内容简介

本书按照快速了解、详细解读、深入进阶的顺序展开，目的是使读者尽快领略 C 语言的全貌，进而产生详解的兴趣和深入探讨的愿望。依据此思想本书分为三篇组织内容：第一篇感知篇，通过一个简单任务驱动，让读者在很短的时间了解 C 语言的主要知识点及 C 语言所能完成的任务；第二篇详解篇，逐步展开对每个知识点的详细研究，按照理解、语法规则、使用方式进行深入的探讨，学会计算机语言的基本要素；第三篇进阶篇分析、设计、实现一些有一定难度的案例，深层次运用各个知识点，进而培养读者结构化程序设计的能力。本书组织方式完全符合人类对语言学习过程：模仿、理解、应用，也符合软件工程迭代式开发过程的思想，对读者从事软件研发大有裨益。

前　言

从事了多年的C语言教学工作，怎样提高教学水平一直是萦绕笔者脑海的问题。总结学生学习中的问题，大致有以下三点：第一，学习过程的初期对计算机语言没有整体的认识，不了解C程序所能完成的功能，以及个知识点所扮演的角色，学习时没有明确的目的性；第二，C语言本身知识点繁多、晦涩难懂，对于初次接触计算机语言的读者来说难于掌握，久而久之，产生畏难情绪，因而丧失了学习的兴趣；第三，只能初步了解各知识点语法规则，不能够综合运用、融会贯通，更谈不上程序设计。

C语言具备一般语言的特点，笔者想计算机语言的学习能不能借鉴人类自然语言的学习方式？人类对自然语言的学习过程，总是从模仿开始，这个阶段只知道这么说，并不理解其语法规则；然后在生活、学习当中逐步懂得语言的结构及语法；最后，通过阅读、分析文章和撰写文件才能够熟练地应用语言。

C语言的学习是为了软件的研发，回顾多年的软件研发经验，其中最困难的当然是了解用户的需求。如果不熟悉问题领域的需求，学习起来很困难。如何解决呢？我们通常采用采用迭代式的软件开发方法，因为开始很难祥知开发问题领域的所有问题，可以选一条主线完成一个基本结构，然后经过多次迭代，每次软件有一定增量，使得每次的迭代产品都越来越接近目标系统。

于是，笔者萌生这样一个想法，C语言对于初学来说，知识点繁多、且难于理解，想一次完全掌握对于大多数读者来说相当困难。C语言的学习与人类学习自然语言以及软件开发过程中理解问题域的知识非常相似，就可以把人类对自然语言的学习过程和软件开发过程的迭代方法应用到C语言学习当中来，我们称之为“三次迭代法”。

三次迭代法采用三个周期，每个周期有一定的增量，递增的学习各个知识点。具体方法如下：感知部分（第一次迭代）让读者在很短的时间内亲身感受到C语言的全貌，建立起对C语言的兴趣，给出一些样例，并给出相似的模仿习题；详解部分（第二次迭代）逐步展开对每个知识点的详细研究，对每个知识点按照理解、语法规则、使用方式的次序做深入的探讨；进阶部分（第三次迭代）综合运用各个知识点分析、设计、实现一些有一定难度的案例，不但再次深层次运用各个知识点，而且培养了程序设计能力。这就完全符合了人类学习自然语言的过程，同时，三次迭代过程每次迭代都是对前一次的深化。

三次迭代法的教材组织与教学方法对传统模式的变革，类似于软件开发过程由“瀑布式开发”到的“迭代式开发”演变，对读者从事软件研发也会大有裨益。

有了以上想法，笔者异常兴奋，废寝忘食，奋笔疾书，望早日呈书于读者面前，希望为C语言的学习者提供些许帮助。

鉴于时间仓促，笔者水平有限，书中难免有纰漏，欢迎广大读者多提宝贵意见。

编　者

2010年6月

目　　录

第 7 章 选择结构

7.1 理解选择结构

人生总是面临许多选择，比如：毕业后是继续深造还是工作？如果有保送攻送读本校研究生的机会，是把握机会还是选择放弃？面临各种选择，必须客观地比较各种因素，冷静权衡利，才能作出准确的判断。现实生活中，人们常需要根据某些条件，作出各种决定，比如：如果明天下雨就不去郊游，而呆在家里网游。这里将“是否下雨”作为明天行动方案的依据，是选择郊游还是网游。

计算机不仅能够进行复杂的数学计算，还能如人一样进行逻辑分析和判断，这些都是通过程序实现的。C 语言是语句的集合，在前面的程序中，这些语句都是按照它们出现的顺序逐条执行的。但在实际应用中，常需要根据特定条件来选择执行哪些语句，或者改变语句的执行顺序。这样就需要一种判断机制来确定条件是否成立，并指示计算机执行相应的语句。C 语言提供如下语句，使程序具有这种判断能力：

（1）if 语句以及 if-else 语句

（2）switch 语句

（3）条件表达式语句

（4）goto 语句

这些语句具有控制程序流程的作用，即能够操作程序执行的顺序，因此称为控制语句。本章讨论由这些控制语句构成的选择结构。

7.2 简单分支语句

程序常用来处理日常生活和工作中的一些业务。比如银行信息系统中涉及信用卡管理功能，对于信用卡业务有如下规定：如果申请人具有一定经济偿还能力，则可以办理信用卡；在某次刷卡消费时，如果信用卡用户的余额不足，可以通过透支付款；如果透支总额小于透支额度，则成功透支；等等。这些规定可以映射成程序所能处理的逻辑业务，用选择结构实现。本节讨论简单 if 语句构成的单分支选择结构和 if-else 语句构成的双分支选择结构。

