제5장

배열

학습 목표

- 배열에 대한 소개
  - 배열의 선언과 참조
  - for 루프와 배열
  - 메모리상의 배열
- 함수에서의 배열
  - 함수 인자, 리턴 값으로서의 배열
- 배열 프로그래밍
  - 부분적으로 채워진 배열, 탐색, 정렬
- 다차원 배열

배열

- 배열이란? 같은 유형의 데이터 집합
  - C와 C++에서는 “정적 배열 " 이 가능
- 배열 선언 → 메모리 할당

```
int score[5];
```

  - "score" 라는 이름으로 크기가 5인 정수형 배열 선언
  - 5개의 변수를 선언하는 것과 유사:

    int score[0], score[1], score[2], score[3], score[4]
- 각각의 부분은 다양한 방식으로 명명:
  - 인덱스 변수 또는 첨자 변수
  - 배열의 요소
  - 괄호 안의 값은 인덱스 또는 첨자라 함
    - 0부터 0 to size - 1의 숫자

배열 접근

- 인덱스/첨자를 이용하여 접근
  - cout << score[3];
- 주목! 괄호 안의 2가지 사용법:
  - 선언 시에는 배열의 크기를 의미
  - 어느 위치에서 사용되면 첨자를 의미
- 크기, 첨자는 상수일 필요가 없음
  - int score[MAX_SCORES];
  - score[n+1] = 99;
    - 만약 n이 2이면, score[3]과 동일

배열의 중요한 함정

- 배열 인덱스는 항상 0부터 시작한다!
- 0은 컴퓨터 과학자에게 첫 번째 수이다.
- C++는 범위를 벗어날 수 있도록 놔둔다
  - 예측할 수 없는 결과
  - 컴파일러는 이러한 에러를 탐지할 수 없다.
- 범위안에 머무르는 것은 프로그래머의 몫

배열의 중요한 함정 예제

- 0부터 배열크기-1의 범위를 인덱스
  - 예제:

    double temperature[24]; // 24는 배열 크기

    // temperature라 명명된 24개의 double형을 가지는 배열 선언
    - 위 배열은 다음과 같이 인덱스된다:

      temperature[0], temperature[1] … temperature[23]
  - 일반적인 실수:

    temperature[24] = 5;
    - 인덱스 24는 범위초과!
    - 경고 없이 치명적인 결과를 초래

배열 크기로서 정의된 상수

- 항상 배열 크기로서 정의된 상수를 사용
- 예제:

```
const int NUMBER_OF_STUDENTS = 5;
int score[NUMBER_OF_STUDENTS];
```

- 가독성 향상
- 융통성 향상
- 유지보수 향상

정의된 상수의 사용

- 배열 크기가 요구되는 곳에 사용
  - 순회하기 위한 for 루프에서:

```
for (idx = 0; idx < NUMBER_OF_STUDENTS; idx++)
{
    // 배열 다루는 코드
}
```

  - 크기가 포함된 계산에서:

```
lastIndex = (NUMBER_OF_STUDENTS – 1);
```

  - 함수에 배열을 전달할 때 (나중에)

