第1章·C++快速入门

由于本章的定位是“快速入门”，所以我们注定会忽略很多细枝末节。但是这并不意味着它不重要。这些细枝末节往往对一个人的编码水平有着决定性的影响。所以我会在后面尽量提到“究竟忽略了哪些细节”，请读者自行查阅学习。相对于前面章节的“讲故事”，本章的内容可能会显得枯燥一些。但是作为后面章节的基础，希望大家能够倾尽全力去学习这些内容。

1. 关于C++

凡是语言，都一定有着自己的规则。而不同于英语、汉语这种自然语言，作为计算机所能识别的语言，它是需要在严格约束下避免二义性的，所以说我们将学到的C++语言，是一门语法规则非常严格，但又“灵活多变”的语言。

所谓非常严格，是指C++中不允许出现歧义，这种没有歧义的文法是经过严格的文法规则的限制、和一些约定来约束的。所以，如果在你的代码中出现了自己都会觉得有歧义的地方，往往就是程序中容易出错的地方。

而所谓灵活多变，就是指在掌握C++严格的语法之后，可以写出无穷无尽充满创意的程序。从黑框(控制台程序)，到带有图形界面(GUI)的程序，再到游戏，都可以利用C++的语法加上你脑中的算法搞定！而所有的这些灵活多变的内容，都是在严格的文法约束下实现的。所以说C++语言，是一门语法规则非常严格，但又灵活多变的语言！

C++是一门高级语言，所谓高级语言正如你们在前面章节中看到的，是不同于机器语言、汇编语言；能更加贴近人类思维的一门语言。C++是在C语言的基础上，增加了例如面向对象编程等诸多特性的程序设计语言。在程序设计竞赛中被广泛采用。

非常不严格地讲：我们常常所说的“C++”，包含了这门语言和一系列的工具。语言顾名思义就不多阐述。而工具实际上包含了很多内容：

我们将C++语言所编写的代码叫做“源代码”。而源代码是不能直接被计算机执行的（回想我们在上一章中讲述的内容）。而是应该利用编译程序，将源代码编译成计算机能够识别的机器代码然后再执行。而执行编译工作的工具就叫做“编译器”。

除此之外，仅仅有C++语言，我们连基本的功能（例如屏幕输出）都实现不了。即便是高级语言，想要与硬件设备交互，一般都需要直接编写一定的机器语言或者汇编语言，所以才能让我们的程序操作硬件：屏幕打印、屏幕输入、文件操作等等。而我们想要通过一己之力实现这些最底层的操作是很困难的。好在C++提供了一组可以被调用的工具。这些工具其实就是别人写好的代码，通过C++语法能够接受的形式所调用，最终辅助我们完成底层调用的功能，让我们的精力能够集中在更重要的算法设计上而不是千篇一律的与底层的通讯上。我们称这种“别人的代码”叫做“库”。就如同仓库一样，我们可以调用仓库中的内容来实现我们的意图。

那么我们的程序是怎么与别人的程序产生关联的呢？这有很多种情况，我只讲述最常见的一种：静态链接。我们的程序在调用库的时候，编译器可以仅仅在相应的位置上做个标记，表明这里的程序是调用了库而不是我自己的。这么做的好处就是可以充分地提高编译效率。因为库中的代码往往经过严格的编写、测试，基本能保证没有问题，所以我们只需要保存该库编译完毕的状态，没有必要在编译自己的程序时还要将别人的库再编译一遍。我们只需要编译自己的程序就可以了。所以我们的程序与库之间是并列关系，而不是包含关系。但是想要让自己的程序能够变成可执行程序，仅仅是这种“做标记”的方法是不可接受的。因为你所需要的库文件在你的电脑上会有，但并不意味着别人的电脑上也会有。再加上可执行程序的复杂原理，我们必须将这些静态库与我们的程序绑定在一起。而将编译完的“目标代码(Object Code)”与库文件链接形成“可执行文件”的工具叫做“链接器”。

也就是说，C++程序设计流程是：1，编写代码。2，编译成目标代码。3，链接成可执行文件。

一般来说，想要实现上面的流程，需要利用操作系统提供的黑框(Linux下的终端、Windows下的控制台)，再加上一大堆非常复杂的命令才能实现。编译命令甚至有可能比源代码还要复杂。如果需要对自己的代码进行调试，那可是更加麻烦。所以聪明人们就为懒人发明了又一件工具：集成开发环境（Integrated Development Environment，IDE）。所谓开发环境，可以将其看作是写代码的一个工具。我们可以在这个开发环境中编写代码、代码着色、自动排查错误等。所谓“集成”就是指将“编译器”、“连接器”都集中在IDE中。程序员可以简简单单地点击按钮，就可以实现整套编译链接运行调试等功能。

在ACM/ICPC的比赛中，通常使用的编译器是GCC(GNU Complier Collection)。GCC是一个支持很多种语言的开源的编译器的集合。它可以安装在Windows（不推荐）和Linux等多种操作系统上。而在比赛中普遍使用的是安装了Linux操作系统和GCC编译器的开发环境，并且配置了某些IDE来让学生进行开发。

关于IDE：对于Windows来说，我门推荐使用Dev-C++。对于Linux，我们推荐使用Code-Blocks。

1. 小试牛刀

说了这么多，是时候看一看C++的代码长得是什么样子了：

代码2-1,传说中的Helloword!

00 #include <iostream>

01 using namespace std;

02

03 int main()

04 {

05 cout<<”Hello IMUDGES!”<<endl;

06 return 0;

07 }

上面就是传说中的C++代码。实现的功能也很简单，输出“Hello IMUDGES”。关于程序的前两行，请参考课后习题2，大家可以查阅资料自行了解。我们主要来看3-6行。

第3行中我们看到了int main()这种写法。在这里你只需要知道，这是一个名字叫做main，返回值是int的函数。具体什么是函数先不用管。第4和第6行之间，用大括号括起来了一段代码。这表明的这部分代码是属于main函数中的。

程序想要执行，必须要找到执行的入口，也就是第一行执行的操作。在C++中，这个入口就是main()函数。程序将会在main函数中第一行数据开始依次执行，而执行完毕，这个main()函数也会返回一个值给操作系统。

也就是说，你所写的程序中，必须有一个叫做main()的函数。而你应该将你要进行的操作放在这个函数中。

如果大家对于上面的这部分内容难以理解也没关系，因为在后面你的教程中我将会为大家仔细解释个中缘由！

1. 高级语言程序的基本结构

正像说话和写文章需要分节分段一样，高级语言的程序也具有层次结构。尽管细节有着些许差别，但是绝大部分高级语言之间对于层次结构都惊人地相似。在很多高级语言中，程序一般由若干个文件构成，文件由类型定义、函数等组成，而函数则是由多条语句按照一定的编程风格组织在一起的。其编程风格可以是具有传统的面向过程的模块化或结构化的程序设计，也可以是比较符合人类认知规律的面向对象程序设计风格。关于面向对象程序设计，我们将会在后面的章节中介绍。

我们再来看这样一段程序：

程序2-2：

1. #include <iostream>
2. using namespace std;

02 void PrintString(char\* str){

03 cout<<”Hello,”<<str<<”!”<<endl;

04 }

05

06 void main(){

07 char Name[80];//name

08 cout<<”Enter your name:”;

09 cin>>Name;

10 PrintString(Name);

11 }

程序运行时的屏幕显示为:

Enter your name: IMUDGES

Hello,IMUDGES!

上面的C++程序由预处理命令、函数定义、主函数、输入输出语句、变量定义语句、函数调用语句、注释等等内容组成。

* 预处理指令

预处理指令主要是以“#”开头的命令。预处理指令是程序员命令编译器所进行的操作，主要是为编译器提供辅助信息所使用。例如程序第00行，#include指令是指载入“<>”中所指定的文件。想要使用系统所提供的各种库功能，通常来说要通过#include来载入指定文件来实现。例如在文件“iostream”中保存着cin,cout,<<,>>等函数引用说明。cstring中则包含了一些关于字符串的操作。

* 关于using namespace

using namespace，顾名思义是使用命名空间的意思。关于命名空间的主题已经超出速成教程的范围。对于你现在编写的程序，“using namespace std;”这样的一行代码一般是必不可少的！

* 函数

在上述代码的第01行和第05行，PrintString(char\* str)和main()被称为函数。函数在此的概念与数学上的概念非常类似！数学上的函数就是对自变量到因变量上的一组映射。而在C++中的函数也类似。其自变量也称为参数，其因变量称为返回值。

每一个应用程序都会拥有一个称为“主函数”的函数main()。这个函数被作为整个程序执行时的入口。即应用程序的执行通常是从main()中的第一条可执行语句开始的。每个程序中只能有一个主函数，在主函数中可以再调用其它函数，在其他函数中再调用另一些函数，从而让整个程序运转起来。

