

6장 데이터 타입(2)

2020. 5. 28

순천향대학교 컴퓨터공학과

목차

- 서론
- 기본 데이터 타입
- 문자 스트링 타입
- 사용자-정의 순서 타입
- 배열 타입
- 연상 배열
- 레코드 타입
- 튜플 타입
- 리스트 타입
- 공용체 타입
- 포인터와 참조 타입
- 타입 검사
- 강타입
- 타입 동등

배열 타입

- **배열**(array)은 동질적인(동일한 타입을 갖는) 데이터 원소들의 집단체(aggregate)이고, 개개의 원소는 그 집단체의 첫번째 원소에 상대적인 위치로 식별
- 설계시 고려사항
 - 어떤 타입이 첨자에 대해서 적법한가?
 - 원소 참조시 첨자 식이 범위 검사되는가?
 - 첨자 범위는 언제 바인딩되는가?
 - 배열 할당은 언제 일어나는가?
 - 튜플형 또는 직사각형 다차원 배열이 허용되는가? 또는 둘 다 허용되는가?
 - 배열이 할당될 때, 초기화 가능한가?
 - 슬라이스(slice)가 존재한다면, 어떤 종류의 슬라이스가 지원되는가?

배열 인덱싱

- 인덱싱은 인덱스로부터 원소로의 사상을 의미
arrayName (인덱스 리스트) => 원소
- 인덱스 구문
 - 괄호 사용
 - FORTRAN, PL/I, Ada
 - 배열 참조와 함수 호출간의 균일성(uniformity)를 보여준다: 이 둘 모두 사상(mapping)을 의미
 - 대괄호 사용
 - 다른 대부분의 언어

배열 인덱스(첨자) 타입

- 첨자 타입

- FORTRAN, C: 정수만 가능
- Ada: 정수, 열거, 불리언, 문자 등 임의 순서 타입
- Java: 정수만 가능

- 인덱스 범위 검사

- C, C++, Perl, FORTRAN: 범위 검사 명세하지 않음
- Java, C#: 범위 검사 명세
- Ada: 디폴트는 범위 검사 요구하나 프로그래머에 의해서 무위화 가능

참자 바인딩 및 배열 유형

- 배열 변수의 참자 타입의 바인딩은 보통 정적이지만, 참자 값 범위는 때때로 동적으로 바인딩
 - C 기반 언어: 참자 범위 하한이 묵시적으로 0
 - Fortran95+: 디폴트로 1 또는 임의 정수 값으로 설정 가능
- 참자 범위 바인딩 시기, 기억공간 바인딩 시기 및 그 할당 위치에 기준하여 배열을 4가지로 유형화
 - 정적
 - 고정 스택-동적
 - 고정 힙-동적
 - 힙-동적

정적 배열, 고정 스택-동적 배열

- 정적 배열(static array)은 첨자 범위가 정적으로 바인딩, 기억공간 할당이 정적
 - 장점: 효율적(efficiency)
 - 예: C/C++의 static 배열
- 고정 스택-동적 배열(fixed stack-dynamic array)은 첨자 범위는 정적 바인딩, 기억 공간은 선언문 세련화시간에 할당
 - 장점: 기억공간의 효율성
 - 예: C/C++의 함수 내부에 선언된 배열

고정 힙-동적 배열

- 고정 힙-동적 배열(fixed heap-dynamic array)은
참자 범위와 기억 공간 바인딩이 실행중 사용자 프
로그램이 요청할 때 이루어지고, 힙으로부터 할당
 - 참자범위와 기억공간이 바인딩된 후에는 고정
 - 유연성 vs. 할당 시간
 - 언어 예
 - C, C++의 동적 배열
 - Java, C#의 배열

힙-동적 배열

- 힙-동적 배열(heap-dynamic array)은 첨자 범위, 기억 공간 바인딩이 동적, 이후 바인딩은 변경 가능
 - 유연성 vs. 할당 및 회수 부담
 - 언어 예:
 - Java의 ArrayList, C#의 List
 - Perl, JavaScript, Python

```
ArrayList band = new ArrayList();  
  
Band.add("Paul");  
Band.add("Pete");  
...  
System.out.println("band.get(1));
```

