제 6 장 데이터 타입

01_데이터 타입의 개요

- 데이터 타입
  - 프로그램의 모든 데이터에는 타입(type)이 있음
  - 예 1 : x = 12 + 3.456;
    - 12는 정수 타입, 3.456은 부동 소수점 타입
  - 예 2: int x;
    - 타입 int를 변수 x에 바인딩
  - 예 3: double y;
    - 타입 double을 변수 y에 바인딩
  - 예 4: const MAX = 100;
    - 상수 MAX에 정수 타입을 묵시적으로 바인딩

➔ 데이터 타입 : 그 타입의 변수가 가질 수 있는 값들의 집합

01_데이터 타입의 개요

- 데이터 타입의 종류
  - 기본 데이터 타입
    - 정수 타입, 부동소수점 타입과 같이 해당 언어에서 기본적으로 제공
  - 사용자 정의 데이터 타입
    - 레코드 타입과 같이 기본 데이터 타입을 이용하여 사용자가 생성

02_수치 타입

- 정수 타입
  - 주요 관심사항은 정수 값을 표현하는 데 사용하는 바이트 수
  - FORTRAN과 같은 언어는 한 가지 크기만을 제공
  - Java와 C와 같은 언어는 여러 가지 크기를 제공
  - Java의 정수 타입

타입	크기	표현 범위
byte	1바이트	-128 ~ 127
short	2바이트	-32,768 ~ 32,767
int	4바이트	-2,147,483,648 ~ 2,147,483,647
long	8바이트	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807

    - 대부분의 컴퓨터는 2의 보수 표기법을 사용

02_수치 타입

- 오버플로우 문제

```
byte x;
x = 100 + 100;
```

byte 타입이 저장할 수 있는 최대 값은 127

- 비부호 정수(unsigned integer) 타입

타입	크기	표현 범위
short int	2바이트	-32,768~32,767
unsigned short int	2바이트	0~65,536
int	2바이트 of 4바이트	-32,768~32,767 또는 2,147,483,648~2,147,483,647
unsigned int	2바이트 of 4바이트	0~65,536 또는 0~4,294,967,295
long int	4바이트	-2,147,483,648~2,147,483,647
unsigned long int	4바이트	0~4,294,967,295

02_수치 타입

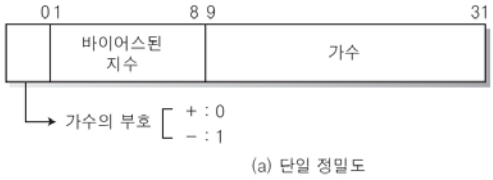
- 부동 소수점 타입
  - 실수 표현
  - 대부분의 컴퓨터들은 IEEE 754 표준 형식을 사용
  - 4바이트 크기의 float와 8바이트 크기의 double 타입 제공
  - C 언어는 sizeof 사용

```
printf("float: %dbytes\ndouble: %dbytes", sizeof(float), sizeof(double));
```

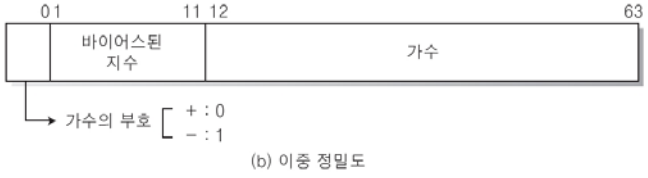
  - 예: float 타입과 double 타입의 크기 확인

02_수치 타입

- IEEE 754 표준 형식



(a) 단일 정밀도



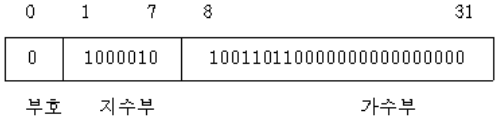
(b) 이중 정밀도

- 단일 정밀도는 이중 정밀도에 비해 연산 속도는 빠름
- 정밀도는 이중 정밀도에 비해 떨어짐

수치와 문자 데이터표현

- 부동 소수점 데이터 형식
  - 가수부는 소수점이하의 유효숫자를 2진수로 변환하여 표현
  - 10진수 155를 부동 소수점 데이터 형식으로 나타내보자.

