Delphi Advanced Programming Technology

Chapter 2

THE DELPHI PROGRAMMING LANGUAGE

Professor Zhaoyun Sun

2.1 Overview

Pascal ---- Delphi programming language

Keywords, Operators

Variables

Predefined data types

User-defined data types

Statements

Procedures and functions

2.2 Keywords

Keywords And Other Reserved Words In The Object Pascal Language

Keyword	Role
absolute	directive (variables)
abstract	directive (method)
and	operator (boolean)
array	type
as	operator (RTTI)
asm	statement
assembler	backward compatibility (asm)
at	statement (exceptions)
automated	access specifier (class)
begin	block marker
case	statement
cdecl	function calling convention
class	Type

Keywords

Keyword	Role (cont.)
const	declaration or directive (parameters)
constructor	special method
contains	operator (set)
default	directive (property)
destructor	special method
dispid	dispinterface specifier
dispinterface	type
div	operator
do	statement
downto	statement (for)
dynamic	directive (method)
else	statement (if or case)
end	block marker
except	statement (exceptions)
export	backward compatibility (class)
exports	declaration
external	directive (functions)

Keywords

Keyword	Role (cont.)
far	backward compatibility (class)
file	type
finalization	unit structure
finally	statement (exceptions)
for	statement
forward	function directive
function	declaration
goto	statement
if	statement
implementation	unit structure
implements	directive (property)
in	operator (set) - project structure
index	directive (dipinterface)
inherited	statement
initialization	unit structure
inline	backward compatibility (see asm)
interface	type

Keywords

Keywords And Other Reserved Words In The Object Pascal Language

Keyword	Role (cont.)
label	declaration
library	program structure
message	directive (method)
mod	operator (math)
name	directive (function)
near	backward compatibility (class)
nil	value
nodefault	directive (property)
not	operator (boolean)
object	backward compatibility (class)
of	statement (case)
on	statement (exceptions)
or	operator (boolean)
out	directive (parameters)
overload	function directive
override	function directive

Keywords

Keywords And Other Reserved Words In The Object Pascal Language

Keyword	Role (cont.)
package	program structure (package)
packed	directive (record)
pascal	function calling convention
private	access specifier (class)
procedure	declaration
program	program structure
property	declaration
protected	access specifier (class)
public	access specifier (class)
published	access specifier (class)
raise	statement (exceptions)
read	property specifier
readonly	dispatch interface specifier
record	type
register	function calling convention
reintroduce	function directive

Keywords

Keywords And Other Reserved Words In The Object Pascal Language

Keyword	Role (cont.)
safecall	function calling convention
set	type
shl	operator (math)
shr	operator (math)
stdcall	function calling convention
stored	directive (property)
string	type
then	statement (if)
threadvar	declaration
to	statement (for)
try	statement (exceptions)
type	declaration
unit	unit structure
until	statement
uses	unit structure
var	declaration

Keywords

Keywords are all the Object Pascal reserved identifiers, which have a role in the language.

Learn keywords when you use them.

Don’t use the keywords as names of your variables or functions !

2.3 Operators and Precedence

*Simple code example: if Z>=100 then X: = y1 + y2*a – y3/b;*

Pascal Language Operators, Grouped By Precedence

Unary Operators (Highest Precedence)	
@	Address of the variable or function (returns a pointer)
not	Boolean or bitwise not

Multiplicative and Bitwise Operators	
*	Arithmetic multiplication or set intersection
/	Floating-point division
div	Integer division
mod	Modulus (the remainder of integer division)
as	Allows a type-checked type conversion among at runtime
and	Boolean or bitwise and
shl	Bitwise left shift
shr	Bitwise right shift

Operators and Precedence

Pascal Language Operators, Grouped By Precedence

Additive Operators	
+	Arithmetic addition, set union, string concatenation, pointer offset addition
-	Arithmetic subtraction, set difference, pointer offset subtraction
or	Boolean or bitwise or
xor	Boolean or bitwise exclusive or

