배열 및 구조체

Array and Structure

배열 및 구조체

배열

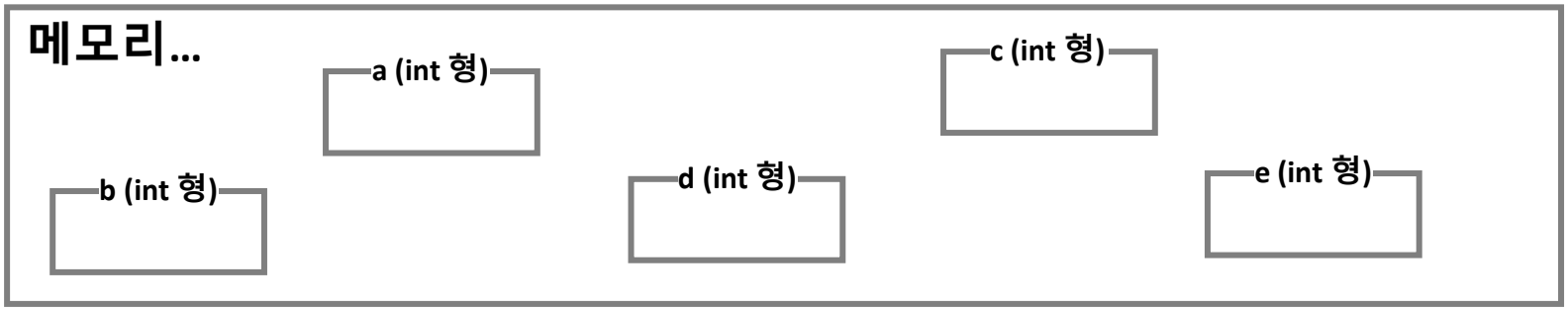
- 다수의 데이터를 저장하고 처리하는 경우 유용하게 사용할 수 있는 자료 구조
- 같은 형 변수를 여러 개 만드는 경우 사용
- 저장 공간을 한꺼번에 확보

배열 및 구조체

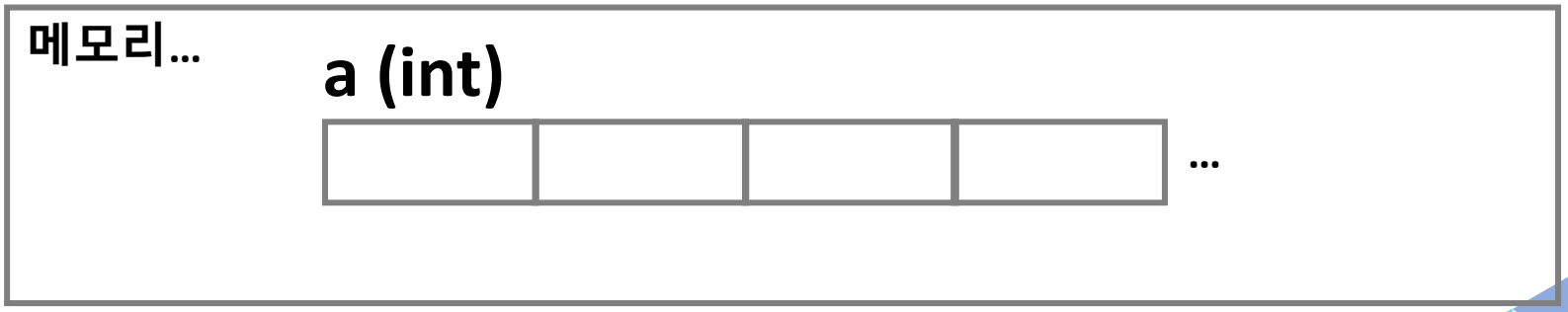
배열 및 변수 차이

- 독립적인 저장 공간 vs. 연속적인 저장 공간

일반 변수(현재 방식) - >int a, b, c, d, e;



배열 변수(앞으로 배울 방식) - > ??? (배열 변수 이름이 a(int)라고 가정)



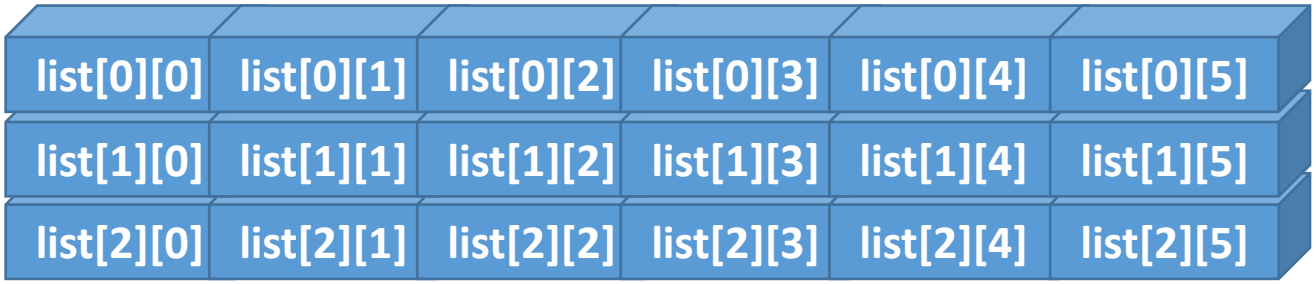
배열 및 구조체

1차원 배열

```
int list[6];
list[0] = 100;
int value = list[0];
```

2차원 배열

```
int list[3][6];
list[0][0] = 100;
```



1차원 배열 기본 실습

배열을 사용한 성적 계산

```c
#include <stdio.h>

int main(void) {
  int std[3], i;
  int first_high_score = 0;
  int scend_high_score = 0;
  for (i = 0; i < 3; i++) {
    scanf_s("%d", &std[i]);
  }
  first_high_score = std[0];

  for (i = 0; i < 3; i++) {
    if (first_high_score < std[i]) {
      scend_high_score = first_high_score;
      first_high_score = std[i];
    }
    else if (std[i] < first_high_score && std[i] > scend_high_score) {
      scend_high_score = std[i];
    }
  }
  printf("1등과 2등의 점수 : %d %d", first_high_score, scend_high_score);
  return 0;
}
```

