# Arithmetic Test for Primary School Students

**小学生用简易四则运算练习软件**

代码规范

目录

[Arithmetic Test for Primary School Students 1](#_Toc474183062)

[一. 排版 2](#_Toc474183063)

[二. 注释 5](#_Toc474183064)

[三. 标识符命名 7](#_Toc474183065)

[四.可读性 9](#_Toc474183066)

[五. 函数，过程 11](#_Toc474183067)

# 一. 排版

1. 1-1：程序块要采用缩进风格编写。对齐只用空格键、不用Tab。缩进的空格数为4个。

说明：对于由开发工具自动生成的代码可以有不一致。使用tab有可能会导致缩进格式编排混乱。

1. 1-2：相对独立的程序块之间、变量说明之后必须加空行。

便于让人了解到这是分开的模块。

1. 1-3：循环、判断、函数调用、或过程中语句中若有较长的表达式或语句，则要进行适应的划分，长表达式要在低优先级操作符处划分新行，操作符放在新行之首。

示例：

if ((taskno < max\_act\_task\_number)

&& (n7stat\_stat\_item\_valid (stat\_item)))

{

... // program code

}

1. 1-4：不允许把多个短语句写在一行中，即一行只写一条语句。

示例：如下例子不符合规范。

rect.length = 0; rect.width = 0;

应如下书写

rect.length = 0;

rect.width = 0;

1. 1-5：if、for、do、while、case、switch、default等语句自占一行，且if、for、do、while等语句的执行语句部分无论多少都要加括号{}。 分界符应独占一行，并且属于同一列。

示例：

for (...)

{

... // program code

}

if (...)

{

... // program code

}

void example\_fun( void )

{

... // program code

}

1. 1-6：使用操作符对多个关键字、变量、常量进行操作时，要加入空格使得代码更加清晰。具体规则如下：

**(1) 逗号、分号只在后面加空格。**

int a, b, c;

**(2) 双目操作符**，包括但不限于赋值、算术、逻辑等运算符，例如"="、"+="、"+"、"&&"，**前后加空格**。

if (current\_time >= MAX\_TIME\_VALUE)

a = b + c;

a \*= 2;

**(3) 单目操作符**，例如"!"（非）、"~"（取反）、"++"（自增）、"--"（自减）、"&"（取地址）、"->"、"."（成员操作）**前后不加空格**。

1. 1-7：一行程序以小于80字符为宜，不要写得过长。

# 二. 注释

**2.1：源程序必须具有一定的注释，便于对程序的理解（注释需要简介明了准确）**

2.2：注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

示例：如下例子不符合规范。

例1：

/\* get replicate sub system index and net indicator \*/

repssn\_ind = ssn\_data[index].repssn\_index;

repssn\_ni = ssn\_data[index].ni;

例2：

repssn\_ind = ssn\_data[index].repssn\_index;

repssn\_ni = ssn\_data[index].ni;

/\* get replicate sub system index and net indicator \*/

应如下书写

/\* get replicate sub system index and net indicator \*/

repssn\_ind = ssn\_data[index].repssn\_index;

repssn\_ni = ssn\_data[index].ni;

2.3：注释与所描述内容进行同样的缩排，并要与其上面的代码用空行隔开。

说明：可使程序排版整齐，并方便注释的阅读与理解。

示例：如下例子，排版不整齐，阅读稍感不方便。

void example\_fun( void )

{

/\* code one comments \*/

CodeBlock One

/\* code two comments \*/

CodeBlock Two

}

应改为如下布局。

void example\_fun( void )

{

/\* code one comments \*/

CodeBlock One

/\* code two comments \*/

CodeBlock Two

}

**2.4：尽量不要在一行代码或表达式中插入注释，对函数，过程，变量的一些注释内容尽量正确合理，使之明确易懂，能够提供明确的信息或是代码以外有用的额外信息。**

**2.5：注释格式尽量统一，建议用/\*......\*/，语言尽量统一，统一使用中文或者英文，为了更准确的表达意思及后续方便阅读，建议统一使用中文。**

# 三. 标识符命名

**3.1：标识符命名尽量要清晰，明了，尽量使用英文全名或是易懂的缩写，和一些特殊约定或缩写。有需要时，要加以注释便于理解。**

3.2：对于变量、函数、数组等命名，采用以下规则：

变量：采取小驼峰法，除第一个单词之外，其他单词首字母大写。

例如 int myStudentNum;

