第3讲：分支和循环（上）

目录

1. if语句
2. 关系操作符
3. 逻辑操作符：&&，||，!
4. switch语句
5. while循环
6. for循环
7. do-while循环
8. break和continue语句
9. 循环的嵌套
10. goto语句

正文开始

C语言是结构化的程序设计语言，这里的结构指的是**顺序结构、选择结构、循环结构**，C语言是能够实现这三种结构的，其实我们如果仔细分析，我们日常所见的事情都可以拆分为这三种结构或者这三种结构的组合。

我们可以使用`if`、`switch`实现分支结构，使用`for`、`while`、`do while`实现循环结构。

1. if语句

1.1 if

if 语句的语法形式如下:

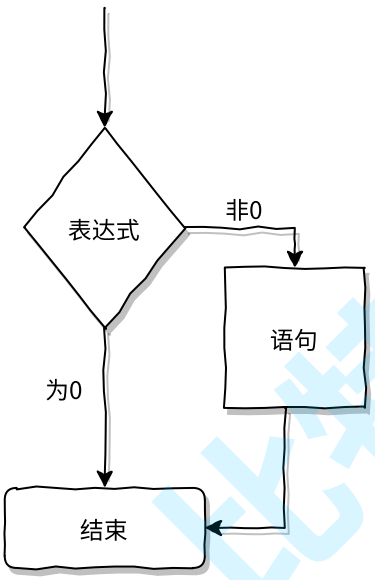
```
1 if ( 表达式 )
2     语句
```

表达式成立（为真），则语句执行，表达式不成立（为假），则语句不执行

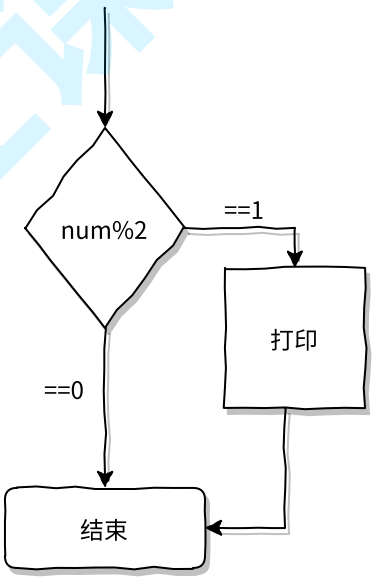
在C语言中，0为假，非0表示真，也就是表达式的结果如果是0，则语句不执行，表达式的结果如果是不是0，则语句执行。

例子：输入一个整数，判断是否为奇数

```c
#include <stdio.h>
int main()
{
    int num = 0;
    scanf("%d", &num);
    if(num % 2 == 1)
        printf("%d 是奇数\n", num);
    return 0;
}
```



if语句执行流程



奇数判断

1.2 else

如果一个数不是奇数，那就是偶数了，如果任意一个整数，我们要清楚的判断是奇数还是偶数怎么表示呢?

这里就需要`if...else...`语句了，语法形式如下：

```c
if ( 表达式 )
    语句1
else
```

```
    语句2
```

例子：输入一个整数，判断是否为奇数，如果是奇数打印是奇数，否则打印数偶数。

```c
#include <stdio.h>
int main()
{
    int num = 0;
    scanf("%d", &num);
    if(num % 2 == 1)
        printf("%d 是奇数\n", num);
    else
        printf("%d 是偶数\n", num);
    return 0;
}
```

练习：输入一个年龄，>= 18岁就输出：成年，否则就输出：未成年

```c
#include <stdio.h>

int main()
{
    int age = 0;
    scanf("%d", &age);
    if(age>=18)
        printf("成年\n");
    else
        printf("未成年\n");
    return 0;
}
```

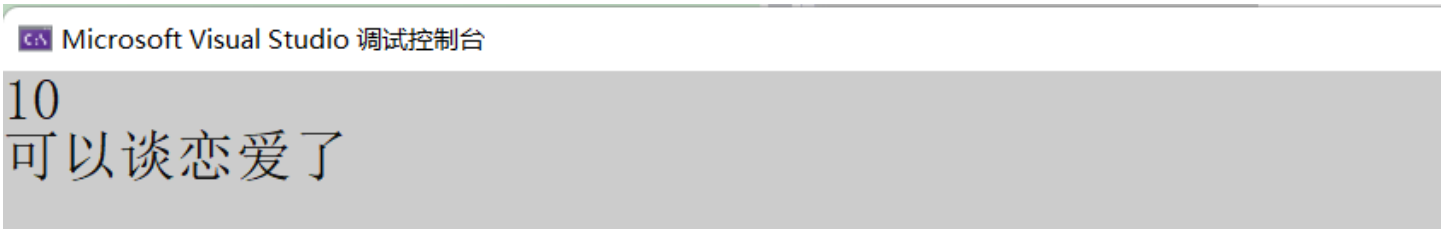
1.3 分支中包含多条语句

默认在 `if` 和 `else` 语句中默认都只控制一条语句，比如：

```c
#include <stdio.h>
int main()
{
    int age = 0;
    scanf("%d", &age);
    if(age >= 18)
        printf("成年了\n");
```

```
8           printf("可以谈恋爱了\n");
9       return 0;
10  }
```

