(a) The extended class `TridiagonalMatrix` is given below. It includes a few operations that you were not asked to provide.

```
template<class T>
class TridiagonalMatrix {
   friend ostream& operator<<
          (ostream&, const TridiagonalMatrix<T>&);
   friend istream& operator>>
          (istream&, TridiagonalMatrix<T>&);
   public:
      TridiagonalMatrix(int size = 10)
         {n = size; t = new T [3*n-2];}
      ~TridiagonalMatrix() {delete [] t;}
      TridiagonalMatrix<T>& Store
              (const T& x, int i, int j);
      T Retrieve(int i, int j) const;
      TridiagonalMatrix(const TridiagonalMatrix<T>& x);
         // copy constructor
      TridiagonalMatrix<T>&
         operator=(const TridiagonalMatrix<T>& x);
      TridiagonalMatrix<T> operator+() const; // unary +
      TridiagonalMatrix<T>
         operator+(const TridiagonalMatrix<T>& x) const;
      TridiagonalMatrix<T> operator-() const; // unary minus
      TridiagonalMatrix<T>
         operator-(const TridiagonalMatrix<T>& x) const;
      TridiagonalMatrix<T>& operator+=(const T& x);
      TridiagonalMatrix<T> Transpose();
   private:
      int n; // matrix dimension
      T *t;  // 1D array for tridiagonal
};
```

```
template<class T>
TridiagonalMatrix<T>::TridiagonalMatrix
                     (const TridiagonalMatrix<T>& x)
{// Copy constructor for tridiagonal matrices.
   n = x.n;
   t = new T[3 * n - 2];               // get space
   for (int i = 0; i < 3 * n - 2; i++) // copy elements
      t[i] = x.t[i];
}


template<class T>
TridiagonalMatrix<T>& TridiagonalMatrix<T>::
          operator=(const TridiagonalMatrix<T>& x)
{// Overload assignment operator.
   if (this != &x) {// not self-assignment
      n = x.n;
      delete [] t; // free old space
      t = new T[3 * n - 2];      // get right amount
      for (int i = 0; i < 3 * n - 2; i++) // copy elements
         t[i] = x.t[i];
      }
   return *this;
}

template<class T>
TridiagonalMatrix<T> TridiagonalMatrix<T>::
          operator+(const TridiagonalMatrix<T>& x) const
{// Return w = (*this) + x.
   if (n != x.n) throw SizeMismatch();

   // create result array w
   TridiagonalMatrix<T> w(n);
   for (int i = 0; i < 3 * n - 2; i++)
       w.t[i] = t[i] + x.t[i];

   return w;
}
```

```
template<class T>
TridiagonalMatrix<T> TridiagonalMatrix<T>::
          operator-(const TridiagonalMatrix<T>& x) const
{// Return w = (*this) - x.
   if (n != x.n) throw SizeMismatch();

   // create result array w
   TridiagonalMatrix<T> w(n);
   for (int i = 0; i < 3 * n - 2; i++)
       w.t[i] = t[i] - x.t[i];

   return w;
}

template<class T>
TridiagonalMatrix<T> TridiagonalMatrix<T>::
          operator-() const
{// Return w = -(*this).
   // create result array w
   TridiagonalMatrix<T> w(n);
   for (int i = 0; i < 3 * n - 2; i++)
       w.t[i] = -t[i];

   return w;
}

template<class T>
TridiagonalMatrix<T>& TridiagonalMatrix<T>::
          operator+=(const T& x)
{// Add x to each element of (*this).
   for (int i = 0; i < 3 * n - 2; i++)
       t[i] += x;
   return *this;
   }
```

```
template<class T>
TridiagonalMatrix<T> TridiagonalMatrix<T>::
          Transpose()
{// Compute the transpose of *this.

   // create result array w
   TridiagonalMatrix<T> w(n);
   // copy lower diagonal of *this to
   // upper diagonal of w and upper of
   // *this to lower of w
   for (int i = 0; i < n - 1; i++) {
       w.t[2 * n - 1 + i] = t[i];
       w.t[i] = t[2 * n - 1 + i];
       }

   // copy main diagonal of *this to
   // main diagonal of w
   for (int i = n - 1; i < 2 * n - 1; i++)
       w.t[i] = t[i];

   return w;
}
```

```
template<class T>
ostream& operator<<(ostream& out,
                    const TridiagonalMatrix<T>& x)
{// Put the elements of x into the stream out.
   out << "Lower diagonal is" << endl;
   for (int i = 0; i < x.n - 1; i++)
      out << x.t[i] << "  ";
   out << endl;

   out << "Main diagonal is" << endl;
   for (int i = x.n - 1; i < 2 * x.n - 1; i++)
      out << x.t[i] << "  ";
   out << endl;

   out << "Upper diagonal is" << endl;
   for (int i = 2 * x.n - 1; i < 3 * x.n - 2; i++)
      out << x.t[i] << "  ";
   out << endl;

   return out;
}
```

```
// overload >>
template<class T>
istream& operator>>(istream& in,
                    TridiagonalMatrix<T>& x)
{// Input the tridiagonal matrix.

   cout << "Enter number of rows"
        << endl;
   in >> x.n;
   if (x.n < 0) throw BadInput();

   // input terms
   cout << "Enter lower diagonal" << endl;
   for (int i = 0; i < x.n - 1; i++)
      in >> x.t[i];

   cout << "Enter main diagonal" << endl;
   for (int i = x.n - 1; i < 2 * x.n - 1; i++)
      in >> x.t[i];

   cout << "Enter upper diagonal" << endl;
   for (int i = 2 * x.n - 1; i < 3 * x.n - 2; i++)
      in >> x.t[i];

   return in;
}
```

(b) The codes are in the files `tridiag.*`.

(c) The complexity of each new function is $\Theta(n)$.