7.2.1 单分支 if 语句

if 语句是选择结构中最常用的语句，具有强大而灵活的判断能力。简单的 if 语句可以构成单分支选择结构，其基本形式如下：

```
if（ 表达式 ）
{  语句块;  }
```

执行该语句时先计算用于判断的表达式，然后根据该表达式的值选择是否执行花括号中的语句块。若判断条件成立，即表达式的值为逻辑真（true，非 0）时，执行该语句块；否则，当表达式的值为逻辑假（false，即为 0）时，不执行语句块，直接执行花括号后的下一条语句。

例 7.1　求整数 a 的绝对值 |a |

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

```
    scanf(" %d",&a);
    /*if 语句*/
    if ( a<0 )
    {
        a   = -a ;
    }
    printf(   "|a|=%d\n " , a ) ;
    return 0;
}
```

使用 if 语句应注意如下问题：

（1） 花括号中的执行语句块可以包含一条或多条语句，如果只有一条语句，则可以省略“{}”。例 7.1 中的 if 语句可以写成如下形式：

```
if (a<0)
  a -= a;
```

但是有时忘记花括号会造成逻辑错误，最好不省略“{}”。

（2） if 语句的圆括号后不应有分号，否则会造成逻辑错误。例如，例 7.1 中的 if 语句写成如下形式：

```
if (a<0) ;
  a-=a;
```

则相当于 if 语句中的可执行语句为空语句，不管条件表达式 a<0 的值为真还是假，总是执行操作 a-=a，这是一种不易发现的逻辑错误。

（3） 判断的表达式通常为关系表达式或者逻辑表达式，也可以是具有逻辑值的其他类型表达式。例如，判断整数 num 是否为奇数时，用 if 语句实现可写成以下形式：

```
if( num% 2 != 0)     printf（“%d 是奇数”, &num）;
if( num% 2 )       printf（“%d is odd number”,&num）;
```

特别注意，赋值表达式也可以作为判断表达式，例如

```
if ( sum = a + b )     printf（“和不为零”）;
```

但要谨慎设计表达式，不能混淆赋值运算符“＝”和逻辑等运算符“==”，例如

```
if ( disc==0 )     printf（“有两相等实根”）;
```

如果写成如下形式为逻辑错误：

```
if ( disc=0 )     printf（“有两相等实根”）;
```

7.2.2 双分支 if—else 语句

用 if 构成的单分支选择结构，可用于判断某种操作是否执行。有时需要根据条件，判断执行这种操作还是另一种操作。这种具有两种选择的逻辑问题，可以用 if-else 构成的双分支选择结构实现，其语法形式为：

```
if（ 表达式 ）
{   语句 1;    }
else
{   语句 2 ;    }
```

其执行顺序为：先计算表达式的值，如果表达式值为逻辑真（true，值为 0），则执行语句块 1，否则执行语句块 2。双分支 if-else 语句的执行过程如图 7.1 所示。

例 7.2 求 a + | b | 的值

求解此问题的算法比较简单，不作深入分析，其流程图见图 7.2。

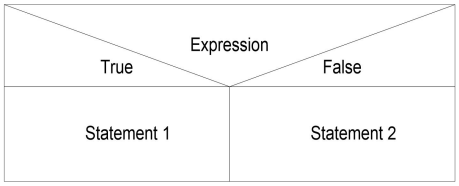

图 7.1 if-else 语句的执行流程

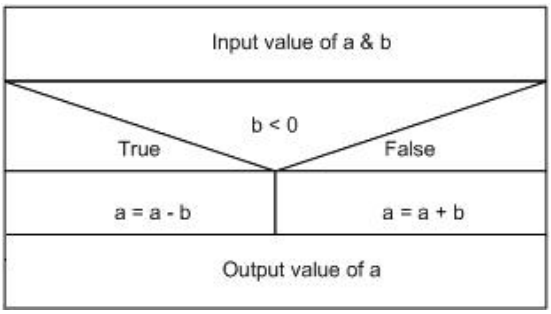

图 7.2　求 a + | b | 算法流程图

```
# include <stdio.h>
int main( )
{
      int a, b;
      puts( " please input 2 numbers :\n ");
      scanf("%d %d", &a, &b);
      /*if-else 语句*/
      if ( b<0 )
      {
            a   -=   b ;
      }
      else
      {
            a   +=   b;
      }
      printf(   "a + | b |=%d\n " , a ) ;
      return 0;
}
```

使用 if-else 语句应注意如下问题：

（1）　双分支选择结构中 if 与 else 配对使用，else 不能单独使用，在单分支结构中省略 else，单独使用 if。

花括号中的执行语句块可以包含一条或多条语句，如果只有一条语句，则可以省略“{}”。但是有时忘记花括号会造成逻辑错误，最好不省略“{}”。

（2）对于 if-else 语句构成的双分支选择结构，可以用简单的 if 语句实现。例 7.2 的分支结构也可以写成：

```
if ( b<0 )
   sum = a - b ;
 if (b>=0)
   sum = a + b ;
```

