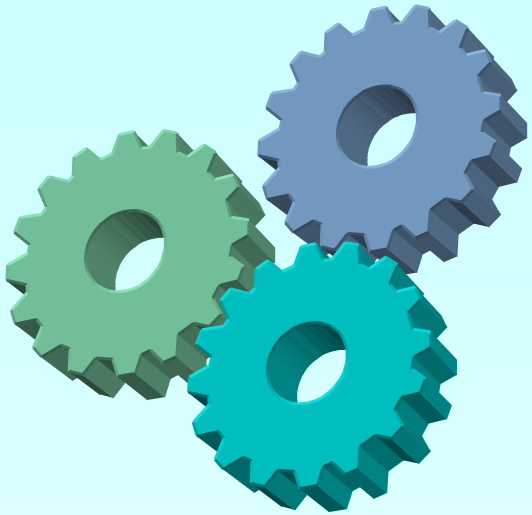
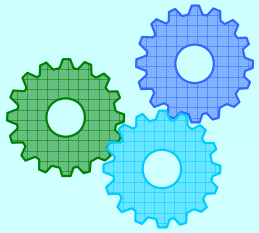


C语言程序设计

李 祥

湖北经济学院信息管理学院

2014.12

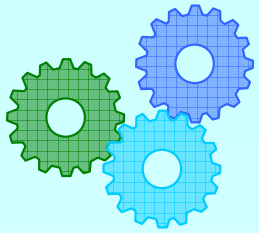


基本语句

§1 语句综述

§2 选择语句

§3 循环语句



§3 循环语句

1. while 语句
2. do-while 语句
3. for 语句
4. break 语句
5. continue 语句

例: 阶乘函数

- 编写函数, 求整数 n ($n \geq 0$)的阶乘 $n!$。

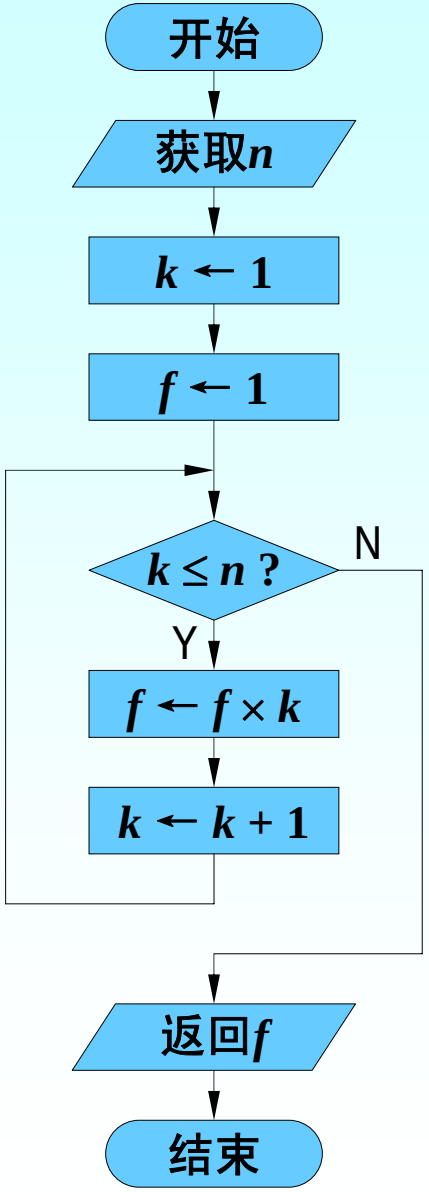
  Fac(n) = 1 × 2 × 3 × … × n, 规定 Fac(0) = 1

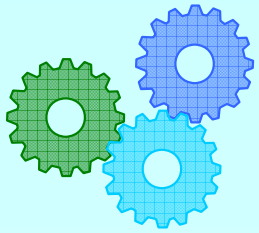
思路:

获取参数n
$k \leftarrow 1$
$f \leftarrow 1$
$f \leftarrow f \times k$
$k \leftarrow k + 1$
$f \leftarrow f \times k$
$k \leftarrow k + 1$
……
$f \leftarrow f \times k$
$k \leftarrow k + 1$
返回结果f

算法:

1. 获取参数n
2. $k \leftarrow 1$
3. $f \leftarrow 1$
4. 当$k \leq n$时重复以下操作
   否则结束循环, 转到第7步
5. $f \leftarrow f \times k$
6. $k \leftarrow k + 1$
   转到第4步
7. 返回结果f





例: 阶乘函数

- **编写函数, 求整数 n ($n \geq 0$)的阶乘 $n!$。**

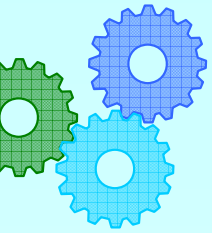
  Fac(n) = 1 × 2 × 3 × … × n, 规定 Fac(0) = 1

