第六章 程序的版式

版式虽然不会影响程序的功能，但会影响可读性。程序的版式追求清晰、美观，是程序风格的重要构成因素。

可以把程序的版式比喻为“书法"。好的“书法”可让人对程序一目了然，看得兴致勃勃。差的程序“书法”如螃蟹爬行，让人看得索然无味，更令维护者烦恼有加。请程序员们学习程序的“书法"，弥补大学计算机教育的漏洞，实在很有必要。

6.1 空行

空行起着分隔程序段落的作用。空行得体（不过多也不过少）将使程序的布局更加清晰。空行不会浪费内存，虽然打印含有空行的程序是会多消耗一些纸张，但是值得。所以不要舍不得用空行。

- **【规则 6-1-1】**在每个类声明之后、每个函数定义结束之后都要加空行。参见示例 6-1（a）
- **【规则 6-1-2】**在一个函数体内，逻辑上密切相关的语句之间不加空行，其它地方应加空行分隔。参见示例 6-1（b ）

```
// 空行
void Function1(…)
{
  …
}
// 空行
void Function2(…)
{
  …
}
// 空行
void Function3(…)
{
  …
}
```

示例 6-1(a) 函数之间的空行

```
// 空行
while (condition)
{
  statement1;
  // 空行
  if (condition)
  {
     statement2;
  }
  else
  {
     statement3;
  }
  // 空行
  statement4;
}
```

示例 6-1(b) 函数内部的空行

6.2 代码行

- **【规则 6-2-1】**一行代码只做一件事情，如只定义一个变量，或只写一条语句。这样的代码容易阅读，并且方便于写注释。
- **【规则 6-2-2】**if、for、while、do 等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加{}。这样可以防止书写失误。

示例 6-2（a）为风格良好的代码行，示例 6-2（b）为风格不良的代码行。

```
int width;   // 宽度
int height;  // 高度
int depth;   // 深度
```

```
int width, height, depth; // 宽度高度深度
```

```
x = a + b;
y = c + d;
z = e + f;
```

```
x = a + b;   y = c + d;  z = e + f;
```

```
if (width < height)
{
  dosomething();
}
```

```
if (width < height) dosomething();
```

```
for (initialization; condition; update)
{
     dosomething();
}
// 空行
other();
```

```
for (initialization; condition; update)
     dosomething();
other();
```

示例 6-2(a) **风格良好**的代码行　　示例 6-2(b) **风格不良**的代码行

✧ **【建议 6-2-1】**尽可能在定义变量的同时初始化该变量（就近原则）

如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。如果引用了未被初始化的变量，可能会导致程序错误。本建议可以减少隐患。例如

```
int width = 10;     // 定义并初始化 width
int height = 10;    // 定义并初始化 height
int depth = 10;     // 定义并初始化 depth
```

6.3 代码行内的空格

- 【规则 6-3-1】关键字之后要留空格。象 const、virtual、inline、case 等关键字之后至少要留一个空格，否则无法辨析关键字。象 if、for、while 等关键字之后应留一个空格再跟左括号‘(’，以突出关键字。
- 【规则 6-3-2】函数名之后不要留空格，紧跟左括号‘(’，以与关键字区别。
- 【规则 6-3-3】‘(’向后紧跟，‘)’、‘,’、‘;’向前紧跟，紧跟处不留空格。
- 【规则 6-3-4】‘,’之后要留空格，如 Function(x, y, z)。如果‘;’不是一行的结束符号，其后要留空格，如 for (initialization; condition; update)。
- 【规则 6-3-5】赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=” “>=”、“<=”、“+”、“*”、“%”、“&&”、“||”、“<<”,“^”等二元操作符的前后应当加空格。
- 【规则 6-3-6】一元操作符如“!”、“~”、“++”、“--”、“&”（地址运算符）等前后不加空格。
- 【规则 6-3-7】象“[ ]”、“.”、“->”这类操作符前后不加空格。

✧ 【建议 6-3-1】对于表达式比较长的 for 语句和 if 语句，为了紧凑起见可以适当地去掉一些空格，如 for (i=0; i<10; i++)和 if ((a<=b) && (c<=d))

void Func1(int x, int y, int z);	// 良好的风格
void Func1 (int x,int y,int z);	// 不良的风格
if (year >= 2000)	// 良好的风格
if(year>=2000)	// 不良的风格
if ((a>=b) && (c<=d))	// 良好的风格
if(a>=b&&c<=d)	// 不良的风格
for (i=0; i<10; i++)	// 良好的风格
for(i=0;i<10;i++)	// 不良的风格
for (i = 0; i < 10; i ++)	// 过多的空格
x = a < b ? a : b;	// 良好的风格
x=a<b?a:b;	// 不好的风格
int *x = &y;	// 良好的风格
int * x = & y;	// 不良的风格
array[5] = 0;	// 不要写成 array [5] = 0;
a.Function();	// 不要写成 a . Function();
b->Function();	// 不要写成 b -> Function();

示例 6-3 代码行内的空格

6.4 对齐

- **【规则 6-4-1】**程序的分界符‘{’和‘}’应独占一行并且位于同一列，同时与引用它们的语句左对齐。
- **【规则 6-4-2】**{ }之内的代码块在‘{’右边数格处左对齐。