(풀이) ① (155)를 2진수로 변환하면 $(1001101)_2$ 이다.
② 변환된 $(1001101)_2$을 16진수로 변환하면$(9B)_{16}$이다.
③ $(9B)16$를 지수형으로 표현하면$(0.9B)_{16} * 10^2$ 이다.
④ 부호는 양수 0, 가수부는 9B, 지수부는 64+2 =66이다.
[단,64가 기준(bias)지수임. 따라서 1제곱은 지수가 65이고, -1 제곱은 63이 된다.].
⑤ 가수부는 $(9b)_{16} = (10011011)_2$ 이고, 지수부는 $(66)_{10} = (1000010)_2$이므로,
32비트로 표현하면, 다음과 같다.



03_불린 타입

- 불린 타입
  - true와 false라는 두 개의 값
  - ALGOL60에서 처음 도입
  - C를 제외한 대부분의 언어에서 제공
  - 불린 타입 데이터에 대한 대표적인 연산

x	y	x and y	x or y	not x
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

03_불린 타입

- C++에서 and 연산자는 &&, or 연산자는 ||, 그리고 not 연산자는 !
- 예: ch에 저장된 값이 영문자인지를 판별하는 C++ 예

```
bool upperCheck, lowerCheck;
      ⋮
upperCheck = (ch >= 'A' && ch <= 'Z');
lowerCheck = (ch >= 'a' && ch <= 'z');
if (upperCheck || lowerCheck) {
      ⋮
}
```

04_문자 타입

- 문자 타입
  - 예 : C 언어의 문자 타입

```
ch = 'a';
```

  - 문자 코딩 기법을 사용하여 표현
  - ASCII
    - 0~127까지 총 128개 문자 표현 가능
  - 유니코드
    - ASCII로 표현할 수 없는 나라별 언어 표현 가능
    - Java, C#에서 사용
    - 예 : Java의 유니코드 사용

```
public class unicode{
  public static void main(String[] args){
    char ch = '\ud55c';
    System.out.println(ch);
  }
}
```

05_문자열 타입

- 문자열 타입
  - 예 : C 언어의 문자열 타입

```
printf( "Hello");
```

  - 1960년대 중반에 들어 프로그래밍 언어에 문자열 처리를 위한 기능들이 크게 요구
    - 변수들이 문자열을 값으로 가질 수 있도록 하는 것
    - 문자들의 순서를 기반으로 하여 관계 연산자를 문자열 비교에 그대로 사용할 수 있도록 하는 것

    ➔ 이러한 기능을 제공한 첫 번째 언어가 PL/ I

  - PL/I

```
DCL A CHAR(10);
```

➔ 문자열의 길이가 10인 문자열 변수 A 선언, 문자열의 길이가 10보다 작으면 공백으로 채워짐

```
DCL A CHAR(50) VARYFYING;
```

➔ 문자열의 길이가 50보다 작으면 해당 문자열의 길이만큼의 크기로 자른다.

05_문자열 타입

- 문자열과 관련된 여러 연산 제공

  | LENGTH SUBSTR INDEX VERIFY TRANSLATE

  - | : 문자열과 문자열 연결

    A = 'PROGRAMMING ' | 'LANGUAGE' ➔ 'PROGRAMMING LANGUAGE'

  - LENGTH : 문자열의 길이를 알아내는 연산

    LENGTH(A) ➔ 20

  - SUBSTR(A, I, J) : 문자열 A의 I번째 문자부터 J개 문자열 추출

    SUBSTR(A, 13, 4) ➔ LANG

05_문자열 타입

- INDEX(A, B)
  - 문자열 A에서 문자열 B의 위치를 찾아 정수값으로 반환
  - 문자열 A에 문자열 B가 없으면 0을 반환
- VERIFY(A, B)
  - 문자열 A에서 문자열 B에 속하지 않는 첫번째 문자의 위치를 정수값으로 반환
  - 문자열 B에 속하지 않는 문자가 없으면 0을 반환

    VERIFY(B, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')

    - 영문자가 아닌 문자의 위치를 찾아 반환
- TRANSLATE(A, B, C)
  - 문자열 A에서 C에 해당하는 문자를 문자 B로 교체

    TRANSLATE(B, '_', '-')

