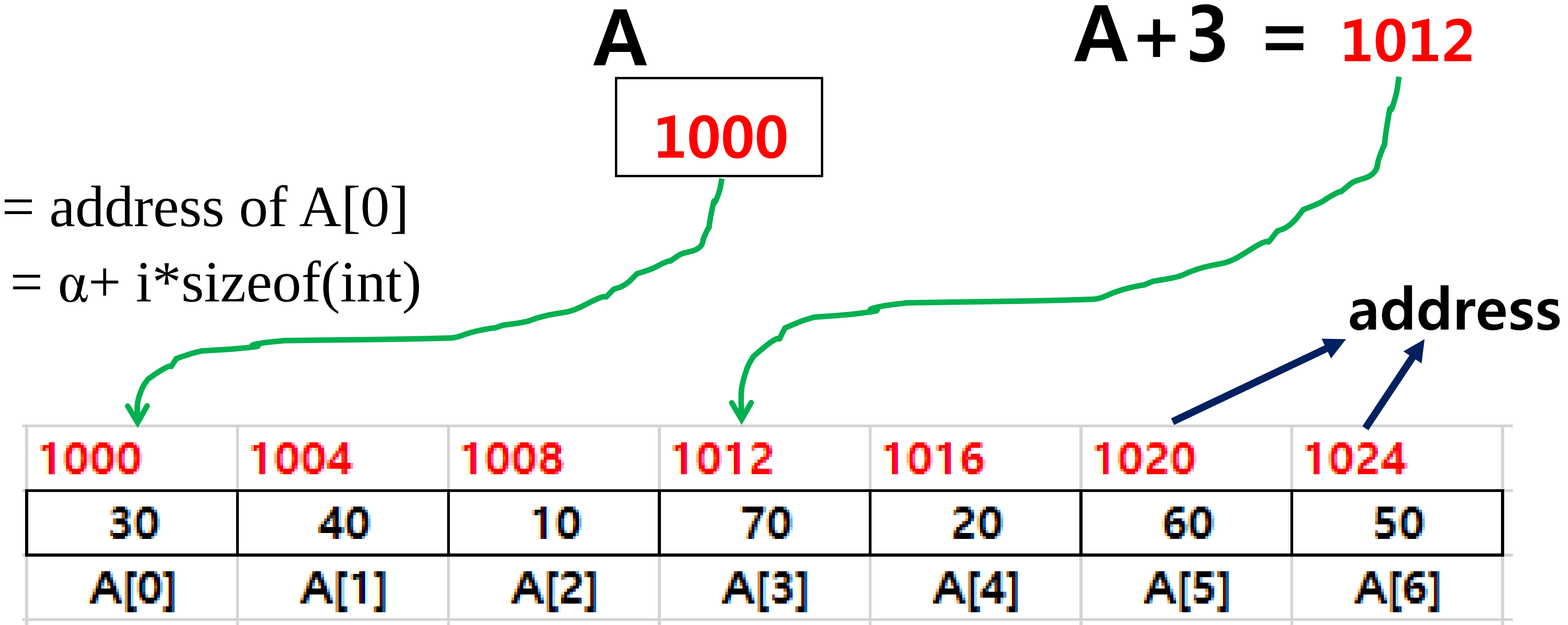
Chap 2. Arrays and Structures

Arrays in C

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
30	40	10	70	20	60	50

- int A[7];
- A[0] = 30; ….
- A:
  - name of array
  - pointer to A[0]
  - *A = A[0]
  - base address α = address of A[0]
  - address of A[i] = α+ i*sizeof(int)
- A+i
  - pointer to A[i]
  - A+i = &A[i]
  - *(A+i) = A[i]