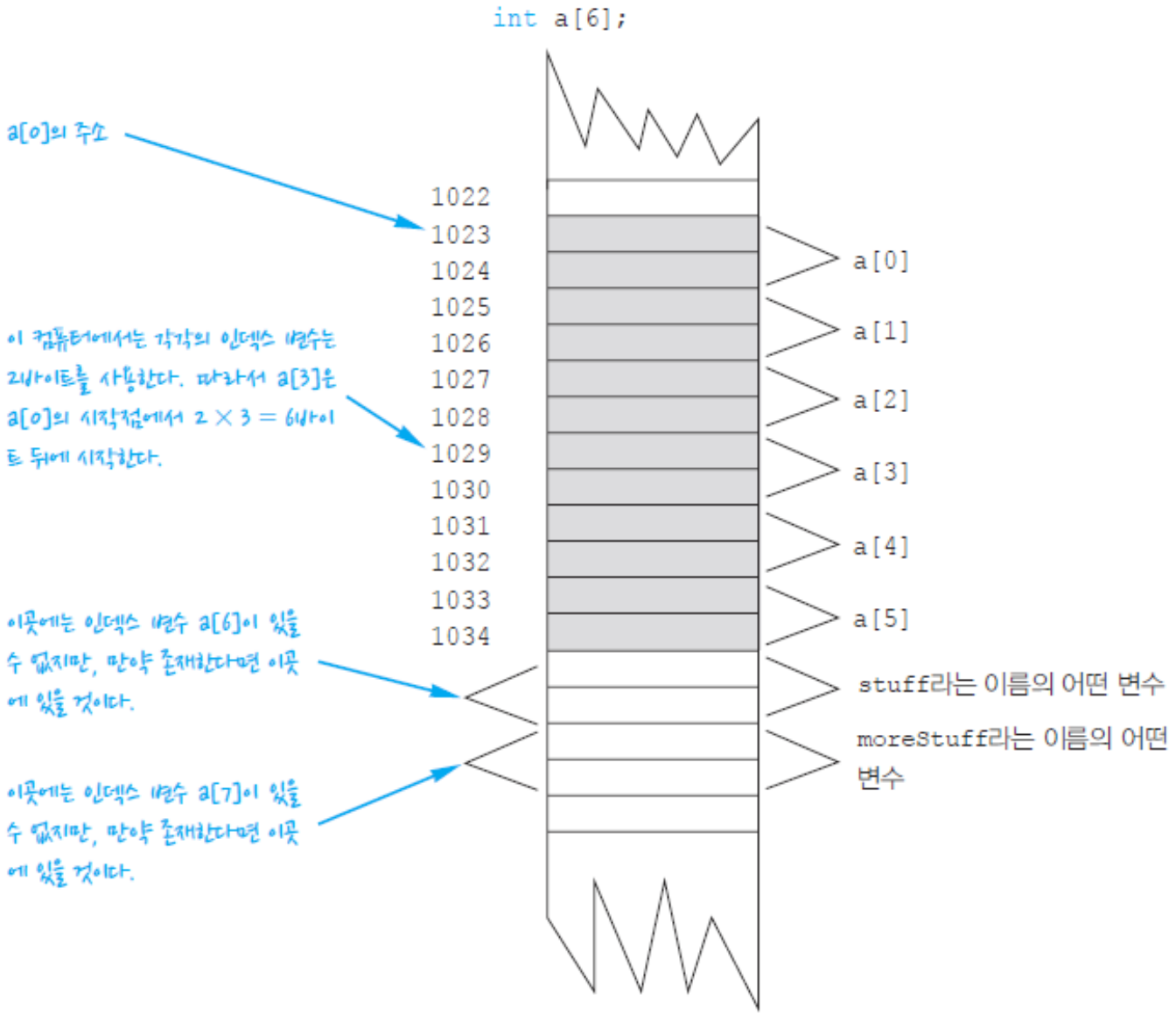
- 만약 크기가 변경되면 → 프로그램에서 하나만 변경하면 된다!

메모리상의 배열

- 일반 변수에서:
  - 주소로서 메모리에 할당됨
- 배열 선언은 전체 배열을 위해 메모리를 할당
- 순차적 할당
  - 앞뒤로 할당된 주소를 의미
  - 인덱싱 계산
    - 단순히 인덱스 0부터 시작하는 배열로부터 덧셈

메모리상의 배열

디스플레이 5.2 메모리상의 배열



배열의 초기화

- 일반 변수로서 선언과 동시에 초기화 가능:

  int price = 0; // 0은 초기값이다.

