第 3 章　C++语言基础

C++是一种静态类型的、编译式的、通用的、对大小写敏感的且不规则的编程语言，常用于系统开发、引擎开发等领域，支持类、封装、继承、多态等特性。C++语言灵活，运算符的数据结构丰富，具有结构化控制语句，程序执行效率高，还具有高级语言与汇编语言的优点。本章将通过 7 个实验介绍 C++语言的基础知识。

3.1　HelloWorld 实验

3.1.1　实验内容

Notepad++是一款非常适合编写计算机程序代码的文本编辑器，不仅有语法高亮显示功能，还有语法折叠功能，并且支持宏及扩充基本功能的外挂模组，这样就可以实现编译和运行的基本功能。本节的实验内容就是搭建基于 Notepad++软件的开发环境，最后基于 Notepad++软件新建一个 HelloWorld.cpp 文件，并对该文件进行编译和执行。

3.1.2　实验原理

1．命名规范

C++语言区分大小写，即标识符 Hello 与 hello 是不同的。本书中的类名、方法名和源文件名的命名规范如下：① 对于所有的类，类名的首字母为大写，如果类名由若干单词组成，那么每个单词的首字母均为大写，如 MyFirstClass；② 所有的方法名都以小写字母开头，如果方法名由若干单词组成，则第一个单词的首字母为小写，其余单词的首字母均为大写，如 analyzeTempData。

2．C++程序结构

下面以 HelloWorld 实验为例介绍 C++的程序结构，示例代码如下：

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main()
5.  {
6.    cout << "Hello World!" << endl;
7.
8.    return 0;
9.  }
```

其中，第 1 行代码为包含头文件操作，即头文件<iostream>，头文件包含了 C++程序中必需的或有用的信息。

第 2 行代码告诉编译器使用 std 命名空间，命名空间是 C++中一个相对新的概念。

第 4 行中，main()方法是 C++应用程序的入口，程序在运行时，第一个执行的就是 main()方法，该方法与其他方法有很大的不同，例如，方法名必须为 main，方法必须是 int 类型等。

第 6 行代码让编译器输出"Hello World!"并换行。

第 8 行代码终止 main()方法，并向调用进程返回值 0。

3.1.3　实验步骤

双击运行本书配套资料包“02.相关软件”文件夹中的 npp.7.8.5.Installer.exe，在弹出的如图 3-1 所示的对话框中，语言选择 English，然后单击 OK 按钮。

图 3-1　Notepad 安装和配置步骤 1

单击 Next 按钮。在弹出的许可界面中，单击 I Agree 按钮，如图 3-2 所示。

保持默认的安装路径，单击 Next 按钮，如图 3-3 所示。

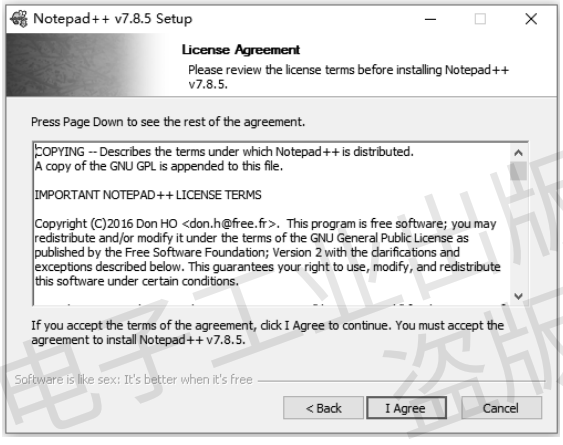

图 3-2　Notepad 安装和配置步骤 2

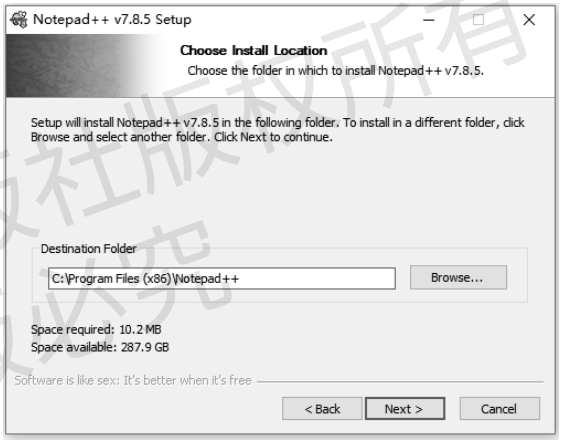

图 3-3　Notepad 安装和配置步骤 3

保持默认的配置，单击 Next 按钮，如图 3-4 所示。

勾选 Create Shortcut on Desktop，单击 Install 按钮，如图 3-5 所示。

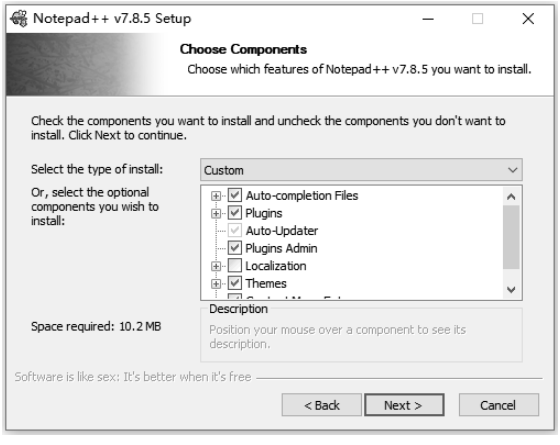

图 3-4　Notepad 安装和配置步骤 4

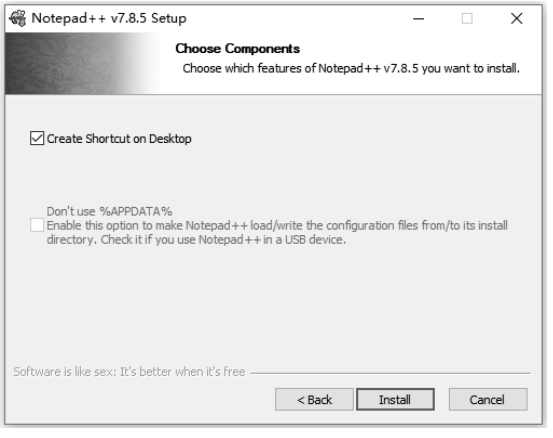

图 3-5　Notepad 安装和配置步骤 5

取消勾选 Run Notepad++ v7.8.5，单击 Finish 按钮，如图 3-6 所示。

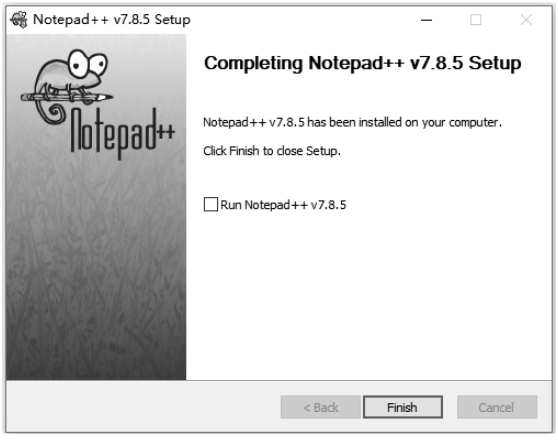

图 3-6　Notepad 安装和配置步骤 6

完成 Notepad++软件安装后，就可以新建和编辑 C++文件，但此时不能在 Notepad++软件中对 C++文件进行编译和执行，还需要安装一些插件。首先，将“02.相关软件\npp 插件”文件夹中的 NppExec 文件夹复制到“C:\Program Files (x86)\Notepad++\plugins”文件夹中。在计算机的开始菜单中，运行 Notepad++软件，执行菜单命令 Plugins→NppExec，选中 Follow $(CURRENT_DIRECTORY)，然后，执行菜单命令 Plugins→NppExec→Execute，或按 F6 键，在弹出的 Execute 对话框（见图 3-7）的 Commands 栏中，输入以下命令：

