C++变量

变量其实只不过是程序可操作的存储区的名称。C++ 中每个变量都有指定的类型，类型决定了变量存储的大小和布局，该范围内的值都可以存储在内存中，运算符可应用于变量上。

变量的名称可以由字母、数字和下划线字符组成。它必须以字母或下划线开头。大写字母和小写字母是不同的，因为 C++ 是大小写敏感的。

类型	描述
bool	存储值 true 或 false。
char	通常是一个字符（八位）。这是一个整数类型。
int	对机器而言，整数的最自然的大小。
float	单精度浮点值
double	双精度浮点值

C++变量的定义

变量定义就是告诉编译器在何处创建变量的存储，以及如何创建变量的存储。变量定义指定一个数据类型，并包含了该类型的一个或多个变量的列表，如下所示：

```
type variable_list;
```

在这里，**type** 必须是一个有效的 C++ 数据类型，可以是 char、int、float、double、bool 或任何用户自定义的对象，**variable_list** 可以由一个或多个标识符名称组成，多个标识符之间用逗号分隔。下面列出几个有效的声明：

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

行 **int i, j, k;** 声明并定义了变量 i、j 和 k，这指示编译器创建类型为 int 的名为 i、j、k 的变量。

变量可以在声明的时候被初始化（指定一个初始值）。初始化器由一个等号，后跟一个常量表达式组成，如下所示：

```
type variable_name = value;
```

下面列举几个实例：

```
int d = 3, f = 5;    // d 和 f 的声明
char x = 'x';        // 变量 x 的值为 'x'
double d = 1.2222;   // 变量 d 的值为 1.2222
```

C++基本输入输出

C++ 标准库提供了一组丰富的输入/输出功能，我们将在后续的章节进行介绍。本章将讨论 C++ 编程中最基本和最常见的 I/O 操作。

C++ 的 I/O 发生在流中，流是字节序列。如果字节流是从设备（如键盘、磁盘驱动器、网络连接等）流向内存，这叫做**输入操作**。如果字节流是从内存流向设备（如显示屏、打印机、磁盘驱动器、网络连接等），这叫做**输出操作**。

下列的头文件在 C++ 编程中很重要。

头文件	函数和描述
<iostream>	该文件定义了 **cin、cout** 对象，分别对应于标准输入流、标准输出流。
<iomanip>	该文件通过所谓的参数化的流操纵器（比如 **setw** 和 **setprecision**），来声明对执行标准化 I/O 有用的服务。

标准输出流

预定义的对象 **cout** 是 **iostream** 类的一个实例。cout 对象"连接"到标准输出设备，通常是显示屏。**cout** 是与流插入运算符 << 结合使用的，如下所示：

```
#include <iostream>

using namespace std;

int main( )
{
    cout << "hello world" << endl;
    return 0;
}
```

C++ 编译器根据要输出变量的数据类型，选择合适的流插入运算符来显示值。<< 运算符被重载来输出内置类型（整型、浮点型、double 型、字符串和指针）的数据项。

流插入运算符 << 在一个语句中可以多次使用，如上面实例中所示，**endl** 用于在行末添加一个换行符。

标准输入流

预定义的对象 **cin** 是 **iostream** 类的一个实例。cin 对象附属到标准输入设备，通常是键盘。**cin** 是与流提取运算符 >> 结合使用的，如下所示：

```
#include <iostream>

using namespace std;

int main( )
{
    int a;
    cin >> a;
    cout << a << endl;
    return 0;
}
```

算术表达式

运算符是一种告诉编译器执行特定的数学或逻辑操作的符号。C++ 内置了丰富的运算符，并提供了以下类型的运算符：

- 算术运算符
- 关系运算符
- 逻辑运算符
- 位运算符
- 赋值运算符

算术运算符

运算符	描述	实例
+	把两个操作数相加	A + B 将得到 30
-	从第一个操作数中减去第二个操作数	A - B 将得到 -10
*	把两个操作数相乘	A * B 将得到 200
/	分子除以分母	B / A 将得到 2
%	取模运算符，整除后的余数	B % A 将得到 0
++	自增运算符，整数值增加 1	A++ 将得到 11
--	自减运算符，整数值减少 1	A-- 将得到 9