1. 获取参数n
2. $k \leftarrow 1$
3. $f \leftarrow 1$
4. 当$k \leq n$时重复以下操作
5. $\quad f \leftarrow f \times k$
6. $\quad k \leftarrow k + 1$
7. 返回结果f

```
double Fac(int n)
{
    int k = 1;
    double f = 1.0;
    while (k <= n)
    {
        f *= k;
        k++;
    }
    return f;
}
```

```
double Fac(int n)
{
    int k = 1;
    double f = 1.0;
    while (k <= n)
    {
        f *= k++;
    }
    return f;
}
```

1. while 语句



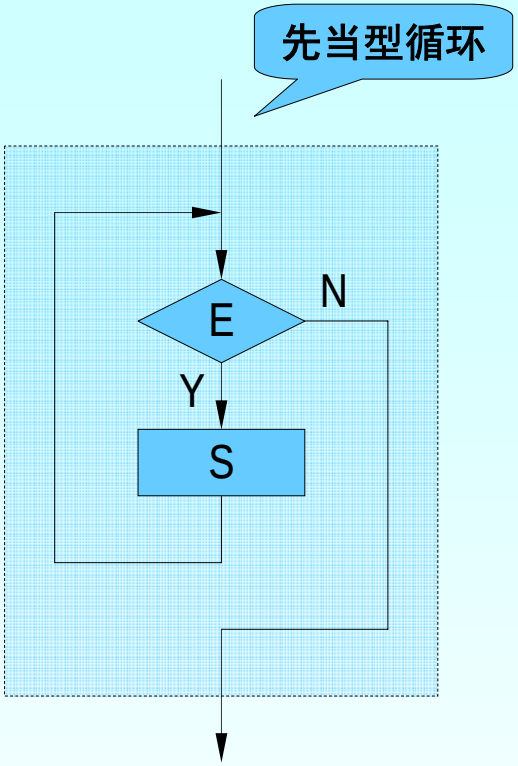
- while 语句的一般形式为:

  while (表达式) 语句

  while (k <= n) f *= k++;

  注: while 语句本身没有分号。

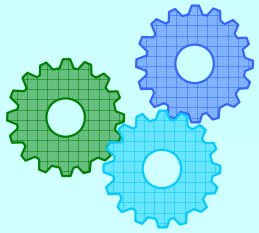
- 执行过程:
  1) 计算表达式, 若值为真(非零值), 则转到第2步; 若值为假(零值), 则直接跳出while语句。
  2) 执行其中的语句。
  3) 回到第1步。
- 其中的语句可以是任何语句, 如表达式语句、复合语句、空语句、流程控制语句等。




```
while (k <= 'C') f *= k++;
```

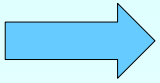
```
while (k <= 'C')
    f *= k++;
```

```
while (k <= 'C')
{
    f *= k++;
}
```

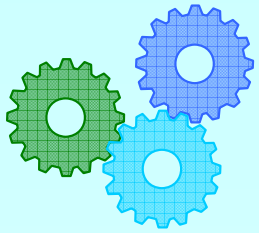


改进

```
double Fac(int n)
{
    int k = 1;
    double f = 1.0;
    while (k <= n)
    {
        f *= k;
        k++;
    }
    return f;
}
```



```
double Fac(int n)
{
    int k = 2;
    double f = 1.0;
    while (k <= n)
    {
        f *= k;
        k++;
    }
    return f;
}
```



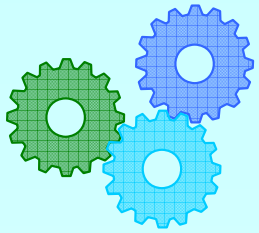
完整的程序

```
#include <stdio.h>

double Fac(int n);

int main()
{
    int a;
    double b;
    printf("a = ? ");
    scanf("%d", &a);
    b = Fac(a);
    printf("b = %g\n", b);
    return 0;
}
```

```
double Fac(int n)
{
    int k = 2;
    double f = 1.0;
    while (k <= n)
    {
        f *= k;
        k++;
    }
    return f;
}
```

```
a = ? 4↵
b = 24
```

```
a = ? 50↵
b = 3.04141e+064
```



判断

```
double Fac(int n)
{
    int k = 1;
    double f = 1.0;
    while (k <= n)
    {
        f *= k;
        k++;
    }
    return f;
}
```

✓

```
double Fac(int n)
{
    int k = 1;
    double f = 1.0;
    while (k <= n)
    {
        f *= k++;
    }
    return f;
}
```

✓

```
double Fac(int n)
{
    int k = 1;
    double f = 1.0;
    while (k++ <= n)
    {
        f *= k;
    }
    return f;
}
```