或者

```
 sum = a + b ;
 if ( b<0 )
    sum = a - b ;
```

使用条件运算符（？:）也可以实现这种选择结构，形式更为简洁。例如，

```
sum = (b<0) ? a-b : a+b;
```

（3）实现选择结构的前提是构造合适的条件。通常使用逻辑表达式实现复杂的判断条件。例如，判断闰年的问题

```
if( year％4==0  &&  year％100!=0  ||  year％400==0 )
    leap = 1;
else
    leap = 0;
```

如何设计判断条件是学习 if 语句的难点。在实现选择结构时，经常由于判断条件构造不合理造成逻辑错误，在调试程序的时候可以采用单步调式的方法跟踪程序的流程，或者在各个分支中添加输出语句来判断程序的执行情况。

7.3 多分支语句

银行开设的信用卡业务规定：用户帐户透支后需按时还款，对于本月的透支消费，如果当月内及时还款，则累计信用积分；否则，不及时还款需按比例缴纳滞纳金；如果 6 个月内不还款，则信用度降级并缴纳罚款；对于更恶劣的欠款行为，将给予严重惩罚，如向法院提起诉讼。对于此类较为复杂的逻辑问题，需要构造多个分支来处理各种情况。本节讨论用嵌套的 if 语句、else if 语句以及 switch 语句构成的多分支选择结构。

7.3.1 嵌套 if 语句

当 if 语句中又包含另一个 if 语句时，就构成了 if 语句的嵌套形式。其一般可表示如为：

```
if(表达式)
{
    if 语句；
}
```

同理，在 if-else 的两个分支中，也可以嵌套其他 if 语句，构成多种选择关系。一般形式可表示如下：

```
if(表达式)
    if 语句；
else
    if 语句；
```

完整的嵌套 if-else 语句表述成如下形式：

```
if (表达式 1)
{
  if (表达式 2)
  {  语句 1  }
  else
  {  语句 2  }
}
else
{
   if (表达式 3)
    {  语句 3  }
    else
```

```
    {   语句 4   }
}
```

执行的流程图见图 7.3。其实，该嵌套语句可以转换成简单的 if 语句并列的形式：

```
if (表达式 1 &&  表达式 2)   {   语句 1   }
if (表达式 1 && !表达式 2)   {   语句 2   }
if (!表达式 1&&  表达式 3)   {   语句 3   }
if (!表达式 1&& !表达式 3)   {   语句 4   }
```

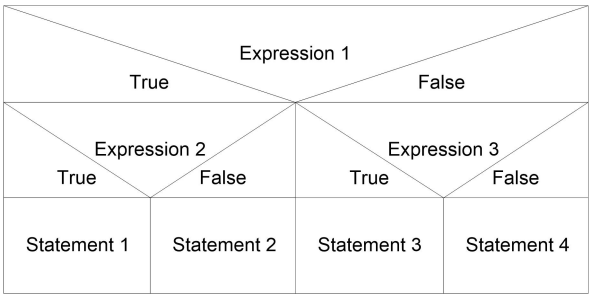

图 7.3 嵌套 if-else 语句执行的流程图

例 7.3 求解一元二次方程 $ax^2 + bx + c = 0$ 的根

按照数学中方程根的解法，根据方程的系数及根判别式，分为如下几种情况：

① 当 $a = 0$ 时，方程不是二次方程；

② 当 $b^2 - 4ac = 0$ 时，有两个等实根 $x_1 = x_2 = -\dfrac{b}{2a}$

③ 当 $b^2 - 4ac > 0$ 时，有两个不同的实根：$x_1 = \dfrac{-b+\sqrt{b^2-4ac}}{2a}, x_2 = \dfrac{-b-\sqrt{b^2-4ac}}{2a}$

④ 当 $b^2 - 4ac < 0$ 时，无实根。

算法由流程图 7.4 描述，实现代码如下：

```
#include<stdio.h>
#include<math.h>
int main()
{
   int a,b,c ,disc; /*方程系数和根的判别式*/
   double x1,x2;

   printf("Input coefficients of the equation:\n");
   scanf("a=%d,b=%d,c=%d",&a,&b,&c);

   if(a==0)  /*非一元二次方程*/
      printf("Not a quadratic");
   else      /*一元二次方程*/
   {
      disc=b*b-4*a*c;
      if(disc==0) /*  有两等实根*/
         printf("Two equal roots:%8.4f\n",-b/(2*a));
```

```
        else
        {
          if(disc>0)   /* 有不等实根*/
          {
            x1=(-b+sqrt(disc))/(2*a);
            x2=(-b-sqrt(disc))/(2*a);
               printf("Distinct real roots:%8.4f and %8.4f\n",x1,x2);
            }
            else /* 无实根*/
                printf("No real roots\n");
           }
      }
      return 0;
  }
```