```
NPP_SAVE
cd $(CURRENT_DIRECTORY)
g++ -o $(NAME_PART).exe $(FILE_NAME)
$(NAME_PART).exe
```

单击 Save 按钮，在 Script name 栏中输入脚本名（不一定是 RUNC++），最后，单击 Save 按钮。

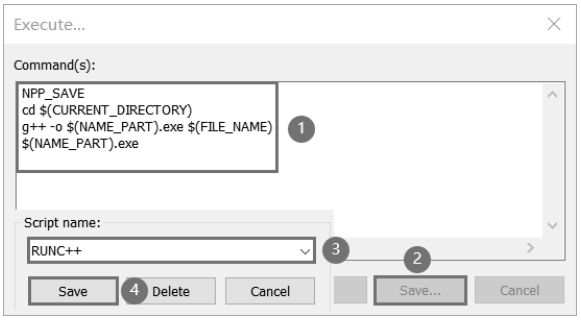

图 3-7　Notepad 安装和配置步骤 7

将上述命令保存为脚本后，由于暂时还不需要编译和执行 C++文件，单击 Cancel 按钮关闭 Execute 对话框，如图 3-8 所示。

下面配置 C++编译环境。右键单击“此电脑”图标（Win7 系统为“计算机”图标），选择“属性”，在如图 3-9 所示的界面中单击“高级系统设置”按钮。

图 3-8　Notepad 安装和配置步骤 8

图 3-9　C++编译环境配置步骤 1

在如图 3-10 所示的“系统属性”对话框中，单击“高级”标签页中的“环境变量”按钮。

图 3-10　C++编译环境配置步骤 2

在如图 3-11 所示的“环境变量”对话框中，双击“系统变量”下的 Path 变量。

图 3-11　C++编译环境配置步骤 3

如图 3-12 所示，在“编辑环境变量”对话框中，单击“新建”按钮，新建一个变量，变量值为 Qt 安装目录下“Tools\mingw730_64\bin”文件夹的绝对路径，本机为“D:\Qt\Qt5.12.0\Tools\mingw730_64\bin”，完成后单击“确定”按钮，关闭“编辑环境变量”对话框，最后在“环境变量”对话框中单击“确定”按钮，保存配置并退出。对于 Win7 系统，直接在 Path 变量值末尾添加“D:\Qt\Qt5.12.0\Tools\mingw730_64\bin”即可。注意，变量值之间需以半角分号隔开，若上一个变量未以分号结尾，则应先添加分号再添加“D:\Qt\Qt5.12.0\Tools\mingw730_64\bin”。

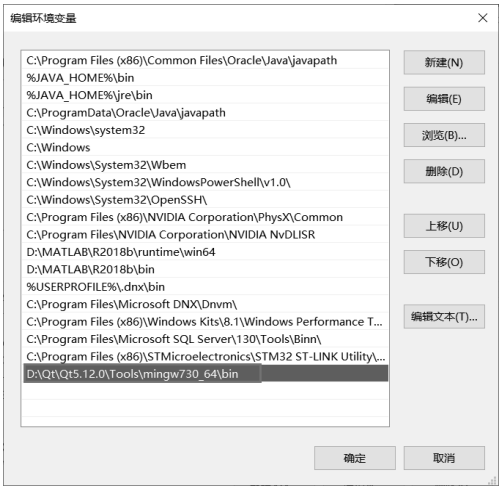

图 3-12　C++编译环境配置步骤 4

然后，利用 Notepad++软件，在“D:\QtProject\CPP01.HelloWorld”文件夹中新建一个 HelloWorld.cpp 文件，输入如图 3-13 所示的代码，其作用类似于 C 语言的 main 函数通过 printf 打印字符串。最后，按 F6 键编译和执行 C++文件，在弹出的 Execute 对话框中，选择图 3-8 中保存的脚本，单击 OK 按钮。执行结果如图 3-13 中的 Console 栏所示，可以看到打印出了“Hello World!”。