Relational and Comparison Operators (Lowest Precedence)	
=	Test whether equal
< >	Test whether not equal
<	Test whether less than
>	Test whether greater than
< =	Test whether less than or equal to, or a subset of a set
> =	Test whether greater than or equal to, or a superset of a set
in	Test whether the item is a member of the set
is	Test whether object is type-compatible (another RTTI operator)

Operators and Precedence

Contrary to most other programming languages, the ***and*** and ***or*** operators have precedence compared to the relational one.

```
a < b and c < d      // error

(a < b) and (c < d)
```

Operators and Precedence

Some operators have different meanings with different data types.

+ ***operator***

add two numbers
concatenate two strings, but not two characters
make the union of two sets

= operator

Test whether equal,
(: = Assignment)
add two numbers

Operators and Precedence

/ :

When dividing any two numbers (real or integers) with the / operator, the result is a real-number.

div:

When dividing two integers with div operator, result is an integer.

10/4 is 2.5 10 div 4 is 2

Operators and Precedence

Declare variables

Use var keyword

After the var keyword comes a list of variable names, followed by a colon and the name of the data type.

```
var
Value: Integer;
IsCorrect: Boolean;
A, B: Char;
```

2.4 Data Types, Variables, Constants

Variables

Pascal requires all variables to be declared before they are used. Every time you declare a variable, you must specify a data type.

The var keyword can be used

at the beginning of the code of a function or procedure
to declare variables local to the routine
inside a unit to declare global variables.

Variables

Once you have defined a variable of a given type, you can perform on it only the operations supported by its data type.

```
Value := 10;
IsCorrect := True;
```

```
Value := IsCorrect; // error
```

Constants

Constants have initial values that do not change during program execution . You can declare a constant with an initial value, and the data type is unnecessary.

```
declarations: const
Thousand = 1000;
Pi = 3.14;
AuthorName = ‘John H. Johnson';

Digits: array[0..5] of Char = ('0', '1', '2', '3', '4', '5');
```

Data Types

Predefined data types

Boolean
Integers
Real numbers
Characters, *Strings*
Date, time

Basic data types
- *Ordinal*
- *Real*
- *String*

User-defined data type

Array, etc.

Ordinal Types

Ordinal types are based on the concept of order or sequence.

Integers, Char, Boolean…

You can ask for the value following (next) or preceding (previous)

You can compare two values to see which is higher.

Ordinal Types

System Routines For Ordinal Types

Routine	Purpose
Dec	Decrements the variable passed as parameter, by one or by the value of the optional second parameter.
Inc	Increments the variable passed as parameter, by one or by the specified value.
Odd	Returns True if the argument is an odd number.
Pred	Returns the value before the argument in the order determined by the data type, the predecessor.
Succ	Returns the value after the argument, the successor.
Ord	Returns a number indicating the order of the argument within the set of values of the data type.
Low	Returns the lowest value in the range of the ordinal type passed as its parameter.
High	Returns the highest value in the range of the ordinal data type.

Ordinal Types

```
// Use ordinal functions
x:Byte=5;    // range from -127 to 128
y:=Pred(x);  w:=ord(x);  Inc(x);  Low(x);
```

```
// use Low and High functions
var
  A:array[15..37] of Integer;
  i:Integer;
begin
  for i:=Low(A) to High(A) do
     A[i]:=i;
end;
```

Boolean Types

Boolean values : *True* or *False*

Ord(False) is 0, Ord(true) is 1.

Boolean is the preferred type.

Boolean type expressions are used in may places.

e.g. If statement

```
// X, Y : integer;
if  X = 1  then  Y=X*2;
if  X  then ……;    // error
```

Character Types

ANSIChar / Char :

ANSI character, 8-bit (length 256)

WideChar :

Unicode characters, 16-bit (length 65535)

Symbolic vs. numeric notation:

‘A’ #65 ‘B’ #66 …
‘@’ #64 tabulator #9 newline #10 …

String:

”This is a string.”