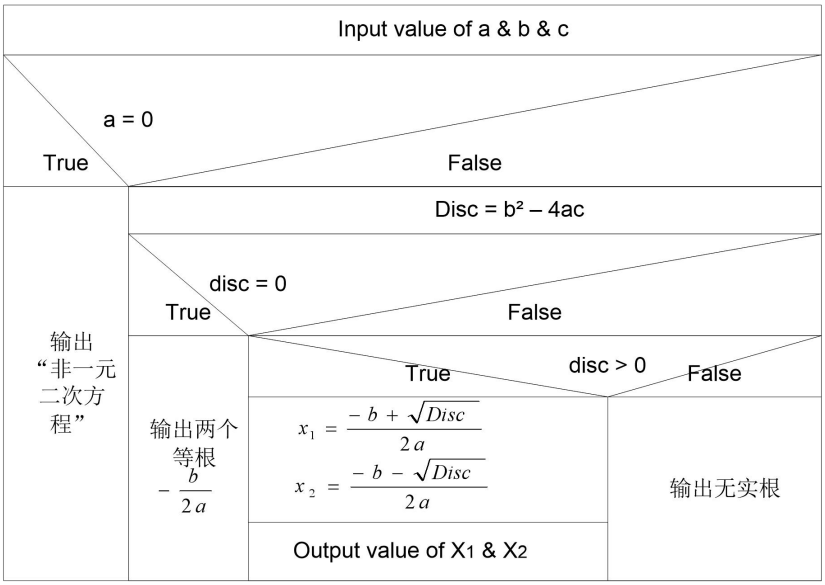

图 7.4 求一元二次方程根算法

if 语句中可能嵌套另一个 if-else 语句，就会出现多个 if 和多个 else 的情况，此时要特别注意 if 和 else 的配对问题。例如，有如下语句：

```
if(表达式 1)
  if(表达式 2)
     语句 1；
else
    语句 2；
```

其中的 else 究竟是与哪一个 if 配对呢?是应理解为如下形式：

```
if(表达式 1)
 {
   if(表达式 2)
        语句 1；
    else
```

```
            语句 2；
    }
```
还是应理解为如下形式：
```
if(表达式 1)
{
    if(表达式 2)
        语句 1；
}
else
    语句 2；
```

为了避免这种匹配的二义性，C 语言规定，else 总是与它最接近且未配对的 if 语句配对，因此对上述例子应按前一种情况理解。当嵌套结构较复杂时，为增强代码的可读性，应使用花括号和适当的缩进形式来明确各个语句的层次关系。

例 7.4 编一程序计算函数 y 的值，输入一个 x 值，输出 y 值。

$$y=\begin{cases}-1 & (x<0)\\ 0 & (x=0)\\ 1 & (x>0)\end{cases}$$

分析如下给出的算法，哪个是正确的？

程序 1：
```
if(x < 0)
   y = -1;
else
  if(x==0)
    y=0;
  else
     y=1;
```
程序 2：
```
if(x>=0)
  if(x>0)
     y=1;
  else
    y=0;
else
     y=-1;
```
程序 3:
```
y = -1;
if(x!=0)
   if(x>0)
    y=1;
else
  y=0;
```
程序 4:
```
y=0;
if(x>=0)
    if(x>0)
      y=1;
else
      y=-1;
```
细心分析程序，后两种方法为错误的，由于没有正确分析 if 和 else 的匹配关系，造成逻辑错误。

7.3.2 多分支 else if 语句

在 if 语句中可包含一个或多个 if 语句，嵌套的层数为任意层。当嵌套层数比较多时，代码的可读性下降。设计这种多分支的选择结构很容易产生混乱，因此建议嵌套层数不要超过 3 层。对于多项选择问题，常使用结构更为清晰对称的 else-if 形式的 if 语句，其语句的一般形式如下：

```
if ( 表达式 1 )
      语句 1
else if (表达式 2 )
      语句 2
…...
else if (表达式 n )
     语句 n
else
     语句 n+1
```

语句的执行过程为：先计算条件表达式 1 的值，如果为真执行语句 1; 否则判断表达式 2 的值，如果为真执行语 2；…，判断表达式 n 的值，如果为真执行语句 n, 否则执行语句 n+1。依次判断表达式的值，当出现某个值为真时，则执行其对应的语句，然后跳到整个 if 语句之外继续执行程序。如果所有的表达式均为假，则执行语句 n+1。然后继续执行后续程序。

这种语句形式适合实现多选一的操作，对一元二次方程求根的问题，要从四种情况中选择一种方式计算根值。上节的例程中使用 3 层嵌套的 if 语句，结构较为复杂。如果用 else-if 语句改写，则容易理清逻辑关系及层次。请对比两种形式的异同。

例 7.5 完善例 7.3 的程序，用 else-if 语句求一元二次程的解，并能计算方程的复数根。

```
#include<stdio.h>
#include<math.h>
int main()
{
   double a,b,c ,       /*方程系数*/
                 disc,       /*根的判别式*/
                x1,x2,   /*方程的实根*/
                realPart,imagPart; /*虚根的实部与虚部*/

   printf("输入方程系数:\n");
   scanf("a=%lf,b=%lf,c=%lf",&a,&b,&c);
   disc = b*b-4*a*c;

   if(fabs(a)<=1e-6)
      printf("非一元二次方程");
   else if(fabs(disc)<=1e-6)
     printf("有两相等实根:%8.4\n",-b/(2*a));
   else if(disc>0)
   {
      x1 = (-b+sqrt(disc))/(2*a);
      x2 = (-b-sqrt(disc))/(2*a);
      printf("有两不不等实根:%8.4f and %8.4f\n",x1,x2);
   }
    else
   {
       realPart = -b/(2*a);
      imagPart = sqrt(-disc)/(2*a);
```

```
        printf("有两虚根：\n");
        printf("%8.4f +%8.4fi\n",realPart,imagPart);
        printf("%8.4f -%8.4fi\n",realPart,imagPart);
    }
    return 0;
}
```