ex1

1차원 배열 기본 실습

1차원 배열 사이즈 계산 예제

```
#include <stdio.h>
int main(void) {
  int arr1[5] = { 1, 2, 3, 4, 5 };int arr2[] = { 1, 2, 3, 4, 5, 6, 7 };
  int arr3[5] = { 1, 2 };int ar1Len, ar2Len, ar3Len, i;
  printf("배열 arr1의 크기 : %d ₩n", sizeof(arr1));
  printf("배열 arr2의 크기 : %d ₩n", sizeof(arr2));
  printf("배열 arr3의 크기 : %d ₩n", sizeof(arr3));
  ar1Len = sizeof(arr1) / sizeof(int);
  ar2Len = sizeof(arr2) / sizeof(int);
  ar3Len = sizeof(arr3) / sizeof(int);
  for (i = 0; i < ar1Len; i++)
    printf("%d ", arr1[i]);
  printf("₩n");
  for (i = 0; i < ar2Len; i++)
    printf("%d ", arr2[i]);
  printf("₩n");
  for (i = 0; i < ar3Len; i++)
    printf("%d ", arr3[i]);
  printf("₩n");
  return 0;
}
```

배열 및 구조체

구조체

- 하나 이상의 변수를 묶어서 새로운 자료형을 정의하는 도구

- 만약, 프로그램 상 마우스 x, y 좌표를 저장하기 위한 변수 선언?

```
int xpos;
int ypos;
```

- 일반 변수로 표현할 경우 독립된 정보 but 구조체의 경우 하나의 정보로 표현

```
struct point{
   int xpos;
   int ypos;
}
```

- 배열 vs 구조체
  . 배열: 같은 타입 데이터들을 하나로
  . 구조체: 다른 타입 데이터들을 하나로

배열 및 구조체

구조체 사용 예

```
struct studentTag{
    char name[10];   // 문자배열로 된 이름
    int age;         // 나이를 나타내는 정수 값
    double gpa;      // 평균 평점을 나타내는 실수값
}
```

정의

```
struct studentTag s1;

strcpy(s.name, “kim”);
s.age = 20;
s.gpa = 4.3;
```

사용

```
typedef struct studentTag{
    char name[10];
    int age;
    double gpa;
} student;
```

```
student s;
S = {“kim”, 20, 4.3};
```

typedef 사용으로 간편화

구조체 기본 실습

구조체 선언과 접근 관련 예제

```
#include <stdio.h>
#include <math.h>

typedef struct point {
      int xpos;
      int ypos;
}Point;
int main(void) {
      Point pos1, pos2;
      double distance;
      printf("point1 pos: ");
      scanf_s("%d %d", &pos1.xpos, &pos1.ypos);
      printf("point2 pos: ");
      scanf_s("%d %d", &pos2.xpos, &pos2.ypos);
      distance = sqrt((double)((pos1.xpos - pos2.xpos) * (pos1.xpos - pos2.xpos) + (pos1.ypos -
         pos2.ypos) * (pos1.ypos - pos2.ypos)));
      printf("두 점의 거리는 %g 입니다. ₩n", distance);
      return 0;
}
```

배열 및 구조체의 응용

다항식

- 다항식의 수학적 일반적인 형태

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

- 프로그래밍으로 처리할 경우 2가지 방법 사용
  - 다항식의 모든 항을 배열에 저장
  - 다항식의 0이 아닌 항만을 배열에 저장

```
#define MAX(a,b) (((a)>(b))?(a):(b))
#define MAX_DEGREE 101
typedef struct {
    int degree;
    float coef[MAX_DEGREE];
} polynomial;

                        10x^5 +   6x+3
polynomial  a = { 5, {10, 0, 0, 0, 6, 3} };
```

배열 및 구조체의 응용

다항식 (다항식의 모든 항을 배열에 저장)

```
polynomial poly_add1(polynomial A, polynomial B){
  polynomial C;                    int Apos = 0, Bpos = 0, Cpos = 0;
  int degree_a = A.degree;         int degree_b = B.degree;
  C.degree = MAX(A.degree, B.degree);
  while (Apos <= A.degree && Bpos <= B.degree) {
    if (degree_a > degree_b) { // A항 > B항
      C.coef[Cpos++] = A.coef[Apos++];
      degree_a--;
    }
    else if (degree_a == degree_b) {
      C.coef[Cpos++] = A.coef[Apos++] + B.coef[Bpos++];
      degree_a--; degree_b--;
    }
    else {
      C.coef[Cpos++] = B.coef[Bpos++];
      degree_b--;
    }
  }
  return C;
}
void print_poly(polynomial p){
  for (int i = p.degree; i>0; i--)
    printf("%3.1fx^%d + ", p.coef[p.degree - i], i);
  printf("%3.1f ₩n", p.coef[p.degree]);
}
```

배열 및 구조체의 응용

다항식 (다항식의 모든 항을 배열에 저장)

```
#include <stdio.h>
#include <math.h>

int main(void){
  polynomial a = { 5,{ 3, 6, 0, 0, 0, 10 } };      //3x^5 + 6x^4 + 10
  polynomial b = { 4,{ 7, 0, 5, 0, 1 } };          //7x^4 + 5x^2 + 1
  polynomial c;
  print_poly(a);
  print_poly(b);
  c = poly_add1(a, b);
  printf("---------------------------------------------------------------------------₩n");
  print_poly(c);
  return 0;
}
```

