東南大學 SCHOOL OF INTEGRATED CIRCUITS, SEU

集成电路学院

计算机科学基础I

——判断和循环

东南大学 集成电路学院 朱彬武

E-mail: bwzhu@seu.edu.cn

回顾

- if语句
- if-else语句
- 嵌套的判断语句
- 级联的if-else语句
- 注意点：花括号、缩进

```
if (math_score >= 60){
    if (science_score >= 90){
        cout << "优秀" << endl;
    }
    else{
        cout << "良好" << endl;
    }
}
```

=

```
if (math_score >= 60)
    if (science_score >= 90)
        cout << "优秀" << endl;
    else
        cout << "良好" << endl;
```

练习（1）


```
int score = 95;
if (score >= 90)
    cout << "A";
else
    cout << "B";
    cout << "结束";
```

输出结果是什么？

A结束

练习（2）


```
int x = 1;
if (x > 0)
    cout << "P";
    cout << "Q";
else
    cout << "R";
```

输出结果是什么？

编译错误

练习（3）


```
int t = -10;
if (t >= 0)
    if (t >= 20)
        cout << "hot";
    else if(t >=10)
        cout << "warm";
else
    cout << "cool";
```

输出结果是什么？

没有输出

练习（4）


```
int a = 0;
if (a = 1)
    cout << "OK";
else
    cout << "NO";
```

输出结果是什么？

OK

switch-case语句

switch-case引入


```
if(season==1){
    cout << "Spring" << endl;
}
else if(season==2){
    cout << "Summer" << endl;
}
else if(season==3){
    cout << "Autumn" << endl;
}
else if(season==4){
    cout << "Winter" << endl;
}
else{
    cout << "请输入数字1-4中的一个数字" << endl;
}
```

多重级联的if-else在分支较多的情况下，有时候需要经过多次判断才能找到对应的分支，效率较低。

修改后的代码——switch-case语句

不需要做多次判断，根据season的结果找到一个入口

```
if(season==1){
    cout << "Spring" << endl;
}
else if(season==2){
    cout << "Summer" << endl;
}
else if(season==3){
    cout << "Autumn" << endl;
}
else if(season==4){
    cout << "Winter" << endl;
}
else{
    cout << "请输入数字1-4中的一个数字" << endl;
}
```

```
switch(season){
    case 1:
        cout << "Spring" << endl;
        break;
    case 2:
        cout << "Summer" << endl;
        break;
    case 3:
        cout << "Autumn" << endl;
        break;
    case 4:
        cout << "Winter" << endl;
        break;
    default:
        cout << "请输入数字1-4中的一个数字" << endl;
}
```

新增关键字：switch、case、break、default

switch-case语句解析（1）

```
switch（控制表达式）{
        case 常量1：
                …
                break;
        case 常量2：
                …
                break;
        …
        default:
                …

}
```

- switch语句可以看作是一种基于计算结果的跳转，计算控制表达式的值后，程序会跳转到相匹配的case（分支标号）处。
- 控制表达式只能是整数（int）或字符型（char）的结果，不能是浮点数（float、double）或字符串（string）

switch-case语句解析（2）


```
switch（控制表达式）{
        case 常量1：
                …
                break;
        case 常量2：
                …
                break;
        …
        default:
                …
}
```

- case后面必须跟常量，常量可以是常数，也可以是常数计算的表达式

```
switch (ch) {
    case 'a':    // 字符常量
    case 65:     // 整数字面量
}
```

```
int x = 1
switch(n){
    case x:      //error
    case 3.14:   //error
}
```

- break 语句：表示“跳出 switch 语句”。
- default可选，用于处理所有未匹配的情况。

可以没有break语句

```
switch（控制表达式）{
        case 常量1：
                …
                break;
        case 常量2：
                …
                break;
        …
        default:
                …

}
```

可以没有break

- case只是判断的入口，在执行完分支中的最后一条语句后，如果后面没有break，就会顺序执行到下面的case里面去（称为“贯穿”），直到遇到下一个break，或者switch结束为止。