程序运行结果：

```
输入方程系数:
a=1,b=1,c=1
有两虚根：
 -0.5000 +    0.8660i
 -0.5000 -    0.8660i
```

设计该程序时注意如下问题：

（1） 在多分支选择结构中 if 与 else 常配对使用，else if 和 else 不能作为独立的语句单独使用。

（2） 各个分支的花括号中的执行语句块可以包含一条或多条语句，若只有一条语句时可以省略“{}”， 若有多条语句时省略“{}”会产生语法错误或者逻辑错误。

（3） 由于计算机不能精确表示所有实数，因此最好不使用运算符“==”直接判断两个实数的关系，而是求两者差的绝对值是否足够小来判断相等关系。例如，对于 double a 判断 a=0 表示为 fabs(a)<=1e-6。

7.3.3 switch 语句

人们常要根据今天是星期几来安排工作或则学习的内容和顺序。就如同作单选题，程序可以从一个数据集合中选择一个值，并与之作相应的操作。对于这种逻辑关系比较简单的多选一问题，可有用 if-elseif-else 语句实现，但最好用 switch 语句实现。

在某些情况下，程序需要根据整数变量或表达式的值，从一组动作中选择一个执行，此时用 swich 语句构成选择结构更为清晰简洁。swich 语句是多分支语句，常和关键字 case，default 及 break 配合使用，一般形式如下：

```
switch ( 表达式 )
{
 case  标号 1  :  语句 1;    break;
 case  标号 2  :  语句 2;    break;
   …
 case  标号 n  :  语句 n;    break;
 default  :  语句 n+1
}
```

该语句的执行过程为：先计算表达式的值，然后将其与每个 case 后的标号进行比较，当和某个标号相匹配时，就执行该 case 分支对应的语句，若所有标号值都不能与其匹配，则执行 default 分支后语句。swtich 的执行流程如图 7.5 所示。

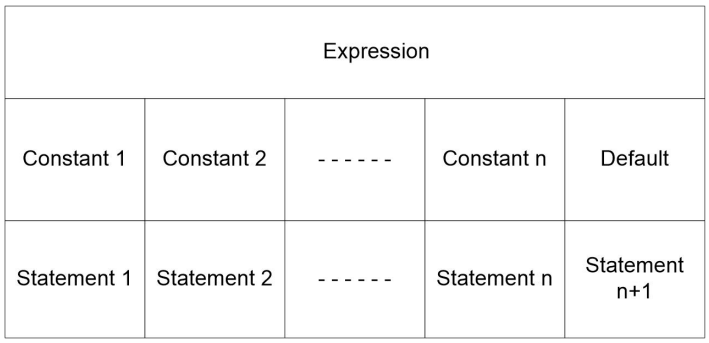

图 7.5 switch 语句执行流程

举一个简单的例子来说明 switch 的语法结构和执行特点。人们一般根据天气情况选择穿什么衣服，假设天气包括“晴”、 “阴”或 “雨”三种，如果晴天要穿 T 恤带遮阳帽，如果阴天则在 T 恤外加件薄外套，如果雨天则要穿风雨衣带雨伞。下面的代码中用到枚举类型，可以使用关键字 enum 来定义一种新的类型，并用标识常量表示该类型变量的值。例如，定义天气类型：

```
enum Weather { Sunny, Cloudy, Rainy};     /*枚举类型的天气*/
enum Weacher today = Cloudy;              /*枚举类型的变量 today，值为 1*/
```

today 为枚举类型 Weacher 的变量，其值只能为 Sunny、Cloudy 和 Rainy 中的一个，分别代表数值 0、1 和 2。使用枚举类型可用标识符表示整数值，使程序有较好的可读性。 用 switch 语句实现天气和穿衣关系：

```
switch (today)
{
    case 0:
        printf("T-shirt + cap\n");
        break;
    case 1:
        printf("T-shirt + outer wear\n");
        break;
    case 2:
        printf("Raincoat+umbrella\n");
        break;
    default:
        printf("whatever\n");
}
```

switch 的圆括号中为用于判断的表达式，这里为枚举类型的变量 today，其值为枚举类型常量 Cloudy，具有整型值 1。将 today 的值和 case 标号后的标号依次匹配，当发现和标号 2 相等，则执行 case 2 分支中的语句，应输出如下信息：

```
T-shirt + outer wear
```

break 是改变程序流程的关键字，其作用是跳出当前的选择结构，即忽略 switch 中的其他语句，转而执行 switch 语句后的程序。若没有 break 语句，则程序的输出结果为：

```
T-shirt + outer wear
T-shirt + raincoat
whatever
```

