第3章　C++程序设计基础

3.1　C++程序设计语言概述

20世纪80年代，美国贝尔实验室的Bjarne Stroustrup博士及其同事在C语言的基础上引入了面向对象的编程思想和一个class关键字，形成了最早的C++语言原型。后来，C++语言被融入越来越多的语言特性，其中其意义和影响最深远的莫过于template（模板）的引入，而后美国国家标准化协会（American National Standard Institute，ANSI）和国际标准化组织（International Standards Organization，ISO）一起对C++语言进行了标准化工作，并于1998年正式发布了C++语言的国际标准ISO/IEC 14882：1998。

本书根据信息学奥赛的需要，只介绍C++的语法，不涉及面向对象的内容，所有题目均不涉及面向对象编程方法，因此语言上与C语言相比基本相同，只是略有扩充，使得语言运用起来更具有便捷性。

3.1.1　程序设计语言的发展历程

计算机的程序设计语言有很多种分类，自20世纪60年代以来，世界上公布的程序设计语言已有上千种之多，但是只有很小一部分得到了广泛的应用。从发展历程来看，程序设计语言可以分为三代。

1）第一代语言（机器语言）

机器语言是由二进制0、1代码指令构成，不同的CPU具有不同的指令系统。机器语言程序难编写、难修改、难维护，需要用户直接对存储空间进行分配，编程效率极低。目前，这种语言的编程方式已经被淘汰。

2）第二代语言（汇编语言）

汇编语言指令是机器指令的符号化，与机器指令存在着直接的对应关系，所以汇编语言同样存在着难学难用、容易出错、维护困难等缺点。但是汇编语言也有自己的优点：可直接访问系统硬件接口，汇编语言程序翻译成的机器语言程序的效率高。

从软件工程角度来看，只有在高级语言不能满足设计要求，或不具备支持某种特定功能的技术性能（如特殊的输入/输出）时，汇编语言才被使用。

3）第三代语言（高级语言）

高级语言是面向用户的、基本上独立于计算机种类和结构的语言。其最大的优点是：形式上接近于算术语言和自然语言，概念上接近于人们通常使用的概念。高级语言的一个命令可以代替几条、几十条甚至几百条汇编语言的指令。因此，高级语言易学易用、通用性强、应用广泛。

C++是C语言的继承，它既可以进行C语言的过程化程序设计，又可以进行以抽象数据类型为特点的基于对象的程序设计，还可以进行以继承和多态为特点的面向对象的程序设计。C++擅长面向对象程序设计的同时，还可以进行基于过程的程序设计，因而C++就问题规模而论，适应性和实用性更强。C++不仅拥有计算机高效运行的实用性特征，同时还致力于提高大规模程序的编程质量与程序设计语言的问题描述能力。

3.1.2　C++语言程序的组成结构

下面从一个最简单的程序入手介绍C++语言程序的组成结构。

【例 3-1】 已知圆半径，求圆周长及面积。

【程序示例】

```
#include <iostream>                          //使用cin、cout,需调用iostream库,否则编译出错
using namespace std;                         //C语言中要省略该语句
int main()                                   //有的C语言可用void main(),如TC++、VC++
{
    int r;                                   //声明一个整型变量r, 此处r是圆半径
    double c,s;                              //声明圆周长c和圆面积s为双精度数
    cin>>r;                                  //输入半径r
    c=2*3.14*r;                              //计算周长
    s=3.14*r*r;                              //计算面积
    cout<<"perimeter="<<c<<endl;             //输出周长并换行
    cout<<"area="<<s<<endl;                  //输出面积并换行
    return 0;                                //结束程序
}
```

运行结果如下：

```
3
perimeter=18.84
area=28.26
```

由如上程序可知，C++语言程序一般至少由三部分组成，分别为头文件、名字空间、主函数。

1）头文件

头文件是C++语言程序对其他程序的引用。头文件作为一种包含功能函数、数据接口声明的载体文件，用于保存程序的声明。格式如下：

```
#include <引用文件名>
```

或

```
#include "引用文件名"
```

编译预处理命令通常放在源程序或源文件的最前面，在调用库函数前，相应的头文件需包含在程序中，C++语言中常用的头文件有：

```
#include <iostream>          //标准C++输入/输出cin、cout库
#include <cstdio>            //C++中调用C标准输入/输出printf和scanf
#include <bits/stdc++.h>     //万能头文件,包括了C++中定义的几乎所有功能程序库的声明
```

