

# C++入门

<http://hwdong.com>

# 1. 程序 = 数据+运算

## Application = Data + Operator

- 程序就是对 **数据** 进行 **加工处理(运算)**。  
程序 = 数据 + 运算
- 数据用变量(常量)表示，用运算符对它们进行运算  
程序 = 变量 + 运算符
- 程序中的数据应有条理地存储在内存中，处理需  
要按照一定步骤或方法进行。  
程序 = 数据结构 + 算法

# 机器语言

- 电子元件'开'和'通'是两个稳定状态，可用以表示1或0.
- 一组0和1二进制串可用来表示数据和运算符(操作码)。

机器指令=操作码+地址码

**操作码：**加、减、乘、除、移动等

**地址码：**第1、2操作数地址、结果地址

# 机器语言/ machine language

机器指令=操作码+地址码/操作数

如对于 [x86/IA-32](#) 处理器，下列指令将一个8位的值移到一个寄存器中

10110000 01100001

其中操作码1011后跟一个3位寄存器AL的标识码000，数据则是01100001

- 指令部份的示例

1. 0000 代表 加载 (LOAD)
2. 0001 代表 存储 (STORE)

...

- 暂存器部份的示例

1. 0000 代表暂存器 A
2. 0001 代表暂存器 B

...

- 存储器部份的示例

1. 000000000000 代表地址为 0 的存储器
2. 000000000001 代表地址为 1 的存储器
3. 000000010000 代表地址为 16 的存储器
4. 100000000000 代表地址为  $2^{11}$  的存储器

- 集成示例

1. 0000, 0000, 0000000010000 代表 LOAD A, 16
2. 0000, 0001, 0000000000001 代表 LOAD B, 1
3. 0001, 0001, 0000000010000 代表 STORE B, 16
4. 0001, 0001, 0000000000001 代表 STORE B, 1

# 汇编语言 / Assembly Language

- 汇编语言是机器语言的助记符.

10110000 01100001

`MOV AL, 61h` ; Load AL with 97 decimal (61 hex)

再如:

`MOV EAX, [EBX]` ; 将内存地址EBX的4字节move到寄存器EAX  
`MOV [ESI+EAX], CL` ; 将寄存器CL的值 move到地址为ESI+EAX的内存中

- 汇编器（ **assembler** ）将汇编语言代码翻译成机器语言代码的工具.

# 高级语言

- 以人们容易理解的接近自然语言的语言表示运算数和运算符。如用字母 $x, y$ 表示变量，用 $x*y$ 表示数学运算。
- 优点很多：易读性、易维护、移植性好、开发效率高等...
- 常用高级语言：C/C++、Java、Fortran、Matlab、Basic、PHP、HTML...

# 编译器Compiler

- 将高级语言程序转换成机器语言程序。

C++程序代码

```
int a, b, sum;  
  
cin >> a;  
cin >> b;  
  
sum = a + b;  
cout << sum << endl;
```