배열 초기화

- 배열의 기억공간 할당 시점에 그 배열을 초기화하는 수단을 제공
 - 예제

```
int list[] = {4, 5, 7, 83};
char name[] = "Gildong";
char *names[] = {"Bob", "Jake", "Joe"};
String[] names = {"Bob", "Jake", "Joe"}; // in Java
```

배열 연산

- 배열 연산은 배열 단위의 연산을 수행
 - 공통된 배열 연산: 배정, 접합, 동등/비동등 비교, 슬라이스
 - 언어 예
 - APL: 가장 강력한 배열 처리 언어
 - 산술연산이 벡터와 행렬에 대해서 정의
 - 벡터(V)와 행렬(M)에 대한 단항 연산자 제공
- | | |
|-------------------------|----------------------------|
| ΦV : V의 원소를 역순으로 | ΦM : M의 열을 역순으로 |
| ΘM : M의 행을 역순으로 | $\emptyset M$: M의 전치행렬 계산 |
| $\div M$: M의 역행렬 계산 | |
- $A+.XB$ // A, B는 벡터이면?
 - Perl: 배정 허용, 비교는 지원하지 않음
 - Python: 배정, 접합(+), 원소 멤버십(in) 허용, 비교 지원
 - C 기반 언어:

직사각형 배열 vs. 톱니형 배열

- 직사각형 배열(rectangular arrays)은 모든 행이 동일한 개수의 원소를, 모든 열이 동일한 개수의 원소를 갖는 다차원 배열
 - 정확히 직사각형 테이블을 모델링
 - 언어 예: Fortran, Ada, C#
`myArray[3, 7]`
- 톱니형 배열(jagged arrays)은 행들의 크기가 동일할 필요가 없는 다차원 배열
 - 다차원 배열에서, 배열의 원소가 배열인 경우(“an array of arrays”)
 - 언어 예: C, C++, Java, C#
`myArray[3][7] // in C#`

`int [][]ma = new int [MAX][]; // in Java`
`for (int i = 0; i <= MAX; i++)`
`ma[i] = new int [i+1];`

슬라이스

- 슬라이스(slice)는 배열의 부분적 구조
 - 배열 일부분에 대한 참조 메커니즘을 제공(예를 들면, 행렬에서 한 개의 행이나 열을 참조 가능)
 - 배열 연산을 제공하는 언어에서 제공

