In short, using `new` to create an array and using a pointer to access the different elements is a simple matter. Just treat the pointer as an array name. Understanding why this works, however, is an interesting challenge. If you actually want to understand arrays and pointers, you should review their mutual relationships carefully. In fact, you've been exposed to quite a bit of pointer knowledge lately, so let's summarize what's been revealed about pointers and arrays to date.

**Summarizing Pointer Points**

**Declaring Pointers:** To declare a pointer to a particular type, use this form:

```
typeName * pointerName;
```

Examples:

```c
double * pn;       // pn points to a double value
char * pc;         // pc points to a char value
```

Here `pn` and `pc` are pointers and `double *` and `char *` are the C++ notations for the types `pointer-to-double` and `pointer-to-char`.

**Assigning Values to Pointers:** You should assign a pointer a memory address. You can apply the `&` operator to a variable name to get an address of named memory, and the `new` operator returns the address of unnamed memory.

Examples:

```c
double bubble = 3.2;
pn = &bubble;      // assign address of bubble to pn
pc = new char;     // assign address of newly allocated char memory to pc
```

**Dereferencing Pointers:** Dereferencing a pointer means referring to the pointed-to value. Apply the dereferencing, or indirect value, operator (`*`) to a pointer to dereference it. Thus, if `pn` is a pointer to `bubble`, as in the last example, then `*pn` is the pointed-to value, or 3.2, in this case.

Examples: