```
#include <stdio.h>

#include "Vector.h"

int main(int argc, char* argv[]) {
  Vector* pVec = NULL;

  try {
    while (1) {
      pVec = new Vector();

      printf("\nInput a vector: ");
      if ( pVec->read() ) break;

      // just print it out
      printf("\nVector: ");
      pVec->print();
      printf("\n");

      delete pVec;
      pVec = NULL;
    }
  }
  catch (char const* s) {
    printf("Exception: %s\n", s);
  }

  if ( pVec ) delete pVec;

  return 0;
}
```



```
#ifndef VECTOR_H
#define VECTOR_H

class Vector {
 private:
  int      capacity;      // allocated space, measured in elements

 protected:
  int      length;        // number of elements in vector
  double*  pVec;          // pointer to array of eleemnts

 private:
  void append(double x);  // adds an element to the end of the vector

 protected:
  int fill(int n);        // reads n data elements from stdin

 public:
  Vector();               // constructor -- doesn't take a length
  ~Vector();              // destructor -- called when object deleted

  virtual int  read();    // initializes from stdin; returns 0 for success
  virtual void print();   // prints to stdout
};

#endif // VECTOR_H
```

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

#include "Vector.h"
#include "Utility.h"

//-------------------------
// the constructor
//-------------------------
Vector::Vector() {
  length = 0;
  capacity = 0;
  pVec = NULL;
}

//-------------------------
// the destructor - called
// when delete is used on
// a Vector object
//-------------------------

Vector::~Vector() {
  if ( pVec == NULL ) return;
  free(pVec);
}

//-------------------------
// append() - protected
//   Add element to end.
//   We double capacity each
//   time that's necessary.
//-------------------------

void Vector::append(double x) {
  double* pNew;
  int     newCapacity;

  // an initial capacity of 1 is probably too small...
  if ( length == 0 ) newCapacity = 1;
  else newCapacity = capacity * 2;

  pNew = (double*)malloc( newCapacity * sizeof(double) );
  if ( !pNew ) throw "Out of memory";
  if ( capacity > 0 ) {
    bcopy(pVec, pNew, length*sizeof(double));
    free(pVec);
  }
```



```
  capacity = newCapacity;
  pVec = pNew;

  pVec[length++] = x;
}

//-------------------------
// fill() - reads data
//   elements from stdin
//-------------------------

int Vector::fill(int n) {
  for (int i=0; i<n; i++) {
    double x;
    if ( readDouble(x) == EOF ) return EOF;
    append(x);
  }

  return 0;
}


//-------------------------
// read from stdin
//   Return 0 for success.
//   Return non-zero for EOF
//-------------------------
int Vector::read() {
  int nElements;
  int rcode = scanf("%d", &nElements);
  if ( rcode == EOF ) return EOF;
  if ( rcode != 1 ) throw "Error reading vector length";
  if ( nElements < 0 ) throw "Illegal vector length";

  return fill(nElements);
}

//-------------------------
// print to stdout
//-------------------------
void Vector::print() {
  if ( length <= 0  ) return;
  for ( int i=0; i<length; i++) {
    printf(" %lf", pVec[i] );
  }
}
```

```
#include <stdio.h>

#include "Polynomial.h"
#include "Utility.h"

int main(int argc, char* argv[]) {
  Polynomial* pPoly = NULL;

  try {
    while (1) {
      pPoly = new Polynomial();

      printf("\nInput a polynomal: ");
      if ( pPoly->read() ) break;

      // just print it out
      printf("\nPolynomial: ");
      pPoly->print();
      printf("\n");

      while (1) {
        double x;
        printf("x: ");
        if ( readDouble(x) == EOF ) break;
        printf("%lf\n", pPoly->evaluate(x));
      }

      delete pPoly;
      pPoly = NULL;
    }
  }
  catch (char const* s) {
    printf("Exception: %s\n", s);
  }

  if ( pPoly ) delete pPoly;

  return 0;
}
```



```
#ifndef POLYNOMIAL_H
#define POLYNOMIAL_H

#include "Vector.h"

class Polynomial : public Vector {
 public:
  int    read();             // overrides parent's read()
  void   print();            // overrides parent's print()

  double evaluate(double x); // peculiar to this class
};

#endif //POLYNOMIAL_H
```

```
#include <stdio.h>

#include "Polynomial.h"

//---------------------------
// read() - takes a degree,
//  rather than a length
//---------------------------

int Polynomial::read() {
  int degree;
  int rcode = scanf("%d", &degree);
  if ( rcode == EOF ) return EOF;
  if ( rcode != 1 ) throw "Error reading polynomial degree";
  if ( degree < 0 ) throw "Illegal polynomial degree";

  return fill(degree+1);
}

//----------------------------
// print()
//----------------------------

void Polynomial::print() {
  int n;

  for ( n=0; n<length && pVec[n]==0; n++ ); // skip initial elements with 0 coefficient
  if ( n <= length-1 ) {
    if ( n == 0 ) printf(" %lf", pVec[n]);
    else          printf(" %lfx**%d", pVec[n], n);

    for ( n=n+1; n<=length-1; n++) {
      if ( pVec[n] != 0.0 ) printf(" + %lfx**%d", pVec[n], n);
    }
  }
  else {
    printf("0");
  }
}

//----------------------------
// evaluate()
//----------------------------

double Polynomial::evaluate(double x) {
  int n;
  double xFactor = 1.0;
  double result = pVec[0];
```



```
  for ( n=1; n<length; n++ ) {
    xFactor *= x;
    result += pVec[n]*xFactor;
  }

  return result;
}
```

```
#ifndef UTILITY_H
#define UTILITY_H

// utility methods to help make the code a little shorter

extern int readInt(int& n);       // returns 0 for success; EOF, or non-zero
extern int readDouble(double& x); // returns 0 for success; EOF, or non-zero

#endif // UTILITY_H
```



```
#include <stdio.h>

#include "Utility.h"

//-----------------------------------------------------
// readInt
//  Takes a reference parameter (a pointer, but without
//   the pointer syntax).
//  Returns: 0 - normal or EOF - eof
//  Throws: char const* error message
//-----------------------------------------------------

int readInt(int& n) {
  int rcode = scanf("%d", &n);
  if ( rcode == EOF ) return EOF;
  if ( rcode != 1 ) throw "Some error reading integer";
  return 0;
}

//-----------------------------------------------------
// readDouble
//  Takes a reference parameter (a pointer, but without
//   the pointer syntax).
//  Returns: 0 - normal or EOF - eof
//  Throws: char const* error message
//-----------------------------------------------------

int readDouble(double& x) {
  int rcode = scanf("%lf", &x);
  if ( rcode == EOF ) return EOF;
  if ( rcode != 1 ) throw "Some error reading double";
  return 0;
}
```

```
  $ nm Polynomial.o >polySymbols.txt

00000000 T _ZN10Polynomial4readEv
000000a6 T _ZN10Polynomial5printEv
000001c2 T _ZN10Polynomial8evaluateEd
         U _ZN6Vector4fillEi
00000000 V _ZTI10Polynomial
         U _ZTI6Vector
         U _ZTIPKc
00000000 V _ZTS10Polynomial
00000000 V _ZTV10Polynomial
         U _ZTVN10__cxxabiv120__si_class_type_infoE
         U __cxa_allocate_exception
         U __cxa_throw
         U __gxx_personality_v0
         U printf
         U putchar
         U scanf
```