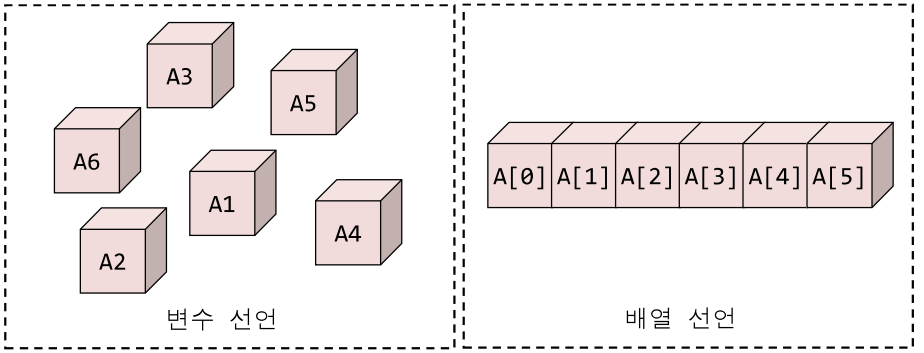
DATA STRUCTURES USING C++

02 CHAPTER

배열과 클래스

배열

- 같은 형의 변수를 여러 개 만드는 경우에 사용
  - 여러 개의 변수 선언: int A0, A1, A2, A3, …,A5;
  - 하나의 배열 선언: int A[6];



배열이 없다면?

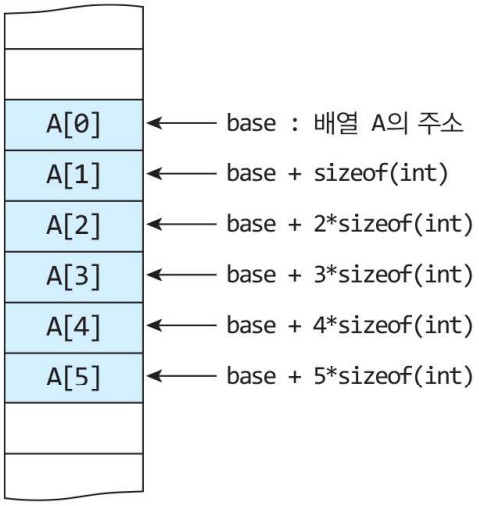
- 반복 코드 등에서 배열을 사용하면 효율적인 프로그래밍이 가능
  - 예) 최대값을 구하는 프로그램

```
tmp=score[0];
for(int i=1;i<n; i++) {
      if( score[i] > tmp )
            tmp = score[i];
}
```

- 만약 배열이 없다면?
  - 반복문을 사용할 수 없다!

1차원 배열

- **선언 : 자료형 배열이름[배열의_크기];**
- int A[6];



문자열 : 특별한 1차원 배열

- `char s[12] = "game over";`

	s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]	s[10]	s[11]
s	'g'	'a'	'm'	'e'	' '	'o'	'v'	'e'	'r'	'\0'		

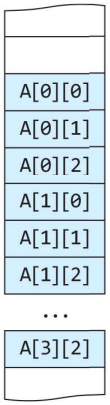
- 문자열 처리
  - 문자열의 복사나 비교를 위해 =나 == 또는 <등의 연산자를 사용할 수 없다.
  - **strcmp(), strcpy(), ...**
  - <string.h> 포함 (또는 <cstring>)

2차원 배열

- **선언** : **자료형** 배열이름**[행의_크기][열의_크기]**;
  - `int A[4][3];`
  - `int A[4][3]= { {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} };`

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]
A[3][0]	A[3][1]	A[3][2]

(a) 2차원 배열

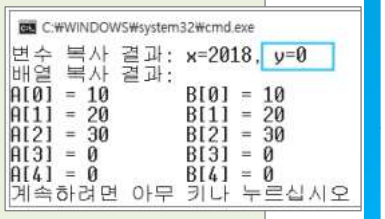


(b) 실제 메모리 안에서의 위치

배열 : 변수의 전달

- 변수의 전달 → 값을 복사 (call by value), cf. call by reference
- **배열의 전달** → 첫 번째 항목의 주소를 전달(주소를 복사)

```
void copy_array(int a[], int b[], int len) {
        int i;
        for (i = 0; i < len; i++)
                b[i] = a[i];
}
void copy_variable(int a, int b) {
        b = a;
}
…
        int A[5] = { 10, 20, 30 };
        int B[5], i, x = 2018, y = 0;
        copy_variable(x, y);
        copy_array(A, B, 5);
```



