배열과 구조

* 자료구조의 구분
- 선형(linear) 과 비선형(non-linear)으로 구분한다.
- 자료구조내의 요소들이 순차적인 형식이면 선형, 아니면 비선형.
- 선형자료구조에는 배열과 연결 리스트가 있다.
- 배열: 순차적 메모리 주소에 의하여 요소들간의 관계를 표현하는 구조
- 연결리스트: 포인터를 사용하여 요소들간의 관계를 표현
- 비선형 자료구조: 트리(tree) 와 그래프(graph)

1. 배열의 특징
- 연속적 기억장소의 집합
- 대부분의 언어에서 제공하는 가장 단순한 구조적 자료형
- 동일한 자료형 (Same data type for elements)
- 선언시 크기지정. 크기보다 많은양의 자료 저장 => overflow
- 정적 자료형 (compile 시 크기를 알아야 하고, 실행 되는 동안 크기가 변하지 않는다)
- Set of mappings between index and values ; **<index, value>**

장점: 이해 쉽고, 사용하기 편함, 자료저장이 용이(예: A[4]=10)

단점: 동일한 자료만 저장, 미리 크기 선언(필요이상 크기 선언시, 공간낭비, 많은 자료이동으로 삽입 삭제 느림.

Structure *Array*

Objects: index의 각 값에 대하여 집합 item에 속한 한 값이 존재하는 <index, value>쌍의 집합. index는 일차원/다차원의 유한 순서 집합.

Functions: **모든 A ∈*Array*, *i* ∈ *index*, *x* ∈ *item*, *j*, *size* ∈ *integer***

***Array* Create(j, list)**::= return j차원의 배열. list는 i번째 원소가 i번째 차원의 크기인 j-tuple이며 item들은 정의되지 않았음.

***Item* Retrieve(A, i)::=** if (*i* ∈ *index*)

return 배열 A의 인덱스 i 값과 관련된 항목.

else return 에러.

***Array* Store(A, i, x)::=** if (*i* ∈ *index*)

return 새로운 쌍<*i*, *x*>가 삽입된 배열 A.

else return 에러.

end *Array*

- **배열의 연산**
  - i. **length *n***
  - ii. **reading (*R* => *L*, *L* => *R*)**
  - iii. **retrieve i^{th} element, 0≤i<n**
  - iv. **update i^{th} element's value, 0≤i<n**
  - v. **insertion (*i 번째 위치, 0≤i≤n*)**
  - vi. **deletion (*i 번째 항목, 0≤i<n*)**

- 배열 프로그램의 예 (ex. C언어)

```
#define MAX_SIZE 100
  float sum(float [], int);
  float input[MAX_SIZE], answer;
  int i;

 void main(void) {
 for (i = 0; i < MAX_SIZE; i++)
       input[i] = i;
       answer = sum(input, MAX_SIZE);
       printf("The sum is: %f\n", answer);
 }

 float sum(float list[], int n) {
  int i;
  float    tempsum = 0;

  for (i = 0; i < n; i++)      tempsum += list[i];
  return tempsum;
 }
```

2. 순서 리스트의 표현

* 메모리(기억장소) 표현

1) 순차 사상(sequential mapping)

   - 물리적 접근성 (arrays)

2) 비순차 사상(non-sequential mapping)

   - 비연속적 기억장소 위치 (링크드 리스트)

– 기호 다항식의 조작

$A(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x^1+a_0x^0$

ax^e a : *coefficient* $a_n >= 0$

e : *exponent* – **unique**

x : *variable x*

- 차수(*degree*): 다항식에서 가장 큰 지수

* 다항식의 표현 (I): 지수들을 내림차순으로 정돈

$A = (n, a_n, a_{n-1}, \ldots, a_1, a_0)$

degree of A *n+1 coefficients*

```
typedef struct {
        int expon;
        float coef[max_degree];
} polynomial;
```

a : polynomial, n < MAX_DEGREE

$a.expon = n$

$a.coef[i] = a_{n-i}, 0 \le i \le n$

예) $3x^4 + 5x^2 + 6x + 4$ 의 경우 A = (4, 3, 0, 5, 6, 4)

expon	**4**										
coef	0	0	0	0	0	0	**3**	**0**	**5**	**6**	**4**
	10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

문제점) $A(x) = x^{1000} + 1$: n = 1000

A = (1000, 1, 0,..., 0, 1) : 1002 elements

999

* 다항식의 표현 (II)

$A(x) = b_{m-1}x^{em-1} + b_{m-2}x^{em-2} + \ ... \ + b_0x^{e0}$

Where $b_i \neq 0$, $0 \leq i \leq m-1$, $e_{m-1} > e_{m-2} > ... > e_o \geq 0$

$A = (m, e_{m-1}, b_{m-1}, e_{m-2}, b_{m-2}, ... , e_0, b_0)$

no. of non-zero terms

예)

$A(x) = x^4+10x^3+3x^2+1$ A = (4, 4, 1, 3, 10, 2, 3, 0, 1)

$A(x) = x^{1000} +1$ A = (2, 1000, 1, 0, 1)