- 배열도 가능:

  int children[3] = {2, 12, 1};

  - 아래와 동일하다:

    int children[3];

    children[0] = 2;

    children[1] = 12;

    children[2] = 1;

배열의 자동 초기화

- 크기보다 작은 값을 제공하면:
  - 시작부터 채워진다.
  - 나머지는 0으로 할당

    int zz[5] = {5, 12, 11}; // zz[3] zz[4] 는 0으로 초기화
- 만약 배열 크기를 지정하지 않는다면
  - 초기화한 값의 수를 기반으로 배열의 크기를 선언함
  - 예제:

    int b[] = {5, 12, 11};
    - 배열 b는 크기 3을 할당

실습 - 문제해결능력

- [배열] 10진수를 2진수로 변환하는 프로그램을 작성하시오.
  - 정수 입력이므로 최대 32크기의 bool 배열? short 배열? char 배열?에 변환된 2진수를 저장하고 출력
  - 아이디어 : 7는 2로 나누면 나머지는 1, 몫은 3 이다. (7 = 3*2 + 1)

    3 을 2로 나누면 나머지는 1, 몫은 1 이다. (3 = 1*1 + 1)

    1 을 2로 나누면 나머지는 1, 몫은 0 이다. (1 = 0*2 + 1)

    그래서 7은 2진수로 1 1 1이다.
  - 아래와 동일하게 앞쪽에 필요없는 0은 제외하고 출력하시오

```
10진수 입력 : 7
2진수는 111
10진수 입력 : 12
2진수는 1100
10진수 입력 : 12345
11000000111001
10진수 입력 : 2147483647
11111111111111111111111111111111
```

실습 - 문제해결능력

- 5개의 정수를 오름차순으로 저장한 배열 두개를 입력받아, 하나의 정렬된 배열로 만드는 프로그램
  - 합병된 10개 크기의 배열은 sort가 아닌 merge로 구현할 것

```
오름차순 정수 5개 입력: 1 2 3 3 7
오름차순 정수 5개 입력: 4 5 8 11 14
합병된 정수 10개는 : 1 2 3 3 4 5 7 8 11 14
```

함수에서의 배열

- 함수의 인자로서
  - 인덱스 변수
    - 배열의 각각의 원소는 함수의 매개변수가 된다.
  - 완전한 배열
    - 배열의 모든 원소가 하나의 독립체로 전달된다.
- 함수로의 리턴 값으로서
  - 가능하다 → 10장에서 다룸

인자로서의 인덱스 변수

- 인덱스 변수는 배열 기본형의 일반 변수로서 다뤄진다.
- 주어진 함수의 선언:

```
void myFunction(double par1);
```

- 그리고, 아래 선언:

```
int i;  double n, a[10];
```

- 다음과 같이 함수 호출 가능:

```
myFunction(i);       // i는 double형으로 변환
myFunction(a[3]);    // a[3]은 double형
myFunction(n);       // n은 double형
```

인덱싱의 미묘함

- 다음을 고려하자:

  myFunction(a[i]);

  - i의 값이 첫 번째로 결정된다.
    - 어느 인덱스 변수가 보내지는지를 결정
  - myFunction(a[i*5]);
  - 컴파일러 입장에선, 완벽하게 합법적
  - 프로그래머가 배열의 경계에 있도록 책임을 져야 한다.

인자로서의 완전한 배열

- 형식 매개변수는 완전한 배열이 될 수 있다.
  - 함수 호출에 전달된 인자는 배열 이름
  - “배열 매개변수”라 함
- 마찬가지로 배열의 크기를 전달
  - 전형적으로 두 번째 매개변수로서 전달
  - 기본 int형 형식 매개변수

디스플레이 5.3 배열 매개변수를 가지는 함수

디스플레이 5.3 배열 매개변수를 가지는 함수

함수 선언

```
void fillUp(int a[], int size);
//선행조건: size는 배열 a의 선언된 크기이다.
//사용자가 size 정수 값을 입력할 것이다.
//사후조건: 키보드로부터 배열 a는
//size 정수를 가지고 채워진다.
```

함수 정의

```
void fillUp(int a[], int size)
{
      cout << "Enter " << size << " numbers:\n";
      for (int i = 0; i < size; i++)
          cin >> a[i];
      cout << "The last array index used is " << (size - 1) << endl;
}
```

인자로서의 완전한 배열 예제

- 주어진 앞선 예제에서 :
- 어떠한 main()함수 정의에서 다음 호출을 고려하자:

```
int score[5], numberOfScores = 5;
fillup(score, numberOfScores);
```

  - 1st 인자는 완전한 배열
  - 2nd 인자는 정수값
- 배열 인자에 괄호가 없는 것을 주목!