결과: $3x^5 + 13x^4 + 5x^2 + 11$

배열 및 구조체의 응용

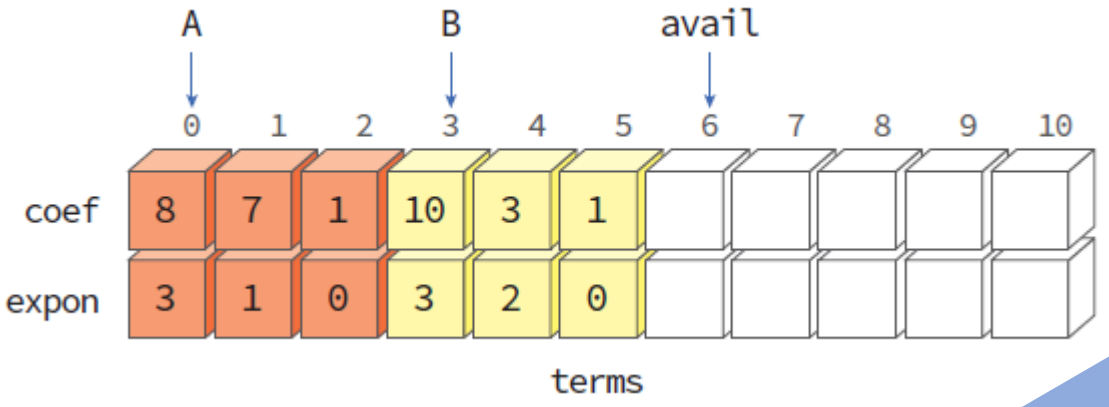
다항식

- 프로그래밍으로 처리할 경우 2가지 방법 사용
  - 다항식의 모든 항을 배열에 저장
  - 다항식의 0이 아닌 항만을 배열에 저장

$8x^3 + 7x + 1$ **((8, 3), (7, 1), (1,0))**

```
#define MAX_ TERMS 101
typedef struct {
   int expon;
   float coef;
} terms[MAX_TERMS];

int avail;
```

$10x^3 + 3x^2 + 1$ **((10, 3), (3, 2), (1,0))**



배열 및 구조체의 응용

다항식 (다항식의 0이 아닌 항만을 배열에 저장)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TERMS 101
typedef struct {
      float coef;
      int expon;
} polynomial;
polynomial terms[MAX_TERMS] = { { 8,3 },{ 7,1 },{ 1,0 },{ 10,3 },{ 3,2 },{ 1,0 } };
int avail = 6;
char compare(int a, int b){
      if (a > b) return '>';
      else if (a == b) return '=';
      else return '< ';
}
void attach(float coef, int expon){
      if (avail > MAX_TERMS) {
            fprintf(stderr, "항의 개수가 너무 많음Wn");
            exit(1);
      }
      terms[avail].coef = coef;
      terms[avail].expon = expon;
      avail++;
}
```

배열 및 구조체의 응용

다항식 (다항식의 0이 아닌 항만을 배열에 저장)

```
void poly_add2(int As, int Ae, int Bs, int Be, int* Cs, int* Ce){
      float tempcoef;              *Cs = avail;
      while (As <= Ae && Bs <= Be)
            switch (compare(terms[As].expon, terms[Bs].expon)) {
                  case '>': // A의 차수 > B의 차수
                        attach(terms[As].coef, terms[As].expon);
                        As++;break;
                  case '=': // A의 차수 == B의 차수
                        tempcoef = terms[As].coef + terms[Bs].coef;
                        if (tempcoef)
                        attach(tempcoef, terms[As].expon);
                        As++; Bs++;break;
                  case '<': // A의 차수 < B의 차수
                        attach(terms[Bs].coef, terms[Bs].expon);
                        Bs++;break;
            }
      for (; As <= Ae; As++)
            attach(terms[As].coef, terms[As].expon);
      for (; Bs <= Be; Bs++)
            attach(terms[Bs].coef, terms[Bs].expon);
      *Ce = avail - 1;
}
```

```
void print_poly(int s, int e){
      for (int i = s; i < e; i++)
            printf("%3.1fx^%d + ", terms[i].coef, terms[i].expon);
      printf("%3.1fx^%dWn", terms[e].coef, terms[e].expon);
}
```

ex5

배열 및 구조체의 응용

다항식 (다항식의 0이 아닌 항만을 배열에 저장)