2) 名字空间

```
using namespace std;
```

C++语言中采用的全部是 std(标准)名字空间,可解决大型程序或多人同时编写程序时名字产生冲突的问题。

3) 主函数

int main(),定义主函数。程序由一个或多个函数组成,一个程序中必须有且只有一个主函数。不论 main()函数在程序中什么位置,程序都从 main()函数开始执行,main()函数执行完毕,程序也就结束了。此外,某些 C++语言还支持 void main()。

在 C++语言中,";"是语句的分隔符,代码分行写只是为了方便阅读。头文件中的预处理语句需独立成行,且行末无分号,即"# include < bits/stdc++.h >",语句后无分号。

注释(comments)是源代码的一部分,是用来对程序、语句、变量等进行简单解释或描述,但它们会被编译器忽略,不会生成任何执行代码。C++语言支持如下两种注释的方法。

(1) 行注释:"//"开始至本行结束的任何内容。

(2) 块注释(段注释):在"/ * "和" * /"符号之间的所有内容,可以包含多行内容。

3.1.3 C++语言的编译环境

Dev C++是一个可视化集成开发环境(IDE),可以用此软件实现 C++语言程序的编辑、预处理、编译/链接、运行和调试。在信息学奥赛中,DEV C++也是比赛的指定编译软件之一。

1) 启动 DEV C++

单击 Windows 任务栏中的"开始"→"程序"→"Bloodshed Dev-C++"→"Dev-C++"命令,如图 3-1 所示。

图 3-1 打开 Dev C++

或双击桌面中 Dev C++的图标,也能打开 Dev C++。

2) 新建源程序

从主菜单选择"文件"→"新建"→"源代码"命令即可,如图 3-2 所示。

如果打开的软件是全英文版本,则可以选择 Tools→Environment Options 命令,在弹出的对话框中默认打开的 General 选项页中,选择 Language 为"简体中文/Chinese",如图 3-3 所示。

此后,所有菜单均设置为简体中文,在右侧留有大片的空白区域,称为"源程序编辑区域",可以在此处输入程序,需注意的是,输入程序前,请关闭中文输入法,在源程序中除需输出中文外,其他一切符号均应在英文模式下输入,否则编译器将报错,如图 3-4 所示。

