COLD CODE-

突然好想学编程

C++入门与实战

苏惟澄　著

前言：

首先感谢您选择了这本书。

本书主要面向零编程基础者，为 C++语言的基础与实战，适合中学生阅读。

本书将不会作为商业使用。

欢迎加入Cold Code 计算机俱乐部QQ 群：729158255

联 系 作 者 苏 惟澄 ：

Email:2507287078@qq.com

Tel:15996539053

QQ:2507287078

目录

暂未完工

一. Hello World

好的，不管你能否看懂，还请先浏览一下这串代码

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
	cout<<"Hello World";
	return 0;
}
```

可能各位要问了：诶，苏惟澄，你不是说零基础开始学习吗， 怎么一跑上来就让我们看代码？

其实我并不是违背“零基础”，如果我们能够先浏览一遍我们等会要学习的代码，在接下来我们详细学习的时候，就能及时想起，方便理解。

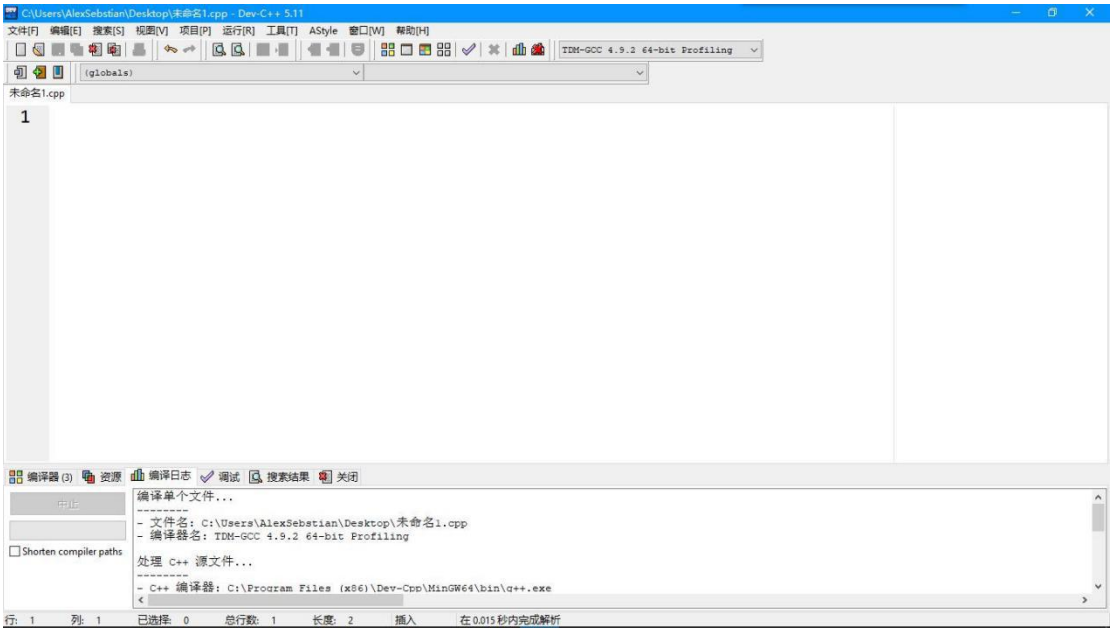
那么废话不多说，我们开始吧。

在开始学习代码之前，我们要先学习编译器的使用。

目前大部分的编程软件都集合了代码编辑器和编译器，这将方便我们的使用。

在本书中，我们将会用Dev C++来进行学习。

首先我们打开Dev C++的窗口



在中间，我们可以编辑我们的代码，随后可以点击上面的按钮进行编译

好的，你已经可以在Dev C++中编译你的第一个程序了。

等等，这本书好像不叫《Dev C++使用手册》

那么Dev C++的使用就讲到这，接下来，我们正式开始学习代码。

好的，现在我给你三个箱子。

第一个箱子是个工具箱，里面装着螺丝刀扳手等。

第二个箱子是急救箱，里面有绷带消毒水等。

第三个箱子是园艺工具箱，里面有小铲子喷壶等。

那么，我再给你布置三个任务：

第一个，处理一个伤口。

第二个，给园子里的花草松松土。

第三个，修一下门把手。

现在你应该理解了，我们的第一句代码的作用就是给计算机“箱子”。当计算机拿到了“箱子”里的东西，就可以完成你布置的任务。

我们的第一个“箱子”的名称是“bits/stdc++.h”，是一个“万能箱子”。当然箱子还有很多例如“iostream” “cstdlib” “Windows.h”等等，作为初学者，我们只需要使用万能的“bits/stdc++.h”即可。

至于第二句，我们还暂时不需要了解，但建议大家不要省略掉，这个在之后的学习中我们会慢慢了解。

第三句作用是设置主函数。主函数是运行程序的关键。要记住所有其他的程序代码都要包在主函数的大括弧里。

第四句是一句输出，输出的内容为“Hello World”。在C++中，我们常用“cout”来输出，格式为 cout<<。在“<<”之

后，我们常加要输出的变量或者是要输出的语句，但请注意，单独输出语句要用引号包住。如果需要换行，输出<<endl;即可

顺带提一下：代码中所有符号

都要使用英文符号!每条程序语句结束后都需要加上结束符号“;”即分号!

牢牢记住，不然有些时候编译失败原因要找半天。

最后一句看似不起眼，其实是与第一句一样重要的。目的是结束整个程序。

练习：

独自编写本章的“Hello World”程序

二．1+1=?

行，我知道 1+1 等于 2，但谁知道这个计算机会不会算呢?

不妨命令它算一下!

首先我们需要了解一下什么是“变量”。

如果我给你两个杯子，半杯量的水和一定的速溶果汁粉，现在要你冲泡出一杯果汁，该怎么做?

我们有三种办法:

将两个杯子分别装入水与果汁粉，将果汁粉导入水杯或将水倒入果汁粉杯。

将一个杯子中放入水，加入果汁粉或将果汁粉放入杯中加入水。

将水与果汁冲泡好后直接放入一个杯子。

那么写程序也是一样。其中，杯子就是我们所说的“变量”，“水”或“果汁粉”就是两个加数。

我们将数值等放入变量中叫做赋值。

赋值操作很简单，代码就是：变量=相应的数值;

记住，赋值操作用远是等号右边的赋给等号左边的。

那么我们先来定义两个变量。

还记得之前提到的“定义主函数”吗，前面是不是有个“int”？

int 的作用是设置整形变量。也就是说，这里面的数只能为整数。