上面的代码，你会发现输入的值不管是>=18 还是小于18，"可以交女朋友了" 都会打印在屏幕上。



这是因为 `if` 语句只能控制一条语句，就是 `printf("成年了\n");`，if语句为真，则打印成年了，if语句为假，则不打印，对于 `printf("可以谈恋爱了\n");` 是独立存在的，不管if语句的条件的真假，都会被执行。那如果我们要if语句同时控制2条语句，怎么办呢？那就要使用 `{}` 将代码括起来，else 后也可以跟上大括号。如下：

```
1   #include <stdio.h>
2   int main()
3   {
4       int age = 0;
5       scanf("%d", &age);
6       if(age >= 18) //if 后使用{} 控制多条语句-这个块也叫：程序块，或者复合语句
7       {
8           printf("成年了\n");
9           printf("可以交女朋友了\n");
10      }
11      return 0;
12  }
13
14
15  #include <stdio.h>
16  int main()
17  {
18      int age = 0;
19      scanf("%d", &age);
20      if(age >= 18) //if 后使用{} 控制多条语句-这个块也叫：程序块，或者复合语句
21      {
```

```
22            printf("成年了\n");
23            printf("可以交女朋友了\n");
24        }
25        else //else 后使用{}控制多条语句-这个块也叫：程序块，或者复合语句
26        {
27            printf("未成年\n");
28            printf("不可以早恋哦\n");
29        }
30        return 0;
31    }
```

1.4 嵌套if

在 `if else` 语句中， `else` 可以与另一个 `if` 语句连用，构成多重判断。

比如：要求输入一个整数，判断输入的整数是0，还是正数或者负数。请看如下代码：

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int num = 0;
6      scanf("%d", &num);
7      if(num == 0)
8          printf("输入的数字是0\n");
9      else if(num > 0) //这里的if 相当于嵌套在els语句中，形成了嵌套结构
10         printf("输入的数字是正数\n");
11     else
12         printf("输入的数字是负数\n");
13
14     return 0;
15 }
```

上图中橙色背景的代码就是嵌套在前面的 `else` 子句中的，构成了嵌套的if语句。

再比如：要求输入一个整数，如果是正数，再判断是奇数还是偶数，并输出；如果不是正数则输出不是整数。

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int num = 0;
6      scanf("%d", &num);
```

```
    if(num>0)
    {
        if(num%2 == 0)
            printf("偶数\n");
        else
            printf("奇数\n");
    }
    else
    {
        printf("输入的值是负数\n");
    }
    return 0;
}
```

上面的代码中橙色背景的代码也是嵌套在 `if` 语句中的，构成了嵌套的 `if` 语句。

以上就是嵌套if语句的语法，有了嵌套if语句，就可以完成更多复杂逻辑的判断。

练习：

```
输入一个人的年龄
如果年龄<18岁，打印"少年"
如果年龄在18岁至44岁打印"青年"
如果年龄在45岁至59岁打印"中老年"
如果年龄在60岁至89岁打印"老年"
如果90岁以上打印"老寿星"
```

参考代码：

```
//方法1
#include <stdio.h>
int main()
{
    int age = 0;
    scanf("%d", &age);
    if(age<18)
        printf("少年\n");
    else if(age<=44)
        printf("青年\n");
    else if(age<=59)
        printf("中老年\n");
```

```c
    else if(age<=90)
        printf("老年\n");
    else
        printf("老寿星\n");
    return 0;
}


//带上大括号更容易看明白
#include <stdio.h>
int main()
{
    int age = 0;
    scanf("%d", &age);
    if(age<18)
    {
        printf("少年\n");
    }
    else
    {
        if(age<=44)
        {
            printf("青年\n");
        }
        else
        {
            if(age<=59)
            {
                printf("中老年\n");
            }
            else
            {
                if(age<=90)
                    printf("老年\n");
                else
                    printf("老寿星\n");
            }
        }
    }
    return 0;
}
```

1.5 悬空else问题

如果有多个 `if` 和 `else` ，可以记住这样一条规则， `else` 总是跟最接近的 `if` 匹配。

我们首先从下面代码开始

```c
#include <stdio.h>
int main()
{
    int a = 0;
    int b = 2;
    if(a == 1)
        if(b == 2)
            printf("hehe\n");
    else
        printf("haha\n");
    return 0;
}
```

程序运行的结果是啥?

很多初学者，上来以判断 `a` 是 0，不等于 1，那就执行 `else` 子句，打印 `haha`

但是当你去运行代码，输出的结果是：啥都不输出。

```c
#include <stdio.h>
int main()
{
    int a = 0;
    int b = 2;
    if (a == 1)
        if (b == 2)
            printf("hehe\n");
    else
        printf("haha\n");
    return 0;
}
```

```
选择 Microsoft Visual Studio 调试控制台

D:\code\2023\test\x64\Debug\test.exe (进程 226
按任意键关闭此窗口. . .
```

为什么呢?

这就是悬空 `else` 的问题，如果有多个 `if` 和 `else` ，可以记住这样一条规则， `else` 总是跟最接近的 `if` 匹配。

上面的代码排版，让 `else` 和第一个 `if` 语句对齐，让我们以为 `else` 是和第一个if匹配的，当 `if` 语句不成立的时候，自然想到的就是执行 `else` 子句，打印 `haha` ，但实际上 `else` 是和第二个 `if` 进行匹配的，这样后边的 `if...else` 语句是嵌套在第一个 `if` 语句中的，如果第一个 `if` 语句就不成立，嵌套 `if` 和 `else` 就没机会执行了，最终啥都不打印。

如果代码改成下面这样就更加容易理解了

```c
#include <stdio.h>
int main()
{
    int a = 0;
    int b = 2;
    if(a == 1)
    {
        if(b == 2)
            printf("hehe\n");
        else
            printf("haha\n");
    }
    return 0;
}
```

或者如果我们希望else确实和第一个if匹配，可以这样修改代码：

```c
#include <stdio.h>
int main()
{
    int a = 0;
    int b = 2;
    if(a == 1)
    {
        if(b == 2)
            printf("hehe\n");
    }
    else
    {
         printf("haha\n");
    }
    return 0;
}
```

只要带上适当的大括号，代码的逻辑就会更加的清晰，所以大家以后在写代码的时候要注意括号的使用，让代码的可读性更高。

2. 关系操作符

C 语言用于比较的表达式，称为 “**关系表达式**” （relational expression），里面使用的运算符就称为“关系运算符” （relational operator），主要有下面6个。