实例如下：

```
#include <iostream>

using namespace std;

int main()
{
   int a = 21;
   int b = 10;
   int c;

   c = a + b;
   cout << "Line 1 - c 的值是 " << c << endl ;
   c = a - b;
   cout << "Line 2 - c 的值是 " << c << endl ;
   c = a * b;
   cout << "Line 3 - c 的值是 " << c << endl ;
   c = a / b;
   cout << "Line 4 - c 的值是 " << c << endl ;
   c = a % b;
   cout << "Line 5 - c 的值是 " << c << endl ;

   int d = 10;   //  测试自增、自减
   c = d++;
   cout << "Line 6 - c 的值是 " << c << endl ;

   d = 10;    // 重新赋值
   c = d--;
   cout << "Line 7 - c 的值是 " << c << endl ;
```

```
   return 0;
}
```

关系运算符

假设变量 A 的值为 10，变量 B 的值为 20，则：

运算符	描述	实例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(A == B) 不为真。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(A != B) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(A > B) 不为真。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(A < B) 为真。
>=	检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。	(A >= B) 不为真。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。	(A <= B) 为真。

实例如下：

```
#include <iostream>

using namespace std;

int main()
{
   int a = 21;
   int b = 10;
   int c ;

   if( a == b )
   {
      cout << "Line 1 - a 等于 b" << endl ;
   }
   else
   {
      cout << "Line 1 - a 不等于 b" << endl ;
   }
   if ( a < b )
   {
      cout << "Line 2 - a 小于 b" << endl ;
   }
   else
   {
      cout << "Line 2 - a 不小于 b" << endl ;
   }
   if ( a > b )
   {
      cout << "Line 3 - a 大于 b" << endl ;
   }
```

```
   else
   {
      cout << "Line 3 - a 不大于 b" << endl ;
   }
   /* 改变 a 和 b 的值 */
   a = 5;
   b = 20;
   if ( a <= b )
   {
      cout << "Line 4 - a 小于或等于 b" << endl ;
   }
   if ( b >= a )
   {
      cout << "Line 5 - b 大于或等于 a" << endl ;
   }
   return 0;
}
```

逻辑运算符

假设变量 A 的值为 1，变量 B 的值为 0，则：

运算符	描述	实例
&&	称为逻辑与运算符。如果两个操作数都非零，则条件为真。	(A && B) 为假。
||	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(A || B) 为真。
!	称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	!(A && B) 为真。

实例如下：

```
#include <iostream>

using namespace std;

int main()
{
   int a = 5;
   int b = 20;
   int c ;

   if ( a && b )
   {
      cout << "Line 1 - 条件为真"<< endl ;
   }
   if ( a || b )
   {
      cout << "Line 2 - 条件为真"<< endl ;
   }
   /* 改变 a 和 b 的值 */
```

```
   a = 0;
   b = 10;
   if ( a && b )
   {
      cout << "Line 3 - 条件为真"<< endl ;
   }
   else
   {
      cout << "Line 4 - 条件不为真"<< endl ;
   }
   if ( !(a && b) )
   {
      cout << "Line 5 - 条件为真"<< endl ;
   }
   return 0;
}
```

位运算符

位运算符作用于位，并逐位执行操作。&、 | 和 ^ 的真值表如下所示：

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

假设如果 A = 60，且 B = 13，现在以二进制格式表示，它们如下所示：

A = 0011 1100

B = 0000 1101

运算符	描述	实例
&	如果同时存在于两个操作数中，二进制 AND 运算符复制一位到结果中。	(A & B) 将得到 12，即为 0000 1100
|	如果存在于任一操作数中，二进制 OR 运算符复制一位到结果中。	(A | B) 将得到 61，即为 0011 1101
^	如果存在于其中一个操作数中但不同时存在于两个操作数中，二进制异或运算符复制一位到结果中。	(A ^ B) 将得到 49，即为 0011 0001
~	二进制补码运算符是一元运算符，具有"翻转"位效果，即0变成1，1变成0。	(~A) 将得到 -61，即为 1100 0011，一个有符号二进制数的补码形式。
<<	二进制左移运算符。左操作数的值向左移动右操作数指定的位数。	A << 2 将得到 240，即为 1111 0000
>>	二进制右移运算符。左操作数的值向右移动右操作数指定的位数。	A >> 2 将得到 15，即为 0000 1111