```
int a;
int b;
```

好了，我们这样就得到了a 与b 两个变量

当然你也可以合成一句

```
int a,b;
```

两个不同变量之间逗号隔开。

我们不妨先用第一种方法，将a 与b 中分别存放数值 1。

完成后代码如下

```
a=1;
b=1;
```

好了，我们现在就可以将a 加入b 了。代码很简单：

```
a=a+b;
```

嗯，我们已经将 b 的数值加进了 a。但此时 b 中的数值没有发生任何改变。

或许有同学已经看不下去了：“苏某人，这个等式不成立啊！”

嘿，我们现在在学编程而不是数学。而且，我们现在所进行的是赋值操作而并非判等。而且，在 C++中，自己可以给自己赋值。这条代码是首先计算等号右边的数值，随后赋值等号左边。

现在我们来输出a 变量。

```
cout<<a;
```

在输出变量的时候，是不需要引号的。

所以程序整体如下：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
	int a,b;
	a=1;
	b=1;
	a=a+b;
	cout<<a;
	return 0;
}
```

那么我们用第二种方法，就是将原来的 a 变量赋值为 1，加上 1 后重新赋值给自己。第三种就是直接将 a 赋值为 1+1。

当然，做其他运算的话，改符号即可，并且

那么如果我们想自己输入数据呢?

我们需要了解一下输入：cin>>;

和“cout”的使用方法一样，只不过是尖括号方向反了

用 cin>>a;便是输入一个数值，然后赋值给a 变量。

练习：

1. 将其他两种方法做出来。
2. 将 a，b 变量改为自己输入。

三．1+1 是不是等于 2?

在让计算机算完 1+1 后，我们让计算机来判断 1+1 是不是等于 2。

我们先写一个比较简单的 1+1 计算程序。

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
	int a,b;
	a=1;
	b=1;
	a=1+1;
	cout<<a;
	return 0;
}
```

我们先来想想该怎么判断。

首先，我们获取 1+1 的计算出的值。随后将其与 2 比较，如果等于 2，输出：是，否则输出“不是”。

那么现在，我们就要用到“if”了。

if 的用法如下：当条件满足，运行括号内程序：

if(条件)

{

程序语句

}

或者满足条件，运行相应程序，不满足条件，运行else 下程序

if(条件)

{

程序语句

}

else

{

程序语句

}

那么我们的 1+1 是否等于 2，就这么写

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
	int a,b;
	a=1;
	a=1+1;
	if(a==2)
	{
		cout<<"是";
	}
```

```
else
{
        cout<<"不是";
}
return 0;
```

