Iostream 标准库输入函数 cin

`cin` 是 C++ 标准库中的一个全局对象，用于从标准输入（通常是键盘）读取数据。它是 `istream` 类的一个实例，`istream` 是输入流类的基本类型。`cin` 是 C++ 输入输出流库（I/O Streams Library）的一部分，通常与 `cout`（用于输出）和 `cerr`（用于错误输出）一起使用。

1.基本用法

`cin` 通常与提取运算符（`>>`）一起使用，从标准输入读取数据并存储到变量中。例如：

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

int main() {
    int number;
    cout << "Enter a number: ";
    cin >> number;
    cout << "You entered: " << number << endl;
    return 0;
}
```

在这个例子中，`cin >> number;` 从标准输入读取一个整数并存储到变量 `number` 中。

2.常见功能

1. **读取不同类型的数据**：
    - `cin` **可以读取多种数据类型，包括整数、浮点数、字符、字符串**等。

    ```
    int i;
    double d;
    char c;
    string s;

    cin >> i >> d >> c >> s;
    ```

2. **处理输入错误**：
    - `cin` 提供了多种成员函数来处理输入错误，如 `cin.fail()`、`cin.clear()` 和 `cin.ignore()`。

    ```
    int number;
    cin >> number;
    if (cin.fail()) {
        cin.clear(); // 清除错误标志
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // 忽略错误输入
        cout << "Invalid input!" << endl;
    }
    ```

3. **读取整行输入**：
    - **使用** `cin.getline()` 或 `getline()` 函数可以读取整行输入。

```
string line;
getline(cin, line);
```

3.注意事项

- **缓冲区**：`cin` 使用缓冲区来存储输入的数据，这意味着**输入的数据在被程序读取之前会先存储在缓冲区中。**
- **空白字符**：默认情况下，`cin` **会忽略空白字符（如空格、换行符）直到遇到有效的输入数据。**
  - **注意：如果涉及字符串和数字的同时输入情况，有可能字符串会输入之前的空格**
  - `(cin>>num).get()`
  - 使用C++的string类不会产生这个问题
- **输入同步**：`cin` 和 `cout` 默认是同步的，这意味着在输入之前会刷新输出缓冲区。可以通过 `cin.tie(nullptr)` 来解除同步以提高性能。
  - “输入同步”以及 `cin` 和 `cout` 的同步行为。

输入同步

在 C++ 中，`cin` 和 `cout` 是两个独立的流对象，分别用于处理输入和输出。为了确保输入和输出的顺序正确，标准库默认将这两个流对象进行同步。这种同步机制意味着在每次使用 `cin` 进行输入操作之前，`cout` 的输出缓冲区会被刷新（即，所有待输出的数据会被立即输出到控制台）。这种机制确保了用户在看到提示信息后才输入数据，而不会因为缓冲区未刷新而错过提示信息。

示例

考虑以下代码：

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

int main() {
    cout << "Enter a number: ";
    int number;
    cin >> number;
    cout << "You entered: " << number << endl;
    return 0;
}
```

在这段代码中，`cout << "Enter a number: ";` 会输出提示信息，然后 `cin >> number;` 会等待用户输入。在输入操作之前，`cout` 的缓冲区会被刷新，确保提示信息先显示出来。

解除同步

在某些情况下，程序员可能希望解除 `cin` 和 `cout` 之间的同步，以提高性能。特别是在需要大量输入输出操作的程序中，解除同步可以减少不必要的缓冲区刷新，从而提高效率。可以使用 `cin.tie(nullptr)` 来解除这种同步关系。

示例

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

int main() {
    ios::sync_with_stdio(false); // 关闭与 C 标准库的同步
    cin.tie(nullptr); // 解除 cin 和 cout 的同步

    cout << "Enter a number: ";
    int number;
    cin >> number;
    cout << "You entered: " << number << endl;
    return 0;
}
```

在这段代码中，`cin.tie(nullptr);` 解除 `cin` 和 `cout` 的同步，这意味着 `cout` 的缓冲区不会在每次 `cin` 输入操作之前自动刷新。`ios::sync_with_stdio(false);` 进一步关闭 C++ 流与 C 标准库 I/O 的同步，以提高性能。

注意事项

解除同步后，需要注意以下几点：

1. **手动刷新输出**：解除同步后，`cout` 的输出可能不会立即显示在控制台上。可以使用 `cout.flush();` 或 `endl` 来手动刷新输出缓冲区。

   ```
   cout << "Enter a number: " << flush;
   ```

2. **调试和可读性**：在调试和开发过程中，解除同步可能会导致输出顺序混乱，影响程序的可读性。因此，通常只在性能要求较高的场合使用这一技巧。

总结

- **同步**：默认情况下，`cin` 和 `cout` 是同步的，以确保输入输出顺序正确。
- **解除同步**：可以通过 `cin.tie(nullptr)` 解除同步，以提高性能，但需要手动管理输出缓冲区的刷新。

4.常见成员函数的用法

`cin` 是 C++ 标准库中的输入流对象，用于从标准输入（通常是键盘）读取数据。`cin` 是 `istream` 类的一个实例，因此它继承了 `istream` 类的所有成员函数。以下是一些常见且有用的 `cin` 成员函数及其用途：

1. `cin >>`

- **作用**：用于从标准输入读取数据并存储到变量中。
- **示例**：