배열에서의 주의사항

- call by value **vs.** call by reference

```
void sub(int x, int arr[]) {
        x = 10;
        arr[0] = 10;
}

void main()
{
        int var=0, list[MAX_SIZE];
        list[0] = 0;
        sub(var, list);
        // var?, list[0]?
}
```

배열에서의 주의사항

- 매개 변수로 배열의 길이도 전달해야 함.
  - 배열의 길이를 알아야 라스트 원소까지만 처리

int arr[10] = {3, 24, 82, 12, 34, 7, 53, 17, 26, 51};
int maxVal = findMaxValue(arr, 10);

```
int findMaxValue( int a[], int len )
{
  int maxVal = a[0];
  for (int i=1; i<len; i++)
       if (maxVal <a[i])
         maxVal = a[i];
  return maxVal;
}
```


배열에서의 주의사항

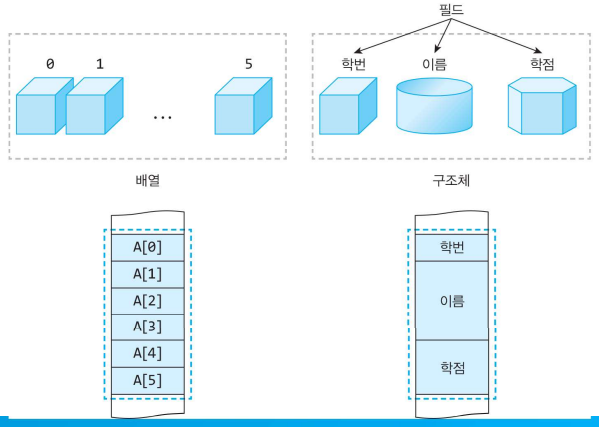
- 2차원 이상의 다차원 배열의 매개 변수 전달에 조심.
  - int findMaxPixel(int a[][5], int h, int w)

```
// 2차원 영상에서 최대 밝기 값을 찾아 반환하는 함수
int findMaxPixel( int a[][5], int h, int w )
{
        int maxVal = 0;         // 영상의 최소 밝기 = 0
        for( int i=0 ; i<h ; i++ )
        for( int j=0 ; j<w ; j++ )
                if( maxVal < a[i][j] ) maxVal = a[i][j];
        return maxVal;
}
void main()
{
        int img[4][5]= {{  3, 24, 82, 12, 22}, { 34,  7, 12, 19, 21},
                        { 99,  7, 65, 73, 58}, { 20,  7,  9, 48, 29}};
        int maxPixel = findMaxPixel( img, 4, 5 );
        printf( "영상의 최대 밝기 = %d\n", maxPixel );
}
```


구조체

구조체

- 기존의 자료형들을 조합해 새로운 자료형을 만드는 방법
- 배열과의 차이
  - **구조체(structure):** 타입이 다른 데이터를 하나로 묶음
  - **배열(array):** 타입이 같은 데이터들을 하나로 묶음



구조체의 정의와 선언(C++, C)

- 정의

```
struct Student {
        int id;
        char name[20];
        double score;
};
```

```
typedef struct Student_t {
        int id;
        char name[20];
        double score;
} Student;
```

- 선언

```
Student a;
```

```
Student a;
```

```
Student a = { 201803156, “홍길동” , 96.3 };
```

- 멤버 접근: 항목 연산자(membership operator) '.'

```
a.id = 30830;
a.score = 92.3;
strcpy(a.name, "Jinyoung");
// a.name = “Jinyoung”;은 오류 발생
```


구조체와 연산자

- 대입 연산자만 가능

```
int x, y=10;
Student a, b={ 201803156, “홍길동” , 96.3 };
x = y;              // OK: int 변수의 복사
a = b;              // OK: 구조체 변수의 복사
```

- 다른 연산자 사용 불가

```
if( a > b )         // 오류: 구조체의 비교연산 불가
      a += b;       // 오류: 구조체의 다른 대입 연산도 불가

int compare(Student a, Student b) {
      return a.id - b.id;
}
```


구조체와 함수

- 함수의 매개 변수나 반환형으로 사용할 수 있음.
  - **Call by value**

- 다음 함수의 동작은?