슬라이스 예제

- In Python

```
vector = [2, 4, 6, 8, 10, 12, 14, 16]  
mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

- `vector(3:6)` => `[8, 10, 12]` // 첨자는 0부터 시작
 - 첫번째 숫자: 슬라이스의 첫번째 첨자
 - 두번째 숫자: 슬라이스의 마지막 첨자 다음의 첫번째 첨자
- `mat[1]`
- `mat[0][0:2]`
- `vector[0:7:2]`

배열 구현

- 배열 원소 접근 함수 구현
 - 배열 원소에 대한 첨자 식을 원소의 주소로 사상
 - 원소에 대한 접근 코드는 컴파일러가 생성
- 1차원 배열의 원소 접근 함수

$$\text{주소}(\text{list}[k]) = \text{주소}(\text{list}[\text{하한}]) + (k - \text{하한}) * \text{원소_크기}$$

- In C, 주소(list[k]) =

다차원 배열 구현

- 다차원 배열의 저장 순서 (1차원 배열로 사상)
 - 행-우선 순서(row major order): 대부분의 언어
 - 열-우선 순서(column major order): Fortran

- Ex.

3	5	7
1	3	8
6	2	5

다차원 배열 구현 (2)

- 다차원 배열에서 접근 함수 (행-우선 순서)
 - 기반 주소에 앞선 원소들의 개수와 원소 크기를 곱한 것을 더한다.
 - 배열의 각 차원에 대해서 한 개의 곱셈과 덧셈 필요

$$\begin{aligned} \text{주소}(a[i,j]) = & \text{주소}(a[0,0]) \\ & + ((i\text{번째 행보다 앞선 행의 개수}) * (\text{열의 크기})) \\ & + (j\text{번째 열의 왼쪽에 위치한 원소 개수}) * \text{원소_크기} \end{aligned}$$

- C의 $a[m][n]$ 배열에서,
주소($a[i][j]$) =

	1	2	...	j-1	j	...	n
1							
2							
⋮							
i-1							
i					⊗		
⋮							
m							

배열 서술자

- 배열 원소 접근 함수에 필요한 정보
- 색인 범위에 대한 실행시간 검사
- 첨자 범위의 정적/동적에 따른 컴파일-시간, 실행시간-시간 서술자

일차원 배열

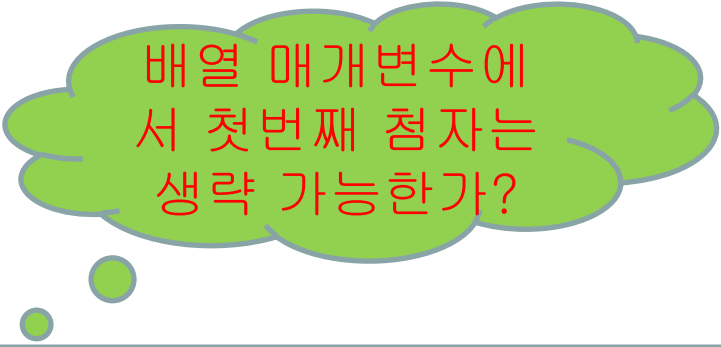
Array
Element type
Index type
Index lower bound
Index upper bound
Address

다차원 배열

Multidimensioned array
Element type
Index type
Number of dimensions
Index range 1
⋮
Index range n
Address

예제

- 언어 예: C



배열 매개변수에서
첫번째 첨자는
생략 가능한가?

```
double sum(double a[], int n) {...}
```

```
int sum (int a[][5], int n, int m) {...}
```

연상 배열

- 연상 배열(associative array)은 원소들간에 순서가 없으며, 키 값을 통해서 원소에 접근하는 배열
 - 사용자-정의 키가 원소와 함께 저장
- 언어 예
 - Perl, Python에서는 내장 타입으로 지원, Java, C++, C#에서는 클래스 라이브러리에 의해서 지원
 - In Perl,

```
%hi_temp = {"Mon"=>77, "Tue"=>79, "Wed"=> 65, ...};  
$hi_temp {"Wed"} = 83;
```
 - In Python,

```
d = {1:'apple', 2:'banana'}  
d['apple']
```

레코드 타입

- 레코드(record) 타입은 이질적일 수 있는 데이터 원소들의 집단체
 - 각 원소들은 이름으로 식별
 - COBOL에서 도입, 대부분 언어에서 제공(Fortran 90 이전 버전 제외)
- 설계 고려사항
 - 필드 참조의 구문 형식은?
 - 생략 참조가 허용되는가?

레코드 정의

- In COBOL

- 수준 번호를 이용하여 중첩된 레코드를 표현

```
01 EMP-REC.  
  02 EMP-NAME.  
    05 FIRST PICTURE X(20).  
    05 MID      PICTURE X(10).  
    05 LAST     PICTURE X(20).  
  02 HOURLY-RATE PICTURE 99V99.
```

20개의 alphanumeric

중간에 소수점을 갖는
4개 십진수 표현

레코드 수준 번호이며, 레코드의 계층적
구조를 나타냄

기억장소 형식을 나타
냄

레코드 정의 (2)

- In Ada

```
type Emp_Name_Type is record
    First: String (1..20);
    Mid: String (1..10);
    Last: String (1..20);
end record;
type Emp_Rec_Type is record
    Employee_Name: Emp_Name_Type;
    Hourly_Rate: Float;
end record;