实例如下：

```
#include <iostream>

using namespace std;

int main()
{
   unsigned int a = 60;       // 60 = 0011 1100
   unsigned int b = 13;       // 13 = 0000 1101
   int c = 0;

   c = a & b;              // 12 = 0000 1100
   cout << "Line 1 - c 的值是 " << c << endl ;

   c = a | b;              // 61 = 0011 1101
   cout << "Line 2 - c 的值是 " << c << endl ;

   c = a ^ b;              // 49 = 0011 0001
   cout << "Line 3 - c 的值是 " << c << endl ;

   c = ~a;                 // -61 = 1100 0011
   cout << "Line 4 - c 的值是 " << c << endl ;

   c = a << 2;             // 240 = 1111 0000
   cout << "Line 5 - c 的值是 " << c << endl ;

   c = a >> 2;             // 15 = 0000 1111
   cout << "Line 6 - c 的值是 " << c << endl ;

   return 0;
}
```

赋值运算符

下表列出了 C++ 支持的赋值运算符：

运算符	描述	实例
=	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 + B 将把 A + B 的值赋给 C
+=	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 相当于 C = 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 相当于 C = C - A
*=	乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数	C *= A 相当于 C = 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 相当于 C = 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 %= A 相当于 C = C % A
<<=	左移且赋值运算符	C <<= 2 等同于 C = C << 2
>>=	右移且赋值运算符	C >>= 2 等同于 C = C >> 2
&=	按位与且赋值运算符	C &= 2 等同于 C = C & 2
^=	按位异或且赋值运算符	C ^= 2 等同于 C = C ^ 2
|=	按位或且赋值运算符	C |= 2 等同于 C = C | 2

实例如下：

```
#include <iostream>

using namespace std;

int main()
{
   int a = 21;
   int c ;

   c =  a;
   cout << "Line 1 - =  运算符实例，c 的值 = : " <<c<< endl ;

   c +=  a;
   cout << "Line 2 - += 运算符实例，c 的值 = : " <<c<< endl ;

   c -=  a;
   cout << "Line 3 - -= 运算符实例，c 的值 = : " <<c<< endl ;

   c *=  a;
```

```
   cout << "Line 4 - *= 运算符实例，c 的值 = : " <<c<< endl ;

   c /=  a;
   cout << "Line 5 - /= 运算符实例，c 的值 = : " <<c<< endl ;

   c  = 200;
   c %=  a;
   cout << "Line 6 - %= 运算符实例，c 的值 = : " <<c<< endl ;

   c <<=  2;
   cout << "Line 7 - <<= 运算符实例，c 的值 = : " <<c<< endl ;

   c >>=  2;
   cout << "Line 8 - >>= 运算符实例，c 的值 = : " <<c<< endl ;

   c &=  2;
   cout << "Line 9 - &= 运算符实例，c 的值 = : " <<c<< endl ;

   c ^=  2;
   cout << "Line 10 - ^= 运算符实例，c 的值 = : " <<c<< endl ;

   c |=  2;
   cout << "Line 11 - |= 运算符实例，c 的值 = : " <<c<< endl ;

   return 0;
}
```