인자로서의 배열 : 어떻게?

- 실제 전달되는 것은 무엇인가?
- 세 부분으로 배열을 생각하자
  - 첫 번째 인덱스 변수의 주소(arrName[0])
  - 배열 기본형
  - 배열의 크기
- 오직, 첫 번째 부분만 전달된다!
  - 단지 배열의 시작주소
  - 참조에 의한 전달과 매우 유사

배열 매개변수

- 이상하지 않는가?
  - 배열 인자에 괄호가 없음
  - 따로 크기를 전달해야만 함
- 하나의 좋은 특성 :
  - 같은 함수로 어떤 크기로도 배열을 채우기 위해 사용가능!
  - 전형적인 함수의 재사용 특성
  - 예제:

```
int score[5], time[10];
fillUp(score, 5);
fillUp(time, 10);
```

const 매개변수 수정자

- 다시 얘기하자면: 배열 매개변수는 실제로 첫 번째 요소의 주소가 전달
  - 참조에 의한 전달과 유사
- 함수는 배열을 수정 가능!
  - 때론 가치가 있지만, 때로는 그렇지 않다!
- 배열 내용이 수정되는 것을 방지
  - 배열 매개변수 앞에 const 수정자 사용
    - “상수 배열 매개변수”라고 함
    - 컴파일러에게 수정을 허락하지 않는다고 알리는 역할

배열을 리턴하는 함수

- 함수는 일반 변수가 리턴되는 것과 같은 방법으로 배열을 리턴할 수 없음
- 포인터의 사용이 요구됨
- 10장에서 다룸

배열 프로그래밍

- 다양한 사용
  - 부분적으로 채워진 배열
    - 어떠한 최대 크기가 선언되어야만 함
  - 정렬
  - 탐색

부분적으로 채워진 배열

- 정확히 요구되는 배열 크기를 아는 것은 어려운 일
- 충분히 사용 가능한 크기를 선언해야만 함
  - 그리고나서 배열 내의 유효 데이터의 추적을 유지해야만 함
  - 추가적인 “추적” 변수가 요구됨
    - int numberUsed;
    - 현재 배열 내의 요소 개수를 추적함

디스플레이 5.5 부분적으로 채워진 배열 (1 of 3)

디스플레이 5.5 부분적으로 채워진 배열

```
1   //각각의 골프 스코어와 스코어 평균의 차이를 보여 준다.

2   #include <iostream>
3   using namespace std;
4   const int MAX_NUMBER_SCORES = 10;

5   void fillArray(int a[], int size, int& numberUsed);
6   //선행조건: size는 배열 a의 선언된 크기이다.
7   //사후조건: numberUsed는 a에 저장된 값의 수이다.
8   //a[0]부터 a[numberUsed-1]는 키보드로부터 음이 아닌 정수
9   //값을 읽어 채워진다.

10  double computeAverage(const int a[], int numberUsed);
11  //선행조건: a[0]부터 a[numberUsed-1]까지 값을 가진다.
12  //numberUsed > 0이면, a[0]부터 a[numberUsed-1]까지의 평균을 리턴한다.

13  void showDifference(const int a[], int numberUsed);
14  //선행조건: a의 첫 번째 numberUsed 인덱스 변수는 값을 가진다.
15  //사후조건: 배열 a의 첫 번째 numberUsed 각각의 요소와 평균의
16  //차이가 얼마인지를 스크린에 출력한다.
17  int main( )
18  {
```

