中国科学技术大学
University of Science and Technology of China

程序设计II

第2讲 编码规范

计算机学院 黄章进

zhuang@ustc.edu.cn

内容

中国科学技术大学
University of Science and Technology of China

- 编码风格

编码风格

- 代码总体原则
  - 清晰第一
    - ✓清晰性是易于维护、易于重构的程序必需具备的特征。代码首先是给人读的，好的代码应当可以像文章一样发声朗诵出来。
  - 简洁为美
    - ✓简洁就是易于理解并且易于实现。代码越长越难以看懂，也就越容易在修改时引入错误
  - 选择合适的风格，与代码原有风格保持一致
    - ✓如果重构/修改其他风格的代码时，比较明智的做法是根据现有代码的现有风格继续编写代码

标识符命名

编码风格

- 常用命名风格：
  - unix like风格：单词用小写字母，每个单词直接用下划线'_'分割，例如text_mutex，kernel_text_address。
  - Windows风格：大小写字母混用，单词连在一起，每个单词首字母大写。不过Windows风格如果遇到大写专有用语时会有些别扭，例如命名一个读取RFC文本的函数，命令为ReadRFCText，看起来就没有unixlike的read_rfc_text清晰了。
  - 匈牙利命名法：计算机程序设计中的一种命名规则，用这种方法命名的变量显示了其数据类型。匈牙利命名主要包括三个部分：基本类型、一个或更多的前缀、一个限定词。
    - ✓ 这种命名法最初在20世纪80年代的微软公司广泛使用，并在win32API和MFC库中广泛的使用，但匈牙利命名法存在较多的争议

匈牙利命名法

编码风格

- 以两条规则为基础的：
  - a. 变量的名字以一个或者多个小写字母前缀开头，前缀能够体现变量数据类型、作用域等信息。
  - b. 在标识符内，前缀以后就是一个或者多个第一个字母大写的单词，这些单词清楚地指出了该标识符的作用

一种命名法

编码风格

1）变量命名加前缀

c	char
uc	unsigned char
s	short
n	int
u	unsigned int
l	long
dw	unsigned long
b	bool
sz	以'\0'结尾的字符串或字符数组
f	float
h	HANDLE（句柄）
d	double

一种命名法

编码风格

p	指针
psz	指向字符串的指针
pn	整型指针
m_	类成员变量
g_	全局变量
s_	静态变量
a	数组
fp	文件指针 FILE *
e	enum类型

一种命名法

编码风格

2）变量名中单词开头字母大写，其他字母小写，例如：nNumOfPoints

- 但是常用的意义明显的变量，如 i,j,k, 坐标 x,y等不必遵循 1),2)

3）常量和宏都是大写，单词之间用下划线'_'分隔

```
#define     MAX_WIDTH    5
#define     ABS(x)      ((x)>=0?(x):-(x))
```

一种命名法

4）函数名字中每个单词的头一个字母大写，其他字母小写。一般采用动词+名词形式

```
void PrintMessage();
int WriteIdToFile( FILE * _fp, int _nId);
```

5）结构定义加大写字母S作为前缀

```
struct SPerson  {
        int nId;
        int nAge;
};
```

一种命名法

编码风格

6) 类定义加大写字母C作为前缀

```
class CPerson  {
        int m_nId;
};
```

7) 类型定义全部大写

```
typedef struct SPerson  PERSON;
typedef struct SPerson *  PPERSON; //指针加'P'
```

标识符命名的一些细节

编码风格

- 标识符号应能提供足够信息，最好是可以发音的。
  - 尽可能给出描述性名称，不要节约空间，让别人很快理解你的代码更重要。
  - 好的命名：

```
int error_number;
int number_of_completed_connection;
```

  - 不好的命名：使用模糊的缩写或随意的字符

```
int n;
int nerr;
int n_comp_conns;
```

标识符命名的一些细节

编码风格

- 为全局变量取长的，描述信息多的名字，为局部变量取短名字
- 名字太长时可以适当采用单词的缩写。但要注意，缩写方式要一致。要缩写就全都缩写。
  - 比如单词Number, 如果在某个变量里缩写成了Num:

```
int nDoorNum;
```