选择结构

if语句

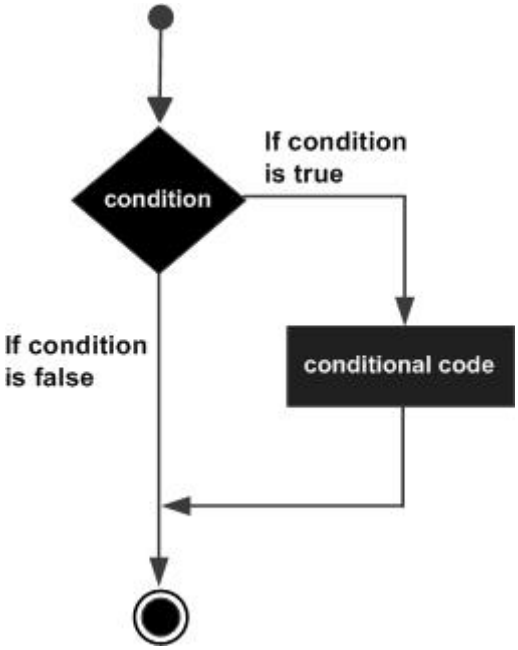
一个 **if 语句** 由一个布尔表达式后跟一个或多个语句组成。

C++ 中 **if** 语句的语法：

```
if(boolean_expression)
{
   // 如果布尔表达式为真将执行的语句
}
```

如果布尔表达式为 **true**，则 if 语句内的代码块将被执行。如果布尔表达式为 **false**，则 if 语句结束后的第一组代码（闭括号后）将被执行。

C 语言把任何**非零**和**非空**的值假定为 **true**，把**零**或 **null** 假定为 **false**。




```
#include <iostream>

using namespace std;

int main ()
{
   // 局部变量声明
   int a = 10;

   // 使用 if 语句检查布尔条件
   if( a < 20 )
   {
       // 如果条件为真，则输出下面的语句
       cout << "a 小于 20" << endl;
   }
   cout << "a 的值是 " << a << endl;

   return 0;
}
```

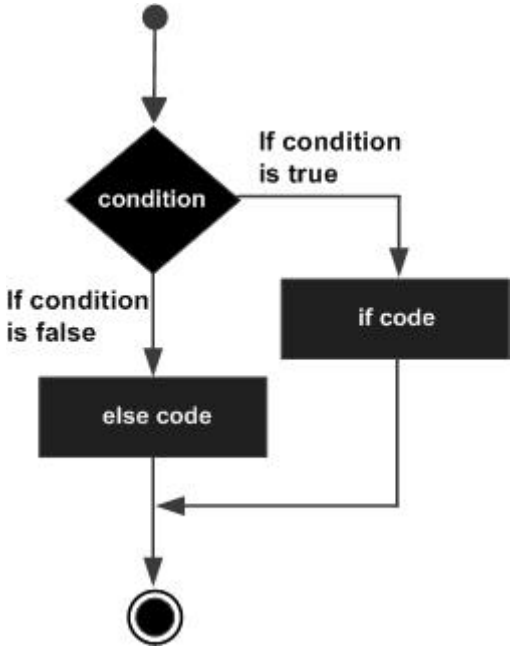
if...else语句

一个 **if 语句** 后可跟一个可选的 **else 语句**，else 语句在布尔表达式为假时执行。

C++ 中 **if...else** 语句的语法：

```
if(boolean_expression)
{
   // 如果布尔表达式为真将执行的语句
}
else
{
   // 如果布尔表达式为假将执行的语句
}
```

如果布尔表达式为 **true**，则执行 **if** 块内的代码。如果布尔表达式为 **false**，则执行 **else** 块内的代码。




```
#include <iostream>

using namespace std;

int main ()
{
   // 局部变量声明
   int a = 100;

   // 检查布尔条件
   if( a < 20 )
   {
       // 如果条件为真，则输出下面的语句
       cout << "a 小于 20" << endl;
   }
   else
   {
       // 如果条件为假，则输出下面的语句
       cout << "a 大于 20" << endl;
   }
   cout << "a 的值是 " << a << endl;

   return 0;
}
```

