cout << *pn; // print the value of bubble
*pc = 'S';   // place 'S' into the memory location whose address is pc

Array notation is a second way to dereference a pointer; for instance, pn[0] is the same as pn. Never dereference a pointer that has not been initialized to a proper address.

**Distinguishing Between a Pointer and the Pointed-to Value:** Remember, if pt is a pointer-to-int, that *pt is not a pointer-to-int; instead, *pt is the complete equivalent to a type int variable. It is pt that is the pointer.

Examples:

```cpp
int * pt = new int; // assigns an address to the pointer pt
pt = 5;            // stores the value 5 at that address
```

**Array Names:** In most contexts, C++ treats the name of an array as equivalent to the address of the first element of an array.

Example:

```cpp
int tacos[10];       // now tacos is the same as &tacos[0]
```

One exception is when you use the name of an array with the sizeof operator. In that case, sizeof returns the size of the entire array, in bytes.

**Pointer Arithmetic:** C++ allows you to add an integer to a pointer. The result of adding 1 equals the original address value plus a value equal to the number of bytes in the pointed-to object. You also can subtract an integer from a pointer and take the difference between two pointers. The last operation, which yields an integer, is meaningful only if the two pointers point into the same array (pointing to one position past the end is allowed, too); it then yields the separation between the two elements.

Examples:

```cpp
int tacos[10] = {5,2,8,4,1,2,2,4,6,8};
int * pt = tacos;       // suppose pf and fog are the address 3000
pt = pt + 1;            // now pt is 3004 if a int is four bytes
int *pe = &tacos[9];    // pe is 3036 if an int is four bytes
pe = pe -1;            // now pe is 3032, the address of tacos[8]
```