디스플레이 5.5 (계속)

```
19        int score[MAX_NUMBER_SCORES], numberUsed;
20        cout << "This program reads golf scores and shows\n"
21             << "how much each differs from the average.\n";

22        cout << "Enter golf scores:\n";

23        fillArray(score, MAX_NUMBER_SCORES, numberUsed);
24        showDifference(score, numberUsed);

25        return 0;
26    }
```

```
27    void fillArray(int a[], int size, int& numberUsed)
28    {
29        cout << "Enter up to " << size << " nonnegative whole numbers.\n"
30             << "Mark the end of the list with a negative number.\n";
31        int next, index = 0;
32        cin >> next;
33        while ((next >= 0) && (index < size))
34        {
35            a[index] = next;
36            index++;
37            cin >> next;
38        }
39        numberUsed = index;
40    }
```

```
41  double computeAverage(const int a[], int numberUsed)
42  {
43      double total = 0;
44      for (int index = 0; index < numberUsed; index++)
45          total = total + a[index];
46      if (numberUsed > 0)
47      {
48          return (total/numberUsed);
49      }
50      else
51      {
52          cout << "ERROR: number of elements is 0 in computeAverage.\n"
53               << "computeAverage returns 0.\n";
54          return 0;
55      }
56  }

57  void showDifference(const int a[], int numberUsed)
58  {
59      double average = computeAverage(a, numberUsed);
60      cout << "Average of the " << numberUsed
61           << " scores = " << average << endl
62           << "The scores are:\n";
63      for (int index = 0; index < numberUsed; index++)
64      cout << a[index] << " differs from average by "
65           << (a[index] - average) << endl;
66  }
```

Sample Dialogue

```
This program reads golf scores and shows
how much each differs from the average.
Enter golf scores:
Enter up to 10 nonnegative whole numbers.
Mark the end of the list with a negative numb
69 74 68 -1
Average of the 3 scores = 70.3333
The scores are:
69 differs from average by -1.33333
74 differs from average by 3.66667
68 differs from average by -2.33333
```

다차원 배열

- 1개 이상의 인덱스를 가진 배열
  - char page[30][100];
    - 2개의 인덱스: 배열의 배열
    - 시각화:

      page[0][0], page[0][1], …, page[0][99]

      page[1][0], page[1][1], …, page[1][99]

      …

      page[29][0], page[29][1], …, page[29][99]
- C++ 어떠한 인덱스의 수도 허락
  - 전형적으로 2개 이상은 사용안함

다차원 배열 매개변수

- 1차원 매개변수랑 유사
  - 첫 번째 차원의 크기는 주어지지 않음
    - 두 번째 매개변수로서 제공됨
  - 두 번째 차원의 크기는 제공됨
- 예제:

```
void DisplayPage(const char p[][100], int sizeDimension1)
{
        for (int index1=0; index1<sizeDimension1; index1++)
        {
             for (int index2=0; index2 < 100; index2++)
                        cout << p[index1][index2];
             cout << endl;
        }
}
```

배열의 표현

- 배열이 a[u1][u2], ….., [un]일 경우
- A의 총 원소수: $\prod_{i=1}^{n} u_i$
- 표현 순서: 행우선(row major order), 열우선(column major order)
  - 행우선(row major order)
    - 총 원소수: 2*3*2*2=24
    - 저장 순서:

      a[0][0][0][0], a[0][0][0][1], a[0][0][1][0], a[0][0][1][1]
      a[0][1][0][0], a[0][1][0][1], a[0][1][1][0], a[0][1][1][1]
      ……
      a[1][2][0][0], a[1][2][0][1], a[1][2][1][0], a[1][2][1][1]

    - 즉, 0000, 0001, …, 1210, 1211

      -> 사전순서(lexicographic order)

