CS2005: Pointers in C^{++}

Code	Meaning	Comments
Declarations.		
`Item x;`	variable	reserves space
`Item *y;`	pointer variable	reserves no space for data, only for pointer
Functions and Parameters.		
`foo(Item x);`	value parameter	Local copy of value is made in function. Changing x has no outside effect.
`foo(Item &x);`	reference parameter	No local copy; the original data is used. Changing x will affect outside data.
`foo(Item* x);`	pointer parameter	Only a pointer is passed. Local copy of the POINTER (not data!) is made. Changing *x will affect data, changing x itself (the pointer) will have no effect on the outside.
`foo(Item*& x)`	pointer reference parameter	The pointer which is passed is not copied. Changing `*x` will affect outside data, changing `x` itself will change the outside pointer (if you called `foo(z)` and changed `x` inside `foo()`, `z` will now point to something else)
`Item foo(...)`	value return parameter	Function returns a value. Whatever is returned is COPIED before the function finishes.
`Item* foo(...)`	reference return parameter	Function returns a pointer to some data. The local pointer itself is copied before the function finishes but not the data that it points to—if the return value points to locally defined data, you will get wrong results.
Assignment. Assume above declarations for x and y		
`Item z = x;`	copy	a copy of the data in x will be copied into `z`. If Item is a class, make sure the copy constructor/assignment operator are overloaded.
`Item z = *y;`	dereference and copy	a copy of the data that y points to is written into `z`.
`Item *z = &x`	copy reference	z points to the address of x in memory. No data is copied.
`Item *z = y;`	copy pointer	z points to the same thing as y. No data is copied. If you change something that y points to, the data that z points to is also affected (since it's the same data).
`Item *z= new Item;` `*z = *y;`	copy data using two pointers	Copies whatever y points to into the memory that z points to. You have to make sure that z points to allocated memory.