    - 문자열 B에서 '-'를 '_'로 교체

05_문자열 타입

- Pascal
  - PL/I의 문제점 보완
    - 문자열 타입 구현 난해 ➔ 문자열을 기본 타입에서 제외
    - 실행 시간면에서 비효율적 ➔ 배열을 이용한 문자열 처리
  - 예: char 타입의 배열을 이용해 문자열 저장이 가능한 name 선언

```
var name: array[1...20] of char;
```

  - C/C++ 활용 예 : char 배열을 이용한 문자열 처리

```
char str[] = "programming";
```

05_문자열 타입

- C/C++ 언어에서 제공하는 문자열 처리 함수

```
strcat   strcmp   strcpy   strlen   strchr
```

  - strcat(s, t) : 문자열 t를 문자열 s 끝에 연결
  - strcmp(s, t) : 문자열 s와 문자열 t의 코드값 비교

```
if (! strcmp(s, t)) {
        ...
}
```

두 문자열이 같으면 if 조건이 참

  - strcpy(s, t) : 문자열 t를 문자열 s에 복사

```
strcpy(str, "programming");
printf("%s", str);
```

➔ programming

  - strlen(s) : 문자열 s의 길이 반환

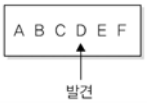
```
strlen("linux")
```

➔ 5

05_문자열 타입

- strchr(s, c) : 문자열 s에서 문자 c가 처음으로 발견된 위치 반환

```
printf("%s", strchr("ABCDEF", 'D'));
```
➔ DEF



