Programação de Computadores

APLICAÇÕES DOS LAÇOS DE REPETIÇÃO

Judson Santiago

Introdução

- A linguagem C++ oferece três tipos de laços de repetição:
  - for
    Fornece uma maneira prática de controlar contadores através da inicialização-teste-atualização
  - while
    Indicado para repetir instruções um número desconhecido de vezes
  - do-while
    Ideal para executar a repetição pelo menos uma vez

Introdução

- Os laços são usados para realizar tarefas repetitivas
  - Como por exemplo:
    - Leitura e exibição de vetores
    - Tratamento de strings
    - Acúmulo e soma de valores
  - Outras aplicações:
    - Tratamento da entrada do usuário
    - Processamento de matrizes

Tratamento da Entrada

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

int main()
{
     char ch;
     int cont = 0;

     cout << "Digite caracteres, use # para sair:\n";
     do
     {
          cin >> ch;       // lê um caractere
          cout << ch;      // exibe o caractere
          ++cont;          // conta o número de caracteres
     }
     while (ch != '#');  // testa o caractere

     cout << endl << --cont << " caracteres lidos.\n";
}
```

Repete1.cpp

Tratamento da Entrada

- A saída do programa:

```
Digite caracteres, use # para sair:
Ele pode correr#muito rápido
Elepodecorrer#
13 caracteres lidos.
```

- A entrada com cin:
  - Ignora os caracteres de espaço e tabulação
  - Utiliza um buffer para ser mais eficiente
    - Por isso é possível digitar após #

Tratamento da Entrada

- Normalmente, programas que lêem a entrada caractere a caractere precisam examinar todos os caracteres (incluindo espaços, tabulações e novas linhas):
  - A função cin.get(ch) lê o próximo caractere da entrada e o atribui a variável ch
  - A entrada com cin.get() também usa um buffer, ainda é possível digitar além do #

```
char ch;
cin.get(ch);
```

Tratamento da Entrada

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

int main()
{
     char ch;
     int cont = 0;

     cout << "Digite caracteres, use # para sair:\n";

     cin.get(ch);         // lê um caractere
     while (ch != '#')    // testa o caractere
     {
         cout << ch;      // exibe o caractere
         ++cont;          // conta o número de caracteres
         cin.get(ch);     // lê o próximo caractere
     }
     cout << endl << cont << " caracteres lidos.\n";
}
```

Repete2.cpp

Tratamento da Entrada

- A saída do programa:

```
Digite caracteres, use # para sair:
Ele pode correr#muito rápido
Ele pode correr
15 caracteres lidos.
```

- C++ suporta funções com o mesmo nome, contanto que os argumentos sejam de tipos (ou em quantidades) diferentes

```
char ch;
cin.get(ch);     // versão que recebe um argumento char
ch = cin.get();  // versão que retorna o caractere lido
```

Tratamento da Entrada

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

int main()
{
     char ch;
     int cont = 0;

     cout << "Digite caracteres, use # para sair:\n";

     // lê e testa o caractere
     while ((ch = cin.get()) != '#')
     {
         cout << ch;     // exibe o caractere
         ++cont;         // conta o número de caracteres
     }
     cout << endl << cont << " caracteres lidos.\n";
}
```

Repete3.cpp

Tratamento da Entrada

- A saída do programa:

```
Digite caracteres, use # para sair:
Ele pode correr#muito rápido
Ele pode correr
15 caracteres lidos.
```

- É possível eliminar a leitura dupla no while usando a versão de cin.get() que retorna o caractere lido

```
while ((ch = cin.get()) != '#')   // lê e testa o caractere
```

Matrizes

- Um vetor é uma sequência de elementos do mesmo tipo

