C++数组

一、一维数组

1. 基本概念

- **定义**：数组是相同类型元素的集合，通过下标访问元素
- **特点**：连续内存空间、固定长度、下标从0开始

2. 声明与初始化

```
// 声明方式
数据类型 数组名[数组长度];

// 示例
int nums[5];            // 声明长度为5的整型数组
double scores[10];      // 声明长度为10的双精度数组

// 初始化方式
int arr1[3] = {1, 2, 3};        // 完全初始化
int arr2[5] = {1, 2};            // 部分初始化，剩余元素自动为0
int arr3[] = {1, 3, 5, 7};       // 自动推导长度（长度为4）
```

3. 访问元素

```
int arr[5] = {10, 20, 30, 40, 50};

// 通过下标访问
cout << arr[0];     // 输出10
arr[2] = 100;       // 修改第三个元素

// 遍历数组
for(int i = 0; i < 5; i++) {
    cout << arr[i] << " ";
}
```

4. 内存结构

- 元素连续存储
- 数组名是首元素的地址

```
int arr[3] = {1, 2, 3};
cout << arr;      // 输出数组首地址（如0x7ffeedd26b1c）
```

5. 常见应用示例

示例1：求数组最大值

```
int nums[] = {12, 45, 23, 67, 8};
int max = nums[0];
for(int num : nums) {
    if(num > max) max = num;
}
cout << "最大值: " << max;
```

示例2：数组排序（冒泡排序）

```
int arr[] = {5, 3, 8, 1, 4};
int n = sizeof(arr)/sizeof(arr[0]);

for(int i = 0; i < n-1; i++) {
    for(int j = 0; j < n-i-1; j++) {
        if(arr[j] > arr[j+1]) {
            swap(arr[j], arr[j+1]);
        }
    }
}
```

二、二维数组

1. 基本概念

- 可以看作"数组的数组"
- 常用于表示表格数据、矩阵等

2. 声明与初始化

```
// 声明方式
数据类型 数组名[行数][列数];

// 示例
int matrix[3][4];    // 3行4列数组

// 初始化方式
int grid1[2][3] = {{1,2,3}, {4,5,6}};  // 完全初始化
int grid2[][2] = {{1}, {3,4}, {5}};     // 自动推导行数（3行）
```

3. 访问元素

```cpp
int matrix[2][3] = {{1,2,3}, {4,5,6}};

// 访问第二行第三列元素
cout << matrix[1][2];  // 输出6

// 遍历二维数组
for(int i = 0; i < 2; i++) {
    for(int j = 0; j < 3; j++) {
        cout << matrix[i][j] << " ";
    }
    cout << endl;
}
```

4. 内存结构

- 按行优先顺序连续存储
- 实际内存仍然是线性的

5. 常见应用示例

示例1：矩阵转置

```cpp
const int N = 3;
int original[N][N] = {{1,2,3}, {4,5,6}, {7,8,9}};
int transposed[N][N];

for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
        transposed[j][i] = original[i][j];
    }
}
```

示例2：学生成绩管理

```cpp
const int STUDENTS = 5;
const int COURSES = 3;
float scores[STUDENTS][COURSES];

// 输入成绩
for(int i = 0; i < STUDENTS; i++) {
    cout << "输入第" << i+1 << "个学生的三门成绩：";
    for(int j = 0; j < COURSES; j++) {
        cin >> scores[i][j];
    }
}
```

三、注意事项

1. **数组越界**：访问不存在的索引会导致未定义行为
2. **数组长度**：使用`sizeof(arr)/sizeof(arr[0])`获取数组长度
3. **数组传参**：传递数组给函数时实际传递的是指针
4. **动态数组**：普通数组长度必须编译时确定，动态数组需用`new`创建

数组作为函数参数

```
// 一维数组
void printArray(int arr[], int size) {
    // ...
}

// 二维数组（必须指定列数）
void printMatrix(int mat[][4], int rows) {
    // ...
}
```

四、总结

- 一维数组用于线性数据存储
- 二维数组适合表格型数据
- 数组访问高效但长度固定
- 实际开发中可结合标准库容器（如vector）使用

可根据听众基础调整讲解深度，建议配合现场代码演示和练习题加深理解。