  ```
  int number;
  cin >> number;
  ```

2. `cin.get()`

- **作用**：读取单个字符，包括空白字符（如空格、换行符）。
- **示例**：

```
char ch;
cin.get(ch);
```

3. `cin.getline()`

- **作用**：从输入流读取一行字符，直到遇到换行符或达到指定的字符数。
- 在string库中，也可以使用getline(cin,nameofstring)
- **示例**：

```
char buffer[100];
cin.getline(buffer, 100);
```

4. `cin.ignore()`

- **作用**：忽略输入流中的字符，常用于清除输入缓冲区。
- 参数：忽略缓冲区中接下来的三个字符。
- **示例**：

```
cin.ignore(numeric_limits<streamsize>::max(), '\n');
//常用代码，需要用到limits库，可用于清空缓冲区
```

5. `cin.peek()`

- **作用**：返回下一个字符，但不从输入流中提取它。
- **示例**：

```
char nextChar = cin.peek();
```

6. `cin.putback()`

- **作用**：将一个字符**放回输入流的当前位置**，以便下次读取时仍能读取到该字符。
- **示例**：

```
char ch;
cin >> ch;
//一些代码导致缓冲区改变
cin.putback(ch);
//在这里ch被重新写入缓冲区
```

7. `cin.eof()`

- **作用**：检查输入流是否到达文件末尾（EOF）。
- **示例**：

```
while (!cin.eof()) {
    int number;
    cin >> number;
    // 处理输入
}
```

8. `cin.fail()`

- **作用**：检查输入流是否处于错误状态。
- **示例**：

```
int number;
cin >> number;
if (cin.fail()) {
    cout << "Input error!" << endl;
}
```

9. `cin.clear()`

- **作用**：清除输入流的错误状态标志，使其可以继续使用。
- **示例**：

```
if (cin.fail()) {
    cin.clear(); // 清除错误标志
    cin.ignore(numeric_limits<streamsize>::max(), '\n'); // 忽略错误输入
}
```

10. `cin.sync()`

- **作用**：清除输入流的缓冲区。
- **示例**：

```
cin.sync();
```

11. `cin.rdbuf()`

- **作用**：获取或设置输入流的缓冲区。
- **示例**：

```
streambuf* pbuf = cin.rdbuf();
```

12. `cin.tie()`

- **作用**：将输入流与输出流绑定，以确保在输入操作之前刷新输出流。
- **示例**：

```
cin.tie(&cout);
```

示例程序：

```
#include <iostream>
#include <limits> // for numeric_limits
using namespace std;

int main() {
    int number;
    char ch;
    string str;

    // 1. cin.get() - 读取单个字符
    cout << "Enter a single character: ";
    ch = cin.get();
    cout << "You entered: " << ch << endl;

    // 清除输入缓冲区
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    // 2. cin.getline() - 读取一行字符串
    cout << "Enter a line of text: ";
    char buffer[100];
    cin.getline(buffer, 100);
    cout << "You entered: " << buffer << endl;

    // 3. cin.ignore() - 忽略输入流中的字符
    cout << "Enter a number (will ignore next 3 characters): ";
    cin >> number;
    cin.ignore(3);
    cout << "You entered: " << number << endl;

    // 4. cin.peek() - 查看下一个字符，但不提取
    cout << "Enter another number: ";
    cin >> number;
    char nextChar = cin.peek();
    cout << "Next character in the stream (peek): " << nextChar << endl;

    // 5. cin.putback() - 将一个字符放回输入流
    cin.putback(nextChar);
    cout << "Next character after putback: " << (char)cin.get() << endl;

    // 6. cin.sync() - 同步输入流
    cout << "Syncing input stream..." << endl;
    cin.sync();

