C程序设计基础

Introduction to C programming
Lecture 6: Loop

张振国 zhangzg@sustech.edu.cn

南方科技大学/理学院/地球与空间科学系

Review on L5 Decision

Relational and Logical operators

If statement

Switch statement

Objective of this lecture

You can use C to control the workflow!

Straight-line code

Flow of control

Logical/boolean operators

Define two variables: int A = 0, B = 1;

Operators	Description	Example
&&	AND operator, if both are on, then on	A&&B = 0 (false)
||	OR operator, if any is on, then on	A ||B = 1 (true)
!	NOT operator, turn opposite	!A = 1 (true)!B = 0 (false)

Logical/boolean operators

short-circuit evaluation “短路”计算

```
&&

||
```

These operators first evaluate the left operand, then the right operand.
If the value of the expression can be deduced from the value of the left operand alone, then the right operand isn’t evaluated.

```
(i!=0)&&(j/i>0)
```

```
i>0 && ++j>0
```

此种情况会影响结果

Precedence

a<=b && b<=c

```
include<stdbool.h>
flat a=2.5,b=7.5,c=5.0,d=6.0;
printf("%d",c/2.0+d <a && !true||c<=d
```

1

Precedence	*Name*	*Symbol(s)*	*Associativity*
1	Array subscripting	[]	Left
1	Function call	()	Left
1	Structure and union member	. ->	Left
1	Increment (postfix)	++	Left
1	Decrement (postfix)	--	Left
2	Increment (prefix)	++	Right
2	Decrement (prefix)	--	Right
2	Address	&	Right
2	Indirection	*	Right
2	Unary plus	+	Right
2	Unary minus	-	Right
2	Bitwise complement	~	Right
2	Logical negation	!	Right
2	Size	sizeof	Right
3	Cast	()	Right
4	Multiplicative	* / %	Left
5	Additive	+ -	Left
6	Bitwise shift	<< >>	Left
7	Relational	< > <= >=	Left
8	Equality	== !=	Left
9	Bitwise *and*	&	Left
10	Bitwise exclusive *or*	^	Left
11	Bitwise inclusive *or*	|	Left
12	Logical *and*	&&	Left
13	Logical *or*	||	Left
14	Conditional	?:	Right
15	Assignment	= *= /= %= += -= <<= >>= &= ^= |=	Right
16	Comma	,	Left

If statement

If statement has a boolean expression followed by one or more statements.

```
if(boolean_expression)
{ /* code 1 */ }
```

```
if(boolean_expression)
{ /* code 1 */ }
else
{ /* code 2 */ }
```

If statement

if(condition)	如果(条件满足)
{option A}	{A选项}
else	否则
{option B}	{B选项}

If statement

condition

Ture/Yes/1

(2<5, 3==3, ’a’!=‘b’)

False/No/0

(1>=5, 3<1,
’a’==‘b’)

If and if-else

Question?

```
int a = 5;
if (a > 10);
   printf("a>10");
```

```
int a = 15;
if (a > 10);
   printf("a>10");
```

a>10

```
if (a > 10);
   printf("a>10");
else
   printf("a<10");
```

error

If - else if

If-elseif has more boolean expression followed by more statements.

```
if( condition 1 )
{ /* code 1 */ }
elseif( condition 2 )
{ /* code 2 */ }
elseif( condition 3 )
{ /* code 3 */ }
elseif( condition 4 )
{ /* code 4 */ }
…
else
{/* code N */}
```

Case study: If - else if

Case: what is the cost of attendance?

To be removed

```
#include <stdio.h>
main()
{
    int a;
    printf("Enter your age:\n");
    scanf("%d", &a);
    if( a < 10 )
    {
        printf("Your cost is 0$\n" );
    }
    else if( a >= 10 && a < 20 )
    {
        printf("Your cost is 25$\n" );
    }
    else
    {
        printf("Your cost is 40$\n" );
    }
}
```

```
Enter your age:
3
Your cost is 0$
```

```
Enter your age:
17
Your cost is 25$
```

```
Enter your age:
45
Your cost is 40$
```

? statement

expression： **expression1 ? expression2 : expression3**

- If *expression1* is true (nonzero), the whole conditional expression has the same value as *expression2*. If *expression1* is false (zero), the whole conditional expression has the same value as *expression3* .

```
if (y < 0)
    x = -y;
else
    x = y;
```

→

```
x = (y < 0) ? -y : y;
```

Nested-if

Nested if-else statement means if can be used inside another if.

```
if( condition 1 )
{
   /* code 1 */
   if( condition 2)
   {
      /* code 2 */
   }
}
```

If statement

- 如果没有花括号指明，else与和它最接近的一个if相匹配

```
if (condition)
  do this;

if (condition)
  do this;
  more jobs; ?
else
  do this;
```

else goes with the most recent if

```
if (condition)
{
  do this;
  if (condition)
    do this;
}
else
  do this;
```

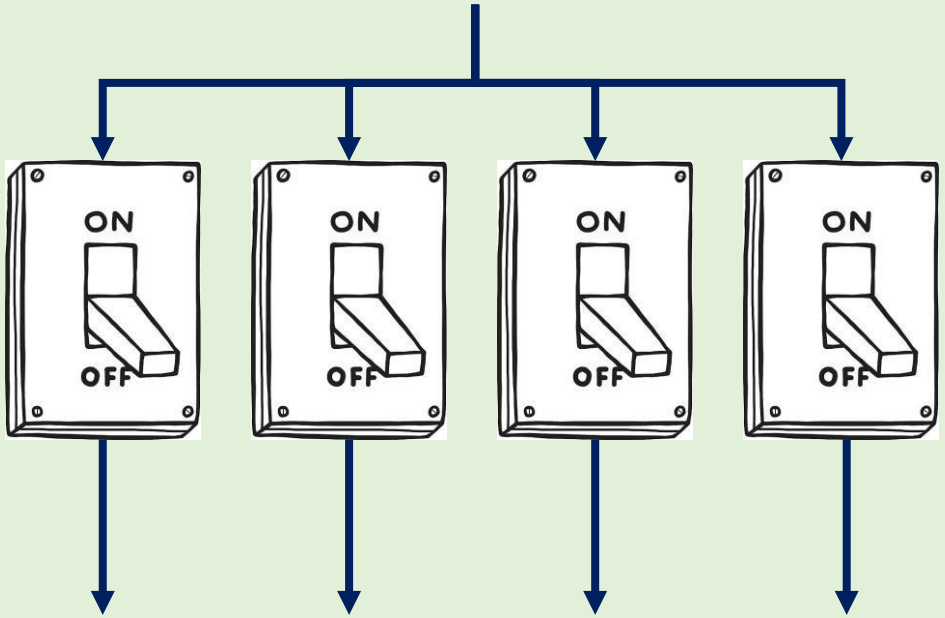
else goes with the first if since braces enclose inner if statements

Switch statement

Switch statement allows a variable to be tested for **<u>equality</u>** against a list of values. Case will be switched on if equality meets

```
switch(variable)
{
   case constant1:
      statement;
      break;
   case constant2:
      statement;
      break;
   default:
      statement;
}
```

optional



Case study: switch

Case: how to evaluate students based on grades?

```c
#include <stdio.h>
main()
{
    char a;
    printf("please input your grade:\n");
    scanf("%c", &a);
    printf("Your grade is %c\n", a );
    switch(a)
    {
        case 'A':
            printf("Excellent!\n" );break;
        case 'B':
            printf("Well done\n" );break;
        case 'C' :
            printf("You passed\n" );break;
        case 'D' :
            printf("Better try again\n" );break;
        default :
            printf("Invalid grade\n" );
    }
}
```

```
please input your grade:
A
Your grade is A
Excellent!
```

```
please input your grade:
B
Your grade is B
Well done
```

```
please input your grade:
C
Your grade is C
You passed
```

```
please input your grade:
D
Your grade is D
Better try again
```

```
please input your grade:
E
Your grade is E
Invalid grade
```

Switch statement

- switch后面括弧内的“表达式”，ANSI标准允许它为任何类型。
- 当表达式的值与某一个case后面的常量表达式的值相等时，就执行此case后面的语句，若所有的case中的常量表达式的值都没有与表达式的值匹配的，就执行default后面的语句，若都不满足则跳出。
- 每一个case的常量表达式的值必须互不相同，否则就会出现互相矛盾的现象（对表达式的同一个值，有两种或多种执行方案）。

```
int a = 3;
switch(a)
{
    case 1:
        //...
        break;
    case 2:
        //...
        break;
    default:
        //...
}
```

```
case 'A':
    printf("Excellent!\n" );break;
case 'A':
    printf("Well done\n" );break;
```

error: duplicate case value

Switch statement

- 各个case和default的出现次序不影响执行结果。例如，可以先出现“default：…”，再出现“case ′D′：…”，然后是“case ′A′：…”。

```
switch(a)
{
    case 'A':
        printf("Excellent!\n" );break;
    default :
        printf("Invalid grade\n" );
    case 'B':
        printf("Well done\n" );break;
    case 'C' :
        printf("You passed\n" );break;
    case 'D' :
        printf("Better try again\n"
    );break;
}
```

??
There is a bug. We will discuss it later.

Switch statement

- 各个case和default的出现次序不影响执行结果。例如，可以先出现“default：…”，再出现“case ′D′：…”，然后是“case ′A′：…”。
- 执行完一个case后面的语句后，流程控制转移到下一 个case继续执行。“case常量表达式”只是起语句标号作用，并不是在条件判断。在执行 switch语句时，根据switch后面表达式的值找到匹配的入口标号，就从此标号开始执行下去，不再进行判断。应该在执行一个case分支后，可以用一个break语句来终止switch语句的执行。

Switch statement

```
#include<stdio.h>
int main(void){
  int grad;
  scanf("%d", &grad);
  switch(grad){
    case 4:
      printf("Excellent\n");
    case 3:
      printf("Good\n");
    case 2:
      printf("Average\n");
    case 1:
      printf("Poor\n");
    case 0:
      printf("Failing\n“);
    default:
      printf("Illegal grad\n");
  }
  return 0;}
```

```
4
Excellent
Good
Average
Poor
Failing
Illegal grad
```

```
2
Average
Poor
Failing
Illegal grad
```

Switch statement

- 多个可以共用一组执行语句。

```
#include<stdio.h>
int main(void){
  int grad;
  scanf("%d", &grad);
  switch(grad){
    case 4:
    case 3:
    case 2:
    case 1:
      printf(“Passing\n");break;
    case 0:
      printf(“Failing\n");break;
    default:
      printf("Illegal grad\n");break;
}
  return 0;}
```

Switch statement

- 多个可以共用一组执行语句。

```
switch(ch){
    case ‘a’:
    case ‘A’: a_ct++;
              break;
    case ‘e’:
    case ‘E’: e_ct++;
              break;
    case ‘i’:
    case ‘I’: i_ct++;
              break;
    default: break;
}
```

- break出现与否结果差别较大；
- 分清程序中丢失的break是故意还是错误；

```
??constant1=‘a’ ||‘A’
```

非语法错误，
‘a’ ||‘A’=0/1

```
case constant1:
    statement;
    break;
```

Switch statement

Switch statement allows a variable to be tested for **equality** against a list of values. Case will be switched on if equality meets

```
switch(variable)
{
   case constant1:
      statement;
      break;
   case constant2:
      statement;
      break;
   default:
      statement;
}
```