复杂的程序通常会有大量的函数，为了让这些代码便于组织和管理，可以将一些关系密切的函数组织在一起，放在同一个文件中，因此程序通常由多个文件组成。按照惯例，含有类型、函数、类等说明的文件都被称为头文件，其后缀名为.h。而定义函数和类等具体实现的文件称为实现文件，后缀名为.cpp。

* 输入与输出

输入和输出是指程序的运行者与程序之间的交互。与外界没有任何交互（不需要输入、也不产生输出）的程序是没有存在的意义的。作为ACM/ICPC等比赛所编写的程序，其检测方法基本上是通过给定输入与输出进行比较，再判断结果的正误。

在上面例子中，我们使用cout来输出，使用cin来输入。

cout<<”aaa”是指将“aaa”输出到屏幕上

cin>>a是指将屏幕上输入的内容存入变量a中。关于这几个箭头大家可以这样去记忆。cout是可以当做屏幕，箭头指向的是输出的方向，也就是将“aaa”输出到屏幕上。而cin可以看做是键盘，从键盘输入，箭头指向目标——变量a。

而endl其实是“End Line”的意思，就是说在当前位置输出一个换行符。

* 语句

语句是计算机程序中最基本的单位。语句通常是一个简单的表达式例如1+1；这个表达式产生值2.当执行完毕之后，这个表达式的结果也就不复存在。我们必须要利用一种方法将它存储起来，与数学上的形式类似：x=1+1，这样，1+1的运算结果就保存在变量x中。这个带有等号的语句称为赋值语句。

* 注释

注释是为程序员为自己的源代码所写的说明文字、或者有其他记录性用途。编译器在看到注释符号之后，会完全忽视注释掉的区域。也就不会将这部分区域的代码编译进程序中。在C++中有单行注释，使用”//”来表示，在该行中，这两个斜杠后面的内容都会被当做注释并被编译器所忽视掉。同时也有多行注释/\* \***/在“/**\*”出现之后，直到“\*/”出现之前，所有的代码都会被当做注释。例如：/\* 我是注释！ \*/。

1. 基本数据类型

在上面的代码中，曾经出现过“变量”这种东西。我们再来详细讨论之。此处的变量与数学中的变量类似。

它首先是一个量。既然是个量，就一定会有值。同时它还可变，说明它的值是可以改变的。也就是说，变量是一个“能够改变数值的一个量”。变量在被使用前必须要先进行定义，在这个时候计算机会在内存中开辟出一块空间，用以存储这个变量。在变量有效的情况下可以使用这个变量中的值。

那么这些变量究竟在内存中是如何表示的呢？首先我们要明确数据类型这个概念。

计算机内存是一种有限的、逻辑上线性排列的电子元件，它的逻辑结构是按照顺序依次递增一个字节。每个字节是8位二进制数。也就是说，我们能表示的数据有从00000000B~11111111B这么多个，也就是2的8次方，共256个。当然，如果我们用两个字节的空间，则可以产生00000000 00000000B~11111111 11111111这么多个，也就是2的16次方，共65536个数字。但是如果这个长度还不够呢？我们还可以用4字节、8字节的方式来表示“一整个”我们想要表示的数据。那这“一整个”究竟表示着啥内容呢？

我们就单独讨论1字节（Byte）的空间都能用来干啥。

如果这一字节用来存储非负整数，我们假定就是仅仅将二进制转换到十进制，那么能够表示0~255这256个数字。

如果这一字节用来储存“可正可负”的整数，我们又规定，这8位数位的首位表示符号，1为负号，0为正号。然后用后面的七位表示数字，则我们能够表示从11111111B~01111111B的数字，其中包括10000000B这个-0和00000000B这个+0，也就是说能保存255个数（因为0有两个）。其实在真实的计算机系统，普遍已经不采用这种看似直观却浪费了一个数字的方法。利用这种方法能够实现整数类型取值范围-128~127.（具体如何实现我们不多做讨论）。

如果我们现在想要表示字符，我们可以规定：65D表示A，用66D表示B以此类推，甚至使用数字48D表示字符‘0’。这个对应规则就称为ASCII字符集。

可见，同样是1个字节的空间，我们可以通过不同的方式规定这8位空间里存储的内容的真实含义。而对于内存而言，它完全不知道这一字节的数据，究竟按照上述哪种“约定”存储数据——其实作为内存也没必要知道，因为这个“约定”是程序员的工作。