```
int visitas[10];
```

- Uma matriz é um vetor bidimensional

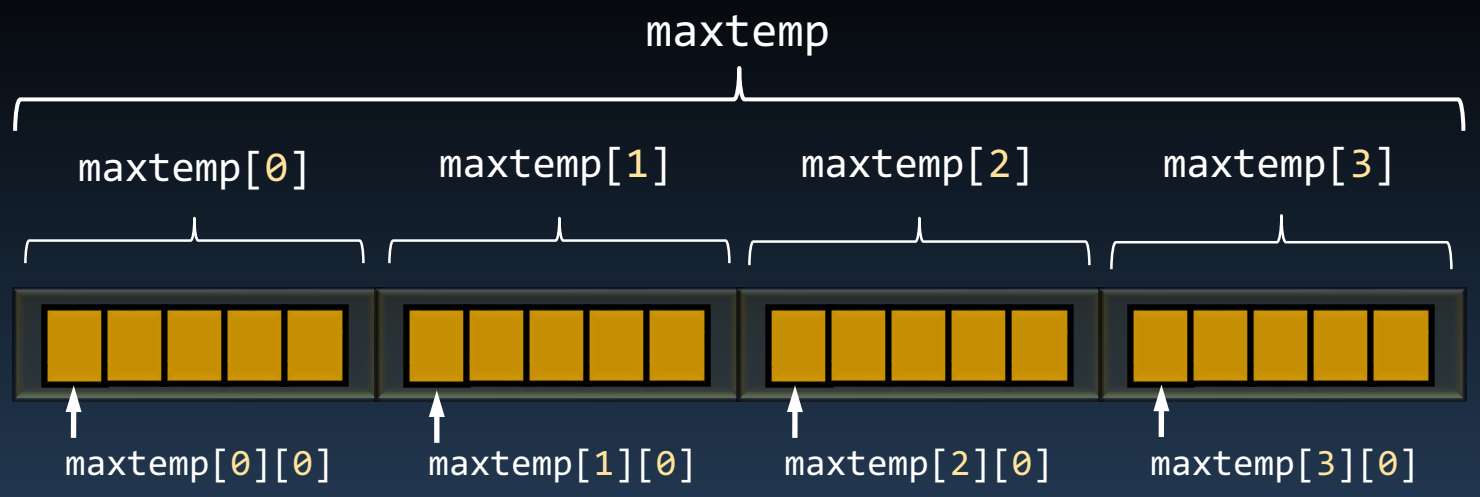
```
int maxtemp[4][5];
```

Matrizes

- Uma matriz é um vetor em que cada elemento é um vetor

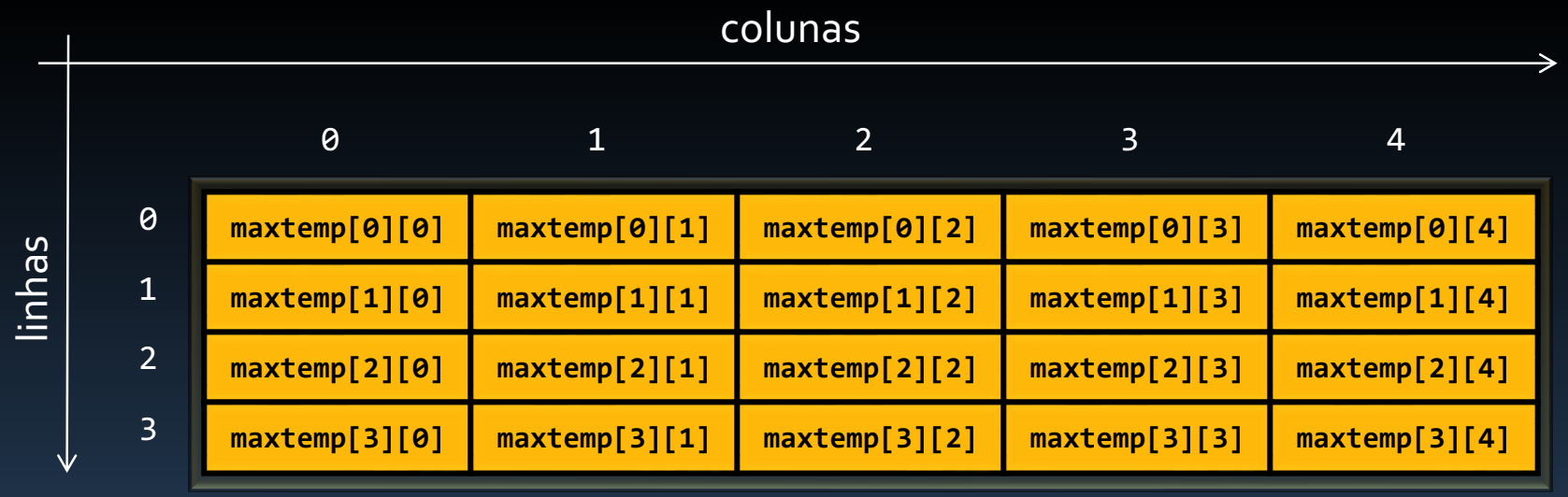
```
int maxtemp[4][5];
```

maxtemp é um vetor de 4 elementos
cada elemento é um vetor de 5 inteiros



