计算机高级编程技术

第二章　Pascal语言基础

第2章　Pascal语言基础

- Pascal语言具有清晰明了的模块结构，丰富的数据类型和语句，运行效率高且便于移植，因此广泛地应用于软件的开发。
  - 2.1 数据类型
  - 2.2 常量与变量
  - 2.3 运算符和表达式
  - 2.4 基本程序设计
  - 2.5 过程与函数
  - 2.6 常用内部函数

2.1 数据类型

- Object Pascal不仅包括一些标准的数据类型，还包括用户自己定义一些较为复杂的数据类型。
  - 2.1.1标准数据类型
  - 2.1.2子界类型
  - 2.1.3枚举类型
  - 2.1.4集合类型
  - 2.1.5指针类型
  - 2.1.6数组类型
  - 2.1.7记录类型
  - 2.1.8文件类型

2.1.1 标准数据类型

- Object Pascal的数据类型包括一些简单数据类型，例如：整型、实型、字符型、字符串型和布尔型。下表列出了标准的数据类型。

类别	类型	大小（byte）	范围
整型（Integer）	Integer	2/4	-32768~32767/-2147483648~2147483647
	Cardinal	2/4	0~ 65535/0~2147483647
	ShortInt	1	-128~127
	SmallInt	2	-32768~32767
	LongInt	4	-2147483648~2147483647
	Byte	1	0~255
	Word	2	0~65535

2.1.1 标准数据类型

类别	类型	大小（byte）	范围
实型（Real）	Real	6	2.9E-39~1.7E38，-2.9E-39~-1.7E38
	Single	4	1.5E-45~3.4E38，-1.5E-45~-3.4E38
	Double	8	5.0E-324~1.7E308，-5.0E-324~-1.7E308
	Extended	10	3.4E-4932~1.1E4932，-3.4E-4932~-1.1E4932
	Comp	8	-9.2E18~9.2E18
	Currency	8	-922337203685477.5808~922337203685477.5807
布尔型（Boolean）	Boolean	1	True/False
	ByteBool	1	True/False
	WordBool	2	True/False
	LongBool	4	True/False
字符型（Char）	Char	1	ASCII码
	String	0~255	ASCII码

2.1.2 子界类型

- 子界类型是Pascal允许用户定义的一个结构数据类型。如果用户预先知道一个变量的范围，就可以通过定义子界类型和子界类型变量来实现由系统自动检查变量是否超出了允许的范围。
- 子界类型的定义：

  Type 〈类型名称〉=〈常量1〉..〈常量2〉。

2.1.2 子界类型

- 例如以下代码：

```
type
  TMonth=1..12;
  TScore='A'..'F';
var
  Month:TMonth;
  Score:TScore;
```

- 如上定义了子界类型变量Month范围从1～12，Score范围从'A'～'F'。

2.1.3 枚举类型

- 枚举类型的定义：

  Type〈类型名称〉=（〈标识符1〉），〈标识符2〉,...,〈标识符n〉）。

- 它是通过列举出该数据所有的可能值来定义的，如下例：

```
type
  TWeekDay=(Sun, Mon, Tue, Wed, Thu, Fri, Sat);
var
  WeekDay:TweekDay;
```

- 每个枚举值只能出现在一个枚举类型的定义当中且只能出现一次。

2.1.3 枚举类型

```
type
  TWeekDay=(Sun, Mon, Tue, Wed, Thu, Fri, Sat);
var
  WeekDay:TweekDay;
```

- 需要注意的是，每个枚举值只能出现在一个枚举类型的定义当中，而且在每个枚举类型的定义当中只能出现一次。

2.1.4 集合类型

- 集合类型是Pascal允许用户定义的一个结构数据类型。它是一些同类型元素的集合，它用来检验某个元素是否被包含在一个集合当中。
- 下例用来说明集合类型数据的定义方法和运用：

```
type
  TWeekDay=Set of 1..7;
var
  WeekDay:TWeekDay;
begin
  WeekDay:=[1,3,5,6];
end;
```

- 注意Pascal编辑程序限制了集合数目，所以整型数据不能作为集合类型的基类型。而且基类型必须是有序类型。

2.1.5 指针类型

- Pascal允许建立动态数据结构，它采用指针类型数据，一个指针类型变量用于保存一个内存地址。
- 例如：

```
Type
  Pbirthday=^TBirthday;
  TBirthday=Record
  Name:String[30];
  Year:Integer;
```