其实也很好写嘛。

输出结果就是“是”，不必多说。

练习：

输入两个数进行加法运算，判断是否等于 2，如果是的话，输出“Yes”否则输出“No”。

示例：

输入 1 5

输出 No

四. 好的代码习惯

其实这一章本来我不想写，但是一个良好的代码习惯是真的很重要。

我本人在学习时，常常会看到有些同学的代码杂乱，无序。虽然这样在编译时几乎不会有任何的错误，但是在我们回头检查的时候会变得十分困难。

例如下面两串代码，虽然在计算机眼中是完全一样的，但是哪个我们看的舒服？

第一串：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
                    int a,b;
          a=1,b=1
a=a+b;
b=
a;

cout<< a<<b;
return 0;
 }
```

第二串：

```
#include <bits/stdc++.h>
using namespace std; 3.
int main()
{
        int a,b;
        a=1,
        b=1
        a=a+b;
        b=a;
        cout<< a<<b;
        return 0;
 }
```

现在，我明确地告诉你，第二串代码是第一串整理完后的。而这串代码中，有一处错误。

第一串或许你很难找到，而第二串就容易许多。

请看第二串的第七行和第八行，分别是将结束符加成了逗号和没加结束符。

由此可见，代码格式的标准是如此重要。

基本上我们就这样整理：

将头处与函数中用一个空行隔开

大括号包装函数中，每条语句首行缩一个tab

在编程中，还有一个不可少的就是注释。

试想一下，你正在完成之前的 1+1 程序，你的代码已经写了这么多：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
        int a,b;
        a=1;
        b=1;

 }
```

但是此时妈妈跑过来让你跑腿，虽然你回家时可能会再想起要怎么继续写，但当我们去完成那种 60 多行的超复杂程序，这个时候你可能就要推翻重来了。那可真的是难受。

不过有一个东西叫注释，使用方法很简单，就是两条斜杠“//”。

例如这样：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
        int a,b;
        a=1;
        b=1;
        //这里要把 a 与 b 加起来
 }
```

但是这个注释只能注释一行，所以这样是错误的：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
        int a,b;
        a=1;
        b=1;
        //这里要把 a 与 b 加起来
        这行这样直接写的话就不算注释了哦
 }
```

那如果要注释多行呢？我们可以用/* 注释内容 */来进行

例如：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
        int a,b;
        a=1;
        b=1;
        /* 这里要把 a 与 b 加起来
哦
        啊对了 别忘了要输出 */
 }
```

这样就是一个完美的多行注释。

在编译的时候编译器可以毫不留情地忽略掉。

所以这样也行：

```
1. int a//我就加这你能拿我咋样？？
2.  ,b;
```

总之只要程序语句不在注释范围内就行。

练习：

将之前写的 1+1=2？判断程序每条代码详细的注释出其用途。

五．复读机复读机复读机

计算机，或许是这个世界上最任劳任怨的家伙。

计算机有个特点，它可以多次且快速地计算。利用这两个特点，人们便开始使用循环来处理一些东西。

首先，我们先看一下这个例子：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
	while(1!=0)
	{
		cout<<"咕";
	}
	return 0;
}
```

运行后的结果，就是输出了无数个“咕”。

~~看来这个程序是UP主必备啊。~~

那么，我们将主程序拆开来，我们会发现，主函数里面好像多了个什么东西。

这个while()，就是登场的第一个角色。

在英语当中，while就可以解释为当...时。在程序当中，也不难理解，也就是当括号中的判断成立时，就循环大括号中的程序语句。

例如在这里，我所设定的判断是：当1不等于0时。

1不等于0是永远成立的事实，所以程序会永远地运行下去，直到程序被终止或者遇到了这个语句：break;

也就是说，之前的写法会输出无数个“咕”，而这样写只会有一个“咕”：

```
while(1!=0)
{
        cout<<"咕";
        break;
}
```

这里的话要补充一个内容，就是在括号中，我们的判断要怎么写。

先来认识这几个符号：

== != >= <= > < && ||

== 这个是判等，判断左右两边的玩意的值是否一样

!= 这个是判断是否不一样

>= 判断左边的玩意是否大于等于右边的

<=和上面的小可爱相反，右边是否大于等于左边

>左边是否大于右边

<右边是否大于左边

&& 用于连接下一个判断，例如 (a>b && b>c)，也就是当a>b与b>c同时成立时才算符合条件。

|| 这个表示“或”，就是连接着的判断只要一个符合了就算符合条件。

while循环还有一个兄弟，叫do while。

它的写法是这样的：

```
do
{
        /*代码*/
}
while();
```

那么上面的while程序就可以写成这样。

```
do
{
        cout<<"咕";
}
while(1!=0);
```

编译一下，发现效果差不多啊。

但在实际应用中，这两个看起来一样的while用起来可是不一样的。

就这么说，while()是“先分析局势再出兵”，do while是“先出兵再分析局势”。也就是说，while()先会判断括号内是否符合条件，如果符合，再执行大括号中内容，不符合就跳过去，每一次循环都判断一次。

但是do while呢。

do while则是先执行do下大括号中的内容，再进行while()中的判断。开个玩笑，当你大喊“刀下留人”的时候，人头已经落地

了。就那种效果。

还有一种十分重要的循环方式就是for()

for的话一开始我们也会觉得没啥用，但是当我们学到数组那一阶段后，我们会发现这可是一非常实用的。

来看一下大概结构

```
for(变量;条件;加减)
```

大概就是这样，举个例子吧

for(int i=1;i<=10;i++)

