以下是 C++ 中整型（Integer）和浮点型（Floating-Point）相关知识的总结：

整型（Integer）

整型用于存储没有小数的数字，主要用于计数或整数运算。

基本类型

1. `int`：
   - 默认整型，通常占用 4 字节。
   - 常见取值范围：`-2,147,483,648` 到 `2,147,483,647`（有符号）。
2. `short`：
   - 短整型，通常占用 2 字节。
   - 取值范围：`-32,768` 到 `32,767`。
3. `long`：
   - 长整型，通常占用 4 或 8 字节（根据编译环境和系统而定）。
   - 取值范围更大。
4. `long long`：
   - 更大的整型，通常占用 8 字节。
   - 取值范围：`-9,223,372,036,854,775,808` 到 `9,223,372,036,854,775,807`。

特点

- **占用内存**：整型的字节大小可以通过 `sizeof()` 查看。
- **溢出行为**：
  - 根据 C++ 标准（C++98 到 C++20），**整数溢出是未定义行为（Undefined Behavior, UB）**。
    - UB 意味着程序的行为无法预测，可能导致程序崩溃、错误计算结果，甚至看似正常运行。
    - 不同的编译器或编译优化选项可能会处理这种情况，但结果不可依赖。

示例：

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

int main() {
    int a = INT_MAX; // 最大值 2147483647
    int b = a + 1;   // 溢出
    cout << "b: " << b << endl; // 结果未定义
    return 0;
}
```

- 在一些编译器中，可能会得到负值 `-2147483648`（因为二进制表示下会循环到最小值）。
- 在某些场景下，可能导致程序直接崩溃。

浮点型（Floating-Point）

浮点型用于存储带小数的数字，通常适合表示连续数据或执行高精度运算。

基本类型

1. `float`：
   - 单精度浮点数，通常占用 4 字节。
   - 精度约 7~8 位十进制数。
   - 常见范围：`1.2E-38` 到 `3.4E+38`。
2. `double`：
   - 双精度浮点数，通常占用 8 字节。
   - 精度约 15-16 位十进制数。
   - 常见范围：`2.3E-308` 到 `1.7E+308`。
3. `long double`：
   - 扩展精度浮点数，通常占用 10、12 或 16 字节（根据编译环境和系统而定）。
   - 提供更高精度。

特点

- **科学计数法**：
  - 可用 `E` 或 `e` 表示指数。例如：`3.14e2` 等价于 `314.0`。
- **默认精度**：
  - 若定义小数，默认类型为 `double`，需显式加后缀 `f` 表示 `float`。
  - 示例：`3.14f` 为 `float`，`3.14` 为 `double`。
- **溢出行为**：
  - 在 C++ 中，浮点型数据类型（如 `float`、`double`、`long double`）发生溢出时，其行为是定义良好的（Well-Defined Behavior），不会导致未定义行为（Undefined Behavior, UB）。溢出会遵循 IEEE 754 浮点数标准（现代计算机几乎都遵循这一标准）进行处理。

整型与浮点型的比较

特性	整型	浮点型
存储内容	整数	带小数的连续值
存储范围	相对较小	范围更大
精度	精确到个位	存在精度损失
内存占用	较小	较大
典型用途	计数、索引	物理计算、统计分析
操作速度	通常更快	通常较慢（涉及浮点运算单元）