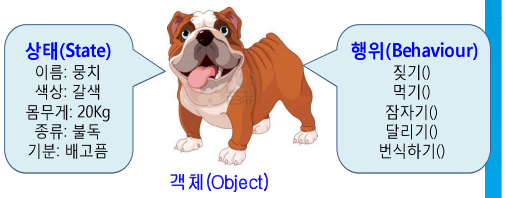
```
void print_complex(Complex c) {
        printf("%4.1f + %4.1fi\n", c.real, c.imag);
}
void reset_complex(Complex c) {
        c.real = c.imag = 0.0;
}
```

```
선택 C:₩WINDOWS₩system32₩cmd.exe
초기화 이전: a =  1.0 +  2.0i
초기화 이후: a =  1.0 +  2.0i
계속하려면 아무 키나 누르십시오 . . .
```


클래스

구조체 vs. 클래스(객체)

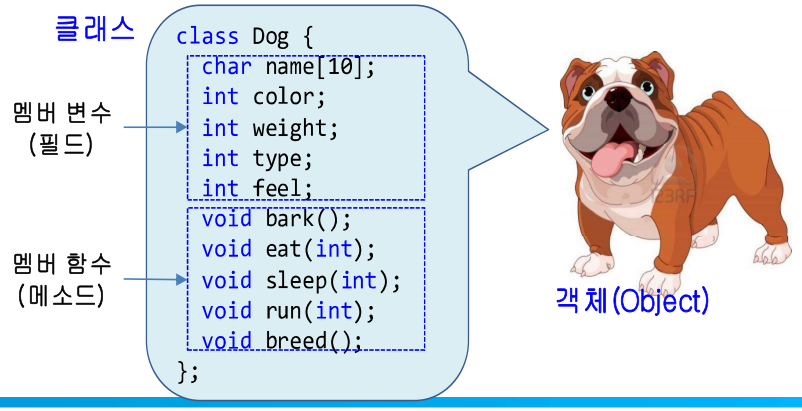
- 객체는 상태와 행위를 가짐



- **절차 지향적** 프로그래밍으로 구현: **구조체+함수**

객체 지향적 프로그래밍으로 구현

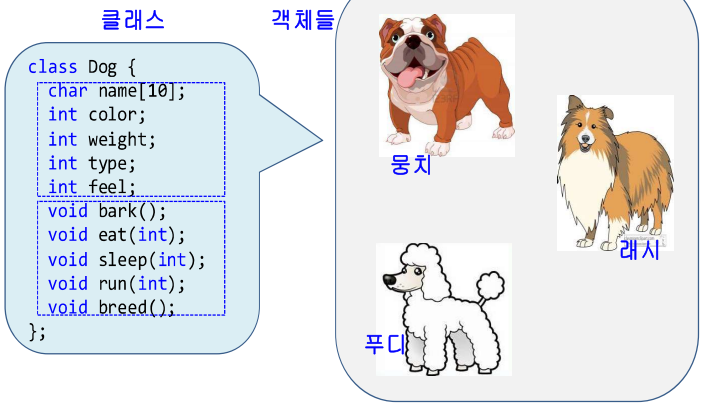
- 구조체 데이터와 함수들을 묶는 것 ➔ 클래스
  - 속성: 멤버 변수(member variable) 또는 필드(field)
  - 행위: 멤버 함수(member function) 또는 메소드(method)



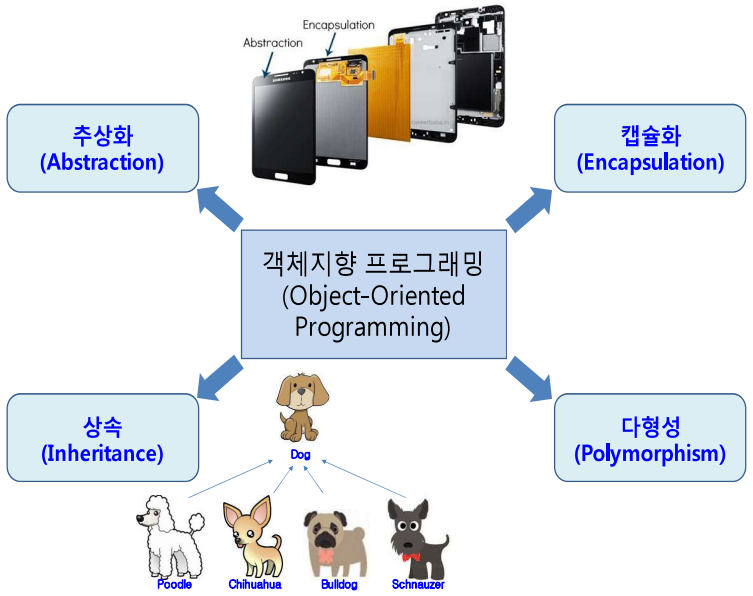
객체(Object)