  - 那么最好包含 Number单词的变量都缩写成 Num。

标识符命名的一些细节

编码风格

- 除了常见的通用缩写以外，不使用单词缩写，不得使用汉语拼音。
  - 较短的单词可通过去掉“元音”形成缩写，较长的单词可取单词的头几个字母形成缩写，一些单词有大家公认的缩写，常用单词的缩写必须统一。

argument 可缩写为 arg
buffer 可缩写为 buff
clock 可缩写为 clk
command 可缩写为 cmd
compare 可缩写为 cmp
configuration 可缩写为 cfg
device 可缩写为 dev
error 可缩写为 err
hexadecimal 可缩写为 hex
increment 可缩写为 inc
initialize 可缩写为 init
maximum 可缩写为 max
message 可缩写为 msg
minimum 可缩写为 min
parameter 可缩写为 para
previous 可缩写为 prev
register 可缩写为 reg
semaphore 可缩写为 sem
statistic 可缩写为 stat
synchronize 可缩写为 sync
temp 可缩写为 tmp
pointer可缩写为 pntr
reset 可缩写为rst
address 可缩写为 addr
signal 可缩写为 sig

标识符命名的一些细节

编码风格

- 注意使用单词的复数形式。如

```
int nTotalStudents, nStudents ;
```

  - 容易让人理解成代表学生数目，而 nStudent 含义就不十分明显
- 对于返回值为真或假的函数，加Is前缀：

```
int     IsCanceled();
int     isalpha(); // C语言标准库函数
bool    IsButtonPushed();
```

- 一般变量和结构名用名词，函数名用动词或动宾词组

标识符命名的一些细节

编码风格

- 对于获取某个数值的函数，加Get前缀

```
char * GetFileName();
```

- 对于设置某个数值的函数，加Set前缀

```
void SetMaxVolume();
```

标识符命名的一些细节

编码风格

- 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。
- 示例：

add/remove begin/end create/destroy
insert/delete first/last get/release
increment/decrement put/get add/delete
lock/unlock open/close min/max
old/new start/stop next/previous
source/target show/hide send/receive
source/destination copy/paste up/down

编码风格

- 程序块采用缩进风格编写，每级缩进为4个空格。
  - 当前各种编辑器/IDE都支持TAB键自动转空格输入，需要打开相关功能并设置相关功能。
  - 编辑器/IDE如果有显示TAB的功能也应该打开，方便及时纠正输入错误。
  - 宏定义、编译开关、条件预处理语句可以顶格

编码风格

- 相对独立的程序块之间、变量说明之后必须加空行
  - 示例：如下例子不符合规范。

```
if (!valid_ni(ni))
{
   // program code
   ...
}
repssn_ind = ssn_data[index].repssn_index;
repssn_ni = ssn_data[index].ni;
```

  - 应如下书写

```
if (!valid_ni(ni))
{
   // program code
   ...
}

repssn_ind = ssn_data[index].repssn_index;
repssn_ni = ssn_data[index].ni;
```

排版与格式

编码风格

- 一条语句不能过长，如不能拆分需要分行写。
  - 换行时有如下建议：
    - ✓换行时要增加一级缩进，使代码可读性更好
    - ✓低优先级操作符处划分新行；换行时操作符应该也放下来，放在新行首；
    - ✓换行时建议一个完整的语句放在一行，不要根据字符数断行

```
if( Condition1() && Condition2()
    && Condition3() ) {
}
```

排版与格式

编码风格

- 注意 '{'和'}'位置不可随意，要统一
  - 如果写了：

```
if ( condition1() ) {
    DoSomething();
}
```

  - 别处就不要写

```
if( condition2() )
{
    DoSomething() ;
}
```

编码风格

- if、for、do、while、case、switch、default等语句独占一行。
  - 执行语句必须用缩进风格写，属于if、for、do、while、case、switch、default等下一个缩进级别；
  - 一般写if、for、do、while等语句都会有成对出现的'{}'，对此有如下建议可以参考
    - ✓如果if/else配套语句中有一个分支有'{}'，那么另一个分支即使一行代码也建议增加'{}';
    - ✓添加'{'的位置可以在if等语句后，也可以独立占下一行；独立占下一行时，可以和if在一个缩进级别，也可以在下一个缩进级别；但是如果if语句很长，或者已经有换行，建议'{'使用独占一行的写法