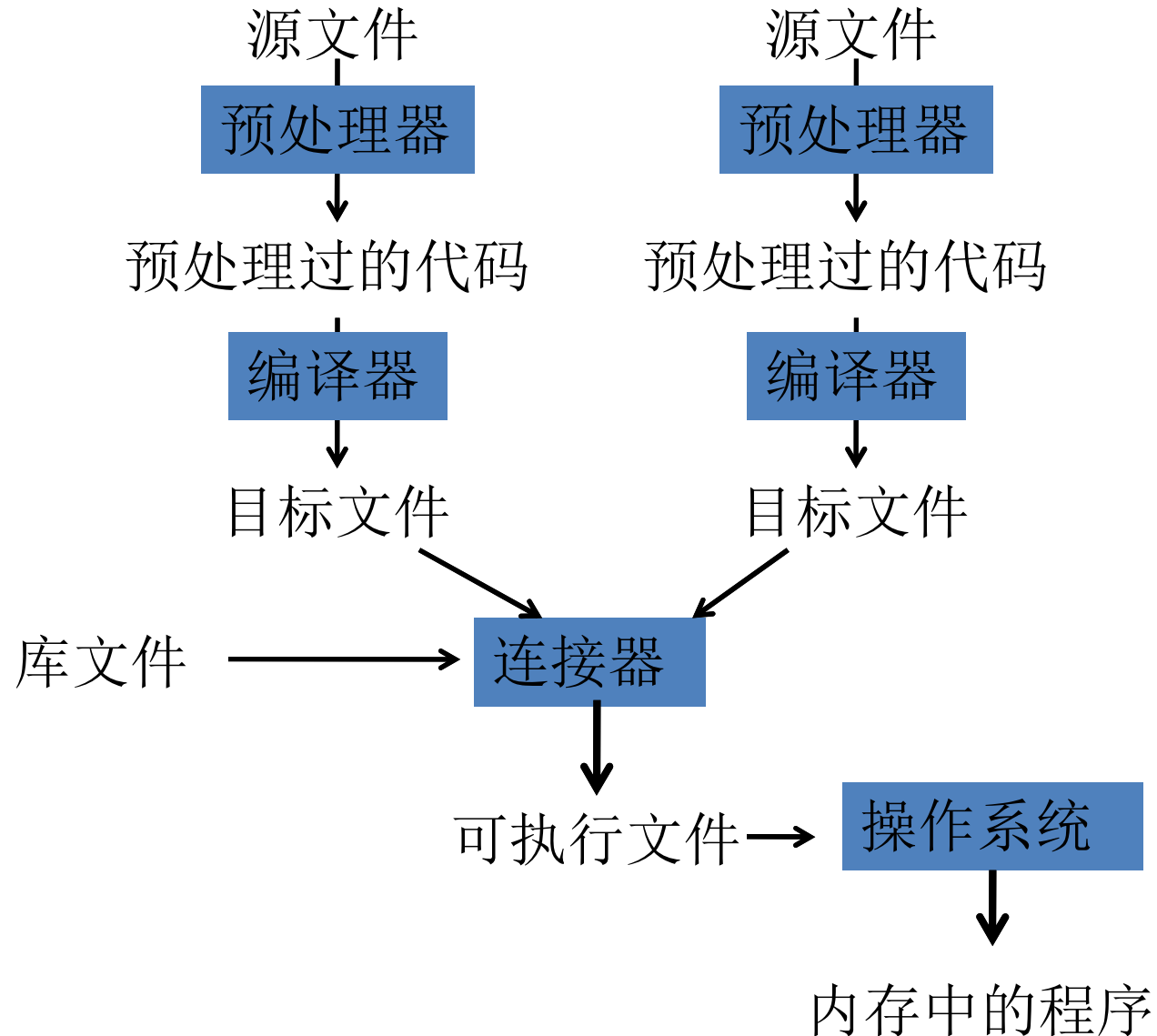
机器

00000	10011110
00001	11110100
00010	10011110
00011	11010100
00100	10111111
00101	00000000

编译器

CodeBlock  
Visual Studio

# 程序的编译链接





# C++ 简短总结

- C++非常流行，特别是对速度有要求、需要访问底层特征的应用程序场合。
- Bjarne Stroustrup（Texas A&M）在1979年发明，对C语言进行了扩展。本课程将先介绍C++中的C语言部分。
- C++程序是大小写敏感的，比如“someName”和“SomeName”是两个不同的标识符。
- 尽管可编写图形程序，但图形程序复杂且难移植，本课程将介绍文本程序(Console Application)。

# Hello world!

```
// my first program in C++  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
}
```

Hello World!

# Hello world!

```
// my first program in C++  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
}
```

双//开头的行是注释，用于解释代码，帮助自己和别人更好地理解程序，注释不是程序代码

Hello World!

# Hello world!

```
// my first program in C++  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
}
```

#开头的是预处理命令，**#include** <iostream>表示包含某个头文件 `iostream`，使得可以使用C++标准库的输入输出功能。

Hello World!

# Hello world!

```
// my first program in C++  
  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
}
```

空白行，对C++程序唔任何影响，仅仅是代码阅读性好！

Hello World!

# Hello world!

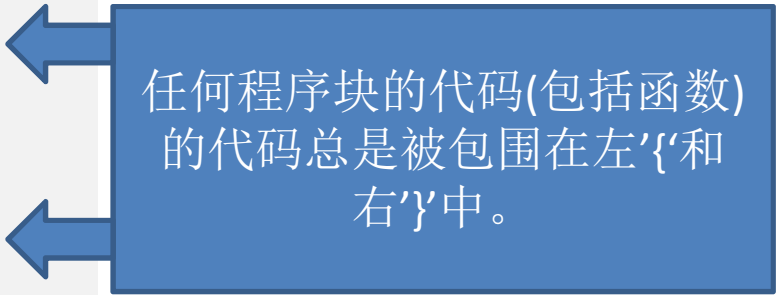
```
// my first program in C++  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
}
```

该行定义一个叫做main的函数，一个函数给一组代码语句起了一个名字。函数规范包括：类型(int)、名字(main)、圆括号(())。圆括号()内也许包含一些参数。每个C++程序都包含一个叫做main的函数，称为程序的主函数。程序总是执行这个主函数main。

Hello World!

# Hello world!

```
// my first program in C++  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
}
```



任何程序块的代码(包括函数)  
的代码总是被包围在左'{'和  
右'}'中。

Hello World!

# Hello world!

```
// my first program in C++  
  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
}
```

这是一行C++语句，一个语句是能实际产生某种结果的一个表达式。函数中的语句按照它们出现的程序依次执行。  
std::cout代表标准输出设备对象；<<是输出流运算符，用于向std::cout输出数据；双引号包围的是一个字符串。  
注意：每个c++语句都以；结果

Hello World!




# Hello world!

```
// my first program in C++  
  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
}
```

Hello World!

该程序只有一行可执行语句  
`std::cout << "Hello World!";`  
虽然我们用缩进换行等对代码进行排版，这仅仅是为了提高可读性，我们完全可以将代码写在一行中：



```
int main(){ std::cout << "Hello World!";}
```

# Hello world!

```
// my first program in C++  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello World!";  
    std::cout << "I'm a C++ program";  
}
```

Hello World! I'm a C++ program

# Hello world!

等价于

```
// my first program in C++
```

```
#include <iostream>
```

```
int main () { std::cout << " Hello World! "; std::cout << " I'm a  
C++ program "; }
```

```
Hello World! I'm a C++ program
```