Dynamic memory allocation of Array

- Static memory allocation

```
int A[7];
A[0] = 30; A[1] = 40; ….
```

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
30	40	10	70	20	60	50

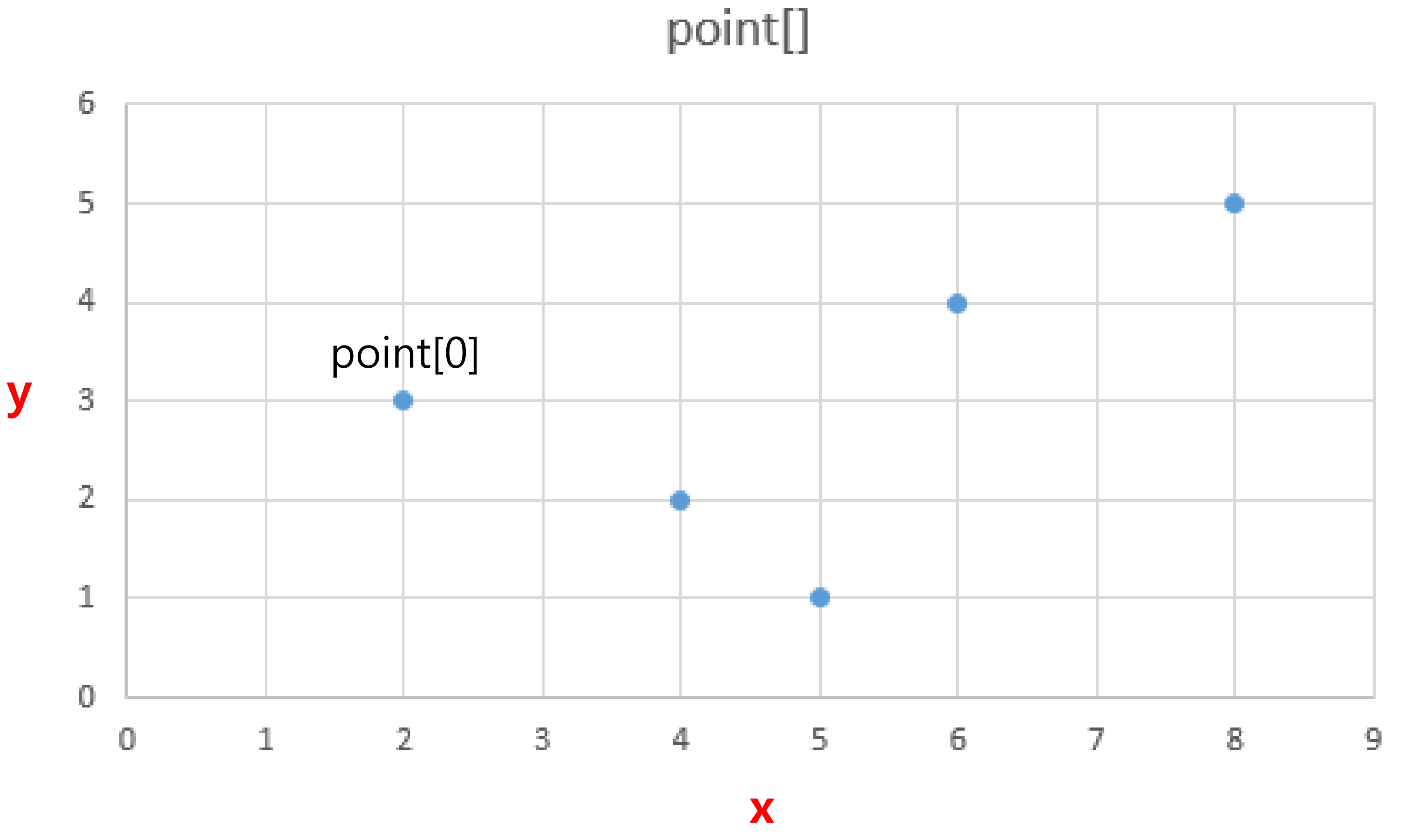
- Dynamic memory allocation
  - Size of array is variable
  - Avoid waste of space

```
int *A;
A = (int *) malloc(7*sizeof(int));
A[0] = 30; A[1] = 40; …
```

Array and structure

```
typedef struct {
        int x;
        int y;
} POINT;
POINT point[5];

point[0].x = 2;
point[0].y = 3;
……
```



Polynomial

- Example
  - $A(x) = 2x^5 - 3x^3 + 7x^2 + 10x - 1$
- Term
  - Coefficient
  - Exponent
  - Polynomial
    - Collection of terms
    - Collection of <coefficient, exponent> pairs
- Operations on polynomials
  - Add
  - Multiply
  - etc.
- Polynomial representations
  - Array of coefficients with degree (max. exponent)
  - Array of terms

Array of coefficients with degree

- Example
  - $A(x) = 2x^5 - 3x^3 + 7x^2 + 10x - 1$
- Degree
  - max. exponent
  - degree of A(x) = 5
- Array of coefficients
  - $A(x) = 2x^5 - 3x^3 + 7x^2 + 10x - 1$
    $= 2x^5 + 0x^4 - 3x^3 + 7x^2 + 10x^1 – 1x^0$
  - Values of coefficient array =
    - increasing order {-1, 10, 7, -3, 0, 2} or
    - decreasing order {2, 0, -3, 7, 10, -1}
- Polynomial
  - Degree
  - Array of coefficients

degree	coef[0]	coef[1]	coef[2]	coef[3]	coef[4]	coef[5]
5	-1	10	7	-3	0	2

Array of coefficients with degree

- Assumption
  - Coefficient의 data type: integer
  - Maximum degree to be supported = X
- Type definition 1: static allocation of coefficient array

```
typedef struct {
    int degree;
    int coef[X+1];
} Polynomial;
```

