c++特有输入输出

cin 与 cout

`cin` 与 `cout` 是 C++中的输入与输出函数，需要添加头文件`#include <iostream>`和`using namespace std;`才能使用。`cin` 和 `cout` 不需要像 C 语言中的 `scanf`、`printf` 函数那样指定输入输出的格式，也不需要使用取地址运算符`&`，而可以直接进行输入输出，十分易用和方便。

cin

`cin` 是 `c` 和 `in` 的合成词，采用输入运算符`>>`来进行输入。如果想要输入一个整数 `n`的话，可以按下面的写法进行输入：

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cin >> n;
    return 0;
}
```

我们会发现，`cin` 的输入不指定格式，也不需要加取地址运算符`&`，直接写变量名就可以了。于是同理也可以知道读入 `double` 型浮点数 `db`、`char` 型字符 `c` 的方法也是一样的：

```
cin >> db;
cin >> c;
```

如果同时读入多个变量也是一样的写法，只需要往后面使用`>>`进行扩展即可。例如下面的代码读入了`int` 型变量 `n`、`double` 型变量 `db`、`char` 型变量 `c`、`char` 型数组 `str[]`：

```
cin >> n >> db >> c >> str;
```

而如果想要读入一整行的话，我们需要使用 `getline` 函数，例如下面的代码就把一整行都读取入 `char` 型数组 `str[100]`中：

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

而如果是 `string` 类的话（`string`本质是个容易，需要添加对应的头文件），则需要用下面的方式输入：

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

cout

`cout` 是 `c` 和 `out` 的合成词，使用方法和 `cin` 几乎是一致的，只不过使用的是输出运算符 `<<`。下面的代码输出了 `int` 型变量 `n`、`double` 型变量 `db`、`char` 型变量 `c`、`char` 型数组 `str[]`：

```
cout << n << db << c << str;
```

但是要注意的是，输出的时候中间并没有加空格。因此我们可以在每个变量之间加上空格：

```
cout << n << " " << db << " " << c << " " << str;
```

当然，如果想要在中间输出字符串也是可以的：

```
cout <<n << "aa" << db << "bb" << c << "cc" << str;
```

对 `cout` 来说，换行有两种方式。第一种和 C中相同，也就是使用 `\n` 来进行换行；第二种方法则是使用 `endl` 来表示换行（`endl` 是 `end line` 的缩写）：

```
cout << n << "\n" << db << endl;
```

如果想要控制 `double` 型的精度，例如输出小数点后两位，那么需要在输出之前加上一些东西，并且要加上 `#include <iomanip>` 头文件。下面的代码会输出 `123.46`：

```
cout << setiosflags(ios::fixed) << setprecision(2) << 123.4567 << endl;
```

事实上，对考试而言，我们并不推荐学弟妹使用 `cin` 跟 `cout` 来进行输入和输出，因为它们在输入输出大量数据的情况下表现得非常糟糕，有时候题目的数据还没有输入完毕就已经超时。因此我们还是推荐学弟妹使用 C 语言的 `scanf` 与 `printf` 函数进行输入输出，只有在十分必要的时候（用到 `string` 类的时候）才使用 `cin` 与 `cout`。

复杂度

一般来说，复杂度可以分为时间复杂度和空间复杂度，有时候还会提到编码复杂度，学习它们非常重要,在数据结构部分中有详解。

(1) 时间复杂度

简单地说，时间复杂度是算法需要执行基本运算的次数所处的等级，其中基本运算就是类似加减乘除这种计算机可以直接实现的运算。时间复杂度是评判算法时间效率的有效标准，详例请参见数据结构部分。

我们在写程序时要特别注意分析算法的时间复杂度，因为较高的时间复杂度会让测评系统返回“运行超时”。不过一般来说，我们只需要大致估计算法的时间复杂度在哪个等级即可，例如对时间复杂度为$O(2)$的算法来说，当 `n` 的规模为 1000 的时候，其运算次数大概为10^6级别；而当 `n` 的规模为 100000 的时候，其运算次数就会有10^10级别。对一般的系统来说，$O(n^2)$的算法当 `n` 的规模为 1000 时是可以承受的，而当 `n` 的规模为 100000 时则是不可承受的。

(2) 空间复杂度

和时间复杂度类似，空间复杂度采用相同的写法，表示算法需要消耗的最大数据空间。例如对某个算法来说，其消耗的最大数据空间是一个二维数组，那么这个算法的空间复杂度就是$O(n^2)$。在一般的应用中，一般来说空间都是足够使用的（只要不开好几个10^7以上的数组即可，例如 `int A[10000][10000]` 的定义就是不合适的），因此其重要性一般没有时间复杂度那么大。另外，$O(1)$的空间复杂度是指算法消耗的空间不随数据规模的增大而增大。

考虑到空间一般够用，因此常常采用以空间换时间的策略，例如 哈希法就是一种以空间换时间的高效方法。

(3) 编码复杂度

编码复杂度是一个定性的概念，并没有什么量化的标准。对一个问题来说，如果使用了冗长的算法思想，那么代码量将会非常巨大，其编码复杂度就会非常大。

到这里为止，三种复杂度已经介绍完毕，学弟妹应当在编程中尝试平衡三种复杂度，以使算法能尽可能高效，又简洁优美。