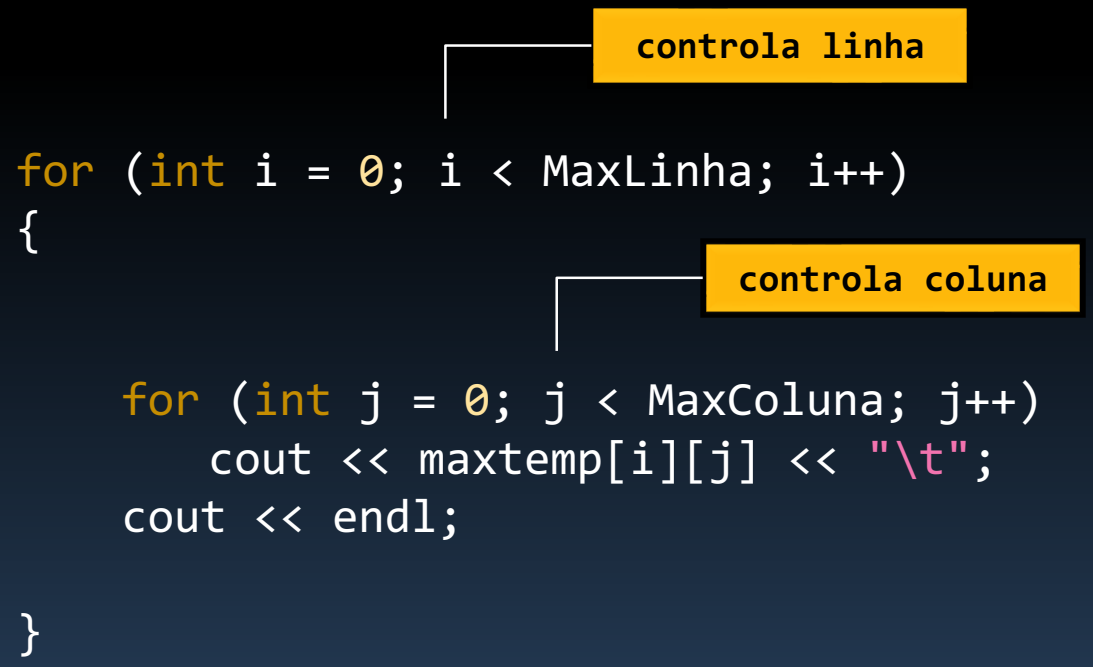
Matrizes

- Um elemento da matriz é acessado através de dois índices:
  - O primeiro é a linha da matriz
  - O segundo é a coluna da matriz



Matrizes

- Laços for aninhados são ideais para processar matrizes
  - Um laço muda a linha e o outro muda a coluna



Matrizes

- Vetores podem ser inicializados na sua declaração

