Programação de Computadores

FUNÇÕES COM VETORES

Judson Santiago

Introdução

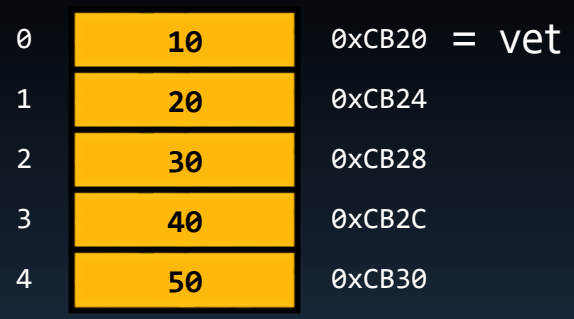
- Vetores são estruturas de dados importantes
  - Amplamente utilizados na solução de problemas
  - São processados de forma eficiente pelas máquinas
    - Os dados são dispostos de forma sequencial na memória
    - A organização sequencial maximiza o uso do cache

Atividade	Tempo de acesso	Comparação
Cache L1	0.9 ns	1x
Cache L2	2.8 ns	3x
Cache L3	12.9 ns	14x
Memória Principal	120 ns	130x

Introdução

- As funções tratam vetores como endereços de memória
  - Evita a cópia de grandes volumes de dados
  - Requer atenção e cuidado na manipulação do vetor

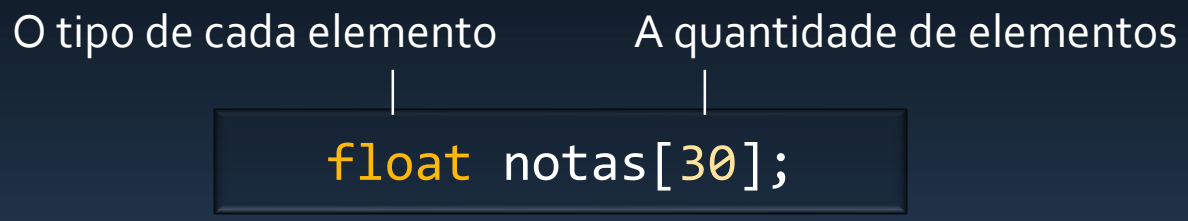
```
int main()
{
    int vet[5] = { 10,20,30,40,50 };

    // função recebe endereço 0xCB20
    cout << SomarVetor(vet);
}
```



Vetores

- Um vetor armazena um conjunto de valores
  - Todos do mesmo tipo
    - Ex.: as notas de 30 alunos (valores tipo float)

- A declaração de um vetor deve conter:



Vetores

```cpp
#include <iostream>
using namespace std;
int main()
{
     float prova[3];   // cria vetor de 3 elementos
     prova[0] = 6.8f;  // atribui valor ao 1º elemento
     prova[1] = 5.5f;  // atribui valor ao 2º elemento
     prova[2] = 6.2f;  // atribui valor ao 3º elemento

     float trabalho[3] = {2.5f, 3.0f, 1.2f}; // cria e inicializa vetor

     cout << "Notas\n";
     cout << prova[0] + trabalho[0] << endl;
     cout << prova[1] + trabalho[1] << endl;
     cout << prova[2] + trabalho[2] << endl;

     float total = 0.0f;
     for (int i = 0; i < 3; ++i)
          total += prova[i] + trabalho[i];

     cout << "A média das notas é " << total/3.0f << endl;
}
```

Vetores.cpp

Vetores

- A saída do programa:

```
Notas
9.3
8.5
7.4
A média das notas é 8.4
```

- Um vetor não inicializado contém valores indefinidos

```
float prova[3];  // os valores armazenados
                 // não são iguais a 0
                 // eles são indefinidos até que
                 // seja feita uma atribuição de valor
```

Funções e Vetores

- Suponha o problema de verificar o número de licenças de um software vendidas em um período de 30 dias
  - Uma solução é armazenar a quantidade de vendas diárias em um vetor de 30 elementos

  - O total é obtido somando os elementos com um laço
    - Somar elementos é uma tarefa comum
    - Vale a pena criar uma função

