

Programação de Computadores

APLICAÇÕES DOS LAÇOS DE REPETIÇÃO

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";
}
```

Tratamento da Entrada

- A saída do programa:

Digite caracteres, use # para sair:

Ele pode correr#muito rápido

Ele pode correr#

13 caracteres lidos.

- A entrada com cin ignora os caractere de espaço e tabulação
- Ela é armazenada em um buffer
 - 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";
}
```

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";
}
```

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];
```

5	4	3	8	0	9	6	7	1	2
---	---	---	---	---	---	---	---	---	---

- Uma **matriz** é um **vetor bidimensional**

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

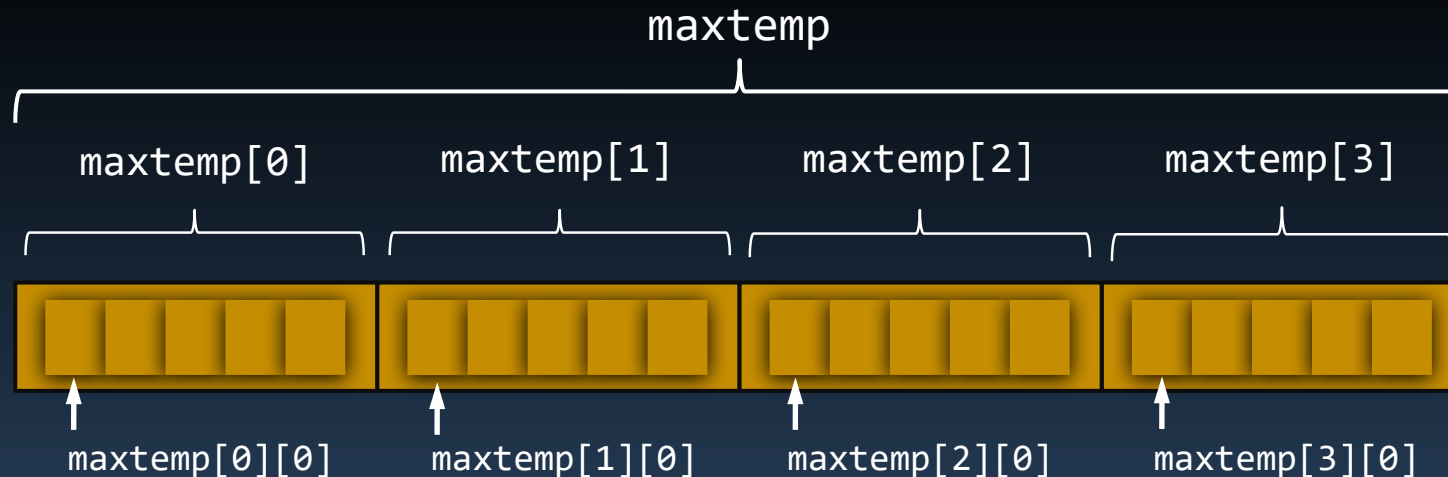
5	4	3	8	0
9	6	7	1	2
2	1	0	7	8
4	6	3	1	4

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



The diagram illustrates a 4x5 matrix. A vertical arrow on the left is labeled 'linhas' (rows) and points downwards, with indices 0, 1, 2, and 3. A horizontal arrow at the top is labeled 'colunas' (columns) and points to the right, with indices 0, 1, 2, 3, and 4. The matrix cells are yellow and contain the following expressions:

	0	1	2	3	4
0	maxtemp[0][0]	maxtemp[0][1]	maxtemp[0][2]	maxtemp[0][3]	maxtemp[0][4]
1	maxtemp[1][0]	maxtemp[1][1]	maxtemp[1][2]	maxtemp[1][3]	maxtemp[1][4]
2	maxtemp[2][0]	maxtemp[2][1]	maxtemp[2][2]	maxtemp[2][3]	maxtemp[2][4]
3	maxtemp[3][0]	maxtemp[3][1]	maxtemp[3][2]	maxtemp[3][3]	maxtemp[3][4]

Matrizes

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

```
for (int i = 0; i < MaxLinha; i++)  
{  
    for (int j = 0; j < MaxColuna; j++)  
        cout << maxtemp[i][j] << "\t";  
    cout << endl;  
}
```

Diagram illustrating the nested loop structure for processing a matrix:

- The outer loop (controlling the line) is represented by the label "controla linha" pointing to the `for (int i = 0; i < MaxLinha; i++)` loop.
- The inner loop (controlling the column) is represented by the label "controla coluna" pointing to the `for (int j = 0; j < MaxColuna; j++)` loop.

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 →

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;
}
}
```

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];
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	8							1									4		

`mat[i*colunas + j]`

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

	0	1	2	3	4
0		8			
1				1	
2					
3			4		

`mat[i][j]`

Matriz Dinâmica

```
#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];

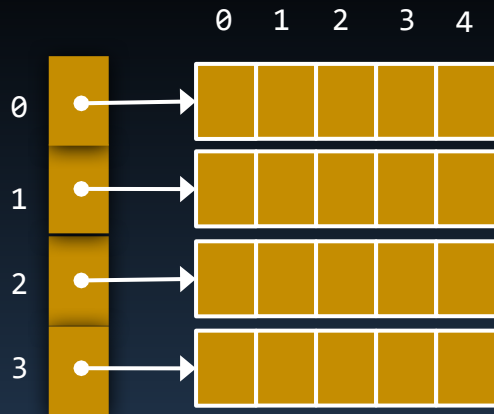
    // lendo elementos
    for (int i = 0; i < linhas; ++i)
        for (int j = 0; j < colunas; ++j)
            cin >> mat[i*colunas + j];

    delete [] mat;
}
```

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];

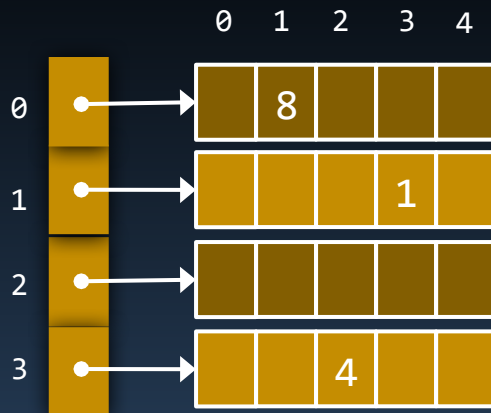
`int` * vet = new `int` [linhas];

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];
```

	0	1	2	3	4
0		8			
1				1	
2					
3			4		

mat[i][j]

Matriz Dinâmica

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

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

    // criando a matriz dinamica
    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;
    }

    // removendo matriz da memória
    for (int i = 0; i < linhas; ++i)
        delete [] mat[i];
    delete [] mat;
}
```


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