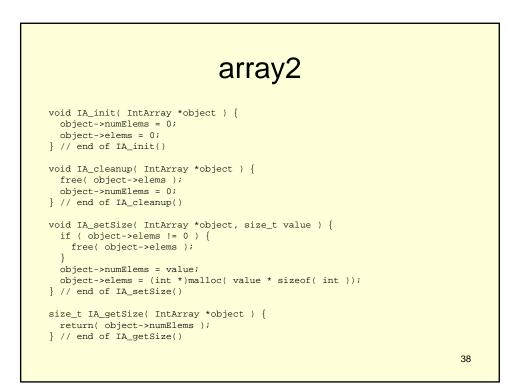
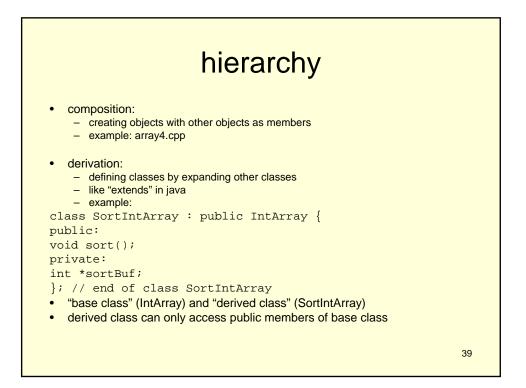
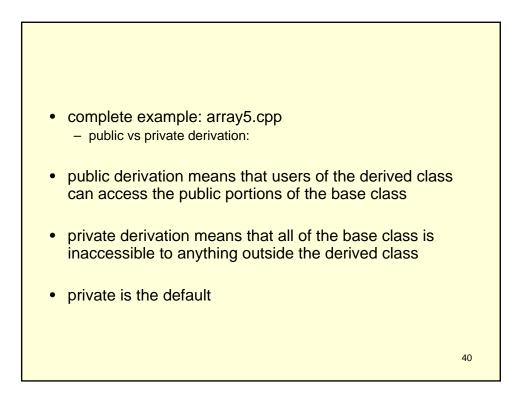


```
array1.cpp
struct IntArray {
  int *elems;
  size_t numElems;
};
main() {
  IntArray powersOf2 = \{0, 0\};
  powersOf2.numElems = 8;
  powersOf2.elems = (int *)malloc( powersOf2.numElems *
  sizeof( int ));
  powersOf2.elems[0] = 1;
  for ( int i=1; i<powersOf2.numElems; i++ ) {</pre>
    powersOf2.elems[i] = 2 * powersOf2.elems[i-1];
  }
  cout << "here are the elements:\n";</pre>
  for ( int i=0; i<powersOf2.numElems; i++ ) {</pre>
  cout << "i=" << i << " powerOf2=" <<
powersOf2.elems[i] << "\n";</pre>
  }
  free( powersOf2.elems );
}
                                                                37
```







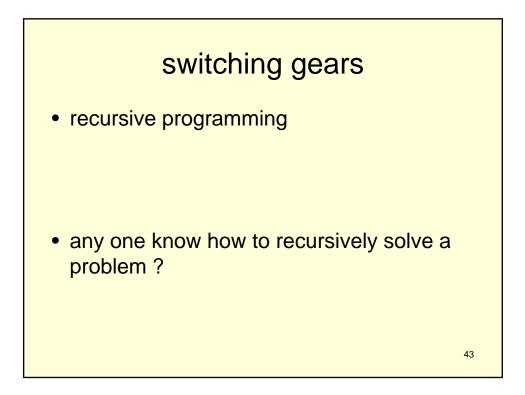
Class derivation

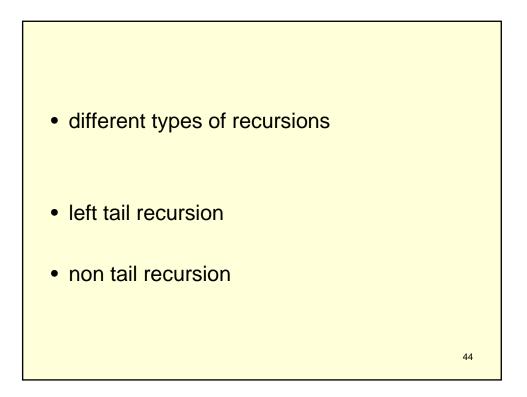
- encapsulation
 - derivation maintains encapsulation
 - i.e., it is better to expand IntArray and add sort() than to modify your own version of IntArray
- friendship
 - not the same as derivation!!
 - example:
- is a friend of
- B2 is a friend of B1
- D1 is derived from B1
- D2 is derived from B2
- B2 has special access to private members of B1 as a friend
- But D2 does not inherit this special access
- nor does B2 get special access to D1 (derived from friend B1)

Derivation and pointer conversion derived-class instance is treated like a base-class instance but you can't go the other way • example: main() { IntArray ia, *pia; // base-class object and pointer StatsIntArray sia, *psia; // derived-class object and pointer pia = &sia; // okay: base pointer -> derived object psia = pia; // no: derived pointer = base pointer psia = (StatsIntArray *)pia; // sort of okay now since: // 1. there's a cast // 2. pia is really pointing to sia, // but if it were pointing to ia, then // this wouldn't work (as below) psia = (StatsIntArray *)&ia; // no: because ia isn't a StatsIntArray

42

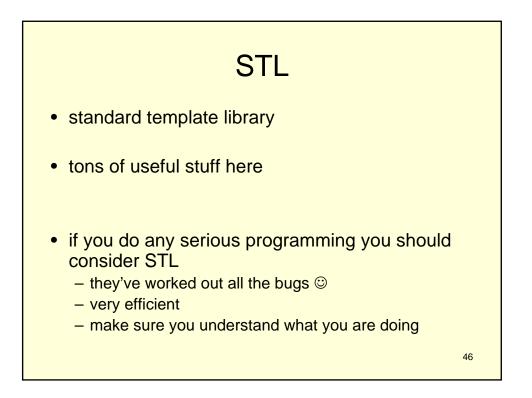
41





Templates

template<typename X>
void foo(X &first, X second){
 first += second;
 }



45