3) 保存源程序到硬盘

一个好的习惯是创建并编辑了源程序后,经常性

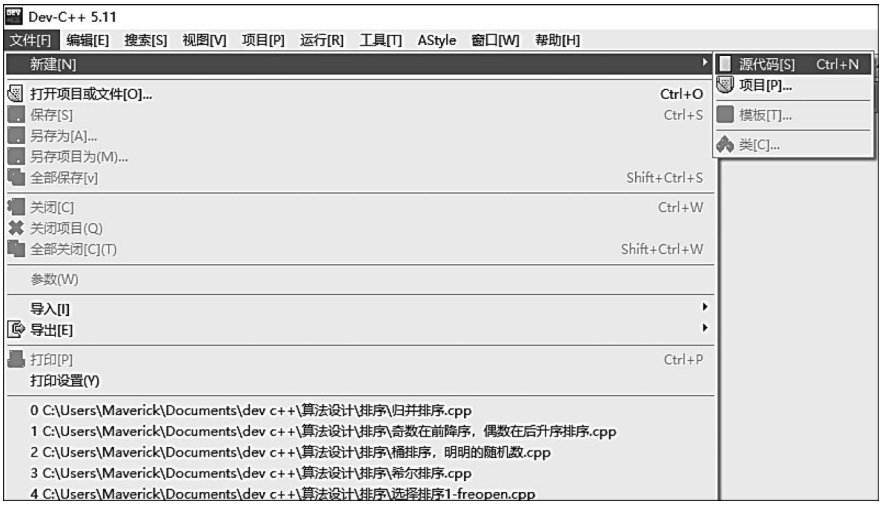

图 3-2　新建源程序

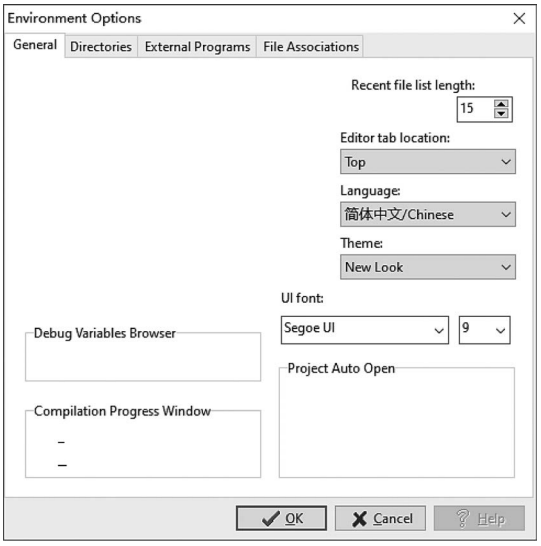

图 3-3　语言设置

地保存源程序，以防止机器突然死机或者程序突然未响应等突发情况发生。要保存程序，只需要选择菜单中的“文件”→“保存”命令就可以将文件保存到指定的硬盘目录，如图 3-5 所示。

此时会弹出一个对话框，在此根据个人需要，将源程序存放到硬盘相应的文件夹，如“D:\ noip \ code”，那么在 D 盘的文件夹 noip 中，有 code 文件夹，在里面存放了 *.cpp 的源程序，其中 * 代表被存放程序的程序名，可以是英文名也可以是中文名，但程序取名应尽量与程序内容相关，以便之后随时调用查看。

4）编译、运行

编译：从主菜单选择“运行”→“编译运行”命令或者按快捷键 F11，如图 3-6 所示。如果程序存在词法、语法错误，则编译过程失败，并且报错。编译器会在屏幕的下方显示错误信息，如果图 3-7 所示，错误信息一般提示错误的行号，以及相应的错误内容。