2.1.5 指针类型

```
  Month:1..12;
  Date:1..31;
end;
var
  Pbirthday1:Pbirthday;
  Pint:^Integer;
begin
  Pbirthday1^.Name:='Smith';
  Pint^:=18;
end;
```

2.1.6 数组类型

- 数组类型数据是一种用户定义的结构数据类型，它是一些具有相同数据类型的元素的集合。
- 例如：

```
type
  TOneArray=Array[3..39] of Integer;
  TdoubleArray=Array[125,1..50] of Real;
Var
  OneArray:TOneArray;
  doubleArray:TdoubleArray;
```

2.1.7 记录类型

- 记录类型包括了一组元素，数据可以包括相同的数据类型，也可以包括不同的数据类型，数据可以是数值型的，也可以是非数值型的。
- 例如，一个学生的记录类型如下：

```
type
  TStudent=Record
  Name:String[30];
  Age:Integer;
  StudNum:Integer;
```

2.1.7 记录类型

```
  Score:real;
end;
var
  Student1:TStudent;
begin
  Student1.Name:='Smith';
  Student1.Age:=20;
  Student1.StudNum:=950007;
  Student1.Score:=87.5;
end;
```

2.1.8 文件类型

- 文件类型是用来对文件进行操作的，它包括同一类元素的线性有序组合。定义文件类型实际上是定义了一个文件类型的指针。
- 例如：

```
type
  TStudent=Record
  Name:String[30];
  Age:Integer;
  StudNum:Integer;
  Score:real;
end;
  Fstudent=File of Tstudent;
  Fstr=File of String;
var
```

2.1.8 文件类型

```
Student1:FStudent;
  Str:Fstr;
begin
 Student1^.Name:='Smith';
  Student1^.Age:=20;
  Student1^.StudNum:=950007;
  Student1^.Score:=87.5;
  Str^='I love Delphi 7.0.'
end;
```

2.2 常量与变量

- 常量和变量是学习各种编程语言进行程序设计的基础概念。常量是指在程序运行过程中其值始终不变的量，变量则是程序运行过程中其值可以改变的量。
  - 2.2.1 常量
  - 2.2.2 变量

2.2.1 常量

- 在Pascal中用const关键字来声明常量，在Object Pascal语言中不需要在对常量赋值时声明常量的类型。
- 例如：

```
const
ADecimalNumber=3.14;
i=10;
ErrorString='danger,Danger,Danger!';
```

- 整型数是最小的Integer类型;字符串值是char类型或string类型;浮点值是extended数据类型;Integer和Char的集合类型被存储为它们自己。

2.2.2 变量

- Object Pascal要求在一个过程、函数或程序前的变量声明段中声明它们。Object Pascal对大小写不敏感，采用大小写只是为了可读性好。
- 当在Object Pascal中声明一个变量时，变量名在类型的前面，中间用冒号隔开，变量初始化通常要跟变量声明分开。

2.2.2 变量

- Delphi能在var块中对全局变量赋初值，这里有一些例子演示：

```
var
  i:Integer=10;
  S:String='Hello World';
  D:Double=3.141579;
```

- 能赋初值的变量仅是全局变量，而且Delphi编译器自动对全局变量赋初值。因此在源代码中不必对全局变量赋零初值。

2.3 运算符和表达式

- 运算符是在代码中对各种数据类型进行运算的符号。例如，有能进行加、减、乘、除的运算符，有能访问一个数组的某个单元地址的运算符。表达式由运算对象和运算符两部分组成。
  - 2.3.1 运算符
  - 2.3.2 表达式

2.3.1 运算符

- Object Pascal语言中的运算符有：

  @、not、^、*、/、div、mod、and、shl、shr、as、+、－、or、xor、=、>、<、<>、<=、>=、in和is等。

- 运算符分为单目运算符和双目运算符。上面的@、not和^为单目运算符，其余的为双目运算符。其中+和-也可以作为单目运算符来使用。
- 单目运算符一般放在操作对象的前面，只有^可以放在操作对象的后面。双目运算符都放在两个操作数之间。

2.3.1 运算符

- 有些运算符是根据给定的操作数的数据类型做相应处理的。例如not对于整型操作数来说是按位取反;对于逻辑类型操作数来说则逻辑取反。
- 除了运算符^、is和in外，其余操作符都可以对Variant类型的数据进行操作。
  - 1.算术运算符
  - 2. 逻辑运算符
  - 3. 位运算符
  - 4. 字符串运算符
  - 5. 指针运算符
  - 6. 类运算符
  - 7. @运算符
  - 8. 集合运算符
  - 9. 关系运算符

