Tools for designing and constructing programs

- Methods for using C++, other OO languages
  - classes: encapsulation
  - inheritance: program to an interface, reuse and re-implement
  - design patterns (e.g., iterators)

Algorithms and Data structure toolkit

- vectors, lists, stacks, queues ...
- sorting algorithms, searching algorithms

- Methods for measuring performance/complexity
debugging: what and how

- add "print" (aka cout) statements
  - takes time, edit/compile/debug, repeat
  - hard to do with pointers, linked lists
- build in debugging/print routines
  - print list
  - verify properties of code, can use assert(...)
- use a debugger: gdb, ddd
  - ddd is very useful, also shows structures graphically
  - gdb sometimes fast to pinpoint error locations
Three (or more) list classes

- managing all the versions is arduous/cumbersome
- harder to use the versions in the same program

- different implementations of the same interface
  - streams: ifstream, cin, istrstream
  - streams: ofstream, cout, ostrstream
    ostream & operator << (ostream & os, const Anaword & a);

- works through inheritance: ofstream “is-a” ostream, cout “is-a” ostream, ...
stacks/queues

● **How does recursion work, where are clones stored?**

```cpp
void Print(Node * list) {
    if (list != 0) {
        Print(list->next);
        cout << list->info << endl;
    }
}
```

● **fruitlist = ("apple", "cherry", "guava", "orange")**

● **results of Print(fruitlist)?**

● **clones are called, then finish, managed on runtime stack**
Runtime stack and recursion

```c
void Print(Node * list)
{
    if (list != 0)
    {
        Print(list->next);
        cout << list->info << endl;
    }
}
```

```c
Print(fruitList);
```

```
apple  cherry  guava  orange
```

![Diagram of a linked list with fruits: apple, cherry, guava, orange, and a call to Print(fruitList).]
stack as ADT and class

- abstract data type for stack
  - operations: construct, push, pop, top, isEmpty
  - items on the stack: anything

push A, push B, push C, push D

push A, push B, push C, push D

pop, pop

push A, push B, push C, push D

top = ?

top = ?

top = ?

template <class Kind>
class Stack
{
    void push(const Kind &);
    ...
}
Reverse Polish/Postfix notation

- see `stack.h`, `stack.cc`, `postfix.cc` for code
  
  4 * 3 + 8 = ???
  
  (7 - 4) * 2 = ???

- what about program to evaluate these expressions?
  
  4 3 * 8 + = ???
  
  7 4 - 2 * = ???

- no parens needed for precedence
  
  ➤ read from left to right
  
  ➤ if operand: (e.g., 7) push on stack
  
  ➤ if operator: pop, pop, operate, push result
what’s a templated class?

- often a container class
  - contains objects of the same type: int, string, ...
  - serves as a template from which code is generated
- the .h file provides the interface, the .cc file provides a template implementation/definition
  - `vector<int> v; stack<double> s;`
  - must have access to definition to instantiate
- choices: all code in .h, .h includes .cc, explicit instantiation
  - `#include “stack.cc”`