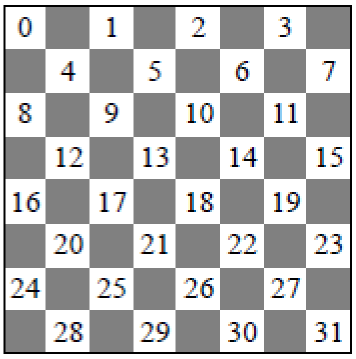
Array in ChessBoard form

Implement the ChessBoardArray class to represent square arrays in which non-zero elements can only be found in the positions that will correspond on a chessboard; as in white squares (see figure). For convenience, we will consider these tables to contain integers and that the default constructor constructs arrays that have zeros everywhere.



Your class will have to support the following functions, which you have to implement:

```
class ChessBoardArray {
protected:
  class Row {
  public:
    Row(ChessBoardArray &a, int i);
    int & operator [] (int i) const;
  };

  class ConstRow {
  public:
    ConstRow(const ChessBoardArray &a, int i);
    int operator [] (int i) const;
  };

public:
  ChessBoardArray(unsigned size = 0, unsigned base = 0);
  ChessBoardArray(const ChessBoardArray &a);
  ~ChessBoardArray();

  ChessBoardArray & operator = (const ChessBoardArray &a); {

  int & select(int i, int j);
  int select(int i, int j) const;

  const Row operator [] (int i);
  const ConstRow operator [] (int i) const;

  friend ostream & operator << (ostream &out, const ChessBoardArray &a);
};
```

Define and use a private method loc:

```
unsigned int loc(int i, int j) const throw(out_of_range);
```

The tables should also print the zero data as shown below. Use setw (4) to align the numbers (defined in <iomanip>, look it up!).

```
  0   0   1   0   2   0   3   0
  0   4   0   5   0   6   0   7
  8   0   9   0  10   0  11   0
  0  12   0  13   0  14   0  15
 16   0  17   0  18   0  19   0
  0  20   0  21   0  22   0  23
 24   0  25   0  26   0  27   0
  0  28   0  29   0  30   0  31
```

You can test your implementation as shown below:

```
int main() {
  ChessBoardArray a(4, 1);  // size 4x4, rows and columns numbered from 1
  a[3][1] = 42;
  a[4][4] = 17;
  try { a[2][1] = 7; }
  catch(out_of_range &e) { cout << "a[2][1] is black" << endl; }
  try { cout << a[1][2] << endl; }
  catch(out_of_range &e) { cout << "and so is a[1][2]" << endl; }
  cout << a;
}
```

Its execution should display:

```
a[2][1] is black
and so is a[1][2]
  0   0   0   0
  0   0   0   0
 42   0   0   0
  0   0   0  17
```

Polynomials in the form of a sorted linked list

Implement the Polynomial class to represent polynomials of a variable (let it be called x), with integer coefficients. Polynomials should be represented in the form of sorted simply linked list: its nodes should be sorted in descending order turn of the exhibitor. There should not be two nodes with the same exponent and there should not be nodes with zero coefficients.

Your class must support the following functions, which you must implement:

```
class Polynomial {
 protected:
  class Term {
   protected:
    int exponent;
    int coefficient;
    Term *next;
    Term(int exp, int coeff, Term *n);
    friend class Polynomial;
  };

 public:
  Polynomial();
  Polynomial(const Polynomial &p);
  ~Polynomial();

  Polynomial & operator = (const Polynomial &p) {

  void addTerm(int expon, int coeff);
  double evaluate(double x);

  friend Polynomial operator+ (const Polynomial &p, const Polynomial &q);
  friend Polynomial operator* (const Polynomial &p, const Polynomial &q);

  friend ostream & operator << (ostream &out, const Polynomial &p);
};
```

The polynomial print must display them as in the following examples:

```
x^2 + 3x - 1
2x^5 + 1
- x^7 - 3x^3 - x
42
0
```

You can test your implementation as shown below:

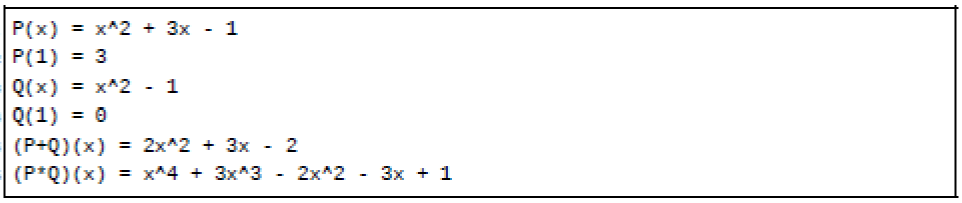
```
int main() {
  Polynomial p;      // 0
  p.addTerm(1, 3);   // 3x
```

```
  p.addTerm(2, 1);   // x^2
  p.addTerm(0, -1);  // -1

  Polynomial q(p);   // x^2 + 3x - 1
  q.addTerm(1, -3);  // -3x

  cout << "P(x) = " << p << endl;
  cout << "P(1) = " << p.evaluate(1) << endl;
  cout << "Q(x) = " << q << endl;
  cout << "Q(1) = " << q.evaluate(1) << endl;
  cout << "(P+Q)(x) = " << p+q << endl;
  cout << "(P*Q)(x) = " << p*q << endl;
}
```

Its execution should display:



```
P(x) = x^2 + 3x - 1
P(1) = 3
Q(x) = x^2 - 1
Q(1) = 0
(P+Q)(x) = 2x^2 + 3x - 2
(P*Q)(x) = x^4 + 3x^3 - 2x^2 - 3x + 1
```