此时不是多选一的单选结构，而是多选多的选择结构。

在使用 switch 语句时，应注意以下几点：

（1） 判断表达式括号应具有整型值，一般为整型、字符型或枚举类型的变量或者表达式。case 后面的标号为常量表达式，其值必须是整型、字符型或枚举常量。

（2） 每个分支须保证唯一性，即 case 后的标号值必须互异，否则在与 switch 的表达式值进行匹配时出现歧异。

（3） 如果希望在执行完相应分支的语句后跳出 swich 结构，必须使用在各个分支中使用 break 语句。

（4） 各个分支的顺序不影响执行结果，并且多个 case 子句可公用同一操作语句。

例 7.6 输入一个字符，判断是否为元音字符。

```
#include <stdio.h>
int main()
{
   char c;
   printf("输入一个字符：");
   scanf("%c", &c);

   switch(c)
   {
      case 'a':   case 'o':   case 'e':
      case 'u':       case 'i':
         printf("小写元音字母\n");
         break;
      case 'A':      case 'O':   case 'E':
      case 'U':   case 'I':
         printf("大写元音字母\n");
         break;
      default:
         printf("其他字符\n");
   }
   return 0;
}
```

为了进一步区分输入的字符是元音还是辅音，将上面程序进行修改。

例 7.7 输入一个字符，判断是元音字符还是辅音字符。

设计和实现该程序有两个难点问题：

（1） 选择结构的嵌套

为了确定字符的种类，首先确定该字符是否为字母，然后判断为元音还是辅音。可以利用嵌套的选择结构来实现，即在 if-else 语句的 else 分支中嵌套 switch 语句。其实，实现选择结构的各种语句都可以相互嵌套。例如，在 switch 的分支中可以包含另一个 switch 语句，也可以包含 if-else 语句。

（2） 有关字符的函数

标准库中提供一些对字符进行操作的函数，在<ctype.h>头文件中进行声明。本程序利用其中的两个函数：isalpha()判断一个字符是否为字母，如果变量 ch 为字母则 isalpha(ch)的值为 1；函数 tolower()的功能是将变量 ch 转换为小写字符，从而简化 switch 中的 case

分支。测试字符函数的用法请参阅其他读物，这里不详细讨论。算法的流程如图 7.6 所示。

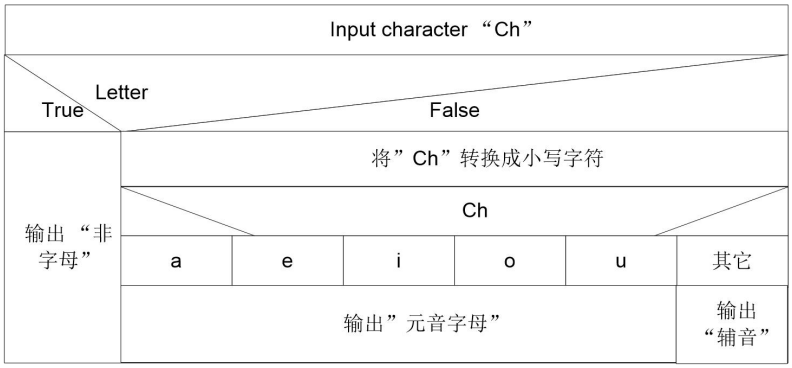

图 7.6 判断字母类型算法

```
#include <stdio.h>
#include <ctype.h>
int main()
{
   char ch;
    printf("输入一个字符\n");
    scanf("%c", &ch);

    if(!isalpha(ch))
       printf("非字母");
    else
    {
       switch(tolower(ch))
       {
          case 'a':     case 'o':
          case 'e':     case 'u':
          case 'i':
             printf("元音字母\n");
             break;
          default:
             printf("辅音字符\n");
          }
     }
    return 0;
}
```

7.4 案例分析

如何使程序能够解决实际问题？这要求程序设计者在深入分析问题的基础上，找到明确的解决方案，并将其映射成计算机能理解的指令序列。除了选择正确的数据表示与操作方法，还必须安排好每个指令的顺序，才能使程序按照程序员的意图运行。就如司马光能想到如何解救掉小伙伴的方法，要将其准确的表述出来，并指挥大家如何有条不紊地行动，才能将掉进水缸中的同伴及时安全地救出来。

程序由若干个语句构成，合理安排语句顺序是非常重要的。执行这些语句的时候可以一条一条按顺序执行，也可以有选择的执行，有时还需要反复执行一段代码。C 语言是一种结构化的程序语言，它的程序可由 3 中基本结构组成：顺序结构、选择结构和循环结构。所有 C 语言开发的程序，不管实现怎样复杂的功能，都是由这些基本结构按照不同的方式组合而成的。就如我们小时候玩的积木玩具，用有限种类的积木块可以堆砌成各种精妙的建筑或物品。本节通过计算个人所得税问题，讨论用顺序结构和选择结构组成算法，解决复杂的逻辑问题。

纳税是每个公民的责任和义务，从 2008 年 3 月 1 日起，规定我国公民个人所得税起征点为 2000 元。个人的工资薪金扣除福利费用（住房公积金、基本养老保险金、医疗保险金和失业保险金等）后，不超过 2000 元无需纳税，否则，对于超过 2000 元的部分，要按不同的税率交纳税金。个人工薪每月所得税计算公式如下：

应纳税额＝应纳税所得额*税率－速算扣除数

应纳税所得额＝应发工资－福利费用－2000

适用九级超额累进税率（5%至 45%）计缴个人所得税，相关数据见表 7.1。

例如，大毛当月应得工资收入为 9403 元，每月个人承担的福利费用共计 1000 元，则大毛当月应纳税所得额为 9403-1000-2000=6403 元。应纳个人所得税税额为 6403×20%-375＝905.60 元。