Funções e Vetores

- A função deve receber o vetor com o seu tamanho e retornar a soma dos elementos

```
// soma os elementos de um vetor de tamanho n
int SomarVetor(int vet[], int n);
```

  - O parâmetro vet não recebe uma cópia dos elementos do vetor, ele é um ponteiro e poderia ser escrito assim:

```
// soma os elementos de um vetor de tamanho n
int SomarVetor(int * vet, int n);
```

Funções e Vetores

```cpp
#include <iostream>
using namespace std;

int SomarVetor(int vet[], int n);

int main()
{
    const int Qtd = 7;
    int vendas[Qtd] = { 1, 1, 0, 4, 1, 3, 0 };
    int total = SomarVetor(vendas, Qtd);
    cout << "Total de vendas: " << total << "\n";
}

int SomarVetor(int vet[], int n)
{
    int soma = 0;
    for (int i = 0; i < n; ++i)
        soma += vet[i];
    return soma;
}
```

SomaVetor.cpp

Funções e Vetores

- Saída do Programa:

```
Total de vendas: 10
```

- Não importa se o parâmetro da função usa a notação de ponteiro ou a notação de vetor, os elementos podem sempre ser acessados com a notação de vetor

```
int SomarVetor(int vet[], int n);
int SomarVetor(int * vet, int n);

soma += vet[i];
```

Funções e Vetores

- A chamada passa um endereço e não o conteúdo do vetor

```
int SomarVetor(int vet[], int n);  // protótipo da função
```

Endereço do vetor vendas

```
SomarVetor( vendas, Qtd);    // chamada da função
```

- Ainda assim, estamos fazendo uma cópia
  - O endereço é copiado para o parâmetro da função

Funções e Vetores

- A passagem do tamanho do vetor é necessária

```
const int Qtd = 7;
int vendas[Qtd] = { 1, 1, 0, 4, 1, 3, 0 };
SomarVetor(vendas, Qtd);
```

  - O tamanho não poderia ser calculado dentro da função?

```
void Imprimir(char nome[])
{
     // cout << nome;
     int tam = strlen(nome);
     for (int i = 0; i < tam; ++i)
        cout << nome[i];
}
```

Funções e Vetores

```cpp
#include <iostream>
using namespace std;

int SomarVetor(int [], int);

int main()
{
    int vendas[7] = { 1, 1, 0, 4, 1, 3, 0 };

    cout << "Endereço de vendas = " << vendas;
    cout << "Tamanho de vendas = "  << sizeof vendas
         << endl << endl;

    cout << "Total de vendas "
         << SomarVetor(vendas, 7) << endl;
    cout << "Três primeiros  "
         << SomarVetor(vendas, 3) << endl;
    cout << "Quatro últimos  "
         << SomarVetor(vendas + 3, 4) << endl;
}
```

```cpp
int SomarVetor(int vet[], int n)
{
    cout << "(vet = "
         << vet
         << ", tamanho = "
         << sizeof vet
         << "): ";

    int soma = 0;
    for (int i = 0; i < n; i++)
       soma += vet[i];
    return soma;
}
```

TamVet.cpp

Funções e Vetores

- Saída do Programa:

```
Endereço de vendas = 0x0065fd24
Tamanho de vendas = 28

Total de vendas (vet = 0x0065fd24, tamanho = 4): 10
Três primeiros  (vet = 0x0065fd24, tamanho = 4): 2
Quatro últimos  (vet = 0x0065fd34, tamanho = 4): 8
```

  - O tamanho de vendas é o tamanho do vetor
  - O tamanho de vet é o tamanho do ponteiro

Aplicações com Vetores

- Ao escolher um vetor para representar dados, estamos tomando uma decisão de projeto
  - Suponha usar um vetor para acompanhar o valor de imóveis
    - Que tipo usar para o valor dos imóveis?
      - float, double, int, unsigned?
    - Quantos imóveis o programa vai gerenciar?
      - Usar um tamanho fixo para o vetor
      - Usar um vetor de tamanho variável