编译器标签页显示的错误信息是寻找程序错误原因的重要信息来源，切记要学会查看

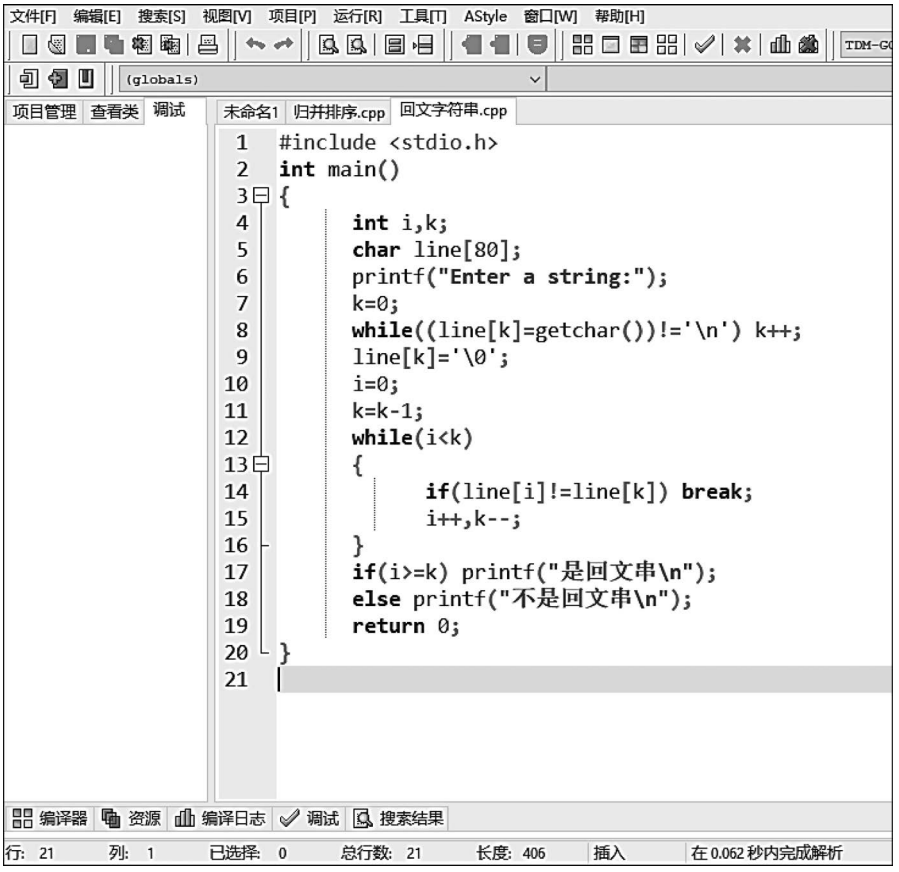

图 3-4　源程序编辑区域

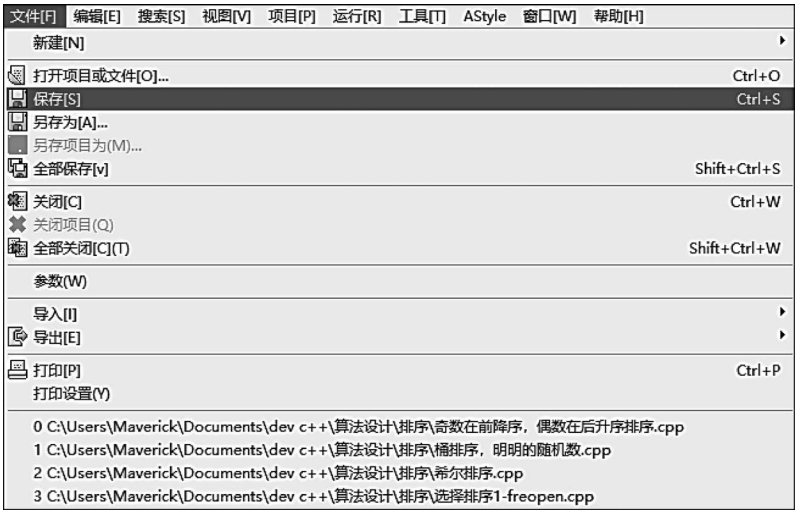

图 3-5　源程序的保存

图 3-6　编译运行

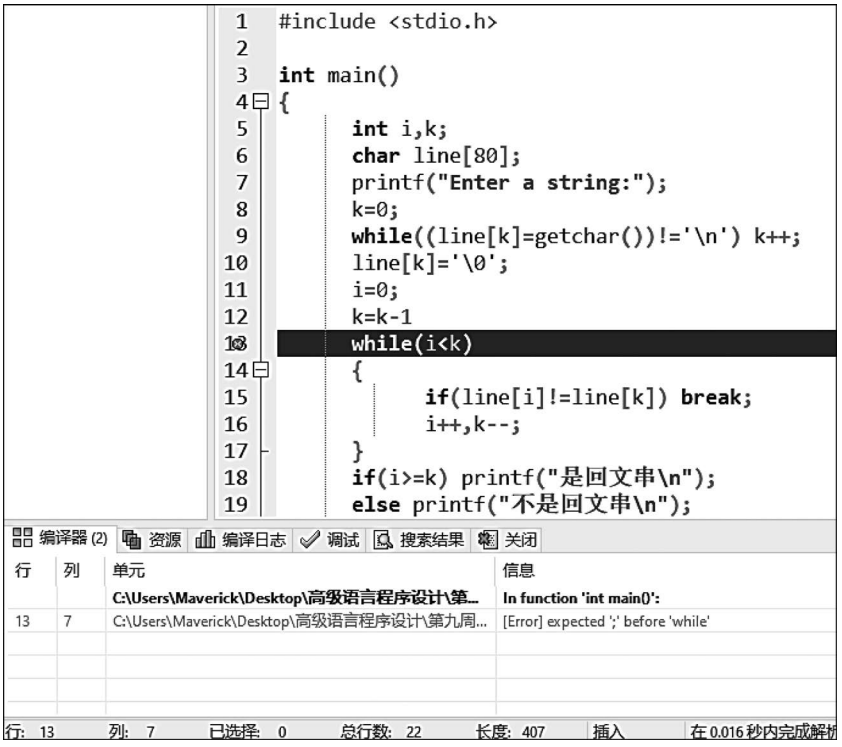

图 3-7　编译出错信息

错误信息，提高程序调试的效率。

5）调试程序

当程序通过了语法检查、编译生成执行文件后，就可以在编程环境或操作系统中运行该程序。语法检查无误的程序并不能保证程序的正确性，一旦程序中存在语义错误（逻辑错误），程序虽然能运行，但是得不出想要的正确结果。例如，“if(i%2==0) sum=sum+i;”中的两个等号= =（等于）改写成 !=（不等于），这样该程序虽然也能通过语法检查（即编译无错误），但语义却正好相反，运行结果从如果 i 是偶数加入 sum 变成了奇数 i 加入 sum。

