```
-----------------------------------------------------------------
lessthan.cpp: function object that works with STL
-----------------------------------------------------------------

template <typename T>
struct lessthan : public binary_function<T,T,bool> {
  bool operator() (const T &A, const T &B) const {return A<B;}
};

example usage:  sort(V.begin(), V.end(), lessthan<string>());

-----------------------------------------------------------------
stack.cpp: std::list class made to work like stack
-----------------------------------------------------------------

#include <...>
#include <list>
using namespace std;

template <class T>
class stack : private std::list<T> {
  typedef std::list<T> CONTAINER;

  public:
    stack() {}
    ~stack() {}

    void push(T const& value) { CONTAINER::push_back(value); }
    void pop() { CONTAINER::pop_back(); }
    T const& top() const { return CONTAINER::back(); }

    using CONTAINER::size;
    using CONTAINER::empty;
};

int main() {
  stack<string> S;

  string input;
  while (cin >> input)
    S.push(input);

  cout << "\n";
  while (!S.empty()) {
    cout << S.size() << ": " << S.top() << "\n";
    S.pop();
  }
}
```

```
-----------------------------------------------------------------
classchain.cpp: contructor/destructor/function calls
-----------------------------------------------------------------

#include ...
using namespace std;

class A {
  public:
    A() { cout << "A: new\n"; }
    virtual ~A() { cout << "A: delete\n"; }
    virtual void hello() { cout << "A: hello\n"; }
};

class B : public A {
  public:
    B() { cout << "B: new\n"; }
    ~B() { cout << "B: delete\n"; }
    void hello() { cout << "B: hello\n"; }
};

-----------------------------------------------------------------

int main() {
   A *myobj = new A;  // using class A pointer to A object
   myobj->hello();    // dynamic binding to A function
   delete myobj;
}

unix> ./classchain
A: new   A: hello   A: delete

-----------------------------------------------------------------

int main() {
   A *myobj = new B;  // using class A pointer to B object
   myobj->hello();    // dynamic binding to B function
   delete myobj;
}

unix> ./classchain
A: new   B: new   B: hello   B: delete   A: delete
```

```
-----------------------------------------------------------------
progression.cpp: base class and derived classes (see HW13)
-----------------------------------------------------------------

class progression {
  public:
    progression(int n_N=10) { N=n_N; }
    virtual ~progression() { ; }
    virtual void print();

  protected:
    int first;
    virtual long next(int) =0;

  private:
    int N;
};

void progression::print() {
  cout << first;
  for (int i=1; i<N; i++)
    cout << " " << next(i);
  cout << "\n";
}

-----------------------------------------------------------------

class arithmetic_prog : public progression {
  public:
    arithmetic_prog(int n_first, int n_inc)
      { first=n_first; inc=n_inc; }

  private:
    int inc;
    long next(int);
};

long arithmetic_prog::next(int i) {
  return first + i*inc;
}

-----------------------------------------------------------------
Hint: arithmetic_prog executes the progression constructor as is,
uses base member first along with its own member inc in function
next which computes the next number in the arithmetic progression.

unix> ../progression -a 2 4
2 6 10 14 18 22 26 30 34 38
```

```
class geometric_prog: public progression {
  public:
    geometric_prog(int n_first, int n_ratio) : progression(5)
      { first=n_first, ratio=n_ratio; }

  private:
    int ratio;
    long next(int);
};

long geometric_prog::next(int i) {
  if (i == 0)
    return first;

  return ratio*next(i-1);
}

-----------------------------------------------------------------
Hint: geometric_prog passes the progression constructor a value
of 5 which sets base member N accordingly, uses base member first
along with its own member ratio in function next which computes
the next number in the geometric progression.

unix> ../progression -g 2 4
2 8 32 128 512

-----------------------------------------------------------------

int main(int argc, char *argv[]) {
  if (argc != 4) {
    cerr << "usage: " << argv[0]
         << " -a first inc | -g first ratio\n";
    return 0;
  }

  progression *p = NULL;

  if (argv[1][1] == 'a')
    p = new arithmetic_prog(argv[2], argv[3]);
  else
  if (argv[1][1] == 'g')
    p = new geometric_prog(argv[2], argv[3]);

  if (p) {
    p->print();
    delete p;
  }
}
```