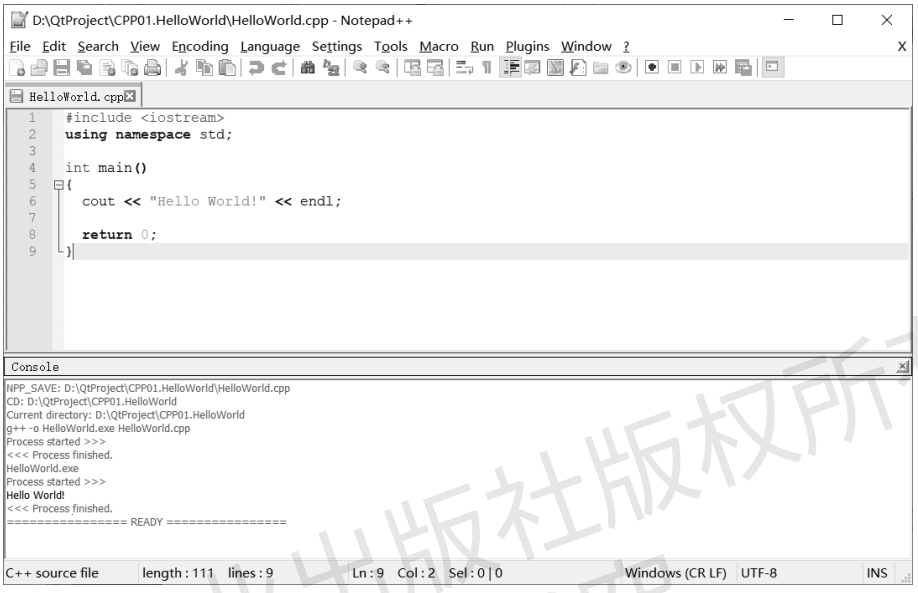

图 3-13　HelloWorld 实验运行结果

此时，软件还不能正常识别中文字符（输出中文字符时会显示乱码）。识别中文字符的设置方法是，执行菜单命令 Settings→Preferences…，选择 New Document，在 Encoding 处选择 ANSI，最后单击 Close 按钮，如图 3-14 所示。

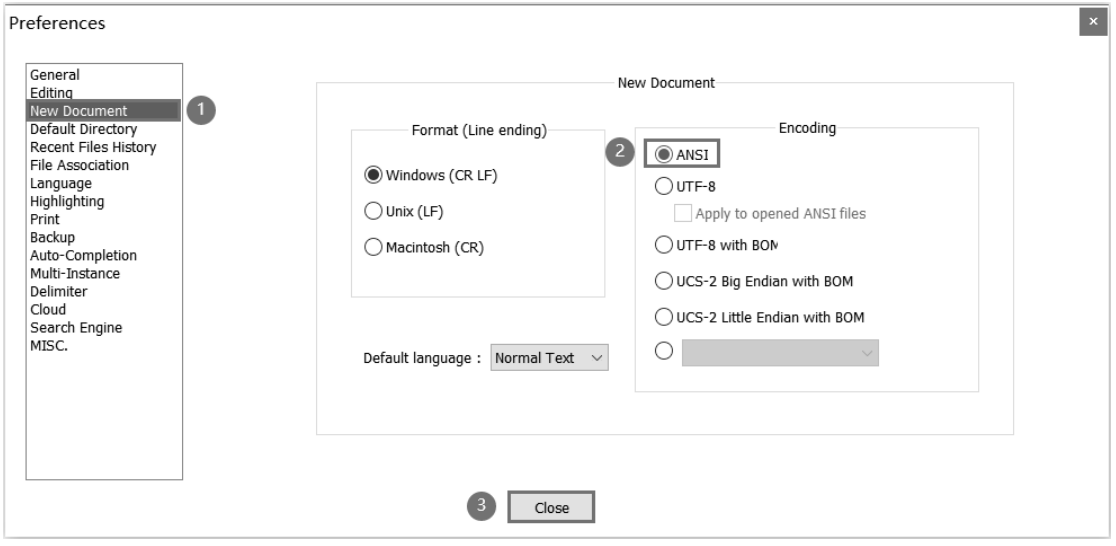

图 3-14　配置 Notepad++

执行菜单命令 Encoding→ANSI，同样将 HelloWorld.cpp 工程的编码格式改为 ANSI，将输出改为中文字符的“你好!”，再次编译，即可看到软件正常输出中文字符了，如图 3-15 所示。

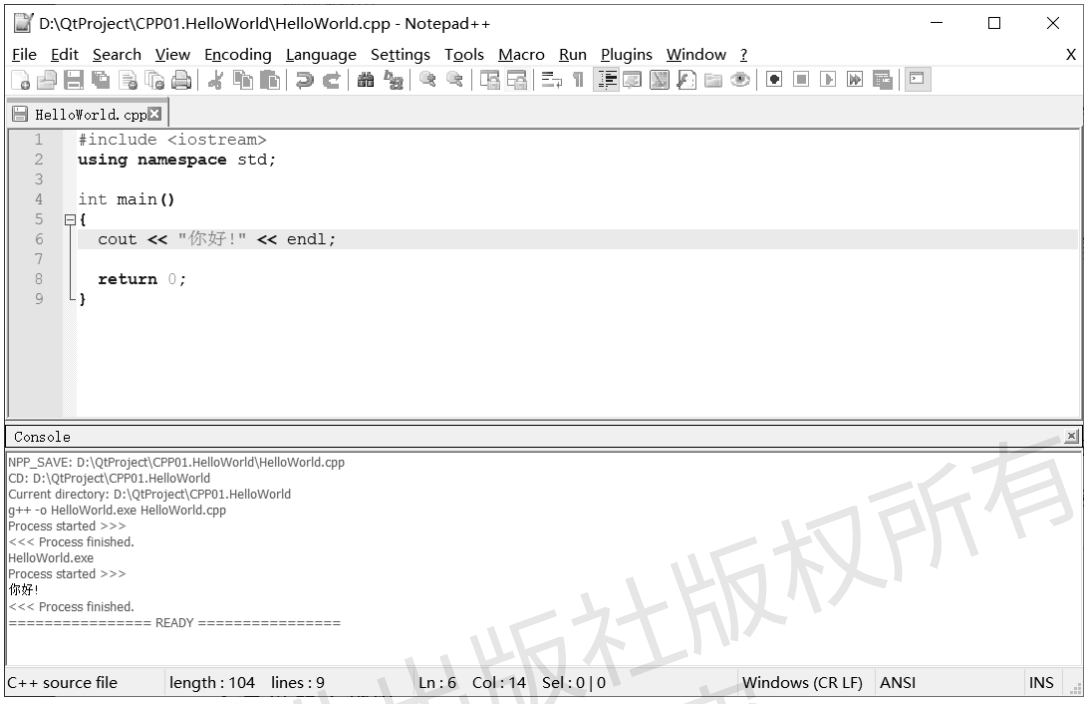

图 3-15　输出中文字符

3.1.4　本节任务

基于 Notepad++软件，新建一个 IntroduceMyself.cpp 文件，编写程序，打印输出自己的姓名、性别、学号和兴趣。

3.2　简单的秒值-时间值转换实验

3.2.1　实验内容

我们知道，一天有 24×60×60 = 86400 秒，如果从 0 开始计算，每天按秒计数，则范围为 0～86399。通过键盘输入一个 0～86399 之间的整数值，包括 0 和 86399，将其转换为小时值、分钟值和秒值，并输出到 Notepad++软件的 Console 栏。

3.2.2　实验原理

1．变量命名规范

通常习惯将类的属性称为类的全局变量（也称成员变量），而将方法中的属性称为局部变量。全局变量在类体中声明，局部变量在方法体中声明，除了全局变量和局部变量，还有一种在类体中以 static 关键字声明的变量，称为静态变量。

（1）全局变量命名以 m 字母开头，后续单词的首字母大写，其余字母小写，如 mECGWave、mHeartRate。

（2）局部变量命名格式为，第一个单词的首字母小写，后续单词的首字母大写，其余字母小写，如 timerStatus、tickVal、restTime。

（3）静态变量命名以 s 字母开头，后续单词的首字母大写，其余字母小写，如 sMaxVal、sScreenResolution。

注意，在 C++语言中，声明一个常量使用 const 关键字，常量命名格式为所有字母大写，不同单词之间用下画线隔开，如 TIME_VAL_HOUR、MAX_VALUE。

2．标准输出流（cout）

cout 是 iostream 类的一个实例，与流插入运算符“<<”结合使用，在代码语句末尾通过 endl 换行，如下所示：