Emp_Rec: Emp_Rec_Type;
```

- In C
- In Java, C#

레코드 필드 참조

- 필드 참조 구문

// in COBOL

field_name OF record_name_1 OF ... OF record_name_n

// 다른 언어: 도트 표기법 사용

record_name_1.record_name_2. ... record_name_n.field_name

레코드 필드 참조 (2)

- 필드 참조 방식

- **완전 자격 참조**(fully qualified references)는 가장 큰 포괄적인 레코드부터 특정 필드에 이르기까지 모든 중간 레코드 이름들이 그 참조에 포함

- Ex: Emp_Rec.Employee_Name.First

- **생략 참조**(elliptical references)는 레코드 이름 일부가 생략되는 것을 허용(참조가 모호하지 않는 경우)

- Ex: First // First 필드에 대한 참조
 Employee_Name.First
 Emp_Rec.First

레코드 연산

- 레코드 단위의 연산

- 배정은 일반적 (타입이 동등할 경우)
- In Ada, 동등/비동등의 비교, 초기화
- In COBOL,
 - MOVE CORRESPONDING
 - 원시 레코드의 필드를 목표 레코드의 상응한 필드로 복사

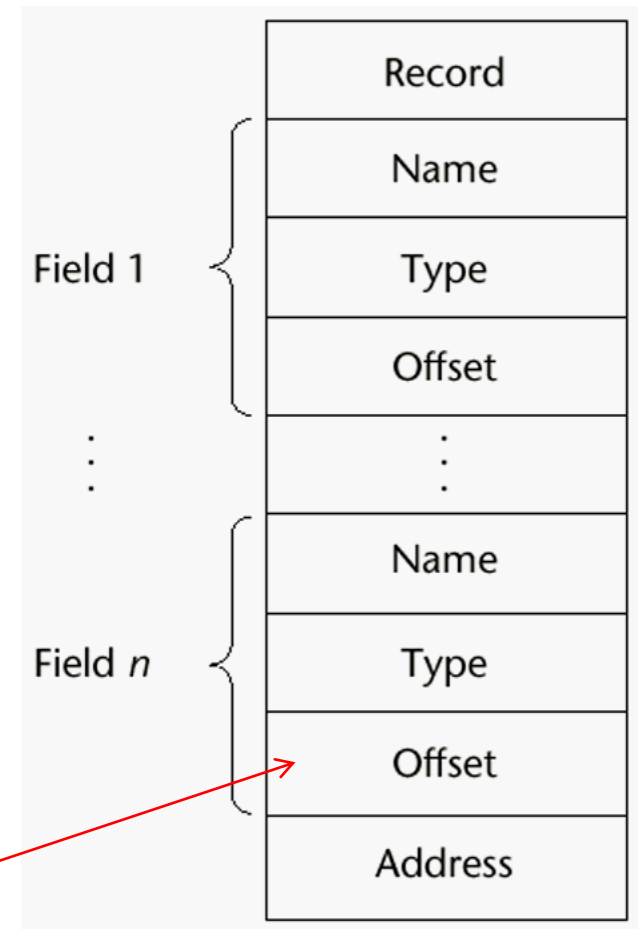
레코드와 배열 비교

- 항목들이 동질적 vs. 이질적
- 항목 접근 방식
 - 배열 원소 접근 vs. 레코드 필드 접근
 - 동적 vs 정적

레코드 타입 구현

- 컴파일시간 서술자
 - 필드들은 메모리에 인접하여 저장
 - 필드의 크기가 동일하지 않기 때문에 각 필드에 레코드의 시작 주소에 상대적인 오프셋 주소가 연관

레코드 시작주소로
부터 이격 거리



튜플 타입

- 튜플(tuple)은 레코드와 유사하나 원소들이 명명되지 않는 데이터 타입
- In Python
 - Immutable 튜플 타입 지원(\leftrightarrow mutable 리스트로 변환 가능)
Ex.