```
int main(void){
      int As = 0, Ae = 2, Bs = 3, Be = 5, Cs, Ce;
      poly_add2(As, Ae, Bs, Be, &Cs, &Ce);
      print_poly(As, Ae);
      print_poly(Bs, Be);
      printf("-------------------------------------------------------------------------------
      ₩n");
      print_poly(Cs, Ce);
      return 0;
}
```

결과:

$$8x^3 + 7x + 1$$
$$10x^3 + 3x^2 + 1$$
$$18x^3 + 3x^2 + 7x + 2$$

배열 및 구조체의 응용

희소 행렬

- 희소행렬? 대부분 항들이 0인 배열
- 배열을 이용하여 행렬을 표현하는 2가지 방법

일반적인 방법

2차원 배열을 이용하여 배열의 전체 요소를 저장하는 방법
. 장점: 행렬 연산들을 간단하게 구현
. 단점: 메모리 공간 낭비

해결 →

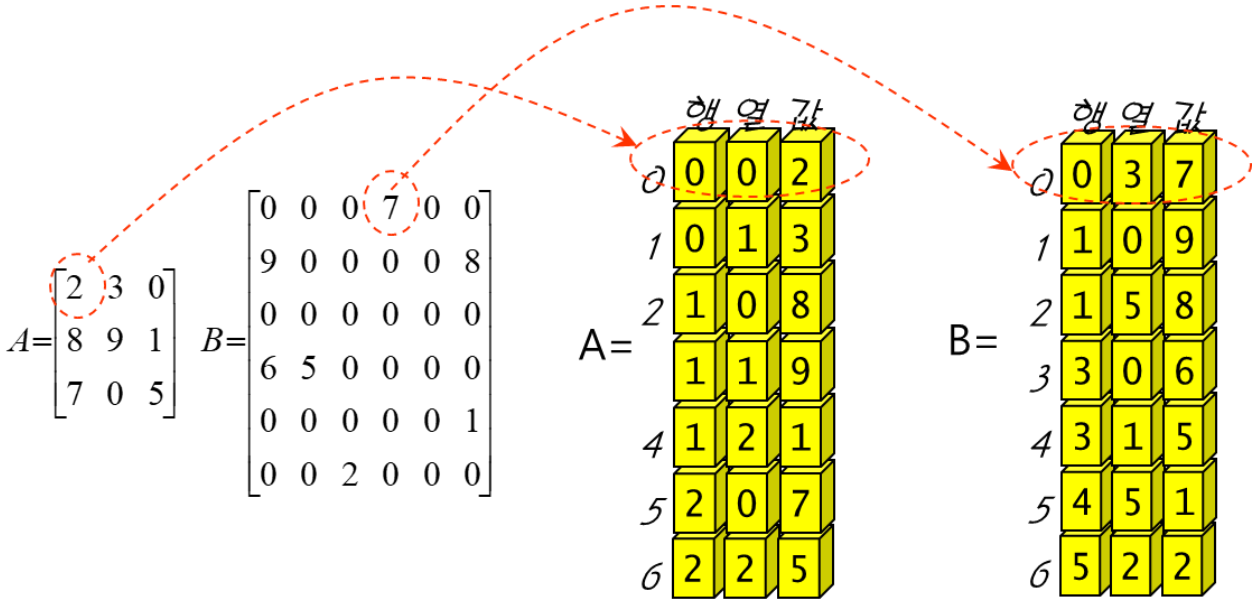
0이 아닌 요소들만 저장하는 방법

$$A=\begin{bmatrix}2 & 3 & 0\\ 8 & 9 & 1\\ 7 & 0 & 5\end{bmatrix} \quad B=\begin{bmatrix}0 & 0 & 0 & 7 & 0 & 0\\ 9 & 0 & 0 & 0 & 0 & 8\\ 0 & 0 & 0 & 0 & 0 & 0\\ 6 & 5 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 2 & 0 & 0 & 0\end{bmatrix}$$

배열 및 구조체의 응용

희소 행렬

- 희소행렬? 대부분 항들이 0인 배열
- 배열을 이용하여 행렬을 표현하는 2가지 방법
  - 2차원 배열을 이용하여 배열의 전체 요소를 저장하는 방법
  - 0이 아닌 요소들만 저장하는 방법
    . 장점: 희소 행렬의 경우, 메모리 공간 절약(아닐 경우에는...)
    . 단점: 연산 구현이 복잡



배열 및 구조체의 응용

희소행렬 (0이 아닌 요소만 저장)

```c
#include <stdio.h>
#include <math.h>
#define MAX_TERMS 101

typedef struct {
    int row;
    int col;
    int value;
} element;
typedef struct SparseMatrix {
    element data[MAX_TERMS];
    int rows;// 행의 개수
    int cols;// 열의 개수
    int terms; // 항의 개수
} SparseMatrix;
```

구조체로 저장했을 때

```c
#include <stdio.h>
#include <math.h>
#define ROWS 3
#define COLS 3
```

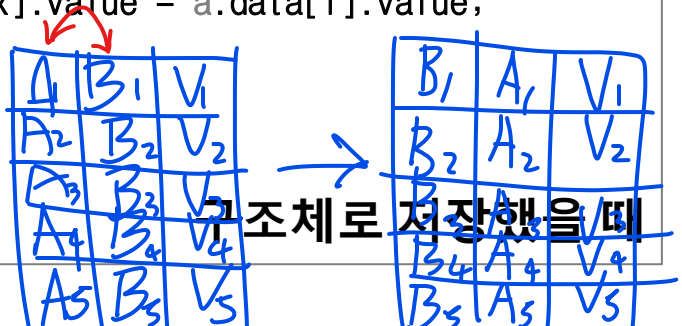
일반 행렬일 때,

배열 및 구조체의 응용

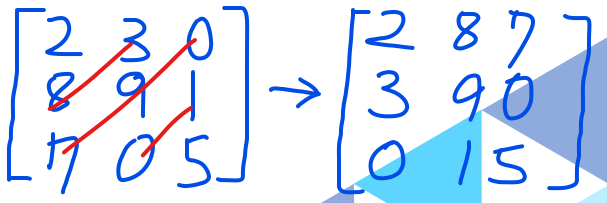
희소행렬 (0이 아닌 요소만 저장)

```
void matrix_print2(SparseMatrix a) {
  printf("==================₩n");
  for (int i = 0; i < a.terms; i++) {
    printf("(%d, %d, %d) ₩n", a.data[i].row, a.data[i].col,
a.data[i].value);
  }   printf("==================₩n");
}
SparseMatrix matrix_transpose2(SparseMatrix a) {
  SparseMatrix b;    int bindex;
  b.rows = a.rows;   b.cols = a.cols;   b.terms = a.terms;
  if (a.terms > 0) {
    bindex = 0;
    for (int c = 0; c < a.cols; c++) {
      for (int i = 0; i < a.terms; i++) {
        if (a.data[i].col == c) {
          b.data[bindex].row = a.data[i].col;
          b.data[bindex].col = a.data[i].row;
          b.data[bindex].value = a.data[i].value;
          bindex++;
        }
      }   }   }
  return b;
}
```

```
void matrix_print(int A[ROWS][COLS]){
  printf("==================₩n");
  for (int r = 0; r < ROWS; r++) {
    for (int c = 0; c < COLS; c++)
      printf("%d ", A[r][c]);
    printf("₩n");
  }
  printf("==================₩n");
}

void matrix_transpose(int A[ROWS][COLS],
int B[ROWS][COLS]) {
  for (int r = 0; r < ROWS; r++)
    for (int c = 0; c < COLS; c++)
      B[c][r] = A[r][c];
}
```



일반 행렬일 때,



ex6

배열 및 구조체의 응용

희소행렬 (0이 아닌 요소만 저장)

```
int main(void) {
  SparseMatrix m = { { { 0, 3, 7 },{ 1, 0, 9 },{ 1, 5, 8 },{ 3, 0, 6 },{ 3, 1, 5 },
                     { 4, 5, 1 },{ 5, 2, 2 } }, 6, 6, 7 };
  SparseMatrix result;
  result = matrix_transpose2(m);
  matrix_print2(result);
  return 0;
}
```

전치하기전의 원본개수

구조체로 저장했을 때

```
int main(void) {
  int array1[ROWS][COLS] = { { 2,3,0 },{ 8,9,1 },{ 7,0,5 } };
  int array2[ROWS][COLS];

  matrix_transpose(array1, array2);
  matrix_print(array1);
  matrix_print(array2);
  return 0;
}
```

일반 행렬일 때,

배열 및 구조체의 응용

포인터

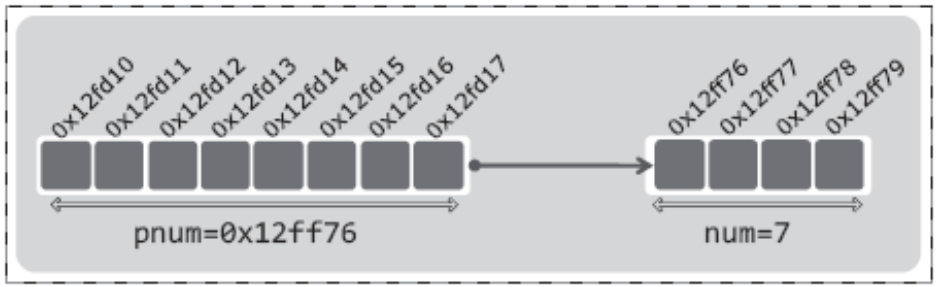
- 포인터 변수? 기존 변수 앞에 *를 붙여 사용
- 메모리의 주소 값을 저장하기 위한 변수
- & 주소 값 반환 시 사용

```
int num = 7;

int * pnum;

pnum = &num;
```

int 자료형을 가지는 변수에 7이라는 숫자 초기화

int 자료형을 가지는 포인터 변수 pnum을 선언

num 변수의 주소 값을 pnum에 반환



배열 및 구조체의 응용

포인터

- 포인터 변수는 주소 값을 나타내므로 저장은 모두 정수 값
- 하지만 가리키고자 하는 변수는 자료형에 따라 포인터 변수 선언 방법이 다름

```
int * pnum1;
```

int * 는 int형 변수를 가리키는 pnum1의 선언을 의미함

```
double * pnum2;
```

double * 는 double형 변수를 가리키는 pnum2의 선언을 의미함

```
unsigned int * pnum3;
```

unsigned int * 는 unsigned int형 변수를 가리키는 pnum3의 선언을 의미함

일반화

```
type * ptr;
```

type형 변수의 주소 값을 저장하는 포인터 변수 ptr의 선언

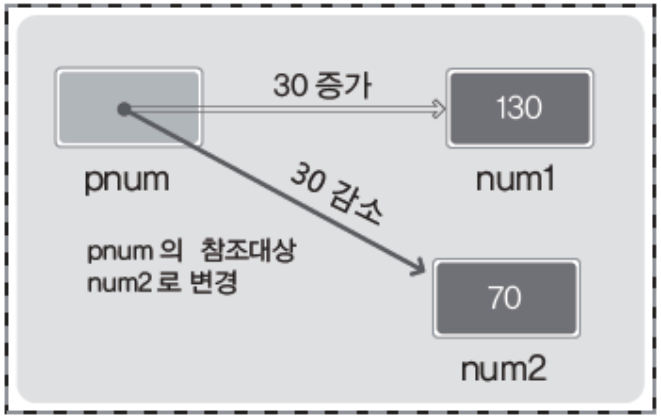
배열 및 구조체의 응용

포인터 기본 예제

```c
#include <stdio.h>

int main(void){
  int num1 = 100, num2 = 100;
  int* pnum;

  pnum = &num1;
  (*pnum) += 30;

  pnum = &num2;
  (*pnum) -= 30;

  printf("num1:%d, num2:%d ₩n", num1, num2);
  return 0;
}
```

num1 = 130

num2 = 70



배열 및 구조체의 응용

포인터 (잘못된 포인터 사용)

- 포인터에 값을 직접 삽입하는 경우