    // 7. cin.clear() - 清除错误标志
    cin.clear();
```

```
    cout << "Enter a number (enter a non-numeric value to generate error): ";
    cin >> number;
    if (cin.fail()) {
        cout << "Error detected. Clearing error state..." << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    // 8. cin.eof() - 检查是否到达输入流的末尾
    cout << "Enter a number (Ctrl+D to trigger EOF): ";
    while (cin >> number) {
        cout << "You entered: " << number << endl;
    }
    if (cin.eof()) {
        cout << "End of file reached." << endl;
        cin.clear(); // 清除EOF状态
    }

    // 9. cin.fail() - 检查输入流是否遇到错误
    cout << "Enter a number (enter a non-numeric value to generate error): ";
    cin >> number;
    if (cin.fail()) {
        cout << "Input failed. Clearing error state..." << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    // 10. cin.good() - 检查输入流是否没有遇到错误
    cout << "Enter a number: ";
    cin >> number;
    if (cin.good()) {
        cout << "Input successful: " << number << endl;
    }

    // 11. cin.bad() - 检查输入流是否遇到不可恢复的错误
    if (cin.bad()) {
        cout << "Unrecoverable error detected." << endl;
    }

    // 12. cin.gcount() - 返回最后一次读取的字符数
    cin.ignore(numeric_limits<streamsize>::max(), '\n'); // 清除缓冲区
    cout << "Enter a line of text: ";
    cin.getline(buffer, 100);
    cout << "You entered: " << buffer << endl;
    cout << "Characters read: " << cin.gcount() << endl;

    return 0;
}
```

输出结果及解释

1. `cin.get()`:
    - 输入单个字符并输出。
    - 输出示例：`Enter a single character: a`, `You entered: a`

2. `cin.getline()`:
    - 读取一行字符串并输出。
    - 输出示例：`Enter a line of text: Hello, World!`，`You entered: Hello, World!`
3. `cin.ignore()`:
    - 忽略输入流中的字符。
    - 输出示例：`Enter a number (will ignore next 3 characters): 123456`，`You entered: 123`
4. `cin.peek()`:
    - 查看下一个字符但不提取。
    - 输出示例：`Enter another number: 789`，`Next character in the stream (peek): 9`
5. `cin.putback()`:
    - 将一个字符放回输入流。
    - 输出示例：`Next character after putback: 9`
6. `cin.sync()`:
    - 同步输入流。
    - 输出示例：`Syncing input stream...`
7. `cin.clear()`:
    - 清除错误标志。
    - 输出示例：`Enter a number (enter a non-numeric value to generate error): abc`，`Error detected. Clearing error state...`
8. `cin.eof()`:
    - 检查是否到达输入流的末尾。
    - 输出示例：`Enter a number (Ctrl+D to trigger EOF): 1`，`You entered: 1`，`End of file reached.`
9. `cin.fail()`:
    - 检查输入流是否遇到错误。
    - 输出示例：`Enter a number (enter a non-numeric value to generate error): abc`，`Input failed. Clearing error state...`
10. `cin.good()`:
    - 检查输入流是否没有遇到错误。
    - 输出示例：`Enter a number: 123`，`Input successful: 123`
11. `cin.bad()`:
    - 检查输入流是否遇到不可恢复的错误。
    - 输出示例：如果没有不可恢复的错误，不会有输出。
12. `cin.gcount()`:
    - 返回最后一次读取的字符数。
    - 输出示例：`Enter a line of text: Hello, World!`，`You entered: Hello, World!`，`Characters read: 13`

这个示例程序涵盖了 `cin` 的常见成员函数的使用，并通过注释和输出示例解释了每个函数的作用。如果你有更多问题或需要进一步的解释，请告诉我！

常见用法：

①清空缓冲区

method1 使用 `cin.ignore()`

这是最常用的方法，可以忽略指定数量的字符，直到遇到特定的终止字符（通常是换行符 `\n`）。

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

int main() {
    // 清空输入缓冲区，忽略所有字符直到遇到换行符
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    return 0;
}
```

method2 暴力清除：`cin.sync()`

`cin.sync()` 是另一种清空缓冲区的方法，它会清除所有未读取的输入字符。虽然它的效率通常不如 `cin.ignore()` 高，但在某些情况下可能更方便。

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

int main() {
    // 清空输入缓冲区
    cin.sync();
    return 0;
}
```

method3 while空循环

另一种方法是手动读取并丢弃所有字符，直到遇到换行符。这种方法在某些情况下可能更高效，特别是当你只关心丢弃特定数量的字符时。

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

int main() {
    char ch;
    // 手动读取并丢弃字符，直到遇到换行符
    while (cin.get(ch) && ch != '\n');
    return 0;
}
```

