1-dimensional arrays

5C - One-dimensional Arrays (author: Tao Yue, state: unchanged)

Suppose you wanted to read in 5000 integers and do something with them. How would you store the integers?

You could use 5000 variables, lapsing into:

```
aa, ab, ac, ad, ... aaa, aab, ... aba, ...
```

But this would grow tedious (after declaring those variables, you have to read values into each of those variables).

An array contains several storage spaces, all the same type. You refer to each storage space with the array name and with a subscript. The type definition is:

```
type
  typename = array [enumerated_type] of another_data_type;
```

The data type can be anything, even another array. Any enumerated type will do. You can specify the enumerated type inside the brackets, or use a predefined enumerated type. In other words,

```
type
  enum_type = 1..50;
  arraytype = array [enum_type] of integer;
```

is equivalent to

```
type
  arraytype = array [1..50] of integer;
```

Aside: This is how strings are actually managed internally — as arrays. Back before modern Pascal compilers added native support for strings, programmer had to handle it themselves, by declaring:

```
type
  String = packed array [0..255] of char;
```

and using some kind of terminating character to signify the end of the string. Most of the time it's the null-character (ordinal number 0, or ord(0)). The packed specifier means that the array will be squeezed to take up the smallest amount of memory.

Arrays of characters representing strings are often referred to as buffers, and errors in handling them in the C or C++ programming languages may lead to buffer overruns. A buffer overrun occurs when you try to put, say, a 200-character string into a 150-length array. If memory beyond the buffer is overwritten, and if that memory originally contained executable code, then the attacker has just managed to inject arbitrary code into your system. This is what caused the famous Slammer worm that ran rampant on the Internet for several days. Try it in Pascal and see what happens.

Arrays are useful if you want to store large quantities of data for later use in the program. They work especially well with for loops, because the index can be used as the subscript. To read in 50 numbers, assuming the following definitions:

```
type
  arraytype = array[1..50] of integer;

var
  myarray : arraytype;
```

use:

```
for count := 1 to 50 do
  read (myarray[count]);
```

Brackets `[ ]` enclose the subscript when referring to arrays.

```
myarray[5] := 6;
```

Category: Object Pascal Introduction