分支语句和循环语句

本章重点学习分支语句和循环语句的使用。

> 分支语句

- if
- switch

> 循环语句

- while
- for
- do while

> goto语句

正文开始@比特科技

什么是语句？

C语言中由一个分号；隔开的就是一条语句。 比如：

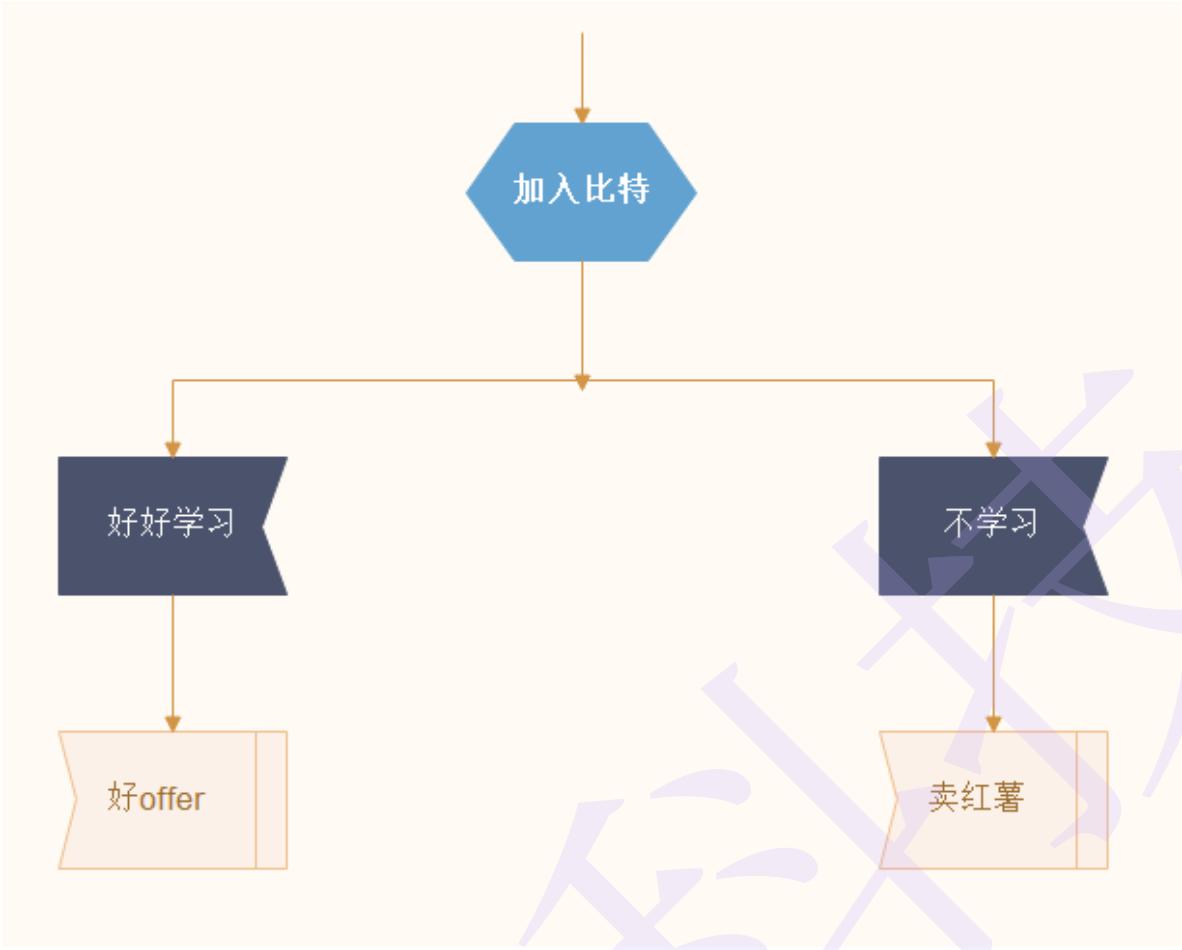
```
printf("hehe");
1+2;
```

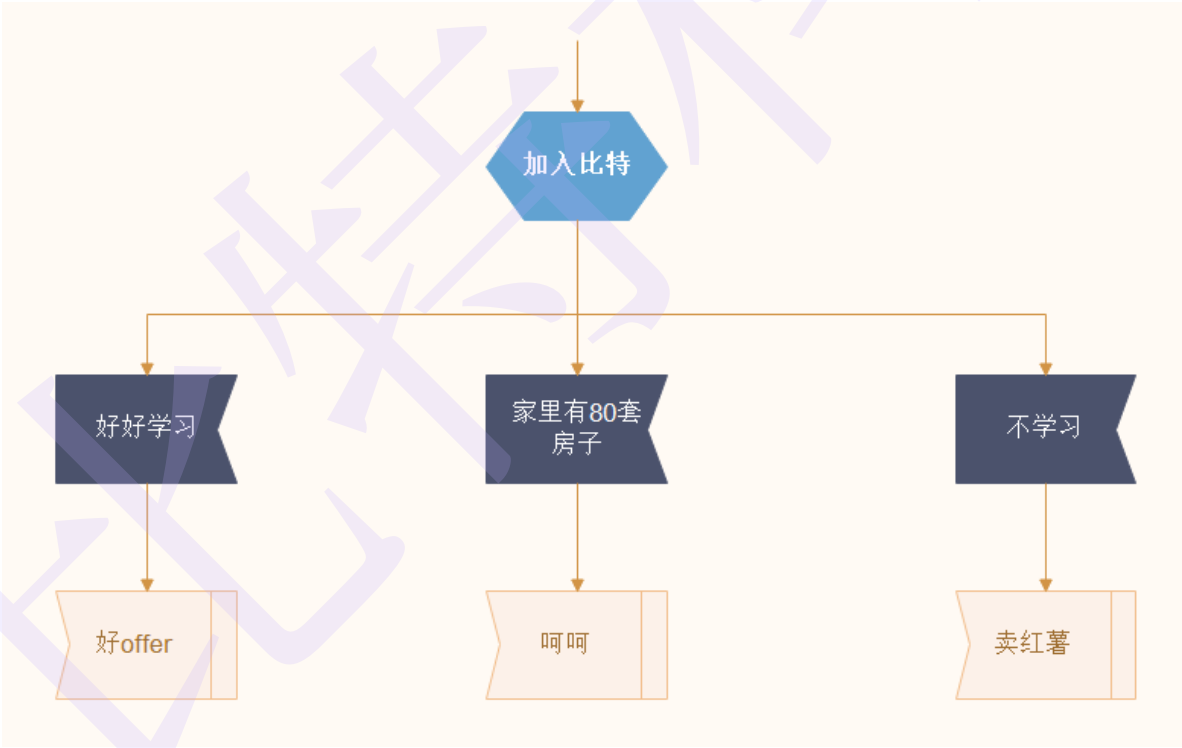
分支语句（选择结构）

如果你好好学习，校招时拿一个好offer，走上人生巅峰。

如果你不学习，毕业等于失业，回家卖红薯。

这就是选择！





if语句

那if语句的语法结构是怎么样的呢？

```
语法结构：
if(表达式)
    语句;

if(表达式)
```

```
    语句1;
else
    语句2;

//多分支
if(表达式1)
    语句1;
else if(表达式2)
    语句2;
else
    语句3;
```

课堂演示代码：

```
#include <stdio.h>
//代码1
int main()
{
    int age = 0;
    scanf("%d", &age);
    if(age<18)
    {
        printf("未成年\n");
    }
}
//代码2
#include <stdio.h>
int main()
{
    int age = 0;
    scanf("%d", &age);
    if(age<18)
    {
        printf("未成年\n");
    }
    else
    {
        printf("成年\n");
    }
}
//代码3
#include <stdio.h>
int main()
{
    int age = 0;
    scanf("%d", &age);
    if(age<18)
    {
        printf("少年\n");
    }
    else if(age>=18 && age<30)
    {
        printf("青年\n");
    }
    else if(age>=30 && age<50)
    {
        printf("中年\n");
```

```
    }
    else if(age>=50 && age<80)
    {
        printf("老年\n");
    }
    else
    {
        printf("老不死\n");
    }

}
```

解释一下： 如果表达式的结果为真，则语句执行。

在C语言中如何表示真假？

> **0表示假，非0表示真。**

如果条件成立，要执行多条语句，怎应该使用代码块。

```
#include <stdio.h>
int main()
{
    if(表达式)
    {
        语句列表1;
    }
    else
    {
        语句列表2;
    }
    return 0;
}
```