```
int main(void)
{
     int * ptr;
     *ptr=200;
     . . . .
}
```

```
int main(void)
{
     int * ptr=125;
     *ptr=10;
     . . . .
}
```

200, 125의 저장되는 위치를 확인할 방법 없음

10도 마찬가지

- 포인터에 값 초기화 방법 (특정 값으로 초기화하지 않는 경우)

int * ptr1 = 0 OR int * ptr1 = NULL

- 특정 값으로 초기화 하지 않을 경우에는 “값이 없다” 라는 의미로 초기화
- C 언어의 “값이 없다” 는 0 또는 NULL 로 사용

배열 및 구조체의 응용

배열과 포인터

- 포인터 변수: 주소 값을 가리키는 변수
- 배열: 연속된 주소 값에 자료형에 따른 값을 삽입하는 변수

- 어떤 연관이...???
  - 배열 이름 = 포인터
  - 배열 요소 * 연산자로 접근 가능
  - 배열 이름과 포인터 변수 비교

비교대상 / 비교조건	포인터 변수	배열의 이름
이름의 존재?	존재	존재
무엇을 나타내거나 저장?	메모리 주소 값	메모리 주소 값
주소 값 변경 가능?	가능	불가능

배열 및 구조체의 응용

배열과 포인터 증감 연산

- 배열에서의 각 요소 접근할 수 있는 증감 연산
- arr[i] == *(arr+i)

*(ptr+0)	*(ptr+1)	*(ptr+2)		*(ptr+i)
=*ptr				
=	=	=	•••	=
arr[0]	arr[1]	arr[2]		arr[i]

- 포인터를 사용해서 값 증감

```
(*ptr)+1;
(*ptr)++;
```

배열 및 구조체의 응용

배열과 포인터

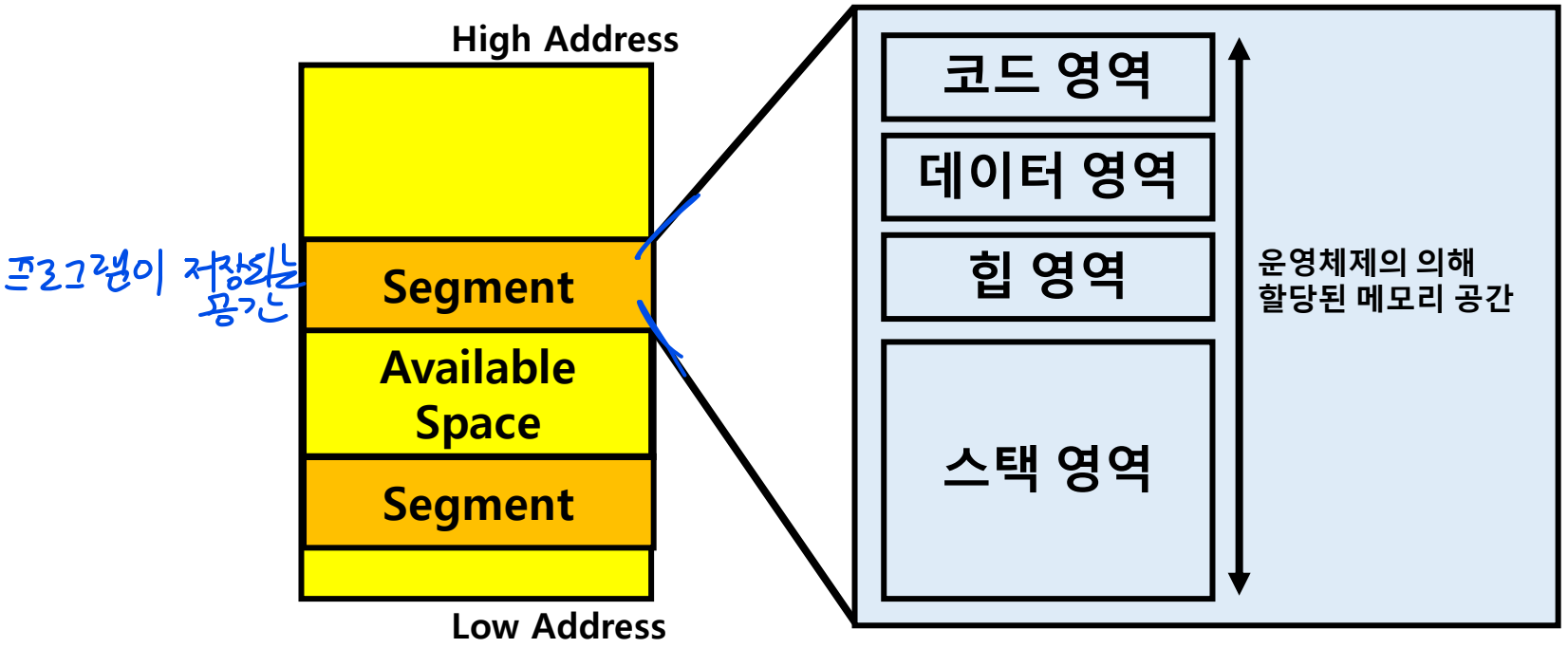
```c
#include <stdio.h>
#define SIZE 6
void get_integers(int list[]) {
    printf("6개의 정수를 입력하시오: ");
    for (int i = 0; i < SIZE; ++i) {
        scanf_s("%d", &list[i]);
    }
}
int cal_sum(int list[]) {
    int sum = 0;
    for (int i = 0; i < SIZE; ++i) {
        sum += *(list + i);
    }
    return sum;
}
int main(void) {
    int list[SIZE];
    get_integers(list);
    printf("합 = %d ₩n", cal_sum(list));
    return 0;
}
```

list[i]

배열 및 구조체의 응용

동적 메모리

- 프로그램 실행 시 마련되는 메모리 구조



배열 및 구조체의 응용

동적 메모리 (상태 변화 - 전역변수)

```
#include <stdio.h>

int sum=25;

int main(void){
        int num1 =10;
        fct(num1);
        num1++;
        fct(num1);
        return 0;
}

void fct(int n){
        int num2=12;
}
```

배열 및 구조체의 응용

동적 메모리 (상태 변화 - 지역변수)

```
#include <stdio.h>

int sum=25;

int main(void){
        int num1 =10;
        fct(num1);
        num1++;
        fct(num1);
        return 0;
}

void fct(int n){
        int num2=12;
}
```

배열 및 구조체의 응용

동적 메모리 (상태 변화 - 함수호출)