```
int btus[4] = {7500, 9000, 12000, 15000};
```

- Matrizes também podem ser inicializadas

```
int maxtemp[4][5] =
{
   {35, 28, 25, 20, 24},    // valores para maxtemp[0]
   {15, 19, 23, 35, 32},    // valores para maxtemp[1]
   {34, 36, 30, 31, 30},    // valores para maxtemp[2]
   {31, 36, 38, 32, 26}     // valores para maxtemp[3]
};
```

Matrizes

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

const int Cids = 4;
const int Anos = 5;

int main()
{
     const char * cidades[Cids] =
     {"Mossoró", "Caraúbas", "Angicos", "Pau dos Ferros"};

     int maxtemp[Cids][Anos] =
     {
        {35, 28, 25, 20, 24},    // valores para maxtemp[0]
        {15, 19, 23, 35, 32},    // valores para maxtemp[1]
        {34, 36, 30, 31, 30},    // valores para maxtemp[2]
        {31, 36, 38, 32, 26}     // valores para maxtemp[3]
     };
```

Continua →

MaxTemp.cpp

Matrizes

```
    int maxtemp[Cids][Anos] =
    {
       {35, 28, 25, 20, 24},    // valores para Mossoró
       {15, 19, 23, 35, 32},    // valores para Caraúbas
       {34, 36, 30, 31, 30},    // valores para Angicos
       {31, 36, 38, 32, 26}     // valores para Pau dos Ferros
    };

    cout << "Temperaturas máximas dos últimos anos:\n\n";

    for (int i=0; i < Cids; ++i)
    {
        cout << cidades[i] << ":\t";
        for (int j=0; j < Anos; ++j)
            cout << maxtemp[i][j] << "\t";
        cout << endl;
    }
}
```

MaxTemp.cpp

Matrizes

- A saída do programa:

```
Temperaturas máximas dos últimos anos:

Mossoró:         35      28      25      20      24
Caraúbas:        15      19      23      35      32
Angicos:         34      36      30      31      30
Pau dos Ferros:  31      36      38      32      26
```

- O programa cria um vetor constante, seus elementos não podem ser modificados

```
const char * cidades[Cids] =
{"Mossoró", "Caraúbas", "Angicos", "Pau dos Ferros"};
```

Matriz Dinâmica

- Em muitas aplicações o tamanho da matriz é conhecido apenas durante a execução do programa
  - Se faz necessário a criação de uma matriz dinâmica

- Existem duas soluções
  - Simular uma matriz com um vetor dinâmico
  - Criar um vetor de vetores dinâmicos (matriz dinâmica)

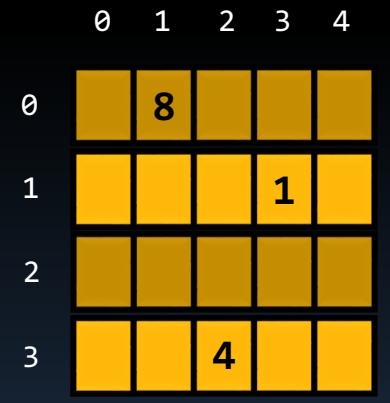
Matriz Dinâmica

- Simulando uma matriz com um vetor dinâmico

```
int linhas = 4;
int colunas = 5;
int * mat = new int[linhas*colunas];
```

mat[i*colunas + j]

```
int mat[4][5];
```



mat[i][j]