2.3.1 运算符

1. 算术运算符

- 算术运算符对浮点数和整数进行加、减、乘、除和取模运算。

运算符	作用	操作数类型	结果类型	举例
+	两个数相加	Integer，real	Integer，real	X+Y
-	两个数相减	Integer，real	Integer，real	Result-1
*	两个数相乘	Integer，real	Integer，real	P*InterestRate
/	两个浮点数相除	Integer，real	Integer，real	X/2
Div	两个整型数相除	Integer	Integer	Total div UnitSize
Mod	取模	Integer	Integer	Z mod 8

2.3.1 运算符

- 2. 逻辑运算符
  - Pascal语言用and和or作为逻辑与和逻辑或运算符，Pascal的逻辑非的运算符是not，它是用来对一个布尔表达式取反。
  - 逻辑运算符对逻辑类型数进行运算，数据结果也为逻辑类型。

运算符	作用	举例
Not	逻辑取反运算	not (C in MySet)
And	逻辑和运算	Done and (Total>0)
Or	逻辑或运算	A or B
Xor	逻辑异或运算	A xor B

2.3.1 运算符

- 3. 位运算符
  - 位运算符对整型的数据进行按位操作，所得结果也为整型数据，如表所示。

运算符	作用	举例
Not	按位取反	not X
And	按位和	X and Y
Or	按位或	X or Y
Xor	按位异或	X xor Y
Shl	按位左移	X shl 2
Shr	按位右移	Y shr 2

2.3.1 运算符

- 4. 字符串运算符
  - 加号（+）还可以作为字符串运算符，它可以将两个字符串联接在一起。如果两个字符串都是短字符串，那么在结果字符串长度超过255个字符的情况下，只有前面255个字符有效。

2.3.1 运算符

5. 指针运算符

- 表中的运算符可以对指针类型的数据进行操作。

运算符	作用	操作数类型	结果类型	举例
+	将指针指向的地址增加一个偏移量	指针integer	指针	P+I
-	将指针指向的地址减去一个偏移量	指针integer	指针	integerP-Q
^	取指针指向地址中的内容	指针	指针指向的数据类型	P^
=	判断两个指针是否指向同一个地址	指针	Boolean	P=Q
<>	判断两个指针是否指向的是不同的地址	指针	Boolean	P<>Q

2.3.1 运算符

- 6. 类运算符
  - 类运算符as和is对类或类的实例进行操作。此外，关系运算符=和<>也可以对类进行操作。
- 7. @运算符
  - @运算符返回一个变量、过程或函数的地址。

2.3.1 运算符

- 8. 集合运算符
  - 集合运算符主要对两个集合进行操作，判断两个集合之间的关系，如表所示。

运算符	作用	操作数类型	结果类型	举例
+	取两个集合的元素	set	set	Set1+Set2
-	取两个集合不同的元素	set	set	S-T
*	取两个集合共同的元素	set	set	S*T
<=	判断左边的集合是否是右边集合的子集	set	Boolean	Q<=MySet
>=	判断左边的集合是否是右边集合的母集	set	Boolean	S1>=S2
=	判断两个集合是否相等	set	Boolean	S2=MySet
<>	判断两个集合是否不等	set	Boolean	MySet<>S1
In	判断左边的集合是否与右边的集合有从属关系有序	set	Boolean	A in Set1

2.3.1 运算符

- 9. 关系运算符
  - 关系运算符可对两个普通数据类型、类、对象、接口类型或字符串类型的数据进行比较，结果数据类型为布尔类型，如表2-7所示。

运算符	作用	举例	运算符
=	判断是否相等	I=Max	=
<>	判断是否不相等	X<>Y	<>
<	判断是否小于	X<Y	<
>	判断是否大于	Len>0	>
<=	判断是否小于等于	Cnt<=I	<=
>=	判断是否大于等于	I>=1	>=

2.3.2 表达式

- 一个表达式由运算对象和运算符两部分组成。运算符可以分为算术运算符、逻辑运算符、串运算符、字符指针运算符、集合运算符、关系运算符以及@运算符。
- 运算符的优先顺序如表所示。

运算符	优先顺序	类别	运算符
^	1	域、指针引用	^
@ not	2	取非	@ not
* / div mod shl shr and as	3	乘除法与类型转换	* / div mod shl shr and as
＋ - or xor	4	加减法	＋ - or xor
= <> > < <= >= in is	5	比较操作	= <> > < <= >= in is