不带break的switch-case语句


```
switch(season){
    case 1:
    case 2:
        cout << "Summer" << endl;
    case 3:
        cout << "Autumn" << endl;
        break;
    case 4:
        cout << "Winter" << endl;
        break;
    default:
        cout << "请输入数字1-4中的一个数字" << endl;
}
```

下面每种情况下，程序的输出是什么？

- season为1时
- season为2时
- season为3时
- season为4时
- season为5时

switch-case程序练习

编写程序将一个百分制成绩转换为五分制成绩，转换规则

- 大于等于90分为A；
- 小于90且大于等于80为B；
- 小于80且大于等于70为C；
- 小于70且大于等于60为D；
- 小于60为E

```
// 用score / 10 来得到十位数，方便分段
switch (score / 10) {
    case 10:  // 100分，利用没有break的贯穿特性输出A
    case 9:
        cout << "A" << endl;
        break;
    case 8:
        cout << "B" << endl;
        break;
    case 7:
        cout << "C" << endl;
        break;
    case 6:
        cout << "D" << endl;
        break;
    default:
        cout << "E" << endl;
}
```

while循环

数数几位数

- 输入一个4位以下（含4位）的非负数，然后输出这个整数的位数。如：
  - 输入：353，输出：3
  - 输入：41，输出：2

人vs计算机

- 人的方式：眼睛一看就知道
  - 352 → 3位！
- 计算机的方式：判断数的范围来决定它的位数
  - $352 \in [100, 999]$ →3位

基于条件判断的程序实现


```
int x;
int n = 1;

cin >> x;
if(x > 999){
    n = 4;
}
else if(x > 99){
    n = 3;
}
else if(x > 9){
    n = 2;
}

cout << "输入的位数是" << n << "位数" << endl;
```

为什么没有写成 (x > 99 && x <= 999)?

因为明确了4位数及以下的非负整数，所以可以简化一些判断

但是任意范围的非负整数怎么办？

任意范围的正整数怎么办？


```
int x;
int n = 1;

cin >> x;
if(x > 999){
    n = 4;
}
else if(x > 99){
    n = 3;
}
else if(x > 9){
    n = 2;
}
```

```
int x;
int n = 1;

cin >> x;
if(x > 9999){
    n = 5;
}
else if(x > 999){
    n = 4;
}
else if(x > 99){
    n = 3;
}
else if(x > 9){
    n = 2;
}
```

```
int x;
int n = 1;

cin >> x;
if(x > 99999){
    n = 6;
}
else if(x > 9999){
    n = 5;
}
else if(x > 999){
    n = 4;
}
else if(x > 99){
    n = 3;
}
else if(x > 9){
    n = 2;
}
```

没完没了！

让计算机数数

- 2345622123834
- 从右往左数，一次划掉一个数字
- 计算机怎么划掉那个数字?

三位数逆序（lab1）——回顾

- 输入一个三位整数，输出其反转后的结果。
- 思路：分别拿出个位、十位、百位。

```
int n;
cin >> n;
int ge = n % 10;
int shi = (n/10) % 10;
int bai = (n/100) % 10;
int reversed = ge * 100 + shi * 10 + bai;
cout << reversed << endl;

return 0;
```

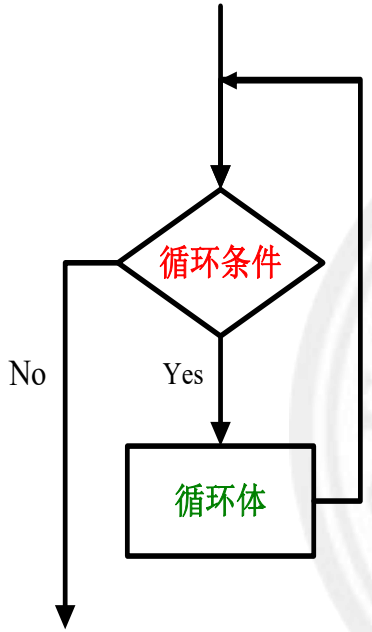
怎么拿到的?