②填充数组问题

method1 使用sstream读取整行输入

```
#include <iostream>
#include <vector>
#include <sstream>  // for std::istringstream
#include <string>   // for std::string

int main() {
    std::vector<int> numbers;
    std::string input;

    std::cout << "Enter a line of numbers separated by spaces: ";
    std::getline(std::cin, input);  // 读取整行输入

    std::istringstream iss(input);  // 创建字符串流
    int number;

    // 逐个读取字符串流中的数字，并存入vector
    while (iss >> number) {
        numbers.push_back(number);
    }

    // 输出存储在vector中的数字
    std::cout << "You entered: ";
    for (const int& num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

method2 巧用cin的返回值

补充：cin的返回值

`std::cin` 是 C++ 标准库中的一个全局对象，用于从标准输入（通常是键盘）读取数据。`std::cin` 的返回值是一个 `std::istream&` 类型的引用，这意味着你可以将多个输入操作链接在一起。此外，`std::cin` 还可以用于检查输入操作的状态。

常见用法和返回值

1. **链式输入操作**

由于`std::cin` **返回的是一个** `std::istream&` **类型的引用**，当我们说 `std::cin` 返回一个 `std::istream&` 类型的引用时，意思是 `std::cin` 返回了一个指向它自己的“指针”。这个指针允许我们继续使用 `std::cin` 来做更多的事情，比如读取更多的数据或者检查输入是否成功。你可以将**多个输入操作**链接在一起：

```cpp
#include <iostream>

int main() {
    int a, b, c;
    std::cout << "Enter three integers: ";
    std::cin >> a >> b >> c;
    std::cout << "You entered: " << a << ", " << b << ", " << c << std::endl;
    return 0;
}
```

在这个例子中，`std::cin >> a >> b >> c` 依次读取三个整数，并将它们存储在 `a`，`b`，和 `c` 中。

2. **检查输入状态**

`std::cin` 还可以用于检查输入操作是否成功。你可以使用它的返回值来判断输入是否有效：

```cpp
#include <iostream>

int main() {
    int number;
    std::cout << "Enter an integer: ";
    if (std::cin >> number) {
        std::cout << "You entered: " << number << std::endl;
    } else {
        std::cout << "Invalid input!" << std::endl;
    }
    return 0;
}
```

在这个例子中，如果用户输入一个有效的整数，`std::cin >> number` 返回一个 `std::istream&` 类型的对象，该对象在布尔上下文中为 `true`。如果输入无效，例如用户输入了一个非整数，`std::cin` 将返回 `false`。

- ## 对象在布尔上下文中的行为

  当 `std::cin` **被用在布尔上下文中（比如 `while` 循环的条件中），它会自动转换为一个布尔值**，表示输入操作是否成功。如果输入操作成功，`std::cin` 会转换为 `true`；如果输入操作失败，`std::cin` 会转换为 `false`。

  ### 为什么会这样？

  `std::istream` 类（`std::cin` 是它的一个实例）有一个重载的布尔类型转换运算符，这个运算符会检查流的状态。如果流处于有效状态（即没有发生错误），则返回 `true`；如果流处于无效状态（如遇到输入错误或到达文件末尾），则返回 `false`。

  ### 具体例子

  下面是一个具体的代码例子，展示了如何在 `while` 循环中使用 `std::cin`：

  ```cpp
  #include <iostream>

  int main() {
      int number;
      std::cout << "Enter numbers (enter a non-integer to stop): ";
  ```

```
    // 这个循环会持续进行，直到输入操作失败
    while (std::cin >> number) {
        std::cout << "You entered: " << number << std::endl;
    }

    std::cout << "Input failed or end of input." << std::endl;
    return 0;
}
```

解释

1. `std::cin >> number`：尝试从标准输入读取一个整数并存储到 `number` 中。
2. **布尔转换**：`std::cin` 会自动转换为一个布尔值。如果读取操作成功（即用户输入了一个有效的整数），`std::cin` 转换为 `true`，循环继续。如果读取操作失败（如用户输入了一个非整数），`std::cin` 转换为 `false`，循环终止。
3. **循环终止**：当用户输入非整数时，`std::cin` 的状态变为无效，转换为 `false`，`while` 循环终止，程序继续执行循环之后的代码。

总结

当 `std::cin` 被用在 `while` 循环的条件中时，它利用了流对象在布尔上下文中的行为特性。具体来说，`std::cin` 会转换为一个布尔值，表示最近一次输入操作是否成功。如果成功，循环继续；如果失败，循环终止。这种机制非常方便，用于处理连续输入的情况。

注意：**cin在读取到非法输入后，会将字符留在缓冲区内部！**

`std::cin` 尝试读取一个整数失败后，不会自动移除那些导致失败的字符。要清除这些不合法的输入并恢复流的有效状态，我们需要手动处理。