Integer Types

Generic integer types for 32-bit implementations of Object Pascal

Type	Range	Format
Integer	-2147483648..2147483647	signed 32-bit
Cardinal	0.. 4,294,967,295	unsigned 32-bit

Fundamental integer types

Type	Range	Format
Shortint	-128..127	signed 8-bit
Smallint	-32768..32767	signed 16-bit
Longint	-2147483648..2147483647	signed 32-bit
Int64	$-2^{63}..2^{63}-1$	signed 64-bit
Byte	0..255	unsigned 8-bit
Word	0..65535	unsigned 16-bit
Longword	0..4294967295	unsigned 32-bit

Real Types

floating-point numbers

Type	Range	Significant digits
Single	1.5 x 10^{-45}..3.4 x 10^{38}	**7 - 8**
Double	5.0 x 10^{-324}..1.7 x 10^{308}	**15 - 16**

Type	Range	Significant digits
Real48	2.9 x 10^{-39}..1.7 x 10^{38}	**6**
Comp	$-2^{63}+1..2^{63}-1$	**8**
Currency	**-922337203685477.5808..922337203685477.5807**	**8**
Extended	3.6 x 10^{-4951}..1.1 x 10^{4932}	**10**

Date and Time

TDateTime stores years, months, days, hours, minutes, seconds in one variable (double).

TDateTime functions:

System Routines For The Tdatetime Type

Now	**Returns the current date and time into a single TDateTime value.**
Date	**Returns only the current date.**
Time	**Returns only the current time.**
DateTimeToStr	**Converts a date and time value into a string, using default formatting; to have more control on the conversion use the FormatDateTime function instead.**
DateTimeToString	**Copies the date and time values into a string buffer, with default formatting.**
DateToStr	**Converts the date portion of a TDateTime value into a string.**
TimeToStr	**Converts the time portion of a TDateTime value into a string.**

Date and Time

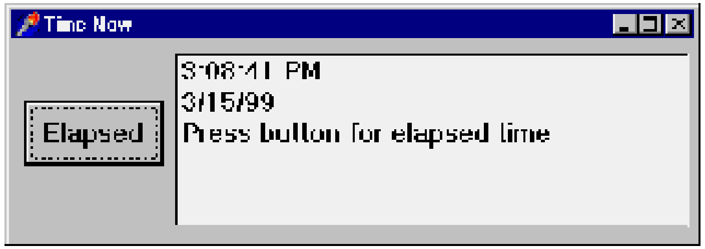
System Routines For The Tdatetime Type

Routine	Description
FormatDateTime	Formats a date and time using the specified format; you can specify which values you want to see and which format to use, providing a complex format string.
StrToDateTime	Converts a string with date and time information to a TDateTime value, raising an exception in case of an error in the format of the string.
StrToDate	Converts a string with a date value into the TDateTime format.
StrToTime	Converts a string with a time value into the TDateTime format.
DayOfWeek	Returns the number corresponding to the day of the week of the TDateTime value passed as parameter.
DecodeDate	Retrieves the year, month, and day values from a date value.
DecodeTime	Retrieves out of a time value.
EncodeDate	Turns year, month, and day values into a TDateTime value.
EncodeTime	Turns hour, minute, second, and millisecond values into a TDateTime value.

Date and Time

Example

```
……
StartTime := Now;
 ListBox1.Items.Add (TimeToStr (StartTime));
 ListBox1.Items.Add (DateToStr (StartTime));
…….
StopTime := Now;
ListBox1.Items [2] := FormatDateTime ('hh:nn:ss',
StopTime - StartTime);
```



```
3:08:41 PM
3/15/99
Press button for elapsed time
```

Typecasting and Type Conversions

Be careful using typecasting:

```
Z:=Integer(X);  C:=Char(N);
```

Use system routines:

```
Z:=Round(X);  C:=IntToStr(N);
```

Routine	Description
Chr	Converts an ordinal number into an ANSI character.
Ord	Converts an ordinal-type value into the number indicating its order.
Round	Converts a real-type value into an Integer-type value, rounding its value.
Trunc	Converts a real-type value into an Integer-type value, truncating its value.
Int	Returns the Integer part of the floating-point value argument.
IntToStr	Converts a number into a string.
IntToHex	Converts a number into a string with its hexadecimal representation.
StrToInt	Converts a string into a number, raising an exception if the string does not represent a valid integer.
StrToIntDef	Converts a string into a number, using a default value if the string is not correct.