不断除以10再取余

重复划掉一位数

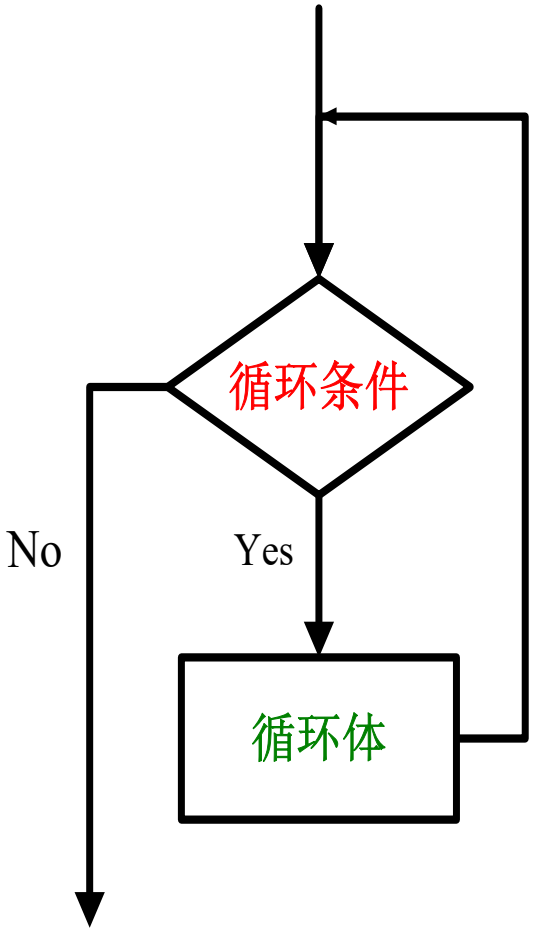
```
int x;
int n = 0;
cin >> x;
if(x > 0){
    n++;
    x /= 10;
    if(x > 0){
        n++;
        x /= 10;
        if(x > 0){
            ...
        }
    }

}
```

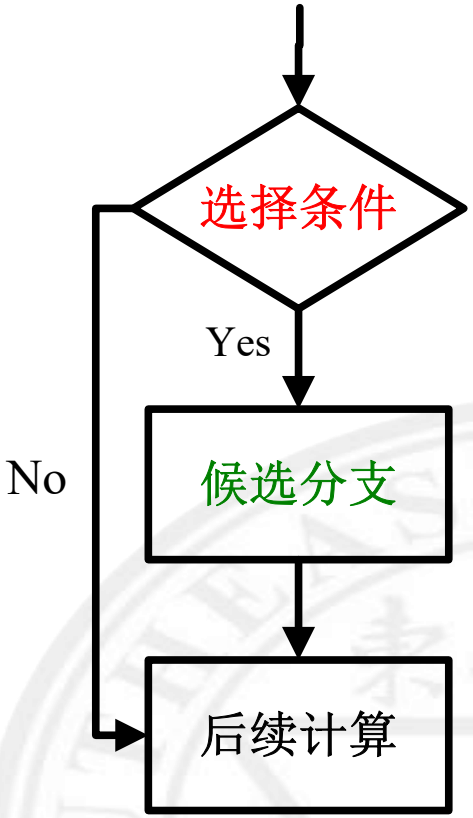
- 看起来还是没完没了
- 但似乎每一段做的事情都是一样的，像是一种循环？



- 循环条件：x > 0
- 循环体：n++; x/=10;

与if语句流程图的对比





和if语句的区别仅仅在于循环要反复多次判断，if只做一次判断

用while循环实现位数统计


```
int x;
int n = 0;
cin >> x;
if(x > 0){
    n++;
    x /= 10;
    if(x > 0){
        n++;
        x /= 10;
        if(x > 0){
            ...
        }
    }

}
```

```
int x;
int n = 0;
cin >> x;
while(x > 0){
    n++;
    x /= 10;
}
```

while语句


```
while(x > 0){
    n++;
    x /= 10;
}
```

- while语句

  while（条件表达式）{
  循环体
  }

- 当条件为真，不断执行{}内的循环体，条件为假则跳过
- 循环体内一定要有改变条件的机会，否则会陷入死循环

不严谨的程序


```
int x;
int n = 0;

cin >> x;
while(x > 0){
    n++;
    x /= 10;
}
```