常量表达式constant expression，
整数或者字符

5；
10+5；
‘A’；

”A” n+1	error: case label does not reduce to an integer constant

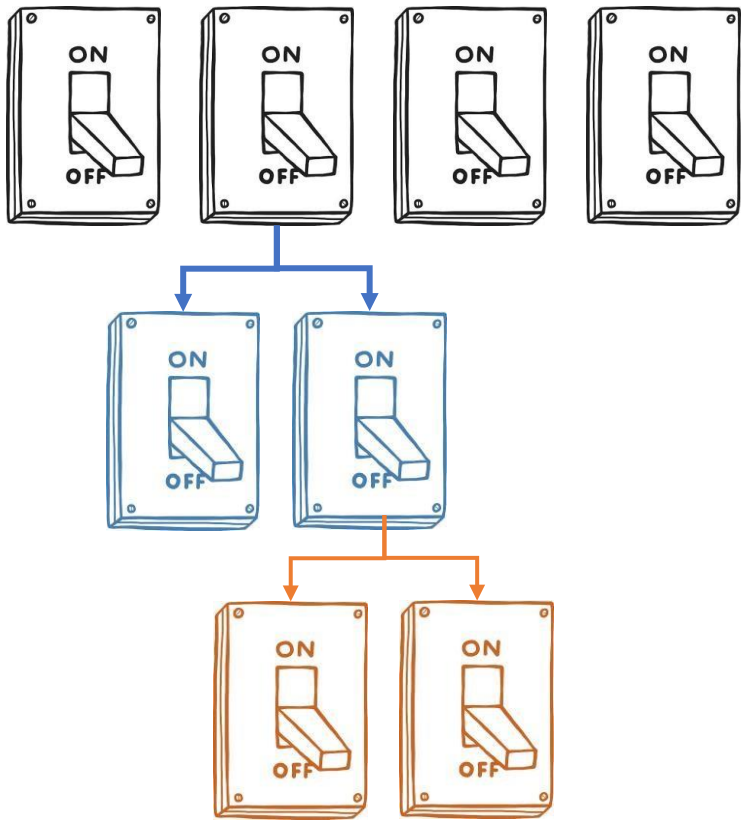
const int n=1;

#define n 1

Nested-switch

Switch can be nested. Even if the case constants of the inner and outer switch are the same, no conflict will arise.

```
switch(ch1) {
case 'A’:
   switch(ch2) {
   case ‘a’:
      statement;
      break;
   case ‘A’:
      statement;
      break;
   }
case ‘B’:
}
```



Case study: nested-switch

Case: create a simple login system!

```c
#include <stdio.h>
main()
{
    char a;
    int pw;
    printf("please input your name(alphabet):\n");
    scanf("%c", &a);
    switch(a) {
        case 'A':
            printf("Hello! Alex, please input your password:\n");
            scanf("%d", &pw);
            switch(pw) {
                case 202:
                printf("Login Successfully!");break;
                default:
                printf("Wrong Password\n");
            }break;
        default:
        printf("Unregistered\n" );
    }
}
```

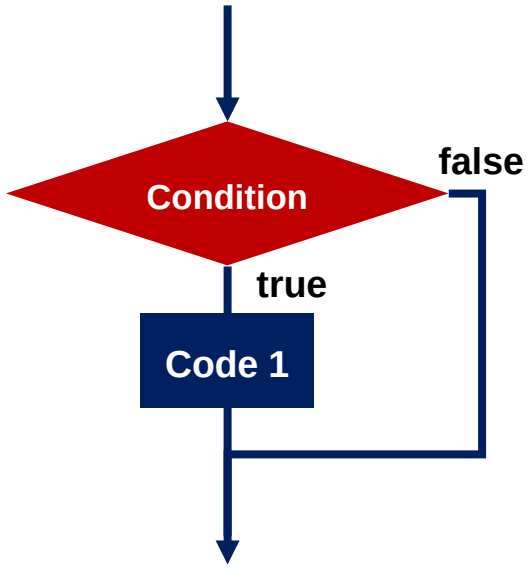
```
please input your name(alphabet):
M
Unregistered
```

```
please input your name(alphabet):
A
Hello! Alex, please input your password:
111
Wrong Password
```

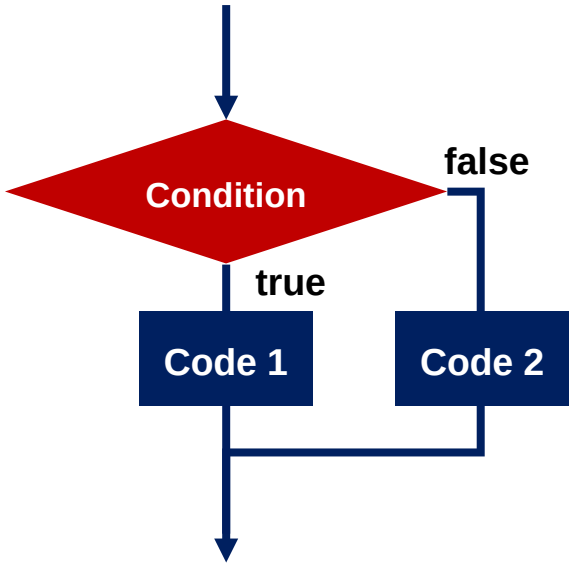
```
please input your name(alphabet):
A
Hello! Alex, please input your password:
202
Login Successfully!
```

Overview of decision-making

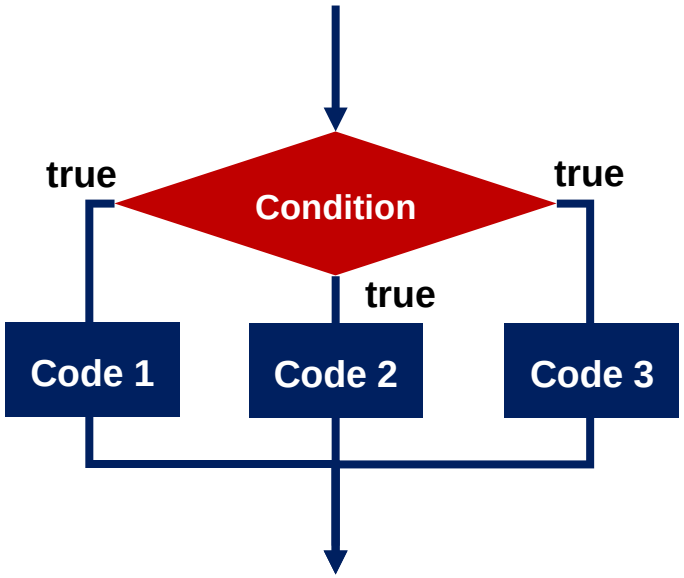
If



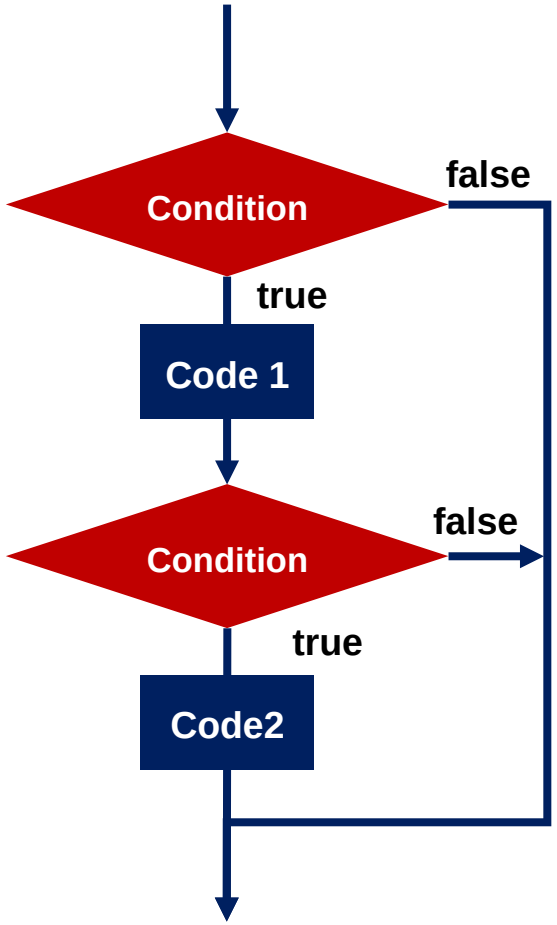
If else



If elseif



Nested if



Suppl.

Conditinal compilation

You can use them to **tell the compiler to accept or ignore blocks** of information or code according to conditions at the time of compilation.

```
(1)#ifdef 标识符
        程序段 1
  #else
        程序段 2
  #endif
```

```
(2)#ifndef 标识符
        程序段 1
  #else
        程序段 2
  #endif
```

```
(3) #if 表达式
        程序段 1
  #else
        程序段 2
  #endif
```

Conditinal compilation

- **#ifdef**指令说明，如果预处理器已定义了后面的标识符，则执行**#else**或**#endif**指令之前的所有指令并编译所有C代码（先出现哪个指令就执行到哪里）。如果预处理器未定义，且有 **#else**指令，则执行**#else**和**#endif**指令之间的所有代码。

```
#ifdef 标识符
    程序段 1
#else
    程序段 2
#endif
```

```
#ifdef MAVIS
     #include "horse.h"  // gets done if MAVIS is #defined
     #define  STABLES     5
#else
     #include "cow.h"    // gets done if MAVIS isn't #defined
     #define  STABLES    15
#endif
```

Conditinal compilation

- **#ifndef**指令判断后面的标识符是否是未定义的，常用于定义之前未定义的常量：

```
#ifndef 标识符
    程序段 1
#else
    程序段 2
#endif
```

```
/* arrays.h  */
#ifndef SIZE
   #define SIZE 100
#endif
```

(Older implementations might not permit indenting the #define directive.)

- 包含多个头文件时，其中的文件可能包含了相同宏定义。**#ifndef**指令可以防止相同的宏被重复定义。

1. 尽量不要使用这种语句，相当于把switch语句当成if语句来使用。不符合switch的设计初衷，也会影响程序的运行速度。

2. scanf(“%d%d”,&a,&b)不能直接读取两位数每个位置上的值，在没有分隔符的情况下，编译器会认为给出的两位数字是第一个变量的数值。可以通过如下的语句来指定每一个变量输入的位数。（或只读入一个两位数，用/10和%10的方法来得到十位和个位数字上的值）

新文件1.cpp

```cpp
#include <stdio.h>

int main() {
    int a, b;
    scanf("%1d%1d", &a, &b);
    printf("%d%d", a, b);
    return 0;
}
```

C:\Users\night\Documents\新

```
12
12
--------------------------------
Process exited after 1.445 seconds with return value 0
请按任意键继续. . .
```

```cpp
switch (w <= 59)
{
case 1:
    printf(" 0~59"); break;
case 0:
    switch (w <= 69)
    {
    case 1:
        printf("60~69"); break;
    case 0:
        switch (w <= 79)
        {
        case 1:
            printf("70~79"); break;
        case 0:
            switch (w <= 89)
            {
            case 1:
                printf(" 80~89"); break;
            case 0:
                printf("90~100");
            }
        }
    }
}
```