仔细看括号里的内容：

我们先定义了一个值为1的变量i，我们要求i小于等于10，每循环一次，i自加一次。

注：这里的i++后门的”==“表示”自加“，也就是自己给自己+1。

所以我们看一下i总共会有几个值

1 2 3 4 5 6 7 8 9 10

也就是说我们总共会循环10次

如果我们一开始赋的值是5的话

i总共会有这些值

5 6 7 8 9 10

总共六次

那么像这样呢?

for(int i=10;i>=0;i--)

这个时候就会有这些值

10 9 8 7 6 5 4 3 2 1 0

11个

循环11次。

但是如果在进行这个循环之前，我们已经有了一个变量i=1；可不可以直接用呢?

答案是可以的，

for(i;i<=10;i++)

这个时候依然是有十个值

但是运行完后，我们下面再次调用该变量，i的值就是10了。

如果你的变量是在for循环的小括号里面定义的，在下面大括号中是可以调用的。但是当我们跳出了这个循环，这个i变量就不存在了。可以说这个i是一次性的。

练习：

1 听话的电子狗

Benjamin造了一只机器狗，这只机器狗可乖了，你叫它叫几声它就叫几声。但是Benjamin有点笨，这个程序他写不出来，你能帮他写吗?

输入：一个数字n，0<n<20

输出：n个“wang”

示例：

输入：3

输出：wangwangwang

请用while do-while 和 for各做一次。

2 我真的很棒?

林沐秋同学最近学业上获得了比较好的成就，但她不觉得自己很棒，于是她打算询问

别人，如果别人认为她真的很棒那么就输出n次“I’ m so good!”，如果别人不这么认为，就输出一次“ok”

输入：两个数a与n，其中a=0或1，0表示否定，1表示认同，0表示否定，0<n<10

输出：相应的答案（每一次I’ m so good!直间用空格隔开）

示例

输入：1 3

输出：I’ m so good! I’ m so good! I’ m so good!

输入：0 3

输出：ok

六．一家人就要整整齐齐

教室里面老师总喜欢把学生分成一大排一大排，这样子方便检查作业嘛。”第一大排把作业送上来！“于是第一大排每个同学就很乖地一个一个拿着本子上去了，直接省去了老师一个一个点名的繁琐步骤。

在编程里面，如果要在程序里面记录两个不同数值，我们可以设两个变量a和b，但如果有五十个数值，26个字母都不够用了！

这个时候，我们就需要用到一种玩意叫数组了。

数组长这样:

a[5]

也就是一个变量，后门跟一个大括号，大括号里放个数字。

大括号里的数字我们叫做下标。

就这么说吧，来到旅馆里，走廊上房间貌似是这样分布的：

0号房	1号房	2号房	3号房	4号房	5号房

排成一排。

如果这个旅馆的名字叫a，那么我们可以这样给他编名字

旅馆a的0号房	旅馆a的1号房	旅馆a的2号房	旅馆a的3号房	旅馆a的4号房	旅馆a的5号房

登记房间碰到个像苏惟澄一样懒的家伙，估计就会写成这样了：

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

但是在程序当中，这不是懒，就是这么写的。

所以我们执行一个int a[5];

我们就相当等于是~~开了6个房。~~

不对啊怎么是6个?
因为在C++中，我们的下表是从0开始的。
所以我们int a[5];
就会生成像上面表一样的6个变量。
那么我们如何调用它呢?
很简单，就把它当普通变量来用。
例如我要在a[1]里面记录一个数值1
就这样
a[1]=1;
好了。
那么现在，我们要在这6个变量中一个个人手动输入，
搞六个输入语句，您在开玩笑？？？
还记得之前的for循环不？使用它的时间到了。
数组a[5]总共6个，那么我们就要让for循环运行6次。
我的计划是设置变量i=0;i<=5;i++
写进for循环大概就是这样

```
for(int i=0;i<=5;i++)
{
        cin>>a[i];
}
```

可能又要有不理解的地方了。
这个a[i]是什么意思呢?
其实在方括号里面我们也可以加入变量名称
在这个for循环里面我们有变量i，可以直接引用。
第一次循环，我们i的数值是0，所以底下的cin调用a[i]即a[0]
第二次，i变成了1，所以调用的是a[1]
以此类推。

练习：

1 输入又输出

Benjamin是真的没事干，他想设计一个程序，他输入n个数字，程序就输出n个数字。他还不会写！要你来帮他

输入：n与n个数，其中0<n<10，每个数大于0且小于20

输出：n个数

示例

输入： 5

1 5 2 7 3

输出：1 5 2 7 3