- `>` 大于运算符

- `<` 小于运算符
- `>=` 大于等于运算符
- `<=` 小于等于运算符
- `==` 相等运算符
- `!=` 不相等运算符

下面是一些例子。

```
a == b;
a != b;
a < b;
a > b;
a <= b;
a >= b;
```

关系表达式通常返回 `0` 或 `1`，表示真假。

C 语言中，`0` 表示假，所有非零值表示真。比如，`20 > 12` 返回 `1`，`12 > 20` 返回 `0`。

关系表达式常用于 `if` 或 `while` 结构。

```
if (x == 3) {
  printf("x is 3.\n");
}
```

注意，相等运算符 `==` 与赋值运算符 `=` 是两个不一样的运算符，不要混淆。有时候，可能会不小心写出下面的代码，它可以运行，但很容易出现意料之外的结果。

```
if (x = 3) ...
```

上面示例中，原意是 `x == 3`，但是不小心写成 `x = 3`。这个式子表示对变量 `x` 赋值 `3`，它的返回值为 `3`，所以 `if` 判断总是为真。

为了防止出现这种错误，有的程序员喜欢将变量写在等号的右边。

```
if (3 == x) ...
```

这样的话，如果把 `==` 误写成 `=`，编译器就会报错。

```
/* 报错 */
if (3 = x) ...
```

另一个需要避免的错误是：**多个关系运算符不宜连用**。

```
i < j < k
```

上面示例中，连续使用两个小于运算符。这是合法表达式，不会报错，但是通常达不到想要的结果，即不是保证变量 `j` 的值在 `i` 和 `k` 之间。因为关系运算符是从左到右计算，所以实际执行的是下面的表达式。

```
(i < j) < k
```

上面式子中，`i < j` 返回 `0` 或 `1`，所以最终是 `0` 或 `1` 与变量 `k` 进行比较。如果想要判断变量 `j` 的值是否在 `i` 和 `k` 之间，应该使用下面的写法。

```
i < j && j < k
```

比如：我们输入一个年龄，如果年龄在18岁~36岁之间，我们输出青年。

如果我们这样写

```
#include <stdio.h>
int main()
{
    int age = 0;
    scanf("%d", &age);
    if(18<=age<=36)
    {
        printf("青年\n");
    }
    return 0;
}
```

当我们输入10的时候，依然输出青年，如下图：

Microsoft Visual Studio 调试控制台

```
10
青年
```

这是因为，我们先拿18和age中存放的10比较，表达式18<=10为假，`18<=age`的结果是0，再拿0和36比较，0<=36为真，所以打印了`青年`，所以即使当age是10的时候，也能打印`青年`，逻辑上是有问题，这个代码应该怎么写呢？

```c
#include <stdio.h>
int main()
{
    int age = 0;
    scanf("%d", &age);
    if(age>=18 && age<=36)
    {
        printf("青年\n");
    }
    return 0;
}
```

以上就是关于操作符，我们需要掌握的，剩下的只要按照字面意思理解使用就行，没有特别注意的。

3. 逻辑操作符：&&，||，!

逻辑运算符提供逻辑判断功能，用于构建更复杂的表达式，主要有下面三个运算符。

- `!`：逻辑取反运算符（改变单个表达式的真假）。
- `&&`：与运算符，就是并且的意思（两侧的表达式都为真，则为真，否则为假）。
- `||`：或运算符，就是或者的意思（两侧至少有一个表达式为真，则为真，否则为假）。

a	!a
非0	0
0	1

逻辑反操作

a	b	a&&b
非0	非0	1
非0	0	0
0	非0	0
0	0	0

逻辑与

a	b	a||b
非0	非0	1
非0	0	1
0	非0	1
0	0	0

逻辑或

> **注：**C语言中，非0表示真，0表示假

3.1 逻辑取反运算符

比如，我们有一个变量叫 `flag` ,如果flag为假，要做一个什么事情，就可以这样写代码：

```
1 if(!flag)
2 {
3     printf("do something\n");
4 }
```

如果 `flag` 为真， `!flag` 就是假，如果 `flag` 为假， `!flag` 就是真

所以上面的代码的意思就是 `flag` 为假，执行if语句中的代码。

3.2 与运算符

`&&` 就是与运算符，也是并且的意思， `&&` 是一个双目操作符，使用的方式是 `a&&b` , `&&` 两边的表达式都是真的时候，整个表达式才为真，只要有一个是假，则整个表达式为假。

比如：如果我们说月份是3月到5月，是春天，那使用代码怎么体现呢?

```
1 int month = 0;
2 scanf("%d", &month);
3 if(month >= 3 && month <= 5)
4 {
5     printf("春季\n");
6 }
```

这里表达的意思就是month既要大于等于3，又要小于等于5，必须同时满足。

3.3 或运算符

`||` 就是或运算符，也就是或者的意思，`||` 也是一个双目操作符，使用的方式是 `a || b`，`||` 两边的表达式只要有一个是真，整个表达式就是真，两边的表达式都为假的时候，才为假。

比如：我们说一年中月份是12月或者1月或者2月是冬天，那么我们怎么使用代码体现呢？

```
int month = 0;
scanf("%d", &month);
if(month == 12 || month==1 || month == 2)
{
    printf("冬季\n");
}
```

3.4 练习：闰年的判断

输入一个年份year，判断year是否是闰年

闰年判断的规则：

1. 能被4整除并且不能被100整除是闰年
2. 能被400整除是闰年

```
#include <stdio.h>
//代码1
int main()
{
    int year = 0;
    scanf("%d", &year);
    if(year%4==0 && year%100!=0)
        printf("是闰年\n");
    else if(year%400==0)
        printf("是闰年\n");

    return 0;
}


//代码2
int main()
{
    int year = 0;
    scanf("%d", &year);
    if((year%4==0 && year%100!=0) || (year%400==0))
        printf("是闰年\n");
```

```
23
24      return 0;
25  }
```