- Type definition 2: dynamic allocation of coefficient array

```
typedef struct {
    int degree;
    int *coef;
} Polynomial;
```

Static allocation of coefficient array

- Example
  - $A(x) = 2x^5 - 3x^3 + 7x^2 + 10x - 1$
- Representation

```
#define MAX_DEGREE 100
typedef struct {
      int degree;
      int coef[MAX_DEGREE +1];
} Polynomial;

Polynomial A;

A.degree = 5;
A.coef[5] = 2; A.coef[4] = 0; ..., A.coef[0] = -1;
```

Dynamic allocation of coefficient array

- Example
  - $A(x) = 2x^5 - 3x^3 + 7x^2 + 10x - 1$
- Representation

```
typedef struct {
    int degree;
    int *coef;
} Polynomial;

Polynomial A;

A.degree = 5;
A.coef = (int *) malloc((A.degree+1)*sizeof(int));
A.coef[5] = 2; A.coef[4] = 0; ..., A.coef[0] = -1;
```

Array of terms

- coefficient array
  - Inefficient for sparse polynomials
  - Waste of memory
  - Example: $A(x) = 2x^{500} + 10x - 1$
- Array of terms

```
typedef struct {
      int coef;
      int exp;
} Term;
Term A[3];

A[0].coef = 2; A[0].exp = 500;
A[1].coef = 10; A[1].exp = 1;
A[2].coef = -1; A[2].exp = 0;
```

	A[0]	A[1]	A[2]
coef	2	10	-1
exp	500	1	0

Array of terms

- Type definition 1: static allocation of terms array

```
typedef struct {
    int coef;
    int exp;
} Term;
typedef struct {
    int num_terms;
    Term term[MAX_TERMS];
} Polynomial;
```

- Type definition 2: dynamic allocation of terms array

```
typedef struct {
    int num_terms;
    Term *term;
} Polynomial;
```

Polynomial addition

- Example
  - $A(x) = 2x^5 - 3x^3 + 7x^2 + 10x – 1$
  - $B(x) = 5x^4 +2x^3 - 7x^2 + 10$
  - $C(x) = A(x) + B(x)$
    $= 2x^5 + 5x^4 - x^3 + 10x + 9$
- Implementation
  - Dependent on polynomial representation
  - Representations
    - Array of coefficients
    - Array of terms

Polynomial addition: Array of coefficients

```
#define MAX_DEGREE 10
typedef struct {
    int degree;
    int coef[MAX_DEGREE +1];
} Polynomial;

Polynomial A, B, C; //C = A + B

A.degree ← 5;
A.coef[] ← {-1, 10, 7, -3, 0, 2};
B.degree ← 4;
A.coef[] ← {10, 0, -7, 2, 5};

Poly_add(&A, &B, &C);
```

$$A(x) = 2x^5 - 3x^3 + 7x^2 + 10x - 1$$
$$B(x) = 5x^4 + 2x^3 - 7x^2 + 10$$
$$C(x) = 2x^5 + 5x^4 - x^3 + 10x + 9$$

C.degree = 5
C.coef[] = {9, 10, 0, -1, 5, 2}

Polynomial addition: Array of terms

Polynomial A, B, C; //C = A + B

……

Poly_add(&A, &B, &C);

	A[0]	A[1]	A[2]	A[3]	A[4]
coef	2	-3	7	10	-1
exp	5	3	2	1	0

	B[0]	B[1]	B[2]	B[3]
coef	5	2	-7	10
exp	4	3	2	0

	C[0]	C[1]	C[2]	C[3]	C[4]
coef	2	5	-1	10	9
exp	5	4	3	1	0

Matrix

- Row, column, element
- Example: 2 by 3 Matrices

$$\mathbf{A}\ \begin{bmatrix} 5 & 10 & 0 \\ -1 & 2 & 9 \end{bmatrix} \qquad \mathbf{B}\ \begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

- Matrix representation
  - 2-dim array

  ```
  int A[2][3]; //A[num_rows][num_cols]
  A[0][1] = 10;
  ```
- Sparse matrix
  - 2-dim array is inefficient

Representation of sparse matrix

- Array of
  - non-zero elements
  - triples <row, col, value>
- Example: matrix B

