lista 06

Classes em C++

Podemos entender uma classe como um tipo de dados, semelhante aos registros (`struct`) da linguagem C.

Considere o seguinte código C++:

```
#include <cstdio>

class Posicao {
    public:
        int l;  // linha
        int c;  // coluna

        // método construtor, inicializa a posição
        Posicao(int linha, int coluna){
            l = linha;
            c = coluna;
        }
};

int main(){

    // criando um objeto do tipo Posicao na variável p
    Posicao p(10, 15);
    printf("A linha é %d e a coluna é %d\n\n", pl, pc);

    printf("Digite a linha da posição: ");
    scanf("%d", &p.l);
    printf("Digite a coluna da posição: ");
    scanf("%d", &p.c);
    printf("Agora a linha é %d e a coluna é %d\n\n", p.l, p.c);

    p.l = 17;
    p.c = 18;
    printf("Já agora a linha é %d e a coluna é %d\n\n", p.l, p.c);
}
```

Este código define uma classe chamada `Posicao` que tem dois atributos `l` e `c`. Além disso, a classe inclui um construtor (função de inicialização) que recebe o mesmo nome da classe.

O comando

```
    Posicao p(10, 15);
```

define uma variável `p` do tipo `Posicao`. Essa variável armazena um objeto do tipo `Posicao` que é criado neste momento. Ao criar o objeto, o método construtor que definimos é executado com os parâmetros `(10, 15)` e inicializa o objeto.

Em seguida, lemos do teclado dois valores inteiros e colocamos nos atributos `l` e `c`. E depois, nós atribuimos valores diretamente aos atributos do objeto.

Obs.: Não é obrigado definir um construtor para uma classe.

Vetores alocados dinamicamente em C++

Em C++, podemos criar um vetor `L` de inteiros da seguinte forma:

```
int* V;  // a variável V se refere a um vetor de inteiros
int n;  // a variável n guardará o tamanho da lista
...
V = new int[n];  // cria o vetor de inteiros L de tamanho n
```

Uma vez criado, o vetor pode ser usado normalmente:

```
V[i] = 327;
```

Analogamente, se quisermos definir um vetor de objetos do tipo `Posicao`, podemos fazer assim:

```
Posicao* V;  // a variável V se refere a um vetor de inteiros
int n;  // a variável n guardará o tamanho da lista
...
V = new Posicao[n];  // cria o vetor de inteiros L de tamanho n
```

Exemplo:

```
#include <cstdio>

class Posicao{
    public:
        int l;
        int c;
};

int main()
{
    Posicao* v;

    int n = 10;

    v = new Posicao[n];

    for (int i = 0; i < n; i++){
        v[i].l = i*i;
        v[i].c = i*i*i;
    }

    for (int i = 0; i < n; i++){
        printf("A linha é %d e a coluna é %d\n", v[i].l, v[i].c);
    }
}
```

Questão 1 (Fila de inteiros)**.** Considere o seguinte fragmento de código que define uma classe que é uma fila de inteiros. Complete os métodos `enfileirar(int x)`, `int desenfileirar()` e `int vazia()` e execute o código

```
#include <cstdio>

class FilaInt {
    public:
        int* v;
        int tam_max;
        int ini;
        int fim;
        int cont;

        // método construtor, inicializa a fila
        FilaInt(int tamanho){
            v = new int[tamanho];  // cria vetor de inteiros com o tamanho desejado
            tam_max = tamanho;
            ini = 0;
            fim = -1;
            cont = 0
        }

        // método enfileirar
        void enfileirar(int x){
            ...
        }

        // método desenfileirar
        int desenfileirar(){
            ...
        }

        // método fila vazia
        int vazia(){
            ...
        }
};

int main(){

    FilaInt f(10);  // cria fila de tamanho 10

    for (int i = 0; i < 10; i++){
        f.enfileirar(i);
    }

    for (int i = 0; i < 5; i++){
        printf("%d, ", f.desenfileirar());
    }
    printf("\n");

    for (int i = 0; i < 5; i++){
        f.enfileirar(i);
    }

    while (!f.vazia()){
        printf("%d, ", f.desenfileirar());
    }
}
```