Typecasting and Type Conversions

System Routines For Type Conversion

Routine	Description
Val	Converts a string into a number
Str	Converts a number into a string, using formatting parameters
StrPas	Converts a null-terminated string into a Pascal-style string. This conversion is automatically done for AnsiStrings in 32-bit Delphi
StrPCopy	Copies a Pascal-style string into a null-terminated string.
StrPLCopy	Copies a portion of a Pascal-style string into a null-terminated string. FloatToDecimal
FloatToStr	Converts the floating-point value to its string representation using default formatting.
FloatToStrF	Converts the floating-point value to its string representation using the specified formatting.
FloatToText	Copies the floating-point value to a string buffer, using the specified formatting. FloatToTextFmt
StrToFloat	Converts the given Pascal string to a floating-point value.
TextToFloat	Converts the given null-terminated string to a floating-point value.

2.5 User-defined Data Types

Make your own data types, then define your variables

Subrange type
Set type
Enumerated type
Array
Record
Pointer
File type

Subrange types

Subrange type defines a range of values within the range of another type

```
type
Ten = 1..10;
OverHundred = 100..1000;
Uppercase = 'A'..'Z';
```

```
Var
MyIndex:Ten;
MyBigNumber: OverHundred;
```

Enumerated types

Enumerated types

```
Type
    Colors = (Red, Yellow, Green, Cyan, Blue);
    Suit = (Club, Diamond, Heart, Spade);
```

Ordinality of enumeration starts from 0

```
Ord(Red) returns 0;      Ord(Yellow)  returns 1;
```

Set types

Set types are a group of values.

```
Type
  // use enumeration to define a set type
  PeopleTypes = (student, teacher, worker, farmer);
  GroupType=set of PeopleTypes;
Var
 Group1,Group2,Group3 : GroupType;
```

```
// use of set
Group1, Group2, Group3: GroupType;
Group1:=[student,teacher];
Group2:= [worker, farmer];
Group3:= [student, teacher, worker, farmer];
```

Array

Array types define lists of a **fixed** number of elements of a specific type.

```
type
  DayTemperatures = array [1..24] of Integer;
var
  DayTemp1: DayTemperatures;
procedure AssignTemp;
begin
  DayTemp1 [1] := 54; DayTemp1 [2] := 52;
  DayTemp1 [24] := 66;
  DayTemp1 [25] := 67; // compile-time error
end
```

Array

Element index may be an integer, string, enumerator or boolean.

Element value may be any type.

Use Low and High to get the lower and upper bounds.

```
Var
  Array1 : Array[5..20] of string;
Begin
  ShowMessage(‘length = '+IntToStr(Length(Array1)));
  ShowMessage(‘lowest index = '+IntToStr(Low(Array1)));
  ShowMessage(‘highest  index = '+IntToStr(High(Array1)));
```

Array

```
 // Use indexing to set values of the array
  for i := 5 to 20 do
    Array1[i] := IntToStr(i * 5);
  // Now use indexing to display 2 of the elements
  ShowMessage(‘element 7 value = ‘+ Array1[7]);
  ShowMessage(‘element 20 value = ‘+ Array1[20]);
end;
```

```
Display:
  length = 16
  lowest index = 5
  highest index = 20

  element 7 value = 35
  element 20 value = 100
```