```
myTuple = (3, 5.8, 'apple') # 요소들은 동일 타입일 필요가 없다
myTuple[1] # 인덱싱을 통한 참조: 튜플의 첫번째 원소 참조
myTuple[1] = 5 # Error: 튜플은 변경 불가(immutable)
myTuple += yourTuple # '+'는 접합 연산자로 새로운 튜플 생성
Del # 삭제
```

- 용도:
 - 배열이 매개변수로 전달되나 변경되지 않아야 하는 경우
 - 함수가 여러 개의 값을 반환할 때

리스트 타입

- LISP에서 처음으로 도입, 최근에 일부 명령형 언어에 도입
- In LISP, Scheme
 - 리스트는 괄호로 구분되고, 원소들은 콤마로 구분되지 않는다.
 - (A B C D)
 - (A (B C) D)
 - 데이터와 코드는 동일한 형식을 갖는다
 - (A B C)가 코드일 때, 함수 A가 매개변수 B C에 적용
 - 리스트가 데이터일 경우에 어포스트로피로 표시한다.
 - '(A B C)는 데이터이다.

리스트 연산

- 리스트 분리 함수

(CAR '(A B C)) // 리스트의 첫번째 원소 반환

(CDR '(A B C)) // 리스트의 첫번째 원소를 제외하고 반환

- 리스트 생성 함수

(cons 'A '(B C)) // 첫번째 매개변수를 두번째 매개변수 리스트에
// 포함시켜서 새로운 리스트 생성

(list 'A 'B '(C D)) // 매개변수들을 원소로 포함하는 리스트 생성/반환

리스트 예: Python

- Python은 리스트 타입 제공
 - 배열로서 역할
 - 리스트는 변경 가능(mutable)
 - 원소들은 임의 타입이나 객체 가능
 - 튜플처럼 사용 가능: 인덱싱, 접합, 슬라이싱(부분 참조), in 등

```
myList = [3, 5.8, "grape"]
```

```
x = myList[1]    //x에 5.8을 할당: 리스트 원소들은 0부터 인덱싱
```

```
del myList[1]    // 두번째 원소 삭제
```

- **리스트 함축**(list comprehension) 지원: 집합 표기법에 기반
[x*x for x in range (12) if x % 3 == 0]