# x+y

```
// x+y
#include <iostream>

void main()
{
    int x = 3;
    int y = 4;
    std::cout<<"x+y=:" <<x+y <<"\n";
}
```

x+y=:7

# 注释/Comment

- 以双斜杠 “//” 开头的行是注释行，仅仅解释代码的含义或作用。
- 以 “/\*” 开头，并以 “\*/” 结尾构成多行注释。

```
/*my second program  
   in C++  
*/  
#include <iostream>  
int main()  
{  
    std::cout << "Hello World!";  
    std::cout << "I'm a C++ program";  
}
```

Hello World! I'm a C++ program

# 使用名字空间Using namespace std

```
/*my third program   in C++ */  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    cout << "Hello World!";  
    cout << "I'm a C++ program";  
}
```

cout是定义在C++标准名字空间std中的一个对象，所以之前我们需要加一个限定词std::，说明cout是std中定义的对象。

我们也可以引入整个名字空间std,这样程序中cout前就不用再加名字空间限定词了。

Hello World! I'm a C++ program

# 转义字符

```
/*my third program   in C++ */  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    cout << "Hello World!\n";  
    cout << "I'm a C++ program";  
}
```

`\n`表示一个换行newline符号，`\`和一个文本常量称为转义字符，用以表示一个特殊字符。

```
Hello World!  
I'm a C++ program
```

# 转义字符

Escape Sequence	Represented Character
<code>\a</code>	System bell (beep sound)
<code>\b</code>	Backspace
<code>\f</code>	Formfeed (page break)
<code>\n</code>	Newline (line break)
<code>\r</code>	“Carriage return” (returns cursor to start of line)
<code>\t</code>	Tab
<code>\\</code>	Backslash
<code>\'</code>	Single quote character
<code>\"</code>	Double quote character
<code>\some integer x</code>	The character represented by <i>x</i>



# Tokens(记号)

- Tokens是对编译器有意义的最小的程序单元。

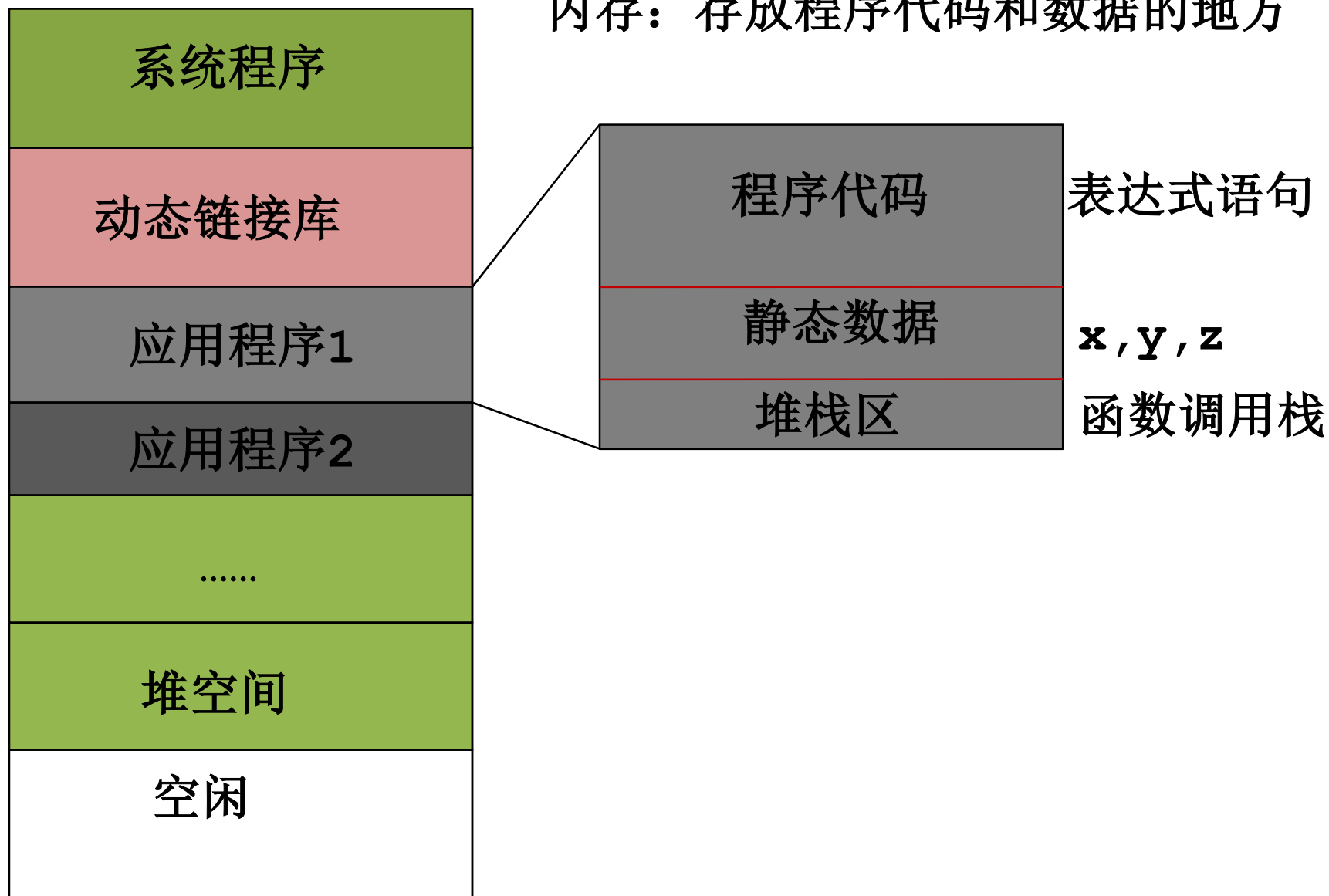
Token 类型	说明/目的	例子
Keywords(关键字)	对编译器有特殊意义的单词，语言保留的名字	int void using namespace
Identifiers(标识符)	不属于语言的名字	
Literals(文字量)	源代码中直接指定的基本常量	'c' '\n' 4 "x+y=: "
Operators(运算符)	数学或逻辑运算	+, -, &&, %, <<
Punctuation/Separators 标点/分隔符	定义程序的结构	{ } ( ) , ;
Whitespace(空格)	不同的空白符，被编译器忽略	回车(\r)、换行(\n)，空格以及制表符(Tab)