2.4 基本程序设计

- 基本程序设计主要包括：程序语句、基本控件、顺序结构、选择结构、循环结构。
  - 2.4.1 程序语句
  - 2.4.2 基本控件
  - 2.4.3 顺序结构
  - 2.4.4 选择结构
  - 2.4.5 循环结构

2.4.1 程序语句

- 1. 赋值语句
  - 实现功能：为变量赋值。
  - 语法形式：<变量>:=<表达式>;
  - 实际举例：

```
x:=y*z;
b:=(x>=1) or (y>=4) and (z<>0);
t:=sqrt(m)+4*sin(x);
```

2.4.1 程序语句

- 2. GOTO语句
  - 实现功能：改变程序流程至标号语句处。
  - 语法形式：goto〈标号〉;
  - 实际举例：goto 100;
- 3. IF语句
  - 实现功能：作条件判断控制流程。

2.4.1 程序语句

- (1) 单分支语句

语法形式：

if 〈布尔型表达式〉 then 〈语句〉;

实际举例：

```
if x<=100 then y:=x;
y:=100;
begin
〈语句组〉
end;
```

2.4.1 程序语句

- (2) 双分支语句

  语法形式：

  if 〈布尔型表达式〉 then

  begin

  〈语句组〉

  end

  else

2.4.1 程序语句

- (3) 多层嵌套语句

语法形式：

```
if <布尔型表达式> then
begin
  <语句组>
end
else if <布尔型表达式> then
begin
  <语句组>
end;
…
else
begin
  <语句组>
end;
```

2.4.1 程序语句

- 4. CASE语句
  - 多层嵌套的条件语句可读性差，实际编程中常采用case语句。
  - 实现功能：实现多条件选择。
  - 语法形式：

```
Case <表达式> of
数值1：
begin
  <语句组>
end;
数值2：
begin
  <语句组>
end;
…
```

2.4.1 程序语句

- 5. REPEAT语句
  - 实现功能：循环。
  - 语法形式：

```
repeat
  <语句组>;
until <表达式>;
```

2.4.1 程序语句

- 6. WHILE语句
  - 实现功能：循环。
  - 语法形式：

```
while <表达式> do
begin
  <语句组>
end;
end;
```

2.4.1 程序语句

- 7. FOR语句
  - 实现功能：循环。
  - 语法形式：

```
for <变量>=<表达式1> to <表达式2> do
begin
  <语句组>
end;
```

2.4.1 程序语句

- 8. WITH语句
  - 实现功能：引用一个域或方法。
  - 语法形式：

```
with <记录名> do
begin
  <语句组>
```

【例2-1】

- 输入长方体的长和宽，然后计算并输出长方体的周长和面积。
  - 分析：设长方体的长和宽分别为a和b，长方体的周长和面积分别为c和s，结合题义可求出计算公式：c=2(a+b)，s=ab。
- 设计步骤：
  - ① 建立应用程序的用户界面：选择“新建”工程，进入窗体设计器，增加四个标签Label1~Label4，四个编辑框Edit1~Edit4和一个按钮Button1。

【例2-1】

■ ② 设置对象属性：如表所示。

对象	属性	属性值	说明
Label1	Caption	输入长方体的长：	标签的内容
Label2	Caption	输入长方体的宽：	标签的内容
Label3	Caption	长方体的周长：	标签的内容
Label4	Caption	长方体的面积：	标签的内容
Edit1	Text		编辑框的内容
Edit2	Text		编辑框的内容
Edit3	Text		编辑框的内容
	ReadOnly	True	文本内容只读
Edit4	Text		编辑框的内容

【例2-1】

■ ③ 编写事件代码。

```
Procedure Tform1.Button1Click(Sender:TObject);
var a,b,c,s:Real;
begin
  a:=strtofloat(edit1.text);
  b:=strtofloat(edit2.text);
  c:=2*(a+b);
  s:=a*b;
  edit3.text:=floattostr(c);
  edit4.text:=floattostr(s);
end;
```

2.4.2 基本控件

- 组件是可视化编程的基础，用户可以通过修改组件属性、建立事件处理过程来决定组件的外观或作用。
- Delphi中有四种基本的组件类型供用户使用或创建：标准控件、自定义控件、图形控件和非可视组件。
- 组件（component）和控件（control）两个术语并不完全相同。在Delphi中，控件是可视的用户界面元素，控件总是组件。组件是一种对象，它不一定是控件。