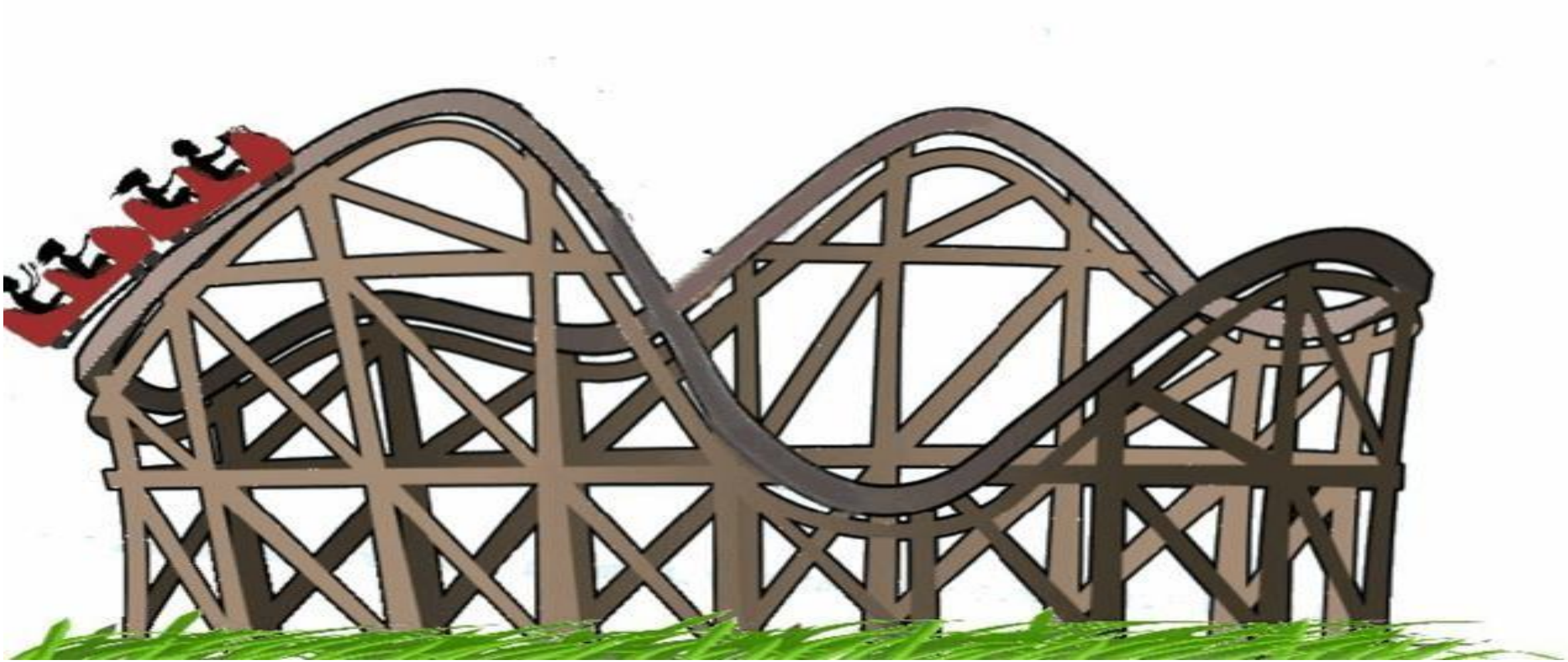
3. 作业要求把所有内容包括截图放到同一个PDF文档里，提交这种作业会导致转码失败而且难以批改。

Content

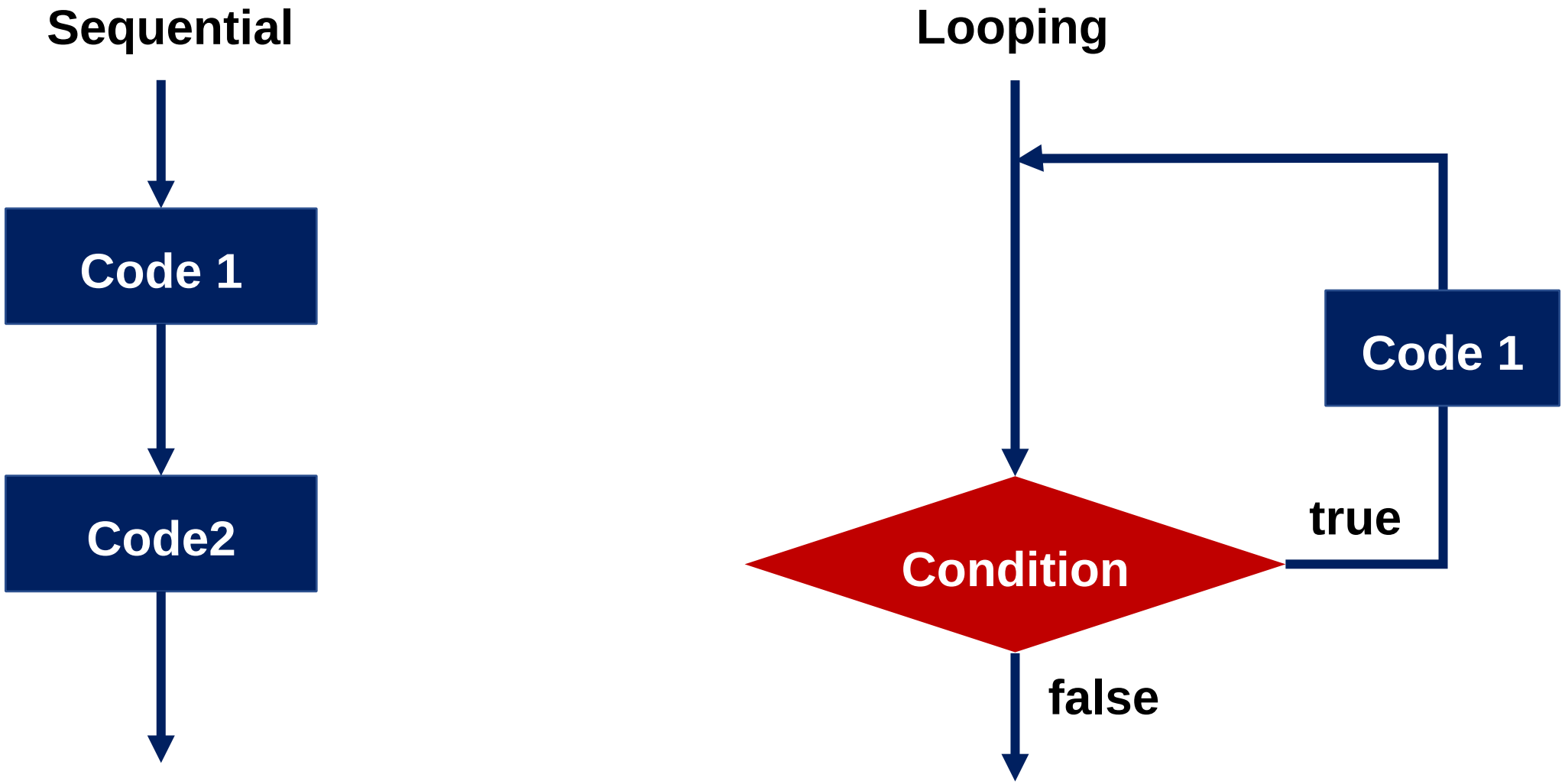
1. while statement
2. do while statement
3. for statement
4. break/continue/goto

Looping in life

Looping in life



Looping in program

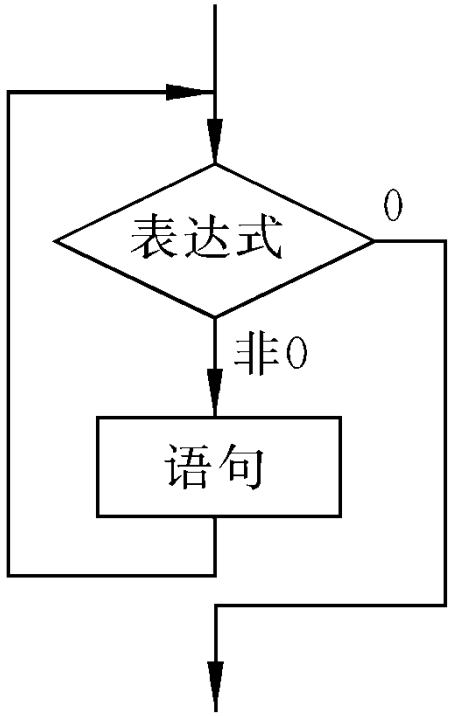


While loop

While loop repeatedly executes a statement as long as the condition is true.

```
while(condition)
    statement;
```

```
while(condition)
{
    statements;
}
```



有可能一次循环都不执行！

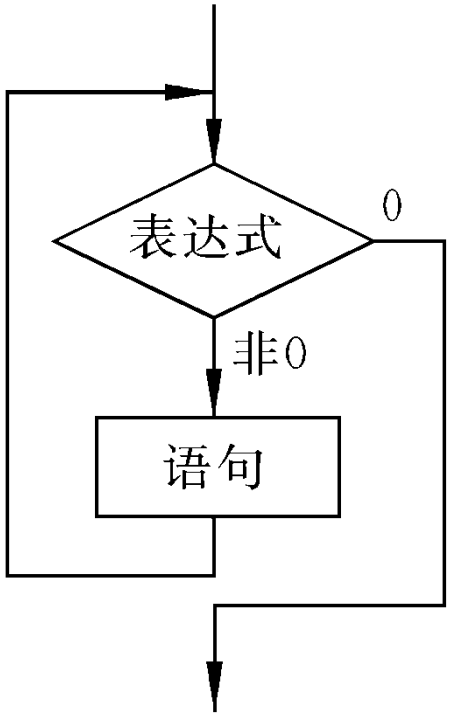
While loop

While loop repeatedly executes a statement as long as the condition is true.

```
i=1;
n=10;
while(i<n)
   i = i*2;
```

思考：i的值

```
i=1;          i is now 1
i<n成立吗？ Yes, continue
i=i*2;        i is now 2
i<n成立吗？ Yes, continue
i=i*2;        i is now 4
i<n成立吗？ Yes, continue
i=i*2;        i is now 8
i<n成立吗？ Yes, continue
i=i*2;        i is now 16
i<n成立吗？ No, exit from loop
```



While loop

```
int a = 0;
while(a < 10)
{
  // ...
  a++;
}
```

```
int a = 100;
while(a >= 10)
{
  // ...
  a--;
}
```

大小关系、
增减要匹配

While loop

Question?

What would happen?

```
while(1)
{
  // ...
}
```

infinite loop

exit the loop with

break/goto/return/

exit

Case study: while loop

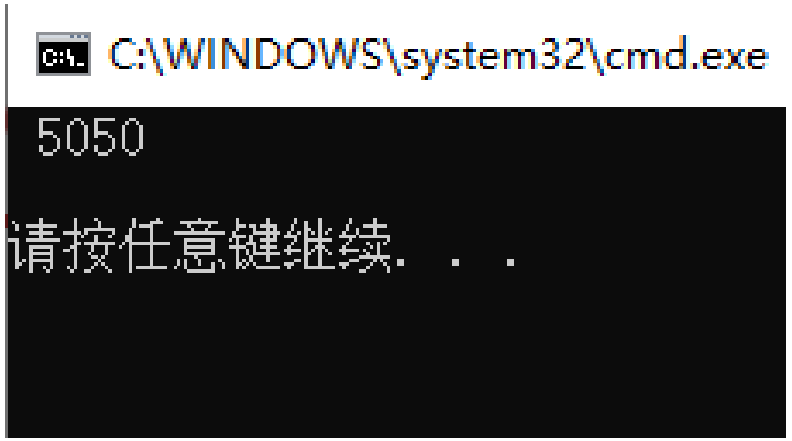
Case: sum the user's input, exit when input -1.

```
#include <stdio.h>
main ()
{
    printf("Enter an integer.\n(-1 to quit)\n");
    int input_num = 0;
    int sum = 0;
    while (input_num != -1)
    {
        scanf_s("%d", &input_num);
        sum = sum + input_num;
    }
    printf("Those integers sum to %d", sum);
}
```

Case study: while loop

Case: take the sum from 1 to 100

```
#include <stdio.h>
void main()
    {int i, sum=0;
     i=1;
     while (i<=100)
      { sum=sum+i;
        i++;
      }
     printf("%d\n", sum);
    }
```



说明： (1)循环体如果包含一个以上的语句，应该用花括弧括起来，以复合语句形式出现.