배열의 표현

- 배열 원소 $a[i_1][i_2],\ldots,[i_n]$의 1차원 배열 위치로의 변환

  (예) a[0][0][0][0] ⇒ 위치 0

  a[0][0][0][1] ⇒ 위치 1

  a[1][2][1][1] ⇒ 위치 23

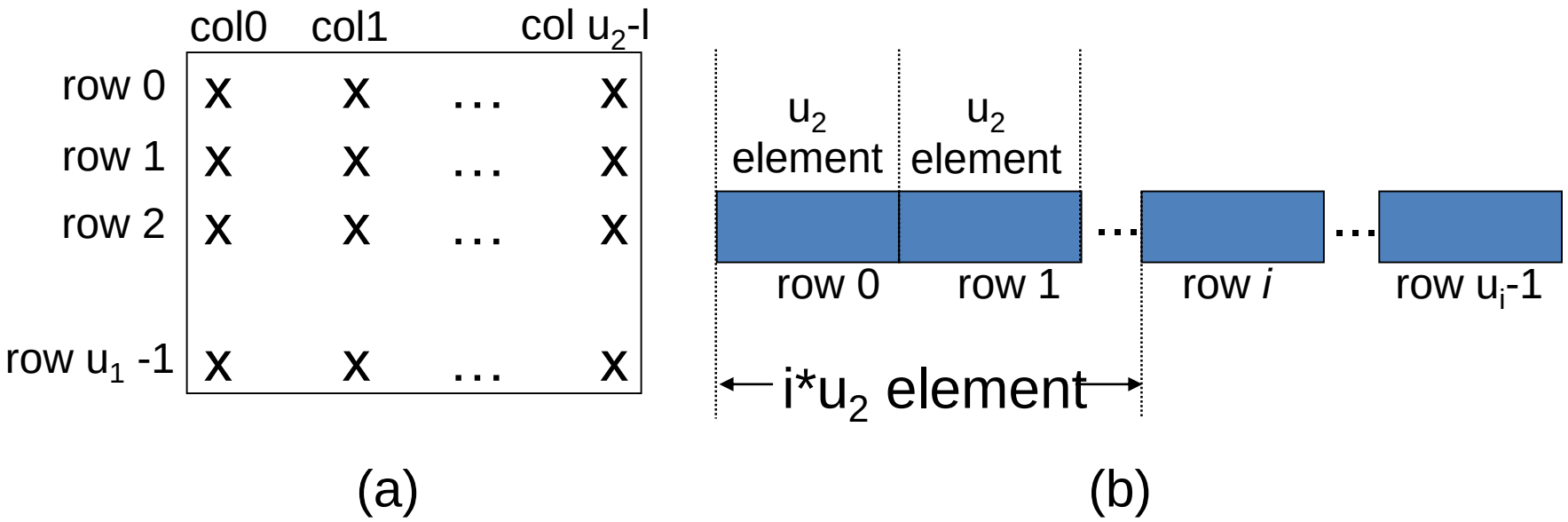
- 1차원 배열 $a[u_1]$
  - α: a[0]의 주소
  - 임의의 원소 a[i]의 주소 : α +i

배열 원소	a[0]	a[1]	...	a[i]	...	$a[u_1-1]$
주소	α	α+1	...	α+i	...	$\alpha+u_1-1$

$a[u_1]$의 순차적인 표현

배열의 표현

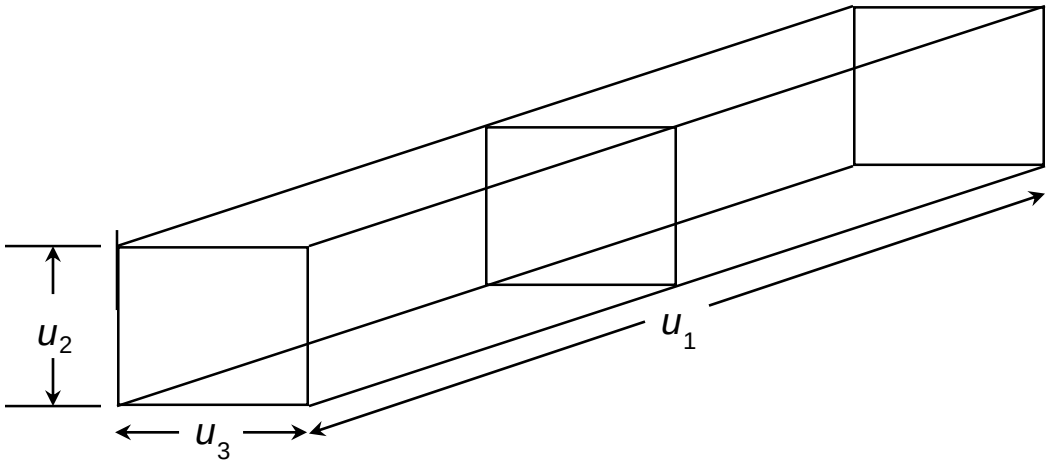
- 2차원 배열
  - α :a[0][0]의 주소
  - a[i][0]의 주소: $\alpha + i*u_2$
  - a[i][j]의 주소: $\alpha+i*u_2+j$