这里的一对 `{ }` 就是一个代码块。

悬空else

当你写了这个代码：

```
#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;
}
```

改正：

```
//适当的使用{}可以使代码的逻辑更加清楚。
//代码风格很重要
#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;
}
```

if书写形式的对比

```
//代码1
if (condition) {
    return x;
}
return y;

//代码2
if(condition)
{
    return x;
}
else
{
    return y;
}

//代码3
int num = 1;
if(num == 5)
{
    printf("hehe\n");
}

//代码4
int num = 1;
if(5 == num)
{
    printf("hehe\n");
}
```

代码2和代码4更好，逻辑更加清晰，不容易出错。

练习

> 1. 判断一个数是否为奇数
> 2. 输出1-100之间的奇数

switch语句

switch语句也是一种分支语句。 常常用于多分支的情况。

比如：

> 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，输出星期七

那我没写成`if...else if ...else if`的形式太复杂，那我们就得有不一样的语法形式。 这就是switch 语句。

```
switch(整型表达式)
{
    语句项;
}
```

而**语句项**是什么呢？

```
//是一些case语句;
//如下:
case 整形常量表达式:
    语句;
```

在switch语句中的 break

在switch语句中，我们没法直接实现分支，搭配break使用才能实现真正的分支。

比如：

```
#include <stdio.h>
int main()
{
    int day = 0;
    switch(day)
    {
        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;
}
```

有时候我们的需求变了：

> 1. 输入1-5输出的是“weekday”;
> 2. 输入6-7输出“weekend”

所以我们的代码就应该这样实现了：

```
#include <stdio.h>
//switch代码演示
int main()
{
    int day = 0;
    switch(day)
    {
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
            printf("weekday\n");
            break;
        case 6:
        case 7:
            printf("weekend\n");
            break;
    }
    return 0;
}
```

