printf函数

在C++中，`printf` 是一个源自C语言的标准库函数，用于格式化输出。尽管C++更推荐使用类型安全的 `iostream` 库（如 `cout`），但 `printf` 在控制输出格式时更为灵活。以下是详细讲解：

---

1. 头文件

使用 `printf` 需要包含C标准输入输出头文件：

```
#include <cstdio>   // C++风格头文件
// 或
#include <stdio.h>  // C风格头文件（不推荐在C++中直接使用）
```

---

2. 基本语法

```
int printf(const char* format, ...);
```

- **参数**：
  - `format`：格式字符串，包含普通字符和格式说明符（如 `%d`, `%f`）。
  - `...`：可变参数列表，按格式说明符的顺序提供数据。
- **返回值**：成功输出的字符数，出错时返回负值。

---

3. 格式说明符

格式说明符以 `%` 开头，后跟类型字符。常用说明符：

说明符	类型	示例
`%d`	整数（int）	`printf("%d", 42);`
`%u`	无符号整数	`printf("%u", 100u);`
`%ld`	长整数（long）	`printf("%ld", 1000L);`
`%f`	浮点数（float/double）	`printf("%f", 3.14);`
`%lf`	双精度浮点数（double）	`printf("%lf", 3.14);`
`%c`	字符（char）	`printf("%c", 'A');`
`%s`	字符串（const char*）	`printf("%s", "Hello");`
`%p`	指针地址	`printf("%p", &x);`
`%x`	十六进制整数	`printf("%x", 255);`
`%%`	输出百分号	`printf("%%");`

---

4. 格式控制

在 `%` 后可以添加修饰符控制输出格式：

宽度与对齐

- `%10d`：输出宽度为10，右对齐。
- `%-10d`：左对齐。

```
printf("%10d", 123);   // 输出 "       123"
printf("%-10d", 123);  // 输出 "123       "
```

精度

- `%.2f`：保留2位小数。
- `%5.2f`：总宽度5，保留2位小数。

```
printf("%.2f", 3.1415); // 输出 "3.14"
```

其他修饰符

- `%+d`：显示正负号。
- `%04d`：用0填充宽度。

```
printf("%+d", 42);   // 输出 "+42"
printf("%04d", 42);  // 输出 "0042"
```

---

5. 转义字符

在格式字符串中使用转义字符：

- `\n`：换行
- `\t`：制表符
- `\\`：输出反斜杠

```
printf("Line1\nLine2\tTabbed\n");
```

---

6. 注意事项

1. **类型匹配**：参数类型必须与格式说明符严格匹配，否则可能导致未定义行为。

```
int num = 100;
printf("%f", num); // 错误！应用 %d
```

2. **C++字符串处理**：使用 `std::string` 时需转换为C风格字符串：

```
std::string name = "Alice";
printf("Name: %s", name.c_str()); // 正确
```

3. **安全性**：`printf` 不会检查参数数量或类型，建议优先使用 `cout` 避免错误。

---

7. 示例代码

```
#include <cstdio>
#include <string>

int main()
{
    int age = 25;
    double score = 85.5;
    std::string name = "Bob";

    printf("Name: %s\nAge: %d\nScore: %.1f\nHex: 0x%x\n",
           name.c_str(), age, score, 255);
    return 0;
}
```

输出：

```
Name: Bob
Age: 25
Score: 85.5
Hex: 0xff
```

---

8. 对比 `cout` 与 `printf`

- **`cout`**：类型安全，支持运算符重载，但格式控制稍繁琐（需 `<iomanip>`）。
- **`printf`**：格式灵活，但风险较高（如类型错误）。

掌握 `printf` 的用法有助于在需要精细控制输出时快速实现，但在C++中更推荐优先使用 `cout` 以提高安全性。