Questão 2 (Fila de posições)**.** Considere o seguinte fragmento de código que define uma classe que é uma fila de posições. Complete os métodos `enfileirar(Posicao x)`, `Posicao desenfileirar()` e `int vazia()` e execute o código

```
#include <cstdio>

class Posicao{
    public:
        int l;
        int c;
};

class FilaPosicao {
    public:
        Posicao* v;
        int tam_max;
        int ini;
        int fim;
        int cont;

        // método construtor, inicializa a fila
        FilaPosicao(int tamanho){
            v = new Posicao[tamanho];  // cria vetor de posicoes com o tamanho desejad
            tam_max = tamanho;
            ini = 0;
            fim = -1;
            cont = 0
        }

        // método enfileirar
        void enfileirar(Posicao p){
            ...
        }

        // método desenfileirar
        Posicao desenfileirar(){
            ...
        }

        // método fila vazia
        int vazia(){
            ...
        }
};

int main(){

    FilaPosicao f(10);  // cria fila de tamanho 10

    Posicao p;

    for (int i = 0; i < 10; i++){
        p.l = i*i;
        p.c = i*i*i;
        f.enfileirar(p);
    }

    for (int i = 0; i < 5; i++){
        p = f.desenfileirar();
        printf("linha %d, coluna %d\n", p.l, p.c);
```

```
    }
    printf("\n");

    for (int i = 0; i < 5; i++){
        p.l = i*i;
        p.c = i*i*i;
        f.enfileirar(p);
    }

    while (!f.vazia()){
        p = f.desenfileirar();
        printf("linha %d, coluna %d\n", p.l, p.c);
    }
}
```

Questão 3 (Matriz de distâncias)**.** Considere uma matriz M de dimensões $n \times n$ preenchida com 0's e 1's.

$$\mathsf{M} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

A matriz de distâncias é a matriz D $n \times n$ onde cada posição é um número correspondente à menor distância entre aquela posição e alguma outra posição que contém um 1 na matriz original M.

$$\mathsf{D} = \begin{bmatrix} 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 0 \\ 7 & 6 & 5 & 4 & 3 & 3 & 3 & 2 & 1 & 1 \\ 6 & 5 & 4 & 3 & 2 & 2 & 3 & 3 & 2 & 2 \\ 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 3 & 3 \\ 4 & 3 & 2 & 1 & 0 & 0 & 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\ 3 & 3 & 2 & 1 & 1 & 0 & 1 & 2 & 3 & 4 \\ 2 & 3 & 3 & 2 & 2 & 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 2 & 3 \end{bmatrix}$$

Implemente o algoritmo visto em sala de aula que calcula a matriz de distâncias em tempo $O(n^2)$.

(Obs.: A seguir uma representação de algumas etapas do desenrolar do algoritmo.)

$$
\mathsf{D} = \begin{bmatrix}
* & * & * & * & * & * & * & * & 0 & 0 \\
* & * & * & * & * & * & * & * & * & * \\
* & * & * & * & * & * & * & * & * & * \\
* & * & * & * & * & * & * & * & * & * \\
* & * & * & * & 0 & 0 & * & * & * & * \\
* & * & * & 0 & 0 & 0 & 0 & * & * & * \\
* & * & * & * & * & 0 & * & * & * & * \\
* & * & * & * & * & * & * & * & * & * \\
* & * & * & * & * & * & * & * & * & * \\
0 & * & * & * & 0 & 0 & 0 & * & * & *
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
* & * & * & * & * & * & * & 1 & 0 & 0 \\
* & * & * & * & * & * & * & * & 1 & 1 \\
* & * & * & * & * & * & * & * & * & * \\
* & * & * & * & 1 & 1 & * & * & * & * \\
* & * & * & 1 & 0 & 0 & 1 & * & * & * \\
* & * & 1 & 0 & 0 & 0 & 0 & 1 & * & * \\
* & * & * & 1 & 1 & 0 & 1 & * & * & * \\
* & * & * & * & * & 1 & * & * & * & * \\
1 & * & * & * & 1 & 1 & 1 & * & * & * \\
0 & 1 & * & 1 & 0 & 0 & 0 & 1 & * & *
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
* & * & * & * & * & * & 2 & 1 & 0 & 0 \\
* & * & * & * & * & * & * & 2 & 1 & 1 \\
* & * & * & * & 2 & 2 & * & * & 2 & 2 \\
* & * & * & 2 & 1 & 1 & 2 & * & * & * \\
* & * & 2 & 1 & 0 & 0 & 1 & 2 & * & * \\
* & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & * \\
* & * & 2 & 1 & 1 & 0 & 1 & 2 & * & * \\
2 & * & * & 2 & 2 & 1 & 2 & * & * & * \\
1 & 2 & * & 2 & 1 & 1 & 1 & 2 & * & * \\
0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 2 & *
\end{bmatrix}
$$