- 클래스의 사례(instance)
  - 변수

```
int     x;
Dog     mungchi;
Dog     puddy;
Dog     Rashi;
Complex a;
```



객체지향 프로그래밍의 주요 특징



클래스의 선언과 활용

- 클래스 선언: class 또는 struct
  - 멤버 변수와 멤버 함수를 클래스 블록에 포함

```
class 클래스명 {          // 새로운 클래스를 선언
private:                  // 멤버 접근 지정자(public, private, protected)
      멤버변수1;          // 멤버 변수는 객체의 속성을 나타냄
      멤버변수2;
      …
public:                   // 멤버에 대한 접근 지정자
      멤버함수1;          // 멤버 함수는 객체의 동작을 나타냄
      멤버함수2;
      …
};                        // 세미콜론(';')를 잊지 말아야 함.
```


클래스의 선언과 활용

- 접근 지정자
  - private: 전용. 외부에서 접근 불가
  - protected: 보호. 자식 클래스까지 접근 허용. 외부 접근 불가
  - public: 공용. 누구나 접근 가능

- class, struct 선언
  - class만 접근 지정자 개념 있음
  - 클래스 블록이 끝나면 반드시 세미콜론(;)

```
struct Complex
{
  // 클래스 몸체
};
```

정확히 동일한 선언

```
class Complex
{
public:
  // 클래스 몸체
};
```


객체의 생성과 멤버 접근

- 복소수 클래스의 선언과 사용 예

```
class Complex {
private:
  double  re;        // 실수부
  double  im;        // 허수부
public:
  void set(double r, double i) {
    re = r;
    im = i;
  }
  void print() {
    printf("%4.1f +%4.1fi \n",
                    re, im);
  }
  double mag () {
    return sqrt(re*re + im*im);
  }
};
```

private
클래스 전용 멤버

클래스 전용 멤버는 클래스 내에서만 사용할 수 있음

클래스 전용 멤버는 클래스 외부에서 **접근 불가**

public
공용 멤버

```
void main()
{
  Complex a, b;
  a.set(1.0, 2.0);
  b.set(3.0, 4.0);
  a.print();
  b.print();
  printf("a의 크기=%lf\n",
                  a.mag ());
  a.re = 5.0; // 잘못된 문장
}
```

공용 멤버는 클래스 외부에서도 사용 가능

  - 객체 a의 멤버 함수 set을 호출
  - 객체 a에게 set하라는 메시지를 보냄(필요한 정보는 인수로)

사례: Complex의 다양한 변신

- 클래스를 다양한 형태로 구현할 수 있음
  - Complex V1: 구조체와 일반 함수로 구현한 복소수
  - Complex V2: 복소수를 클래스로 전환 (데이터 + 함수)
  - Complex V3: 멤버 이름의 단순화
  - Complex V4: 모든 멤버 함수를 inline으로 구현

Complex V1: 구조체와 함수 (1)

Complex.h

```
#pragma once
#include <cstdio>
struct Complex {
  double   real;
  double   imag;
};
inline void setComplex(Complex &c, double r, double i){
  c.real = r;
  c.imag = i;
};
extern Complex readComplex(char* msg = " 복소수 =");
extern void printComplex(Complex c, char* msg = "복소수 = ");
extern Complex addComplex(Complex a, Complex b);
```

Complex 구조체 선언

inline 함수 선언

함수 원형 선언

Complex V1: 구조체와 함수 (2)

Complex.cpp

```
#include "Complex.h"
Complex readComplex( char* msg ) {
  Complex c;
  printf(" %s ", msg);
  scanf("%lf%lf", &c.real, &c.imag);
  return c;
}
void printComplex(Complex c, char* msg) {
  printf(" %s %4.2f + %4.2fi\n", msg, c.real, c.imag);
}
Complex addComplex(Complex a, Complex b) {
  Complex c;
  c.real = a.real + b.real;
  c.imag = a.imag + b.imag;
  return c;
}
```