3.5 短路

C语言逻辑运算符还有一个特点，它总是先对左侧的表达式求值，再对右边的表达式求值，这个顺序是保证的。如果左边的表达式满足逻辑运算符的条件，就不再对右边的表达式求值。这种情况称为**“短路”**。

如前面的代码:

```
1  if(month >= 3 && month <= 5)
```

表达式中&& 的左操作数是 `month >= 3` ，右操作数是 `month <= 5` ,当左操作数 `month >= 3` 的结果是0的时候，及时不判断 `month <= 5` ，整个表达式的结果也是0（不是春季）。

所以，对于&&操作符来说，左边操作数的结果是0的时候，右边操作数就不再执行。

对于 `||` 操作符是怎么样呢？我们结合前面的代码：

```
1  if(month == 12 || month==1 || month == 2)
```

如果month == 12，则不用再判断month是否等于1或者2，整个表达式的结果也是1（是冬季）。

所以， `||` 操作符的左操作数的结果不为0时，就无需执行右操作数。

像这种仅仅根据左操作数的结果就能知道整个表达式的结果，不再对右操作数进行计算的运算称为**短路求值**。

4. switch语句

除了 `if` 语句外，C语言还提供了 `switch` 语句来实现分支结构。

`switch` 语句是一种特殊形式的 `if...else` 结构，用于判断条件有多个结果的情况。它把多重的 `else if` 改成更易用、可读性更好的形式。

```
1  switch (expression) {
```

```
    case value1: statement
    case value2: statement
    default: statement
}
```

上面代码中，根据表达式 `expression` 不同的值，执行相应的 `case` 分支。如果找不到对应的值，就执行 `default` 分支。

注：

- `switch` 后的 `expression` 必须是整型表达式
- `case` 后的值，必须是整形常量表达式

4.1 if语句和switch语句的对比

练习：输入任意一个整数值，计算除3之后的余数

如果使用if语句完成，如下：

```c
#include <stdio.h>

int main()
{
    int n = 0;
    scanf("%d", &n);
    if(n%3 == 0)
        printf("整除，余数为0\n");
    else if(n%3 == 1)
        printf("余数是1\n");
    else
        printf("余数是2\n");
    return 0;
}
```

如果使用switch语句改写，就可以是这样的：

```c
#include <stdio.h>

int main()
{
    int n = 0;
    scanf("%d", &n);
    switch(n%3)
```

```
    {
    case 0:
        printf("整除，余数为0\n");
        break;
    case 1:
        printf("余数是1\n");
        break;
    case 2:
        printf("余数是2\n");
        break;
    }
    return 0;
}
```

上述的代码中，我们要注意的点有：

1. `case` 和后边的数字之间必须有空格
2. 每一个 `case` 语句中的代码执行完成后，需要加上 `break` ，才能跳出这个switch语句。

4.2 switch语句中的break

前面的代码中，如果我们去掉case语句中的break，会出现什么情况呢？

```
#include <stdio.h>

int main()
{
    int n = 0;
    scanf("%d", &n);
    switch(n%3)
    {
    case 0:
        printf("整除，余数为0\n");
    case 1:
        printf("余数是1\n");
    case 2:
        printf("余数是2\n");
    }
    return 0;
}
```

测试一组，看一下结果：

我们发现，7除以3本来余数是1，但是我们发现程序运行的结果多了一行“余数是2”的打印。

这是为什么呢?

原因是 `switch` 语句也是分支效果的，只有在 `switch` 语句中使用 `break` 才能在跳出 `switch` 语句，如果某一个 `case` 语句的后边没有 `break` 语句，代码会继续玩下执行，有可能执行其他 `case` 语句中的代码，直到遇到 `break` 语句或者 `switch` 语句结束。就比如上面的代码就执行了 `case 2` 中的语句。

所以在 `switch` 语句中 `break` 语句是非常重要的，能实现真正的分支效果。

当然， `break` 也不是每个 `case` 语句都得有，这就得根据实际情况来看了。

练习：

> 输入一个1~7的数字，打印对应的星期几
>
> 例如：
>
> 输入：1 输出：星期一
>
> 输入：2 输出：星期二
>
> 输入：3 输出：星期三
>
> 输入：4 输出：星期四
>
> 输入：5 输出：星期五
>
> 输入：6 输出：星期六
>
> 输入：7 输出：星期天

参考代码如下：

```c
#include <stdio.h>
int main()
{
    int day = 0;
    scanf("%d", &day);
    switch(day)
    {
```

```c
        case 1:
            printf("星期一\n");
            break;
        case 2:
            printf("星期二\n");
            break;
        case 3:
            printf("星期三\n");
            break;
        case 4:
            printf("星期四\n");
            break;
        case 5:
            printf("星期五\n");
            break;
        case 6:
            printf("星期六\n");
            break;
        case 7:
            printf("星期天\n");
            break;
    }
    return 0;
}
```

如果需求发生变化，变为：

1. 输入1-5，输出的是“工作日”；
2. 输入6-7，输出“休息日”

那参考代码如下：

```c
#include <stdio.h>

int main()
{
    int day = 0;
    scanf("%d", &day);

    switch(day)
    {
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
```

```
            printf("工作日\n");
            break;
        case 6:
        case 7:
            printf("休息日\n");
            break;
    }
    return 0;
}
```