(2)在循环体中应有使循环趋向于结束的语句

Case study: while loop

Case: prints a table of squares

```
#include <stdio.h>
int main()
{
        int i, n;
        printf("This program prints a table
of squares.\n");
        printf("Enter number of entries in
table:");
        scanf("%d", &n);
        i = 1;
        while (i <= n)
        {
                printf("%10d%10d\n", i, i * i);
                i++;
        }
        return 0;
}
```

C:\WINDOWS\system32\cmd.exe

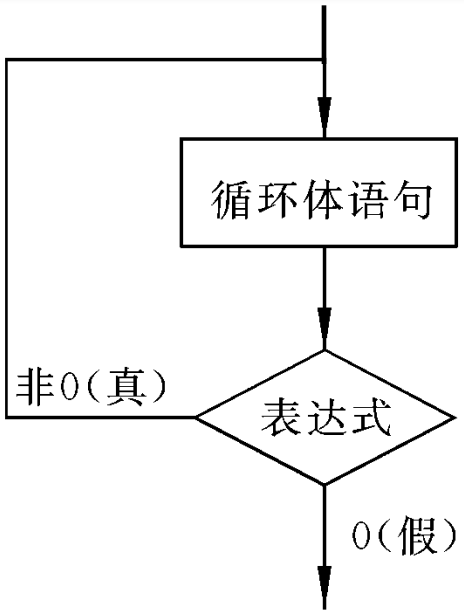
```
This program prints a table of squares.
Enter number of entries in table:5
         1         1
         2         4
         3         9
         4        16
         5        25

请按任意键继续. . .
```

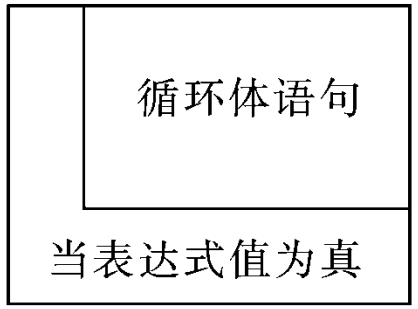
说明：使用像%10d这样的转换规范，而不是仅仅使用d，利用了当指定字段宽度时printf对数字进行右对齐。

Do-while loop

do-while loop is similar to while loop,
it guarantees to execute **at least one time**.

```
do
{
   statements;
}while( condition );
```





(b)

至少执行一次循环！

Do-while loop

```
int a = 0;
while(a < 10)
{
  // ...
  a++;
}
```

```
int a = 0;
do
{
  // ...
  a++;
}while(a < 10);
```

Do-while loop

```
int a = 0;
do
{
  // ...
  a++;
}while(a < 10);
```

<<= 建议写成左边标准格式（包含{}）

L06_do_while.c (/hom...es/CS111/Code) - GVIM

File Edit Tools Syntax Buffers Window Help

```
1 #include <stdio.h>
2
3 int main(void) {
4     int i,j;
5     i = 0;j=1;
6     do
7       printf("i=%d\n", i++);
8     while (i < 10);
"L06_do_while.c" 9L, 115B written 8,4          Top
```

L06_do_while.c (/hom...es/CS111/Code) - GVIM

File Edit Tools Syntax Buffers Window Help

```
1 #include <stdio.h>
2
3 int main(void) {
4     int i,j;
5     i = 0;j=1;
6     do printf("i=%d\n", i++); while (i < 10);
7 }
~
                                  6,16-19       All
```

Case study: do-while loop

Case: find the secrete number.

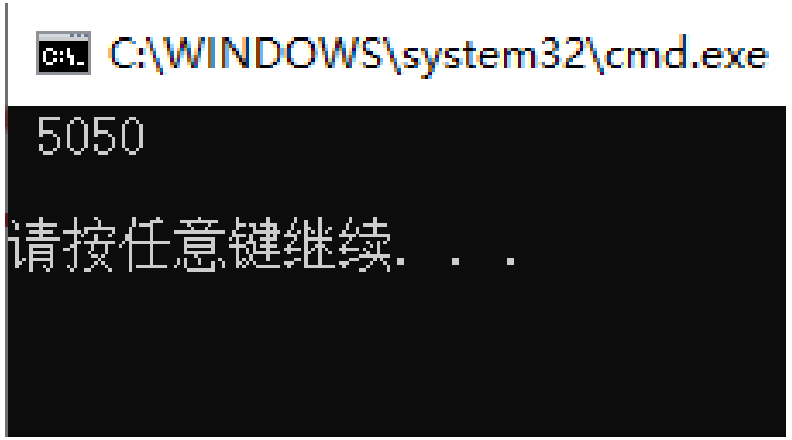
```
#include <stdio.h>
main ()
{
    int num;
    int secret_num = 13;
    do{
        printf("Please guess\n");
        scanf("%d", &num);
        if (num > secret_num) {
            printf("Secret number is smaller than %d\n", num);}
        if (num < secret_num) {
            printf("Secret number is larger than %d\n", num);}
    } while (secret_num!=num);
    printf("Got it!\n");
}
```

```
Please guess
55
Secret number is smaller than 55
Please guess
n27
Secret number is smaller than 27
Please guess
13
Got it!
```

Case study: while loop

Case: take the sum from 1 to 100

```
#include <stdio.h>
int main(void)
{
      int i, sum = 0;
      i = 1;
      do
      {
            sum = sum + i;
            i++;
      } while (i <= 100);
      printf("%d\n", sum);
      return 0;
}
```



while versus do-while

while语句和用do-while语句的比较:

在一般情况下，用while语句和用do-while语句处理同一问题时，若二者的循环体部分是一样的，它们的结果也一样。**但是如果while后面的表达式一开始就为假(0值)时，两种循环的结果是不同的。**

while versus do-while

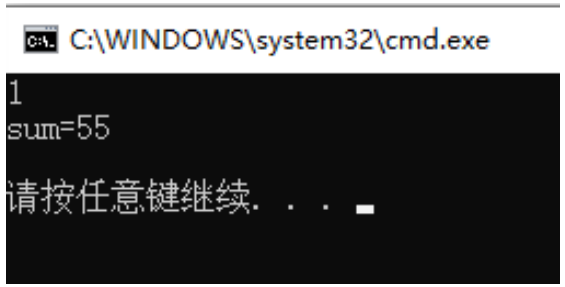
```c
#include <stdio.h>
void main()
{
     int sum = 0, i;
     scanf_s("%d", &i);
     while (i <= 10)
     {
          sum = sum + i;
          i++;
     }
     printf("sum=%d\n", sum);
}
```

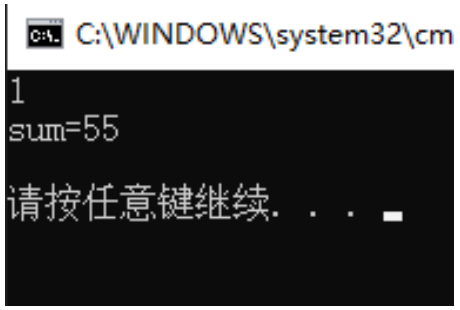
```c
#include <stdio.h>
void main()
{
     int sum = 0, i;
     scanf_s("%d", &i);
     do
     {
          sum = sum + i;
          i++;
     } while (i <= 10);
     printf("sum=%d\n", sum);
}
```

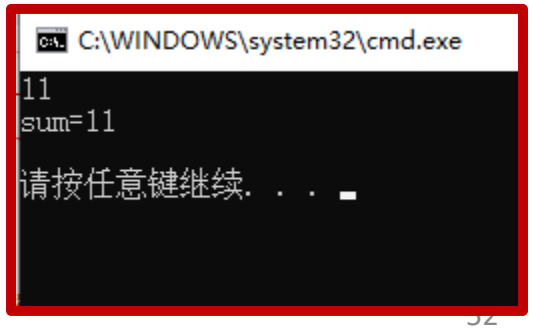
while versus do-while

```c
#include <stdio.h>
void main()
{
	int sum = 0, i;
	scanf_s("%d", &i);
	while (i <= 10)
	{
		sum = sum + i;
		i++;
	}
	printf("sum=%d\n", sum);
}
```

```c
#include <stdio.h>
void main()
{
	int sum = 0, i;
	scanf_s("%d", &i);
	do
	{
		sum = sum + i;
		i++;
	} while (i <= 10);
	printf("sum=%d\n", sum);
}
```







while versus do-while

```
#include <stdio.h>
void main()
{
     int digits = 0, n;
     scanf_s("%d", &n);
     while (n > 0)
     {
          n/=10;
          digits++;
     }
printf( “The number has %d
digits(s).\n", digits);
}
```

```
#include <stdio.h>
void main()
{
     int digits = 0, n;
     scanf_s("%d", &n);
     do
     {
          n/=10;
          digits++;
     } while (n  > 0);
     printf( “The number has
%d digits(s).\n", digits);
}
```

while versus do-while

```c
#include <stdio.h>
void main()
{
     int digits = 0, n;
     scanf_s("%d", &n);
     while (n > 0)
     {
          n/=10;
          digits++;
     }
printf( “The number has %d
digits(s).\n", digits);
}
```

```c
#include <stdio.h>
void main()
{
     int digits = 0, n;
     scanf_s("%d", &n);
     do
     {
          n/=10;
          digits++;
     } while (n  > 0);
     printf( “The number has
%d digits(s).\n", digits);
}
```

C:\Users\Zhenguo\Doc...

```
12
It has  2 digits(s).
```

C:\Users\Zhenguo\D...

```
0
It has  0 digits(s).
```

选择 C:\Users\Zhenguo...

```
12
It has  2 digits(s).
```

C:\Users\Zhenguo\D...

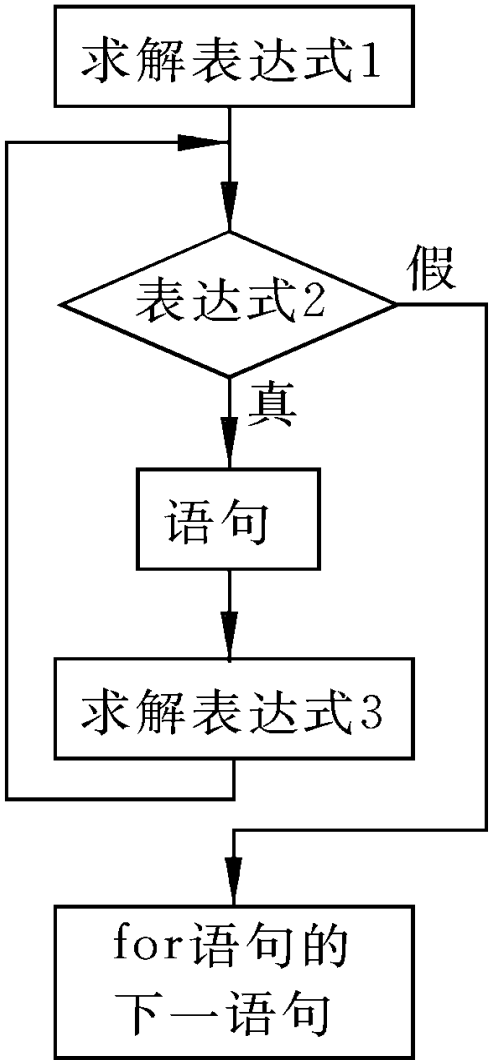
```
0
It has  1 digits(s).
```

For loop

For loop is a control structure that allows repeating the same operation (but different input values) for a specific number of times.