```
#include <iostream>

int main() {
    int number;
    std::cout << "Enter numbers (enter a non-integer to stop): ";

    while (std::cin >> number) {
        std::cout << "You entered: " << number << std::endl;
    }

    std::cout << "Input failed or end of input." << std::endl;
    return 0;
}
```

情况分析

1. 如果用户输入一个非整数（比如字母 `a`），`std::cin >> number` 操作会失败。
2. `std::cin` 的状态会变为无效，`std::cin` 转换为 `false`，导致 `while` 循环终止。
3. 非法字符 `a` 仍然留在输入缓冲区中。

处理方法

为了清除不合法的输入并恢复流的有效状态，我们可以使用 `std::cin.clear()` 和 `std::cin.ignore()`。`std::cin.clear()` 用于清除流的错误状态，而 `std::cin.ignore()` 用于忽略（丢弃）输入缓冲区中的字符。

修改后的代码（该程序可自动跳过非法输入并继续恢复流的有效状态）

```
#include <iostream>

int main() {
    int number;
    std::cout << "Enter numbers (enter a non-integer to stop): ";

    while (true) {
        if (std::cin >> number) {
            std::cout << "You entered: " << number << std::endl;
        } else {
            // 清除错误状态
            std::cin.clear();
            // 忽略输入缓冲区中的不合法输入
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            std::cout << "Invalid input. Please enter an integer." <<
std::endl;
        }
    }

    return 0;
}
```

解释

1. `std::cin.clear()`：清除流的错误状态，使得 `std::cin` 可以继续进行新的输入操作。
2. `std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n')`：忽略输入缓冲区中的所有字符，直到遇到换行符。这确保了不合法的输入被丢弃，不会影响后续的输入操作。

总结

当 `std::cin` 读取到不合法的输入时，不合法的输入会留在输入缓冲区中。为了清除这些不合法的输入并恢复流的有效状态，我们需要使用 `std::cin.clear()` 和 `std::cin.ignore()`。这样可以确保输入流恢复正常，并且不合法的输入不会影响后续的输入操作。

希望这个解释能帮助你理解如何处理 `std::cin` 读取到不合法输入的情况。如果你还有其他问题，欢迎继续提问！

3. **检查流的状态**

你还可以使用流状态标志来检查输入操作的结果：

- `std::cin.fail()`：如果上一次输入操作失败，返回 `true`。

- `std::cin.eof()`：如果到达输入流的末尾，返回 `true`。
- `std::cin.good()`：如果没有发生错误，返回 `true`。
- `std::cin.bad()`：如果发生不可恢复的错误，返回 `true`。

```
#include <iostream>

int main() {
    int number;
    std::cout << "Enter an integer: ";
    std::cin >> number;

    if (std::cin.fail()) {
        std::cout << "Input failed!" << std::endl;
    } else {
        std::cout << "You entered: " << number << std::endl;
    }

    return 0;
}
```

- `std::cin` 的返回值是一个 `std::istream&` 类型的引用。
- 你可以利用这个返回值进行链式输入操作。
- 你可以通过检查 `std::cin` 的返回值或使用流状态标志来判断输入操作是否成功。

③ 读取到回车键（空白字符）后停止

一般的cin是不会读取空白字符的，但不带参数的cin.get()可以读取空白字符

```
vector<int> nums;
    int num;
    while (cin>>num) {
        nums.push_back(num);
        counter++;
        if(cin.get()=='\n'){
            //使用cin读取下一个字符
            break;
        }
    }
    cout<<nums.size();
```

也可以使用string中的getline

```
#include <iostream>
#include <string>

int main() {
    std::string input;

    std::cout << "Enter lines of text (press Enter on an empty line to stop):" <<
std::endl;

    while (true) {
```

```
        std::getline(std::cin, input); // 读取一整行输入

        if (input.empty()) { // 检查输入是否为空行
            break; // 如果是空行，退出循环
        }

        std::cout << "You entered: " << input << std::endl;
    }

    std::cout << "Input stopped." << std::endl;
    return 0;
}
```

总结

`cin` 是一个功能强大的输入流对象，用于从标准输入读取数据。它提供了多种成员函数和操作符来处理不同类型的数据和输入错误。在编写 C++ 程序时，`cin` 是处理用户输入的主要工具之一。

如果你有更多问题或需要更详细的解释，请告诉我！