这个工作就是“给定变量以数据类型”。例如我们有一个变量，它是4字节（通常来说）长的整数类型，我们用int来“修饰它”。又比如我们有一个变量是4字节长的无符号整数类型，我们用unsigned int来“修饰它”。这种修饰主要体现在了变量定义上：

程序2-3：变量定义

00 int a = -10;

01 unsigned int b = 100;

02 float c = 0.12345;

03 char d = ‘R’;

04 short e = 123;

05 double f = 1.23456;

06 bool g;

07 g = true;

每行语句开始前，都有一个“表示类型”的单词，例如int，例如unsinged int，例如float等等。这些单词由系统规定，不能随便更改（但是可以自己创造类型）。后面紧跟着一个“表示名称”的单词，这就是变量名。变量名可以由字母、下划线、数字组成，但是变量名的第一个字母不能以数字开头。

等号的前半部分叫做“变量定义”。变量的赋值操作在定义之后随时可以进行，甚至可以在定义的同时进行. 所谓赋值操作就是指“=xxx”。例如程序2-3中，从00~05行都是在变量定义的同时进行赋值，而06和07行分别在进行变量定义、赋值。在C++中，我们必须遵循“先定义、后使用”的约束。

上面的代码中我们定义了几个变量。

变量a是int类型（称整型），通常占用4字节。

变量b是unsigned int类型（称无符号整型），通常占用4字节。

变量c是float类型（称单精度浮点型），通常占用4字节。

变量d是char类型（称字符型），通常占用1字节

变量e是short类型（称短整型），通常占用2字节

变量f是double类型（称单精度浮点型），通常占用8字节

变量g是bool类型（布尔类型），通常占用1字节。它的用途我们后面会提到。

我们再来考虑一下变量类型。一个变量无论是什么类型，它在内存中一定是以1和0的形式存在的。而上述代码所做出的“约束”，只是为了约束变量“看起来”的样子。也就是说，如果我们用一种“不正确”的方式去访问变量的内容，我们得到的结果一定会匪夷所思。例如我们把一个存储着int的变量，用float的方式去打开，得到的一定是一个“看起来很像浮点数”的家伙，而我们自己也不明白这个数究竟代表着啥，因为它只有在以“int”的方式被访问的时候我们才能看得懂。

类似的，假如有一个char类型的变量，其中存储着字符‘A’。如果我们用字符的方式去访问，则它一定输出‘A’这个字符。如果我们用整数的方式去访问它，得到的结果会是65。

通常来说，只要我们“观察”变量的方式对了，就不会出现太大的问题。甚至有些时候我们需要用特殊的方式进行特殊的操作，例如我们想要知道‘A’的ASCII码是多少，只需要以int的方式去访问一个存储着‘A’的变量。就能返回它在内存中的真实值的十进制形式，也就是‘A’的ASCII码。

1. 运算与表达式

* 啥是运算？啥是表达式？

作为一门计算机编程语言，用到最多的就是形形色色的运算了。例如数学上的加减乘除，逻辑上的与或非，甚至是位运算。这些形形色色的运算往往有两个核心：操作类型和操作数。

操作类型就是指要进行怎样的运算，例如加、与、右移运算等。操作数则代表着将要施加运算操作的数据。例如整数、浮点数等，它可以是变量（如 int a），也可以是常量，如“x”。

不同的种类的操作数，能够进行的操作类型可能有所不同。而用来描述一种运算的式子，就是表达式。这里的表达式与数学上的表达式有着非常相似的性质，例如“表达式总是有值的”。不过也有不同的地方，例如“表达式的值不一定直接就是数字”。所谓“表达式总是有值的”是指，所有的表达式均应该能被计算出一个确切的值，并且可以通过“赋值”等方式将其值赋予给某个变量。而被赋值的变量，也可以看做是一个“仅仅由一个变量，没有运算符组成的表达式”

我们将等号左边的值称为“左值”，右边的值称为“右值”。表达式的左值右值都应该是表达式。简单的表达式可能只有一个运算符和两个操作数。复杂的表达式可能由很多简单的表达式组成。

如果非要给表达式来分类的话（其实我一直认为表达式没有类别的区别），可以分成：算术表达式、逻辑表达式、关系表达式、赋值表达式等等。

所谓算数表达式就是我们最常见的数学表达式，可乘除可加减，并且有自己的值。

而所谓的逻辑表达式则是大家以前不太常见的一种。逻辑表达式对应的运算则是逻辑运算。逻辑表达式的值称为布尔值。布尔值就是“真”或者“假”。非真即假，非true即false。逻辑运算包括“与运算”、“或运算”、“非运算”。运算符分别用“&&”、“||”、“！”表示。他们的运算规则如下。