表达式1　　表达式2　　表达式3

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



For loop

```
for(a = 0; a < 10; a++)
{
  // ...
}
```

increment

```
for(a = 100; a >= 0; a--)
{
  // ...
}
```

decrement

For loop

计算n次

[0 n-1]

```
for(i = 0; i < n; i++)
{
  // ...
}
```

increment

[1 n]

```
for(i = 1; i <= n; i++)
{
  // ...
}
```

For loop

计算n次

[n-1 0]

```
for(i = n-1; i >=0; i--)
{
  // ...
}
```

decrement

[n 1]

```
for(i = n; i > 0; i--)
{
  // ...
}
```

For loop

易错点：

- >与<，>=与<=写反；

```
for(i = n-1; i >=0; i--)
{
  // ...
}
```

循环几次？

```
for(i = n-1; i <=0; i--)
{
  // ...
}
```

0次

For loop

易错点：

- >与<，>=与<=写反；
- <与<=，>与>=弄混；

```
for(i = n-1; i >=0; i--)
{
  // ...
}
```

循环几次？

```
for(i = n-1; i >0; i--)
{
  // ...
}
```

n-1次

Case study: for loop

Case: how to make a counter?

```
#include <stdio.h>
main ()
{
   for(int sec = 10; sec>0; sec--)
   {
      printf("%d second\n", sec);
   }
   printf("Stop!");
}
```

For versus while

```
for(int a = 0; a < 10; a++)
{
   sum = sum + a;
}
```

same

```
int a = 0;
while(a < 10)
{
   sum = sum + a;
   a++;
}
```

```
for(int a = 100; a >= 0; a--)
{
   sum = sum + a;
}
```

same

```
int a = 100;
while(a >= 0)
{
   sum = sum + a;
   a--;
}
```

For loop

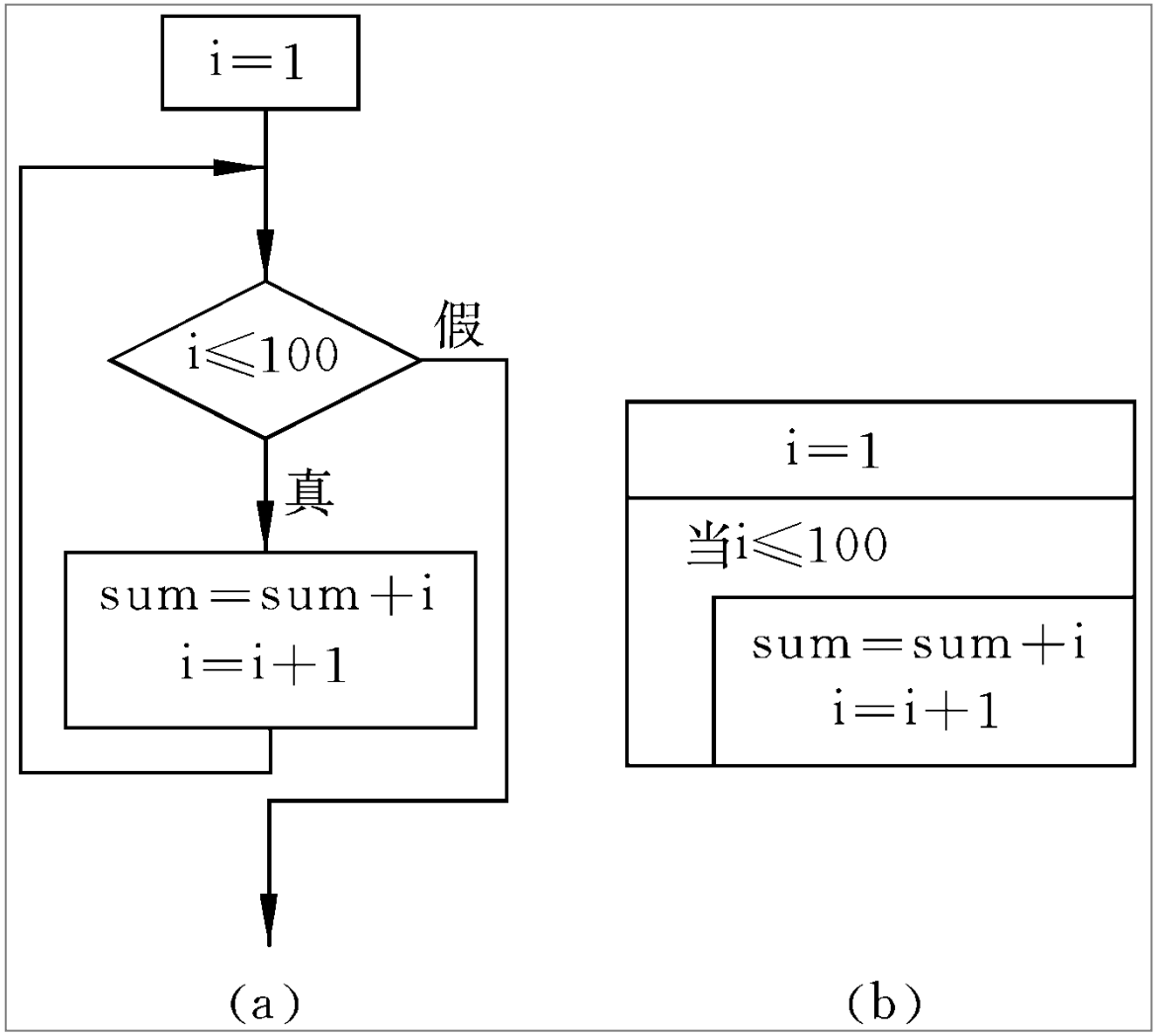
(1) for语句的一般形式中的“表达式1”可以省略，此时应在for语句之前给循环变量赋初值。注意省略表达式1时，其后的分号不能省略。如

```
for(; i<=100;i++) sum=sum+i;
```

执行时，跳过“求解表达式1”这一步，其他不变。

(2) 如果表达式2省略，即不判断循环条件，循环无终止地进行下去。也就是认为表达式2始终为真。如：

```
for(i=1; ;i++) sum=sum+i;
```



For loop

(3) 表达式3也可以省略，但此时程序设计者应另外设法保证循环能正常结束。

如：

```
for(i=1;i<=100;)
    {sum=sum+i;i++;}
```

在上面的for语句中只有表达式1和表达式2，而没有表达式3。i++的操作不放在for语句的表达式3的位置处，而作为循环体的一部分，效果是一样的，都能使循环正常结束。

For loop

(4) 可以省略表达式1和表达式3，只有表达式2，即只给循环条件。

```
for(;i<=100;)                    while(i<=100)
    {sum=sum+i; i++;}  相当于        {sum=sum+i; i++;}
```

在这种情况下，完全等同于while语句。可见for语句比while语句功能强，除了可以给出循环条件外，还可以赋初值，使循环变量自动增值等。

(5)3个表达式都可省略，如：

```
for(; ;) 语句        相当于        while(1) 语句
```

即不设初值，不判断条件(认为表达式2为真值)，循环变量不增值。无终止地执行循环体。

For loop

(6) 表达式1可以是设置循环变量初值的赋值表达式，也可以是与循环变量无关的其他表达式。如：

```
for (sum=0;i<=100;i++)
    sum=sum+i;
```

表达式3也可以是与循环控制无关的任意表达式。

(7) 可以编写主体为空的循环，例如：

```
for (d = 2; d < n && n % d != 0; d++)
/* empty loop body */
```

For loop

(8) 表达式一般是关系表达式(如i<=100)或逻辑表达式（如a<b && x<y)，但也可以是数值表达式或字符表达式，只要其值为非零，就执行循环体。

```
for(i=0;(c=getchar())!='\n';i+=c);
```

在表达式2中先从终端接收一个字符赋给c，然后判断此赋值表达式的值是否不等于'\n'(换行符)，如果不等于'\n'，就执行循环体。此for语句的循环体为空语句，把本来要在循环体内处理的内容放在表达式3中，作用是一样的。

For loop

```
#include <stdio.h>
int main()
{
    char c;
    for (; (c = getchar()) != '\n';)
    {
        printf("%c", c);
    }
    return 0;
}
```

运行情况：

Computer↙ (输入)

Computer (输出)

而不是

CCoommppuutteerr

for语句中只有表达式2，而无表达式1和表达式3。其作用是每读入一个字符后立即输出该字符，直到输入一个“换行”为止。**但是，从终端键盘向计算机输入时，是在按Enter键以后才将一批数据一起送到内存缓冲区中去的。**

For loop

（9）在C99中，for 语句中的第一个表达式可以被声明替换。如：

```
for (int i = 0; i < n; i++)
```

变量i不需要在这条语句之前声明。(事实上，如果i的声明已经存在，该语句将创建一个新版本的i，仅在循环中使用。)由for语句声明的变量不能在循环体之外访问(我们说它在循环之外是不可见的):

```
for (int i = 0; i < n; i++)
{printf( “%d” , i);  /*legal ; i is visible inside loop*/}
printf("%d", i);       /*****WRONG*****/
```

For loop

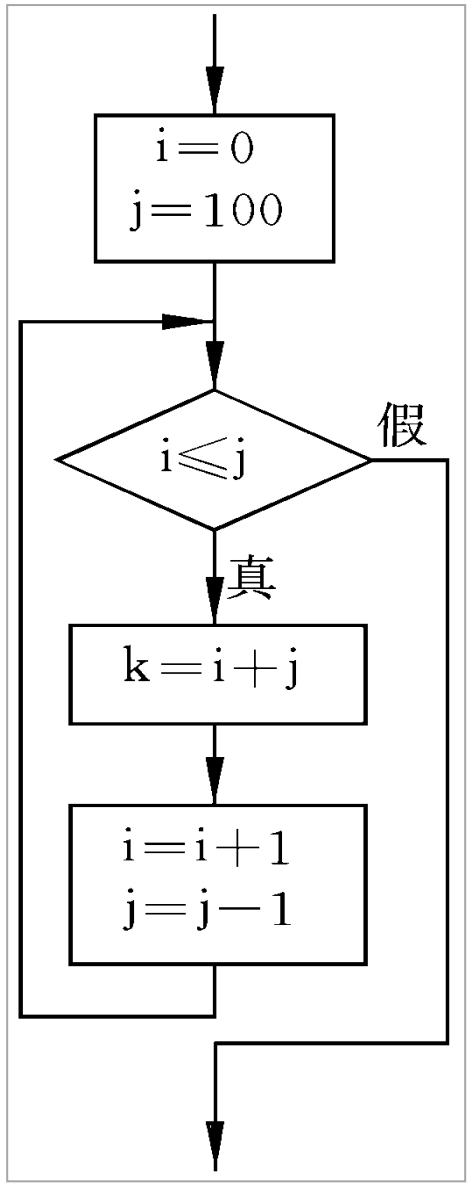
(10)表达式1和表达式3可以是一个简单的表达式，也可以是逗号表达式，即包含一个以上的简单表达式，中间用逗号间隔。如

```
for(sum=0, i=1;i<=100;i++) sum=sum+i;
```

或

```
for(i=0, j=100;i<=j;i++, j--) k=i+j;
```

表达式1和表达式3都是逗号表达式，各包含两个赋值表达式，即同时设两个初值，使两个变量增值.