# 程序结构

- 语句(statement)是程序的基本构造块，程序就是由一系列语句构成的。
- 表达式(expression)是一个具有值(Value)的语句. 如  
40    x+y    “hello world\n”
- 运算符对 表达式进行运算，得到一个新表达式。如表达式(4+2)/3：对表达式4和2进行加法，得到的表达式再和表达式3进行除法。
- 运算符类型：  
数学Mathematical： +   -   \*   /   %  
逻辑Logical: &&   ||   ...  
位运算Bitwise: &   |   ...

# 程序内存布局

内存：存放程序代码和数据的地方



# 程序错误

- 语法错误：编译错误或链接错误  
编译器和连接器会告诉我们错误信息！

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

void main(){
    x = 3;
    cout<<x;
}
```

error C2065: 'x' : undeclared identifier

# 程序错误

- 语法错误：编译错误或链接错误  
编译器和连接器会告诉我们错误信息！

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

void main(){
    x = 3;
    cout<<x;
}
```

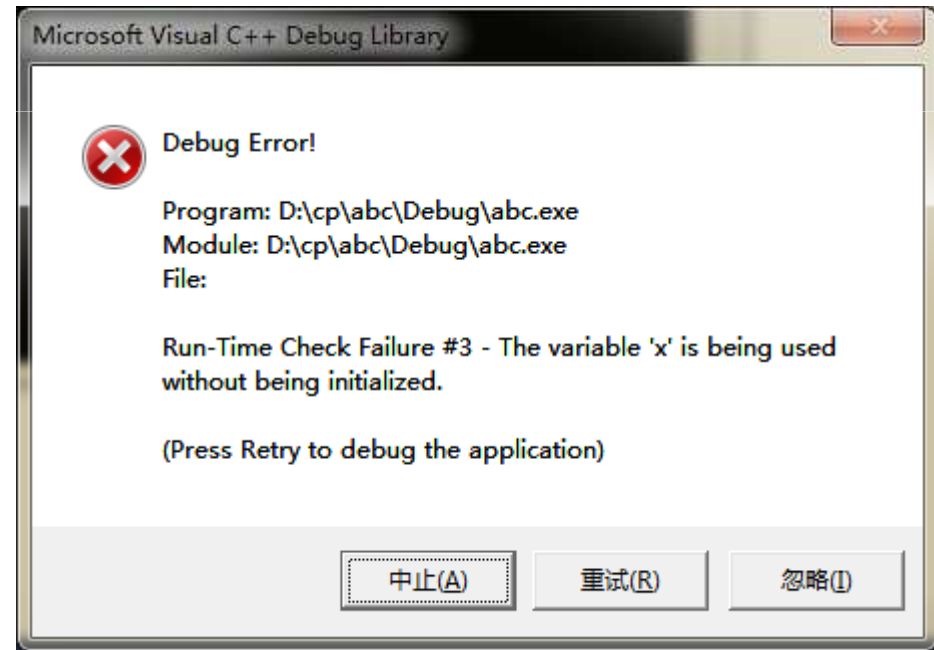
```
error C2065: 'x' : undeclared identifier
error C2065: 'cout' : undeclared identifier
```

# 程序错误

- 语法错误：编译错误或链接错误  
编译器和连接器会告诉我们错误信息！
- 逻辑错误：运行的结果和预想的不一致！

```
#include <iostream>
```

```
void main(){  
    int x, y =40;  
    int z = x+y;  
    std::cout<<z;  
}
```



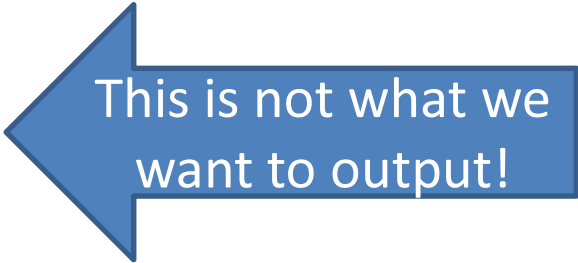
Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped

# 如何发现逻辑错误？

- 方法1：输出程序运行过程中的一些数据或信息。  
如printf 或 std::cout

```
#include <iostream>
void main(){
    std::cout<<"helo word!"
    std::cout<<"second line!"
}
```

Helo word! second line!