a[u_1][u_2]의 순차적 표현

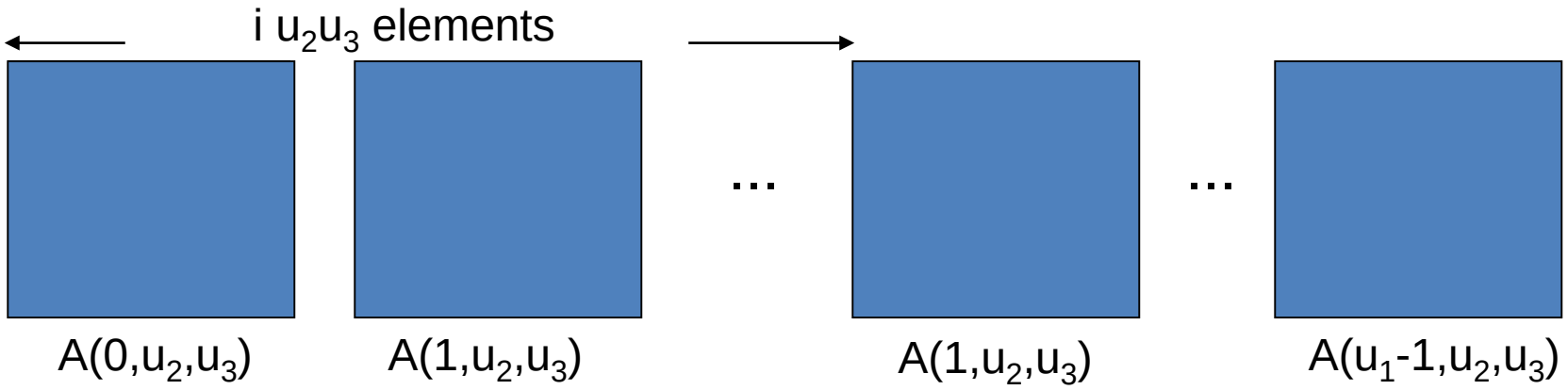
배열의 표현

- 3차원 배열
  - α ;A[0][0][0]의 주소
  - A[i][0][0]의 주소: α +i
  - A[i][j][k]의 주소: $\alpha + i*u_{2*}u_3 + j*u_3 + k$



(a) 3차원 배열 A[u_1][u_2][u_3] 가 u_1개의 2차원 배열로 취급됨

다차원 배열의 표현



(b) 3차원 배열의 순차 행 우선 표현

- **n차원 배열 a[u_1][u_2]…,[u_n]**
  - α ; A[0][0],…,[0]의 주소
  - a[i_1][i_2][0],…,[0]의 주소: $\alpha + i_1 u_2 u_3 \ldots u_n$
  - a[i_1][i_2][0],…,[0]의 주소: $\alpha + i_1 u_2 u_3 \ldots u_n + u_3 + i_2 u_2 u_3 \ldots u_n$
  - a[i_1][i_2],…,[i_n]의 주소

$$\alpha + \sum_{j=1}^{n} i_j a_j \qquad \text{여기서} \quad \begin{cases} a_j = \prod_{k=j+1}^{n} u_k, & 1 \le j < n \\ a_n = 1 \end{cases}$$

다항식 표현

- 첫번째 결정 : 서로 다른 지수들은 내림차순으로 정돈
- [표현 1] Polynomial의 전용 데이타 멤버 선언

$a(x)=3x^2+2x-4$
$b(x)=x^8-10x^5-3x^3+1$

```
class Polynomial
{
private:
      int degree;              //degree ≤  MaxDegree
      float coef [MaxDegree + 1];     // 계수 배열
```