복소수 객체 입력 함수

복소수 객체 출력 함수

두 객체의 합을 구해 반환하는 함수

Complex V1: 구조체와 함수 (3)

ComplexTest.cpp

```
#include "Complex.h"
void main()
{
  Complex a, b, c;
  a = readComplex ( "A =" );
  b = readComplex("B =");
  c = addComplex(a, b);
  printComplex( a, " A = ");
  printComplex( b, " B = ");
  printComplex( c, " A+B = ");
}
```

Complex 객체 생성

복소수 객체 a와 b의 정보 입력

복소수 덧셈 연산
c = a + b

복소수 객체 출력

Complex V2: 클래스로 변환 (1)

Complex.h

```
#pragma once
#include <cstdio>
class Complex
{
  double   real;
  double   imag;
public:
  void setComplex( double r, double i ) {
    real = r;
    imag = i;
  }
  void readComplex( char* msg = " 복소수 = " );
  void printComplex( char* msg = "복소수 = " );
  void addComplex ( Complex a, Complex b );
};
```

데이터 멤버. 모두 private 로 선언됨

inline 으로 구현된 멤버 함수. 매개변수가 하나 줄고, 코드도 단순해 짐.

멤버 함수들

모든 멤버 함수에 범위 연산자(::)가 적용됨
모두 Complex 클래스의 멤버 함수임을 나타냄

Complex V2: 클래스로 변환 (2)

Complex.cpp

```
#include "Complex.h"
void Complex::readComplex( char* msg ) {
  printf(" %s ", msg);
  scanf("%lf%lf", &real, &imag);
}
void Complex::printComplex( char* msg ) {
  printf(" %s %4.2f + %4.2fi\n", msg, real, imag);
}
void Complex::addComplex( Complex a, Complex b ) {
  real = a.real + b.real;
  imag = a.imag + b.imag;
}
```

반환형이 void로 바뀜
scanf() 의 인수가 단순해 짐

매개변수가 줄어듦
직접 real, imag 사용

반환형이 void로 바뀜
직접 real, imag 사용

Complex V2: 클래스로 변환 (3)

ComplexTest.cpp

```
#include "Complex.h"
void main()
{
  Complex a, b, c;
  a.readComplex ( "A =" );
  b.readComplex ( "B =" );
  c.addComplex (a, b);
  a.printComplex( " A = ");
  b.printComplex( " B = ");
  c.printComplex( " A+B = ");
}
```

Complex 객체 생성

객체에게 복소수 값을 읽으라는 메시지를 보냄. 처리 결과는 그 객체에 저장됨. 반환 불필요

객체 c에게 a와 b를 더하라는 메시지를 보냄. 결과는 c에 저장되므로 반환이 필요 없음.

객체에게 자신의 정보를 출력하라는 메시지를 보냄. 객체를 매개변수로 보낼 필요가 없음.

Complex V3: 이름 단순화 (1)

Complex.h

```
#pragma once
#include <cstdio>
class Complex
{
  double   real;
  double   imag;
public:
  void set ( double r, double i ) {
    real = r;
    imag = i;
  }
  void read ( char* msg = " 복소수 = " );
  void print ( char* msg = " 복소수 = " );
  void add ( Complex a, Complex b );
};
```

일반 멤버 함수들의 이름을 단순하게 변경함. 이 클래스에서만 의미가 있으므로 예를 들어, readComplex() 대신에 read()만 하더라도 의미가 명확함

Complex V3: 이름 단순화 (2)

Complex.cpp

```
#include "Complex.h"
void Complex::read ( char* msg ) {
  printf(" %s ", msg);
  scanf("%lf%lf", &real, &imag);
}
void Complex::print ( char* msg ) {
  printf(" %s %4.2f + %4.2fi\n", msg, real, imag);
}
void Complex::add ( Complex a, Complex b ) {
  real = a.real + b.real;
  imag = a.imag + b.imag;
}
```

함수 이름 단순화에 따른 수정

Complex V3: 이름 단순화 (3)

ComplexTest.cpp

```
#include "Complex.h"
void main()
{
  Complex a, b, c;
  a.read ( "A =" );
  b.read ( "B =" );
  c.add (a, b);
  a.print ( " A = ");
  b.print ( " B = ");
  c.print ( " A+B = ");
}
```

멤버 함수 이름의 단순화에 따라 호출 코드도 간단해 짐.