break语句的实际效果是把语句列表划分为不同的部分。

编程好习惯

> 在最后一个 case 语句的后面加上一条 break语句。（之所以这么写是可以避免出现在以前的最后一个 case 语句后面忘了添加 break语句）。

default子句

如果表达的值与所有的case标签的值都不匹配怎么办？

其实也没什么，结构就是所有的语句都被跳过而已。

程序并不会终止，也不会报错，因为这种情况在C中并不认为适合错误。

但是，如果你并不想忽略不匹配所有标签的表达式的值时该怎么办呢？

你可以在语句列表中增加一条default子句，把下面的标签

`default:`

写在任何一个case标签可以出现的位置。

当 switch表达式的值并不匹配所有case标签的值时，这个default子句后面的语句就会执行。

所以，每个switch语句中只能出现一条default子句。

但是它可以出现在语句列表的任何位置，而且语句流会像贯穿一个case标签一样贯穿default子句。

编程好习惯

> 在每个 switch 语句中都放一条default子句是个好习惯，甚至可以在后边再加一个 break 。

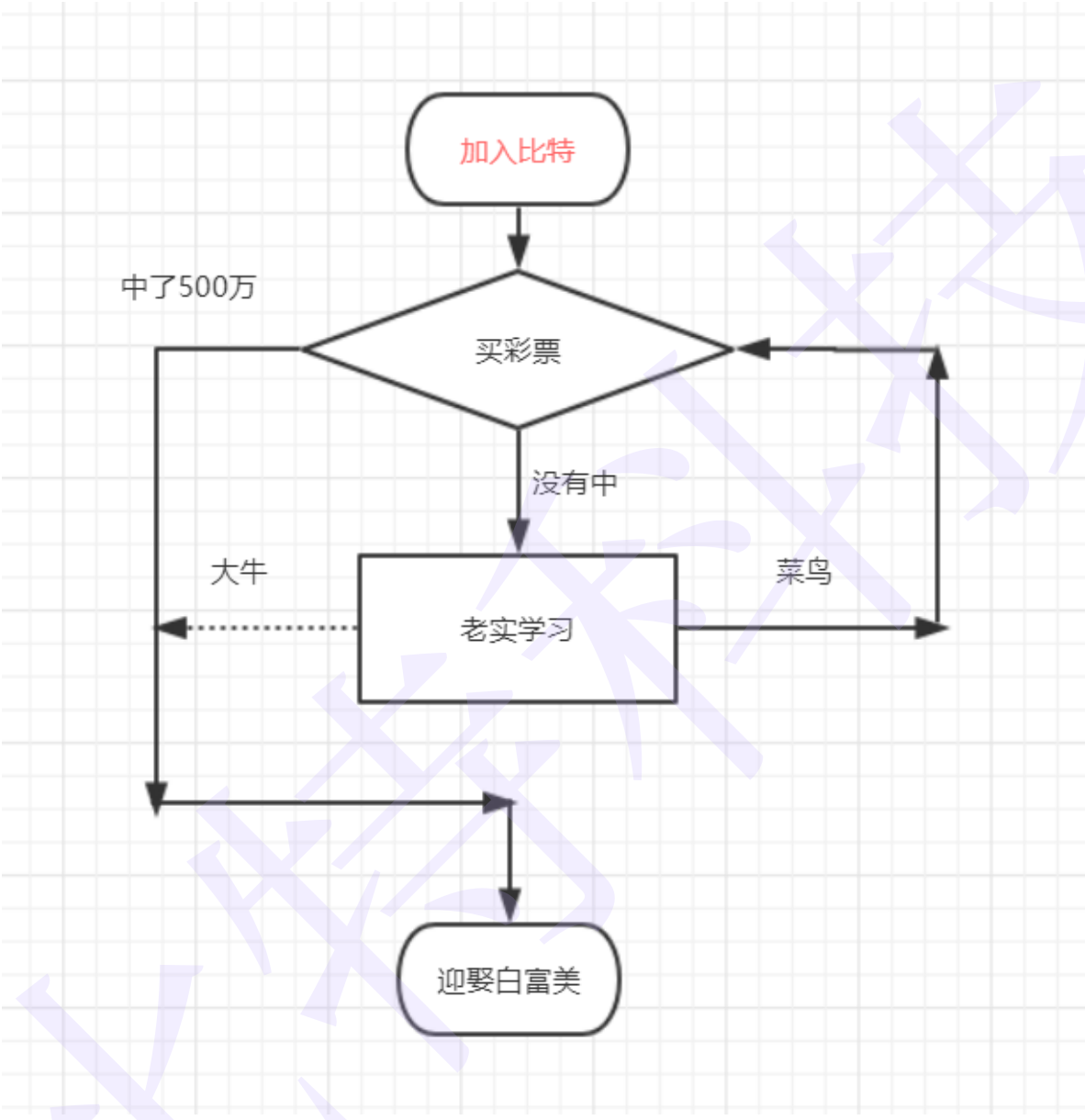
练习

```
#include <stdio.h>

int main()
{
    int n = 1;
    int m = 2;
    switch (n)
    {
    case 1:
            m++;
    case 2:
            n++;
    case 3:
             switch (n)
            {//switch允许嵌套使用
             case 1:
                    n++;
             case 2:
                    m++;
                    n++;
                    break;
            }
    case 4:
            m++;
             break;
    default:
             break;
    }
    printf("m = %d, n = %d\n", m, n);
    return 0;
}
```

循环语句

- while
- for
- do while



while循环

我们已经掌握了，if语句：

```
if(条件)
    语句;
```

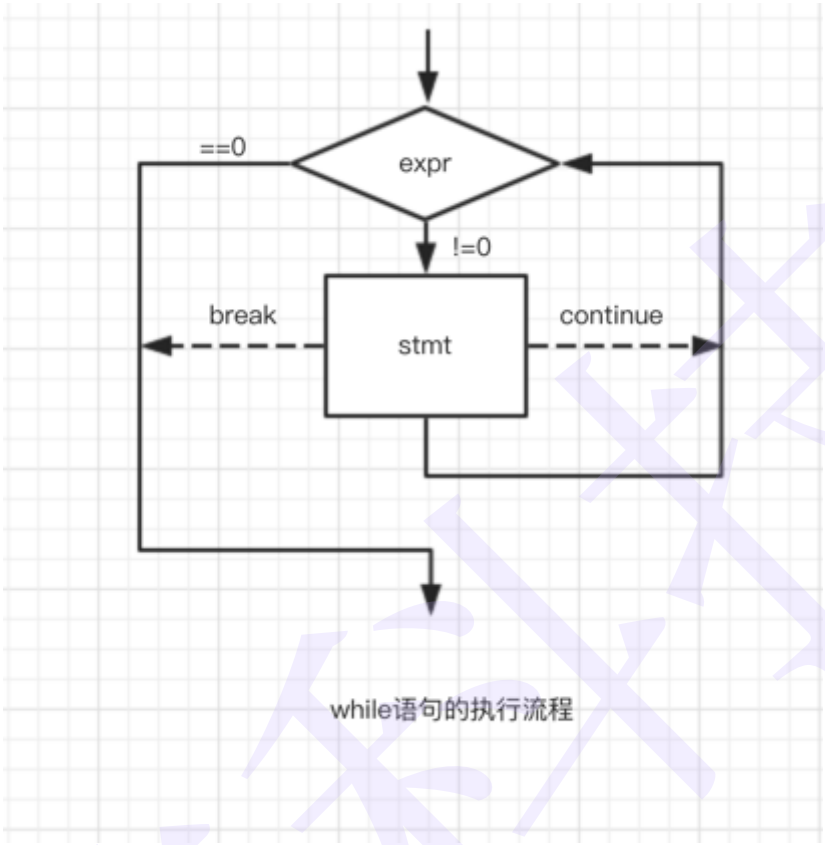
当条件满足的情况下，if语句后的语句执行，否则不执行。但是这个语句只会执行一次。