2.4.3 顺序结构

- 顺序结构是程序中最常见的基本结构。在该结构中，各操作模块按照出现的先后顺序依次执行，它是任何程序的主体基本结构。
- 在顺序结构中，通常使用基本控件完成输入及输出操作，使用赋值语句等简单的操作语句组成顺序结构即可实现顺序结构程序的编写。

2.4.4 选择结构

- 选择结构是计算机科学用来描述自然界和社会生活中分支现象的重要手段。它根据所给定的条件成立（真）或否（假），而决定从实际可能的不同分支中执行某一分支的相应操作。
- 在Delphi中，实现选择结构的是If和Case这两种条件语句，条件语句的功能就是根据表达式的值有选择地执行一组语句。

2.4.4 选择结构

- 1. If语句
  - 在执行一段代码以前，If语句能让用户判断某个条件是否满足。
  - 若一条if语句中有多个条件，则要把这几个条件分别用括号括起来。
  - 在Pascal中的begin和end，就像是C和C++中的“{”和“}”，例如，下面的代码是当一个条件满足时要执行多条语句：

```
if x=6 then
begin
  DoSomething;
  DoSomethingElse;
  DoAnotherThing;
end;
```

2.4.4 选择结构

- 用if...else能组合多个条件：

```
if x=100 then
  SomeFunction
else if x=200 then
  SomeOtherFunction
else begin
  SomethingElse;
  Entirely
end;
```

2.4.4 选择结构

- 2. case语句
  - 在Pascal中的case语句用来在多个可能的情况中选择一个条件，而不再需要用一大堆if...else if...else if结构，代码如下：

```
case SomeIntegerVariable of
    101: DoSomething;
    202: begin
            DoSomething;
```

2.4.4 选择结构

```
            DoSomethingElse;
        end;
    303: DoAnotherthing;
    else DoTheDefault;
end;
```

- 注意case语句的选择因子必须是有序类型，而不能用非有序的类型如字符串作为选择因子。

2.4.5 循环结构

- 循环是一种能重复执行某一动作的语言结构。
- 1. for循环
  - for循环适合用在事先知道循环次数的情况下，下面的代码在一个循环中把控制变量加到另一个变量中，共重复10次：

```
var
  i,x:Integer;
begin
  x:=0;
  for i:=1 to 10 do
      inc(x,i);
end;
```

2.4.5 循环结构

- 2. while循环
  - while循环结构用在先判断某些条件是否为真，然后重复执行某一段代码的情况下。while的条件是在循环体执行前进行判断的。下面的例子演示了每次从文件中读一行并写到屏幕上：

```
Program FileIt;
{$APPTYPE CONSOLE}
Var
 f:=TextFile;
 S:=String;
```

2.4.5 循环结构

```
begin
  AssignFile(f,'foo.txt');
  Reset(f);
  while not EOF(f) do begin
  readln(f,S);
  writeln(S);
end;
  CloseFile(f);
end.
```

2.4.5 循环结构

- Pascal中的while循环基本上跟C中的while循环和Visual Basic中Do While循环一样。

3. repeat...until

- repeaat...until循环与while循环相似，但它在某个条件为真前一直执行给定的代码。因为条件测试在循环的结尾，所以循环体至少要执行一遍。

2.4.5 循环结构

- 例如，下面的代码不断地把一个计数器加1，直到它大于100为止：

```
var
  x:Integer;
begin
  x:=1;
  repeat
    inc(x);
  until x>100;
end;
var
  i:Integer;
begin
  for i:=1to 100000 do
  begin
    MessageBeep(0);
if i=5 then break;
  end;
end;
```

2.4.5 循环结构

- 4. Break()过程
  - 在while、for或repeat循环中调用Break()，使得程序的执行流程立即跳到循环的结尾。下面的代码演示了5次循环后跳出循环。

```
var
  i:Integer;
begin
  for i:=1to 100000 do
  begin
   MessageBeep(0);
if i=5 then break;
  end;
end;
```

2.4.5 循环结构

- 5. Continue()过程
  - 如果想跳过循环中部分代码重新开始下一次循环，就调用continue()过程。下面的例子在执行第一次循环时continue()后的代码不执行：

```
var
  i:Integer;
begin
  for i:=1 to 3 do
begin
  writeln(i,'.Before Continue');
  if i=1 then continue;
  writeln(i,'.After continue');
end;
end;
```

