데이터 구조 3장 실습과제

20223100 박신조

```c
//실습과제 p.75 프로그램 3.1 구조체 선언
#include <stdio.h>

typedef struct studentTag{ //구조체 선언
  char name[10];
  int age;
  double gpa;
} student; // struct studentTag -> student로 변환

int main(void){
  student a = {"kim", 20, 4.3};// 구조체 a, b 선언
  student b = {"park", 21, 4.2};

  // 구조체 출력
  printf("이름 : %s, 나이: %d, 평균평점: %f\n", a.name,a.age, a.gpa);
  printf("이름 : %s, 나이: %d, 평균평점: %f\n", b.name,b.age, b.gpa);
  return 0;
}
```

```
./main
이름 : kim, 나이: 20, 평균평점: 4.300000
이름 : park, 나이: 21, 평균평점: 4.200000
```

코드설명

#include <stdio.h>

stdio.h는 표준 입출력 함수들이 선언되어 있는 헤더 파일로, printf와 같은 입출력 함수들을 사용하기 위해 포함시킵니다.

typedef struct studentTag{ ... } student;

typedef는 새로운 데이터 타입을 정의하는 키워드입니다.
struct studentTag는 구조체(struct)의 선언 부분으로, name, age, gpa라는 필드(변수)를 가집니다.
student는 struct studentTag의 별칭(alias)입니다. 즉, student는 이제 struct studentTag와 같은 의미로 사용될 수 있습니다.

int main(void)

main 함수는 컴파일 후 가장 먼저 실행되는 함수입니다.

student a = {"kim", 20, 4.3};

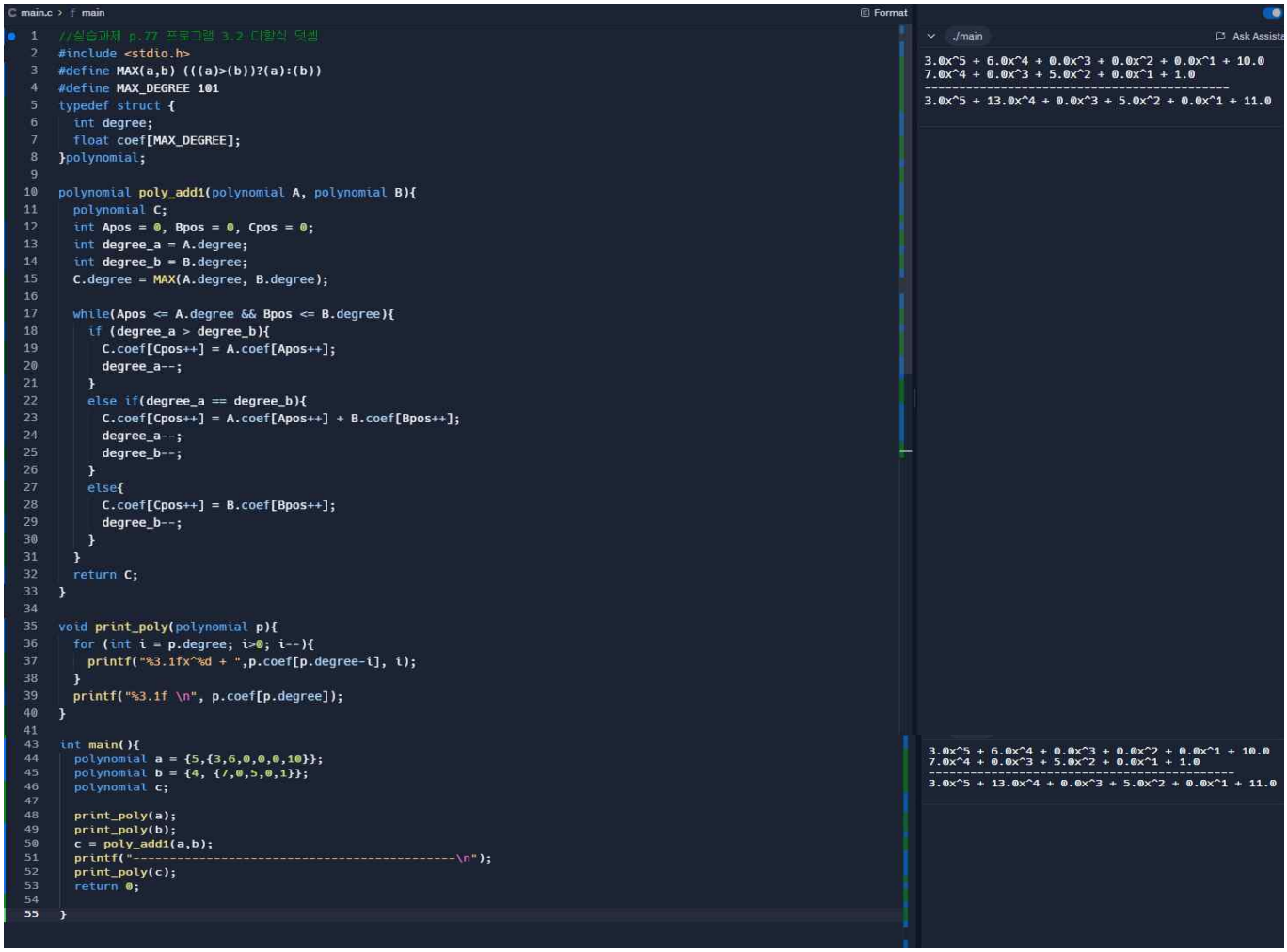
student 타입의 변수 a를 선언하고 초기화합니다.
a는 a.name "kim", a.age 20, a.gpa 4.3으로 초기화