```
// range 함수는 [0, 1, 2, ..., 11, 12]의 배열 생성
```

```
// 조건이 배열 각 원소에 적용된 결과 값들로 구성된 배열 구성
```


공용체 타입

- 공용체(union)는 그 변수가 프로그램 실행 중에 다른 시기에 다른 타입의 값이 저장될 수 있는 타입
- 설계 고려사항
 - 타입 검사가 요구되는가? => 그러한 타입 검사는 동적

공용체 유형

- 자유 공용체(free union)
 - 타입 검사를 지원하지 않음
 - 언어 예: Fortran, C, C++
- 판별 공용체(discriminated union)
 - 타입 검사 지원
 - 판별자(discriminant) 또는 태그(tag)라는 타입 지시자를 포함
 - 언어 예: Ada

예제: 자유 공용체

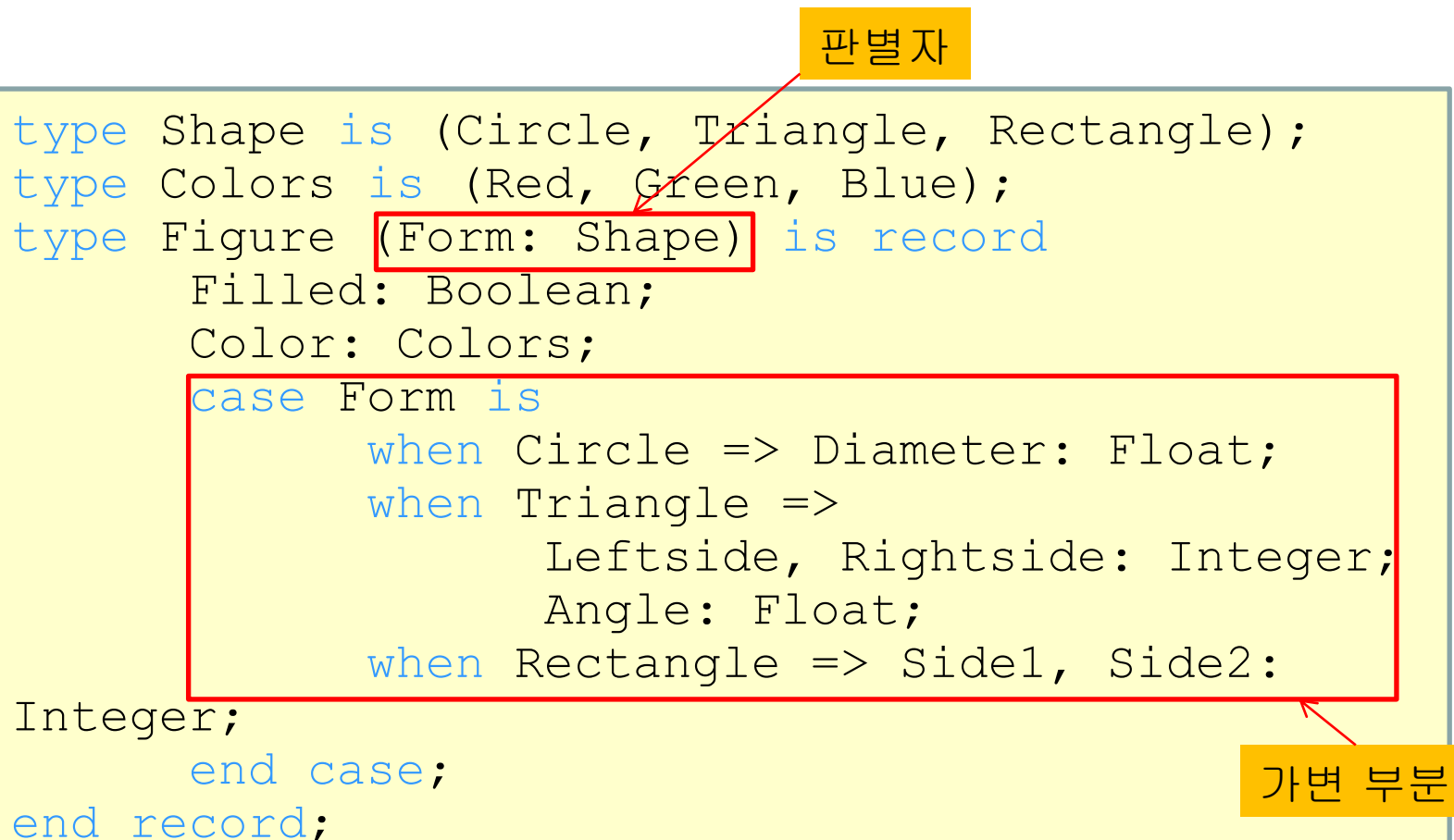
- In C

```
union flexType {  
    int intEl;  
    float floatEl;  
};  
  
union flexType e;  
float x;  
...  
e.intEl = 27;  
x = e.floatEl; // 타입 검사가 이루어지는가?
```

예제: 판별 공용체

- Ada에서 가변 레코드(variant record)로 명세

```
type Shape is (Circle, Triangle, Rectangle);
type Colors is (Red, Green, Blue);
type Figure (Form: Shape) is record
    Filled: Boolean;
    Color: Colors;
    case Form is
        when Circle => Diameter: Float;
        when Triangle =>
            Leftside, Rightside: Integer;
            Angle: Float;
        when Rectangle => Side1, Side2:
            Integer;
    end case;
end record;
```



```
type Shape is (Circle, Triangle, Rectangle);
type Colors is (Red, Green, Blue);
type Figure (Form: Shape) is record
    Filled: Boolean;
    Color: Colors;
    case Form is
        when Circle => Diameter: Float;
        when Triangle =>
            Leftside, Rightside: Integer;
            Angle: Float;
        when Rectangle => Side1, Side2:
            Integer;
    end case;
end record;
```

```
figure1: Figure;
Figure2: Figure(Form => Triangle);    // 제한 가변변수
                                       // 정적타입검사

...
figure1 := (filled=> True, color=>Blue, Form=>rectangle,
            side1=>12, side2=>3);
    // 태그 포함 모든 필드 배정 필요, 이후에 태그 변경 가능