[0]	0	0
[1]	0	0
[2]	0	1
[3]	0	0
[4]	0	0
[5]	0	-10
[6]	0	0
[7]	0	-3
[8]	3	0
[9]	2	0
[10]	-4	1

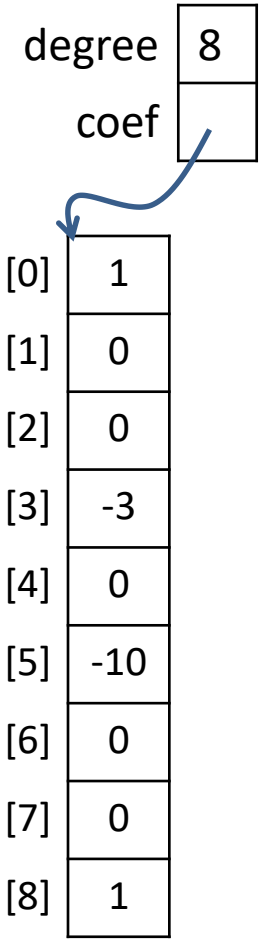
- a가 Polynomial 클래스 객체, n≤MaxDegree

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

- a.coef[i]는 x_{n-i}의 계수, 각 계수는 지수의 내림차순으로
- 대부분 다항식의 연산(덧셈, 뺄셈, 계산, 곱셈 등)을 위 알고리즘을 간단하게 구성

다항식 표현

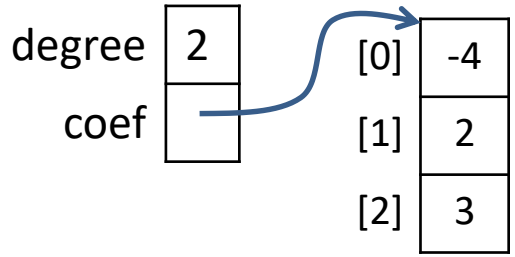
$b(x)=x^8-10x^5-3x^3+1$



- [표현 2] Polynomial의 전용 데이타 멤버 선언

```
class Polynomial {
private:
      int degree;
      float *coef;
```

$a(x)=3x^2+2x-4$



  - 생성자와 파괴자를 Polynomial에 추가

```
Polynomial::Polynomial(int d) {
    degree=d;
    coef=new float[degree+1];
}
```

```
Polynomial::~Polynomial() {
    delete [] coef ;
}
```

  - 희소 다항식에서 기억 공간 낭비
    - (예) 다항식 $x^{1000}+1$ → coef에서 999개의 엔트리는 0
      → 표현 3방법(클래스 term) 이용

실습 - 문제해결능력

- 아래와 같은 두개의 행렬을 자동으로 생성하고, 두 행렬의 합과 곱을 계산하여 출력하는 프로그램
  - 클래스 사용하지 않아도 됨
  - 두 행렬 $\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}\begin{bmatrix}1 & -1 & 0\\0 & -1 & 1\\-1 & 1 & 0\end{bmatrix}$ 은 값을 하나하나 입력하는 방식으로 초기화 하지 말것

```
두 행렬의 합은
|2 1 3|
|4 4 7|
|6 9 9|

행렬의 곱은
|-2  0 2|
|-2 -3 5|
|-2 -6 8|
```

요약

- 배열은 같은 자료형의 집합
- 배열의 인덱스 변수는 기본형의 다른 변수와 같이 사용됨
- for 루프는 배열을 횡단하기 위한 자연적인 방법
- 프로그래머는 배열의 범위에 있어야 하는 책임이 존재
- 배열 매개변수는 참조에 의한 호출과 유사
- 배열 요소는 순차적으로 저장됨
  - 연속적인 메모리의 위치. 오직 첫 번째 요소의 주소가 함수에 전달됨
- 부분적으로 채워진 배열 → 사용된 크기 추적
- 상수 배열 매개변수
  - 배열 내용의 수정을 보호
- 다차원 배열
  - 배열의 배열을 생성