switch语句

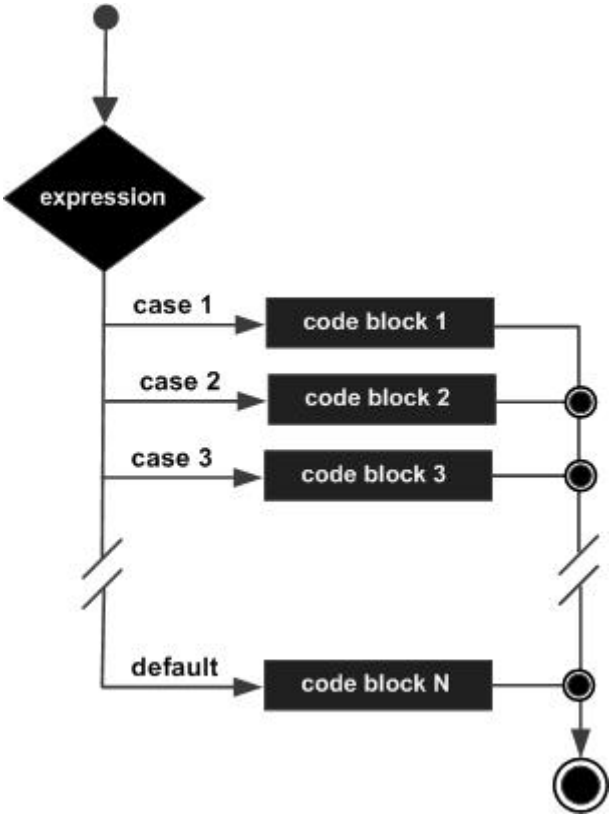
一个 **switch** 语句允许测试一个变量等于多个值时的情况。每个值称为一个 case，且被测试的变量会对每个 **switch case** 进行检查。

C++ 中 **switch** 语句的语法：

```
switch(expression){
    case constant-expression  :
       statement(s);
       break; // 可选的
    case constant-expression  :
       statement(s);
       break; // 可选的

    // 您可以有任意数量的 case 语句
    default : // 可选的
       statement(s);
}
```

switch 语句必须遵循下面的规则：

- **switch** 语句中的 **expression** 必须是一个整型或枚举类型，或者是一个 class 类型，其中 class 有一个单一的转换函数将其转换为整型或枚举类型。
- 在一个 switch 中可以有任意数量的 case 语句。每个 case 后跟一个要比较的值和一个冒号。
- case 的 **constant-expression** 必须与 switch 中的变量具有相同的数据类型，且必须是一个常量或字面量。
- 当被测试的变量等于 case 中的常量时，case 后跟的语句将被执行，直到遇到 **break** 语句为止。
- 当遇到 **break** 语句时，switch 终止，控制流将跳转到 switch 语句后的下一行。
- 不是每一个 case 都需要包含 **break**。如果 case 语句不包含 **break**，控制流将会 *继续* 后续的 case，直到遇到 break 为止。
- 一个 **switch** 语句可以有一个可选的 **default** case，出现在 switch 的结尾。default case 可用于在上面所有 case 都不为真时执行一个任务。default case 中的 **break** 语句不是必需的。




```
#include <iostream>

using namespace std;

int main ()
{
```

```
   // 局部变量声明
   char grade = 'D';

   switch(grade)
   {
   case 'A' :
      cout << "很棒！" << endl;
      break;
   case 'B' :
   case 'C' :
      cout << "做得好" << endl;
      break;
   case 'D' :
      cout << "您通过了" << endl;
      break;
   case 'F' :
      cout << "最好再试一下" << endl;
      break;
   default :
      cout << "无效的成绩" << endl;
   }
   cout << "您的成绩是 " << grade << endl;

   return 0;
}
```