【例2-2】

电脑设定一个固定的值（1-100），玩家输入一个随机的数字，如果输入的数字比设定的数字大，提示您输入的数字太大，请输入一个小点的数字；如果输入的数字比设定的数字小，提示您输入的数字太小，请输入一个大点的数字，直到输入正确的数字为止。以玩家猜数字的次数判定输赢。

【例2-2】

- 1、建立界面

【例2-2】

- 2、关键函数

  Random

  function Random [(Range: Integer)];：产生一个0<=X<Range的随机数。

- 3、变量定义

```
var
 Form1: TForm1;
 InNumber,Tnumber:integer;
```

- 4、主要代码

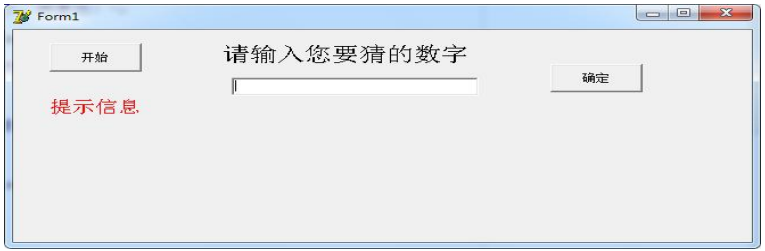
```
Tnumber:=strtoint(Edit1.Text);
if Tnumber=InNumber then
```

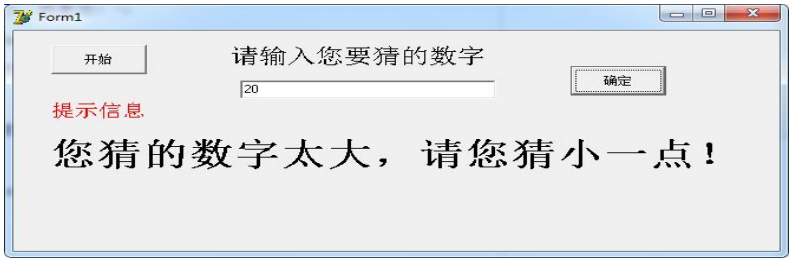
【例2-2】

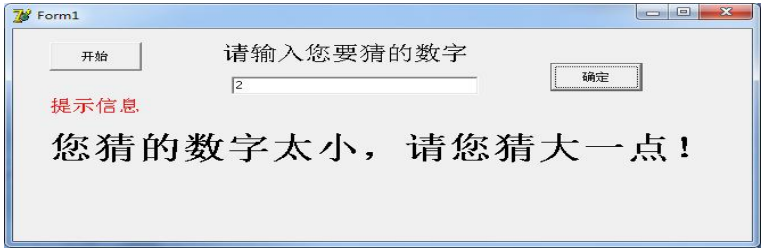
```
begin
   label2.Caption:='您猜对了，恭喜您！';
 end
 else if Tnumber>InNumber then
 begin
   label2.Caption:='您猜的数字太大，请您猜小一点！';
 end
   else
 begin
   label2.Caption:='您猜的数字太小，请您猜大一点！';
 end;
```

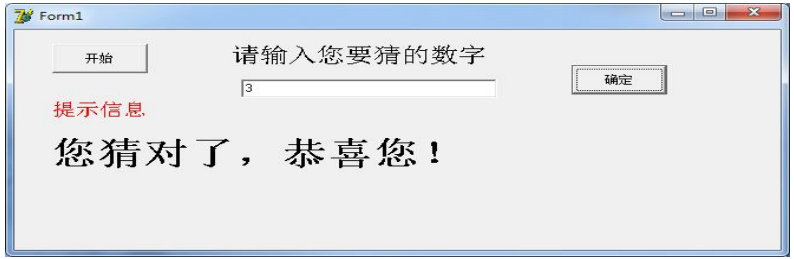
【例2-2】

□ 5、运行界面


Form1
开始
请输入您要猜的数字
确定
提示信息


Form1
开始
请输入您要猜的数字
20
确定
提示信息
您猜的数字太大，请您猜小一点！


Form1
开始
请输入您要猜的数字
2
确定
提示信息
您猜的数字太小，请您猜大一点！


Form1
开始
请输入您要猜的数字
3
确定
提示信息
您猜对了，恭喜您！

2.5 过程与函数

- 过程与函数是实现一定功能的语句块。
- 过程与函数的区别在于：过程没有返回值，而函数有返回值。
  - 2.5.1 过程
  - 2.5.2 函数
  - 2.5.3 参数
  - 2.5.4 子程序的嵌套与递归
  - 2.5.5 变量的作用域
  - 2.5.6 Delphi的程序结构