为了验证程序的正确性，我们会随机假设一些输入，看看输出是什么。

- 输入：352，输出：3
- 输入：2，输出：1
- 输入：0，输出：？

程序测试

程序测试的目的并不只是“看看能不能运行”，而是验证程序在各种情况下都能得到预期的结果。尤其在大型项目中，代码模块众多、逻辑复杂，任何一个小错误都可能引发系统性问题。因此，即使是大公司出品的软件，也经常需要通过更新来修复BUG。

边界条件处理

- 要想测试你的程序是否正确，必须要考虑的输入之一就是边界条件，如果输入范围是 1~100，测试时就要考虑：0、1、100、101 等；对除法操作要考虑：除数为 0；对循环要考虑：循环执行 0 次或仅 1 次；……
- 良好的测试和边界处理不仅是技术要求，更是编程规范的一部分。写代码时应该始终问自己：“如果输入的是边界值、特殊值、甚至错误的值，程序会怎样？”

修正的程序


```
int x;
int n = 0;

cin >> x;
n++;
x /= 10;
while(x > 0){
    n++;
    x /= 10;
}
```

为了验证程序的正确性，我们会随机假设一些输入，看看输出是什么。

- 输入：352，输出：3
- 输入：2，输出：1
- 输入：0，输出：？

do-while循环

数位数的算法


```
int x;
int n = 0;

cin >> x;
n++;
x /= 10;
while(x > 0){
    n++;
    x /= 10;
}
```

step 1. 用户输入x；

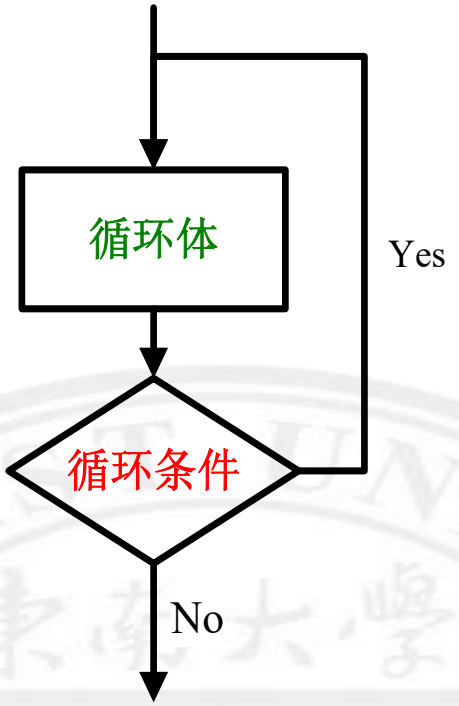
step 2. 初始化n为0；

step 3. n++；

step 4. x = x/10，去掉个位；

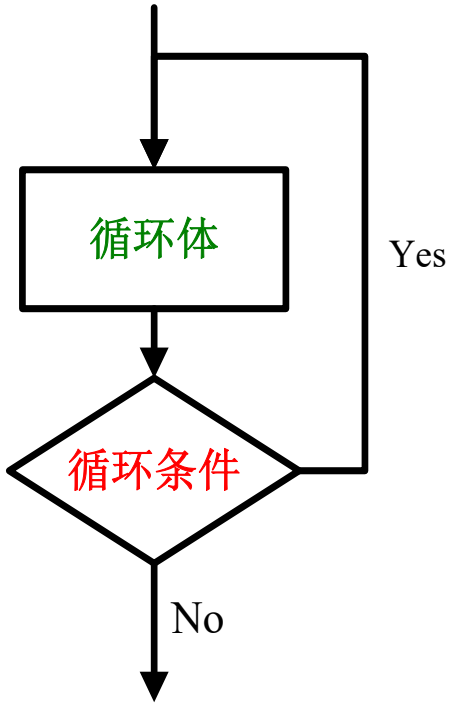
step 5. 如果x>0，回到step 3；

step 6. 否则n就是结果



- 循环条件：x > 0
- 循环体：n++; x/=10;

