```
-----------------------------------------------------------------
rnumgen_uniform.cpp: generating uniform random variables
-----------------------------------------------------------------

#include <...>
using namespace std;

int main(int argc, char *argv[]) {
  int N=10, M=100, W=20;

  srand(0);

  for (int i=0; i<N; i++)
    cout << (rand() % M) + W << "\n";
}

-----------------------------------------------------------------
rnumgen_dist.cpp: generating non-uniform random variables
-----------------------------------------------------------------

#include <...>
using namespace std;

class rnumgen {
  public:
    rnumgen(int seedvalue, std::vector<int> &v);
    int rand();

  private:
    std::vector<float> F;
};

rnumgen::rnumgen(int seedvalue, vector<int> &v) {
  srand(seedvalue);

  F.resize(v.size());
  partial_sum(v.begin(), v.end(), F.begin());
  transform(F.begin(), F.end(), F.begin(),
  bind2nd(divides<float>(), F.back()));
}

int rnumgen::rand() {
  const double randmax = RAND_MAX+1.0;
  const double p = (double)std::rand()/randmax;

  return upper_bound(F.begin(), F.end(), p) - F.begin();
}
```

```
int main(int argc, char *argv[]) {
  if (argc < 4) {
    cerr << "usage: " << argv[0]
         << " seed N label1=ratio1 ... labelN=ratioN\n";
    return 0;
  }

  int seedvalue = atoi(argv[1]);
  int N = atoi(argv[2]);

  vector<string> label;
  vector<int> ratio;

  for (int i=3; i<argc; i++) {
    string new_label = strtok(argv[i], "=");
    int new_ratio = atoi(strtok(NULL, "="));

    label.push_back(string(new_label));
    ratio.push_back(new_ratio);
  }

  if (seedvalue == 0) seedvalue = time(NULL);

  rnumgen RNG(seedvalue, ratio);

  for (int i=0; i<N; i++)
    cout << " " << label[RNG.rand()] << "\n";
}
-----------------------------------------------------------------
unix> ./rnumgen_dist 0 1000 dog=1 cat=2 | more
 dog
 cat
 cat
 ...
unix> ./rnumgen_dist 0 1000 dog=1 cat=2 | sort | uniq -c
 664  cat
 336  dog
-----------------------------------------------------------------
Hint: Labels are stored in a vector. The random number generator
produces integer indices into that vector. The distribution is
generated based on the ratio numbers associated with the labels.

Hint: See cplusplus.com sections on algorithm, functional and
numeric support for explanations of the various STL algorithms.

Hint: Function upper_bound() returns an iterator to the F-entry
which is greater than probability p. Subtracting off F.begin()
yields the index needed. This is similar to i == &F[i] - &F[0].
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
randperm.cpp: random permutation of list data
-----------------------------------------------------------------

#include <...>
using namespace std;

template <typename T>
void randperm(vector<T> &v) {
  for (int i=(int)v.size()-1; i>0; --i) {
    swap(v[i], v[rand() % (i+1)]);
  }
}

int main(int argc, char *argv[]) {
  srand(time(NULL));

  string din;
  vector<string> A;

  while (cin >> din)
    A.push_back(din);

  randperm(A);

  for (int i=0; i<(int)A.size(); i++)
    cout << A[i] << "\n";
}

-----------------------------------------------------------------
unix> echo 1 2 3 4 | ./randperm1
2
3
1
4

-----------------------------------------------------------------
Hint: Data stored in A is permuted by randomly selecting elements
to swap while incrementally taking elements out of the pool of
available data to process.

Hint: The code to the right permutes integer indices "pointing"
to the data stored in A. The data is put in the right random
order by reading them in the order specified by Ap. This is
similar to what you are doing for the smart pointers in Lab 3.
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
randperm.cpp: indirect random permutation of list data
-----------------------------------------------------------------

#include <...>
using namespace std;

template <typename T>
void randperm(vector<T> &v) { ... }

int main(int argc, char *argv[]) {
  srand(time(NULL));

  string s, tmp;
  vector<string> A;

  int i=0, j, nextj;
  vector<int> Ap;

  while (cin >> s) {
    A.push_back(s);
    Ap.push_back(i++);
  }

  randperm(Ap);

  for (i=0; i<(int)A.size(); i++) {
    if (Ap[i] != i) {
      tmp = A[i];
      for (j=i; Ap[j] != i; j=nextj) {
        nextj = Ap[j];
        A[j] = A[nextj];
        Ap[j] = j;
      }
      A[j] = tmp;
      Ap[j] = j;
    }
  }

  for (int i=0; i<(int)A.size(); i++)
    cout << A[i] << "\n";
}
```