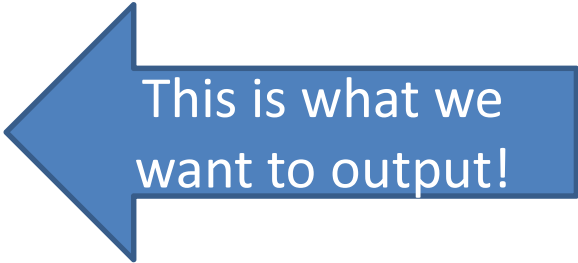
This is not what we  
want to output!

# 如何发现逻辑错误？

- 方法1：输出程序运行过程中的一些数据或信息。  
如printf 或 std::cout

```
#include <iostream>
void main(){
    std::cout<<"Hello Word!\n"
    std::cout<<"Second Line!"
}
```

Hello Word!  
Second Line!



This is what we  
want to output!



# 如何发现逻辑错误？

- 方法1：输出程序运行过程中的一些数据或信息。  
如printf 或 std::cout
- 方法2：利用IDE开发环境提供的调试功能，如断点调试、单步调试、进入函数...

F9	光标处设置断点
F5	调试执行程序(断点处暂停)
Ctrl+F7	编译程序
Ctrl+F5	执行程序
F10	单步执行
F11	进入函数执行

```
#include <iostream>
```

```
void main(){  
    int x, y =40;  
    int z = x+y;  
    std::cout<<z;  
}
```



```
#include <iostream>  
void main() {  
    int x, y =40;  
    int z = x+y;  
    std::cout<<z;  
}
```

Watch 1			
Name	Value	Type	
x	-858993460	int	
y	40	int	
z	-858993460	int	

# 数据及其类型

- 不同类型的数据占据的内存大小是不同的。可对它们进行的运算也是不同的。

```
void main() {  
    int a = 3, b=4;  
    std::cout << a+b;  
    char which = 'A';  
    bool k, h,r;  
    r = k*h;  
}
```

类型	存储大小	运算
int	32位	算术运算：+、-、*、/ 、 %
char	8位	
bool		逻辑运算：!、&&、 

- 每个表达式(数据)都具有一种类型：表示它的值是什么样的数据。如a, a+b是int型，k是bool类型，which是char类型。

# 数据类型

- 数据类型分为：基本数据类型、用户定义类型

Type Names	Description	Size	Range
char	Single text character or small integer. Indicated with single quotes ('a', '3').	1 byte	signed: -128 to 127 unsigned: 0 to 255
int	Larger integer.	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean (true/false). Indicated with the keywords <code>true</code> and <code>false</code> .	1 byte	Just <code>true</code> (1) or <code>false</code> (0).
double	"Doubly" precise floating point number.	8 bytes	+/- 1.7e +/- 308 ( 15 digits)

# 类型-基本类型

Group	Type names*	Notes on size / precision
Character types	<code>char</code>	Exactly one byte in size. At least 8 bits.
	<code>char16_t</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>char32_t</code>	Not smaller than <code>char16_t</code> . At least 32 bits.
	<code>wchar_t</code>	Can represent the largest supported character set.
Integer types (signed)	<code>signed char</code>	Same size as <code>char</code> . At least 8 bits.
	<code>signed short int</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>signed int</code>	Not smaller than <code>short</code> . At least 16 bits.
	<code>signed long int</code>	Not smaller than <code>int</code> . At least 32 bits.
	<code>signed long long int</code>	Not smaller than <code>long</code> . At least 64 bits.
Integer types (unsigned)	<code>unsigned char</code>	(same size as their signed counterparts)
	<code>unsigned short int</code>	
	<code>unsigned int</code>	
	<code>unsigned long int</code>	
	<code>unsigned long long int</code>	
Floating-point types	<code>float</code>	
	<code>double</code>	Precision not less than <code>float</code>
	<code>long double</code>	Precision not less than <code>double</code>
Boolean type	<code>bool</code>	
Void type	<code>void</code>	no storage
Null pointer	<code>decltype(nullptr)</code>	

# 类型-基本类型

- 类型的位越多能够表示的不同值越多

Size	Unique representable values	Notes
8-bit	256	$= 2^8$
16-bit	65 536	$= 2^{16}$
32-bit	4 292 967 296	$= 2^{32}$ (~4 billion)
64-bit	18 446 744 073 309 551 616	$= 2^{64}$ (~18 billion billion)

数据有变量和常量之分：

```
#include <iostream>
#define PI 3.1415926
void main(){
    float r = 3;
    r = 4.5;
    float circle = PI*r*r;
    std::cout<< circle;
}
```



宏定义：PI是常量3.1415926  
常量不可以被修改

3种声明常量的方式：

- 1) 文字量，如3.1415926
- 2) 宏定义常量，如PI
- 3) **const**关键字定义，如

```
const float PI = 3.1415926;
```

数据有变量和常量之分：

```
#include <iostream>
#define PI 3.1415926
void main(){
    float r = 3;
    r = 4.5;
    float circle = PI*r*r;
    std::cout<< circle;
}
```



r和circle都是浮点数类型float的变量, 变量可以被修改。



## 用户定义类型- ostream

- 在C++标准名字空间中定义了许多实用的类型，其中有一个用户定义的**输出流类型ostream**的变量**cout**. 可用于输出信息.
- 要使用std中的变量**cout**,需要包含相应的输入输出流头文件:**#include <iostream>**.

```
#include <iostream>
int main( )
{
    std::cout << "Hello world! \n";
}
```