循环结构

while循环

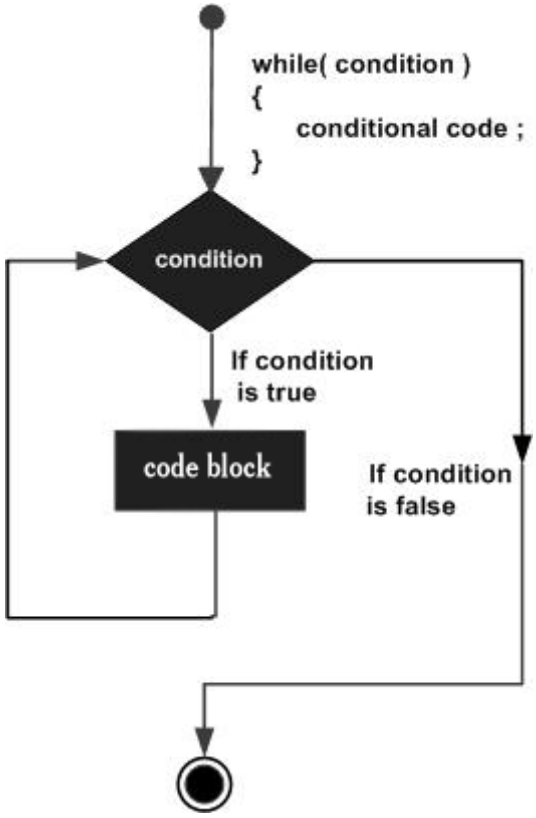
只要给定的条件为真，**while** 循环语句会重复执行一个目标语句。

C++ 中 **while** 循环的语法：

```
while(condition)
{
   statement(s);
}
```

在这里，**statement(s)** 可以是一个单独的语句，也可以是几个语句组成的代码块。**condition** 可以是任意的表达式，当为任意非零值时都为真。当条件为真时执行循环。

当条件为假时，程序流将继续执行紧接着循环的下一条语句。



在这里，*while* 循环的关键点是循环可能一次都不会执行。当条件被测试且结果为假时，会跳过循环主体，直接执行紧接着 while 循环的下一条语句。

```
#include <iostream>

using namespace std;

int main ()
{
   // 局部变量声明
   int a = 10;

   // while 循环执行
   while( a < 20 )
   {
       cout << "a 的值: " << a << endl;
       a++;
   }

   return 0;
}
```