printf("이름 : %s, 나이: %d, 평균평점: %f\n", a.name, a.age, a.gpa);

printf는 화면에 출력하는 함수입니다.
%s는 문자열, %d는 정수형 변수, %f는 실수형 변수를 출력할 때 사용됩니다.
구조체를 출력할 때 “%d”, a.age 형식으로 자료형에 맞게 출력해야 합니다.
(함수에서 선언한 구조체 변수).(구조체 필드에서 생성된 변수)

코드 실행 시 순서:

1. student라는 구조체를 정의하고, a와 b라는 두 학생의 정보를 초기화합니다.
2. printf를 이용하여 각 학생의 이름, 나이, 평균 평점을 출력합니다.



```c
//실습과제 p.77 프로그램 3.2 다항식 덧셈
#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);

  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;

}
```

```
./main
3.0x^5 + 6.0x^4 + 0.0x^3 + 0.0x^2 + 0.0x^1 + 10.0
7.0x^4 + 0.0x^3 + 5.0x^2 + 0.0x^1 + 1.0
-----------------------------------------
3.0x^5 + 13.0x^4 + 0.0x^3 + 5.0x^2 + 0.0x^1 + 11.0
```


```
3.0x^5 + 6.0x^4 + 0.0x^3 + 0.0x^2 + 0.0x^1 + 10.0
7.0x^4 + 0.0x^3 + 5.0x^2 + 0.0x^1 + 1.0
-----------------------------------------
3.0x^5 + 13.0x^4 + 0.0x^3 + 5.0x^2 + 0.0x^1 + 11.0
```


코드설명

#define MAX(a,b) (((a)>(b))?(a):(b))
#define MAX_DEGREE 101

#define A B 는 코드에서 A를 B로 바꿔주는 매크로입니다.
#define MAX(a,b) (((a)>(b))?(a):(b))는 a 와 b 의 값을 비교해서 더 큰 값을 반환해주는 매크로입니다.
#define MAX_DEGREE 101는 MAX_DEGREE를 호출하면 101를 반환해주는 매크로입니다.

typedef struct { ... } polynomial;

polynomial 구조체는 다항식을 표현하는 구조체입니다.
다항식은 차수(degree)와 각 차수에 해당하는 계수(coefficient)들로 구성됩니다.
그렇기 때문에 구조체 필드에는 degree(다항식의 차수), coef[MAX_DEGREE](다항식의 계수)를 선언해 줍니다.
ex) 다항식이 5x^2-30x+45 일 때 5x^2d의 차수는 2 이고 계수는 5이다.

polynomial poly_add1(polynomial A, polynomial B)

poly_add1 함수는 두 다항식 A와 B를 더하는 함수입니다.
두 다항식을 더하기 위해 각 다항식의 차수가 높은 항부터 비교하여 더해줍니다.
Apos, Bpos, Cpos는 각각 A, B, C 다항식의 계수를 나타내는 인덱스 변수입니다.
degree_a와 degree_b는 A와 B의 차수를 나타냅니다.
C.degree = MAX(A.degree, B.degree)는 결과 다항식 C의 차수를 결정합니다.
두 다항식 중 더 큰 차수를 C의 차수로 설정합니다.

while 반복문 설명:

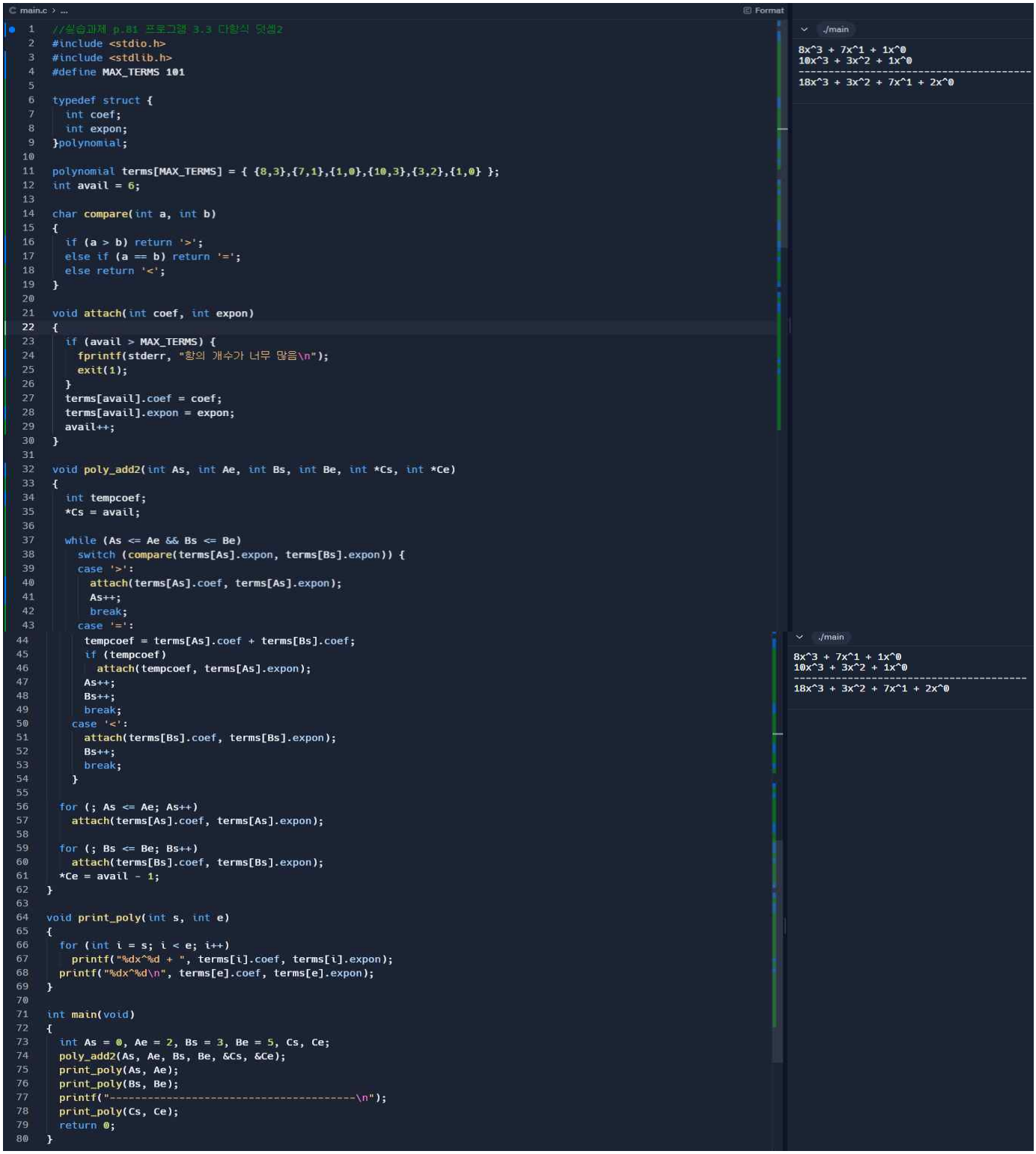
if 조건문의 역할은 A와 B의 각 차수를 비교하면서 더합니다.
차수가 더 큰 항은 그 항을 결과 다항식에 추가하고, 해당 항을 A 또는 B에서 하나씩 내려가며 처리합니다.
차수가 같으면 A와 B에서 해당 항을 더하여 결과 다항식에 저장합니다.
차수가 작은 항이 남으면, 해당 항을 그대로 결과 다항식에 추가합니다.

void print_poly(polynomial p)

print_poly 함수는 다항식을 출력하는 함수입니다.
for 루프는 p.degree(최고 차수)에서 1까지 내려가며 각 차수의 계수를 출력합니다.

코드 실행 시 순서:

1. 두 다항식을 구조체로 정의하고, 각 다항식의 차수를 degree, 계수를 coef[]에 저장합니다.
2. poly_add1 함수를 통해 두 다항식(A,B)을 더하고, 그 결과를 다항식(C)에 저장합니다.
3. print_poly 함수를 사용하여 최고 차수부터 계수를 차례대로 출력합니다.



```c
//실습과제 p.81 프로그램 3.3 다항식 덧셈2
#include <stdio.h>
#include <stdlib.h>
#define MAX_TERMS 101

typedef struct {
  int 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(int coef, int expon)
{
  if (avail > MAX_TERMS) {
    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)
{
  int 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("%dx^%d + ", terms[i].coef, terms[i].expon);
  printf("%dx^%d\n", terms[e].coef, terms[e].expon);
}

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;
}
```

```
8x^3 + 7x^1 + 1x^0
10x^3 + 3x^2 + 1x^0
----------------------------------------
18x^3 + 3x^2 + 7x^1 + 2x^0
```

코드설명

#include <stdlib.h>

stdlib.h: exit 함수와 같은 메모리 관련 함수들을 사용하기 위해 포함합니다.

typedef struct { ... } polynomial;

polynomial 구조체는 다항식의 한 항을 나타냅니다.

coef: 계수 (예: 3x^2에서 3), expon: 지수 (예: 3x^2에서 2)

polynomial terms[MAX_TERMS] = { {8,3},{7,1},{1,0},{10,3},{3,2},{1,0} };

구조체 배열은 배열 1칸마다 구조체 변수(polynomial a;)와 같은 역할을 합니다.

terms 배열은 다항식의 항들을 저장하는 배열입니다. 각 항은 계수와 지수로 이루어집니다.

ex) {8,3} 8x^3으로 저장

char compare(int a, int b)

compare 함수는 두 지수 a와 b를 비교하여, a > b이면 '>'를, a == b이면 '='를, a < b이면 '<'를 반환합니다.

