}
```

Para leitura de dados pelo dispositivo de entrada com o *scanf*, devemos nos atentar a um detalhe: ele **não** lê o caractere de controle **(“\0”)**, mas, se o scanf estiver dentro de uma repetição, haverá tentativa de leitura desse caractere.

Logo, em comandos de leitura de variáveis do tipo **char**, coloque a função *getchar()* para resolver esse problema, conforme o exemplo anterior.

Exibição

Podemos usar %s para formatação de *strings* e a exibirmos direto, conforme fizemos com *scanf*. Veja o exemplo:

```
printf ("%s",cadeia)
```

O comando anterior seria equivalente ao trecho de código:

```
for (ind=0;ind<5;ind++) {
   printf ("%c" ,nome[ind]);
}
```

Exemplos de uso e manipulação de vetores e cadeias de caracteres

Exemplo 1

O comando declara um vetor para armazenar notas de 10 alunos de uma turma:

```
// Portugol Studio
real vet[10]

// Linguagem C
double vet[10];
```

Exemplo 2

O comando declara um vetor para armazenar o sexo – masculino (M) ou feminino (F) – de 50 alunos de uma turma ou, ainda, para representar o nome de uma pessoa com até 49 caracteres:

```
// Portugol Studio
caracter vet[50]

// Linguagem C
```

char vet[50];

Exemplo 3

O comando declara um vetor para armazenar notas de 10 alunos de uma turma e inicializa cada nota com 0 (zero):

// Portugol Studio

real vet[10] = {0,0,0,0,0,0,0,0,0,0}

// Linguagem C

double vet[10] = {0,0,0,0,0,0,0,0,0,0};

Exemplo 4

Se fossem 100 alunos em vez de 10, seria inviável inicializar o vetor com 100 zeros entre as chaves. Se fossem 500 alunos, mais inviável ainda.

Como inicializar cada elemento do vetor com 0 (zero)?

Nesses casos, podemos usar um comando de repetição para percorrer cada elemento do vetor e atribuir o valor 0 (zero) a cada um.

O comando de repetição mais adequado é o PARA (FOR), pois a repetição é para um número fixo e conhecido de vezes, no caso 100.

Vamos iniciar com a variável de controle, de nome **posicao**, começando do 0 (primeiro elemento do vetor) até 99 e, para cada posição, fazer com que **vet[posição]=0**.

// Portugol Studio

```
Para (posicao=0;posicao<100;posicao++) {
   vet[posicao]=0
}
```

// Linguagem C

```
for (posicao=0;posicao<100;posicao++) {
   vet[posicao]=0;
}
```

Exemplo 5

O comando declara um vetor para armazenar a quantidade de vezes que cada vogal aparece em um texto e inicializar com 0 (zero) cada quantidade. São 5 vogais. Então, precisamos de 5 posições no vetor. O algoritmo deve simular o mapeamento da seguinte forma:

1. Posição 0 = quantidade de vezes que a letra A apareceu.
2. Posição 1 = quantidade de vezes que a letra E apareceu.
3. Posição 2 = quantidade de vezes que a letra I apareceu.
4. Posição 3 = quantidade de vezes que a letra O apareceu.
5. Posição 4 = quantidade de vezes que a letra U apareceu.

Em formato de código, fica da seguinte forma:

```
// Portugol Studio
int vet[5]

// Linguagem C
int vet[5];
```

Exemplo 6

O trecho de código declara e lê as notas de 80 alunos de uma turma:

```
// Portugol Studio
Para (posicao=0;posicao<80;posicao++)
{
   leia (vet[posicao])
}
```

```
// Linguagem C
for (posicao=0;posicao<80;posicao++)
{
   scanf ("%d",&vet[posicao]);
}
```

Exemplo 7

O trecho de código exibe no dispositivo de saída as notas dos 80 alunos da turma:

```
// Portugol Studio
Para (posicao=0;posicao<80;posicao++)
{
   escreva (vet[posicao])
}
```

```
// Linguagem C
for (posicao=0;posicao<80;posicao++)
{
   printf ("%d",&vet[posicao]);
}
```

Exemplo 8

O trecho de código calcula e mostra a média da turma (soma das notas dos alunos dividida por 80, que é a quantidade de alunos):

```
// Portugol Studio
soma=0
para (posicao=0;posicao<80;posicao++) {
   soma=soma+vet[posicao]
```

```
}
escreva (“A media é :“, media/80)