The Comma Operator(逗号表达式)

```
for(sum=0, i=1;i<=100;i++) sum=sum+i;
```

逗号运算符有两个属性：

- 它保证了被它分隔的表达式从左往右求值（换言之，逗号是一个序列点，所以逗号左侧项的所有副作用都在程序执行逗号右侧项之前发生）。
- 其次，整个逗号表达式的值是右侧项的值。

  x = (y = 3, (z = ++y + 2) + 5);

  效果是：先把3赋给y，递增y为4，然后把4加2之和（6）赋给z，接着加上5，最后把结果11赋给 x。

逗号运算符优先级低于其他运算符

Nested loops

C allows using one loop inside another loop.

```
while ()
{
   // xxxx
   while()
   {
      // xxxx
   }
}
```

```
do
{
   // xxxx
   do
   {
      // xxxx
   }while();
}while();
```

```
for ( ; ;  )
{
   for ( ; ;  )
   {
      // xxxx
   }
}
```

Nested loops

C allows using one loop inside another loop.

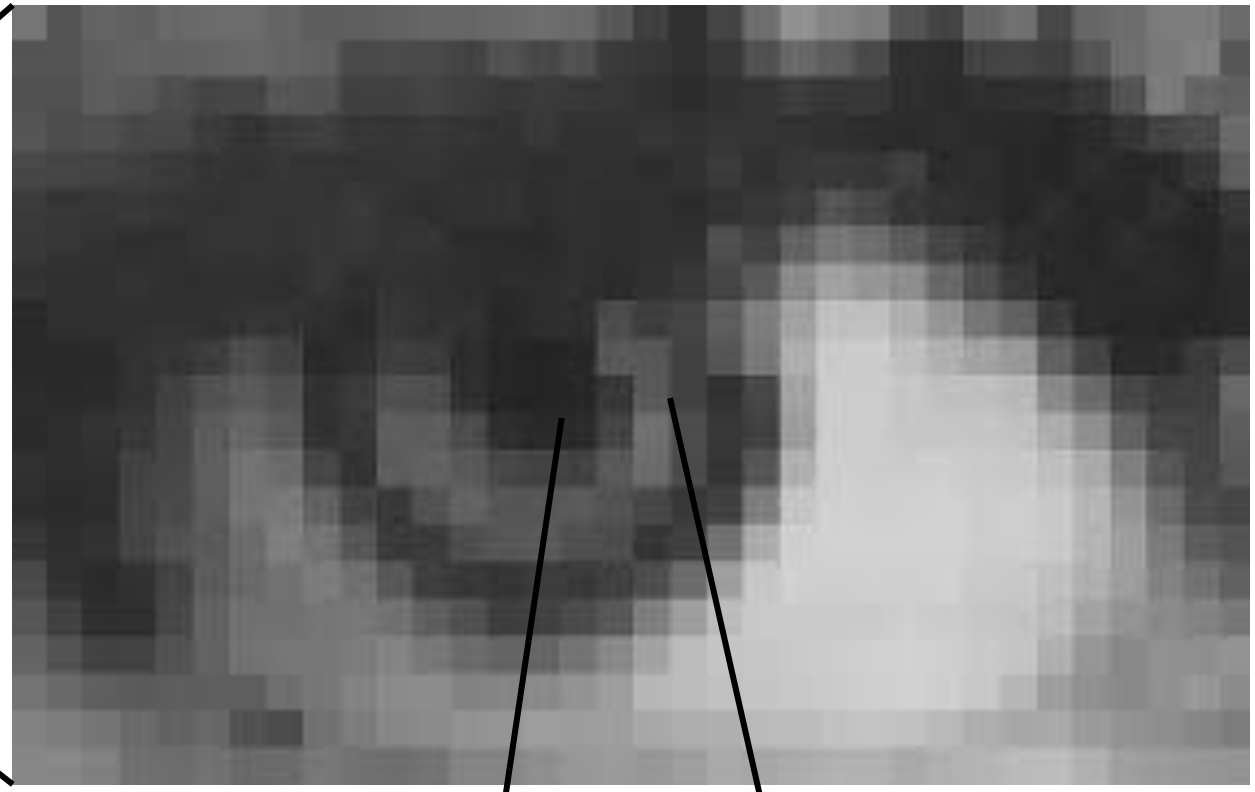
```
while ()
{
   // xxxx
   do
   {
      // xxxx
   }while()
}
```

```
do
{
   // xxxx
   for (; ;)
   {
      // xxxx
   }
}while();
```

```
for ( ; ;  )
{
   while()
   {
      // xxxx
   }
}
```

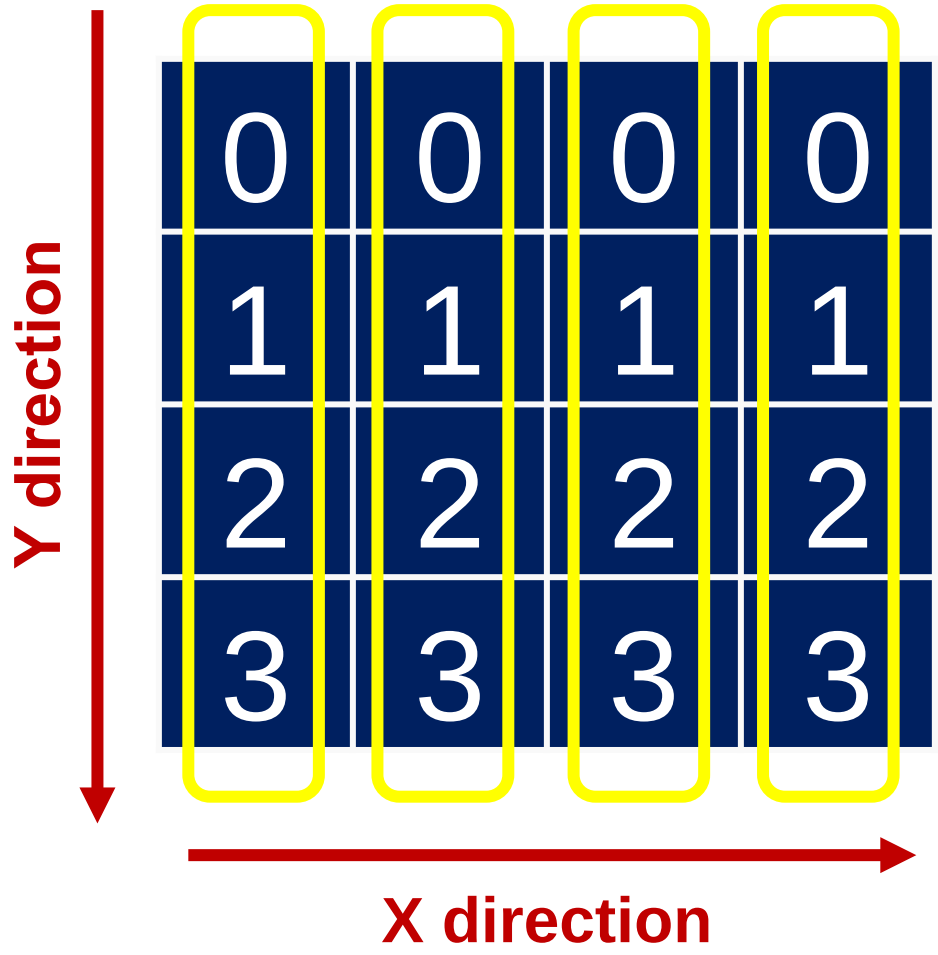
Nested loops

Nested loops can create suchmatrix!



10	20	20	26
20	41	10	80
60	22	16	84

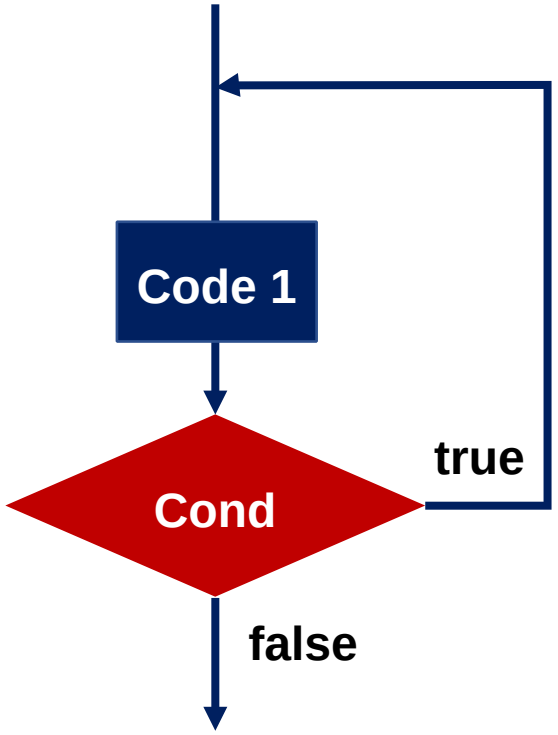
Nested loops



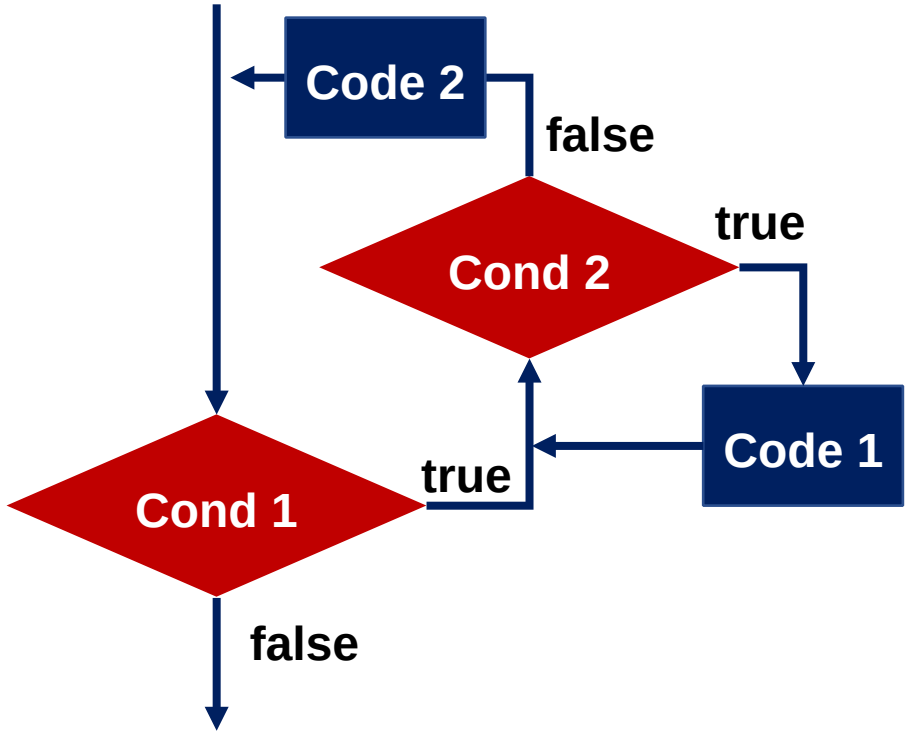

```
for (int x = 0; x < 4; x++)
{
        for (int y = 0; y < 4; y++)
        {
                // fill y at <x, y>
        }
}
```