void attach(int coef, int expon)

attach 함수는 새로운 항을 terms 배열에 추가해주는 함수입니다.
avail 값이 MAX_TERMS를 초과하면 오류 메시지를 출력하고 프로그램을 종료합니다.
다음에 또 새로운 항이 추가 될 수 있도록 avail 값을 증가시켜 줍니다.

void poly_add2(int As, int Ae, int Bs, int Be, int *Cs, int *Ce)

poly_add2 함수는 두 개의 다항식 구간을 더하는 함수입니다.
As, Ae: 첫 번째 다항식의 시작 인덱스와 끝 인덱스
Bs, Be: 두 번째 다항식의 시작 인덱스와 끝 인덱스
*Cs, *Ce: 결과 다항식의 시작 인덱스와 끝 인덱스를 저장해줄 포인터

switch 문을 사용하여 char compare 함수를 호출해 다항식의 항을 비교하여 case를 나눠 줍니다.
> (지수가 더 큰 경우): 더 큰 지수를 가진 항을 결과에 추가
= (지수가 같은 경우): 두 항의 계수를 더하여 결과에 추가합니다. (합계가 0일 경우는 제외)
< (지수가 더 작은 경우): 더 작은 지수를 가진 항을 결과에 추가

for문을 통해 attach 함수를 호출하여 나머지 항들도 차례대로 결과에 추가합니다.

void print_poly(int s, int e)

print_poly 함수는 다항식의 항을 출력하는 함수입니다.
s: 시작 인덱스, e: 끝 인덱스

int main(void)

두 다항식을 구분해 주기 위해 As,Ae,Bs,Be를 정수형으로 선언
As = 0, Ae = 2: 첫 번째 다항식의 시작 인덱스와 끝 인덱스
Bs = 3, Be = 5: 두 번째 다항식의 시작 인덱스와 끝 인덱스
poly_add2(As, Ae, Bs, Be, &Cs, &Ce)함수를 호출하여 두 다항식을 더하고 새로운 다항식의 위치(시작:Cs, 끝Ce)를 저장합니다.
print_poly(As, Ae)와 print_poly(Bs, Be)는 첫 번째 다항식과 두 번째 다항식을 출력합니다.
print_poly(Cs, Ce)는 두 다항식을 더한 결과 다항식을 출력합니다.

코드 실행 시 순서:

1. 두 개의 다항식이 섞여있는 terms 배열과 두 다항식의 길이 avail가 전역변수로 선언됩니다.
2. main 함수에 있는 지역변수(다항식들의 시작, 끝 인덱스)와 함수를 선언하여 다항식의 덧셈을 진행합니다.
3. poly_add2 함수를 호출하여 두 다항식의 각 항을 char compare 함수를 통해 비교하고 attach 함수를 통해 새로운 결과를 저장합니다.
4. 이후 새로운 다항식의 위치를 포인터를 통해 받아 print_poly 함수를 통해 각 다항식과 결과 다항식을 출력합니다.

```c
//실습과제 p.85 프로그램 3.4 행렬 전치 프로그램
#include <stdio.h>
#define ROW 3
#define COL 3

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

void matrix_print(int A[ROW][COL])
{
  printf("====================\n");
  for (int r = 0; r < ROW; r++) {
    for (int c = 0; c < COL; c++)
      printf("%d ",A[r][c]);
    printf("\n");
  }
  printf("====================\n");
}

int main()
{
  int array1[ROW][COL] = { {2,3,0},{8,9,1},{7,0,5} };
  int array2[ROW][COL];

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