# 用户定义类型-istream

- 在C++标准名字空间中有一个用户定义的输入流类型istream的变量**cin**. 可用于输入信息.
- 要使用std中的**cin**,需要包含相应的输入输出流头文件:**#include** <iostream>.

```
#include <iostream>
int main( )
{
    int x;  double y;
    std::cin>>x>>y;
    std::cout<<x+y <<"\n";
}
```

# 用户定义类型-string

- C++标准库中定义了一个非常实用的用户定义类型-字符串类型 **string**

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    string mystring;
    mystring = "This is a string";
    cout << mystring;
    return 0;
}
```

其他初始化方式:

```
string mystring = "This is a string";
string mystring ("This is a string");
string mystring {"This is a string"};
```

更多运算: 如+

```
string s = "This is a string";
string t ("This is a string");
string h = s+t;
std::cout<<h<<"\n";
```

# 用户定义类型-vector

- 标准模板库std中的vector是一个向量(数组)类型的模板。用于从一个参数类型生成一个用户定义类型。如 **vector<int>** 就是一个可存储多个整数的向量类型

```
#include <iostream>
```

```
#include <vector>
```

```
void main(){
```

```
    std::vector<int> ints;
```

```
    ints.push_back(3);
```

```
    ints.push_back(8);
```

```
    ints[0] = 2;
```

```
    std::cout<<ints[0]<<"\n"<<ints[1];
```

```
}
```

2

8

# 附录：运算符

- 算术运算符:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$
- 赋值运算符:  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$ ,  $>>=$ ,  $<<=$ ,  $\&=$ ,  $\wedge=$ ,  $|=$
- 增量运算符:  $++$ ,  $--$
- 比较运算符:  $==$ ,  $!=$ ,  $>$ ,  $<$ ,  $>=$ ,  $<=$
- 逻辑运算符:  $!$ ,  $\&\&$ ,  $||$
- 位运算符:  $\&$ ,  $|$ ,  $\wedge$ ,  $\sim$ ,  $<<$ ,  $>>$
- 条件运算符:  $?$
- 逗号运算符:
- **sizeof**运算符
- 显式类型转换运算符: **(type)**
- ...

# 算术运算符

- C++提供5个基本的算术运算符

**Arithmetic operators.**

Operator	Name	Example
+	Addition	12 + 4.9 // gives 16.9
-	Subtraction	3.98 - 4 // gives -0.02
*	Multiplication	2 * 3.4 // gives 6.8
/	Division	9 / 2.0 // gives 4.5
%	Remainder	13 % 3 // gives 1

# 关系运算符

- C++提供6个用于比较数值的关系运算符,返回值1或0。因此关系运算符不能用于比较字符串

“Hello” > “world”

## Relational operators.

Operator	Name	Example
==	Equality	5 == 5 // gives 1
!=	Inequality	5 != 5 // gives 0
<	Less Than	5 < 5.5 // gives 1
<=	Less Than or Equal	5 <= 5 // gives 1
>	Greater Than	5 > 5.5 // gives 0
>=	Greater Than or Equal	6.3 >= 5 // gives 1

# 运算-逻辑运算符

- 3个逻辑运算符用于比较逻辑表达式

Logical operators.

Operator	Name	Example
!	Logical Negation	!(5 == 5) // gives 0
&&	Logical And	5 < 6 && 6 < 6 // gives 1
	Logical Or	5 < 6    6 < 5 // gives 1

- 非0数表示逻辑值true，而0表示逻辑值false

```
!20 // gives 0
10 && 5 // gives 1
10 || 5.5 // gives 1
10 && 0 // gives 0
```



# 运算符-增量/减量运算符