但是我们发现生活中很多的实际的例子是：同一件事情我们需要完成很多次。

那我们怎么做呢？ C语言中给我们引入了：while语句，可以实现循环。

```
//while 语法结构
while(表达式)
    循环语句;
```

while语句执行的流程：



while语句的执行流程

比如我们实现：

> 在屏幕上打印1-10的数字。

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

这个代码已经帮我了解了while语句的基本语法，那我们再了解一下：

while语句中的break和continue

break介绍

```
//break 代码实例
#include <stdio.h>
int main()
{
    int i = 1;
    while(i<=10)
    {
```

```
        if(i == 5)
            break;
        printf("%d ", i);
        i = i+1;
    }
    return 0;
}
```

这里代码输出的结果是什么？

> 1 2 3 4
>
> 1 2 3 4 5
>
> 1 2 3 4 5 6 7 8 9 10
>
> 1 2 3 4 6 7 8 9 10

总结： break在while循环中的作用：

> 其实在循环中只要遇到break，就停止后期的所有的循环，直接终止循环。 所以：while中的break是用于**永久**终止循环的。

continue介绍

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

这里代码输出的结果是什么？

> 1 2 3 4
>
> 1 2 3 4 5
>
> 1 2 3 4 5 6 7 8 9 10
>
> 1 2 3 4 6 7 8 9 10

```
//continue 代码实例2
#include <stdio.h>
int main()
{
    int i = 1;
    while(i<=10)
    {
        i = i+1;
        if(i == 5)
            continue;
```

```
        printf("%d ", i);
    }
    return 0;
}
```

这里代码输出的结果是什么？

> 1 2 3 4
>
> 1 2 3 4 5
>
> 1 2 3 4 5 6 7 8 9 10
>
> 1 2 3 4 6 7 8 9 10
>
> 2 3 4 6 7 8 9 10

总结: continue在while循环中的作用就是：

> continue是用于终止本次循环的，也就是本次循环中continue后边的代码不会再执行，而是直接跳转到while语句的判断部分。进行下一次循环的入口判断。

再看几个代码：

```
//代码什么意思？
//代码1
#include <stdio.h>
int main()
{
    int ch = 0;
    while ((ch = getchar()) != EOF)
       putchar(ch);
    return 0;
}

//代码2
#include <stdio.h>
int main()
{
    while ((ch = getchar()) != EOF)
    {
        if (ch < '0' || ch > '9')
            continue;
        putchar(ch);
    }
    return 0;
}
```