```
cout << "字符串" << 变量 << endl;
```

例如，执行以下语句：

```
int currNum = 12;
cout << "Current num is " << currNum << "." << endl;
```

会打印出如下信息：

```
Current num is 12.
```

3．标准输入流（cin）

cin 是 iostream 类的一个实例，与流提取运算符“>>”结合使用的，如下所示：

```
int currNum;
cout << "请输入数字：";
cin >> currNum;
```

4．标识符与关键字

标识符为有效字符序列，用于标识类名、对象名、变量名、常量名、方法名、数组名和文件名等，标识符可以有一个或多个字符，构成规则如下。

（1）标识符由数字（0～9）、字母（A～Z 和 a～z）、美元符号（$）、下画线（_）及 Unicode 字符集中所有大于 0xC0 的符号组合构成（各符号之间没有空格），C++标识符内不允许出现标点字符，如@、&和%。

（2）标识符的第一个符号为字母、下画线或美元符号，后面可以是任意字母、数字、美元符号或下画线。

标识符分为两类：关键字和用户自定义标识符。关键字是有特殊含义的标识符，如 if、else、true、false 等。关键字是对编译器有特殊意义的固定单词，因此不可以把关键字作为标识符来使用。C++语言的关键字如表 3-1 所示。

表 3-1　C++语言关键字

关 键 字 名
asm、else、new、this、auto、enum、operator、throw、bool、explicit、private true、break、export、protected、try、case、extern、public、typedef、catch、false、register、typeid、char、float、reinterpret_cast、typename、class、for、return、union、const、friend、short、unsigned、const_cast、goto、signed、using、continue、if、sizeof、virtual、default、inline、static、void、delete、int、static_cast、volatile、do、long struct、wchar_t、double、mutable、switch、while、dynamic_cast、namespace、template

5．数据类型

表 3-2 列举了各种变量类型在内存中存储值时需要占用的空间，以及该类型的变量所能存储的最大值和最小值。

表 3-2　C++数据类型

类　型	存储空间	范　围
char	1 字节	-27～（27-1）或 0～（28-1）
unsigned char	1 字节	0～（28-1）
signed char	1 字节	-27～（27-1）
int	4 字节	-231～（231-1）
unsigned int	4 字节	0～（232-1）
signed int	4 字节	-231～（231-1）
short int	2 字节	-215～（215-1）
unsigned short int	2 字节	0～（216-1）
signed short int	2 字节	-215～（215-1）
long int	8 字节	-263～（263-1）
signed long int	8 字节	-263～（263-1）
unsigned long int	8 字节	0～（264-1）
float	4 字节	-2128～（2128-1）
double	8 字节	-21024～（21024-1）
long double	16 字节	-216384～（216384-1）
wchar_t	2 字节或 4 字节	1 个宽字符

6．运算符

C++中的运算符可以分为 6 类，分别是算术运算符、比较运算符、逻辑运算符、位运算符、赋值运算符和杂项运算符。下面依次介绍这些运算符。

（1）算术运算符

算术运算符分为单目运算符和双目运算符，其中，单目运算符包括“++”和“--”，双目运算符包括“+”“-”“*”“/”和“%”，如表 3-3 所示。

表 3-3　算术运算符

运 算 符	格　式	说　明
+	A + B	加法，相加运算符两侧的值
-	A - B	减法，左操作数减去右操作数
*	A * B	乘法，相乘操作符两侧的值
/	A / B	除法，左操作数除以右操作数的商
%	A % B	取余，左操作数除以右操作数的余数
++	A++或++A	自增，操作数的值增加 1
- -	A--或--A	自减，操作数的值减少 1

（2）比较运算符

比较运算符用来比较两个操作数，因此，比较运算符属于双目运算符，运算结果是一个布尔型数。比较运算符如表 3-4 所示。

表 3-4　比较运算符

运 算 符	格　式	说　明
>	A > B	大于，比较左边操作数是否大于右边操作数，结果为 true 或 false
<	A < B	小于，比较左边操作数是否小于右边操作数，结果为 true 或 false
==	A == B	等于，比较左边操作数是否等于右边操作数，结果为 true 或 false
>=	A >= B	大于等于，比较左边操作数是否大于等于右边操作数，结果为 true 或 false
<=	A <= B	小于等于，比较左边操作数是否小于等于右边操作数，结果为 true 或 false
!=	A != B	不等于，比较左边操作数是否不等于右边操作数，结果为 true 或 false

（3）逻辑运算符

逻辑运算符分为单目运算符和双目运算符，其中，单目运算符只有“!”，双目运算符包括“&&”和“||”，如表 3-5 所示。

表 3-5　逻辑运算符

运 算 符	格　式	说　明
&&	A && B	逻辑与，当且仅当两个操作数都为真时，结果才为真
||	A || B	逻辑或，两个操作数中任一个为真，结果为真
!	!A	逻辑非，用于反转操作数的逻辑状态，如果操作数为 true，则结果为 false

（4）位运算符

位运算符主要用于二进制运算，包括“位与”“位或”“位异或”“位非”“左移”“右移”，如表 3-6 所示。

表 3-6　位运算符

运 算 符	格　式	说　明
&	A & B	位与，将两个操作数转换为二进制，然后从高位开始按位进行与操作
|	A | B	位或，将两个操作数转换为二进制，然后从高位开始按位进行或操作
^	A ^ B	位异或，将两个操作数转换为二进制，然后从高位开始按位进行异或操作
~	~A	位非，将操作数转换为二进制，然后从高位开始按位取反
<<	A << n	左移，将左边操作数在内存中的二进制数左移右边操作数指定的位数，左边移空的位填 0
>>	A >> n	右移，将左边操作数在内存中的二进制数右移右边操作数指定的位数，如果最高位是 0，左边移空的位填 0，如果最高位是 1，左边移空的位填 1

（5）赋值运算符

赋值运算符以符号“=”表示，属于双目运算符，如表 3-7 所示。

表 3-7　赋值运算符

运 算 符	格　式	说　明
=	C = A + B	简单的赋值运算符，把右边操作数的值赋给左边操作数
+=	C += A	加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数
−=	C −= A	减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数
*=	C *= A	乘且赋值运算符，把左边操作数乘以右边操作数的结果赋值给左边操作数

续表

运　算　符	格　　式	说　　明
/=	C /= A	除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数
%=	C %= A	求模且赋值运算符，求两个操作数的模赋值给左边操作数
<<=	C <<= 2	左移且赋值运算符
>>=	C >>= 2	右移且赋值运算符
&=	C &= 2	按位与且赋值运算符
^=	C ^= 2	按位异或且赋值运算符
|=	C |= 2	按位或且赋值运算符

（6）杂项运算符

C++还支持其他一些重要的运算符，如表 3-8 所示。

表 3-8　杂项运算符

运　算　符	说　　明
sizeof	sizeof 运算符返回变量的大小。如 sizeof(a)将返回 4，其中 a 为整数
Condition ? X : Y	条件运算符。如果 Condition 为真？则值为 X；否则值为 Y
,	逗号运算符会顺序执行一系列运算
.和>	成员运算符用于引用类、结构和共用体的成员
Cast	强制转换运算符把一种数据类型转换为另一种数据类型。如 int(2.2000)将返回 2
&	指针运算符&返回变量的地址。如&a;将给出变量的实际地址
*	指针运算符*指向一个变量。如*var;将指向变量 var

不同类型的运算符与同类型的运算符一样，有优先级顺序。一个表达式中可以包括同类型的运算符和不同类型的运算符。当多种运算符出现在同一个表达式中时，应先按照不同类型运算符的优先级顺序进行运算。通常运算符优先级由高到低的顺序依次是：算数运算符、比较运算符、逻辑运算符、赋值运算符。如果两个运算符有相同的优先级，那么左边的表达式要比右边的表达式先被处理。可以用括号改变优先级顺序，使得括号内的运算优先于括号外的运算，对于多重括号，总是由内到外强制表达式的某些部分优先运行，括号内的运算总是最优先计算的。

C++语言中运算符的优先级共分为 16 级，其中 1 级为最高，16 级为最低，表 3-9 列出了所有运算符的优先级。

表 3-9　运算符的优先级

优　先　级	运　算　符	描　　述
1	()、[]、->、.	括号、箭头、点
2	++、--、(type)*、sizeof	自增、自减、类型、变量大小
3	*、/、%	乘、除、取余
4	+、-	加和减
5	>>、<<	右移、左移
6	>、<、>=、<=	比较运算符

续表

优 先 级	运 算 符	描　述
7	==、!=	等于、不等于
8	&	位与
9	^	位异或
10	|	位或
11	!	逻辑非
12	&&	逻辑与
13	||	逻辑或
14	?:	条件运算符
15	=、+=、　=、*=、/=、%=、>>=、<<=、&=、^=、|=	赋值运算符
16	,	逗号

3.2.3　实验步骤

首先，基于 Notepad++软件，新建一个 ConvertTime.cpp 文件，保存至“D:\QtProject\CPP02.简单的秒值-时间值转换实验”文件夹中。然后，将程序清单 3-1 中的代码输入 ConvertTime.cpp 文件中。下面按照顺序对部分语句进行解释。

（1）第 1 行代码：包含头文件<iostream>。

（2）第 2 行代码：使用 std 命名空间。

（3）第 6 至 10 行代码：在 main()方法中定义 4 个局部变量，tick 用于保存时间值对应的秒值，hour、min 和 sec 分别用于保存小时值、分钟值和秒值。

（4）第 12 至 13 行代码：通过 cout 打印提示信息，提示用户输入一个 0～86399 之间的值，然后，通过 cin 获取键盘输入的内容。

（5）第 15 至 17 行代码：将 tick 依次转换为小时值、分钟值和秒值。

（6）第 19 至 20 行代码：通过 cout 打印转换之后的时间结果，格式为“小时-分钟-秒”。

程序清单 3-1

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main()
5.  {
6.      int tick = 0;               //0~86399
7.
8.      int hour;                   //小时值
9.      int min;                    //分钟值
10.     int sec;                    //秒值
11.
12.     cout << "Please input a tick between 0~86399" << endl;
13.     cin  >> tick;
14.
15.     hour = tick / 3600;         //tick 对 3600 取模赋值给 hour
16.     min  = (tick % 3600) / 60;  //tick 对 3600 取余后再对 60 取模赋值给 min
17.     sec  = (tick % 3600) % 60;  //tick 对 3600 取余后再对 60 取模赋值给 sec
```

```
18.
19.       //打印转换之后的时间结果
20.       cout << "Current time : " << hour << "-" << min << "-" << sec << endl;
21.
22.       return 0;
23.  }
```