Array

An array can have multiple dimensions.

```
type
YearTemps = array [1..12, 1..31] of Integer;
Var
DayTemp2011 : YearTemps;
// use array: DayTemps2011[9,18]:= 56;
```

```
Type
DayTemperatures = array [1..24] of Integer;
MonthTemps = array [1..31] of DayTemperatures;
YearTemps = array [Jan..Dec] of MonthTemps;
Var
HourTemp2011 : YearTemps;
```

Record

Record types define fixed collections of items of different types.

```
type
Date = record Year: Integer; Month: Byte; Day: Byte;
end;

var
BirthDay: Date;
begin
BirthDay.Year := 1997;
BirthDay.Month := 2;
BirthDay.Day := 14;
```

Record

```
// define
Type
    Student=record
      StuId:integer;
      Name:string[20];
      Age:1..100;
      Addr:string[40];
end;
var student1,student2:student;
…..
// use
      student2.StuId :='20002318';
     student2.Name :='孙刚';
     student2.Age :=25;
     student2.Addr:=‘长安大学';
```

File

File types represent physical disk files.

Rich system components support users to store, load data from files, and to make serialization and work with database.

```
// have  a file to read and write  integer data
type
IntFile = file of Integer;
Var
myFile: IntFile;
```

Pointers

A pointer type defines a variable that holds the memory address of another variable of a given data type

```
var
P: ^Integer;
  X: Integer;
begin
  P := @X;
// change the
value
  X := 10;
  P^ := 20;
```

```
var
P: ^Integer;
Begin
// initialization
New (P);
// use pointer
P^ := 20;
// assign nil to pointer
P:= nil;
// after use, clear it
Dispose (P);
end;
```

Chapter 2 Review I

Delphi programming language

- ✓ Keywords, Operators
- ✓ Variables
- ✓ Predefined data types:

- ✓ Boolean
- ✓ Integer, Real
- ✓ Char, Strings
- ✓ TDateTime

- ✓ User-defined data types

Chapter 2 Review I

- ✓ User-defined data types
  - ✓Subrange
  - ✓Set
  - ✓Enumeration
  - ✓Array
  - ✓File
  - ✓Record
  - ✓Pointer

Chapter 2

- ✓ Keywords, Operators
- ✓ Variables
- ✓ Predefined data types
- ✓ User-defined data types

Statements

Procedures, functions

Code examples

2.6 Statements

Simple and compound statements

```
// Simple statement
X := Y + Z;     // assignment
```

```
// Simple statement
Randomize;      // procedure call
```

```
// Compound
begin
  A := B;
  C := A * 2
end;
```

Assignment statements

```
X := Y + Z;
M:= DoubleValue(X);
```

Conditional Statements

Test an expression, then execute one of statements or none.

If statements

```
begin
  if CheckBox1.Checked then
  ShowMessage (‘C1  is checked')
end;
```

```
if CheckBox2.Checked then
  ShowMessage ('C2 is checked')
else
  ShowMessage ('C2 is NOT
checked');
```

Conditional Statements

Nested if statements :

```
procedure TForm1.Button4Click(Sender: TObject);
begin
    // compound if statement
   if CheckBox1.Checked then
      if CheckBox2.Checked then
         ShowMessage ('CheckBox1 and 2 are checked')
      else
            ShowMessage ('Only CheckBox1 is checked')
   else
      ShowMessage ('Checkbox1 is not checked.')
end;
```

Conditional Statements

Case Statements

The execution depends on an expression used to select a value, a list of possible values, or a range of values.

```
case Number of
1: Text := 'One';
2: Text := 'Two';
3: Text := 'Three';
end;
```

Conditional Statements

Case Statements

```
case MyChar of
'+' : Text := 'Plus sign';
'-' : Text := 'Minus sign';
'*', '/': Text := 'Multiplication or division';
'0'..'9': Text := 'Number';
'a'..'z': Text := 'Lowercase character';
'A'..'Z': Text := 'Uppercase character';
else
Text := 'Unknown character';
end;
```