for循环

我们已经知道了while循环，但是我们为什么还要一个for循环呢？ 首先来看看for循环的语法：

语法

```
for(表达式1；表达式2；表达式3)
    循环语句;
```

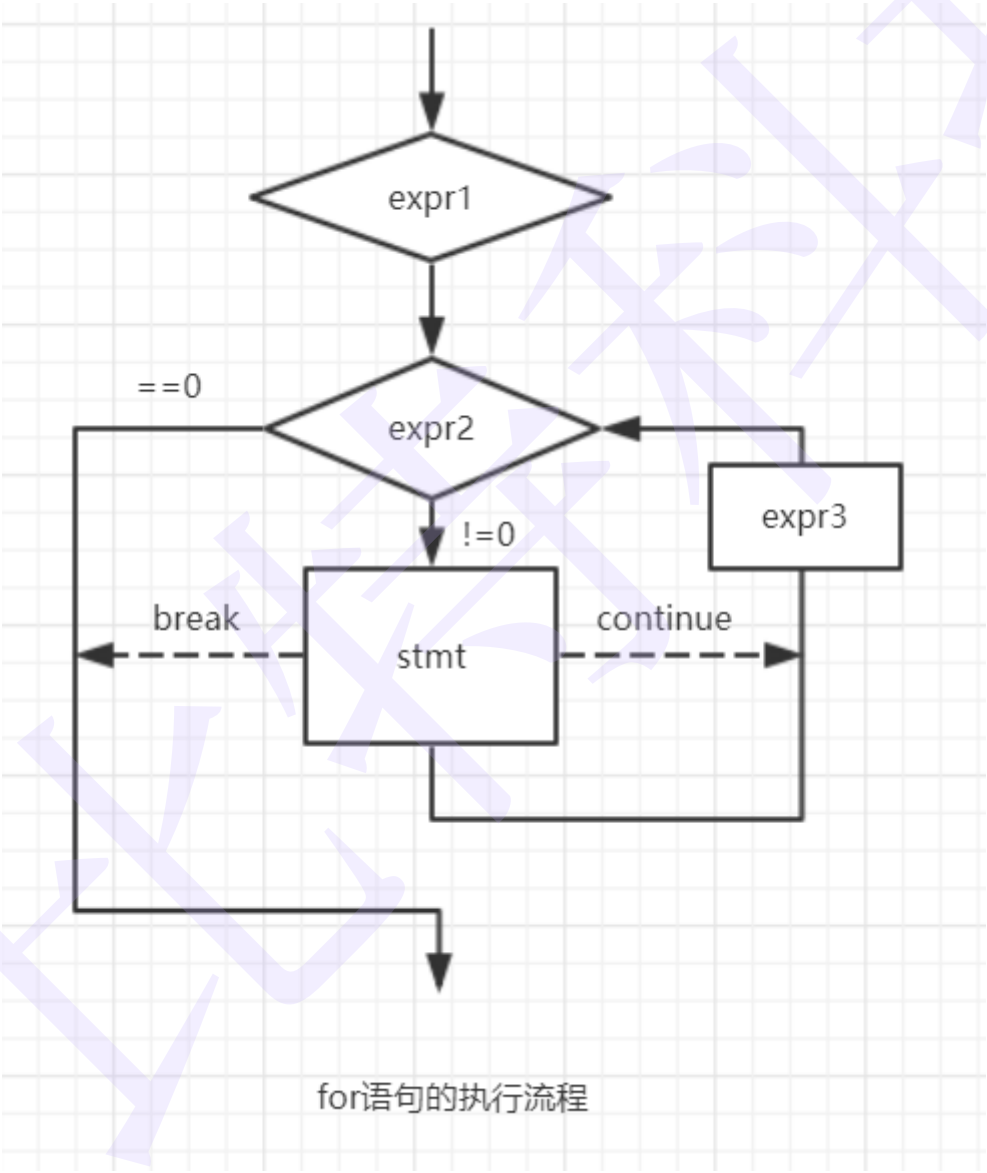
表达式1 表达式1为**初始化部分**，用于初始化循环变量的。 **表达式2** 表达式2为**条件判断部分**，用于判断循环时候终止。 **表达式3** 表达式3为**调整部分**，用于循环条件的调整。

实际的问题：

> 使用for循环 在屏幕上打印1-10的数字。

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

for循环的执行流程图：



for语句的执行流程

现在我们对比一下for循环和while循环。

```
int i = 0;
//实现相同的功能，使用while
i=1;//初始化部分
while(i<=10)//判断部分
{
```

```
    printf("hehe\n");
    i = i+1;//调整部分
}

//实现相同的功能，使用while
for(i=1; i<=10; i++)
{
    printf("hehe\n");
}
```

可以发现在while循环中依然存在循环的三个必须条件，但是由于风格的问题使得三个部分很可能偏离较远，这样查找修改就不够集中和方便。所以，for循环的风格更胜一筹。 for循环使用的频率也最高。

break和continue在for循环中

我们发现在for循环中也可以出现break和continue，他们的意义和在while循环中是一样的。 但是还是有些差异：

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

//代码2
#include <stdio.h>
int main()
{
    int i = 0;
    for(i=1; i<=10; i++)
    {
        if(i == 5)
            continue;
        printf("%d ",i);
    }
    return 0;
}
```

for语句的循环控制变量

一些建议：

> 1. 不可在for 循环体内修改循环变量，防止 for 循环失去控制。
> 2. 建议for语句的循环控制变量的取值采用“前闭后开区间”写法。