上面的练习中，我们发现应该根据实际的情况，来在代码中觉得是否使用 `break` ，或者在哪里使用 `break` ，才能正确完成实际的需求。

4.3 switch语句中的default

在使用 `switch` 语句的时候，我们经常可能遇到一种情况，比如 `switch` 后的表达式中的值无法匹配代码中的 `case` 语句的时候，这时候要不就不做处理，要不就得在 `switch` 语句中加入 `default` 子句。

```
switch (expression) {
  case value1: statement
  case value2: statement
  default: statement
}
```

`switch` 后边的 `expression` 的结果不是 `value1` ，也不是 `value2` 的时候，就会执行 `default` 子句。

就比如前面做的打印星期的练习，如果 `day` 的输入不是1~7的值，如果我们要提示：输入错误，则可以这样完成代码：

```
#include <stdio.h>

int main()
{
    int day = 0;
    scanf("%d", &day);

    switch(day)
    {
        case 1:
        case 2:
```

```
        case 3:
        case 4:
        case 5:
            printf("工作日\n");
            break;
        case 6:
        case 7:
            printf("休息日\n");
            break;
        default:
            printf("输入错误\n");
            break;
    }
    return 0;
}
```

4.4 switch语句中的case和default的顺序问题

在 `switch` 语句中 `case` 子句和 `default` 子句有要求顺序吗？ `default` 只能放在最后吗？

其实，在 `switch` 语句中 `case` 语句和 `default` 语句是没有顺序要求的，只要你的顺序是满足实际需求的就可以。

不过我们通常是把 `default` 子句放在最后处理的。

5. while循环

C语言提供了3种循环语句， `while` 就是其中一种，接下来就介绍一下 `while` 语句。

`while` 语句的语法结构和 `if` 语句非常相似。

5.1 if 和 while的对比

```
if(表达式)
    语句;


while(表达式)
    语句;//如果循环体想包含更多的语句，可以加上大括号
```

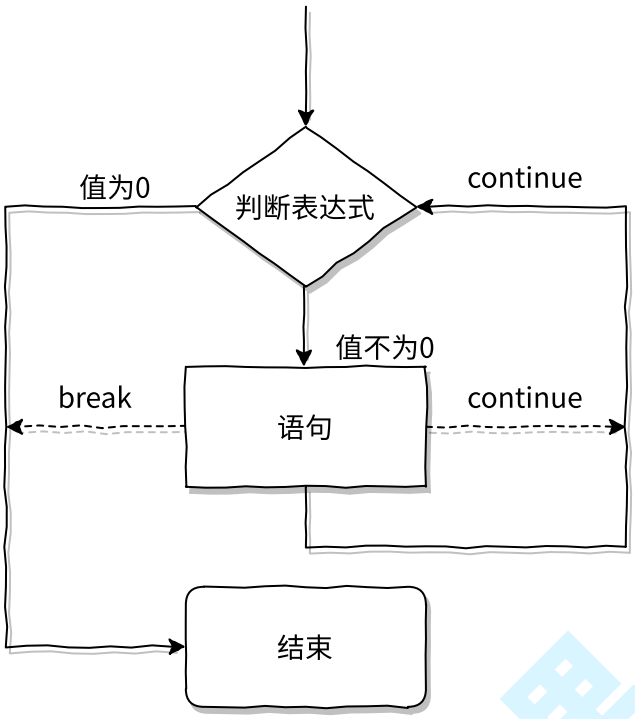
你可以对比来看一下，具体写个代码吧

```c
//代码1
#include <stdio.h>

int main()
{
    if(1)
        printf("hehe\n");  //if后边条件满足，打印一次hehe
    return 0;
}
```

```c
//代码2
#include <stdio.h>

int main()
{
    while(1)
        printf("hehe\n"); //while后边的条件满足，死循环的打印hehe
    return 0;
}
```

这就是他们的区别，while语句是可以实现循环效果的。

5.2 while语句的执行流程



while语句的执行流程图

首先上来就是执行判断表达式，表达式的值为0，循环直接结束；表达式的值不为0，则执行循环语句，语句执行完后再继续判断，是否进行下一次判断。

5.3 while循环的实践

练习：在屏幕上打印1~10的值

参考代码：

```c
#include <stdio.h>
int main()
{
    int i = 1;
    while(i<=10)
    {
        printf("%d ", i);
        i = i+1;
    }
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
1 2 3 4 5 6 7 8 9 10
D:\code\2022\test\test_6_3\Debug\test_6_3.exe (进程 25656)已退出，代码为 0。
按任意键关闭此窗口. . .
```

5.4 练习

> 输入一个正的整数，逆序打印这个整数的每一位
>
> 例如：
>
> 输入：1234，输出：4 3 2 1
>
> 输入：521，输出：1 2 5

题目解析

1. 要想得到n的最低位，可以使用n%10的运算，得到的余数就是最低位，如：1234%10得到4
2. 要想去掉n的最低位，找出倒数第二位，则使用 `n=n/10` 操作就可以去掉最低位的，如：n=1234/10得到123，123相较于1234就去掉了最低位，123%10就得到倒数第二位3。
3. 循环1和2两个步骤，在n变成0之前，就能到所有的位。

参考代码：

```c
#include <stdio.h>

int main()
{
    int n = 0;
    scanf("%d", &n);
    while(n)
    {
        printf("%d ", n%10);
        n /= 10;
    }
    return 0;
}
```