函数名、类名：采取大驼峰法，所有单词首字母大写。

例如 class DatabaseUser;

数组名：采取小驼峰法，并且用下划线分隔单词。

例如 char rand\_Operator[4];

3.3：命名规范的风格要保持一致，并在同一项目中统一。

3.4：除非必要，不要用数字或较奇怪的字符来定义标识符。

3.5:用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。

说明：下面是一些在软件中常用的反义词组。

add / remove begin / end create / destroy

insert / delete first / last get / release

add / delete lock / unlock open / close

min / max old / new start / stop

# 四.可读性

**4.1：注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级。**

说明：防止阅读程序时产生误解，防止因默认的优先级与设计思想不符而导致程序出错。

4.2：避免使用不易理解的数字，用有意义的标识来替代。涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的枚举或宏来代替。

示例：如下的程序可读性差。

if (Trunk[index].trunk\_state == 0)

{

Trunk[index].trunk\_state = 1;

... // program code

}

应改为如下形式。

#define TRUNK\_IDLE 0

#define TRUNK\_BUSY 1

if (Trunk[index].trunk\_state == TRUNK\_IDLE)

{

Trunk[index].trunk\_state = TRUNK\_BUSY;

... // program code

}

4.3：源程序中关系较为紧密的代码应尽可能相邻。并且尽量不使用难懂的技巧性高的语句，除非很有必要时。

示例：以下代码布局不太合理。

rect.length = 10;

char\_poi = str;

rect.width = 5;

若按如下形式书写，可能更清晰一些。

rect.length = 10;

rect.width = 5; // 矩形的长与宽关系较密切，放在一起。

char\_poi = str;

# 五. 函数，过程

5.1：明确函数功能，精确（而不是近似）地实现函数设计。

*5*.2：防止将函数的参数作为工作变量。

说明：将函数的参数作为工作变量，有可能错误地改变参数内容，所以很危险。对必须改变的参数，最好先用局部变量代之，最后再将该局部变量的内容赋给该参数。

示例：下函数的实现不太好。

void sum\_data( unsigned int num, int \*data, int \*sum )

{

unsigned int count;

\*sum = 0;

for (count = 0; count < num; count++)

{

\*sum += data[count]; // sum成了工作变量，不太好。

}

}

若改为如下，则更好些。

void sum\_data( unsigned int num, int \*data, int \*sum )

{

unsigned int count ;

int sum\_temp;

sum\_temp = 0;

for (count = 0; count < num; count ++)

{

sum\_temp += data[count];

}

\*sum = sum\_temp;

}

**5.3：函数规模尽量限制在200行以内，并且为简单功能也编写函数，同时函数不要设计用途多方面的函数。**

**5.4：检查所有函数参数以及非函数参数的输入的有效性；**

**5.5：函数名应准确描述好函数的功能，函数的返回值应清楚明了，并且函数在调用点应该要明确易懂。**

**5.6：改进模块中函数的结构，降低函数间的耦合度，并提高函数的独立性以及代码可读性、效率和可维护性。优化函数结构时，要遵守以下原则：**

（1）不能影响模块功能的实现。

（2）仔细考查模块或函数出错处理及模块的性能要求并进行完善。

（3）通过分解或合并函数来改进软件结构。

（4）考查函数的规模，过大的要进行分解。

（5）降低函数间接口的复杂度。

（6）不同层次的函数调用要有较合理的扇入、扇出。

（7）函数功能应可预测。

（8）提高函数内聚。（单一功能的函数内聚最高）

说明：对初步划分后的函数结构应进行改进、优化，使之更为合理。4

**5.7：对于提供了返回值的函数，在引用时最好使用其返回值。**