Example: Number of days in month

```
var
      year:1..3000;    month:1..12;    days:28..31;
begin
      year:=strtoint(edit1.Text );         // get year input
      month:=strtoint(edit2.Text );        // get month input
      case month of
            1,3,5,7,8,10,12: days:=31;
            4,6,9,11:           days:=30;
            2:    if ((year mod 4=0)and(year mod 100<>0))
                      or (year mod 400=0)   then days:=29
                  else days:=28            //判断是否闰年输出二月份天数
      end;
      label3.Caption :='该月天数为'+inttostr(days);         // show result
end;
```

Loops

For Loop

Execution is based on a counter, which can be either increased or decreased by 1 each time the loop is executed.

```
var
   K, I: Integer;
begin
   K := 0;
   for I := 1 to 10 do
      // for I := 10 downto 1 do
      K := K + I;
End;
```

Example: Buying chickens

1 rooster for $5,1 hen for $3, 3 chicks for $1. What are possible combinations to buy 100 chickens using $100?

x rooster, y hen, z chick

x 0..19 y 0..33 $z = 100 – x – y$

$$5x + 3y + z/3 = 100$$

Example: Buying chickens

```
var x,y,z:integer;
    p:string;
begin
    for x:=0 to 19 do
        for y:=0 to 33 do
           begin
              z:=100-x-y;
              if   5*x+3*y+z/3=100 then
                  begin
   p:=format(‘解：公鸡%d只，母鸡%d只，小鸡%d只',[x,y,z]);
                      showmessage(p);
                  end;
          end;
end;
```

Loops

While and Repeat

```
while (I <= 100) and (J <= 100) do
begin
  // code using I and J ...
  I := I + 1; J := J + 1;
end;
```

```
repeat
// code using I and J ...
// it executes at least once
  I := I + 1; J := J + 1;
until (I > 100) or (J > 100);
```

Statements

With Statement is a shorthand to refer to a record type variable or an object.

```
type
  Date = record Year: Integer; Month: Byte; Day: Byte;
end;
var
  BirthDay: Date;
```

```
begin
  BirthDay.Year := 2011;
  BirthDay.Month := 5;
  BirthDay.Day := 14;
End;
```

```
with BirthDay do
begin
  Year := 2011;
  Month := 5;
  Day := 14;
end;
```

2.7 Procedures and Functions

A procedure or function is a routine made of a series of statements with a unique name.

It can be activated many times by calling its name with parameters.

A function has a result, a return value, while a procedure doesn't.

Advantage： easy to manage, reuse and change code.

Procedures and Functions

Define

```
procedure Hello;
begin
   ShowMessage ('Hello world!');
end;

function Double (Value: Integer) : Integer;
begin
  Result := Value * 2;
  // Double := Value * 2;
end;
```

Procedures and Functions

Call

```
procedure TForm1.Button1Click (Sender: TObject);
begin
  Hello;
end;

procedure TForm1.Button2Click (Sender: TObject);
var
  X, Y: Integer;
begin
  X := Double (StrToInt (Edit1.Text)); Y := Double (X);
  ShowMessage (IntToStr (Y));
end;
```

Procedures and Functions

Passing parameters by value (Default)

When you change parameter value inside routine, original value are unchanged.

```
procedure DoubleTheValue (Value: Integer);
begin
  Value := Value * 2;
end;

var
  X: Integer;
begin
  X := 10; DoubleTheValue (X);    //  still X=10
```

Procedures and Functions

Passing parameters by reference

When you change parameter value inside routine, original value will be changed.

```
procedure DoubleTheValue (var Value: Integer);
begin
  Value := Value * 2;
end;

var
  X: Integer;
begin
  X := 10; DoubleTheValue (X);   // now X=20
```

Procedures and Functions

Constant parameters

Original value won't be affected by the routine, while performance is optimized.

```
function DoubleTheValue (const Value: Integer): Integer;
begin
  Value := Value * 2; // compiler error
  Result := Value;
end
……
  X := 10; DoubleTheValue (X);    //  still X=10
```

