# The Problem

- In chapter 2, we have covered "sparse polynomials" and "sparse matrices" as examples for the applications of arrays. Here we want to combine the two.
- Consider polynomials with two variables, $p(x,y)$. The complexity difference between the non-sparse and sparse representations are even bigger here than for one-variable polynomials.
  - Just for thinking: Try to extend the three representations of polynomials mentioned in the class to two-variable cases. What are the space and time complexities? How about polynomials of more variables? You can give your thoughts on the discussion board for class participation credits.

# The Tasks

- Now, for a two-variable polynomial, we can use a sparse matrix to represent its coefficients.
- For the `MatrixTerm` class, change the type of `value` to `double`.
- For the `SparseMatrix` class, you can start from the definition code in textbook. However, you need to add this function member:

```
void SetTerms(const int *row, const int
   *column, const double *value, int n);
```

  - This function sets all the terms in the matrix; the inputs `row`, `column`, and `value` are all arrays of size `n`.
  - You should ensure that the input row/column pairs are in the correct order.

# The Tasks

- For the **SparseMatrix** class, you also need to implement:
  - A default constructor to initialize the data members;
  - Overloaded operator **=** (assignment) for "deep-copying" the data (to be explained in class);
  - A copy constructor that uses the assignment operator;
  - A destructor that cleans up the data;
  - Function member **Add**; its implementation should be a direct extension of the textbook code that adds two polynomials.
    - Note: It's better to pass the argument by reference.

```
class A
{
      int *a;
int n;
}
A::A(int m)
{
      a=mew int[m];
n=m;
}
A::~A(){delete [] a;}
```

```
A& operator = (const A&b)
{
      n=b.n;
a=new int[n];
memcpy(a,b.a,sizeof);
return *this;
}
A::A(const A&b)
{
      *this=b;
}
```

避免default的copy assignment是將原資料原封的複製
使用時會用到同一塊位置

# The Tasks

- Derive a C++ class `SparsePoly2D` from `SparseMatrix`.
- Implement the following for this class:

```
void SetTerms(const int *x_exp, const int
   *y_exp, const double *coef, int n);

SparsePoly2D Add(const SparsePoly2D &p);

double Eval(double x, double y);

void Print();
```

  - The first two can be easily handled by calling the corresponding parent-class functions.
  - The `Print` function should output the polynomial in `<x_exp,y_exp,coef>` triples. For example, for $x^3-2x^2y+15$, the output should look like `<3,0,1><2,1,-2><0,0,15>`.

# The Guidelines

- Allowed environments: VS2012/2013/2015, Dev-C++. Indicate your environment at the beginning of your code.
- You need to write your own `main` function to test your code. You do not need to include this `main` function in your submission. Your class template should be packaged like a reusable module. The instructor will provide a test `main` function for you next week.
- No usage of STL class templates allowed.
- Include documentation; this will be part of your grade.
- Demo: Only a randomly selected subset of students; will be announced separately after the due date.

# The Guidelines

- Submission:
  - Use E3 only.
  - Submit all your code in a single header file (`.h`). Name it `P1_xxxxxx.h`, where `xxxxxx` is your ID. **Do not** submit your `main` function or any file that is not your code (such as the *.sln file). No compressed file (*.zip, *.rar, etc.). **Only the header file!!!**
  - Due date: **10/16/2014**. There's a grace period of 4 days with 10% deduction per day. (The deduction kicks in only when you have accumulated more than three days of delay during the semester.)