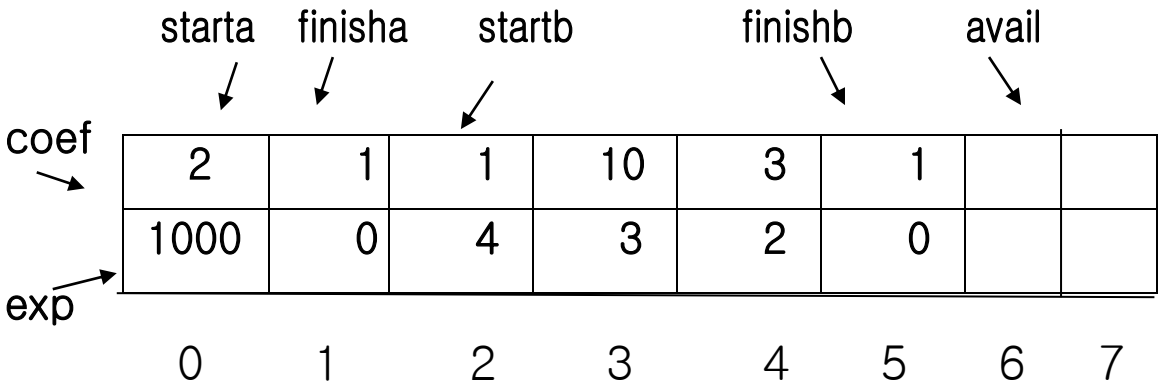
```
MAX_TERMS 100        /* 항 배열의 크기 */
        typedef struct {
                float coef;
                int expon;
                } Polynomial;
        Polynomial   terms[MAX_TERMS];
```

coef	3	5	6	4
expon	4	2	1	0

ex) 두개의 다항식 A(x), B(x)

$A(x) = 2x^{1000} + 1$

$B(x) = x^4 + 10x^3 + 3x^2 + 1$



startA = 0, finishA = 1, startB = 2, finishB = 5, Avail = 6

A(x) : <*starta*, *finisha*> B(x): <startb, finish>

다항식 덧셈 D = A + B

```
void padd (int starta, int finisha, int startb, int finishb, int *startd,
int *finishd);

{  /* A(x) 와 B(x)를 더하여 D(x)를 생성한다 */
   float coefficient;          *startd = avail;

while (starta <= finisha && startb <= finishb)

    switch(COMPARE(terms[starta].expon, terms[startb].expon))
    {
       case -1: /* a의 expon이 b의 expon보다 작은 경우 */
                attach(terms[startb].coef, terms[startb].expon);
                startb++;   break;
```

```
        case 0: /* 지수가 같은 경우 */
                coefficient= terms[starta].coef + terms[startb].coef;
                if(coefficient)
                     attach(coefficient, terms[starta].expon);
                    starta++;   startb++; break;

       case 1: /*   a의 expon이 b의 expon보다 큰 경우 */
                attach(terms[starta].coef, terms[starta].expon);
                starta++;
     }

 /* A(x)의 나머지 항들을 첨가한다 */
    for(; starta <= finisha; starta++)
          attach(terms[starta].coef, terms[starta].expon);

 /* B(x)의 나머지 항들을 첨가한다 */
   for(; startb <= finishb; startb++)
          attach(terms[startb].coef,terms[startb].expon);
   *finishd = avail-1;      }


void attach(float coefficient, int exponent)
{ /* 새 항을 다항식에 첨가한다. */
     if (avail >= MAX_TERMS) {
          fprintf(stderr, "다항식에 항이 너무 많다.");
           exit(1);
      }
     terms[avail++].coef = coefficient;
     terms[avail++].expon = exponent;
}
```

- 알고리즘 padd 의 분석 :
  - m, n (>0): 각각 A와 B의 0이 아닌 항의 수
  - while 루프
    . 각 반복마다 starta, startb 또는 둘 다 값이 증가
    . 반복 종료 -> *starta* <= *finisha && startb* <= *finishb*
    . 반복 횟수 ≤ m+n-1
    ∴ 연산시간 = *O*(n+m)

- **문제점**
  avail = *MAX_TERMS* ?

⇨ 불필요한 다항식 제거후
배열 끝에 연속적인 가용공간 생성 - 데이타 이동시간

3. 희소행렬 (Sparse Matrix)

1) 개요

m × *n matrix A* ≡ *A[MAX_ROWS][MAX_COLS]*

[number of non-zero elements / total elements] << small

- 효율적 기억장소 사상
  ⅰ) <i, j, value> : 3-tuples (triples)
  ⅱ) no. of rows
  ⅲ) no. of columns
  ⅳ) no. of non-zero elements
  ⅴ) ordering (column major or row major)

• *sparse matrix (row major)*

	0	1	2	3	4	5
0	15	0	0	22	0	-15
1	0	11	3	0	0	0
2	0	0	0	-6	0	0
3	0	0	0	0	0	0
4	91	0	0	0	0	0
5	0	0	28	0	0	0

(row) (col) (value)

6	6	8
0	0	15
0	3	22
0	5	-15
1	1	11
1	2	3
2	3	-6
4	0	91
5	2	28

Sparse Matrix ‘a’

• *sparse matrix (column major)*

(row) (col) (value)

6	6	8
0	0	15
0	4	91
1	1	11
2	1	3
2	5	28
3	0	22
3	2	-6
5	0	-15

Sparse Matrix ‘b’

- 희소 행렬의 전치 (Transpose)

```
void transpose( SMarray a[], SMarray b[])
/* a 를 전치시켜 b 를 생성,  예:(0,3,22) -> (3,0,22) */
{
   int    i, j, currentb;
     b[0].row = a[0].col;
     b[0].col = a[0].row;
     b[0].value = a[0].value;

   if (a[0].value   >   0)   { /* 0 이 아닌 행렬 */
      currentb = 1;
      for (i =0; i <a[0].col; i++) /* a 에서 열별로 전치*/
         for (j = 1;   j  ≤  a[0].value;   j++)
              /* 현재의 열로부터 원소를 찾는다. */
              if (a[j].col ==i ) {
                 /*현재 열에 있는 원소를 b 에 첨가 */
                 b[currentb].row = a[j].col;
                 b[currentb].col = a[j].row;
                 b[currentb].value=a[i].value;
                 currentb++;
            }
         }
 }
```