Overview of loops

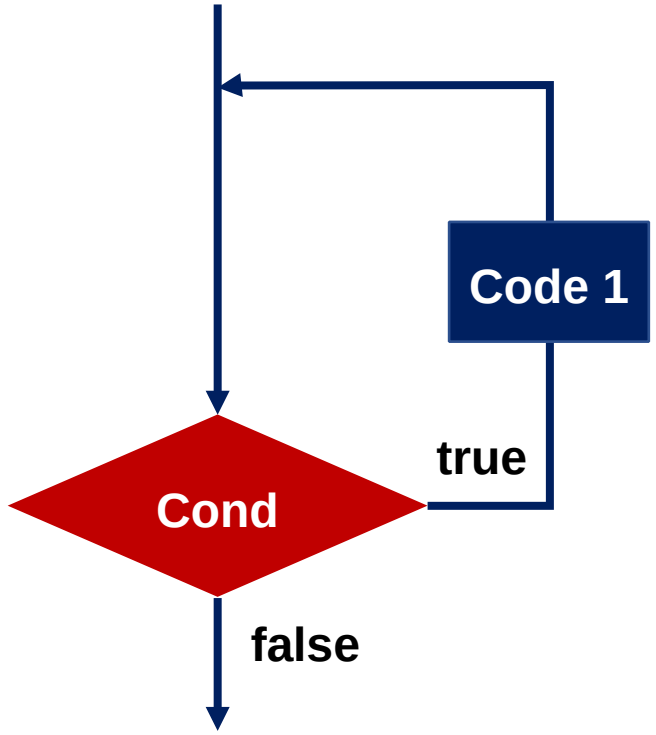
do-while loop



nested for loop



for/while loop



Same task in 3 looping formats

Calculate the sum = 1+2+..100

For loop

```
#include <stdio.h>
int main()
{
    int sum = 0;
    for (int i=1;i<=100;i++)
    {
        sum += i;
    }
    printf("sum=%d\n", sum);
    return 0;
}
```

While loop

```
#include <stdio.h>
int main()
{
    int sum = 0, i = 1;
    while(i <= 100) {
        sum += i;
        i++;
    }
    printf("sum=%d\n", sum);
    return 0;
}
```

Do-while loop

```
#include <stdio.h>
int main()
{
    int sum = 0, i = 1;
    do {
        sum += i;
        i++;
    } while (i <= 100);
    printf("sum=%d\n", sum);
    return 0;
}
```

Same task in 3 looping formats

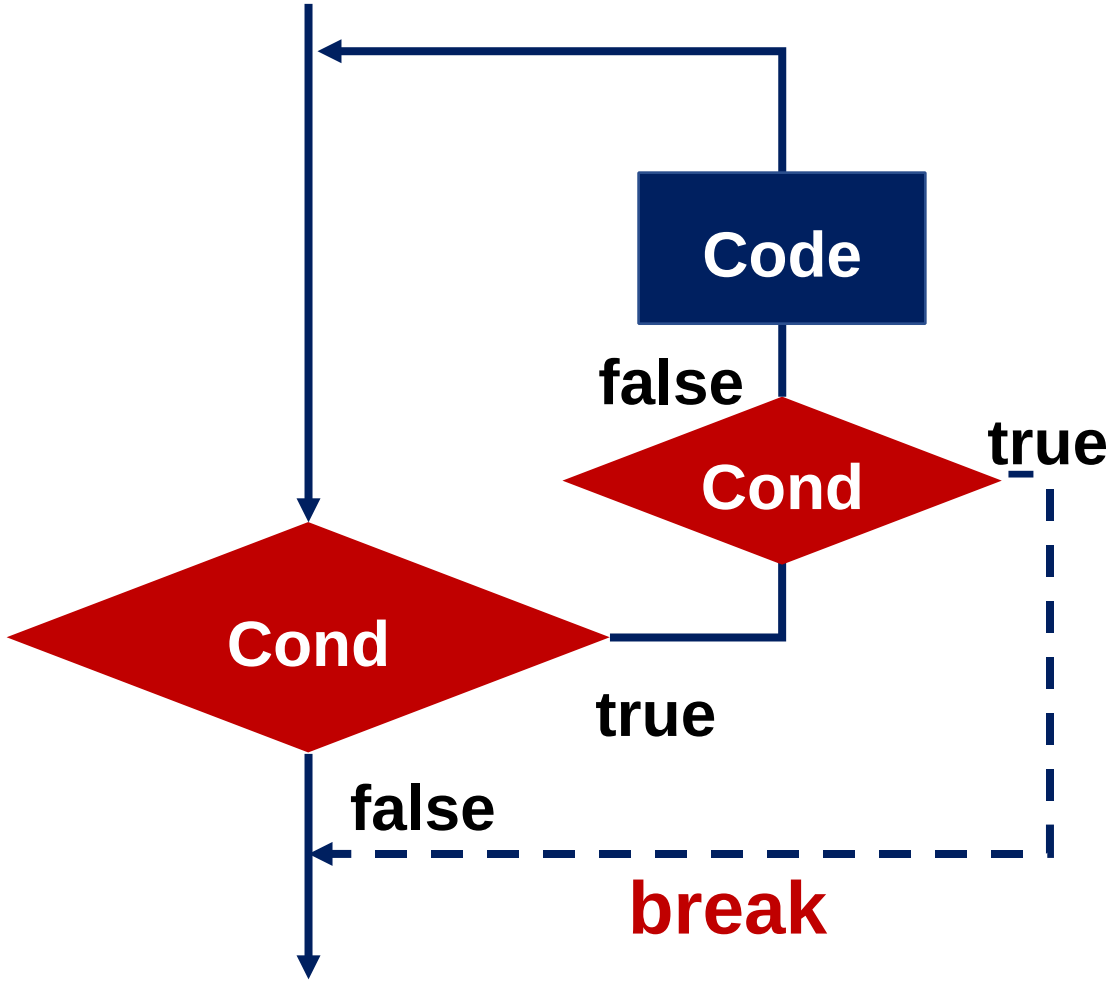
- 在**while**循环和**do-while**循环中，只在**while**后面的括号内指定循环条件，因此为了使循环能正常结束，应在循环体中包含使循环趋于结束的语句(如i++，或**i=i+1**等)。**for**循环可以在表达式3中包含使循环趋于结束的操作，甚至可以将循环体中的操作全部放到表达式3中。因此**for**语句的功能更强，凡用**while**循环能完成的，用**for**循环都能实现。

- 用**while**和**do-while**循环时，循环变量初始化的操作应在**while**和**do-while**语句之前完成。而**for**语句可以在表达式1中实现循环变量的初始化。

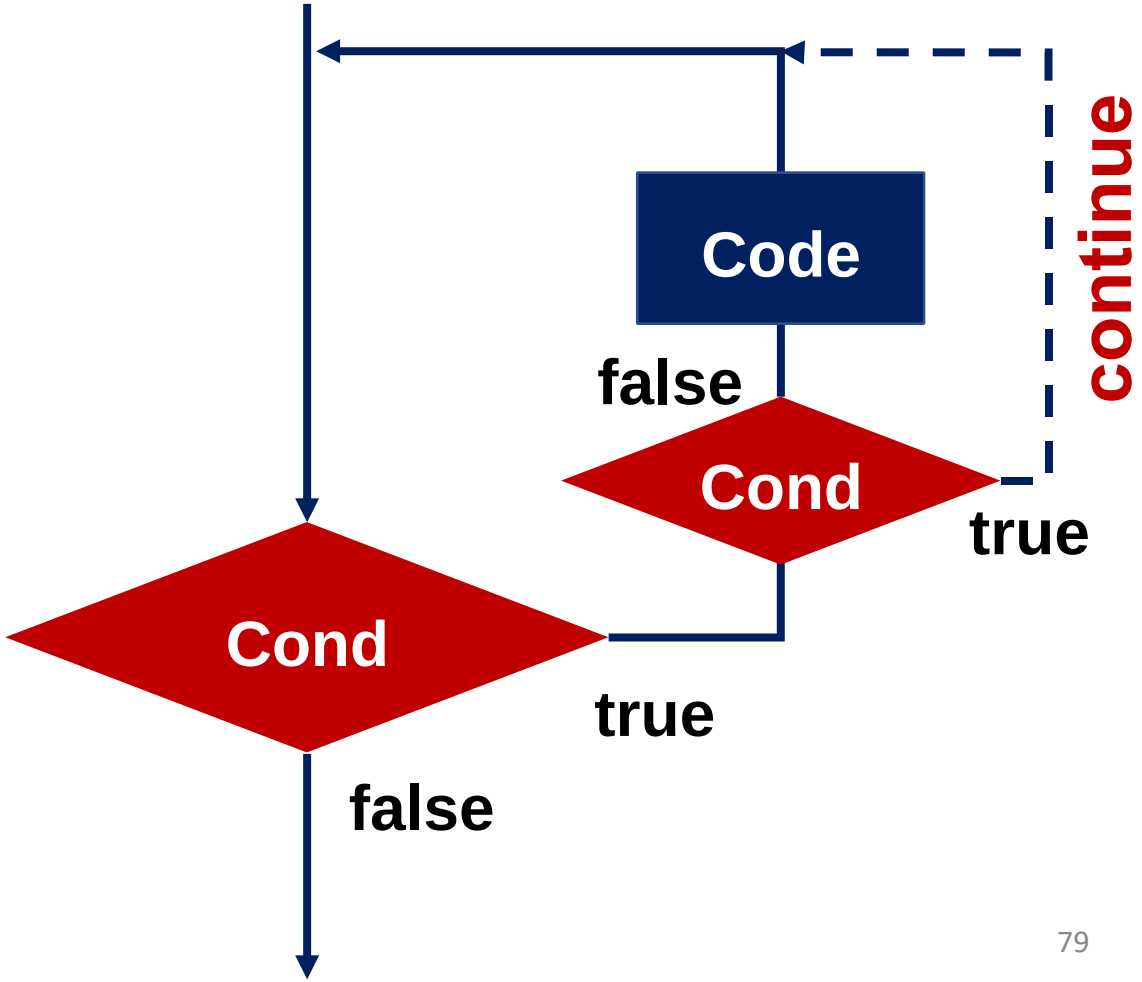
- **while**循环、**do-while**循环和**for**循环，可以用**break**语句跳出循环，用**continue**语句结束本次循环(**break**语句和**continue**语句见下节)。而对用**goto**语句和**if**语句构成的循环，不能用**break**语句和**continue**语句进行控制。

Break and continue

Break loop



Continue loop



Break statement

Break terminates the loop when meeting the criterion.

```
for ( init; condition; increment )
{
   if (statement)
      break;
}
```

Break is needed for brute-force searching!

break语句不能用于循环语句和switch语句之外的任何其他语句中！

Case study: break statement

Case: output the smallest integer divisible by 17 but greater than 500

```
#include <stdio.h>
int main ()
{
    int num = 500;
    while (1){
        if (num % 17 == 0) {
            printf("%d is the smallest integer divisible by 17.", num);
            break;
        }
        num++;
    }
    return 0;
}
```

Break statement

Break terminates the loop when meeting the criterion.

```
while(...){
        switch(...){
                ...
                break;
                ...
        }
}
```

The break statement **can escape only one level of nesting.** The break statement transfers control out of the switch statement. but not out of the while loop.

Continue statement

Continue forces execution to the next iteration, skipping the code in between.

```
for ( init; condition; increment )
{
   if (condition)
      continue;
   // …
}
```

Continue can skip unwanted rounds in looping!

continue仅限于循环中

Case study: continue statement

Case: calculate the average score of 5 students with valid scores in [0, 100].