Matriz Dinâmica

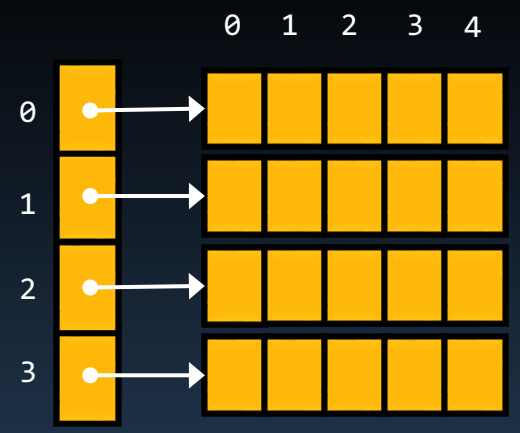
```cpp
#include <iostream>
using namespace std;

int main()
{
     cout << "Digite quantidade de linhas e colunas da matriz:";
     int linhas, colunas;
     cin >> linhas >> colunas;

     // usando um vetor para representar uma matriz
     int * mat = new int[linhas*colunas];

     // lendo elementos
     for (int i = 0; i < linhas; ++i)
        for (int j = 0; j < colunas; ++j)
                cin >> mat[i*colunas + j];

     delete [] mat;
}
```

Vetor2D.cpp

Matriz Dinâmica

- Criando um vetor de vetores dinâmicos

```
int linhas = 4;
int colunas = 5;
int ** mat = new int* [linhas];
```



Matriz Dinâmica é um vetor dinâmico de ponteiros

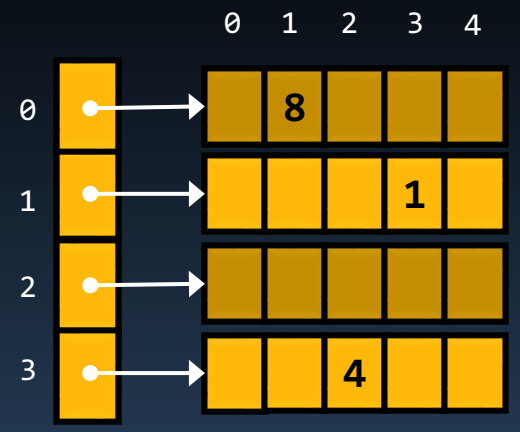
```
int*  * mat = new int* [linhas];
int   * vet = new int  [linhas];
```

Vetor Dinâmico de inteiros

Matriz Dinâmica

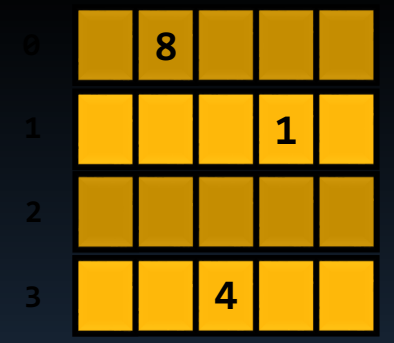
- Criando um vetor de vetores dinâmicos

```
int linhas = 4;
int colunas = 5;
int ** mat = new int*[linhas];
```




```
for (int i = 0; i < linhas; i++)
    mat[i] = new int[colunas];

mat[0][1] = 8;
mat[1][3] = 1;
mat[3][2] = 4;
```

```
int mat[4][5];
```



mat[i][j]

Matriz Dinâmica

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

int main()
{
    int linhas, colunas;
    cin >> linhas >> colunas;

    // criando a matriz dinâmica
    int ** mat = new int*[linhas];
    for (int i = 0; i < linhas; ++i)
        mat[i] = new int[colunas];

    // lendo elementos
    for (int i = 0; i < linhas; ++i)
       for (int j = 0; j < colunas; ++j)
               cin >> mat[i][j];

    // exibindo elementos
    for (int i = 0; i < linhas; ++i)
    {
        for (int j = 0; j < colunas; ++j)
            cout << mat[i][j] << " ";
        cout << endl;
    }

    // liberando a matriz dinâmica
    for (int i = 0; i < linhas; ++i)
        delete [] mat[i];
    delete [] mat;
}
```

Matriz.cpp

Resumo

- Os laços permitem a repetição de instruções
  - C++ oferece três variações de laços de repetição:
    - `for`, `while`, `do-while`
- A entrada do usuário pode ser lida caractere a caractere usando uma das versões da função cin.get( )
    - `ch = cin.get()`
    - `cin.get(ch)`
- Laços aninhados fornecem um método fácil para percorrer vetores bidimensionais