measuring, say, the water level of the Pacific Ocean as the tides varied, we might want to be able to represent negative feet-and-inches quantities. (Tide levels below mean-lower-low-water are called *minus tides*; they prompt clam diggers to take advantage of the larger area of exposed beach.)

Let's derive a new class from `Distance`. This class will add a single data item to our feet-and-inches measurements: a sign, which can be positive or negative. When we add the sign, we'll also need to modify the member functions so they can work with signed distances. Here's the listing for ENGLEN:

```
// englen.cpp
// inheritance using English Distances
#include <iostream>
using namespace std;
enum posneg { pos, neg };          //for sign in DistSign
////////////////////////////////////////////////////////////////
class Distance                     //English Distance class
   {
   protected:                      //NOTE: can't be private
      int feet;
      float inches;
   public:                         //no-arg constructor
      Distance() : feet(0), inches(0.0)
         {  }                      //2-arg constructor)
      Distance(int ft, float in) : feet(ft), inches(in)
         {  }
      void getdist()               //get length from user
         {
         cout << "\nEnter feet: ";  cin >> feet;
         cout << "Enter inches: ";  cin >> inches;
         }
      void showdist() const        //display distance
         { cout << feet << "\'-" << inches << '\"'; }
   };
////////////////////////////////////////////////////////////////
class DistSign : public Distance  //adds sign to Distance
   {
   private:
      posneg sign;                 //sign is pos or neg
   public:
                                   //no-arg constructor
      DistSign() : Distance()      //call base constructor
         { sign = pos; }           //set the sign to +
```

```
                                  //2- or 3-arg constructor
      DistSign(int ft, float in, posneg sg=pos) :
              Distance(ft, in)    //call base constructor
         { sign = sg; }           //set the sign

      void getdist()              //get length from user
         {
         Distance::getdist();     //call base getdist()
         char ch;                 //get sign from user
         cout << "Enter sign (+ or -): ";  cin >> ch;
         sign = (ch=='+') ? pos : neg;
         }
      void showdist() const       //display distance
         {
         cout << ( (sign==pos) ? "(+)" : "(-)" );  //show sign
         Distance::showdist();                     //ft and in
         }
   };
////////////////////////////////////////////////////////////////
int main()
   {
   DistSign alpha;                  //no-arg constructor
   alpha.getdist();                 //get alpha from user

   DistSign beta(11, 6.25);         //2-arg constructor

   DistSign gamma(100, 5.5, neg);   //3-arg constructor

                                    //display all distances
   cout << "\nalpha = ";  alpha.showdist();
   cout << "\nbeta = ";  beta.showdist();
   cout << "\ngamma = ";  gamma.showdist();
   cout << endl;
   return 0;
   }
```

Here the `DistSign` class adds the functionality to deal with signed numbers. The `Distance` class in this program is just the same as in previous programs, except that the data is protected. Actually in this case it could be private, because none of the derived-class functions accesses it. However, it's safer to make it protected so that a derived-class function could access it if necessary.