- C/C++ : char 포인터 이용한 문자열 처리 가능

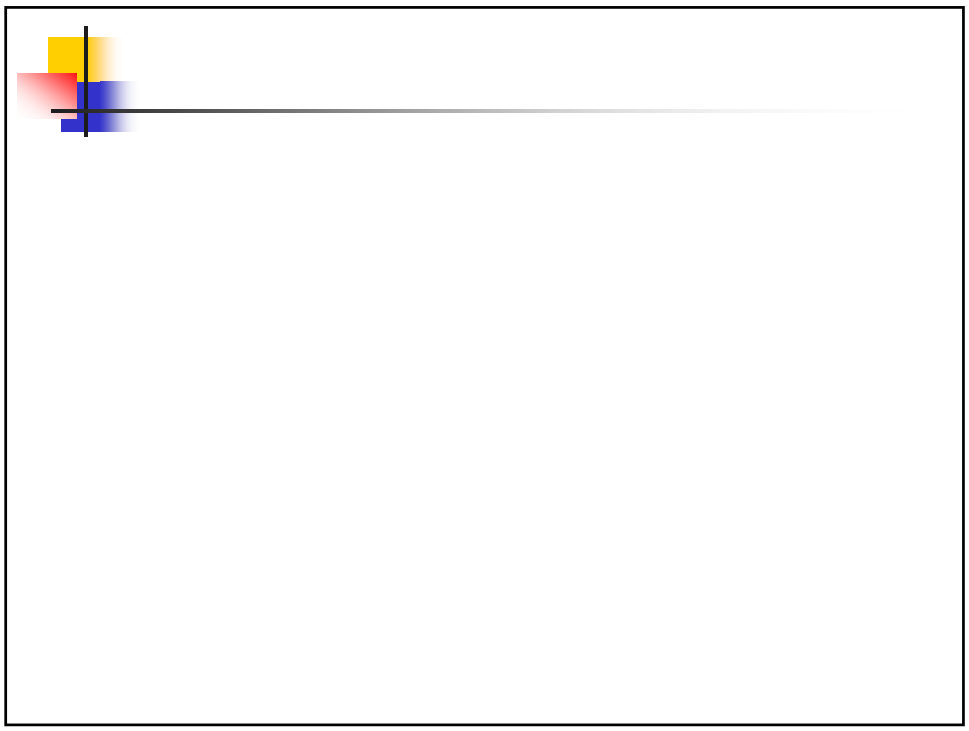
```
char *str = "programming";
```

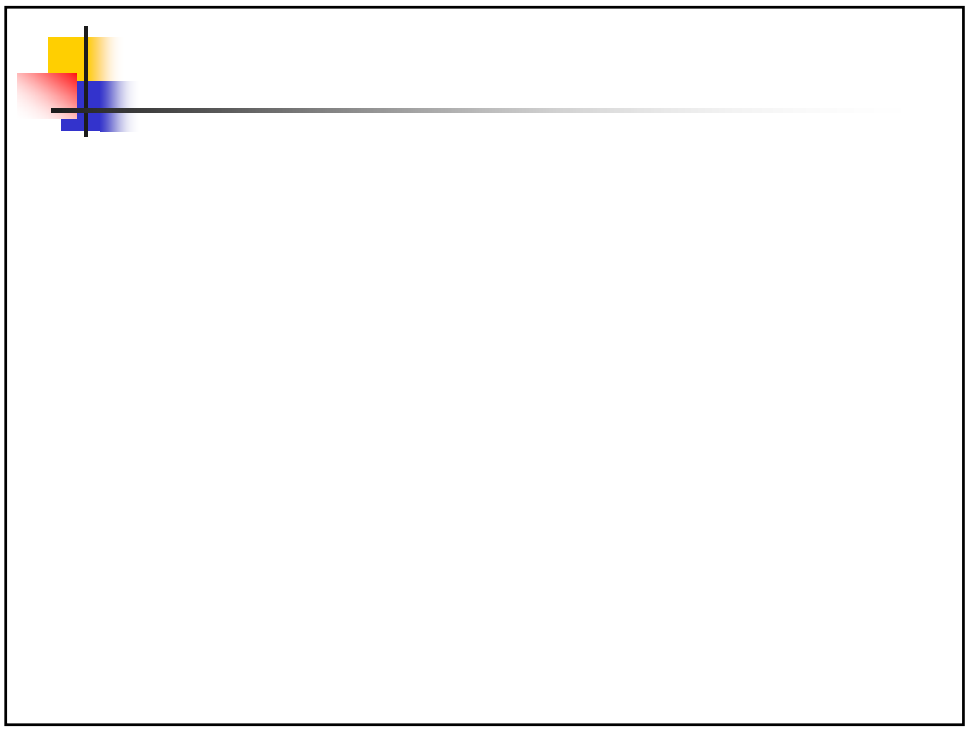
- C++ : string 클래스를 이용한 문자열 처리 가능

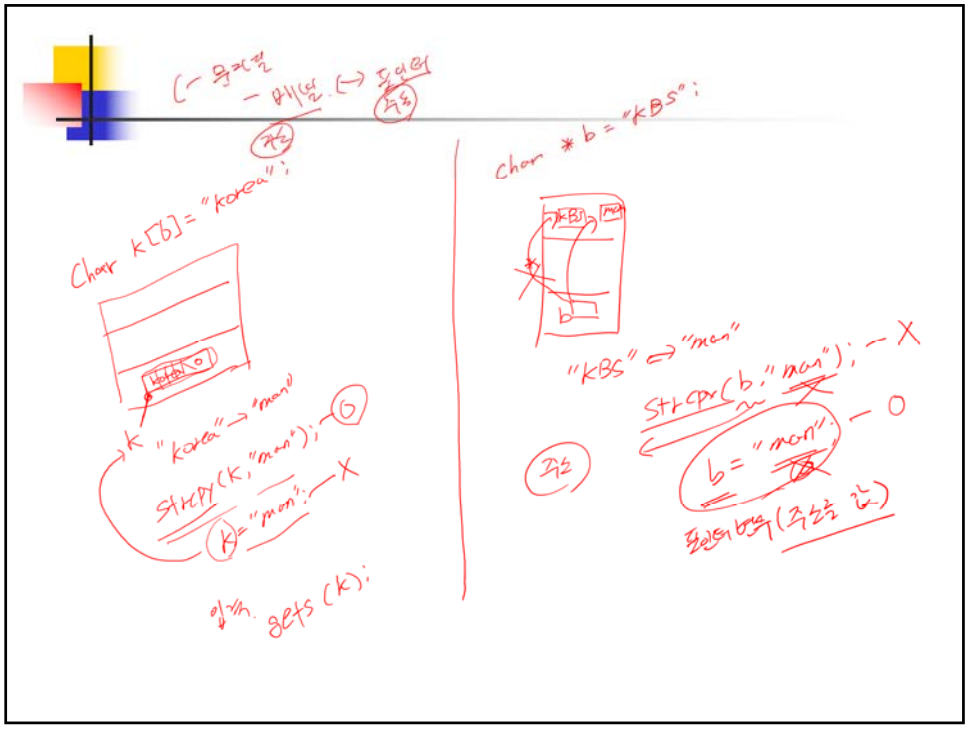
```
string str = "programming";
```

- Java : char 배열을 이용한 문자열 처리 가능

```
char[ ] str = {'p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g'};
```