6. for循环

6.1 语法形式

`for` 循环是三种循环中使用最多的， `for` 循环的语法形式如下：

```c
for(表达式1; 表达式2; 表达式3)
```

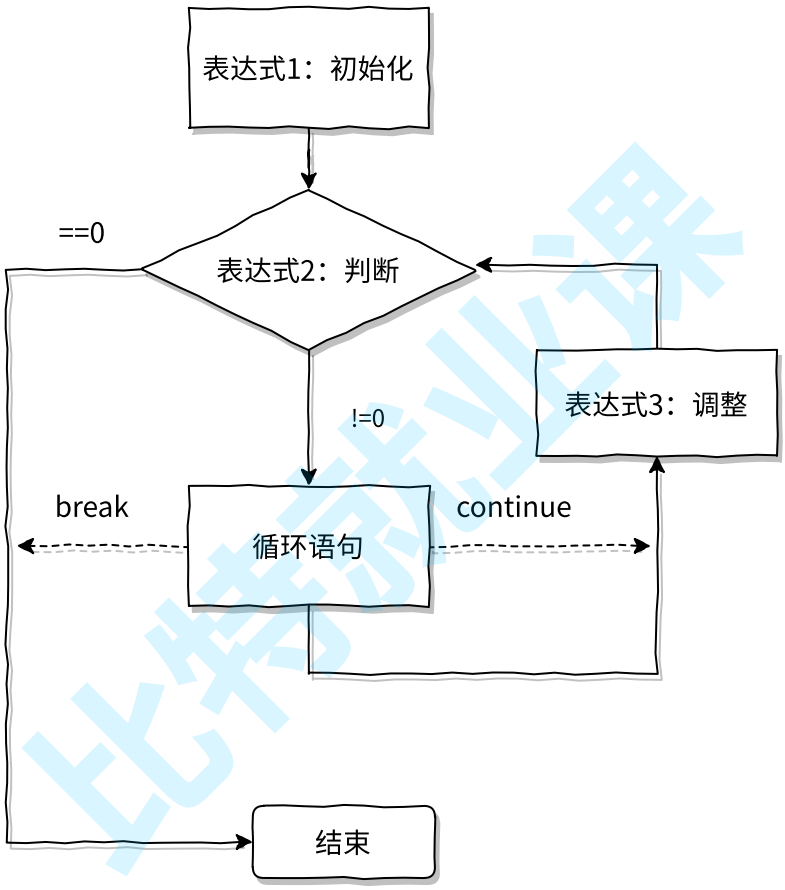
```
    语句；//如果循环体想包含更多的语句，可以加上大括号
```

表达式1 用于循环变量的初始化

表达式2 用于循环结束条件的判断

表达式3 用于循环变量的调整

6.2 for循环的执行流程



for循环的执行流程

首先执行 表达式1 初始化循环变量，接下来就是执行 表达式2 的判断部分， 表达式2 的结果如果==0，则循环结束； 表达式2 的结果如果!=0则执行循环语句，循环语句执行完后，再去执行表 达式3 ,调整循环变量，然后再去 表达式2 的地方执行判断， 表达式2 的结果是否为0，决定循环是否继续。

整个循环的过程中，表达式1初始化部分只被执行1次，剩下的就是表达式2、循环语句、表达式3在循环。

6.3 for循环的实践

练习：在屏幕上打印1~10的值

参考代码：

```c
#include <stdio.h>

int main()
{
    int i = 0;
    for(i=1; i<=10; i++)
    {
        printf("%d ", i);
    }

    return 0;
}
```

运行结果：

```
Microsoft Visual Studio 调试控制台
1 2 3 4 5 6 7 8 9 10
D:\code\2022\test\test_6_3\Debug\test_6_3.exe (进程 12132)已退出，代码为 0。
按任意键关闭此窗口. . .
```

6.4 while循环和for循环的对比

```c
#include <stdio.h>
int main()
{
    int i = 1;  // 1 初始化
    while (i <= 10)  // 2 判断
    {
        printf("%d ", i);
        i = i + 1;  // 3 调整
    }
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int i = 0;
    // 1 初始化  2 判断  3 调整
    for (i = 1; i <= 10; i++)
    {
        printf("%d ", i);
    }
    return 0;
}
```

`for` 和 `while` 在实现循环的过程中都有初始化、判断、调整这三个部分，但是 `for` 循环的三个部分非常集中，便于代码的维护，而如果代码较多的时候 `while` 循环的三个部分就比较分散，所以从形式上 `for` 循环要更优一些。

6.5 练习

练习1：

> 计算1~100之间3的倍数的数字之和

参考代码：

```c
#include <stdio.h>

int main()
{
    int i = 0;
    int sum = 0;
    for(i=1; i<=100; i++)
    {
        if(i % 3 == 0)
            sum += i;
    }
    printf("%d\n", sum);
    return 0;
}


//小小的优化
//如果能直接产生3的倍数的数字就省去了多余的循环和判断
#include <stdio.h>

int main()
{
    int i = 0;
    int sum = 0;
    for(i=3; i<=100; i+=3)
    {
            sum += i;
    }
    printf("%d\n", sum);
    return 0;
}
```

7. do-while循环

7.1 语法形式

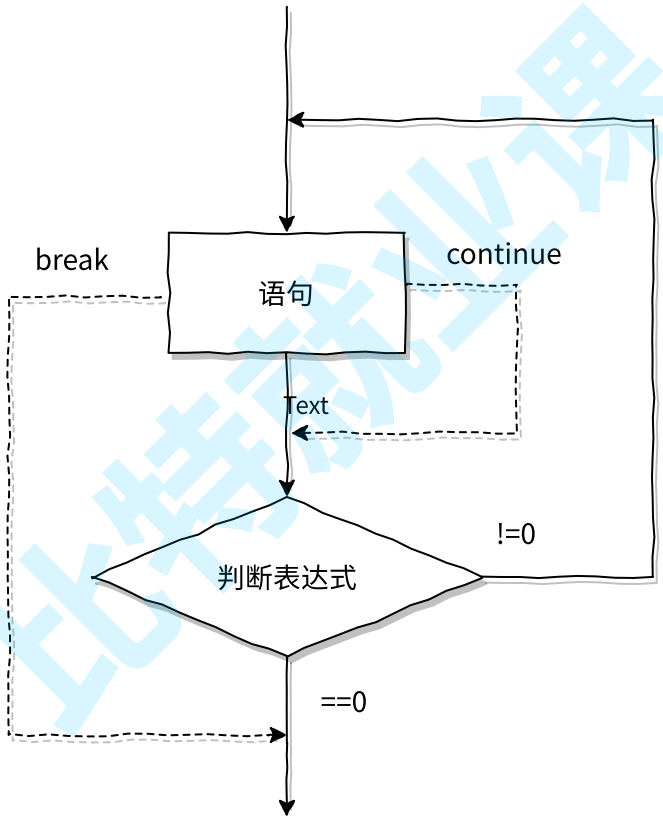
在循环语句中`do while`语句的使用最少，它的语法如下：

```
do
    语句;
while(表达式);
```