✗

```
double Fac(int n)
{
    int k = 1;
    double f = 1.0;
    while (++k <= n)
    {
        f *= k;
    }
    return f;
}
```

✓

练习

- **菲波那切数列:**

  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

  编写函数, 求菲波那切数列的第n (n ≥ 1)项。

f_1	+	f_2	→	f
1		1		2
1		2		3
2		3		5
3		5		8

思路: 获取参数n

$k \leftarrow 3$

f_1、$f_2 \leftarrow 1$

$f \leftarrow f_1 + f_2$
$f_1 \leftarrow f_2$
$f_2 \leftarrow f$
$k \leftarrow k + 1$

$f \leftarrow f_1 + f_2$
$f_1 \leftarrow f_2$
$f_2 \leftarrow f$
$k \leftarrow k + 1$

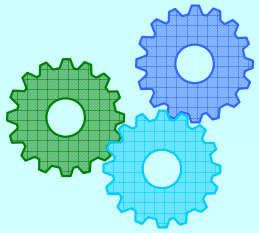
……

$f \leftarrow f_1 + f_2$
$f_1 \leftarrow f_2$
$f_2 \leftarrow f$
$k \leftarrow k + 1$

返回结果f

算法:

1. 获取参数n
2. $k \leftarrow 3$
3. f_1、$f_2 \leftarrow 1$
4. 如果n ≤ 2, 则
5. $f \leftarrow 1$
6. 否则
7. 当$k \leq n$时重复以下操作
   否则结束循环, 转到第12步
8. $f \leftarrow f_1 + f_2$
9. $f_1 \leftarrow f_2$
10. $f_2 \leftarrow f$
11. $k \leftarrow k + 1$
    转到第7步
12. 返回结果f



练习

- **菲波那切数列:**

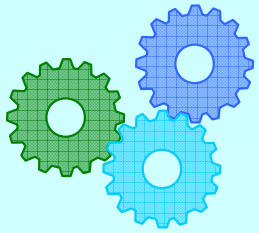
  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

  编写函数, 求菲波那切数列的第n (n ≥ 1)项。

算法:

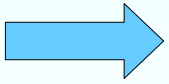
1. 获取参数n
2. $k \leftarrow 3$
3. f_1、$f_2 \leftarrow 1$
4. 如果n ≤ 2, 则
5. $f \leftarrow 1$
6. 否则
7. 当$k \le n$时重复以下操作
   否则结束循环, 转到第12步
8. $f \leftarrow f_1 + f_2$
9. $f_1 \leftarrow f_2$
10. $f_2 \leftarrow f$
11. $k \leftarrow k + 1$
    转到第7步
12. 返回结果f

```
double Fib(int n)
{
    int k = 3;
    double f1 = 1.0, f2 = 1.0, f;
    if (n <= 2)
    {
        f = 1.0;
    }
    else
    {
        while (k <= n)
        {
            f = f1 + f2;
            f1 = f2;
            f2 = f;
            k++;
        }
    }
    return f;
}
```

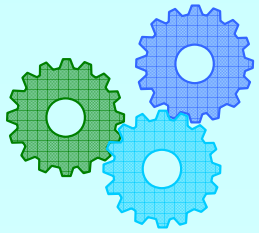


优化

```
double Fib(int n)
{
    int k = 3;
    double f1 = 1.0, f2 = 1.0, f;
    if (n <= 2)
    {
        f = 1.0;
    }
    else
    {
        while (k <= n)
        {
            f = f1 + f2;
            f1 = f2;
            f2 = f;
            k++;
        }
    }
    return f;
}
```



```
double Fib(int n)
{
    int k = 3;
    double f1 = 1.0, f2 = 1.0, f = 1.0;
    while (k <= n)
    {
        f = f1 + f2;
        f1 = f2;
        f2 = f;
        k++;
    }
    return f;
}
```



完整的程序

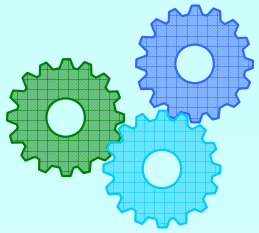
```
#include <stdio.h>

double Fib(int n);

int main()
{
    int a;
    double b;
    printf("a = ? ");
    scanf("%d", &a);
    b = Fib(a);
    printf("b = %g\n", b);
    return 0;
}
```

```
double Fib(int n)
{
    int k = 3;
    double f1 = 1.0, f2 = 1.0, f = 1.0;
    while (k <= n)
    {
        f = f1 + f2;
        f1 = f2;
        f2 = f;
        k++;
    }
    return f;
}
```

```
a = ? 4↵
b = 24
```

```
a = ? 50↵
b = 1.25863e+010
```

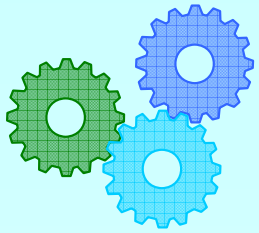
速度极快!