```
#include <stdio.h>

int sum=25;

int main(void){
        int num1 =10;
        fct(num1);
        num1++;
        fct(num1);
        return 0;
}

void fct(int n){
        int num2=12;
}
```

num1을 받아옴

배열 및 구조체의 응용

동적 메모리 (상태 변화 - 지역변수)

```
#include <stdio.h>

int sum=25;

int main(void){
        int num1 =10;
        fct(num1);
        num1++;
        fct(num1);
        return 0;
}

void fct(int n){
        int num2=12;
}
```

배열 및 구조체의 응용

동적 메모리 (상태 변화 – fct 함수/지역변수 반환)

```
#include <stdio.h>

int sum=25;

int main(void){
        int num1 =10;
        fct(num1);
        num1++;
        fct(num1);
        return 0;
}

void fct(int n){
        int num2=12;
}
```

함수 종료

배열 및 구조체의 응용

동적 메모리 (상태 변화 – fct 함수호출)

```
#include <stdio.h>

int sum=25;

int main(void){
        int num1 =10;
        fct(num1);
        num1++;
        fct(num1);
        return 0;
}

void fct(int n){
        int num2=12;
}
```

배열 및 구조체의 응용

동적 메모리 (상태 변화 – fct 함수반환)

```
#include <stdio.h>

int sum=25;

int main(void){
        int num1 =10;
        fct(num1);
        num1++;
        fct(num1);
        return 0;
}

void fct(int n){
        int num2=12;
}
```

배열 및 구조체의 응용

동적 메모리 (상태 변화 – 프로그램 종료)

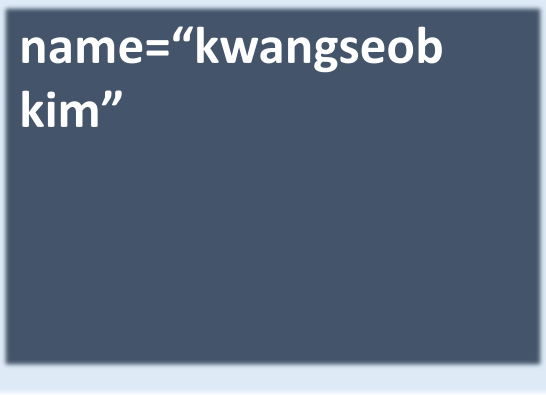
```
#include <stdio.h>

int sum=25;

int main(void){
        int num1 =10;
        fct(num1);
        num1++;
        fct(num1);
        return 0;
}

void fct(int n){
        int num2=12;
}
```

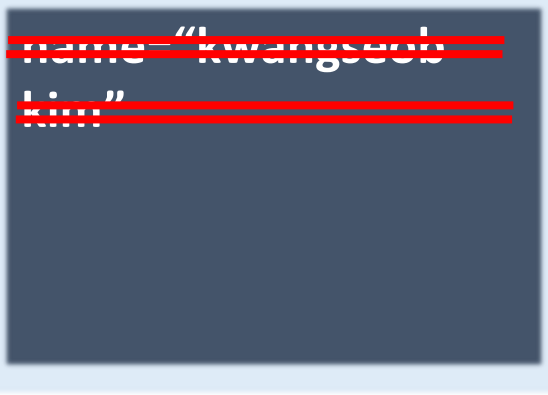
배열 및 구조체의 응용

동적 메모리 (지역 변수 한계점) 함수가 끝나면 사라짐

```
#include <stdio.h>

char * ReadUserName(void)
{
        char name[30];
        printf("What's your name? ");
        gets(name);
        return name;
}
```

```
int main(void)
{
        char * name1;
        char * name2;
        name1=ReadUserName();
        printf("name1: %s ₩n", name1);
        name2=ReadUserName();
        printf("name2: %s ₩n", name2);
        return 0;
}
```





배열 및 구조체의 응용

둘은 다른 포인터 변수이지만,

동적 메모리 (전역 변수 한계점)

같은 주소값을 바라보고 있어서

```
#include <stdio.h>

char name[30];

char * ReadUserName(void)
{
	printf("What's your name? ");
	gets(name);
	return name;
}
```

```
int main(void)
{
	char * name1;
	char * name2;
	name1=ReadUserName();
	printf("name1: %s ₩n", name1);
	name2=ReadUserName();
	printf("name2: %s ₩n", name2);

	printf("name1: %s ₩n", name1);
	printf("name2: %s ₩n", name2);
return 0;
}
```

데이터가 똑같은 이름으로 저장됨

왜 이런 문제점이 생기는 것일까??

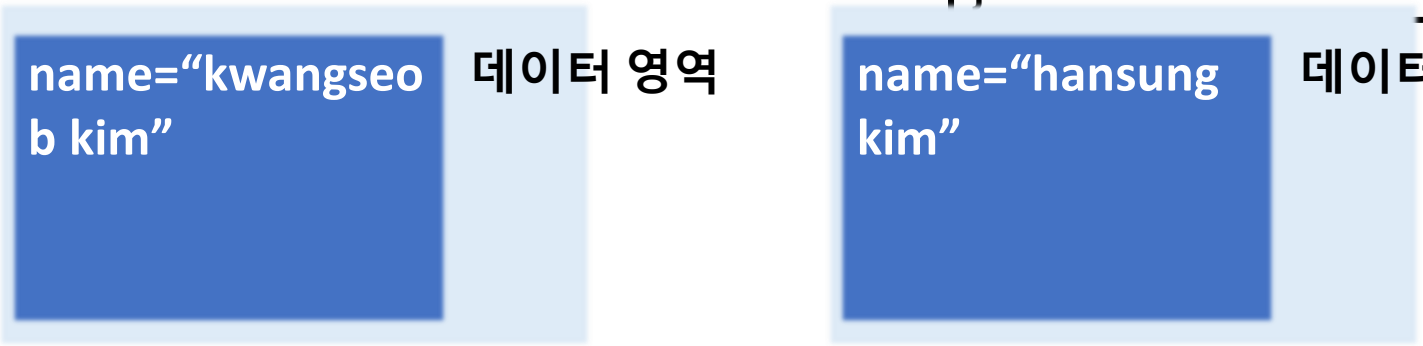
```
What's your name? kwangseob kim
name1: kwangseob kim
What's your name? hansung kim
name2: hansung kim
name1: hansung kim
name2: hansung kim
```

배열 및 구조체의 응용

동적 메모리 (전역 변수 한계점)