2.5.1 过程

- 过程的定义包括过程原型、过程体的定义。
- 过程定义的形式如下：

```
procedure procedureName(parameterList);directives;
local Declarations;
begin
statements
end;
```

2.5.1 过程

- 其中procedureName是过程名，为一个有效的标识符。
- parameterList为参数列表，需要指明参数的个数及其数据类型。
- directives是关于函数的指令字，可以一次设置多个，用分号隔开。
- 在begin与end之间是在函数调用时实现特定功能的一系列语句。
- 上面的parameterList、directives、localDeclarations和statements等为可选部分。

2.5.2 函数

- 函数的定义与过程非常类似，只是使用的保留字不同，而且多了一个返回值类型。
- 具体形式如下：

```
function functionName(parameterList):returnType;directives;
localDeclarations;
begin
 statements
end;
```

2.5.2 函数

- 可以将函数要返回的数值赋值给Result。
- 如果函数体中存在着一些由于判断而产生的分支语句时，要在每一个分支中设置返回值。通常要根据函数的返回值来确定下一步的操作。

2.5.3 参数

- 一般将函数定义时参数列表中的参数称为形参，将函数调用时参数列表中的参数称为实参。
- 一般来说，形参列表和实参列表完全匹配是指参数的个数一样，而且顺序排列的数据类型也完全一致。
- 可以为过程和函数的参数指定默认数值。指定默认数值的参数要放在参数列表的后部，没有指定默认数值的参数放在参数列表的前部。

【例2-3】

- 定义一个函数ShowNum，将一个浮点数按指定的精度输出在屏幕上。

```
program Project1;
{$APPTYPE CONSOLE}
uses Sysutils;  // 为了使用函数Format
//以一定精度显示一个浮点数
function ShowNum(Num:Double;Precision:Integer=4):Boolean;
var
MesStr:string;  // 浮点数显示输出的内容
```

【例2-3】

```
begin
if Preci
begin
Result:=False;
Exit;                    // 退出显示函数
end
else
begin
// 设置显示的格式
```

【例2-3】

```
MesStr:=Format('%*.*f',[10,Precision,Num]);
Result:=True;
end;
Writeln(MesStr);      // 显示数据
end;
sion<=-1 then         // 小数点后的位数要大于或等于零
begin
ShowNum(123.456789);          // Precision默认为4
ShowNum(123,5);   // 参数对数据类型进行升级
```

【例2-3】

```
// 下面一句代码不正确,故屏蔽掉
// ShowNum(123.456789,9.13);// 参数对数据类型不能降级
ShowNum(22 div 7,5);              // 调用函数
// 可以根据函数的返回值确定下一步的操作
if ShowNum(123.456789,-3)=False then
Writeln('数据格式设置错误,输出失败。');
Writeln('按下回车键<Enter>退出。');
Readln;
end.
```

【例2-3】

- 运行结果如下：

  123.4568

  123.00000

  3.14286

  数据格式设置错误，输出失败。

  按〈Enter〉键退出。

- 说明：

【例2-3】

- ① 为使用函数Format，需要在uses语句中将Sysutils单元包含进去。
- ② 由于小数点后的位数不可以设置为负数，所以当出现负数的时候，ShowNum函数返回False，并调用Exit函数立刻退出。
- ③ 在语句ShowNum(123,5);中，首先将整型常数123转换为浮点型常数，然后进行参数传递。
- ④ 在语句ShowNum(22 div 7,5);中，22 div 7的结果为3，然后再转换为浮点型常数进行参数传递。

2.5.4 子程序的嵌套与递归

- 在一个子程序（过程或函数）中包含另外一个子程序（过程或函数）的调用，称为子程序的嵌套。
- 子程序的递归调用指一个过程直接或间接调用自己本身，子程序直接调用自身称为直接递归，子程序间接调用自己称为间接递归。
- 递归调用在处理阶乘运算、级数运算、幂指数运算方面特别有效。

