3월2주차 개념정리

상태	완료
데드라인	@March 28, 2025
PROCESS	💚 데이터구조

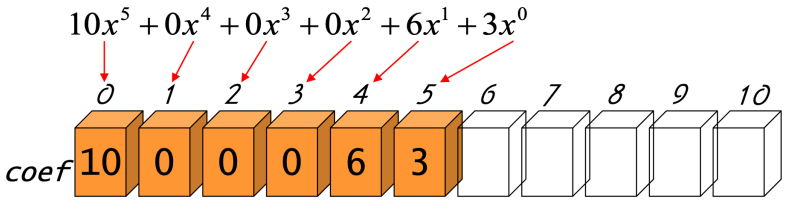
▼ 배열

- 같은 형의 변수를 여러 개 만드는 경우에 사용
- 객체 : <인덱스, 값> 쌍의 집합
- 연산
  - create(size) :: = size개의 요소를 저장할 수 있는 배열 생성
  - get(A, i) ::= 배열 A의 i번째 요소 반환
  - set(A, i, v) ::= 배열 A의 i번째 위치에 값 v 저장

▼ 다항식 응용

프로그램에서 다항식을 처리하려면 다항식을 위한 자료구조가 필요함, 어떤 자료구조가 연산시 편리하고 효율적일까?

▼ 다항식의 모든 항을 배열에 저장




```
#include <stdio.h>
#define MAX(a, b) (((a)>(b))?(a):(b))
#define MAX_DEGREE 101

typedef struct{
    int degree;
    float coef[MAX_DEGREE];
} polynomial;

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); //결과 다항식 차수

    while(Apos <= A.degree && Bpos <= B.degree){
        if(degree_a > degree_b){
            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]);
}
int main(){
    polynomial a = {5,{3,6,0,0,0,10}};
    polynomial b = {4, {7,0,5,0,1}};
    polynomial c;
    print_poly(a);
    print_poly(b);
    c = poly_add1(a, b);
    printf("------------------------\n");
    print_poly(c);
    return 0;
}
```

▼ 다항식의 0이 아닌 항만을 배열에 저장

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

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TERM 101

typedef struct{
   float coef;
   int expon;
}polynomial;

polynomial terms[MAX_TERM] = {{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_TERM){
      fprintf(stderr, "항의 개수가 너무 많음\n");
      exit(1);
   }
   terms[avail].coef = coef;
   terms[avail].expon = expon;
   avail++;
}

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 '>':
                attach(terms[As].coef ,terms[As].expon);
                As++; break;
            case '=':
                tempcoef = terms[As].coef + terms[Bs].coef;
                if(tempcoef)
                    attach(tempcoef, terms[As].expon);
                As++; Bs++; break;
            case '<':
                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^%d \n", terms[e].coef, terms[e].expon);
}
int main(){
    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;
}
```

▼ 희소행렬

: 대부분의 항들이 0인 배열

▼ 2차원 배열을 이용하여 배열의 전체요소를 저장하는 방법

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

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];
   }
}

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

   matrix_transpose(array, array2);
   matrix_print(array);
   matrix_print(array2);

   return 0;
}
```

▼ 0이 아닌 요소들만 저장하는 방법

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

#define MAX_TERMS 100
```

```
typedef struct{
    int row, col, value;
}element;
typedef struct {
    element data[MAX_TERMS];
    int rows, cols, terms;
}SpareMatrix;

SpareMatrix matrix_transpose2(SpareMatrix a){
    SpareMatrix b;

    int bindex;
    b.rows=a.cols;
    b.cols=a.rows;
    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(SpareMatrix 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");
}

int main(){
    SpareMatrix 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
```

```
    };
    SpareMatrix result;

    result = matrix_transpose2(m);
    matrix_print(result);
    return 0;
}
```

▼ 구조체

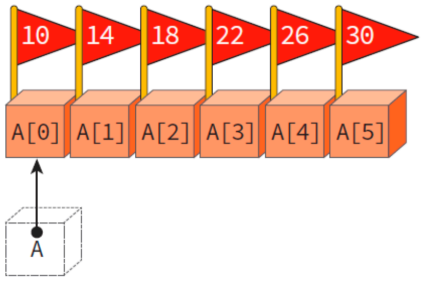
- structure : 타입이 다른 데이터를 하나로 묶는 방법
- ⇒ 배열은 타입이 같은 데이터를 하나로 묶음

▼ Pointer

다른 변수의 주소를 가지고 있는 변수

- 포인터가 가리키는 내용의 변경 ⇒ `*`
- & 연산자 : 변수의 주소를 추출
- * 연산자 : 포인터가 가리키는 곳의 내용을 추출
- 함수 안에서 매개변수로 전달된 포인터를 이용하여 외부 변수의 값을 변경할 수 있음

▼ 배열의 이름 : 사실상의 포인터와 같은 역할




```
#include <stdio.h>
#define SIZE 6

void get_integers(int list[]){
   printf("6개의 정수를 입력하세요 : ");
   for(int i = 0 ; i < SIZE; i++)
      scanf("%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(){
    int list[SIZE];
    get_integers(list);
    printf("합 = %d\n", cal_sum(list));
    return 0;
}
```

▼ 동적메모리 할당

- 프로그램의 실행 도중에 메모리를 할당 받는 것
- 필요한 만큼만 할당을 받고 또 필요한 때에 사용하고 반납
- 메모리를 매우 효율적으로 사용 가능

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

#define SIZE 10

int main(){
    int *p;
    p = (int *)malloc(SIZE * sizeof(int));
    if(p == NULL){
        fprintf(stderr, "메모리가 부족해서 할당할 수 없습니다. \n");
        exit(1);
    }

    for(int i = 0 ; i < SIZE; i++)
        p[i] = i;

    for(int i = 0; i < SIZE; i++)
        printf("%d ", p[i]);
```

```
    free(p);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

int main(){
    student *s;
    s=(student *)malloc(sizeof(student));
    if(s==NULL){
        fprintf(stderr, "메모리 존나 부족");
        exit(1);
    }
    strcpy(s→name, "PARK");
    s→age = 20;
    free(s);
    return 0;
}
```