1장 연습문제 - 1번

1. 교재 ADT 1.1의 function으로 Predecessor(x)를 추가하고자 한다. 다른 function들의 정의를 참조하여 그 정의를 기술하시오.

NaturalNumber Predecessor(x) ::= if(x==0) return 0
else return x-1

2장 연습문제 - 1번

1. 교재 Program 2.5의 다항식 더하기(padd) 슈도 코드는 다항식의 표현 체계와 무관하게(representation-independent) 작성된 것이다. Program 2.5는 정확한지 아니면 오류가 있는지 설명하시오.

Program 2.5 코드의 중간부분에『 if (sum) { 』바로 아랫줄을 보면 Attach 함수에 오류가 있습니다. Attach함수의 매개변수를 통해 항을 추가하는데 다항식이 지정되지 않았습니다. 따라서

Attach(d, sum, LeadExp(a)); ==>> d = Attach(d, sum, LeadExp(a));

로 수정되어야 합니다.

2장 연습문제 - 2번

2. 교재 Program 2.6의 다항식 더하기 C 함수 padd()는 교재 2.4.2절에 설명된 다항식 표현(representation) 방법 두가지 중 항(term) 배열로 다항식을 표현하였을 때의 코드이다. 나머지 다른 한가지 방법 즉, 계수(coefficient) 배열로 다항식을 표현하였을 경우 Program 2.6.에 상응하는 다항식 더하기 C 함수 padd()를 작성하시오. (작성된 padd()를 호출하는 프로그램 및 padd()의 실행 결과는 제출하지 않음.)

```
polynomial padd(polynomial polyA, polynomial polyB, polynomial polyC)
{
        float coefficient;              // polyA와 polyB의 차수가 같고 계수가 모두 0이 아닌 경우에 두 계수를 더한 것의 값을 저장할 변수

        switch(COMPARE(polyA.degree, polyB.degree))             // 덧셈에 이용될 다항식 polyA와 polyB의 degree는
                                                                // 본격적인 adding작업에서 degree를 유실할 것이기 때문에
                                                                // 사전에 polyC.degree의 값을 지정한다.
                                                                // switch문을 이용하여 degree가 더 큰 값을 polyC의 degree에 넣는다.
        {

        case -1 :
                polyC.degree = polyB.degree;
                break;
        case 0 :
                polyC.degree = polyB.degree;
                break;
        case 1 :
                polyC.degree = polyA.degree;
                break;
        }


        while(polyA.degree!=0 && polyB.degree!=0)               // polyA와 polyB의 degree는 while문과 다음의 switch문에서
                                                                // (polyC에 최고차항부터 계수가 복사됨에 따라) 1씩 감소하고,
                                                                // 모두 옮겨진 후 두 다항식의 degree가 0이 되면 루프를 빠져나간다.
        {

                switch(COMPARE(polyA.degree, polyB.degree))
                {

                case -1 :                       // polyA의 차수가 polyB의 차수보다 낮은 경우
```

```
                polyC.coef[polyB.degree] = polyB.coef[polyB.degree];
                polyB.degree -= 1;
                break;


        case 0 :                        // polyA와 polyB의 차수가 같은 경우

                if(polyA.coef[polyA.degree] == 0 && polyB.coef[polyB.degree] != 0)
                // polyA의 계수가 0, polyB의 계수가 0이 아닌 경우
                        polyC.coef[polyB.degree] = polyB.coef[polyB.degree];

                else if(polyB.coef[polyB.degree] == 0 && polyA.coef[polyA.degree] != 0)
                // polyB이 계수가 0, polyA의 계수가 0이 아닌 경우
                        polyC.coef[polyA.degree] = polyA.coef[polyA.degree];

                else if(polyB.coef[polyB.degree] == 0 && polyA.coef[polyA.degree] == 0)
                // polyA와 polyB의 계수가 모두 0인 경우
                        break;

                else
                // polyA와 polyB의 계수가 모두 0이 아닌 경우
                {
                        coefficient = polyA.coef[polyA.degree] + polyB.coef[polyB.degree];
                        polyC.coef[polyA.degree] = coefficient;
                }

                polyA.degree -= 1;
                polyB.degree -= 1;
                break;

        case 1 :                        // polyA의 차수가 polyB의 차수보다 높은 경우


                polyC.coef[polyA.degree] = polyA.coef[polyA.degree];
                polyA.degree -= 1;

        }
    }
}
```

3장 연습문제 – 1번

자료구조 3장 연습문제
1. 교재 3.6.3절 연습문제 1번(a) 및 (d)의 infix식을 postfix 식으로 변환할 때, infix 식의 token 별로 stack의 변화 과정 및 postfix 식의 생성 과정을 보이시오. (답안 형식은 3장 강의 슬라이드(1) 40쪽 “better presentation" 참조)

(a)

a * b * c ==> (a * b) * c

Token(Infix)	Stack	Postfix
	$	
(	$ (	
a	$ (	a
*	$ (*	a
b	$ (*	ab
)	$	ab*
*	$ *	ab*
c	$ *	ab*c
$	$	ab*c*

(d)

(a + b) * d + e / (f + a * d) + c

==> (((a + b) * d) + (e / (f + (a * d)))) + c

Token(Infix)	Stack	Postfix
	$	
(	$ (	
(	$ ((	
(	$ (((	
a	$ (((	a
+	$ (((+	a
b	$ (((+	a b
)	$ ((	a b +
*	$ ((*	a b +
d	$ ((*	a b + d
)	$ (	a b + d *
+	$ (+	a b + d *
(	$ (+ (	a b + d *
e	$ (+ (	a b + d * e
/	$ (+ (/	a b + d * e
(	$ (+ (/ (	a b + d * e
f	$ (+ (/ (	a b + d * e f
+	$ (+ (/ (+	a b + d * e f
(	$ (+ (/ (+ (	a b + d * e f
a	$ (+ (/ (+ (	a b + d * e f a
*	$ (+ (/ (+ (*	a b + d * e f a
d	$ (+ (/ (+ (*	a b + d * e f a d
)	$ (+ (/ (+	a b + d * e f a d *
)	$ (+ (/	a b + d * e f a d * +
)	$ (+	a b + d * e f a d * + /
)	$	a b + d * e f a d * + / +
+	$ +	a b + d * e f a d * + / +
c	$ +	a b + d * e f a d * + / + c
$	$	a b + d * e f a d * + / + c +