表 7.1 工资薪金所得适用九级超额累进税率表

级数	含税级距	税率（%）	速算扣除数
1	不超过 500 元的	5	0
2	超过 500 元至 2000 元的部分	10	25
3	超过 2000 元至 5000 元的部分	15	125
4	超过 5000 元至 20000 元的部分	20	375
5	超过 20000 元至 40000 元的部分	25	1375
6	超过 40000 元至 60000 元的部分	30	3375
7	超过 60000 元至 80000 元的部分	35	6375
8	超过 80000 元至 100000 元的部分	40	10375
9	超过 100000 元的部分	45	15375

问题描述

编程计算当月应交纳的个人工资薪金所得税，要求输入当月工薪收入和福利费用值，输出个人所得税金额。为了简化程序的分支，这里假设每月工薪不超过 20000 元。

算法分析

这是多选一的逻辑问题，将需纳税的收入划分为如下几个区间，按如下方式确定相应的税率和速算扣除数(简称速算数)：

需纳税的收入<0　　税率＝0　　速算数＝0

需纳税的收入∈[0,500]　　税率＝5%　速算数＝0

需纳税的收入∈(500,2000]　　税率＝10%　速算数＝25

需纳税的收入∈(2000,5000]　　税率＝15%　速算数＝125

需纳税的收入∈(5000,20000]　税率＝20%　速算数＝375

数据需求

问题常量：THREDHOLD 2000　个人所得税的起征点

问题输入：double salary　当月收入

　　doube welfare　福利费用

问题输出：double tax　　应缴税金

中间变量：double deduct　速算扣除数

　　double income　应纳税所得额

算法设计

（1）　输入变量 salary 和 welfare 的值；

　　if(salary<0 || salary >20000)　输出“错误输入”，结束程序。

（2）　计算 income = salary - welfare – 2000；

（3）　根据 income 的值计算 rate 和 decudt

　　if(income≤0)　rate=0; deduct = 0;

　　if（0<income≤500）rate=0.05; deduct = 0;

　　if（500<income≤2000）rate=0.1; deduct = 25

　　if（2000<income≤5000）rate=0.15; deduct = 125

　　if（5000<income≤20000）rate=0.2; deduct = 375

（4）　计算 tax = income *rate – deduct；

（5）　输出税金 tax。

其中关键步骤流程图如图 7.7 所示。

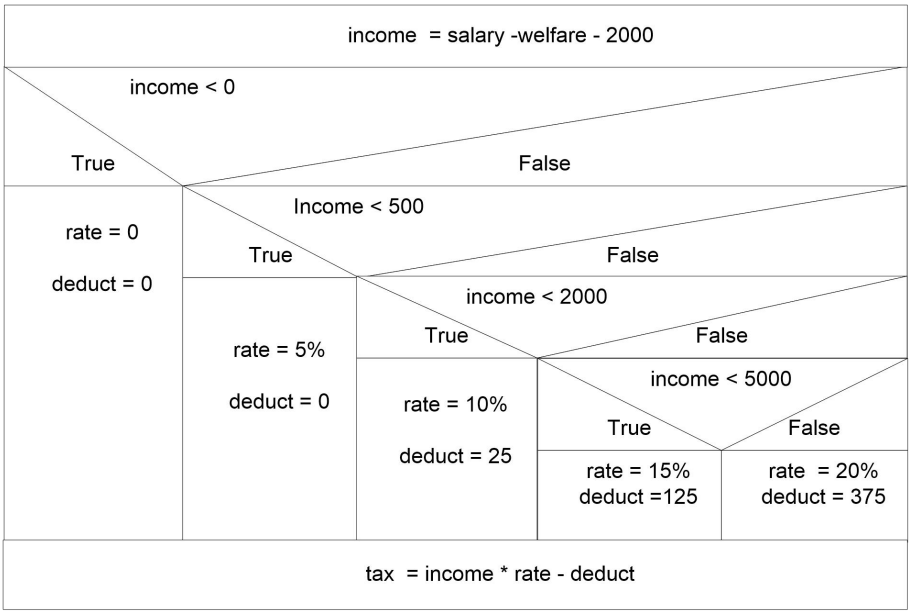

图 7.7　个人所得税算法

多分支选择结构可以 if 语句构造，包括多个简单的 if 语句、嵌套的 if 语句以及 else-if 语句等形式。如果选用简单的 if 语句，实现该问题的关键代码为：

```
if( income≤0) { rate=0; deduct = 0;}
if（0<income&& income≤500）　{rate=0.05; deduct = 0;}
if（500<income&& income≤2000）　{rate=0.1; deduct = 25}
if（2000<income&& income≤5000）{rate=0.15; deduct = 125;}
```

if（5000<income&& income≤20000　{rate=0.2; deduct = 375;}

这种方式的优点是结构对称清晰，但是效率不高，要顺序执行所有 if 语句，计算 5 个逻辑表达式的值。

如果采用嵌套的 if 语句，由于分支较多，嵌套层次较深，则代码的可读性不好。而采用 if-else if-else 的语句实现，不仅结构对称清晰，可读性好，而且执行效率高，因此选择这种方式实现算法。