Procedures and Functions

Out parameters

```
procedure DoIt(Const A : Integer; Out B : Integer);
 begin
   B := A * 2;
 end;

 procedure TForm1.FormCreate(Sender: TObject);
 var   A, B : Integer;
 begin
   A := 22;
   DoIt(A, B);
   ShowMessageFmt(‘B has been set to = %d’,[B]);
 end;
```

Procedures and Functions

Open array parameters is a way to pass a varyingnumber of parameters to a routine.

```
function Sum (const A: array of Integer): Integer;
var
  I: Integer;
begin
  Result := 0;
  for I := Low(A) to High(A) do
      Result := Result + A[I];
end;
```

```
// calling function
X := Sum ([10, Y, 27*I]);
```

Procedural Types (advanced topic)

Declare
a procedural type

```
type
  IntProc = procedure (var Num: Integer);
```

Create a compatible procedure

```
procedure DoubleValue (var Value:Integer);
begin
  Value := Value * 2;
end;
```

Specify the name of actual procedure, then use procedural routine

```
var  IP: IntProc; X: Integer;
begin
   IP := DoubleValue; X := 5;
   IP (X);
end;
```

Example: ordering a set of integers

Assign random number to 20 integers.
(range 0..50).

Order them from big to small.

Example: ordering a set of integers

```
 var     A:Array[1..20] of Integer;
procedure AssignRandomNumber();
 //产生随机数
Var   i:Integer;
begin
      Randomize;
      for i:=Low(A) to High(A) do
        begin
           A[i]:=random(50);
//产生一个大于等于0 小于50的随机数
        end;
 end;
```

Example: ordering a set of integers

```
// 变量
var     A:Array[1..20] of Integer;

// 过程：产生随机数
procedure AssignRandomNumber();
Var   i:Integer;
begin
      Randomize;
      for i:=Low(A) to High(A) do
        begin
           A[i]:=random(50);
//产生一个大于等于0 小于50的随机数
        end;
end;
```

Example: ordering a set of integers

```
// 过程：对比与交换数值
procedure MakeBigerFirst
            (var n1:integer,
var n2:integer);
Var temp:integer;
begin
     (if (n1<n2) then
        begin
          temp:=n2;
          n2:=n1;
          n1:=temp;
        end
End
```