最后，按 F6 键编译和执行 C++文件，在 Notepad++的 Console 栏中，输入 80000 后按回车键，可以看到运行结果，即输出“Current time : 22-13-20”，说明实验成功。Console 栏中的输出信息如下：

```
NPP_SAVE: D:\QtProject\CPP02.简单的秒值-时间值转换实验\ConvertTime.cpp
CD: D:\QtProject\CPP02.简单的秒值-时间值转换实验
Current directory: D:\QtProject\CPP02.简单的秒值-时间值转换实验
g++ -o ConvertTime.exe ConvertTime.cpp
Process started >>>
<<< Process finished.
ConvertTime.exe
Process started >>>
Please input a tick between 0~86399
80000
Current time : 22-13-20
<<< Process finished.
================ READY ================
```

3.2.4　本节任务

2020 年有 366 天，将 2020 年 1 月 1 日作为计数起点，即计数 1，2020 年 12 月 31 日作为计数终点，即计数 366。计数 1 代表“2020 年 1 月 1 日-星期三”，计数 10 代表“2020 年 1 月 10 日-星期五”。参考本节实验，通过键盘输入一个 1～366 之间的值，包括 1 和 366，将其转换为年、月、日、星期，并输出转换结果。

3.3　基于数组的秒值-时间值转换实验

3.3.1　实验内容

通过键盘输入一个 0～86399 之间的整数值，包括 0 和 86399，将其转换为小时值、分钟值和秒值。小时值、分钟值和秒值为数组 arrTimeVal 的元素，即 arrTimeVal[2]为小时值、arrTimeVal[1]为分钟值、arrTimeVal[0]为秒值，并输出转换结果。

3.3.2　实验原理

1．创建一维数组

数组是相同类型数据的有序集合，数组是相同类型的若干数据按照一定的先后次序排列组合而成的。其中，每一个数据称为一个元素，每个元素可以通过一个索引（下标）访问。数组有 3 个基本特点：① 长度确定，数组一旦被创建，其元素个数不可改变；② 各元素类型必须相同，不允许出现混合类型；③ 数组类型可以是任何数据类型，包括基本类型和引用类型。数组变量属于引用类型，数组也可以被看成对象，数组中的每个元素相当于该对象的

成员变量。可根据数组的维数将数组分为一维数组、二维数组……这里只介绍一维数组。

一维数组的创建有两种方式。第一种方式是直接声明，数组大小必须为常量，例如：

```
int arr[4]; //声明一个 int 型数组，包含 4 个数组元素
```

第二种创建方式是使用 new 运算符生成无名动态数组，需要使用指针，其中数组大小可以是常量或变量，但都必须事先给定，可以通过键盘输入大小，例如：

```
int num;
cout << "输入 num 的值：" << endl;
cin >> num;

int *arr1 = new int[4]; //数组大小为常量
int *arr2 = new int[num]; //数组大小为变量
```

2．数组赋值

数组可以在定义的时候就进行初始化赋值，例如：

```
int arr1[] = {1, 2, 3, 4};
int arr2[4] = {1, 2, 3, 4};
```

也可以先定义数组，再赋值，例如：

```
int arr1[4] ;
int *arr2 = new int[4];

arr1 [4] = {1, 2, 3, 4};
for(int i = 0; i < 4; i++)
{
    *(arr1 + i) = i + 1;
}
```


3.3.3　实验步骤

首先，基于 Notepad++软件，新建一个 ConvertTime.cpp 文件，保存至“D:\QtProject\CPP03.基于数组的秒值-时间值转换实验”文件夹中，然后，将程序清单 3-2 中的代码输入 ConvertTime.cpp 文件中。下面按照顺序对部分语句进行解释。

（1）第 8 行代码：声明一个 int 型数组，数组名为 arrTimeVal，并分配内存空间，可以存放 3 个 int 型数据。

（2）第 13 至 15 行代码：通过 tick 计算小时值、分钟值和秒值，分别赋值给 arrTimeVal[2]、arrTimeVal[1]、arrTimeVal[0]。

（3）第 17 至 18 行代码：通过 cout 打印转换之后的时间结果，格式为“小时-分钟-秒”。

程序清单 3-2

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main()
5.  {
6.      int tick = 0;   //0~86399
7.
```