```
#include <stdio.h>

char name[30];

char * ReadUserName(void)
{
        printf("What's your name? ");
        gets(name);
        return name;
}
```

```
int main(void)
{
        char * name1;
        char * name2;
        name1=ReadUserName();
        printf("name1: %s ₩n", name1);
        name2=ReadUserName();
        printf("name2: %s ₩n", name2);

        printf("name1: %s ₩n", name1);
        printf("name2: %s ₩n", name2);
return 0;
}
```



배열 및 구조체의 응용

동적 메모리

- 데이터 영역과 스택 영역에서 제한적인 변수 정의 부분이 생기는 경우가 있음

- 이를 힙 영역을 통해 해결 가능

> • 프로그래머가 원하는 시점에 메모리 공간에 할당 및 소멸하기 위한 영역

배열 및 구조체의 응용

동적 메모리 할당 / 해제

```
#include <stdlib.h>
void * malloc(size_t size);       // 힙 영역으로의 메모리 공간 할당
void free(void * ptr);     // 힙 영역에 할당된 메모리 공간 해제
```

➡ malloc 함수는 성공 시 할당된 메모리의 주소 값, 실패 시 NULL 반환

malloc & free 함수 호출의 기본 모델

```
int main(void)
{
    void * ptr1 = malloc(4);     // 4바이트가 힙 영역에 할당
    void * ptr2 = malloc(12);    // 12바이트가 힙 영역에 할당
    . . . .
    free(ptr1);    // ptr1이 가리키는 4바이트 메모리 공간 해제
    free(ptr2);    // ptr2가 가리키는 12바이트 메모리 공간 해제
    . . . .
}
```

배열 및 구조체의 응용

동적 메모리 할당 / 해제

```
#include <stdlib.h>
void * malloc(size_t size);       // 힙 영역으로의 메모리 공간 할당
void free(void * ptr);    // 힙 영역에 할당된 메모리 공간 해제
```

➜ malloc 함수는 성공 시 할당된 메모리의 주소 값, 실패 시 NULL 반환

- Malloc 함수는 void 형 포인터를 반환
- Void 형 포인터란? 어떠한 주소 값도 저장이 가능한 포인터

```
void * ptr1 = malloc(sizeof(int));
void * ptr2 = malloc(sizeof(double));
void * ptr3 = malloc(sizeof(int)*7);
void * ptr4 = malloc(sizeof(double)*9);
```

배열형

```
void * ptr1 = malloc(4);
void * ptr2 = malloc(8);
void * ptr3 = malloc(28);
void * ptr4 = malloc(72);
```

malloc 함수의 일반적인 호출형태

sizeof 연산 이후 실질적인 malloc의 호출

```
int * ptr1 = (int *)malloc(sizeof(int));
double * ptr2 = (double *)malloc(sizeof(double));
int * ptr3 = (int *)malloc(sizeof(int)*7);
double * ptr4 = (double *)malloc(sizeof(double)*9);
```

정수형을 집어넣을 것이다.

- 포인터 형의 변환을 통해 직접 결정

배열 및 구조체의 응용

동적 메모리 할당 / 해제

```c
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int* ptr1 = (int*)malloc(sizeof(int));
    int* ptr2 = (int*)malloc(sizeof(int) * 7);
    int i;
    *ptr1 = 20;
    for (i = 0; i < 7; i++)
        ptr2[i] = i + 1;
    printf("%d ₩n", *ptr1);

    for (i = 0; i < 7; i++)
        printf("%d ", ptr2[i]);

    free(ptr1);
    free(ptr2);
    return 0;
}
```

배열 및 구조체의 응용

동적 메모리 할당 / 해제

- 동적 할 당을 실패한 경우 NULL을 반환 함

- 그러므로 할당 성공 여부를 확인할 필요가 있으므로 아래 코드로 성공 여부 확인

```
int * ptr = (int *)malloc(sizeof(int));
if(ptr==NULL){
        ....
}
```

- 왜 동적 할당? 이라고 하는가?
  - 컴파일 시 할당에 필요한 메모리 계산을 하지 않기 때문에

배열 및 구조체의 응용

구조체와 포인터

- 구조체 변수를 포인터 형태로 사용

- 선언 방법 = 일반 포인터 변수 선언과 동일
  - 구조체 명 *p
- 사용 시 (*p).i 보다는 ps->i (화살표로 사용함)

배열 및 구조체의 응용

구조체와 포인터

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct studentTag {
  char name[10]; // 문자배열로 된 이름
  int age; // 나이를 나타내는 정수값
  double gpa; // 평균평점을 나타내는 실수값
} student;
int main(void) {
  student* p;
  p = (student*)malloc(sizeof(student));
  if (p == NULL) {
    fprintf(stderr, "메모리가 부족해서 할당할 수 없습니다.₩n");
    exit(1);
  }
  strcpy(p->name, "Park");
  p->age = 20;
  free(s);
  return 0;
}
```

ex10