```
./main
====================
2 3 0
8 9 1
7 0 5
====================
====================
2 8 7
3 9 0
0 1 5
====================
```

행렬과 전치행렬

행렬이란

수 또는 다항식 등을 직사각형 모양으로 배열한 것
행렬의 가로줄을 행, 세로줄을 열 이라고 표현함

전치 행렬이란

주어진 행렬의 행과 열을 서로 바꾸는 것

2	3	0
8	9	7
7	0	5

3행 3열의 A행렬

→

2	8	7
3	9	0
0	7	5

A행렬의 전치행렬

코드설명

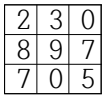
#define ROW 3와 #define COL 3

ROW는 행렬의 행 수 3행을 정의하고, COL은 열 수 3열 정의합니다.
이 경우, 3x3 크기의 행렬이라고 가정할 수 있습니다.

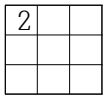
void matrix_transpose(int A[ROW][COL], int B[ROW][COL])

matrix_transpose 함수는 행렬 A를 전치하여 그 결과를 행렬 B에 저장하는 함수입니다.
A는 기본 행렬, B는 전치된 행렬입니다. ※이때 B[r][c] = A[r][c]로 하면 A행렬과 B행렬이 같게 복사됨
이중 for문을 사용하여 행렬 A의 각 요소 A[r][c]를 전치하여 행렬 B의 위치 B[c][r]에 저장합니다.

ex) B[0][0] = A[0][0]

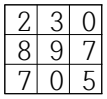


A행렬

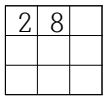


B행렬

B[1][0] = A[0][1]

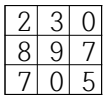


A행렬

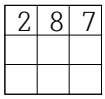


B행렬

B[2][0] = A[0][2]

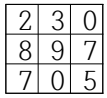


A행렬

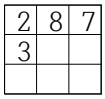


B행렬

B[0][1] = A[1][0]



A행렬



B행렬

void matrix_print(int A[ROW][COL])

matrix_print 함수는 주어진 행렬을 출력하는 함수입니다.
이중 for문을 사용하여 행렬의 각 요소를 출력합니다.

		array1	array2
ex)	r = 0일 때 c는 0,1,2 -> A[0][0]A[0][1]A[0][2]	출력: 2 3 0	2 8 7
	r = 1일 때 c는 0,1,2 -> A[1][0]A[1][1]A[1][2]	출력: 8 9 7	3 9 0
	r = 1일 때 c는 0,1,2 -> A[2][0]A[2][1]A[2][2]	출력: 7 0 5	0 7 5

코드 실행 시 순서:

1. 2차원 배열 array1에 ROWxCOL크기의 행렬 값 저장 후 전치행렬을 저장할 array2 배열 선언합니다.
2. matrix_transpose 함수를 호출하여 array1 행렬을 array2에 전치하여 저장합니다.
   이때 main함수에서 matrix_transpose 함수를 호출할 때 &array2 인자를 주지 않는 이유는
   c언어에선 배열이 포인터와 같은 역할을 함으로 &,* 사용 없이도 matrix_transpose 함수 내에서 A배열이 B배열에 영향을 준 것을
   array1 배열이 array2 배열에 영향을 준 것 처럼 동일하게 적용되어 포인터 또는 리턴값 없이 array2배열에 전치행렬이 저장됩니다.
3. void matrix_print 함수를 호출하여 array1 배열과 전치된 행렬인 array2 배열을 출력합니다.

main.c › main

Format

```c
//실습과제 p.87 프로그램 3.5 행렬 프로그램
#include <stdio.h>
#include <stdlib.h>
#define MAX_TERMS 100

typedef struct {
  int row;
  int col;
  int value;
}element;

typedef struct SparseMatrix {
  element data[MAX_TERMS];
  int row;
  int col;
  int terms;
}SparseMatrix;

SparseMatrix matrix_transpose(SparseMatrix a)
{
  SparseMatrix b;

  int bindex;
  b.row = a.row;
  b.col = a.col;
  b.terms = a.terms;

  if (a.terms > 0) {
    bindex = 0;
    for (int c = 0; c < a.col; 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(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");
}

int main()
{
  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_transpose(m);
  matrix_print(result);
  return 0;
}
```

./main

```
====================
(0, 1, 9)
(0, 3, 6)
(1, 3, 5)
(2, 5, 2)
(3, 0, 7)
(5, 1, 8)
(5, 4, 1)
====================
```

./main

