Templated libraries

- Common to use same ADT with different stored data, e.g. a list of integers, a list of strings, a list of student records, etc
- Most of the code stays the same, just the stored data type that changes (assuming a few common operations supported across the data types, e.g. assignment, copy, equality tests, etc)
- If supported by language/compiler, we write a template for the ADT (e.g. the list), but using a placeholder for the stored data type
- When needed, we declare “L is a list of ints” and let the compiler use our template to generate the actual code for a list of ints

C++ standard template library (STL)

- The C++ STL provides a wide variety of ADTs in templated form, e.g. for stacks, queues, lists, vectors, sets, etc
- When we want to create, say, a stack of ints, we include the appropriate library (<stack>) and when we declare a stack we specify the datatype to be used: stack<int> S;
- We can then use any of the provided methods, e.g. S.push(27);
- ADTs in the STL are widely used, robust – try them!

Creating our own templates

- We can create our own templated classes and/or functions
- We tell compiler that the thing we're about to define is just a template for the compiler to use if/when needed, specifying the name we'll use for the "unknown" data type

```
template <class T>
void swap(T &x, T&y) {
    T tmp = x;  x = y;  y = tmp;
}
```

Compiler builds what it sees used

- For the swap example, if compiler sees instruction like swap(user1,user2); it looks at the data types of user1 and user2 – if they're both ints then it builds an int version of swap, if they're both strings then it builds a string version of swap, etc
- We can also include code that explicitly tells the compiler some of the things to build (instantiate), e.g. syntax for telling it to build an int version of swap is

```
template void swap<int>(int, int)
```

Separate compilation issues

- Suppose we have a prototype for swap in swap.h and the code for swap in swap.cpp
- Suppose we also have a file, somecode.cpp, that #includes swap.h and calls swap on a pair of doubles
- When the compiler processes swap.cpp, it has no inkling of what's in somecode.cpp, so doesn't build a doubles version
- When compiler processes somecode, it has no idea what was put into swap.o, just trusts there's a double version
- When we go to link swap.o and somecode.o we get a linking error

Two possible solutions

- In swap.h we could put a list of instantiations for swap, specifying all the data types permitted for swap
- That works, but not very flexible – need to edit and recompile if we think of an additional data type
- Alternative is to put the entire template in the swap.h file, so that the whole template gets included by any file making use of swap
- This works as desired, but does expose the swap implementation details in the .h file