for循环

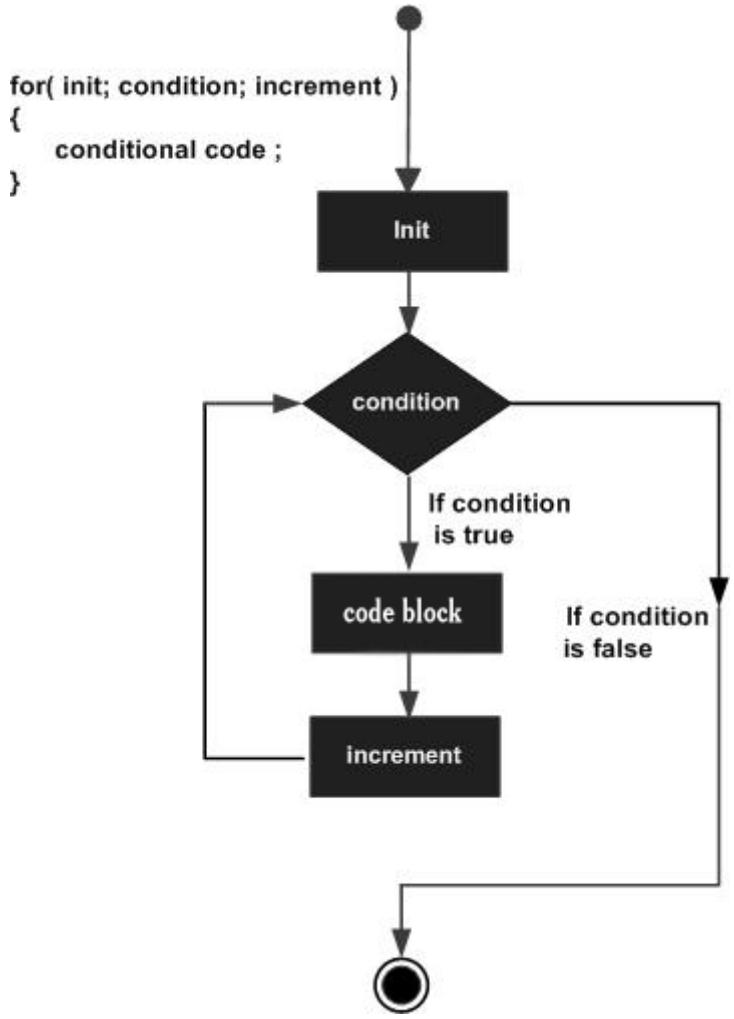
for 循环允许您编写一个执行特定次数的循环的重复控制结构。

C++ 中 **for** 循环的语法：

```
for ( init; condition; increment )
{
   statement(s);
}
```

下面是 for 循环的控制流：

1. **init** 会首先被执行，且只会执行一次。这一步允许您声明并初始化任何循环控制变量。您也可以不在这里写任何语句，只要有一个分号出现即可。
2. 接下来，会判断 **condition**。如果为真，则执行循环主体。如果为假，则不执行循环主体，且控制流会跳转到紧接着 for 循环的下一条语句。
3. 在执行完 for 循环主体后，控制流会跳回上面的 **increment** 语句。该语句允许您更新循环控制变量。该语句可以留空，只要在条件后有一个分号出现即可。
4. 条件再次被判断。如果为真，则执行循环，这个过程会不断重复（循环主体，然后增加步值，再然后重新判断条件）。在条件变为假时，for 循环终止。




```
#include <iostream>

using namespace std;

int main ()
{
   // for 循环执行
   for( int a = 10; a < 20; a = a + 1 )
   {
       cout << "a 的值: " << a << endl;
   }

   return 0;
}
```

do...while循环

不像 **for** 和 **while** 循环，它们是在循环头部测试循环条件。**do...while** 循环是在循环的尾部检查它的条件。

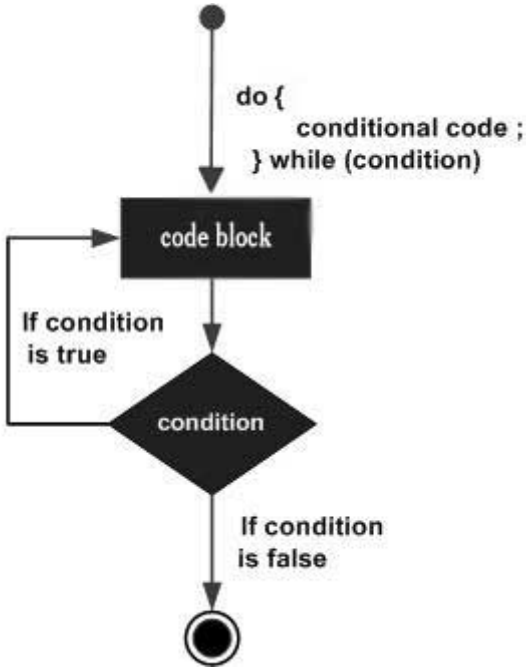
do...while 循环与 while 循环类似，但是 do...while 循环会确保至少执行一次循环。