```
====================
(0, 1, 9)
(0, 3, 6)
(1, 3, 5)
(2, 5, 2)
(3, 0, 7)
(5, 1, 8)
(5, 4, 1)
====================
```

희소 행렬이란

대부분의 요소가 0인 행렬을 의미합니다.

코드설명

#define MAX_TERMS 100

MAX_TERMS를 100으로 지정

typedef struct { ... } element;

element 구조체는 희소 행렬에서 0이 아닌(비제로) 항을 나타냅니다.

row: 항의 행 위치, col: 항의 열 위치, value: 항의 값

typedef struct SparseMatrix { ... } SparseMatrix;

SparseMatrix 구조체는 희소 행렬 전체를 나타냅니다.

data[MAX_TERMS] 비제로 항 저장, row: 행렬의 총 행 수, col: 행렬의 총 열 수, terms: 비제로 항의 개수

SparseMatrix matrix_transpose(SparseMatrix a)

이 함수는 희소 행렬 a을 전치하여 행렬 b에 반환하는 함수입니다.

b행렬의 행,열,비제로 항을 a행렬과 동일하게 선언합니다.

b.row는 a.col로 설정되고, b.col은 a.row로 설정됩니다. -> 전치 후 행과 열이 바뀌기 때문입니다.

b.terms는 a.terms와 동일하게 설정됩니다. - 전치해도 비제로 값이 변하지 않습니다.

즉. 각 a행렬의 비제로 항을 전치하여 b행렬에 저장합니다.

void matrix_print(SparseMatrix a)

이 함수는 주어진 희소 행렬 a의 비제로 항들을 출력하는 함수입니다.

int main()

SparseMatrix m: 원본 희소 행렬을 정의합니다.

SparseMatrix result: 전치된 행렬을 저장할 변수입니다.

matrix_transpose(m) 함수를 호출하여 m을 전치한 결과를 result에 저장합니다.

matrix_print(result) 함수를 호출하여 전치된 행렬을 출력합니다.

코드 실행 시 순서:

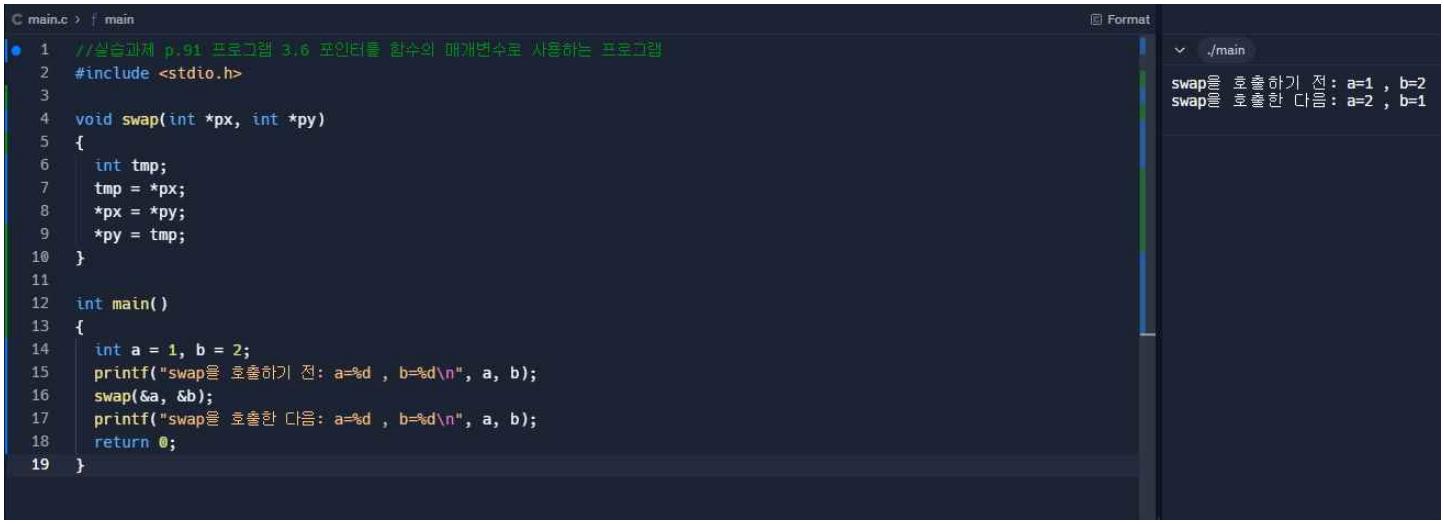
1. 비제로 항 구조체와 전체 행렬 구조체를 선언합니다.
2. SparseMatrix matrix_transpose 함수를 선언하여 원본 희소 행렬에서 비제로 값만 전치하여 새로운 희소 행렬에 저장합니다.
3. 전치된 새로운 희소 행렬을 result에 저장하고 void matrix_print 함수를 호출하여 출력합니다.

희소 행렬의 특징:

희소 행렬의 전치: 전치 연산을 수행할 때 비제로 항만 처리합니다.

비제로 항의 관리: 비제로 항만 처리하므로 메모리 절약이 가능하며, 행렬의 크기와 상관없이 비제로 항만을 저장합니다.

출력 형식: 희소 행렬의 출력은 (행, 열, 값) 형식으로, 비제로 항들만 출력합니다.



```c
//실습과제 p.91 프로그램 3.6 포인터를 함수의 매개변수로 사용하는 프로그램
#include <stdio.h>

void swap(int *px, int *py)
{
  int tmp;
  tmp = *px;
  *px = *py;
  *py = tmp;
}

int main()
{
  int a = 1, b = 2;
  printf("swap을 호출하기 전: a=%d , b=%d\n", a, b);
  swap(&a, &b);
  printf("swap을 호출한 다음: a=%d , b=%d\n", a, b);
  return 0;
}
```