// Linguagem C
soma=0;
for (posicao=0;posicao<80;posicao++) {
   soma=soma+vet[posicao];
}
printf(“media = %.2f“, (soma/80));
```

Exemplo 9

O trecho de código lê uma cadeia de 8 caracteres e mostra o texto invertido. Por exemplo, ao ler “programa”, o algoritmo deve mostrar “amargorp”:

```
// Portugol Studio
caracter nome[8]
inteiro ind
leia (“%s”, nome)
para (ind=8;ind>=0;ind--)
   escreva (nome[ind])

// Linguagem C
char nome[8];
int ind;
scanf ("%s",nome);
for (ind=8;ind>=0;ind--) {
   printf ("%c", nome[ind]);
}
```

Observe que o programa lê a cadeia de uma vez, usando **%s**, e exibe caractere por caractere para mostrar do final ao início, invertendo a cadeia original.

Vamos praticar

Vamos por em prática o que aprendemos no decorrer desse conteúdo. Sugerimos que você realize os seguintes procedimentos:

Desenvolvimento

Desenvolva, você mesmo, o algoritmo em Portugol Studio e, depois, na linguagem C;

Confirmação

Garanta que sua solução funcione com dados distintos, executando algumas vezes;

Comparação

Veja a solução que sugerimos e compare com a sua.

Vamos iniciar com uma prática que motivou o estudo das estruturas homogêneas de dados.

Aqui, alternativamente, vamos executar os programas na linguagem C, na ferramenta Online C++ Compiler – online editor, para que ganhe experiência em novo ambiente, além do DEVC++. Essa ferramenta, que você pode buscar na internet, tem a vantagem de não precisarmos instalar algo no computador. Após compilar o programa, é possível executá-lo normalmente.

Prática 1

Faça um programa que leia 100 números inteiros e mostre-os na ordem inversa em que foram lidos. Em relação à **estrutura de dados**, o vetor vai armazenar 100 números inteiros.

Temos a seguinte resolução:

1. Inicialmente, devem ser lidos cada um dos 100 números do dispositivo de entrada e armazenados em uma posição do vetor;
2. Depois, percorre-se o vetor de trás para frente, ou seja, iniciando do último elemento (posição 99) até o primeiro (posição 0) e exibindo cada elemento.

Comando de repetição: PARA (FOR), pois a repetição tem número conhecido e fixo de vezes =100.

Observe os seguintes encaminhamentos:

Ao ler os dados, usamos o PARA (FOR), começando da posição 0 (zero) e incrementando de 1 em 1, enquanto for menor que 100 (99), quando acaba a repetição.

Para exibir os dados em ordem inversa, percorremos o vetor da posição 99 (último elemento) até a posição 0, decrementando de 1 em 1, e mostramos cada elemento do vetor.

Preenchemos o vetor da posição 0 a 99 e, na sequência, mostramos seus elementos da posição 99 até a posição 0 (zero). Assim, mostramos os elementos na ordem inversa daquela que lemos.

Agora, pratique você mesmo usando o emulador a seguir:

Este é o trecho de código da solução tanto em Portugol Studio quanto em linguagem C:

```
// Portugol Studio
inteiro vet[100], posicao
para (posicao=0;posicao<100;posicao++)
{
leia (vet[posicao])
}
para (posicao=99;posicao>=0;posicao--)
{
escreva (vet[posicao])
}
#include<stdio.h>


int main ()
{
   int vet[100];
   int i;
```

```
   for (i=0;i<100;i++)
      scanf ("%d",&(vet[i]));
   for (i=99;i>=0; i--)
      printf ("%d, ",vet[i]);
   return 0;
}
```

Prática 2

Faça um programa que leia a nota de 20 alunos da turma e mostre as que são iguais ou superiores à média da turma, no emulador a seguir.

Estrutura de dados: O vetor armazenará 20 números reais.

```
#include <stdio.h>

int main()

{

   float vet[20],soma=0,media; int posicao;

   for (posicao=0;posicao<20;posicao++)

   {

      scanf ("%f",&vet[posicao]);

      soma=soma+vet[posicao];

   }

   media=soma/20;
```

```
    printf ("numeros maiores quemedia %.2f \n",media);

    for (posicao=0;posicao<20;posicao++)

    {

        if (vet[posicao] >= media)

        printf ("%.2f \n",vet[posicao]);

    }

}
```

Prática 3

Leia uma sequência de letras, terminada na letra (”z”), e mostre quantas vezes cada vogal (a, e, i, o, u) apareceu, no emulador a seguir.

Estrutura de dados: O vetor vai armazenar 5 números inteiros. Cada posição do vetor vai acumular a quantidade de vezes que cada vogal (A, E, I, O, U) apareceu. O índice 0 (zero) corresponde ao total de vogais “A”, o índice 1 corresponde à vogal “E”, e assim sucessivamente, até o índice 4, que vai armazenar a vogal “U”.