do-while循环

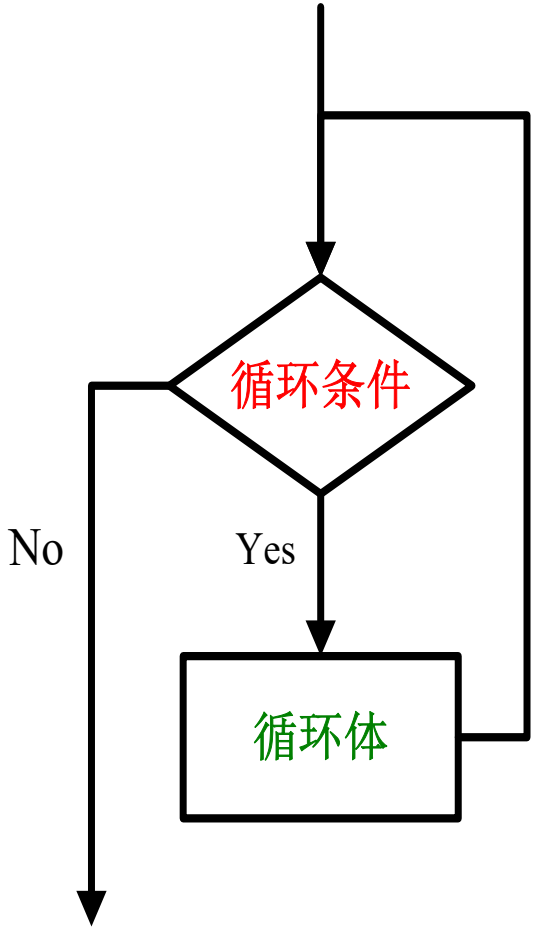


- 在进入循环的时候不做检查，而是在执行完一轮循环体的代码之后，再来检查循环的条件是否满足，如果满足则继续下一轮循环，不满足则结束循环。
- do-while语句

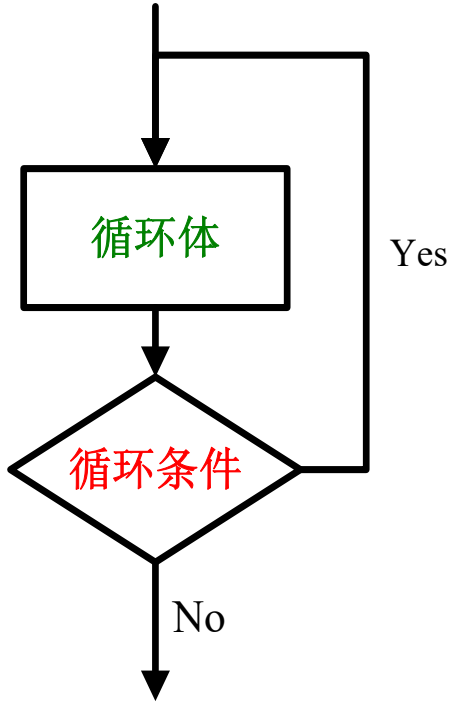
```
do{
        循环体
}while（条件表达式）;
```

- 注意while后面要有分号，表明do-while语句的结束——结束一个语句，要么是大括号“}”，要么是“；”

do-while循环和while循环对比



while循环



do-while循环

- do-while循环和while循环很像，区别只是do-while循环是在循环体执行结束的时候才来判断条件。
- 换言之，无论如何，do-while循环都会执行至少一遍，然后再来判断条件。而while循环可能一遍都不执行。
- 两者共同点是条件满足时执行循环，条件不满足时结束循环。

用do-while循环实现位数统计


```
int x;
int n = 0;
cin >> x;
do{
    x /= 10;
    n++;
}while(x > 0);
```

=

```
int x;
int n = 0;
cin >> x;
n++;
x /= 10;
while(x > 0){
    n++;
    x /= 10;
}
```