```
./main
swap을 호출하기 전: a=1 , b=2
swap을 호출한 다음: a=2 , b=1
```

코드설명

void swap(int *px, int *py):

이 함수는 두 개의 int 타입 값을 교환하는 역할을 합니다.

swap 함수는 포인터를 매개변수로 받습니다.

이때 포인터는 변수의 메모리 주소를 의미하는데, 이를 통해 함수 내에서 원본 변수의 메모리를 통해 값을 수정할 수 있습니다.

함수 내부에서 임시 변수 tmp를 사용해 두 값을 교환합니다.

int main():

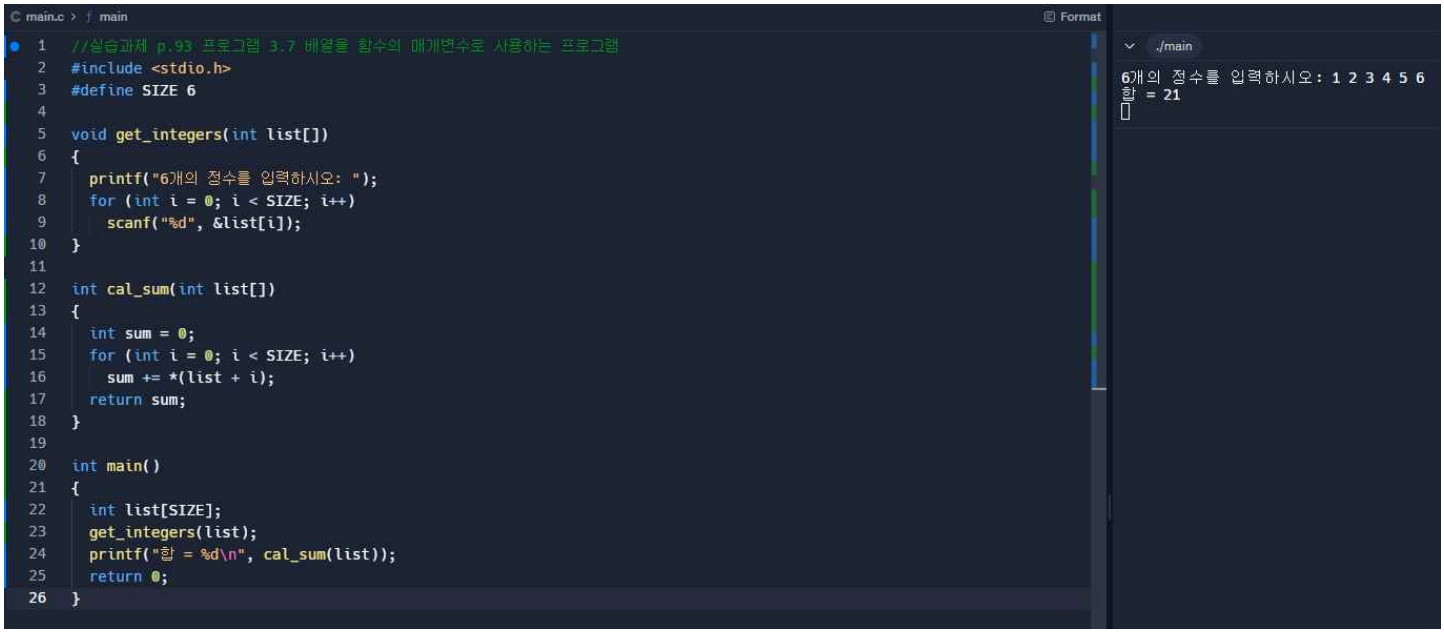
main 함수에서는 두 개의 정수 변수 a와 b를 선언하고 초기값을 지정합니다.

swap 함수를 호출할 때는 a와 b의 메모리 주소를 &a, &b을 통해 전달합니다.

printf()를 통해 서로 값이 바뀐 a와b를 출력합니다.

포인터의 특징:

포인터를 사용하여 main함수의 전역 변수(a,b)의 값을 다른 함수에서 매개변수로 준 메모리 주소를 통해 원본 변수의 값을 변경할 수 있습니다.



```c
//실습과제 p.93 프로그램 3.7 배열을 함수의 매개변수로 사용하는 프로그램
#include <stdio.h>
#define SIZE 6

void get_integers(int list[])
{
  printf("6개의 정수를 입력하시오: ");
  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;
}
```

```
./main
6개의 정수를 입력하시오: 1 2 3 4 5 6
합 = 21
```

코드설명

#define SIZE 6

SIZE를 6으로 지정

void get_integers(int list[]):

void get_integers 함수는 list배열의 요소를 입력받는 함수입니다.

scanf()문을 통해 list배열의 요소를 저장합니다.

이때 list[i]는 포인터와 같은 역할을 하여 입력받을 메모리 주소(&)를 통해 값을 입력받습니다.

int cal_sum(int list[]):

int cal_sum 함수는 list배열의 요소를 모두 더해 값을 리턴하는 함수입니다.

배열의 요소를 입력받을 sum을 0으로 초기화 해줍니다.

for 반복문을 통해 배열의 요소 하나하나를 sum값에 더합니다.

이때 *(list + l)를 해주는 이유는 list의 맨 처음 주소부터 +4(정수형의 메모리) 칸 마다 배열의 요소들이 들어있기 때문입니다.

int main():

크기가 SIZE(6)인 int형 배열인 list를 선언해줍니다.

void get_integers 함수를 통해 배열의 요소를 받아옵니다.

int cal_sum 함수를 통해 배열의 합을 구하여 출력합니다.