示例 6-4（a）为风格良好的对齐，示例 6-4（b）为风格不良的对齐。

void Function(int x)
{
… // program code
}
	void Function(int x){
	… // program code
	}

if (condition)
{
… // program code
}
else
{
… // program code
}
	if (condition){
	… // program code
	}
	else {
	… // program code
	}

for (initialization; condition; update)
{
… // program code
}
	for (initialization; condition; update){
	… // program code
	}

While (condition)
{
… // program code
}
	while (condition){
	… // program code
	}

如果出现嵌套的{ }，则使用缩进对齐，如：
{
…
{
…
}
…
}	

示例 6-4(a) **风格良好**的对齐　　示例 6-4(b) **风格不良**的对齐

6.5 长行拆分

- **【规则 6-5-1】**代码行最大长度宜控制在 70 至 80 个字符以内。代码行不要过长，否则眼睛看不过来，也不便于打印。
- **【规则 6-5-2】**长表达式要在低优先级操作符处拆分成为新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

```
if ((very_longer_variable1 >= very_longer_variable12)
    && (very_longer_variable3 <= very_longer_variable14)
    && (very_longer_variable5 <= very_longer_variable16))
{
    dosomething();
}
```

```
virtual CMatrix CMultiplyMatrix (CMatrix leftMatrix,
                                 CMatrix rightMatrix);
```

```
for (very_longer_initialization;
     very_longer_condition;
     very_longer_update)
{
    dosomething();
}
```

示例 6-5 长行的拆分

6.6 修饰符的位置

修饰符 * 和 & 应该靠近数据类型还是该靠近变量名，是个有争议的话题。

若将修饰符 * 靠近数据类型，例如：int* x; 从语义上讲此写法比较直观，即 x 是 int 类型的指针。

上述写法的弊端是容易引起误解，例如：int* x, y; 此处 y 容易被误解为指针变量。虽然将 x 和 y 分行定义可以避免误解，但并不是人人都愿意这样做。

- **【规则 6-6-1】**应当将修饰符 * 和 & 紧靠变量名。

例如：

```
    char  *name;
int   *x, y;    // 此处 y 不会被误解为指针
```

6.7 注释

C 语言的注释符为“/*…*/”。C++语言中，程序块的注释常采用“/*…*/”，行注释一般采用“//…”。

注释通常用于：

（1）版本、版权声明；

（2）函数接口说明；

（3）重要的代码行或段落提示。

虽然注释有助于理解代码，但注意不可过多地使用注释。参见示例 6-7。

- **【规则 6-7-1】**注释是对代码的“提示”，而不是文档。程序中的注释不可喧宾夺主，注释太多了会让人眼花缭乱。注释的花样要少。
- **【规则 6-7-2】**如果代码本来就是清楚的，则不必加注释。否则多此一举，令人厌烦。例如

```
i++;    // i 加 1，多余的注释
```

- **【规则 6-7-3】**边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除掉。
- **【规则 6-7-4】**注释应当准确、易懂，防止注释有二义性。错误的注释不但无益反而有害。
- **【规则 6-7-5】**尽量避免在注释中使用缩写，特别是不常用缩写。
- **【规则 6-7-6】**注释的位置应与被描述的代码相邻，可以放在代码的上方或右方，不可放在下方。
- **【规则 6-7-8】**当代码比较长，特别是有多重嵌套时，应当在一些段落的结束处加注释，便于阅读。

```
/*
* 函数介绍：
* 输入参数：
* 输出参数：
* 返回值   ：
*/
void Function(float x, float y, float z)
{
  …
}
```

```
if (…)
{
  …
  while (…)
  {
    …
  } // end of while
  …
} // end of if
```

示例 6-7 程序的注释

6.8 类的版式

类的版式主要有两种方式：

（1）将 private 类型的数据写在前面，而将 public 类型的函数写在后面，如示例 6-8（a）。采用这种版式的程序员主张类的设计“以数据为中心"，重点关注类的内部结构。

（2）将 public 类型的函数写在前面，而将 private 类型的数据写在后面，如示例 6-8（b）采用这种版式的程序员主张类的设计“以行为为中心"，重点关注的是类应该提供什么样的接口（或服务）。

估计很多 C++教课书受到 Bjarne Stroustrup 第一本著作的影响，不知不觉地采用了“以数据为中心”的书写方式，并不见得有多少道理。

我建议读者采用“以行为为中心”的书写方式，即首先考虑类应该提供什么样的函数。这是很多人的经验——“这样做不仅让自己在设计类时思路清晰，而且方便别人阅读。因为用户最关心的是接口，谁愿意先看到一堆私有数据成员！”

```
class A
{
  private:
    int    i, j;
    float  x, y;
    …
  public:
    void Func1(void);
    void Func2(void);
    …
}
```

示例 6-8(a) 以数据为中心版式

```
class A
{
  public:
    void Func1(void);
    void Func2(void);
    …
  private:
    int    i, j;
    float  x, y;
    …
}
```

示例 6-8(b) 以行为为中心的版式