```
#include <stdio.h>
int main() {
    int vet [5] = {0,0,0,0,0},posicao;
    char letra;
    scanf ("%c",&letra);
    while (letra != 'z') {
        switch (letra) {
            case 'a':
```

```
            vet[0]++;
            break;

        case 'e':

            vet[1]++;
            break;

        case 'i':

            vet[2]++;
            break;

        case 'o':

            vet[3]++;
            break;

        case 'u':

            vet[4]++;
            break;
    }
    scanf("%c",&letra);
}

printf ("Quantidade de cada vogal, em ordem \n");

for (posicao=0;posicao<5;posicao++) {
    printf("%d\n", vet[posicao]);
}
```

```
    return 0;
}
```

Prática 4

Faça um algoritmo que leia 50 números inteiros e armazene-os em um vetor. Copie para um segundo vetor de 50 números inteiros cada elemento do primeiro, observando as seguintes regras:

1. Se o número for par, preencha a mesma posição do segundo vetor com o número sucessor do contido na mesma posição do primeiro vetor;
2. Se o número for ímpar, preencha a mesma posição do segundo vetor com o número antecessor do contido na mesma posição do primeiro vetor.

Ao final, mostre o conteúdo dos dois vetores simultaneamente no emulador a seguir:

Estrutura de dados: Dois vetores de 50 posições de números inteiros.

```
#include <stdio.h>
int main()
{
    const int tamvet=50;
    int vet1[tamvet],vet2[tamvet],posicao;
    for (posicao=0;posicao<tamvet;posicao++)
    {
        scanf ("%d",&vet1[posicao]);
        if (vet1[posicao]%2 == 0)
            vet2[posicao]=vet1[posicao]+1;
        else
            vet2[posicao]=vet1[posicao]-1;
    }
    printf ("Elementos de VET 1 e VET2: ");
```

```
    for (posicao=0;posicao<tamvet;posicao++)
    {
        printf ("%d\n",vet1[posicao]);
        printf ("%d\n",vet2[posicao]);
    }
}
```

Prática 5

Faça um algoritmo que leia 50 números inteiros e armazene-os em um vetor. Na sequência, leia uma lista de 10 números inteiros e verifique se cada um deles está contido em alguma posição do vetor. Em caso positivo, mostre a posição do vetor em que ele se encontra.

Estrutura de dados: Um vetor de 50 posições de números inteiros.

Comando de repetição: PARA (FOR), pois sabemos que leremos e processaremos 50 elementos. Logo, teremos uma solução com número fixo e conhecido de vezes.

```
#include <stdio.h>
int main() {
    const int tamvet=50, tamlista=10;
    int vet[tamvet],posicao,posvet,achou,numero;
    printf ("\n Digite os dados do vetor \n");
    for (posicao=0;posicao<tamvet;posicao++)
        scanf ("%d",&vet[posicao]);
        printf ("\n Digite numeros a procurar: \n");
    for (posicao=1;posicao<=tamlista;posicao++) {
        scanf ("%d",&numero);
        // verifica se o numero está no vetor
        posvet=0;
        achou=0;
        while (posvet<=tamvet-1 && achou==0) {
            if (numero==vet[posvet])
```