常见的写法

```
double Fac(int n)
{
    int k = 2;
    double f = 1.0;
    while (k <= n)
    {
        f *= k;
        k++;
    }
    return f;
}
```

```
double Fib(int n)
{
    int k = 3;
    double f1 = 1.0, f2 = 1.0, f = 1.0;
    while (k <= n)
    {
        f = f1 + f2;
        f1 = f2;
        f2 = f;
        k++;
    }
    return f;
}
```



怎样编写循环程序

- 明确需要重复的操作

  将需要反复执行的操作放到循环体中。如: 累乘运算: f *= k。

- 循环前的准备工作

  在循环开始前应准备好基础数据。如: 初始化累乘器和计数器变量: f = 1.0、k = 2。

- 循环中的数据调整工作。

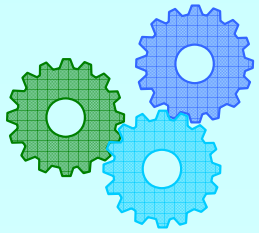
  每一轮循环的动作是相同的, 但所操纵变量的值却是不同的。如: 循环末尾应该及时调整变量的值, 为下一轮循环准备数据。如: 调整计数器变量的值, 变成下一轮循环累乘操作所需的乘数: k++。

- 确立循环条件。

  循环必须在重复执行有限的次数之后结束, 否则将陷入无限循环(俗称“死循环”)。如: 计数器变量k的值从1(或2)开始, 逐步增大到n, 当超过n时结束循环, 因此循环条件为: k <= n。

- 检验程序是否正确。

  检查一般情况和特殊情况下程序功能是否正确, 是否需要做特殊处理。如: 容易验证当n = 3, 4, ...时, 结果是正确的。检验当n = 0, 1时, 结果也正确, 因此不需要做特殊处理。

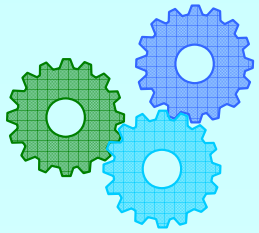


例: 显示n个星号

- 编写函数, 在屏幕上显示n个星号。

```
#include <stdio.h>

void Star(int n);

int main()
{
    int a;
    printf("a = ? ");
    scanf("%d", &a);
    Star(a);
    return 0;
}
```

```
void Star(int n)
{
    int k = 1;
    while (k <= n)
    {
        putchar('*');
        k++;
    }
}
```



其它版本

```
void Star(int n)
{
    int k = 1;
    while (k <= n)
    {
        putchar('*');
        k++;
    }
}
```

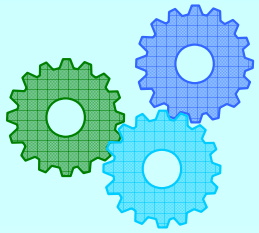
```
void Star(int n)
{
    int k = n;
    while (k > 0)
    {
        putchar('*');
        k--;
    }
}
```

```
void Star(int n)
{
    int k = n;
    while (k)
    {
        putchar('*');
        k--;
    }
}
```

```
void Star(int n)
{
    int k = 1;
    while (k++ <= n)
    {
        putchar('*');
    }
}
```

```
void Star(int n)
{
    int k = n;
    while (k-- > 0)
    {
        putchar('*');
    }
}
```

```
void Star(int n)
{
    int k = n;
    while (k--)
    {
        putchar('*');
    }
}
```



练习

- **编写函数** $P(x, n) = 1 + x + x^2 + x^3 + x^4 + \ldots + x^n$ **。**

观点一:

$$P(x, n-1) = 1 + x + x^2 + x^3 + x^4 + \ldots + x^{n-1}$$

$$P(x, n) = 1 + x + x^2 + x^3 + x^4 + \ldots + x^{n-1} + x^n$$

$$P(x, n) = P(x, n-1) + x^n$$

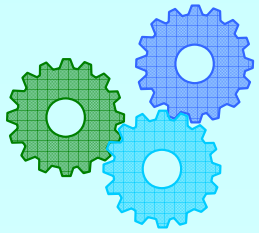
观点二:

$$P(x, n-1) = 1 + x + x^2 + x^3 + x^4 + \ldots + x^{n-1}$$

$$P(x, n) = 1 + x + x^2 + x^3 + x^4 + \ldots + x^{n-1} + x^n$$

$$= 1 + x\,(1 + x + x^2 + x^3 + \ldots + x^{n-2} + x^{n-1})$$

$$P(x, n) = 1 + x \cdot P(x, n-1)$$



练习

- 编写函数 $P(x, n) = 1 + x + x^2 + x^3 + x^4 + \ldots + x^n$ 。

方法一

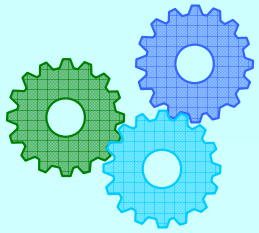
```
double P(double x, int n)
{
    double p = 1.0, y = 0.0;
    int k = 0;
    while (k <= n)
    {
        y += p;
        p *= x;
        k++;
    }
    return y;
}
```

k用于计数, 从0数到n
p为一项的值, 由前一项推算出后一项
y为函数值, 从0开始逐项累加求和

方法二

```
double P(double x, int n)
{
    double y = 0.0;
    int k = 0;
    while (k <= n)
    {
        y = 1 + x * y;
        k++;
    }
    return y;
}
```

k用于计数, 从0数到n
y为函数值, 由前k项的和推算出前k+1项的和



练习

- **编写叠数函数** $\text{Redup}(d,n)=\underbrace{dd\cdots d}_{n个}$。

例: Redup(8, 5) = 88888

观点一:

$$88888 = 8888 + 8 \times 10^4$$

$$\text{Redup}(8, 5) = \text{Redup}(8, 4) + 8 \times 10^4$$

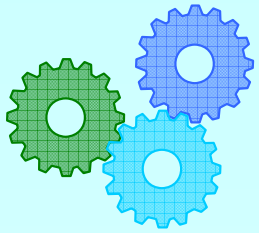
$$\text{Redup}(d, n) = \text{Redup}(d, n - 1) + d \times 10^{n-1}$$

观点二:

$$88888 = 8888 \times 10 + 8$$

$$\text{Redup}(8, 5) = \text{Redup}(8, 4) \times 10 + 8$$

$$\text{Redup}(d, n) = \text{Redup}(d, n - 1) \times 10 + d$$



练习

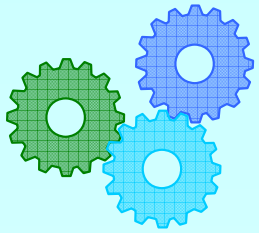
- **编写叠数函数** $\mathrm{Redup}(d,n)=\underbrace{dd\cdots d}_{n个}$。

方法一

```
int Redup(int d, int n)
{
    int p = d, k = 1, r = 0;
    while (k <= n)
    {
        r += p;
        p *= 10;
        k++;
    }
    return r;
}
```

方法二

```
int Redup(int d, int n)
{
    int k = 1, r = 0;
    while (k <= n)
    {
        r = r * 10 + d;
        k++;
    }
    return r;
}
```



统计字符数量

- 编写程序, 输入一段文字(以回车结束), 统计字符的数量。

思路:

字符数清零
读取第一个字符
字符数加一
读取下一个字符
字符数加一
读取下一个字符
字符数加一
读取下一个字符
……
字符数加一
读取下一个字符
输出字符数n

算法:

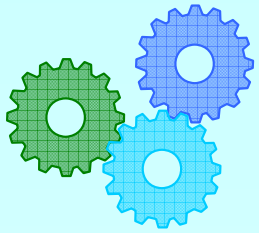
1. 开始
2. n ← 0
3. 读取第一个字符c
4. 当c ≠ 回车时重复以下操作
   否则转到第7步
5. n ← n + 1
6. 读取下一个字符c
   回到第4步
7. 输出n
8. 结束

```
int main()
{
    int n = 0;
    char c;
    c = getchar();
    while ( c != '\n' )
    {
        n++;
        c = getchar();
    }
    printf("%d\n", n);
    return 0;
}
```

读第一个字符

读下一个字符

Do one thing at a time,
and do it well.↵
39



逗号运算

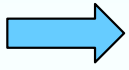
运算符	作用	结合性
,	连续计算	→

表达式$_1$, 表达式$_2$, … , 表达式$_n$

- 依次计算各个表达式的值，将最后一个表达式的值作为整个表达式的值。
- 当完成某计算任务需要写多个表达式，但语法限定只能写一个表达式时，可以利用逗号运算符来实现。

```
x = ( 1 , 2 , 3 , 4 , 5 );    5
```

```
int a, b, c;
c = ( a = 1 , b = 2 , a + b );
```



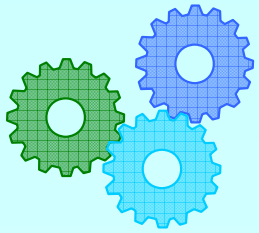
```
a = 1;
b = 2;
c = a + b;
```

三个表达式(语句)

```
c = (a = 1) + (b = 2);
```

```
a = 1, b = 2, c = a + b;
```

一个表达式(语句)