```
8.        int arrTimeVal[3];
9.
10.       cout << "Please input a tick between 0~86399" << endl;
11.       cin  >> tick;
12.
13.      arrTimeVal[2] =  tick / 3600;     //tick 对 3600 取模赋值给 arrTimeVal[2]，即小时值
14.      arrTimeVal[1] = (tick % 3600) / 60; //tick 对 3600 取余后再对 60 取模赋值给 arrTimeVal[1],
                                                                                    即分钟值
15.      arrTimeVal[0] = (tick % 3600) % 60; //tick 对 3600 取余后再对 60 取模赋值给 arrTimeVal[0],
                                                                                      即秒值
16.
17.       //打印转换之后的时间结果
18.       cout << "Current time : " << arrTimeVal[2] << "-" << arrTimeVal[1] << "-" << arrTimeVal[0]
                                                                                  << endl;
19.
20.       return 0;
21.  }
```

最后，按 F6 键编译和执行 C++文件，在 Notepad++的 Console 栏中，输入 80000 后按回车键，可以看到运行结果，即输出“Current time : 22-13-20”，说明实验成功。

3.3.4　本节任务

对于 3.2.4 节的任务，采用数组来实现。

3.4　基于方法的秒值-时间值转换实验

3.4.1　实验内容

通过键盘输入一个 0～86399 之间的整数值，包括 0 和 86399，使用 calcHour()方法计算小时值，用 calcMin()方法计算分钟值，用 calcSec()方法计算秒值，在主方法中通过调用上述三个方法实现秒值-时间值转换，并输出转换结果。

3.4.2　实验原理

1．方法

方法是指一组执行一个任务的语句。每个完整的 C++程序至少有一个方法，即 main() 方法，所有简单的程序都可以定义其他方法，当遇到同类问题时，可以直接调用定义的方法进行处理，减少代码量。

可自定义如何划分代码到不同的方法中，通常是根据每个方法执行一个特定的任务来进行划分。

方法声明告诉编译器方法的名称、返回类型和参数；方法定义提供了方法的实际主体。

C++标准库提供了大量的内置方法，可以被程序调用，如 strcat() 方法用来连接两个字符串，memcpy()方法用来复制内存到另一个位置。

方法也可称为函数、子例程或程序等。

2．方法的定义格式

方法的定义格式如下：

```
修饰符 返回值类型 方法名(参数类型 参数名 1，参数类型 参数名 2……)
{
    方法体
    return 返回值;
}
```

其中，修饰符是可选的，用于定义该方法的访问类型，如 virtual、static。返回值类型是方法返回值的数据类型，如 int、float，有些方法执行所需的操作，但没有返回值，这种情况下返回值类型是关键字 void。方法名是方法的实际名称，方法命名采用第一个单词的首字母小写，后续单词的首字母大写，其余字母小写的格式，如 calcHeartRate、playWave。参数列表是带有数据类型的变量名列表，称为形参，参数之间用逗号隔开，若方法没有参数，则参数列表可以为 void 或为空。方法体包含具体的语句，用于实现该方法的功能。关键字 return 包含两层含义，首先是宣布该方法结束，其次将计算结果返回，如果返回值类型为 void，则不需要 return 语句。

3.4.3　实验步骤

首先，基于 Notepad++软件，新建一个 ConvertTime.cpp 文件，保存至“D:\QtProject\CPP04.基于方法的秒值-时间值转换实验”文件夹中，然后，将程序清单 3-3 中的代码输入 ConvertTime.cpp 文件中。下面按照顺序对部分语句进行解释。

（1）第 4 至 23 行代码：在 ConvertTime 类中定义计算小时值的 calcHour()方法、计算分钟值的 calcMin()方法和计算秒值的 calcSec()方法。

（2）第 36 至 38 行代码：调用 calcHour()、calcMin()、calcSec()方法分别计算小时值、分钟值和秒值。

程序清单 3-3

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int calcHour(int tick)
5.  {
6.       int hour;
7.      hour = tick / 3600;              //tick 对 3600 取模赋值给 hour
8.      return(hour);
9.  }
10.
11. int calcMin(int tick)
12. {
13.     int min;
14.     min = (tick % 3600) / 60;        //tick 对 3600 取余后再对 60 取模赋值给 min
15.     return(min);
16. }
17.
18. int calcSec(int tick)
19. {
20.     int sec;
21.     sec = (tick % 3600) % 60;        //tick 对 3600 取余后再对 60 取模赋值给 sec
22.     return(sec);
23. }
```

```
24.
25.  int main()
26.  {
27.       int tick = 0;                    //0~86399
28.
29.      int hour;                        //小时值
30.      int min;                         //分钟值
31.      int sec;                         //秒值
32.
33.       cout << "Please input a tick between 0~86399" << endl;
34.       cin  >> tick;
35.
36.      hour = calcHour(tick);           //计算小时值
37.      min  = calcMin(tick);            //计算分钟值
38.      sec  = calcSec(tick);            //计算秒值
39.
40.       //打印转换之后的时间结果
41.       cout << "Current time : " << hour << "-" << min << "-" << sec << endl;
42.
43.       return 0;
44.  }
```

最后，按 F6 键编译和执行 C++文件，在 Notepad++的 Console 栏中，输入 80000 后按回车键，可以看到运行结果，即输出“Current time : 22-13-20”，说明实验成功。

3.4.4　本节任务

对于 3.2.4 节的任务，采用方法来实现。

3.5　基于枚举的秒值-时间值转换实验

3.5.1　实验内容

通过键盘输入一个 0～86399 之间的整数值，包括 0 和 86399，使用 calcTimeVal()方法计算时间值（小时值、分钟值和秒值），通过枚举区分具体是哪一种时间值，返回值为是否计算成功标志，在 main()方法中通过调用 calcTimeVal()实现秒值-时间值转换，并输出转换结果。

3.5.2　实验原理

1．枚举类型

常量可以通过关键字 const 和 static 定义在类或接口中，这样在程序中就可以直接使用，并且该常量不能被修改，示例代码如下：

```
static const int TIME_VAL_HOUR = 0x01;
static const int TIME_VAL_MIN  = 0x02;
static const int TIME_VAL_SEC = 0x03;
```

如果一个变量只有几种可能的值，则可以定义为枚举类型，枚举就是把可能的值一一列举出来，变量的值只限于所列举值的范围内。而且，枚举类型提供了参数类型检测功能，如枚举类型作为某方法的形参时，调用该方法只接受枚举类型的常量作为参数。使用枚举类型

定义常量的示例代码如下：

```
enum EnumTimeVal
{
    TIME_VAL_HOUR,
    TIME_VAL_MIN,
    TIME_VAL_SEC,
    TIME_VAL_MAX
};
```

其中，enum 是定义枚举类型的关键字，当需要在类中使用该常量时，可以使用 EnumTimeVal.TIME_VAL_HOUR 来表示。注意，在 switch…case…语句中使用枚举常量时，不需要枚举类型，直接使用 TIME_VAL_HOUR 即可。

2．switch…case…语句

switch…case…语句用于判断一个变量与一系列值中的某个值是否相等，每个值称为一个分支，switch…case…语句的语法如下：

```
switch(表达式)
{
    case 常量值 1:
        语句块 1
        [break;]
    …
    case 常量值 n:
         语句块 n
         [break;]
     default :
         语句块 n+1
         [break;]
}
```

switch 语句中表达式的值必须是整型、字符型或字符串类型。同样，case 常量值也必须是整型、字符型或字符串类型，而且表达式的值必须与 case 常量值的数据类型相同。switch…case…语句遵循以下规则：