  <0, 0, 5>
  <1, 1, 2>

$$B \begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

	row	col	value
B[0]	2	3	2
B[1]	0	0	5
B[2]	1	1	2

- Declaration

```
typedef struct {
    int row,
    int col;
    int value;
} Element;
Element B[MAX_ELEM+1];
```

Representation of sparse matrix

```
typedef struct {
        int row,
        int col;
        int value;
} Element;
Element B[MAX_ELEM+1];
```

Max number of non-zero elements to be supported

Number of non-zero elements

Number of rows

Number of columns

$$\mathbf{B}\begin{bmatrix}5 & 0 & 0\\ 0 & 2 & 0\end{bmatrix}$$

	row	col	value
B[0]	2	3	2
B[1]	0	0	5
B[2]	1	1	2

Sorting of non-zero elements

$$\mathbf{A}\begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \\ 7 & 0 & 4 \end{bmatrix}$$

	row	col	value
A[0]	4	3	5
A[1]	0	0	5
A[2]	1	1	2
A[3]	2	1	1
A[4]	3	0	7
A[5]	3	2	4

- Sorted on row first
- Sorted on col for the same rows

Transpose a matrix

- Interchange rows and columns
- Example:

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \\ 7 & 0 & 4 \end{bmatrix} \qquad \begin{bmatrix} 5 & 0 & 0 & 7 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

Transpose a matrix: 2-dim array

- Interchange rows and columns
  - A: input matrix
  - B: transposed matrix
  - A[i][j] → B[j][i]
- A simple nested loop: n x m matrix

```
for i ← 1 to n do //row:1..n
    for j ← 1 to m do //col:1..m
        B[j][i] ← A[i][j]
```

- Complexity: O(numCols * numRows)

Transpose a sparse matrix

- Representation: 1-dim array of non-zero elements
- Array A: original matrix
- Array B: transposed matrix

$$A \begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \\ 7 & 0 & 4 \end{bmatrix} \quad B \begin{bmatrix} 5 & 0 & 0 & 7 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

	row	col	value
A[0]	4	3	5
A[1]	0	0	5
A[2]	1	1	2
A[3]	2	1	1
A[4]	3	0	7
A[5]	3	2	4

	row	col	value
B[0]	3	4	5
B[1]	0	0	5
B[2]	0	3	7
B[3]	1	1	2
B[4]	1	2	1
B[5]	2	3	4

Algorithm 1

- For each column of A, makes rows of B
- For each column of A, the array A is fully scanned
- Example: for column 0,

	row	col	value
A[0]	4	3	5
A[1]	0	0	5
A[2]	1	1	2
A[3]	2	1	1
A[4]	3	0	7
A[5]	3	2	4

	row	col	value
B[0]	3	4	5
B[1]	0	0	5
B[2]	0	3	7

→ Next non-zero element will be put here

Algorithm 1 (cont'd)

- Example: for the next column, i.e., column 1,

	row	col	value
A[0]	4	3	5
A[1]	0	0	5
A[2]	1	1	2
A[3]	2	1	1
A[4]	3	0	7
A[5]	3	2	4

	row	col	value
B[0]	3	4	5
B[1]	0	0	5
B[2]	0	3	7
B[3]	1	1	2
B[4]	1	2	1
Next non-zero element will be put here			

- Complexity: O(numCols * numNonZeroElements)

Algorithm 2

- Faster transpose of a sparse matrix than Algo 1
- Step 1: For each column of A, count non-zero elements
- Step 2: For each row of B, compute the start position in array B
- Step 3: For each element in array A, move it to array B
- Complexity: O(numCols + numNonZeroElements)

Step 1

- For each column of A, count non-zero elements

	row	col	value
A[0]	4	3	5
A[1]	0	0	5
A[2]	1	1	2
A[3]	2	1	1
A[4]	3	0	7
A[5]	3	2	4

count[0]	count[1]	count[2]	
0	0	0	initialization
1	0	0	
1	1	0	
1	2	0	
2	2	0	
2	2	1	final counts

Step 2

- For each row of B, compute the start position in array B

count[0]	count[1]	count[2]
2	2	1

startPos[0]	startPos[1]	startPos[2]	
1			initialization
	1+2=3	3+2=5	
1	3	5	

Step 3

- For each element in array A, move it to array B

	row	col	value
A[0]	4	3	5
A[1]	0	0	5
A[2]	1	1	2
A[3]	2	1	1
A[4]	3	0	7
A[5]	3	2	4

	row	col	value
B[0]	3	4	5
B[1]	0	0	5
B[2]			
B[3]	1	1	2
B[4]			
B[5]			

- Where in Array B ??
  - startPos[]

startPos[0]	startPos[1]	startPos[2]
1	3	5
2	4	5