C++ 中 **do...while** 循环的语法：

```
do
{
   statement(s);

}while( condition );
```

请注意，条件表达式出现在循环的尾部，所以循环中的 statement(s) 会在条件被测试之前至少执行一次。

如果条件为真，控制流会跳转回上面的 do，然后重新执行循环中的 statement(s)。这个过程会不断重复，直到给定条件变为假为止。



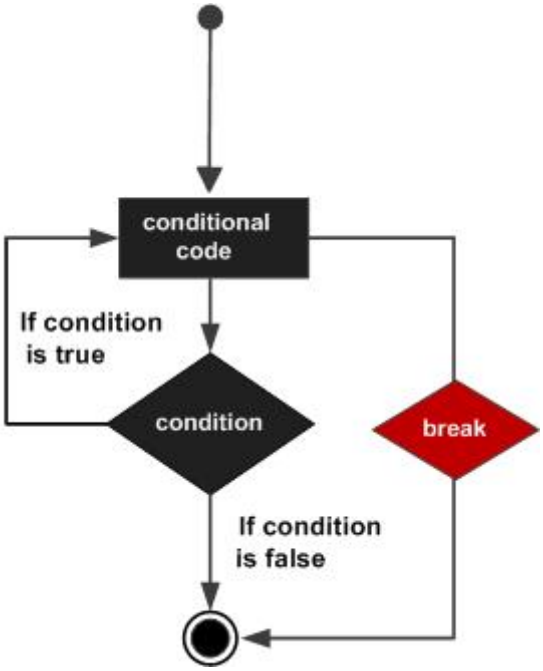

```
#include <iostream>

using namespace std;

int main ()
{
   // 局部变量声明
   int a = 10;

   // do 循环执行
   do
   {
       cout << "a 的值：" << a << endl;
       a = a + 1;
   }while( a < 20 );

   return 0;
}
```

break语句

C++ 中 **break** 语句有以下两种用法：

1. 当 **break** 语句出现在一个循环内时，循环会立即终止，且程序流将继续执行紧接着循环的下一条语句。
2. 它可用于终止 **switch** 语句中的一个 case。

如果您使用的是嵌套循环（即一个循环内嵌套另一个循环），break 语句会停止执行最内层的循环，然后开始执行该块之后的下一行代码。

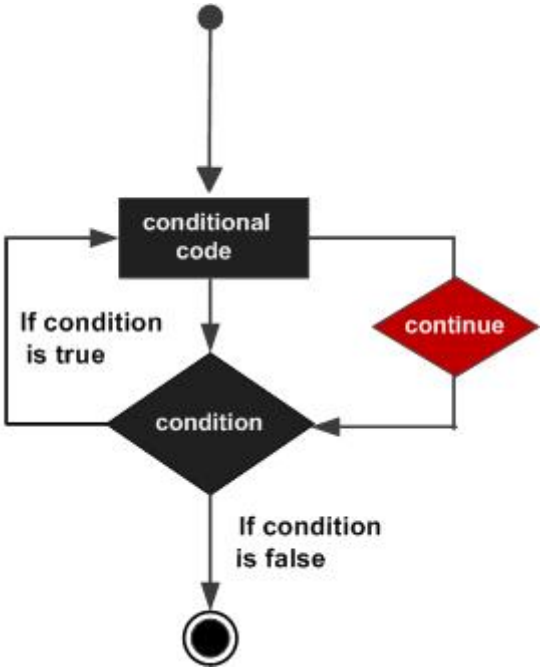



```
#include <iostream>

using namespace std;

int main ()
{
   // 局部变量声明
   int a = 10;

   // do 循环执行
   do
   {
       cout << "a 的值： " << a << endl;
       a = a + 1;
       if( a > 15)
       {
          // 终止循环
          break;
       }
   }while( a < 20 );

   return 0;
}
```

continue语句

C++ 中的 **continue** 语句有点像 **break** 语句。但它不是强迫终止，continue 会跳过当前循环中的代码，强迫开始下一次循环。

对于 **for** 循环，**continue** 语句会导致执行条件测试和循环增量部分。对于 **while** 和 **do...while** 循环，**continue** 语句会导致程序控制回到条件测试上。




```
#include <iostream>

using namespace std;

int main ()
{
   // 局部变量声明
   int a = 10;

   // do 循环执行
   do
   {
       if( a == 15)
       {
          // 跳过迭代
          a = a + 1;
          continue;
       }
       cout << "a 的值：" << a << endl;
       a = a + 1;
   }while( a < 20 );

   return 0;
}
```

ACM输入

对于多组输入，以a+b为例，以下为三种常见的多组输入方法：

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a, b;
    while (cin >> a >> b) {
        cout << a + b << endl;
    }
    return;
}
```

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a, b;
    while (scanf("%d%d", &a, &b) != EOF) {
        printf("%d\n", a + b);
    }
    return;
}
```

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a, b;
    while (~scanf("%d%d", &a, &b)) {
        printf("%d\n", a + b);
    }
    return;
}
```