如果程序有语义错误就需要对程序进行调试。调试是在程序中查找错误并修改错误的过程。调试最主要的工作是找出程序逻辑错误发生的地方。一般程序的编程环境都提供相应的调试手段,其中最主要的方法是:设置断点并观察变量值。

设置断点(Break Point Setting):可以在程序的任何一个语句前的行号上单击,生成一个红色的断点标记,进入调试模式后,程序运行到这里时会暂停下来。

观察变量(Variable Watching):当程序运行到断点的地方停下来时,可以观察各种变量的值,并判断此时的变量值是不是所希望的。如果不是,说明该断点之前肯定有错误发生。这样,就可以把查错的范围集中在断点之前的程序段上。

另外,还有一种常用的调试方法是单步跟踪(Trace Step by Step),即一步一步跟踪程序的执行过程,同时观察变量的变化情况。

调试是一个需要耐心和经验的工作,也是在程序设计中应掌握的最基本的技能。

3.1.4 算法和算法描述

1. 程序设计的基本方法

学习计算机语言的目的是利用该语言工具设计出可供计算机运行的程序。一个程序应包括以下两方面的内容:

(1) 对数据的描述:数据结构(Data Structure)。

(2) 对操作的描述:算法(Algorithm)。

通过程序设计以解决实际应用问题的一般步骤如图 3-8 所示。

图 3-8 程序设计的一般步骤

【例 3-2】 已知半径求球表面积及球体体积问题。

【问题分析】

根据半径求球表面积及体积。

【确定数学模型与数据结构】

(1) 数学模型:使用球表面积公式 $S=4\pi r^2$,$V=\frac{4}{3}\pi r^3$。

(2) 数据结构:设计三个变量空间,分别为半径 r、表面积 S、体积 V。

【设计算法】

算法是指解决一个问题所采取的具体步骤和方法。

【算法设计】

(1) 输入半径 r。

(2) 依据球表面积公式求 S,依据球体积公式求 V。

(3) 输出表面积 S 和体积 V。

【编写程序】

编写程序是用计算机语言描述算法的过程,这一步常称为“编码”,程序的质量主要由算

法决定。

【程序编译调试和运行】

通过编译调试和运行程序，获得正确的编码和正确的结果。

2. 算法

程序设计中最关键的一步就是设计算法，程序设计能力水平的高低在于能否设计出优秀的算法。

算法是解决问题方法的精确描述，解决一个问题的过程就是实现一个算法的过程。

算法应具有以下特点。

(1) 输入：一个算法可以有零个或多个输入量。

(2) 输出：一个算法必须有一个或多个输出量，输出量是算法计算的结果。

(3) 确定性：算法的描述必须无歧义，以保证算法的执行结果是确定的。

(4) 有穷性：算法必须在有限步骤内实现。

(5) 有效性：又称可行性，算法中的每个步骤都应该能有效地执行，执行算法最后应该能得到确定的结果。

对于确定的算法，可以把一个算法看作一个“黑箱子”，根据输入得到确定的输出即黑盒测试。信息学奥赛测评选手算法程序的正确性采用的就是黑盒测试法，测试的核心自然是设计算法解决实际问题的能力。