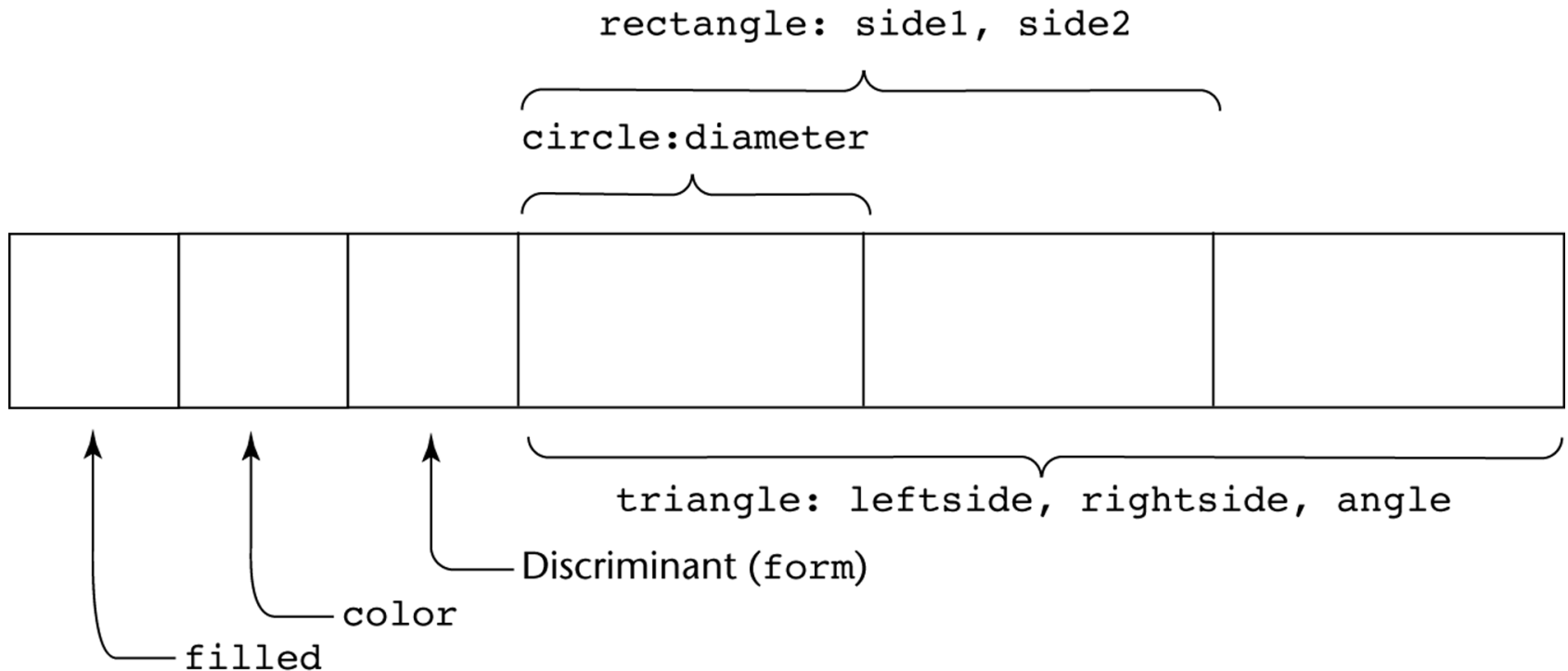
...
if (figure1.Diameter > 3.0)... // 실행시간중 동적 타입 검사
    // Form 태그가 Circle인지를 판단
```

공용체 평가

- 공용체가 안전한 구조인가?
 - 자유 공용체 vs. 판별 공용체
- 언어 예
 - C, C++, Ada
 - Java, C#은 공용체를 포함하는가?

공용체 타입 구현

- 가장 큰 가변 요소를 위한 충분한 기억공간 할당



공용체 타입 구현 (2)

- 서술자의 태그 항목에 case 테이블 연관하여 가변 요소 기술

```
type Node (Tag : Boolean) is  
  record  
    case Tag is  
      when True => Count : Integer;  
      when False => Sum : Float;  
    end case;  
  end record;
```