统计字符数量

- 编写程序，输入一段文字(以回车结束)，统计字符的数量。

```
int main()
{
    int n = 0;
    char c;
    c = getchar();
    while ( c != '\n' )
    {
        n++;
        c = getchar();
    }
    printf("%d\n", n);
    return 0;
}
```

读第一个字符

读下一个字符

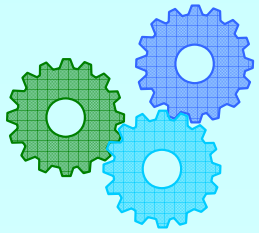
```
int main()
{
    int n = 0;
    char c;
    while ( c = getchar(), c != '\n' )
    {
        n++;
    }
    printf("%d\n", n);
    return 0;
}
```

读一个字符

判断是否为回车

```
int main()
{
    int n = 0;
    char c;
    while ( ( c = getchar() ) != '\n' )
    {
        n++;
    }
    printf("%d\n", n);
    return 0;
}
```

读一个字符，判断是否为回车



练习

- 仿照scanf函数, 自编函数以十进制格式输入自然数。
  void ScanDec(int *x);

2 5 1 7 3

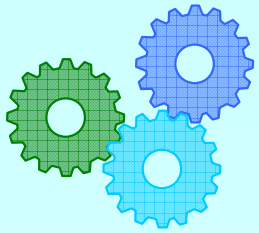
思路:

				0
2	0	× 10	+ 2 =	2
5	2	× 10	+ 5 =	25
1	25	× 10	+ 1 =	251
7	251	× 10	+ 7 =	2517
3	2517	× 10	+ 3 =	25173

```
void ScanDec(int *x)
{
    char c;
    int n = 0;
    c = getchar();
    while (isdigit(c))
    {
        n = n * 10 + c - '0';
        c = getchar();
    }
    *x = n;
}
```

读第一个字符

读下一个字符

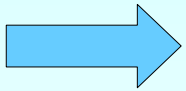


改进

```
void ScanDec(int *x)
{
    char c;
    int n = 0;
    c = getchar();
    while ( isdigit(c) )
    {
        n = n * 10 + c - '0';
        c = getchar();
    }
    *x = n;
}
```

读第一个字符

读下一个字符



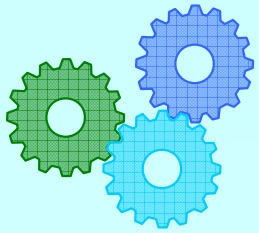
```
void ScanDec(int *x)
{
    char c;
    int n = 0;
    while ( c = getchar() , isdigit(c) )
    {
        n = n * 10 + c - '0';
    }
    *x = n;
}
```

读一个字符

判断是否为数字

```
void ScanDec(int *x)
{
    char c;
    int n = 0;
    while ( isdigit( c = getchar() ) )
    {
        n = n * 10 + c - '0';
    }
    *x = n;
}
```

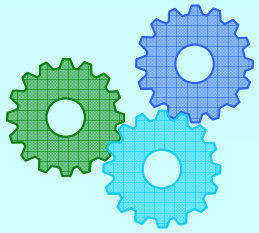
读一个字符，
判断是否为数字



完美版

- 自动跳过前导空白字符;
- 可输入正、负号;
- 退回最后一个不符合要求的字符;
- 逻辑值为逻辑值, 标识是否成功。

```
int ScanDec(int *x)
{
    char c;
    int n = 0, ok = 0, neg = 0;
    c = getchar();
    while ( isspace(c) )
    {
        c = getchar();
    }
    if (c == '+' || c == '-')
    {
        neg = c == '-';
        c = getchar();
    }
    while ( isdigit(c) )
    {
        ok = 1;
        n = n * 10 + c - '0';
        c = getchar();
    }
    ungetc(c, stdin);
    if (ok)
    {
        *x = neg ? -n : n;
    }
    return ok;
}
```



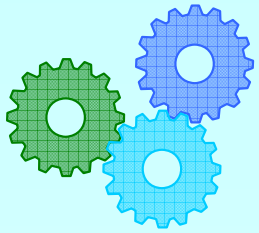
完整的程序

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

int ScanDec(int *x);
void PrintDec(int x);

int main()
{
    int a;
    printf("a = ? ");
    ScanDec(&a);
    printf("a = ");
    PrintDec(a);
    putchar('\n');
    return 0;
}
```

```
a = ? ␣␣-3876152␣↵
a = -3876152
```

```
int ScanDec(int *x)
{
    char c;
    int n = 0, ok = 0, neg = 0;
    c = getchar();
    while ( isspace(c) )
    {
        c = getchar();
    }
    if (c == '+' || c == '-')
    {
        neg = c == '-';
        c = getchar();
    }
    while ( isdigit(c) )
    {
        ok = 1;
        n = n * 10 + c - '0';
        c = getchar();
    }
    ungetc(c, stdin);
    if (ok)
    {
        *x = neg ? -n : n;
    }
    return ok;
}
```