（1）当表达式的值与 case 常量值相等时，执行 case 语句后面的语句块，直至遇到 break 语句。

（2）当遇到 break 语句时，switch…case…语句终止，程序跳转到 switch…case…语句后面的语句执行。

（3）case 语句不一定必须包含 break 语句，如果没有 break 语句，程序则继续执行下一条 case 语句，直至出现 break 语句。

（4）switch 语句可以包含一个 default 分支，该分支通常是 switch 语句的最后一个分支（可以在任意位置，但建议是最后一个），default 在没有 case 语句的值和变量值相等的情况下执行，default 分支可以不包含 break 语句。

3.5.3　实验步骤

首先，基于 Notepad++软件，新建一个 ConvertTime.cpp 文件，保存至“D:\QtProject\CPP05.基于枚举的秒值-时间值转换实验”文件夹中，然后，将程序清单 3-4 中的代码输入

ConvertTime.cpp 文件中。下面按照顺序对部分语句进行解释。

（1）第 4 至 10 行代码：定义一个名称为 EnumTimeVal 的枚举类型，并使用该枚举类型定义 4 个常量，分别为 TIME_VAL_HOUR、TIME_VAL_MIN、TIME_VAL_SEC 和 TIME_VAL_MAX。

（2）第 12 至 31 行代码：基于枚举和 switch…case…语句，计算小时值、分钟值和秒值，这里的枚举常量不需要枚举类型前缀。

（3）第 44 至 46 行代码：通过调用 calcTimeVal()方法计算小时值、分钟值和秒值，类型通过枚举常量区分，这里的枚举常量必须带有枚举类型前缀。

程序清单 3-4

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  enum EnumTimeVal
5.  {
6.       TIME_VAL_HOUR,
7.       TIME_VAL_MIN,
8.       TIME_VAL_SEC,
9.       TIME_VAL_MAX
10. };
11.
12. int calcTimeVal(int tick, EnumTimeVal type)
13. {
14.      int TimeVal = 0;
15.
16.      switch(type)
17.      {
18.           case TIME_VAL_HOUR:
19.                TimeVal = tick / 3600;
20.                break;
21.           case TIME_VAL_MIN:
22.                TimeVal = (tick % 3600) / 60;
23.                break;
24.           case TIME_VAL_SEC:
25.                TimeVal = (tick % 3600) % 60;
26.           default:
27.                break;
28.      }
29.
30.      return TimeVal;
31. }
32.
33. int main()
34. {
35.      int tick = 0;   //0~86399
36.
37.     int hour; //小时值
38.     int min;  //分钟值
39.     int sec;  //秒值
40.
```

```
41.      cout << "Please input a tick between 0~86399" << endl;
42.      cin  >> tick;
43.
44.      hour = calcTimeVal(tick, TIME_VAL_HOUR);
45.      min  = calcTimeVal(tick, TIME_VAL_MIN);
46.      sec  = calcTimeVal(tick, TIME_VAL_SEC);
47.
48.      //打印转换之后的时间结果
49.      cout << "Current time : " << hour << "-" << min << "-" << sec << endl;
50.
51.      return 0;
52.  }
```

最后，按 F6 键编译和执行 C++文件，在 Notepad++的 Console 栏中，输入 80000 后按回车键，可以看到运行结果，即输出“Current time : 22-13-20”，说明实验成功。

3.5.4　本节任务

对于 3.2.4 节的任务，采用枚举来实现。

3.6　基于指针的秒值-时间值转换实验

3.6.1　实验内容

通过键盘输入一个 0～86399 之间的整数值，包括 0 和 86399，将其转换为小时值、分钟值和秒值，并且将小时值、分钟值和秒值分别存放在指针 p 的(p+2)地址、(p+1)地址和(p+0)地址，最后输出转换结果。

3.6.2　实验原理

1．指针

指针是一个变量，它的值为另一个变量的地址，即内存位置的直接地址。在使用指针存储其他变量的地址前，首先要对指针进行声明，示例代码如下：

```
int *ip;          //一个整型的指针
double *dp;       //一个 double 型的指针
float *fp;        //一个浮点型的指针
char *chp;        //一个字符型的指针
```

所有指针的值不论实际数据类型是整型、浮点型、字符型的，还是其他的数据类型，都是一个代表内存地址的十六进制数。不同数据类型的指针之间唯一不同的是指针所指向的变量或常量的数据类型不同。

2．NULL 指针

NULL 指针是一个定义在标准库中的值为 0 的常量，赋为 NULL 值的指针称为空指针。在大多数操作系统中，程序不允许访问地址为 0 的内存，因为该内存是为操作系统保留的。内存地址 0 有特别重要的意义，它表明该指针不指向一个可访问的内存位置。

很多时候未初始化的变量存有一些垃圾值，导致程序难以调试。通常在声明指针时，如果没有确切的地址可以赋值，则可以为指针变量赋一个 NULL 值。若将所有未使用的指针赋

值为 NULL，同时在使用指针前对指针进行判空处理，则可以防止误用一个未初始化的指针。

3．释放内存

当需要一个动态分配的变量时，必须向系统申请获取相应的内存来存储该变量。当该变量使用结束时，需要显式释放它所占用的内存，使得系统可以回收该内存，以防止内存泄露，同时可以对该内存进行再次分配，做到重复利用。

释放内存主要有两种方式：delete 和 free。一般通过 new 分配的内存需要配套使用 delete 来释放内存；通过 malloc 分配的内存需要配套使用 free 来释放内存。注意，使用 malloc 时需要包含头文件 stdlib.h。

delete 和 free 只释放了指针指向的内存，指针本身未被释放，即指针指向的地址不变。内存释放并不意味着把指针指向地址存储的值赋值为 0，内存被释放的指针此时存储的是一些垃圾值，此时的指针未被赋值，也就是通常所说的“野指针”。

指针仍指向已经释放的内存会很危险，因为该内存可能已经被系统回收，重新分配给了其他变量，而在编程过程中稍有不慎，会误以为这是一个合法的指针而对该指针进行赋值，从而导致该内存中存放的变量值被覆盖，下面举例说明。

```
#include <iostream>
using namespace std;

int main()
{
     int *p = new int;  //声明一个指针变量 p，分配一个 int 类型的内存
    *p = 99;            //将 99 赋值给 p 的地址

     cout << "打印指针 p 指向的地址："<< p << endl;
    cout << "打印赋值为 99 后指针 p 存放的值："<< *p << endl;

    delete p;           //释放指针 p 的内存

    cout << "打印指针 p 释放内存后存放的值："<< *p << endl;

    int *k = new int;   //声明一个指针变量 k，分配一个 int 类型的内存
    *k = 100;           //将 100 赋值给 k 的地址

    cout << "打印指针 p 释放内存后指向的地址："<< p << endl;
    cout << "打印指针 k 指向的地址："<< k << endl;