Aplicações com Vetores

- Decisões de projeto devem envolver não só a forma de armazenar os dados mas também a forma de manipular os dados
  - Que operações serão feitas sobre os imóveis?
    - A resposta define que funções serão necessárias
      - Ler o valor de cada imóvel: **ler vetor**
      - Mostrar o valor atual dos imóveis: **mostrar vetor**
      - Reavaliar o valor dos imóveis: **modificar vetor**

Ler Vetor

- Uma função que recebe um vetor e deve:
  - Preencher o vetor com valores lidos do teclado
  - Modificar o conteúdo do vetor original
  - Tratar vetores de qualquer tamanho
  - Retornar o número de elementos lidos

```
// lendo valores do teclado para um vetor
int LerVetor(double vet[], int tam);
```

Ler Vetor

```
int LerVetor(double vet[], int tam)
{
     double temp;
     int i = 0;
     cout << "Digite valor #1: ";
     while (cin >> temp && i < tam && temp >= 0)
     {
        vet[i++] = temp;
        cout << "Digite valor #" << (i+1) << ": ";
     }
     if (cin.fail())
     {
        cin.clear();
        while (cin.get() != '\n')
            continue;
        cout << "Entrada inválida. Leitura encerrada!\n";
     }
     return i;
}
```

Imoveis.cpp

Mostrar Vetor

- Uma função para mostrar o conteúdo do vetor deve:
  - Receber um vetor e o seu número de elementos
  - Usar um laço para percorrer cada elemento e mostrar seu valor

- Como a função trabalha com os valores originais do vetor, podemos protegê-lo contra alterações usando const

```
// mostrando o conteúdo de um vetor
void MostrarVetor(const double vet[], int tam);
```

Mostrar Vetor

```
// mostrando o conteúdo de um vetor
void MostrarVetor(const double vet[], int tam)
{
     for (int i = 0; i < tam; ++i)
     {
        cout << "Imóvel #" << (i+1) << ": R$";
        cout << vet[i] << endl;
     }
}
```

- O ponteiro vet aponta para um valor constante e assim não pode ser usado para alterar o conteúdo do vetor original

Modificar Vetor

- A reavaliação do imóvel consiste em multiplicar o seu valor atual por um fator de ajuste
  - A função deve receber o fator de ajuste
  - Ela deve receber também o vetor e seu tamanho

```
// reajusta valor de imóveis por um fator
void AjustarVetor(double fator, double vet[], int tam)
{
     for (int i = 0; i < tam; ++i)
         vet[i] *= fator;  // vet[i] = vet[i] * fator;
}
```

Construindo a Aplicação

```
#include <iostream>
#include "imovel.h"     // contém o protótipo das funções
using namespace std;

int main()
{
     const int Max = 5;
     double imoveis[Max];

     int tam = LerVetor(imoveis, Max);
     MostrarVetor(imoveis, tam);

     cout << "Digite o fator de ajuste: ";
     double ajuste;
     cin >> ajuste;
     AjustarVetor(ajuste, imoveis, tam);
     MostrarVetor(imoveis, tam);
}
```

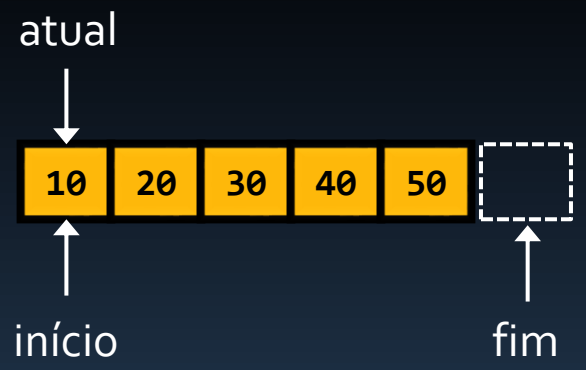
Imoveis.cpp

Faixa de Elementos

- Funções que trabalham com vetores precisam receber:
  - A localização do início do vetor
  - O número de elementos do vetor
- Existe uma outra abordagem:
  - Passar uma faixa de elementos†
  - Isso pode ser feito usando dois ponteiros:
    - Um identificando o início do vetor
    - Outro identificando o fim do vetor