在实际的应用系统中，需要对数据的有效性进行判断。本例中假设本月收入值的范围为 0 元≤收入≤20000 元，如果用户输入的数值不在此范围内，则需要输出提示信息，无需再进行下面的操作，使程序直接结束。

代码实现

```
#include <stdio.h>
#define THREDHOLD 2000 /*税金起征点*/
int main()
{
    double salary, welfare, /*收入和费用*/
           tax, rate,       /*税金和税率*/
           income,          /*  应纳税所得额*/
           deduct;          /*速算扣除数*/
     printf("输入本月收入(0<收入<20000 元)：");
     scanf("%lf",&salary);
     if(salary<0 || salary>20000) /*数据有效性判断*/
     {
       printf("输入错误！");
       return 1;          /*程序结束*/
     }
     printf("输入应扣除费用：");
     scanf("%lf",&welfare);

     /*计算 rate 和 deduct*/
     income = salary - welfare - THREDHOLD;
     if(income<=0)
     {
       rate = 0;
       deduct = 0;
     }
     else if(income <=500)
     {
        rate = 0.05;
       deduct = 0;
     }
     else if(income <=2000)
     {
          rate = 0.10;
       deduct = 25;
```

```
    }
    else if(income <=5000)
    {
          rate = 0.15;
       deduct = 125;
    }
    else
    {
        rate = 0.20;
       deduct = 375;
    }
        /*计算并输出税金*/
    tax = income *rate - deduct;
    printf("本月应交个人所得税为%.2f\n",tax);
    return 0;
}
```

简单测试

程序源码完成后，需要构造多组数据调试程序，企图发现存在的错误和漏洞。测试数据的选取原则，必须将所有的逻辑分支测试到，即多次运行程序后所有代码都要被执行过。如果运行结果或执行的顺序与预期不符合，则需要重新设计算法或者修改程序代码，直到排除所有的 Bug。

构造多组输入数据，检测各个分支的执行情况和输出的结果，列举 4 个测试用例，输出结果如下：

- 测试用例 1：

输入本月收入(0<收入<20000 元)：9403
输入应扣除费用：1000
本月应交个人所得税为 905.60

- 测试用例 2：

输入本月收入(0<收入<20000 元)：2500
输入应扣除费用：256
本月应交个人所得税为 12.20

- 测试用例 3：

输入本月收入(0<收入<20000 元)：1500
输入应扣除费用：223
本月应交个人所得税为 0.00

- 测试用例 4：

输入本月收入(0<收入<20000 元)：-2
输入错误！

习题

1．从键盘输入三个整数，按从小到大的顺序输出。
2．判断某人是否属于肥胖体型，常使用”体指数”指标：

体指数 t=体重/((身高)*(身高))(单位：kg, m)

当 t<18 时偏瘦；当 18<=t<=25 时正常体重，当 25<t<27 时超重体重，而 t> 27 属于肥胖。编写程序，输入身高和体重，判断出你属于何种体型，输出判断结果。

3.学生考试成绩可用百分制和等级制两种表示方式，规定成绩大于或等于 85 分时等级为 A，在 70 分到 85 分之间等级为 B，在 60 到 70 分之间时等级为 C，在 60 以下为不及格，其等级为 D。编写程序实现百分制和等级制的成绩转换。

(1) 输入成绩等级，输出相应百分制的分数段。

(2) 输入百分制的分数，输出相应成绩等级。

4． 编写程序计算货物运费，要求分别 3 中选择语句实现该程序。设货物运费每吨单价 p（美元）与运输距离 s（公里）之间有如下关系：

$$p = \begin{cases} 30 & s < 100 \\ 27.5 & 100 \le s < 200 \\ 25 & 200 \le s < 300 \\ 22.5 & 300 \le s < 400 \\ 20 & s \ge 400 \end{cases}$$

输入要托运的货物重量为 w 吨，托运距离 s 公里，计算并输出总运费 $t = p * w * s$。

5．字符型数据可以简单分为数字、大写字母、小写字母及其他字符四类。从键盘输入一个字符，输出它的类型。

6．设计一个程序，推断三角形的类别和面积。输入三角形的三条边的边长，如果三条边能构成三角形，则输出三角形的面积及种类。(注种：种类包括一般三角形、直接三角形或等边三角形)。

7．设计简单的计算器的程序。要求根据用户输入的表达式：

操作数 1　运算符 op　操作数 2

指定的算术运算符为: +、-、*和/。

(1) 如果操作数为整数，计算并输出计算结果。

(2) 如果希望程序能进行浮点数运算，如何修改程序?

(3) 如果要求连续作多次运算，程序该如何修改?

8．编写程序判断日期是否有效。用户输入日期数据（年、月和日），输出相应判断结果。提示：需要分别判断年月日的有效性，可以假设年 year 为正整数，月 month 为 1～12 的整数，而日 day 为 0～maxDay 的整数，其中 maxDay 的值取决与该日期所在的年份（是否为闰年）和月份（是大月、小月还是 2 月）。

9．设计简单的日期计算器，分别输入年、月和日，输出该日期的前一天和后一天。

10．设计复杂的日期计算器。要求根据用户输入一个日期，实现如下功能：

（1）判断日期是否有效；

（2）输出 n 天后的日期；（n 为用户输入，可正可负）；

（3）输出当天是星期几。

11．输入一个小于 1 亿的正数，将其转换为大写数字。