排版与格式

编码风格

- 多个短语句（包括赋值语句）不要写在同一行内，即一行只写一条语句
  - 示例：

```
int a = 5; int b = 10;  //不好的排版
```

  - 较好的排版

```
int a = 5;
int b = 10;
```

排版与格式

编码风格

- 在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如一>），后不应加空格。
  - (1) 逗号、分号只在后面加空格

```
int a, b, c;
```

  - (2) 比较操作符, 赋值操作符"="、"+="，算术操作符"+"、"%"，逻辑操作符"&&"、"&"，位域操作符"<<"、"^"等双目操作符的前后加空格。

```
if (current_time >= MAX_TIME_VALUE)
a = b + c;
a *= 2;
a = b ^ 2;
```

排版与格式

编码风格

- (3) "!"、"~"、"++"、"--"、"&"（地址操作符）等单目操作符前后不加空格。

```
*p = 'a';       // 内容操作"*"与内容之间
flag = !is_empty; // 非操作"!"与内容之间
p = &mem;       // 地址操作"&" 与内容之间
i++;            // "++","--"与内容之间
```

- (4) "->"、"."前后不加空格。

```
p->id = pid;    // "->"指针前后不加空格
```

- (5) if、for、while、switch等与后面的括号间应加空格，使if等关键字更为突出、明显

```
if (a >= b && c > d)
```

一些好的编程习惯

编码风格

- 尽量不要用立即数，而用const 定义成常量，以便以后修改

```
const  int MAX_STUDENTS = 20;
struct SStudent aStudents [MAX_STUDENTS];
```

比

```
struct SStudent aStudents [20];
```

好

  - 也不要定义成宏

```
#define TOTAL_ELEMENTS  100
for(i = 0; i < TOTAL_ELEMENTS; i++)   {
}
```

一些好的编程习惯

编码风格

- 使用sizeof，不直接使用变量所占字节数的数值

```
int nAge;
for ( j = 0; j < 100; j++ )
    fwrite( fpFile, &nAge, 1, sizeof(int) );
```

比

```
for ( j = 0; j < 100; j++ )
    fwrite( fpFile, &nAge, 1, 4);
```

好

一些好的编程习惯

编码风格

- 稍复杂的表达式中要积极使用括号，以免优先级理解上的混乱

```
n = k +++ j;   //不好
n = ( k ++ ) + j;  //好一点
```

- 不很容易理解的表达式应分几行写：

```
n = ( k ++ ) + j;
```

应该写成：

```
n = k + j;
k ++;
```

一些好的编程习惯

编码风格

- 不提倡在表达式中使用 ? : 形式，而用 if .. else语句替代

```
xp =  2 * k < ( n-m) ? c[k+1] : d[k--];


if( 2 * k < (n-m) )
        xp = c[k+1];
else
        xp = d[k--];
```

一些好的编程习惯

编码风格

- 嵌套的if else 语句要多使用 { }

```
if ( Condition1() )
    if ( Condition2() )
        DoSomething();
    else
        NoCondition2();
```

不够好，应该：

```
if ( Condition1() ) {
    if ( Condition2() )
        DoSomething();
    else
        NoCondition2();
}
```

一些好的编程习惯

编码风格

- 应避免 if else 的多重嵌套，而用并列的完成

```
if ( Condition1() ) {
   if ( Condition2() ) {
      if ( Condition3() ) {
         Condition123();
      }else  {
         NoCondition3();
      }
   }else {
      NoCondition2();
   }
}else {
   NoCondiction1();
}
```

替换为：

```
if ( ! Condition1() ) {
   NoCondition1();
}else if ( ! Condition2() ) {
   NoCondition2();
}else if ( ! Condition3() ) {
   NoCondition3();
}else {
   Condition123();
}
```

一些好的编程习惯

编码风格

◆写出来的代码应该容易读出声

比如

```
if( !( n > m ) && !( s > t))
```

就不如

```
if( ( m <= n ) && ( t <= s ))
```

```
if( !( c == 'y' || c == 'z'))
```

不如

```
if( c!= 'y' && c!= 'z');
```