与运算：A与B必须在两个都为真时，表达式值为真，否则都为假。

或运算：A或B只要有一个为真时，表达式的值为真，否则都为假。

非运算：就是取反，如果A为真，则非A为假。反之亦然。

逻辑表达式是两个逻辑值经过逻辑运算得到另一个布尔值。

还有一种表达式，被称为关系表达式。它则是两个可比较的类型通过关系运算符运算，得到布尔值。整型、浮点型甚至是字符串一般都是可比较的。而关系运算符则是用来表明两操作数之间的关系的。例如大于、大于等于、小于、小于等于、等于、不等于。分别用“>”、“>=”、“<”、“<=”、“==”、“!=”表示。其中“等于”是两个等号，为的是与赋值时用的一个等号所区别。

赋值表达式则是一种用来操作变量值的表达式。当其他表达式运算完毕，得到一个确切的结果（可能是算数值，也可能是布尔值），并将这儿结果作为右值。用一个等号连接一个左值变量。则右值将会被传递给左值。

例如a=1+2\*3。程序首先运算1+2\*3=7，再将“7”作为右值、“a”作为左值，将7的值赋予变量a。

有一种特殊的表达式，形式如同“a+=b”它的实际含义是a=a+b。同时还有“-=”、“\*=”、“/=”等等。

还有一种形如“i++”和“++i”的表达式。前者是“先取i的值”，再“i+=1”；后者是先“i+=1”再“取i的值”。与之相对的有“i--”和“--i”。

* 混合类型表达式的值

上面我曾经说过，我曾经一直觉得表达式没有什么不同的类型，所有的表达式都是类似的。这是因为表达式“宏观”地表现为“有值、能运算”等特性。对于单一类型的表达式运算，例如1+2来说，值就是3。但是对于1+true来说呢？它的值是多少？其实，布尔值在内存中是以“1字节”的存储空间表示的，true是00000001，false是00000000。当1+true，实际上就是1+1。1+false则是1+0。反过来说，0表示false，任何非0的数都表示true。这样的话我们得到的结果又是什么类型的呢？这要看你用什么方式去存储了。例如bool a = 1+true：虽然1+true的结果是2，但是由于它是非0的数，所以系统会把它当做“true”，然后是bool a = true，这时a的值就是true，而且在内存中存储的就是00000001，而不是00000002。这种混合了算数与逻辑的运算不是我们所提倡的，甚至在其他一些语言中是明令禁止的！但是在C++中我们可以利用这个特性实现一些颇具技巧性的代码。

* 表达式的运算顺序

讨论运算顺序是一件非常痛苦的事情，因为运算符太多了、规则太多了。C++的算数表达式满足一些基本的“先乘除后加减”的数学规则。先哲们把这种规则成为“运算优先级”。乘运算的优先级比加运算要高！那遇到1+2\*3的情况就要先运算2\*3后运算1+。同样的，如同数学运算中的括号，C++也是有小括号的，它们拥有最高的优先级，例如(1+2)\*3=9，而且括号可以嵌套，例如((1+2)+3)\*4=24。

上面提到的仅仅是简单的算术表达式。这种表达式的运算我们已经学过了十多年。但是当出现多种不同类型的表达式进行混合运算的时候就会非常头大了，例如：1+true&&2\*3+2这种奇葩的表达式。而且如果运算中的运算符太多了，优先级就会变得非常混乱，例如： a&&b||c&&!d。我不主张写写出这样的代码（但是这是考试会考到的一个重要方面）。因为这种写法会由于运算符优先级不明确导致各种“主观错误”。所以当你还不熟悉运算优先级的时候，我建议多使用“括号”的方式。例如上面的式子改写为：(a&&b)||(c&&(!d))。

* 各种类型的各种转换

我们在前面曾经讨论过数据类型的问题。我们知道，float类型的1.0与int类型的1，在内存中的表示是完全不同的。假设我们有一个int类型的变量，让它等于1.1会发生什么（int a = 1.1）？由于int类型“先天不足”，它是无法保存小数点后面的内容，所以系统就会自动将小数点去掉（注意是去掉，不是四舍五入）。然后再赋值到int类型的变量中。它的具体过程是这样的：