while 和 for 这两种循环都是先判断，条件如果满足就进入循环，执行循环语句，如果不满足就跳出循环；

而 do while 循环则是先直接进入循环体，执行循环语句，然后再执行 while 后的判断表达式，表达式为真，就会进行下一次，表达式为假，则不再继续循环。

7.2 do while循环的执行流程



do while 循环的执行流程

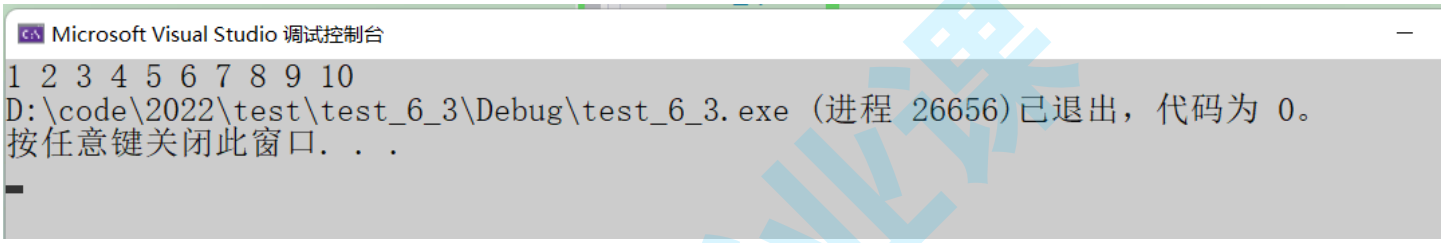
在 do while 循环中先执行图上的“语句”，执行完语句，在去执行“判断表达式”，判断表达式的结果是!=0，则继续循环，执行循环语句；判断表达式的结果==0，则循环结束。

所以在 do while 语句中循环体是至少执行一次的，这是 do while 循环比较特殊的地方。

7.3 do while循环的实例

在屏幕上打印1~10的值

```c
#include <stdio.h>

int main()
{
    int i = 1;
    do
    {
        printf("%d ", i);
        i = i + 1;
    }while(i<=10);

    return 0;
}
```



一般 `do while` 使用在循环体至少被执行一次的场景下，所以较少一些。

7.4 练习

> 输入一个正整数，计算这个整数是几位数?
>
> 例如:
>
> 输入：1234 输出：4
>
> 输入：12 输出：2

参考代码：

```c
#include <stdio.h>

int main()
{
    int n = 0;
    scanf("%d", &n);
    int cnt = 0;
    do
    {
        cnt++;
```

```
        n = n / 10;
    } while (n);
    printf("%d\n", cnt);

    return 0;
}
```

这里并非必须使用 `do while` 语句，但是这个代码就比较适合使用 `do while` 循环，因为n即使是0，也是1位数，要统计位数的。

8. break和continue语句

在循环执行的过程中，如果某些状况发生的时候，需要提前终止循环，这是非常常见的现象。C语言中提供了 `break` 和 `continue` 两个关键字，就是应该到循环中的。

- `break` 的作用是用于永久的终止循环，只要 `break` 被执行，直接就会跳出循环，继续往后执行。
- `continue` 的作用是跳过本次循环 `continue` 后边的代码，在 `for` 循环和 `while` 循环中有所差异的。

8.1 while循环中的break和continue

8.1.1 break举例

```
#include <stdio.h>
int main()
{
    int i = 1;
    while(i<=10)
    {
        if(i == 5)
            break;//当i等于5后，就执行break，循环就终止了
        printf("%d ", i);
        i = i+1;
    }
    return 0;
}
```

执行的结果：

```
1 2 3 4
D:\code\2022\test\test_6_3\Debug\test_6_3.exe (进程 35460)已退出，代码为 0。
按任意键关闭此窗口. . .
```

打印了1,2,3,4后，当i等于5的时候，循环正 `break` 的地方终止，不再打印，不再循环。

所以 `break` 的作用就是永久的终止循环，只要 `break` 被执行， `break` 外的第一层循环就终止了。

那以后我们在循环中，想在某种条件下终止循环，则可以使用 `break` 来完成我们想要的效果。

8.1.2 continue举例

`continue` 是继续的意思，在循环中的作用就是跳过本次循环中 `continue` 后边的代码，继续进行下一次循环的判断。

上面的代码，如果把 `break` 换成 `continue` 会是什么结果呢?

```c
#include <stdio.h>
int main()
{
    int i = 1;
    while(i<=10)
    {
        if(i == 5)
            continue;
            //当i等于5后，就执行continue，直接跳过continue的代码，去循环的判断的地方
            //因为这里跳过了i = i+1，所以i一直为5，程序陷入和死循环
        printf("%d ", i);
        i = i+1;
    }
    return 0;
}
```

到这里我们就能分析出来， `continue` 可以帮助我们跳过某一次循环 `continue` 后边的代码，直接到循环的判断部分，进行下一次循环的判断，如果循环的调整是在 `continue` 后边的话，可能会造成死循环。