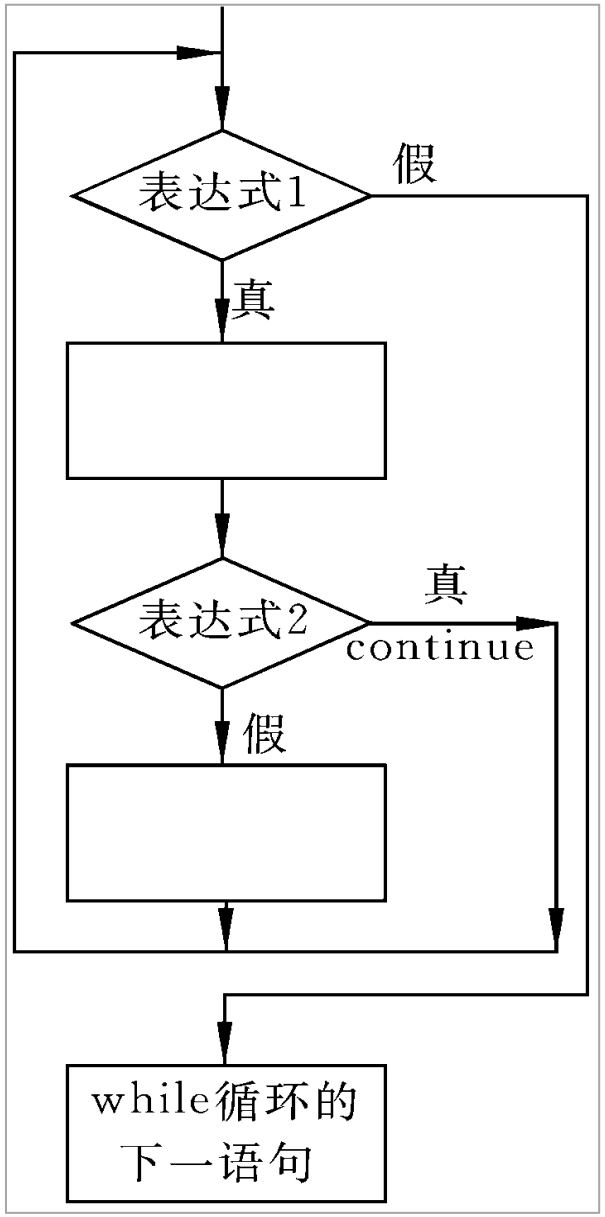
```
#include <stdio.h>
main ()
{
    int number = 0, scores = 0,sum =0;
    printf("Input the score\n");
    for (int i = 0;i < 5;i++) {
        scanf ("%d", &scores);
        if (scores < 0 || scores >100) {
            printf("Not valid!\n");
            continue;
        }
        number++;sum += scores;
    }
    printf("There are %d students with valid scores.\nThe mean is %f\n",
number,sum * 1.0/ number);
}
```

```
Input the score
90
-3
Not valid!
98
120
Not valid!
87
There are 3 students with valid scores.
The mean is 91.666667
```

Break versus continue

continue语句只结束本次循环，而不是终止整个循环的执行。

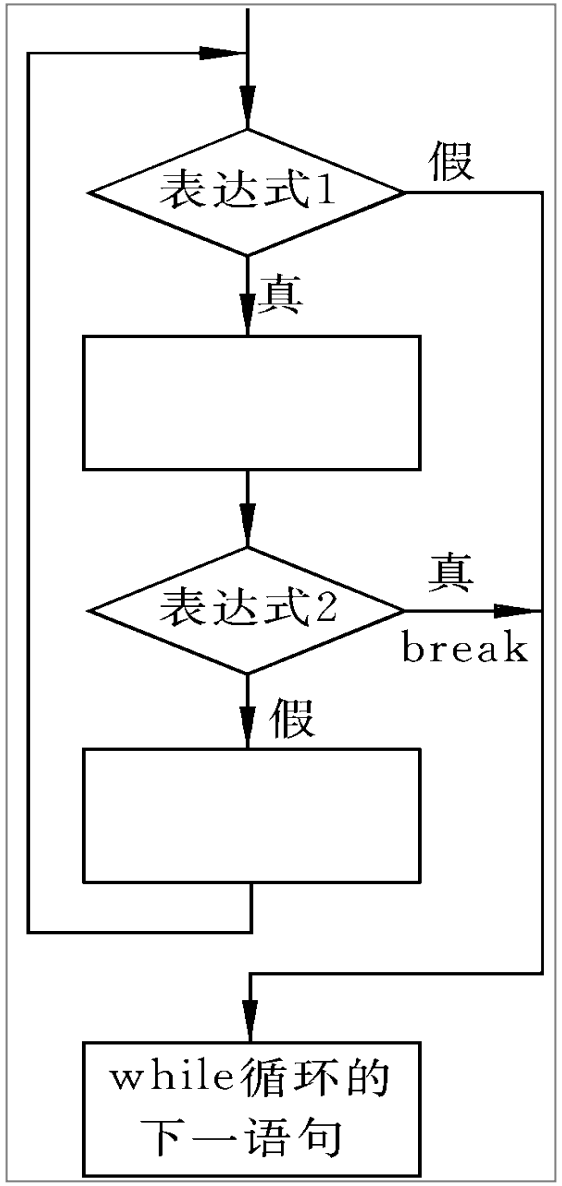
```
while（表达式1） for
{ …
      if（表达式2） continue;
       …
 }
```



Break versus continue

break语句则是结束整个循环过程，不再判断执行循环的条件是否成立。

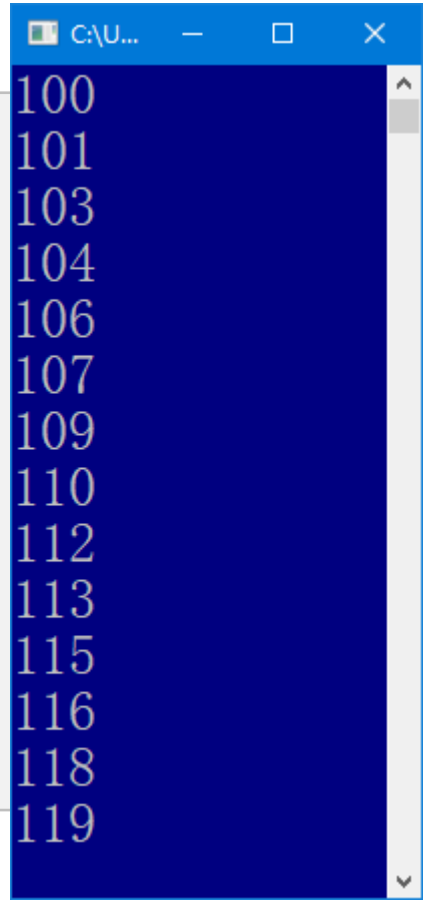
```
while（表达式1） for
{ …
    if（表达式2） break;
…
 }
```



Case

Case: output the numbers between 100 and 200 that are not divisible by 3.

```
#include <stdio.h>
int main(void)
{
        int n;
        for (n = 100; n <= 200; n++)
        {
                if (n % 3 == 0)
                        continue;
                printf("%d\n", n);
        }
    return 0;
}
```

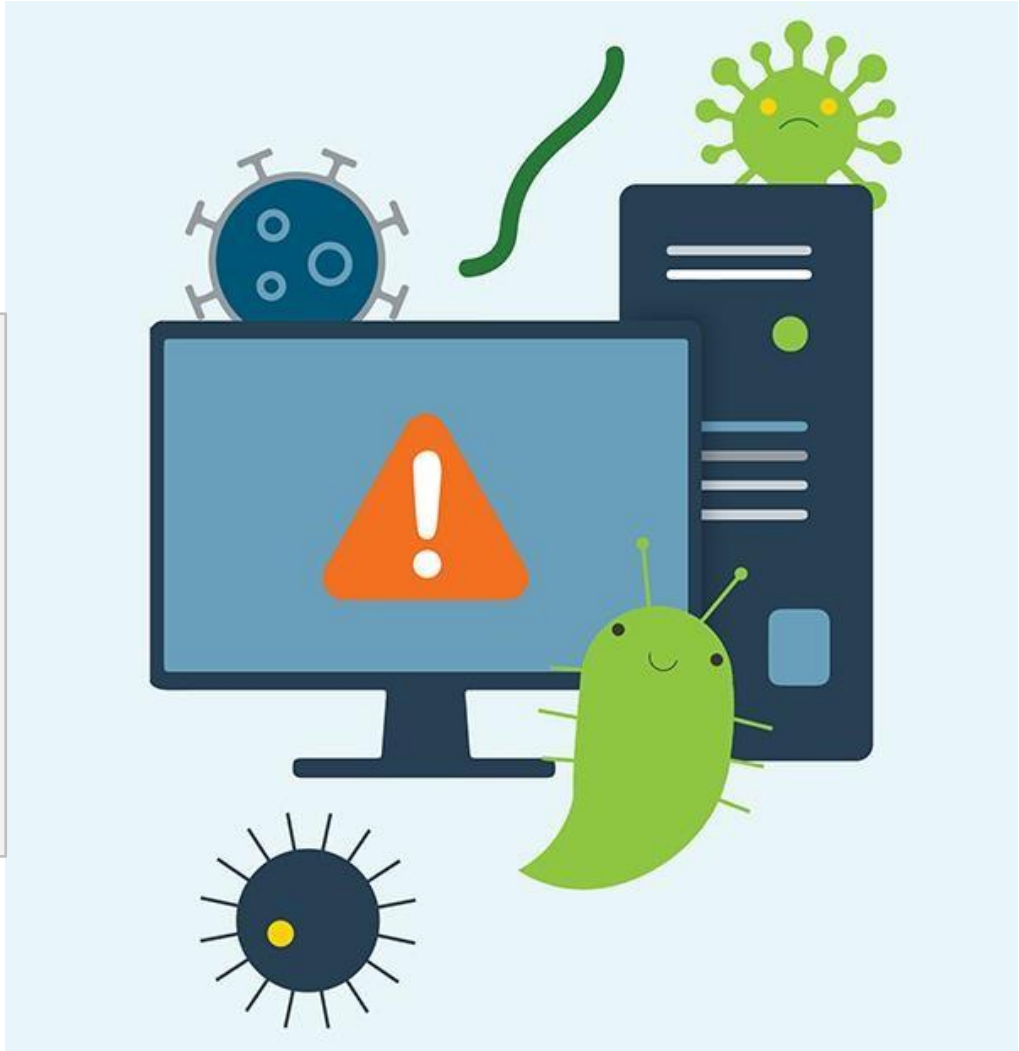


说明： 当n能被3整除时，执行continue语句，结束本次循环(即跳过printf函数语句)，只有n不能被3整除时才执行printf函数。

Infinite loop - Virus!

NOTE: A loop becomes infinite if a condition never becomes false!

```
#include <stdio.h>
int main (void)
{
    for( ; ; ) // while(true)
        { printf("This loop will run
    forever.\n");
    }
    return 0;
}
```



goto

goto跳到**同一**函数中任何有标号（identifier）的地方。

```
…
identifier : statement
…
goto identifier;
```

goto

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    goto done;
done:
if (d < n)
  printf("%d is divisible by
  %d\n", n, d);
else
  printf("%d is prime\n", d);
```

其他方案？

goto

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    break;

if (d < n)
  printf("%d is divisible by
  %d\n", n, d);
else
  printf("%d is prime\n", d);
```

其他方案？

不是所有的goto 都可以用break代替

goto

```
while (…){
  switch (…){
    …
    goto loop_done
    …
  }
}
loop_done: …
```

其他方案？

不是所有的goto 都可以用break代替

- 少用goto（容易混乱）
- 与continue, break, exit, return等混合使用

Summary

- Two major workflow controls provided in C: decision-making and looping
- Two types of statement for making decisions: `if-else` and `switch`, if-else is more popular, switch is for equality check
- Two types of statement for looping: `for` loop and `while/do-while` loop, both are essentially the same
- `Break`, `continue` and `goto` statements can be used to influence loops, jump out from the loop or skip specific loops
- Time to write you C program to control workflows