对于同一个问题，可以有不同的解决方法。

【例 3-3】 求 $S=1-2+3-4+\cdots+99-100$。

方法 1：按顺序直接计算，做 99 次加或减的计算，得出最后的结果。

方法 2：将表达式分成两部分，分别求正数的和 $S1$、负数的和 $S2$，再对两个和做减法运算 $S1-S2$。

$$S=(1+3+5+\cdots+99)-(2+4+8+\cdots+100)$$

方法 3：将相邻的每一对数化成一组，整个表达式就变成了 50 个减法计算的和，而 50 个计算的结果都是 -1，最后的结果就是 $50\times(-1)$。

$$S=(1-2)+(3-4)+\cdots+(99-100)=50\times(-1)$$

……

为了有效解决问题，不仅需要保证算法正确，还要考虑算法的质量。选择合适的算法，不仅算法简单，而且运算步骤少。

3. 算法的描述

为了描述一个算法，可以采用不同的方法，常用的有自然语言、流程图、N-S 流程图、伪代码等。下面以一个实际问题为例，研究不同的方法在描述具体问题的算法时的情况。

【例 3-4】 某主持人大赛中有 7 名评委。每名参赛者得分的计算方式为，输入 7 名评委的打分，去掉一个最高分和一个最低分后求出平均分。

1) 自然语言

自然语言就是人们日常使用的语言，可以用汉语、英语或其他语言表示描述算法。

用自然语言描述算法通俗易懂，但文字冗长，容易出现歧义性，不方便描述包含分支和循环的算法。因此，通常使用自然语言描述算法思路或大框架。

2) 流程图

美国国家标准化协会(ANSI)规定了一些常用的流程图符号。并用流程图符号绘制程序设计的三种基本结构(顺序、选择、循环)。1973 年,美国学者 Nassi 和 Shneiderman 提出了一种新的流程图形式。在这种流程图中完全去掉了带箭头的流程线。全部算法写在一个矩形框内,在该框内还可以包含其他从属于它的框,或者说由一些基本的框组成一个大的框,这种流程图又称 N-S 流程图。简单问题采用流程图描述算法尚可,但复杂问题的算法描述有诸多的不便。

3) 伪代码

伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法的,它不用图形,因此比较紧凑,也比较好懂。

伪代码使用中的一些基本符号如下。

(1) 运算符号。

简单算术运算符号:+、-、×、/、mod(整除取余)。

关系运算符号:>、>=、<、<=、=、<>(不等于)。

逻辑运算符号:and(而且)、or(或者)、not(相反)。

括号:( )

(2) 处理语句。

赋值:←,如 i←1,即把 i 设为 1。

如果 p 成立则 A,否则 B:if p then A else B。

当 p 成立时,执行 A:while p do A 或 do A while p。

执行 A,直到 p 成立:do until p

A

enddo

输入和输出(打印):input、print。

基本块起、止符号:{、}。

算法开始和结束:BEGIN、END。

用伪代码表示例 3-4 的算法如下:

```
BEGIN
    s← 0
    输 入 x
    s← s + x
    max ← x
    min ← x
    i← 2
    do until i > 7
        读 入 x
        s← s + x
        if x> max then
            max ← x
        else if x < min then
                min ← x
        i← i + 1
```

```
    enddo
    s← s - max - min
    输 出 s/5
END
```

3.2　顺序结构程序设计

3.2.1　赋值语句

1. 赋值语句的格式

C++中赋值语句的一般格式如下：

变量 = 表达式 ;

其功能为将等式右边的表达式的值赋给等式左边的变量，符号“=”为赋值号。

C++中允许连等式，即允许赋值号右边的表达式为赋值语句，其一般形式如下：

变量 = 变量 = … = 表达式 ;

赋值运算符采用右结合原则。如 a=b=c=5 可理解为 a=(b=(c=5))。

2. 赋值中的类型转换

进行赋值运算时，如果赋值运算符两侧的数据类型不一致，系统将会自动进行类型转换，即把赋值号右边的数据类型转换成左边的变量类型。

【例 3-5】 数据类型的转换实例。

【参考程序】

```
#include <iostream>
#include <iomanip>                    // C++中输出流操纵算子必须包含头文件 iomanip
using namespace std;
int main()
{
    int a,b,c = 322;
    float x,y = 3.14;
    char ch1 = 'a',ch2;
    a = y;                                                    //实型转换为整型
    x = c;                                                    //整型转换为实型
    b = ch1;                                                  //字符转换为整型
    ch2 = c;                                                  //整型转换为字符型
    cout <<"  a = "<< a <<"  x = "<< x <<"  b = "<< b <<"  ch2 = "<< ch2 << endl;
    cout <<"  x = "<< setprecision(5)<< x << endl;            //格式输出,5 位小数位
    cout <<"  x = "<< setprecision(2) << x << endl;           //2 位小数位
    cout << fixed <<"  x = " << x << ndl;                     //普通小数格式
    cout <<"  x = "<< setprecision(2)<< x << endl;           //2 位小数位
    cout <<"  x = "<< setprecision(5)<< x << endl;           //5 位小数位
    cout <<"  x = "<< x << endl;                              //不更改新的格式
    return 0;
}
```