Complex V4: inline 구현 (1)

Complex.h

```
#pragma once
#include <cstdio>
class Complex
{
  double   real;
  double   imag;
public:
  void set ( double r, double i ) {
    real = r;
    imag = i;
  }
  void read ( char* msg = "복소수 = ") {
    printf(" %s ", msg);
    scanf("%lf%lf", &real, &imag);
  }
  void print ( char* msg " 복소수 = ") {
    printf(" %s %4.2f + %4.2fi\n", msg, real, imag);
  }
  void add ( Complex a, Complex b ) {
    real = a.real + b.real;
    imag = a.imag + b.imag;
  }
};
```

모든 멤버함수를 inline으로 구현. Complex.cpp가 필요 없음

Complex V4: inline 구현 (2)

ComplexTest.cpp

```
#include "Complex.h"
void main()
{
  Complex a, b, c;
  a.read ( "A =" );
  b.read ( "B =" );
  c.add (a, b);
  a.print ( " A = ");
  b.print ( " B = ");
  c.print ( " A+B = ");
}
```

멤버 함수 이름의 단순화에 따라 호출 코드도 간단해 짐.

C++문법 요약 : Car 클래스

```
class Car {
protected:
    int speed;          // 속도 (private)
    char name[40];      // 이름 (private)
public:
    int gear;           // 기어 (public)
    Car(){ }            // 기본 생성자
    ~Car(){ }           // 소멸자
    Car(int s, char* n, int g) // 매개변수가 있는 생성자
        : speed(s), gear(g) { // 멤버 초기화 리스트 (멤버변수 초기화)
            strcpy(name, n); // 생성자 함수 몸체 (name 멤버 초기화)
    }
    void changeGear(int g=4){  // 기어 단수를 변경하는 멤버 함수
        gear = g;
    }
    void speedUp(){             // 속도를 5씩 증가 멤버 함수
        speed += 5;
    }
    void display(){           // 자동차의 정보를 화면에 출력함.
        printf("[%s] : 기어=%d단 속도=%dkmph\n", name, gear, speed);
    }
    void whereAmI() { printf("객체 주소 = %x\n", this); }
    };
```

멤버접근지정자

함수 오버로딩

생성자, 소멸자

멤버초기화리스트

디폴트 매개변수

멤버접근

this 포인터

C++문법 요약 : SportsCar 클래스

```
#include "Car.h" // 자동차 클래스 헤더파일 포함
// 스포츠카 클래스 (자식 클래스) : 자동차 클래스에 터보 기능 추가
class SportsCar : public Car
{
public:
     bool bTurbo;        // 터보 장치 ON?
     void setTurbo(bool bTur) { bTurbo = bTur; }
     void speedUp() {    // 터보가 ON이 되어 있으면 가속이 빨리됨
          if( bTurbo ) speed += 20;
          else Car::speedUp();
     }
};
```

- 클래스 상속
- 오버라이딩
- 기본 대입 연산자 (=)
- 기본 복사 생성자
- 다른 연산자? (>, +=)
- 객체 배열
- 레퍼런스(Reference)

- 연산자 오버로딩
- 복사 생성자와 깊은 복사
- 동적 바인딩
- 다중 상속
- 템플릿
- 예외 처리

배열 + 클래스

배열+클래스 응용 : 다항식 클래스

- 다항식의 일반적인 형태

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

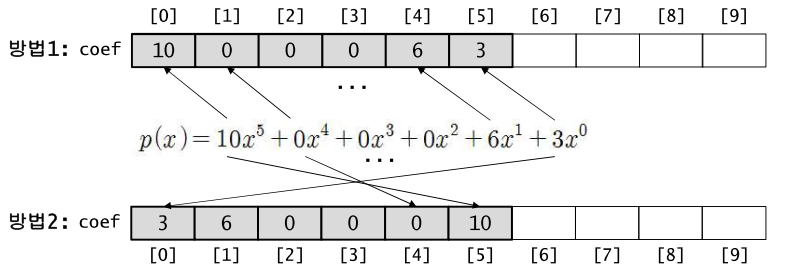
  - 처리를 위한 다항식의 자료구조가 필요
  - 어떤 자료구조가 다항식의 연산을 편리하게 할까?