```
int i = 0;
//前闭后开的写法
for(i=0; i<10; i++)
{}

//两边都是闭区间
for(i=0; i<=9; i++)
{}
```

一些for循环的变种

```
#include <stdio.h>
int main()
{
    //变种1
    for(;;)
    {
        printf("hehe\n");
    }
    //变种2
    int x, y;
    for (x = 0, y = 0; x<2 && y<5; ++x, y++)
    {
        printf("hehe\n");
    }
    return 0;
}
```

一道笔试题：

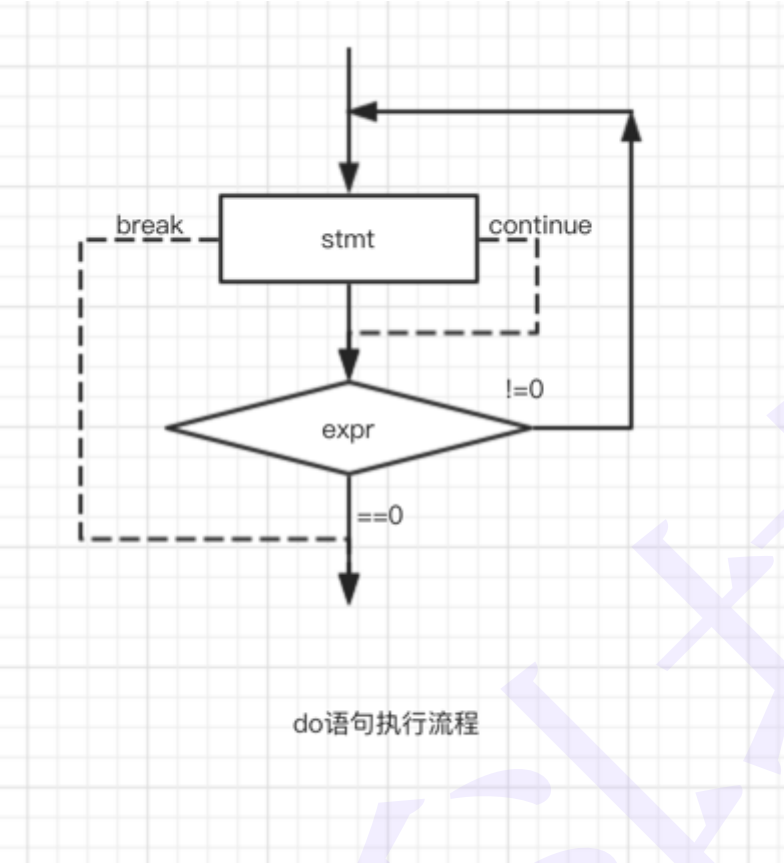
```
//请问循环要循环多少次?
#include <stdio.h>
int main()
{
    int i = 0;
    int k = 0;
    for(i =0,k=0; k=0; i++,k++)
        k++;
    return 0;
}
```

do...while()循环

do语句的语法：

```
do
    循环语句;
while(表达式);
```

执行流程



do语句执行流程

do语句的特点

循环至少执行一次，使用的场景有限，所以不是经常使用。

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

do while循环中的break和continue

```
#include <stdio.h>
int main()
{
    int i = 10;
    do
    {
        if(5 == i)
            break;
        printf("%d\n", i);
    }while(i<10);
    return 0;
}

#include <stdio.h>
int main()
```

```
{
    int i = 10;
    do
    {
        if(5 == i)
            continue;
        printf("%d\n", i);
    }while(i<10);
    return 0;
}
```

练习

1. 计算 n的阶乘。
2. 计算 1!+2!+3!+.....+10!
3. 在一个有序数组中查找具体的某个数字n。 编写int binsearch(int x, int v[], int n); 功能：在v[0]<=v[1]<=v[2]<= ....<=v[n-1]的数组中查找x.
4. 编写代码，演示多个字符从两端移动，向中间汇聚。
5. 编写代码实现，模拟用户登录情景，并且只能登录三次。（只允许输入三次密码，如果密码正确则提示登录成，如果三次均输入错误，则退出程序。

练习参考代码：

```
//代码1
//编写代码，演示多个字符从两端移动，向中间汇聚
#include <stdio.h>
int main()
{
    char arr1[] = "welcome to bit...";
    char arr2[] = "##################";
    int left = 0;
    int right = strlen(arr1)-1;
    printf("%s\n", arr2);
    //while循环实现
    while(left<=right)
    {
        Sleep(1000);
        arr2[left] = arr1[left];
        arr2[right] = arr1[right];
        left++;
        right--;
        printf("%s\n", arr2);
    }
    //for循环实现
    for (left=0, right=strlen(src)-1;
         left <= right;
         left++, right--)
    {
        Sleep(1000);
        arr2[left] = arr1[left];
        arr2[right] = arr1[right];
        printf( "%s\n", target);
    }
    retutn 0;
}
```