2.5.4 子程序的嵌套与递归

- 递归函数论是现代数学的一个重要分支，数学上常常采用递归的办法来定义一些概念。例如，自然数n的阶乘可以递归定义为：

$$n!=\begin{cases}1 & n=0\\ n\times(n-1)! & n>0\end{cases}$$

- 递归在算法描述中有着重要的地位，在间接递归调用中，子程序必须超前引用，即在子程序的首部后面加上保留字forward。

【例2-4】

- 编写程序打印菲波拉西（Fibonacci）数列。
  - 菲波拉西数列排列如下：　1 1 2 3 5 8 13 21 34 55 …
  - 分析：形成此数列的规律是，它的头两个数为1，从第三个数开始其值是它前面的两个数之和。即：

$$\text{fibo}=\begin{cases}1 & n=1\\ 1 & n=2\\ fibo(n-1)+fibo(n-2) & n>2\end{cases}$$

  - 这里只写出了菲波拉西函数的代码，事件代码用户可自己设计。

【例2-4】

```
Function fact(n:Real):Real;
begin
  if n=1 Or n=2 Then
    fibo:=1;
  else
    fibo:=fibo(n-1)+fibo(n-2);
end;
```

- 利用递归算法能简单有效地解决一些特殊问题，但是由于递归调用过程比较繁琐，所以执行效率很低，在选择递归时要慎重。

2.5.5 变量的作用域

- 作用域是指一个过程、函数和变量能被编译器识别的范围，全局常量的作用域是整个程序，而过程中的局部变量的作用域是那些过程。
- 作用域的演示的示例

```
Program Foo;
{$APPTYPE CONSOLE}
const
  SomeConstant=100;
var
  SomeGlobal:Integer;
  R:Real;
procedure SomeProc(var R:Real);
var  LocalReal:Real;
begin
  LocalReal:=10.0;
  R:=R-LocalReal;
end;
begin
  SomeGlobal:= SomeConstant;
  R:=4.593;
  SomeProc(R);
end;
```

2.5.5 变量的作用域

- SomeConstant、SomeGlobal和R是全局变量。过程SomeProc()有两个变量R和LocalReal，如果试图在SomeProc()过程外访问LocalReal，编译器将显示有未知识别符的错误。如果在SomeProc()中访问R，用的是局部变量的R，如果在这个过程外用R，则用的是全局变量的R。

2.5.6 Delphi的程序结构

- 一个Delphi应用程序对应一个工程，它由一个主程序与若干个单元组成，而程序段则是构成主程序和单元的基本结构要素。
- 1. 主程序
  - Delphi工程文件中的代码即为Delphi应用程序的主程序。主程序的结构为：

    program工程文件名;
    uses语句
    {$R*.RES}
    执行部分

2.5.6 Delphi的程序结构

- Delphi中几乎所有功能都在uses语句所引用的单元中。主程序由程序系统自动生成，程序员所编写的只是程序所引用的单元文件中的代码。
- 2. 单元与作用域
  - Delphi编程强调单元或模块的使用。在引入类之前，单元是模块化编程的基础，而类也是建立在单元概念基础上的。
  - 当创建工程时，或是向工程中添加一个新窗体时，Delphi都将自动添加一个新的单元文件，该单元为新窗体定义类与对象。

2.5.6 Delphi的程序结构

- 1.单元的结构
  - 单元文件的结构如下所示：

```
unit 单元名                    // 单元首部
interface                      // 接口部分
implementation                 // 实现部分
end                            // 单元结束
```

2.5.6 Delphi的程序结构

- 2.作用域
  - 在单元的接口部分声明的标识符在整个单元内有效。其他客户单元引用此单元时，则在该客户单元内有效。
  - 在实现部分隐含声明的标识符，不能在单元外使用。这些标识符在本单元的实现部分自声明处起至实现部分结束的任何程序段中有效。
  - 在实现部分的任何子程序段中声明的标识，则遵循程序段中标识符的作用域规则。

2.5.6 Delphi的程序结构

- 3. 程序段与作用域
  - 一个程序段是由声明部分和语句部分构成的结构，其一般结构为：

    声明部分

    begin

    语句

    End
  - 声明部分用来声明类型、常量、变量、函数，语句则是执行部分。
  - 函数体和过程体就是典型的程序段。

2.6 常用内部函数

- 为了尽可能地减少开发应用程序的难度和工作量，Delphi提供了一个内容十分丰富的程序库Run-Time Library（RTL），其中包括了大量的基本函数、过程、常量和变量定义。
  - 2.6.1 数学运算函数
  - 2.6.2 字符处理函数
  - 2.6.3 时间和日期函数
  - 2.6.4 数据类型转换函数
  - 2.6.5 格式输出函数