- 다항식을 위한 자료구조?
  - 배열을 사용하는 방법
    - 다항식의 모든 항을 배열에 저장
    - 다항식의 0이 아닌 항만을 배열에 저장: 희소 다항식
  - 연결 리스트를 사용하는 방법
    - 포인터 이후에 공부
    - 희소 다항식에 적합

배열을 이용한 다항식 클래스

- 모든 차수에 대한 계수 값을 배열로 저장
- 하나의 다항식을 하나의 배열로 표현

```
class Polynomial {
        int degree;
        float coef[MAX_DEGREE];
        ...
};
```



Polynomial 클래스 설계

- 다항식 클래스를 사용하는 코드의 예

```
#include “Polynomial.h”
void main() {
     Polynomial a, b, c;
     a.read(); // 다항식 a를 읽음 (키보드로 입력)
     b.read(); // 다항식 b를 읽음 (키보드로 입력)
     c.add (a, b); // c = a + b
     a.display("A = "); // 다항식 a를 화면에 출력
     b.display("B = "); // 다항식 b를 화면에 출력
     c.display("A+B="); // 다항식 c=a+b를 화면에 출력
}
```

UML 다이어그램

Polynomial
-degree : int -coef[] : *float*
+Polynomial() +isZero(): bool +negate() +add(Polynomial p1, Polynomial p2) +mult(Polynomial p1, Polynomial p2) +read() +display()

Polynomial 구현 (C++)

```
#define MAX_DEGREE 80                  // 다항식의 처리 가능한 최대 차수+1
class Polynomial {
     int        degree;                // 다항식의 최고 차수
     float      coef[MAX_DEGREE];      // 각 항에 대한 계수
public:
     Polynomial() { degree = 0; }      // 생성자: 최대 차수를 0으로 초기화
     // 다항식의 내용을 입력받는 멤버함수
     void read() {
         printf("다항식의 최고 차수를 입력하시오: ");
         scanf( "%d", &degree );
         printf("각 항의 계수를 입력하시오 (총 %d개): ", degree+1);
         for( int i=0 ; i<=degree ; i++)
               scanf( "%f", coef+i );
     }
    // 다항식의 내용을 화면에 출력하는 함수
     void display(char *str=" Poly = “) {      // 디폴트 매개변수 사용
         printf("\t%s", str);
          for( int i=0 ; i<degree ; i++)
               Printf("%5.1f x^%d + ", coef[i], degree-i);
          printf( "%4.1f\n", coef[degree] );
};
```

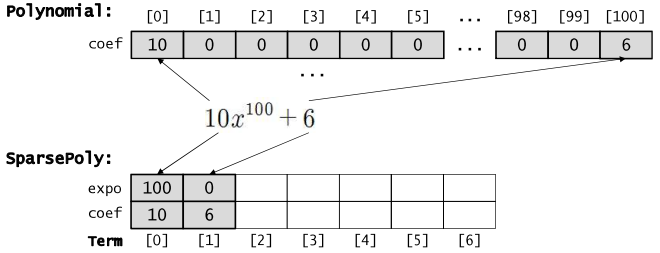

다항식의 덧셈 연산

- 다항식 덧셈 알고리즘?
- 단순화 방법? c=a+b ➔ c = a; c += b;

```
   // 다항식 a와 b를 더하는 함수. a와 b를 더해 자신의 다항식 설정.
    void add( Polynomial a, Polynomial b) {
         if (a.degree > b.degree) {           // a항 > b항
             *this = a;                       // a 다항식을 자기 객체에 복사
             for( int i=0 ; i<=b.degree ; i++ )
                  coef[i+(degree-b.degree)] += b.coef[i];
         }
         else {                               // a항 <= b항
              *this = b;                      // b 다항식을 자신에 복사
              for( int i=0 ; i<=a.degree ; i++ )
                   coef[i+(degree-a.degree)] += a.coef[i];
         }
    }
    bool isZero() { return degree == 0; }     // 최고차수가 0 인가?
    void negate() {...}                       // 모든 계수의 부호를 바꿈
};
```


희소 다항식의 표현

- **희소 다항식(Sparse Polynomial) 이란?**
  - 대부분 항의 계수가 0인 다항식




```
class SparsePoly {
       int      nTerms;
       Term     term[MAX_TERMS];
       ...      // 멤버 함수들
};
```

```
struct Term {
       int      expon;
       float    coeff;
};
```

Thank you !