```
            achou=1;
          else posvet++;
       }
    }
    if (achou==1)
       printf("achou na posição: %d\n", posvet);
    else
       printf("nao achou o numero\d");
}
```

Prática 6

Considere o seguinte problema em uma escola primária: em uma turma com 50 alunos, cada um faz 3 provas por semestre.

Além de armazenar as 3 provas dos 50 alunos, existe a necessidade de mostrar:

- A média de cada prova;
- A média de cada aluno;
- A média da turma.

Agora, vamos praticar as duas ações a seguir:

Identificar

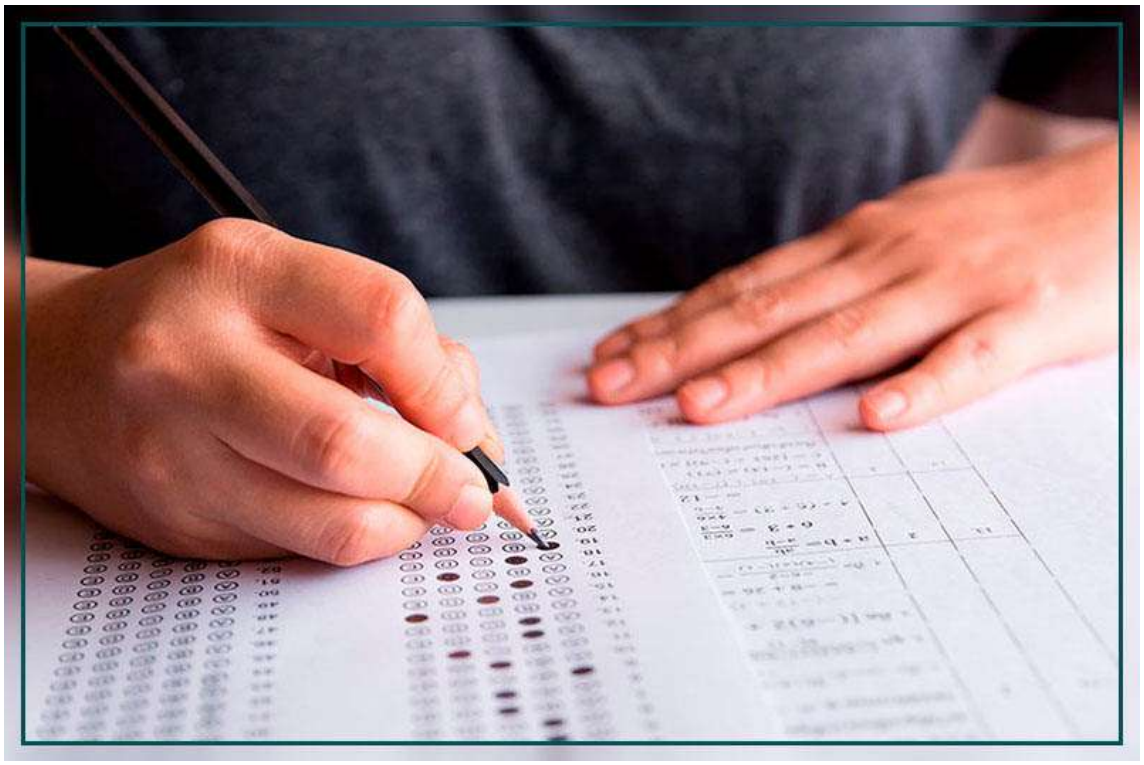
As estruturas de dados necessárias à solução do problema.

Escrever

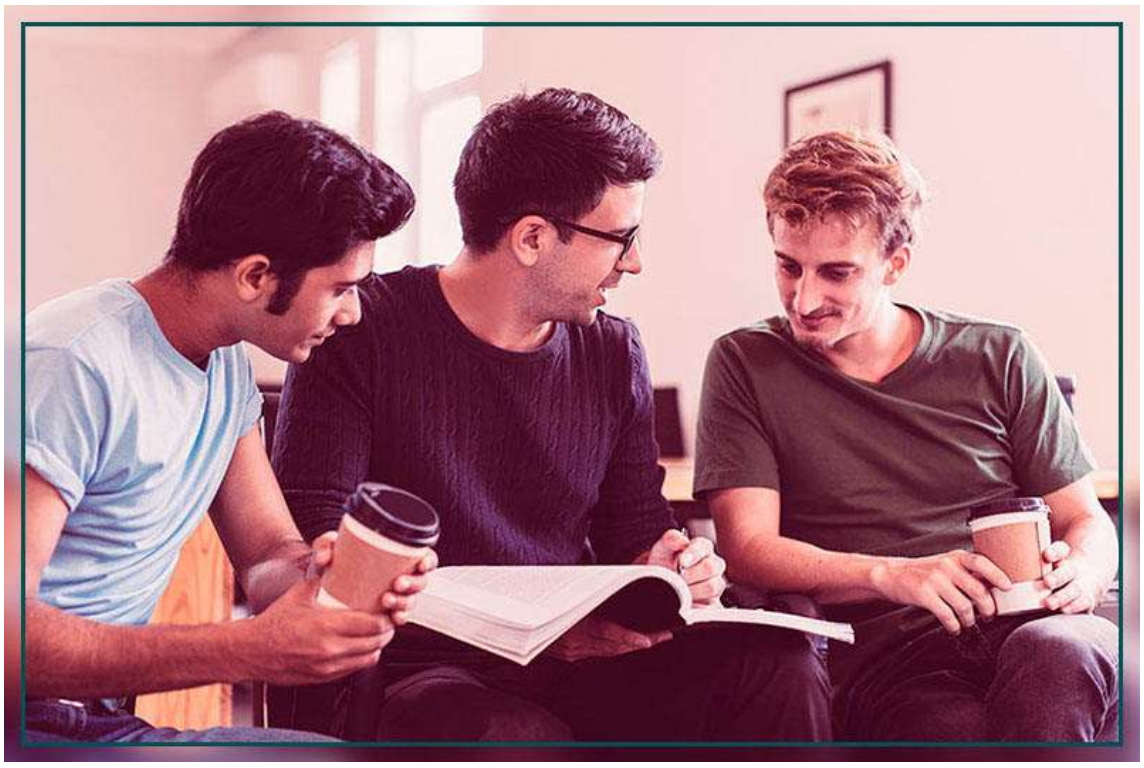
O trecho em linguagem C para calcular, mostrando as médias de cada prova, aluno e turma.

Resolução

A grande questão é a organização da estrutura para armazenar os dados. Existem outras soluções: vamos usar 3 vetores de 50 posições de números reais. Veja o que acontece a seguir:



Cada vetor armazena dados de 1 prova: prova1, prova2, prova3.



Cada aluno está representado no mesmo índice dos 3 vetores.

As posições de cada vetor ficam da seguinte forma:

- A posição 0 do vetor prova1 é a nota da prova1 do aluno1.
- A posição 0 do vetor prova2 é a nota da prova2 do aluno1.
- A posição 0 do vetor prova3 é a nota da prova3 do aluno1.
- A posição 1 de cada vetor representa as provas do aluno 2, e assim por diante.
- A posição 49 de cada vetor armazena as provas do último aluno.

Observe o exemplo em desenho das 3 estruturas:

0	1	2	3	.....			47	48	49
7.00	8.50	9.35	8.45	.....	.....	.....	9.00	7.90	9.60

Vetor prova 1

0	1	2	3	.....			47	48	49
9.70	9.20	8.95	9.00	.....	.....	.....	10.00	6.40	4.30

Vetor prova 2

0	1	2	3	.....			47	48	49
10.00	8.50	7.50	8.80	.....	.....	.....	9.20	8.00	9.00

Vetor prova 3

Veja que o trecho de código, em linguagem C, declara os vetores necessários:

float prova1[50], prova2[50], prova3[50];

Média de cada prova

Usamos todas as posições de cada vetor.

Prova 1

7.00, 8.50, 9.35, 8.45 ... 9.00, 7.90, 9.60.

Prova 2

9.70, 9.20, 8.95, 9.00 ... 10.00, 6.40, 4.30.

Prova 3

10.00, 8.50, 7.50, 8.80 ... 9.20, 8.00, 9.00.

Em outras palavras, para calcular a média de cada prova, basta acumular a nota de cada vetor prova e dividir a soma por 50.

O seguinte trecho de código em linguagem C calcula a média da prova 1:

```
Soma1=0;
   for (pos=0;pos<49;pos++) {
      soma1=soma1+prova1[pos];
   }
   mediaprova1=soma/50;
   printf ("a media da prova1 e: %f.02", mediaprova1);
```

Para calcular a média das provas 2 e 3, basta substituir as variáveis **soma1**, **mediaprova1** e o vetor **prova1** neste trecho de código.

Média de cada aluno

Usamos a mesma posição dos 3 vetores. Dessa maneira, as notas de cada aluno são:

Aluno 1

7.00 9.70 10.00 → média = 8.90.

Aluno 2

8.50 9.20 8.50 → média = 8.73; e assim por diante.

Em outras palavras, para calcular a média de cada aluno, basta somar as notas nas mesmas posições dos 3 vetores e dividir por 3.

O seguinte trecho de código, em linguagem C, calcula a média de cada aluno:

```
for (pos=0;pos<50;pos++) {
      medialuno=(prova1[pos] + prova2[pos] + prova3[pos])/3;
   }
   printf ("a media do aluno e: %f.02", mediaaluno);
```

Média da turma

Pode ser calculada de 2 formas:

Pela média das provas

Anteriormente, quando calculamos a média de cada prova, chegamos à **mediaprova1**, **mediaprova2** e **mediaprova3**. A média da turma pode ser obtida somando a média das 3 provas e dividindo por 3, conforme trecho do código:

```
mediaturma = (mediaprova1 + mediaprova2 + mediaprova3)/3;
```

Pela média dos alunos

Usando o mesmo código para tirar a média de cada aluno, basta acrescentar as linhas de código marcadas em amarelo:

```
somaturma=0;
for (pos=0;pos<50;pos++)
{
   medialuno = (prova1[pos] + prova2[pos] + prova3[pos])/3;
   somaturma = somaturma + medialuno;
}
mediaturma = somaturma/50;
printf ("a media da turma e: %f.02",mediaturma);
```