```
int k=5;  k++; --k;
```

**Increment and decrement operators.**

Operator	Name	Example
++	Auto Increment (prefix)	++k + 10 // gives 16
++	Auto Increment (postfix)	k++ + 10 // gives 15
--	Auto Decrement (prefix)	--k + 10 // gives 14
--	Auto Decrement (postfix)	k-- + 10 // gives 15

# 运算-位运算符

- 6个位运算符用于处理整数中的单独的位

**Bitwise operators.**

Operator	Name	Example
~	Bitwise Negation	~'\011' // gives '\366'
&	Bitwise And	'\011' & '\027' // gives '\001'
	Bitwise Or	'\011'   '\027' // gives '\037'
^	Bitwise Exclusive Or	'\011' ^ '\027' // gives '\036'
<<	Bitwise Left Shift	'\011' << 2 // gives '\044'
>>	Bitwise Right Shift	'\011' >> 2 // gives '\002'

# 运算-位运算符

*unsigned char x = '011';*

*unsigned char y = '027';*

How the bits are calculated.

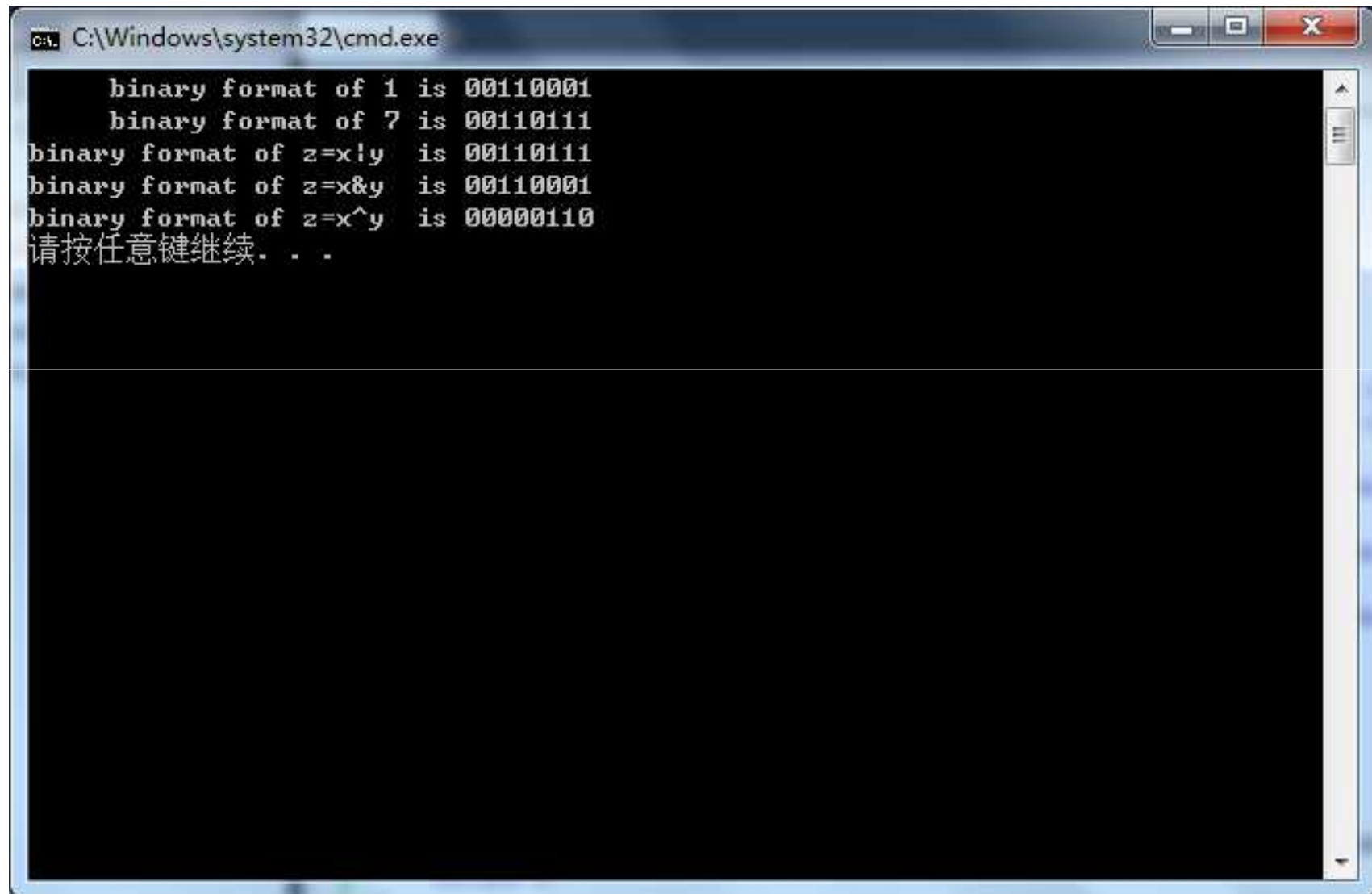
Example	Octal Value	Bit Sequence							
x	011	0	0	0	0	1	0	0	1
y	027	0	0	0	1	0	1	1	1
~x	366	1	1	1	1	0	1	1	0
x & y	001	0	0	0	0	0	0	0	1
x   y	037	0	0	0	1	1	1	1	1
x ^ y	036	0	0	0	1	1	1	1	0
x << 2	044	0	0	1	0	0	1	0	0
x >> 2	002	0	0	0	0	0	0	1	0

# 运算-位运算符

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

int main(){
    unsigned char x = '011';
    unsigned char y = '027';
    cout<<"    binary format of "<<x<<" is "<<bitset<sizeof(unsigned char)*8>(x)<<endl;
    cout<<"    binary format of "<<y<<" is "<<bitset<sizeof(unsigned char)*8>(y)<<endl;
    unsigned char z = x|y;
    cout<<"binary format of z=x|y "<<" is "<<bitset<sizeof(unsigned char)*8>(z)<<endl;
    z = x&y;
    cout<<"binary format of z=x&y "<<" is "<<bitset<sizeof(unsigned char)*8>(z)<<endl;
    z = x^y;
    cout<<"binary format of z=x^y "<<" is "<<bitset<sizeof(unsigned char)*8>(z)<<endl;
    return 0;
}
```

# 运算-位运算符



```
C:\Windows\system32\cmd.exe

    binary format of 1 is 00110001
    binary format of 7 is 00110111
binary format of z=x!y is 00110111
binary format of z=x&y is 00110001
binary format of z=x^y is 00000110
请按任意键继续. . .
```

The image shows a Windows command prompt window with a black background and white text. The title bar at the top reads 'C:\Windows\system32\cmd.exe'. The command prompt displays several lines of text showing the binary representation of numbers and the results of bitwise operations. The operations shown are: 1 (00110001), 7 (00110111), z=x!y (00110111), z=x&y (00110001), and z=x^y (00000110). The prompt ends with '请按任意键继续. . .' (Press any key to continue. . .).