Faixa de Elementos

- Uma convenção é fazer o ponteiro de fim apontar para a posição imediatamente após o último elemento
  - Uma faixa de elementos está vazia quando inicio == fim
  - Simplifica o teste para encerrar o laço




```
while (atual != fim)
{
    cout << *atual << " ";
    ++atual;
}
```

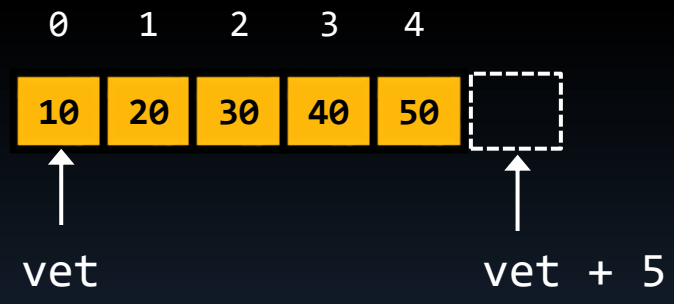
Faixa de Elementos

- Os ponteiros de início e fim podem ser obtidos facilmente através do nome do vetor e do seu tamanho
  - Considere a declaração abaixo:

```
double vet[5];
```

  - Os dois ponteiros seriam:

```
vet           // início da faixa
vet + 5       // fim da faixa
```



  - vet + 4 aponta para o último elemento (vet[4]), assim vet + 5 aponta para o fim da faixa, um elemento após o último

Faixa de Elementos

```cpp
// Funções recebendo faixa de elementos do vetor
#include <iostream>
using namespace std;

int SomarVetor(const int * inicio, const int * fim);

int main()
{
     const int Tam = 8;
     int potencias[Tam] = { 1, 2, 4, 8, 16, 32, 64, 128 };

     int total = SomarVetor(potencias, potencias + Tam);
     cout << "Soma das potências de dois: " << total << "\n";

     total = SomarVetor(potencias, potencias + 3);
     cout << "As três primeiras somam " << total << "\n";

     total = SomarVetor(potencias + 4, potencias + 8);
     cout << "As quatro últimas somam " << total << "\n";
}
```

FaixaVetor.cpp

Faixa de Elementos

```
// retorna a soma dos valores do vetor
int SomarVetor(const int * inicio, const int * fim)
{
     int soma = 0;
     for (int * atual = inicio; atual != fim; ++atual)
        soma += *atual; // soma = soma + *atual;

     return soma;
}
```

- Saída do Programa:

```
Soma das potências de dois: 255
As três primeiras somam 7
As quatro últimas somam 240
```

Ponteiros Constantes

- O const pode ser usado com ponteiros de duas formas:
  - Ponteiro aponta para conteúdo constante:

```
int idade = 20, total = 10;  // variáveis inteiras
const int * ptr = &idade;    // *ptr é constante
*ptr = 30;                   // inválido x
ptr = &total;                // válido   √
```

  - O ponteiro em si é uma constante

```
int idade = 20, total = 10;  // variáveis inteiras
int * const ptr = &idade;    // ptr é constante
*ptr = 30;                   // válido   √
ptr = &total;                // inválido x
```

Resumo

- Ao passar vetores para funções manipula-se o vetor original
  - Para proteger o vetor contra alterações utiliza-se const

```
// mostrando o conteúdo de um vetor
void MostrarVetor(const double vet[], int tam)
```

- Um vetor pode ser passado para funções de duas formas:
  - O nome do vetor e o número de elementos
  - Ponteiros indicando o início e fim do vetor

```
// retorna a soma dos valores do vetor
int SomarVetor(const int * inicio, const int * fim)
```

Resumo

- O const pode ser usado de duas formas com ponteiros
  - Ponteiros para conteúdo constante
    ```
    const int * pt;
    ```
  - Ponteiros constantes
    ```
    int * const pt;
    ```
  - A primeira forma é bastante empregada em parâmetros de funções
    - Impede a modificação do conteúdo dentro da função
    - Funciona como uma documentação do código