    *p = 88;            //将 88 赋值给 p 的地址

    cout << "将 88 赋给 p 的地址后，指针 p 存放的值："<< *p << endl;
    cout << "将 88 赋给 p 的地址后，指针 k 存放的值："<< *k << endl;

    delete k;           //释放指针 k 的内存

    return 0;
}
```

输出结果如下：

```
Process started >>>
```

```
打印指针 p 指向的地址：0x1f1100
打印赋值为 99 后指针 p 存放的值：99
打印指针 p 释放内存后存放的值：2064824
打印指针 p 释放内存后指向的地址：0x1f1100
打印指针 k 指向的地址：0x1f1100
将 88 赋给 p 的地址后，指针 p 存放的值：88
将 88 赋给 p 的地址后，指针 k 存放的值：88
<<< Process finished.
================ READY ================
```

从输出结果可以看到，指针 p 被释放内存后其存放的值不为 0，而是一个垃圾值，且指针指向的地址未变；指针 p 被释放内存后，系统回收了该内存并赋给新建的指针 k，这时指针 k 与指针 p 指向的地址相同，导致后面误用指针 p 时，对指针 p 指向的地址赋值 88，覆盖了指针 k 指向的地址赋的值 100，从而导致数据不准确，所以在执行释放内存操作后，应该及时将指针指向 NULL 地址。

3.6.3　实验步骤

首先，基于 Notepad++软件，新建一个 ConvertTime.cpp 文件，保存至“D:\QtProject\CPP06.基于指针的秒值-时间值转换实验”文件夹中，然后，将程序清单 3-5 中的代码输入 ConvertTime.cpp 文件中。下面按照顺序对部分语句进行解释。

（1）第 8 行代码：声明一个指针变量 p，动态分配 3 个 int 类型内存，同时将内存空间全赋初值为 0。

（2）第 15 至 17 行代码：通过 tick 计算小时值、分钟值和秒值，分别赋值给 *(p + 2)、*(p + 1)和*(p + 0)。

（3）第 23 至 24 行代码：通过 delete 释放内存，并将指针指向 NULL 地址。

程序清单 3-5

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int main()
5.  {
6.      int tick = 0;               //0~86399
7.
8.      int *p = new int[3]();      //动态分配 3 个 int 类型内存，同时内存空间全赋初值为 0
9.
10.     cout << "Please input a tick between 0~86399" << endl;
11.     cin  >> tick;
12.
13.     if(NULL != p)               //对指针 p 判空
14.     {
15.         *(p + 2) =  tick / 3600;         //tick 对 3600 取模赋值给*(p + 2)，即小时值
16.         *(p + 1) = (tick % 3600) / 60;   //tick 对 3600 取余后再对 60 取模赋值给*(p + 1)，
即分钟值
17.         *(p + 0) = (tick % 3600) % 60;   //tick 对 3600 取余后再对 60 取模赋值给*(p + 0)，
即秒值
18.
19.         //打印转换之后的时间结果
20.         cout << "Current time : " << *(p + 2) << "-" << *(p + 1) << "-" << *(p + 0) << endl;
```

```
21.       }
22.
23.       delete p; //释放内存空间，指示系统随时可回收内存，指针指向地址不变
24.       p = NULL; //指针指向 0 地址，即置空
25.
26.       return 0;
27.  }
```

最后，按 F6 键编译和执行 C++文件，在 Notepad++的 Console 栏中，输入 80000 后按回车键，可以看到运行结果，即输出“Current time : 22-13-20”，说明实验成功。

3.6.4　本节任务

对于 3.2.4 节的任务，采用指针来实现。

3.7　基于引用的秒值-时间值转换实验

3.7.1　实验内容

通过键盘输入一个 0～86399 之间的整数值，包括 0 和 86399，声明小时值、分钟值的引用，操作引用别名获取转换的小时值和分钟值；定义一个转换秒值的方法，将引用作为形参，然后调用该方法获取转换的秒值，最后输出转换结果。

3.7.2　实验原理

1．引用

引用是指对某个已存在的变量重新定义一个名称，一旦把引用初始化为某个变量，就可以使用该引用名或变量名来指向变量。它只表示该引用名是目标变量名的一个别名，本身不是一种数据类型，因此引用不占存储单元。

2．引用与指针的异同

（1）引用是别名，指针是实体。

（2）引用必须连接到一片合法的内存，不存在空引用。指针存在空指针。

（3）引用在被声明时就必须初始化，指针可以单独初始化。

（4）引用被初始化为一个对象后，就不能被指向另一个对象，而指针可以随意更改指向的对象。

（5）引用不需要分配内存，指针需要分配内存。

（6）引用与指针都是与地址相关的概念，指针指向内存的地址，引用为内存地址的一个别名。

3.7.3　实验步骤

首先，基于 Notepad++软件，新建一个 ConvertTime.cpp 文件，保存至“D:\QtProject\CPP07.基于引用的秒值-时间值转换实验”文件夹中，然后，将程序清单 3-6 中的代码输入 ConvertTime.cpp 文件中。下面按照顺序对部分语句进行解释。

（1）第 4 至第 8 行代码：定义一个计算秒值的方法，形参部分为引用。

（2）第 18 至 20 行代码：分别声明小时与分钟的引用。

（3）第 25 至 27 行代码：通过 tick 计算小时值和分钟值，分别赋值给 s_hour 与 s_min；将 tick 与 sec 作为实参传入，调用 calcSec()方法获取 sec 的值。

程序清单 3-6

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  //把引用作为形参，调用时实参可直接为变量本身
5.  void calcSec(int& s_tick, int& s_sec)
6.  {
7.       s_sec  = (s_tick % 3600) % 60; //tick 对 3600 取余后再对 60 取模赋值给 sec
8.  }
9.
10. int main()
11. {
12.      int tick = 0;               //0~86399
13.
14.     int hour;                    //小时值
15.     int min;                     //分钟值
16.     int sec;                     //秒值
17.
18.      //定义引用
19.     int& s_hour = hour;          //小时值引用
20.     int& s_min  = min;           //分钟值引用
21.
22.      cout << "Please input a tick between 0~86399" << endl;
23.      cin  >> tick;
24.
25.      s_hour = tick / 3600;       //tick 对 3600 取模赋值给 hour
26.      s_min  = (tick % 3600) / 60; //tick 对 3600 取余后再对 60 取模赋值给 min
27.      calcSec(tick, sec);         //直接将变量作为实参计算 sec 的值
28.
29.      //打印转换之后的时间结果
30.      cout << "Current time : " << hour << "-" << min << "-" << sec << endl;
31.
32.      return 0;
33. }
```

最后，按 F6 键编译和执行 C++文件，在 Notepad++的 Console 栏中，输入 80000 后按回车键，可以看到运行结果，即输出“Current time : 22-13-20”，说明实验成功。

3.7.4　本节任务

对于 3.2.4 节的任务，采用引用来实现。

本 章 任 务

本章共有 7 个实验，首先学习各实验的实验原理，然后按照实验步骤完成实验，最后按照要求完成本节任务。

本 章 习 题

1．在 C++程序中，第 1 个执行的方法是什么？该方法有什么特点？

2．标识符是指什么？

3．signed 与 unsigned 通常用于修饰什么数据类型？在数据类型前添加 signed 与 unsigned 有什么意义？

4．简述数组的特点。

5．什么是指针？

6．简述 NULL 指针的意义和作用。

7．什么是引用？

8．简述引用与指针的异同点。