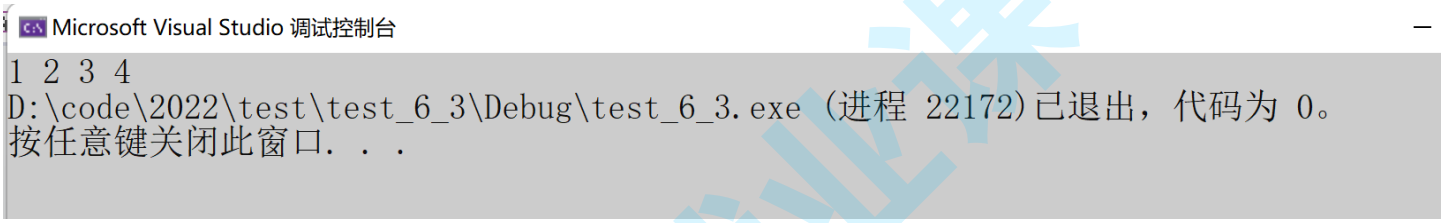
8.2 for循环中的break和continue

8.2.1 break举例

其实和 `while` 循环中的 `break` 一样， `for` 循环中的 `break` 也是用于终止循环的，不管循环还需要循环多少次，只要执行到了 `break` ，循环就彻底终止，我们上代码。

```c
#include <stdio.h>
int main()
{
    int i = 1;
    for(i=1; i<=10; i++)
    {
        if(i == 5)
            break;
        printf("%d ", i);
    }
    return 0;
}
```

运行结果：



`break` 的作用是永久的终止循环，未来我们在某个条件发生的时候，不想再继续循环的时候，就可以使用 `break` 来完成。

8.2.2 continue举例

上面的代码，如果把 `break` 换成 `continue` 会是什么结果呢？

```c
#include <stdio.h>
int main()
{
    int i = 1;
    for(i=1; i<=10; i++)
    {
        if(i == 5)
            continue;//这里continue跳过了后边的打印，来到了i++的调整部分
        printf("%d ", i);
    }
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
1 2 3 4 6 7 8 9 10
D:\code\2022\test\test_6_3\Debug\test_6_3.exe (进程 33276)已退出，代码为 0。
按任意键关闭此窗口. . .
```

所以在 `for` 循环中 `continue` 的作用是跳过本次循环中 `continue` 后的代码，直接去到循环的调整部分。未来当某个条件发生的时候，本次循环无需再执行后续某些操作的时候，就可以使用 `continue` 来实现。

在这里我们也可以对比一下 `while` 循环和 `for` 循环中 `continue` 的区别：

```c
#include <stdio.h>
int main()
{
    int i = 1;
    while (i <= 10)
    {
        if (i == 5)
            continue;
        printf("%d ", i);
        i = i + 1;
    }
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int i = 1;
    for (i = 1; i <= 10; i++)
    {
        if (i == 5)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

8.3 do while循环中的break和continue

`do.while` 语句中的 `break` 和 `continue` 的作用和 `while` 循环中几乎一模一样，大家下来可以自行测试并体会。

```c
#include <stdio.h>

int main()
{
    int i = 1;
    do
    {
        if(i == 5)
            break;
        printf("%d ", i);
        i = i + 1;
    }while(i<=10);

```

```c
#include <stdio.h>

int main()
{
    int i = 1;
    do
    {
        if(i == 5)
            continue;
        printf("%d ", i);
        i = i + 1;
    }while(i<=10);

```

```
    return 0;
}
```

```
    return 0;
}
```

9. 循环的嵌套

前面学习了三种循环 `while` ， `do while` ， `for` ，这三种循环往往会嵌套在一起才能更好的解决问题，就是我们所说的：循环嵌套，这里我们就看一个例子。

9.1 练习

找出100~200之间的素数，并打印在屏幕上。

注：素数又称质数，只能被1和本身整除的数字。

9.2 题目解析：

1. 要从100~200之间找出素数，首先得有100~200之间的数，这里可以使用循环解决。
2. 假设要判断i是否为素数，需要拿2~i-1之间的数字去试除i，需要产生2~i-1之间的数字，也可以使用循环解决。
3. 如果2~i-1之间有数字能整除i，则i不是素数，如果都不能整除，则i是素数。

9.3 参考代码：

```c
#include <stdio.h>
int main()
{
    int i = 0;
    //循环产生100~200的数字
    for(i=100; i<=200; i++)
    {
        //判断i是否为素数
        //循环产生2~i-1之间的数字
        int j = 0;
        int flag = 1;//假设i是素数
        for(j=2; j<i; j++)
        {
            if(i % j == 0)
            {
                flag = 0;
                break;
```

```
            }
        }
        if(flag == 1)
            printf("%d ", i);
    }
    return 0;
}
```

10. goto语句

C语言提供了一种非常特别的语法，就是 `goto` 语句和跳转标号， `goto` 语句可以实现在**同一个函数内**跳转到设置好的标号处。

例如：

```
#include <stdio.h>
int main()
{
    printf("hehe\n");
    goto next:
    printf("haha\n");

next:
    printf("跳过了haha的打印\n");
    return 0;
}
```

`goto` 语句如果使用的不当，就会导致在函数内部随意乱跳转，打乱程序的执行流程，所以我们的建议是能不用尽量不去使用；但是 `goto` 语句也不是一无是处，在多层循环的代码中，如果想快速跳出使用 `goto` 就非常的方便了。

```
for(...)
{
    for(...)
    {
        for(...)
        {
            if(disaster)
                goto error;
        }
    }
```

```
11 }
12
13 error:
14     //...
```

本来 `for` 循环想提前退出得使用 `break` ，一个 `break` 只能跳出一层 `for` 循环，如果3层循环嵌套就得使用3个 `break` 才能跳出循环，所以在这种情况下我们使用 `goto` 语句就会更加的快捷。

完