```
void PrintDec(int x)
{
    if (x < 0)
    {
        putchar('-');
        PrintDec(-x);
    }
    else // x >= 0
    {
        if (x >= 10)
        {
            PrintDec(x / 10);
        }
        putchar(x % 10 + '0');
    }
}
```



练习

- 仿照scanf函数, 自编函数以二进制格式输入自然数。
  void ScanBin(int *x);

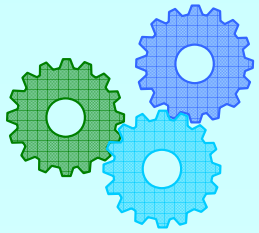
1 0 0 1 0

思路:

				0
1	0	× 10	+ 1 =	1
0	1	× 10	+ 0 =	10
0	10	× 10	+ 0 =	100
1	100	× 10	+ 1 =	1001
0	1001	× 10	+ 0 =	10010

```
int IsBDigit(char x)
{
    int ok;
    if ('0' <= x && x <= '1')
    {
        ok = 1;
    }
    else
    {
        ok = 0;
    }
    return ok;
}
```

```
int ScanBin(int *x)
{
    char c;
    int n = 0, ok = 0, neg = 0;
    c = getchar();
    while ( isspace(c) )
    {
        c = getchar();
    }
    if (c == '+' || c == '-')
    {
        neg = c == '-';
        c = getchar();
    }
    while ( IsBDigit(c) )
    {
        ok = 1;
        n = n * 2 + c - '0';
        c = getchar();
    }
    ungetc(c, stdin);
    if (ok)
    {
        *x = neg ? -n : n;
    }
    return ok;
}
```



完整的程序

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

int ScanBin(int *x);
void PrintBin(int x);
int IsBDigit(char x);

int main()
{
    int a;
    printf("a = ? ");
    ScanBin(&a);
    printf("a = ");
    PrintBin(a);
    putchar('\n');
    return 0;
}
```

```
a = ? ␣␣-1001101␣↵
a = -1001101
```

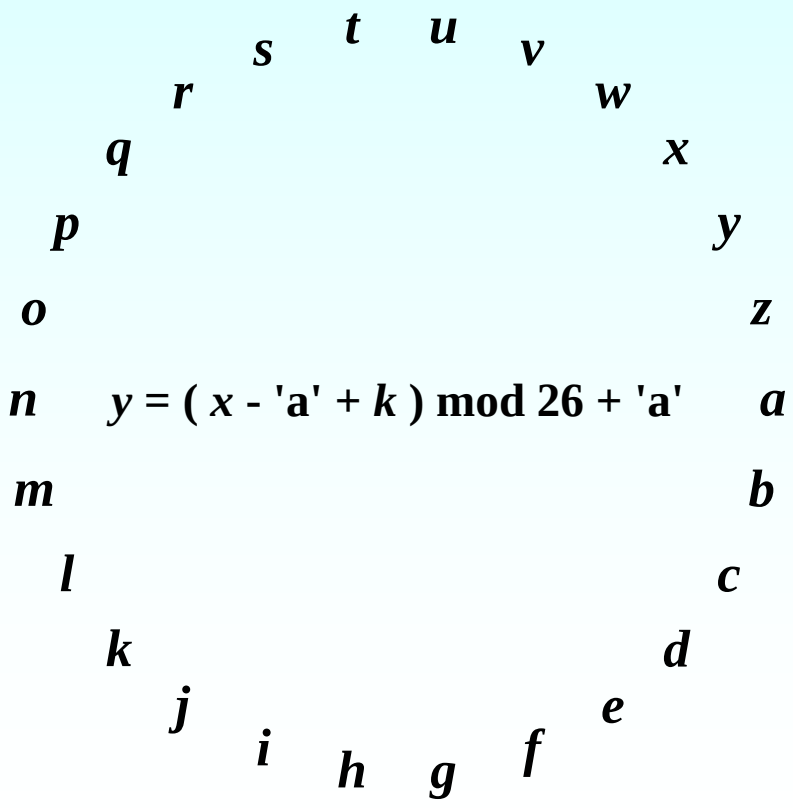
```
int ScanBin(int *x)
{
    char c;
    int n = 0, ok = 0, neg = 0;
    c = getchar();
    while ( isspace(c) )
    {
        c = getchar();
    }
    if (c == '+' || c == '-')
    {
        neg = c == '-';
        c = getchar();
    }
    while ( IsBDigit(c) )
    {
        ok = 1;
        n = n * 2 + c - '0';
        c = getchar();
    }
    ungetc(c, stdin);
    if (ok)
    {
        *x = neg ? -n : n;
    }
    return ok;
}
```