首先，将float类型常量“1.1”转换成int类型的常量“1”。然后将这个常量赋值到int类型的变量“a”中。这种类型转换是强制的。为什么叫做强制的？这是因为系统强制将1.1变成1，导致精度的损失，而C++在设计之初就想尽量避免精度损失等问题。同时这种转换还是“隐式的”，因为它在转换之前没有什么明文的告知，编译器也不会进行太多的检查。所以这中转换就叫做“隐式强制类型转换”。

既然有隐式的，就一定会有“显式”的。所谓显示类型转换，就是在你转换之前，显式地告知要转换成为的类型。例如有这样的代码：

程序 2-5: 强制类型转换

00 float a = 1.2;

01 float b = 2.4;

02 float c = a + (int)b

03 cout <<c<<endl;

大家可以想象一下，上面的代码输出为3.2。但为什么不是3.6？这是因为在运算时，b的值先从2.4变成了2，然后加上a=1.2，最后就变成了3.2了。

既然类型转换有强制的，那也就有非强制的。非强制的类型转换叫做“隐式类型转换”。大家看上面的代码的第02行。其中有一个a+(int)b的地方。显然，(int)b的类型应该是int类型。而a的类型是float类型，那么这两个类型的数据相加，会发生什么情况？如果发生了自动的转换，则应该将a转换成int类型？还是要将(int)b转换成float类型？我们前面说到过，C++是在设计之初就考虑到尽量避免数据精度的损失。如果说将a转换成int，则运算很有可能会将小数部分丢弃，造成精度损失。所以应该是将(int)b转换成float再与a相加。这种转换叫做隐式类型转换。何种转换时自动发生的，而且这种自动发生的转换也一定是隐式的，因为显示的转换只能由程序员主动发起。

从int到float，保证精度不损失，其实是有一套转换规则的：“低类型”的操作数向“高类型”的操作数转换。具体规则如下：

int🡪unsigned int🡪long🡪unsigned long🡪float🡪double

规则似乎很麻烦，但只要记住“隐式类型转换要尽量保证不丢失精度”！

1. 语句

语句是构成程序的基本单元。通常，一个简单的语句表示一种单一的操作。复杂的操作则由一系列语句组成，包括复合语句、选择语句、循环语句等等。在C++中，语句以分号作为结束标志。程序在书写时，通常每行为一条语句。当语句过长时，可以通过一些方法拆分成多行。多个简单的短语句也可以写在同一行以节省空间。

多条语句可以通过大括号括起来成为一个整体，这种特性在后面讲到的选择和循环结构中经常用来划分“循环体”等部分。例如有语句：

程序2-6 语句块

00 a = 10

01 b = a+10;

02 {

03 d = b+10;

04 e = d+10;

05 }

上面的代码中，00和01行都是单条的语句，02~05行表示语句块。

通常来说，语句是逐行逐条执行的。但是这种基本的顺序结构很难满足我们对复杂问题的刻画。例如我们要计算一个分段函数值，或者计算从1累加到100的值。这些问题如果用顺序结构都是很难简单地实现的，所以在C++中制定了如下的程序结构：顺序结构、选择结构、循环结构。

* 顺序结构

当物理上相邻的多条语句在执行时，按照其物理排列的先后顺序逐条进行，这样的一组语句放在一起便称为是控制结构。

* 选择结构

选择结构即分支结构，是一种“依据条件”，“执行动作”的结构。例如通常的分段函数：这就是一种“选择性的表示”。在C++中我们利用if语句来表示选择结构：

程序2-7 选择结构

1. if(express)
2. {do something;}
3. else if
4. {do something;}
5. else
6. {do something else;}

上面的代码中描述了if语句表示的选择结构。这种选择结构叫做“条件选择”

* 循环结构

1. 小结、习题etc

习题1.查阅资料解释什么是目标文件？目标文件是什么工具产生的？内部应该有哪些内容？

习题2.什么是#include <iostream>？“std::”这种形式表示什么？

习题3.指出下列代码中都进行了哪些“隐式强制类型转换”，哪些“显示强制类型转换”和“隐式自动类型转换”：

float a = 1.2;

float b = 2.4;

float c = a+(int)b;

int d = a+(int)b

习题4.执行下面程序，理解i++与++i的区别。

#include <iostream>

using namespace std

int main(){

int a=1,b=1;

cout<<a++<<” ”<<b++<<endl;

cout<<a<<” ”<<b<<endl;

}

习题4.

1. s