```c
//실습과제 p.96 프로그램 3.8 동적 메모리 할당의 예
#include <stdio.h>
#include <stdlib.h>
#include <malloc.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;
}
```

코드설명

#include <malloc.h>:

malloc 함수를 불러오기 위해 헤더파일 선언합니다.

int main()

malloc(SIZE * sizeof(int))는 SIZE 개수의 int형 데이터를 저장할 수 있는 메모리를 동적으로 할당합니다.

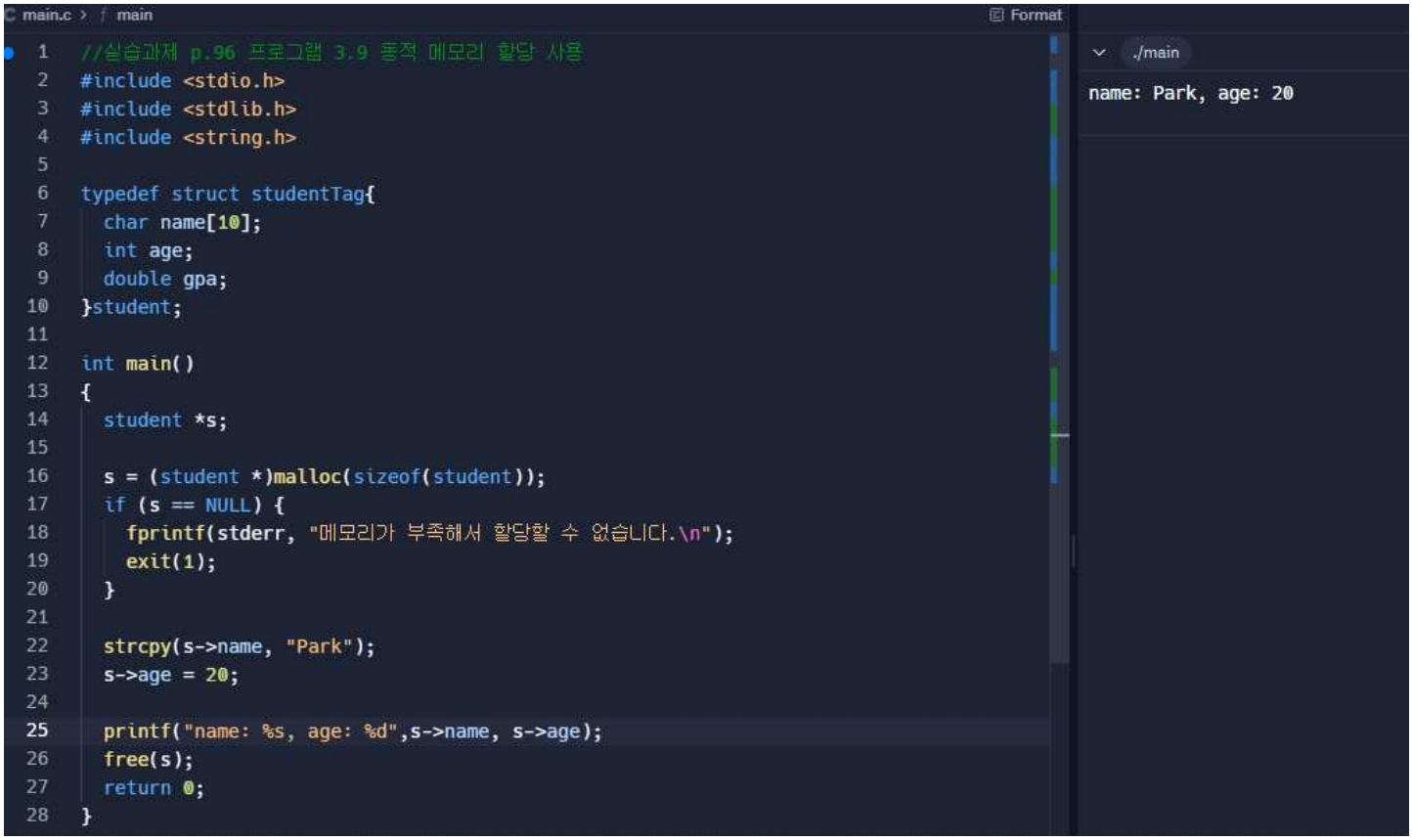
ex) SIZE는 10으로 정의되어 있으므로, 10 * sizeof(int)만큼 메모리를 할당합니다.

할당된 메모리의 시작 주소는 p 포인터에 저장됩니다.

p == NULL일 경우, 메모리 할당에 실패한 것이므로, fprintf로 오류 메시지를 출력하고 exit(1)로 프로그램을 종료합니다.

p에 동적메모리가 정상적으로 할당이 되면 for 반복문을 통해 p의 요소들을 저장하고 출력합니다.

free()를 통해 할당한 메모리를 해제합니다.



코드설명

#include <string.h>:

문자열 처리 함수들을 사용하기 위한 헤더 파일입니다.(strcpy, strcmp 등)

typedef struct studentTag{...}student;:

student 구조체는 name, age, gpa를 저장하는 필드를 가집니다.

name[10]: 문자열(크기 10)

age: 정수형

gpa: 실수형

int main()

malloc(sizeof(student)): student 구조체 크기만큼 메모리를 동적으로 할당합니다.

ex) name 배열의 크기 10+ int형 변수 age(4) + double형 변수 gpa(8)를 합친 크기

할당된 메모리의 시작 주소는 s 포인터에 저장됩니다.

s == NULL일 경우, 메모리 할당에 실패한 것이므로, fprintf로 오류 메시지를 출력하고 exit(1)로 프로그램을 종료합니다.

strcpy(A,B)함수는 B의 문자열을 A에 복사하는 함수입니다.

ex) strcpy(s->name, "Park") : s가 가리키는 student 구조체의 name 필드에 "Park"라는 문자열을 복사

구조체 s의 age필드에 20을 저장합니다.

이때 s->age라고 적혀있지만 (*s).age도 같은 역할을 합니다.

printf()를 통해 s 의 name 값과 age 값을 출력합니다.

free()를 통해 할당한 메모리를 해제합니다.