```
//代码2

int main()
{
    char psw[10] = "" ;
    int i = 0;
    int j = 0;
    for (i = 0; i < 3 ; ++i)
    {
        printf( "please input:");
        scanf("%s", psw);
        if (strcmp(psw, "password" ) == 0)
            break;
    }
    if (i == 3)
        printf("exit\n");
    else
        printf( "log in\n");
}
```

折半查找算法

比如我买了一双鞋，你好奇问我多少钱，我说不超过300元。你还是好奇，你想知道到底多少，我就让你猜，你会怎么猜？

答案：你每次猜中间数。

代码实现：

```
实现在主函数内：
int main()
{
    int arr[] = {1,2,3,4,5,6,7,8,9,10};
    int left = 0;
    int right = sizeof(arr)/sizeof(arr[0])-1;
    int key = 7;
    int mid = 0;
    while(left<=right)
    {
        mid = (left+right)/2;
        if(arr[mid]>key)
        {
            right = mid-1;
        }
        else if(arr[mid] < key)
        {
            left = mid+1;
        }
        else
            break;
    }
    if(left <= right)
        printf("找到了,下标是%d\n", mid);
    else
        printf("找不到\n");
```

```
}
```

如果实现一个二分查找函数：

```
int bin_search(int arr[], int left, int right, int key)
{
    int mid = 0;
    while(left<=right)
    {
        mid = (left+right)>>1;
        if(arr[mid]>key)
        {
            right = mid-1;
        }
        else if(arr[mid] < key)
        {
            left = mid+1;
        }
        else
            return mid;//找到了，返回下标
    }
    return -1;//找不到
}
```

猜数字游戏实现

参考代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void menu()
{
    printf("*********************************\n");
    printf("***********  1.play     **********\n");
    printf("***********  0.exit     **********\n");
    printf("*********************************\n");
}
//TDD-测试驱动开发。
//RAND_MAX--rand函数能返回随机数的最大值。
void game()
{
    int random_num = rand()%100+1;
    int input = 0;
    while(1)
    {
        printf("请输入猜的数字>:");
        scanf("%d", &input);
        if(input > random_num)
        {
            printf("猜大了\n");
        }
        else if(input < random_num)
        {
            printf("猜小了\n");
        }
```

```
        else
        {
            printf("恭喜你，猜对了\n");
            break;
        }
    }
}
int main()
{
    int input = 0;
    srand((unsigned)time(NULL));
    do
    {
        menu();
        printf("请选择>:");
        scanf("%d", &input);
        switch(input)
        {
        case 1:
            game();
            break;
        case 0:
            break;
        default:
            printf("选择错误,请重新输入!\n");
            break;
        }
    }while(input);
    return 0;
}
```

goto语句

C语言中提供了可以随意滥用的 goto语句和标记跳转的标号。

从理论上 goto语句是没有必要的，实践中没有goto语句也可以很容易的写出代码。

但是某些场合下goto语句还是用得着的，最常见的用法就是终止程序在某些深度嵌套的结构的处理过程，例如一次跳出两层或多层循环。

这种情况使用break是达不到目的的。它只能从最内层循环退出到上一层的循环。

下面是使用goto语句的一个例子：

一个关机程序

```
#include <stdio.h>
int main()
{
    char input[10] = {0};
    system("shutdown -s -t 60");
again:
    printf("电脑将在1分钟内关机，如果输入：我是猪，就取消关机!\n请输入:>");
    scanf("%s", input);
    if(0 == strcmp(input, "我是猪"))
    {
        system("shutdown -a");
    }
```

```
    else
    {
        goto again;
    }
    return 0;
}
```

而如果不适用goto语句，则可以使用循环：

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char input[10] = {0};
    system("shutdown -s -t 60");
    while(1)
    {
        printf("电脑将在1分钟内关机，如果输入：我是猪，就取消关机!\n请输入:>");
        scanf("%s", input);
        if(0 == strcmp(input, "我是猪"))
        {
            system("shutdown -a");
            break;
        }
    }
    return 0;
}
```

关于shutdown命令的扩展-（请点这里）

goto语言真正适合的场景如下：

```
for(...)
    for(...)
    {
        for(...)
        {
            if(disaster)
                goto error;
        }
    }
    …
error:
    if(disaster)
        // 处理错误情况
```

本章完。