# 运算符-赋值运算符

- 用于将一个值存储在内存的某个位置（变量）
- 左边的操作数必须是一个左值(left Value)，右操作数可以是任意一个表达式。因此左值通常是一个变量，但也可能是引用或指针指向的内存。如

```
int sum = (a+b)*(b-a)/2;
```

```
bool f = f1&&f2;
```

```
double d +=3;
```

```
n <<= 4;
```

# 运算符-赋值运算符

## Assignment operators.

Operator	Example	Equivalent To
=	n = 25	
+=	n += 25	n = n + 25
-=	n -= 25	n = n - 25
*=	n *= 25	n = n * 25
/=	n /= 25	n = n / 25
%=	n %= 25	n = n % 25
&=	n &= 0xF2F2	n = n & 0xF2F2
=	n  = 0xF2F2	n = n   0xF2F2
^=	n ^= 0xF2F2	n = n ^ 0xF2F2
<<=	n <<= 4	n = n << 4
>>=	n >>= 4	n = n >> 4

# 运算符-赋值运算符

- 赋值运算符本身也是表达式，其值就是左值的值，因此可以继续用于其他表达式中

```
int m, n, p;  
m = n = p = 100;           // means: n = (m = (p = 100));  
m = (n = p = 100) + 2;     // means: m = (n = (p = 100)) + 2;  
  
m = 100;  
m += n = p = 10;           // means: m = m + (n = p = 10);
```



## 运算符-条件运算符 ? :

- 根据operand1的值是true或false，其值取operand2或operand3

*operand1 ? operand2 : operand3*

```
int m = 1, n = 2;  
int min = (m < n ? m : n);           // min receives 1
```

# 运算符-逗号运算符,

- 其值取右操作数  
opdn1,opnd2

```
int m, n, min;  
int mCount = 0, nCount = 0;  
//...  
min = (m < n ? mCount++, m : nCount++, n);
```

# 运算符-sizeof运算符

- 返回任何数据项或类型占用的空间，以字节为单位。

```
1  #include <iostream.h>
2  int main (void)
3  {
4      cout << "char    size = " << sizeof(char) << " bytes\n";
5      cout << "char*   size = " << sizeof(char*) << " bytes\n";
6      cout << "short   size = " << sizeof(short) << " bytes\n";
7      cout << "int     size = " << sizeof(int) << " bytes\n";
8      cout << "long    size = " << sizeof(long) << " bytes\n";
9      cout << "float   size = " << sizeof(float) << " bytes\n";
0      cout << "double  size = " << sizeof(double) << " bytes\n";

1      cout << "1.55    size = " << sizeof(1.55) << " bytes\n";
2      cout << "1.55L   size = " << sizeof(1.55L) << " bytes\n";
3      cout << "HELLO   size = " << sizeof("HELLO") << " bytes\n";
4  }
```

# 运算符-`sizeof`运算符

- 返回任何数据项或类型占用的空间，以字节为单位。

```
char    size = 1 bytes
char*   size = 2 bytes
short   size = 2 bytes
int      size = 2 bytes
long    size = 4 bytes
float    size = 4 bytes
double  size = 8 bytes
1.55    size = 8 bytes
1.55L   size = 10 bytes
HELLO   size = 6 bytes
```

# 运算符的优先级别

- 表达式中运算符执行的次序很重要，由运算符优先规则决定。
- 如  $a == b + c * d$  中运算符执行的次序是：

$* \quad + \quad ==$

- 而  $a == (b + c) * d$  中运算符执行的次序是：

$+ \quad * \quad ==$

Level	Precedence group	Operator	Description	Grouping
1	Scope	::	scope qualifier	Left-to-right
2	Postfix (unary)	++ --	postfix increment / decrement	Left-to-right
		()	functional forms	
		[]	subscript	
		. ->	member access	
3	Prefix (unary)	++ --	prefix increment / decrement	Right-to-left
		~ !	bitwise NOT / logical NOT	
		+ -	unary prefix	
		& *	reference / dereference	
		new delete	allocation / deallocation	
		sizeof	parameter pack	
		(type)	C-style type-casting	
4	Pointer-to-member	. * ->*	access pointer	Left-to-right
5	Arithmetic: scaling	* / %	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	+ -	addition, subtraction	Left-to-right
7	Bitwise shift	<< >>	shift left, shift right	Left-to-right
8	Relational	< > <= >=	comparison operators	Left-to-right
9	Equality	== !=	equality / inequality	Left-to-right
10	And	&	bitwise AND	Left-to-right
11	Exclusive or	^	bitwise XOR	Left-to-right
12	Inclusive or		bitwise OR	Left-to-right
13	Conjunction	&&	logical AND	Left-to-right
14	Disjunction		logical OR	Left-to-right
15	Assignment-level expressions	= *= /= %= += -=>>= <<= &= ^=  =	assignment / compound assignment	Right-to-left
		?:	conditional operator	
16	Sequencing	,	comma separator	Left-to-right

# 练习

- 写程序解决下列问题：
  1. 判断一个数是否是奇数；
  2. 将一个int类型的数的 二进制的第16位设为1；
  3. 确定一个数的绝对值；
  4. 从键盘分别输入一个整数和一个实数，在窗口输出两者的和；
  5. 在程序中定义3个string类型的变量，从键盘输入字符串给两个变量，另外一个变量是这两个变量对应字符串的拼接，并输出3个字符串变量的内容。