一、`scanf`与`cin`对比讲稿

1.1 基本功能概述

- `scanf`：
  - C语言中的输入函数，C++中也可使用（需包含`<cstdio>`）。
  - 通过格式说明符（如`%d`、`%f`）读取数据。
  - 支持基本数据类型输入。
- `cin`：
  - C++标准库中的输入流对象。
  - 通过`>>`操作符读取数据。
  - 支持面向对象编程，可与 C++ 特性（如类、模板）结合。

1.2 输入各种数据类型对比

1.2.1 整型数据

- `scanf`：

```c
#include <cstdio>
int main()
{
    int num;
    printf("输入整数：");
    scanf("%d", &num);
    printf("你输入的是：%d\n", num);
    return 0;
}
```

  - 需要显式指定格式说明符（如`%d`）。
  - 可读取多种整型（`int`、`long`、`unsigned int`等）。
- `cin`：

```cpp
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout << "输入整数：";
    cin >> num;
    cout << "你输入的是：" << num << endl;
    return 0;
}
```

- 自动识别变量类型，无需格式说明符。
- 简洁易用，适合 C++ 风格编程。

1.2.2 浮点型数据

- **`scanf`**：

```
#include <cstdio>
int main()
{
    float f;
    double d;
    printf("输入 float 和 double: ");
    scanf("%f %lf", &f, &d);
    printf("float: %.2f, double: %.2lf\n", f, d);
    return 0;
}
```

  - 格式说明符分别为 `%f`（`float`）、`%lf`（`double`）。
- **`cin`**：

```
#include <iostream>
using namespace std;
int main()
{
    float f;
    double d;
    cout << "输入 float 和 double: ";
    cin >> f >> d;
    cout << "float: " << f << ", double: " << d << endl;
    return 0;
}
```

  - 同样无需格式说明符，自动识别类型。

1.2.3 字符型数据

- **`scanf`**：

```
#include <cstdio>
int main()
{
    char c;
    printf("输入字符: ");
    scanf(" %c", &c); // 前导空格跳过空格符
    printf("你输入的是: %c\n", c);
    return 0;
}
```

  - 使用 `%c` 格式说明符，前导空格可跳过空白符（如换行符）。

- `cin`：

```cpp
#include <iostream>
using namespace std;
int main()
{
    char c;
    cout << "输入字符：";
    cin >> c;
    cout << "你输入的是：" << c << endl;
    return 0;
}
```

  - 自动跳过空白符，无需特殊处理。

1.2.4 字符串数据

- `scanf`：

```cpp
#include <cstdio>
int main()
{
    char str[50];
    printf("输入字符串（不含空格）：");
    scanf("%s", str);
    printf("你输入的是：%s\n", str);
    return 0;
}
```

  - 使用 `%s` 只能读取单词（以空格分隔）。
- `cin`：

```cpp
#include <iostream>
using namespace std;
int main()
{
    string str;
    cout << "输入字符串（包含空格）：";
    getline(cin, str); // 读取整行
    cout << "你输入的是：" << str << endl;
    return 0;
}
```

  - 使用 `getline(cin, str)` 可读取包含空格的字符串。

1.3 优劣势对比分析

1.3.1 优势对比

- `scanf`：

- **高效**：执行效率高，适合大量数据输入。
- **格式可控**：通过格式说明符精确控制输入格式。

- `cin`：
  - **易用**：自动识别变量类型，代码简洁。
  - **现代**：符合 C++ 面向对象编程风格，支持流操作。

1.3.2 劣势对比

- `scanf`：
  - **格式敏感**：格式说明符与输入不匹配易出错。
  - **不易用**：无法直接输入复杂对象或处理高级输入。
- `cin`：
  - **性能较低**：效率低于 `scanf`。
  - **灵活性受限**：部分格式控制需额外操作。

1.4 适用场景

- `scanf`：
  - 适用于简单、高效的输入场景，尤其是对格式要求严格的 C 语言项目。
- `cin`：
  - 适用于复杂的 C++ 项目，需与类、模板等特性配合，或对代码可读性、可维护性要求较高的场景。

二、源代码汇总

2.1 `scanf` 示例代码

```
#include <cstdio>
int main()
{
    int num;
    float f;
    double d;
    char c;
    char str[50];

    printf("输入整数: ");
    scanf("%d", &num);
    printf("输入浮点数: ");
    scanf("%f", &f);
    printf("输入双精度数: ");
    scanf("%lf", &d);
    printf("输入字符: ");
    scanf(" %c", &c);
    printf("输入字符串: ");
    scanf("%s", str);

    printf("整数: %d\n", num);
```

```
    printf("浮点数：%.2f\n", f);
    printf("双精度数：%.2lf\n", d);
    printf("字符：%c\n", c);
    printf("字符串：%s\n", str);

    return 0;
}
```

2.2 `cin` 示例代码

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

int main()
{
    int num;
    float f;
    double d;
    char c;
    string str;

    cout << "输入整数：";
    cin >> num;
    cout << "输入浮点数：";
    cin >> f;
    cout << "输入双精度数：";
    cin >> d;
    cout << "输入字符：";
    cin >> c;
    cout << "输入字符串：";
    cin.ignore(); // 忽略输入缓冲区中的换行符
    getline(cin, str);

    cout << "整数：" << num << endl;
    cout << "浮点数：" << f << endl;
    cout << "双精度数：" << d << endl;
    cout << "字符：" << c << endl;
    cout << "字符串：" << str << endl;

    return 0;
}
```

三、 `scanf` 与 `cin` 的对比

特性	`scanf`	`cin`
格式控制	需要显式指定格式说明符（如 `%d`）	自动识别变量类型（无需格式说明符）
输入效率	较高，适合大量简单数据输入	略低，但能满足大多数需求
安全性	格式敏感，输入不匹配易出错	支持类型检查和异常处理
适用场景	C 语言项目或高效数据读取场景	C++ 项目及复杂输入控制场景
字符串输入	需额外处理空格和多行	通过 `getline` 灵活处理字符串输入

四、 总结

- `scanf`：适用于对输入效率要求较高且格式固定的场景，如嵌入式开发等。
- `cin`：适用于现代 C++ 开发，具有更高的代码可读性和安全性，支持更灵活的输入操作。

在实际开发中，可根据项目的需求和语言风格选择合适的输入方式。