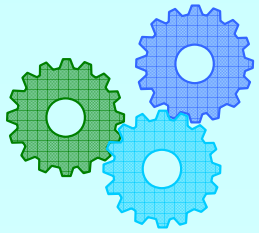
```
void PrintBin(int x)
{
    if (x < 0)
    {
        putchar('-');
        PrintBin(-x);
    }
    else // x >= 0
    {
        if (x >= 2)
        {
            PrintBin(x / 2);
        }
        putchar(x % 2 + '0');
    }
}

int IsBDigit(char x)
{
    int ok;
    if ('0' <= x && x <= '1')
    {
        ok = 1;
    }
    else
    {
        ok = 0;
    }
    return ok;
}
```

加密

- 有一个简单的加密方法是: 将26个字母摆成下图所示的一个环, 找到明文字母后按顺时针方向走几步就得到对应密文字母。请编写程序, 输入步长(即密钥)和一段明文文字(以回车结束), 输出加密后的密文文字。

```
char Encode(char x, int k)
{
    char y;
    if ( isupper(x) )
    {
        y = (x - 'A' + k) % 26 + 'A';
    }
    else if ( islower(x) )
    {
        y = (x - 'a' + k) % 26 + 'a';
    }
    else
    {
        y = x;
    }
    return y;
}
```



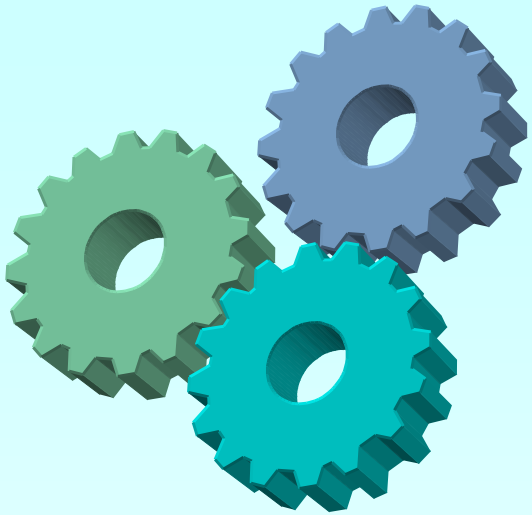


加密

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

char Encode(char x, int k);

int main()
{
    char p, c;
    int k;
    printf("密钥: ");
    scanf("%d", &k); // 输入密钥(步长)
    puts("明文:");
    fflush(stdin); // 清除缓冲区
    p = getchar(); // 读取第一个明文字符
    puts("密文:");
    while ( p != '\n' )
    {
        c = Encode(p, k); // 加密
        putchar(c); // 输出密文字符
        p = getchar(); // 读取下一个明文字符
    }
    putchar('\n');
    return 0;
}
```

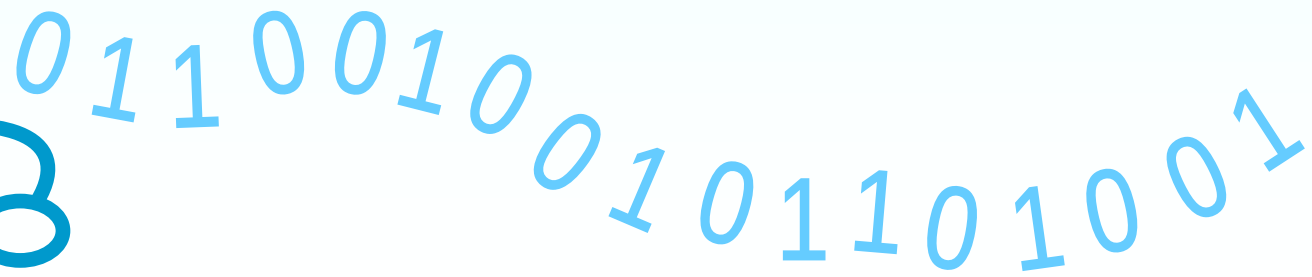
```
char Encode(char x, int k)
{
    char y;
    if ( isupper(x) )
    {
        y = (x - 'A' + k) % 26 + 'A';
    }
    else if ( islower(x) )
    {
        y = (x - 'a' + k) % 26 + 'a';
    }
    else
    {
        y = x;
    }
    return y;
}
```

```
密钥: 3↵
明文:
Hello! I'm John Smith.↵
密文:
Khoor! L'p Mrkq Vplwk.
```



勤有功，戏无益，
戒之哉，宜勉力。

——《三字经》



勤奋才能取得成功，不严肃认真对待生活与事业则毫无益处。后生晚辈应该谨记这些道理，努力奋斗争取幸福光明的前程。