$$
\Rightarrow
\begin{bmatrix}
* & * & * & * & * & 3 & 2 & 1 & 0 & 0 \\
* & * & * & * & 3 & 3 & 3 & 2 & 1 & 1 \\
* & * & * & 3 & 2 & 2 & 3 & 3 & 2 & 2 \\
* & * & 3 & 2 & 1 & 1 & 2 & 3 & 3 & 3 \\
* & 3 & 2 & 1 & 0 & 0 & 1 & 2 & 3 & * \\
3 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
3 & 3 & 2 & 1 & 1 & 0 & 1 & 2 & 3 & * \\
2 & 3 & 3 & 2 & 2 & 1 & 2 & 3 & * & * \\
1 & 2 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & * \\
0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 2 & 3
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
* & * & * & * & 4 & 3 & 2 & 1 & 0 & 0 \\
* & * & * & 4 & 3 & 3 & 3 & 2 & 1 & 1 \\
* & * & 4 & 3 & 2 & 2 & 3 & 3 & 2 & 2 \\
* & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 3 & 3 \\
4 & 3 & 2 & 1 & 0 & 0 & 1 & 2 & 3 & 4 \\
3 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
3 & 3 & 2 & 1 & 1 & 0 & 1 & 2 & 3 & 4 \\
2 & 3 & 3 & 2 & 2 & 1 & 2 & 3 & 4 & * \\
1 & 2 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & 4 \\
0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 2 & 3
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
* & * & * & 5 & 4 & 3 & 2 & 1 & 0 & 0 \\
* & * & 5 & 4 & 3 & 3 & 3 & 2 & 1 & 1 \\
* & 5 & 4 & 3 & 2 & 2 & 3 & 3 & 2 & 2 \\
5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 3 & 3 \\
4 & 3 & 2 & 1 & 0 & 0 & 1 & 2 & 3 & 4 \\
3 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
3 & 3 & 2 & 1 & 1 & 0 & 1 & 2 & 3 & 4 \\
2 & 3 & 3 & 2 & 2 & 1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & 4 \\
0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 2 & 3
\end{bmatrix}
$$

$$
\Rightarrow
\begin{bmatrix}
* & * & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 0 \\
* & 6 & 5 & 4 & 3 & 3 & 3 & 2 & 1 & 1 \\
6 & 5 & 4 & 3 & 2 & 2 & 3 & 3 & 2 & 2 \\
5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 3 & 3 \\
4 & 3 & 2 & 1 & 0 & 0 & 1 & 2 & 3 & 4 \\
3 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
3 & 3 & 2 & 1 & 1 & 0 & 1 & 2 & 3 & 4 \\
2 & 3 & 3 & 2 & 2 & 1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & 4 \\
0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 2 & 3
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
* & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 0 \\
7 & 6 & 5 & 4 & 3 & 3 & 3 & 2 & 1 & 1 \\
6 & 5 & 4 & 3 & 2 & 2 & 3 & 3 & 2 & 2 \\
5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 3 & 3 \\
4 & 3 & 2 & 1 & 0 & 0 & 1 & 2 & 3 & 4 \\
3 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
3 & 3 & 2 & 1 & 1 & 0 & 1 & 2 & 3 & 4 \\
2 & 3 & 3 & 2 & 2 & 1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & 4 \\
0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 2 & 3
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 0 \\
7 & 6 & 5 & 4 & 3 & 3 & 3 & 2 & 1 & 1 \\
6 & 5 & 4 & 3 & 2 & 2 & 3 & 3 & 2 & 2 \\
5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 3 & 3 \\
4 & 3 & 2 & 1 & 0 & 0 & 1 & 2 & 3 & 4 \\
3 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
3 & 3 & 2 & 1 & 1 & 0 & 1 & 2 & 3 & 4 \\
2 & 3 & 3 & 2 & 2 & 1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & 4 \\
0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 2 & 3
\end{bmatrix}
$$