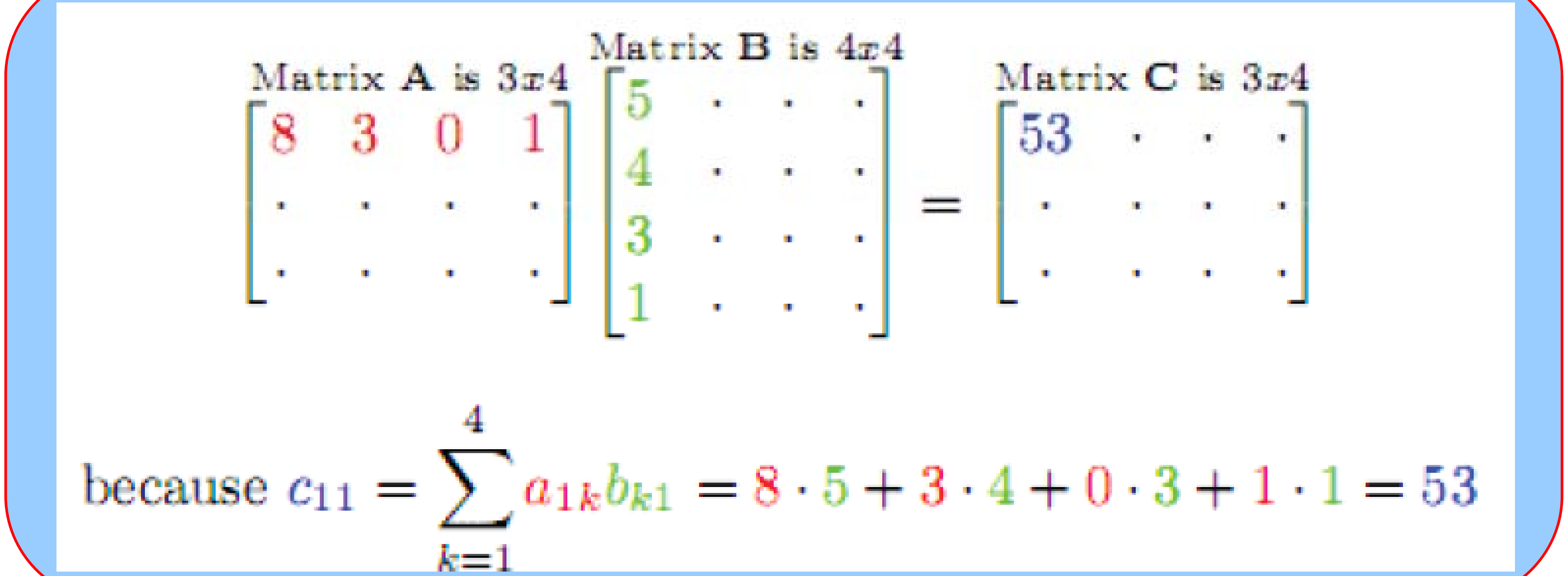
```
//过程：排序
procedure Ordering();
var i,j,temp:integer;
begin
  for i:=Low(A) to High(A) do
    for j:=i+1 to High(A) do
        //调用过程：
    MakeBigerFirst(A[i], A[j]);
end;
```

Example: ordering a set of integers

```
// 变量
var     A:Array[1..20] of Integer;

// 调用过程
begin
    AssignRandomNumber();
    Ordering();
end;
```

Example: Matrix multiplication



$$\overset{\text{Matrix }\mathbf{A}\text{ is }3x4}{\begin{bmatrix}8&3&0&1\\ \cdot&\cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot&\cdot\end{bmatrix}}\overset{\text{Matrix }\mathbf{B}\text{ is }4x4}{\begin{bmatrix}5&\cdot&\cdot&\cdot\\ 4&\cdot&\cdot&\cdot\\ 3&\cdot&\cdot&\cdot\\ 1&\cdot&\cdot&\cdot\end{bmatrix}}=\overset{\text{Matrix }\mathbf{C}\text{ is }3x4}{\begin{bmatrix}53&\cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot&\cdot\end{bmatrix}}$$

because $c_{11}=\sum_{k=1}^{4}a_{1k}b_{k1}=8\cdot5+3\cdot4+0\cdot3+1\cdot1=53$

Example: Matrix multiplication

```
// 二维动态数组变量
Var a:array of array of integer;
        b:array of array of integer;
        c:array of array of integer;
```

```
// 规定长度
 setlength(a,3,4);
 setlength(b,4,4);
 setlength(c,
high(a)+1,high(b[0])+1);
// 用双重For loop赋值
//    a[i,j] = ….. b[i,j]=….
```

```
// 矩阵乘法运算
begin
for i:=0 to high(a) do
          begin
              for j:=0 to high(b[0]) do
                 begin
                   c[i,j]:=0;
                   for k:=0 to high(b) do
                     c[i,j]:=a[i,k]*b[k,j]+c[i,j];
                 end;
           end
end;
```

Chapter 2 Review II

DELPHI PROGRAMMING LANGUAGE

✓Statements:

- ✓If statement
- ✓Case statement
- ✓For statement

✓Procedures, functions:

- ✓Define and use
- ✓Passing parameters ….

Chapter 2 Summary

Pascal ---- Delphi programming language

- ✓Keywords, Operators
- ✓Variables
- ✓Predefined data types
- ✓User-defined data types
- ✓Statements
- ✓Procedures and functions

Chapter 2

DELPHI PROGRAMMING LANGUAGE

Q & A