05_문자열 타입

- Java : string 클래스를 이용한 문자열 처리

```
string str = "programming";
```

- 0으로 끝나는 문자열 형식
  - C, C++, Java 등의 다양한 언어에서 사용
  - 예: 메모리에 표현된 'String'

'S'	't'	'r'	'i'	'n'	'g'	0

  - 0은 1바이트를 차지하므로, 총 7바이트 차지

06_사용자 정의 순서 타입

- 순서 타입(ordinal type)
  - 변수가 가질 수 있는 값들을 나열해 놓은 타입으로 양의 정수 집합과 연관
  - 열거 타입
    - 가질 수 있는 모든 값들은 타입 정의에서 정해지는 이름 상수
    - 이름 상수를 열거 상수라 함
    - C 예제

```
enum day {SUN, MON, TUE, WED, THU, FRI, SAT};

enum months {JAN=1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP,
             OCT, NOV, DEC};

enum day currentDay;
```

06_사용자 정의 순서 타입

- 가독성을 향상시키기 위해 사용

```
if (currentDay == 1)        ➔   if (currentDay == MON)

currentDay = MON;           ➔   currentDay = 1;
```

- C++에서는 열거 타입 변수에 수치 값을 배정하면 오류
- 캐스트 연산자 사용하여 명시적인 타입 변환으로 해결

```
currentDay = (day)1;
```

06_사용자 정의 순서 타입

- 부분 범위 타입
  - 미리 정의된 열거 타입 또는 정수의 부분 집합을 값으로 하는 타입
  - Pascal과 Ada에서 제공
    - Ada에서는 subtype을 사용해서 부분 범위 타입을 정의
  - 예 : Ada에서 day라는 열거 타입을 정의

```
type day is (SUN, MON, TUE, WED, THU, FRI, SAT);
```

    - day 타입의 부분 집합을 값으로 하는 부분 범위 타입인 workday 정의

```
subtype workday is day range MON..FRI;
```

    - 정의된 workday 타입의 변수 선언

```
today : workday;
```

06_사용자 정의 순서 타입

- 변수 today에 workday 타입에 속하는 값의 배정

```
today := TUE;
```

- day에는 속하나 workday에 속하지 않는 값은 workday 타입 변수에 배정할 수 없다.

07_배열 타입

- 구조적 데이터 타입
  - 여러 데이터를 묶어서 하나의 단위로 처리하는 데이터 타입
  - 배열, 레코드, 공용체로 구분

  - 배열
    - 집합체의 첫 번째 원소의 상대적 위치로 원소를 식별하는 동질형 데이터의 모임
    - 배열의 원소는 배열 이름과 첨자에 의해 참조
      - FORTRAN 또는 Ada의 예 : A(2)
        - 부프로그램 호출 형식과 유사하여 구별이 어려움
      - ALGOL 60, PASCAL, C의 예 : A[2]
      - C의 예 : int A[5]
        - 이름 A, 크기 5, 원소의 타입이 int인 배열

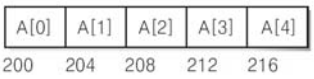
07_배열 타입

- Ada의 예 : A: array (−2..2) of integer
  - 첨자가 −2로 시작하는 배열 선언

- 1차원 배열 원소의 주소
  - 첫 번째 원소의 첨자가 a, 배열의 시작 주소가 base,
    각 원소의 크기가 size일 때 A[i]의 주소는?

A[i]의 주소 = base + (i - a) × size

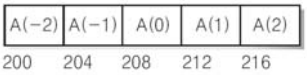
  - 예 : A[3]의 주소 계산하기



    - A[0]의 시작 주소는 200이고, 각 원소의 크기는 4바이트라고 가정
      base + (i − a) × size
      = 200 + (3 − 0) × 4
      = 212

07_배열 타입

- 예 : A(0)의 주소 계산하기



  - A(−2)의 시작 주소는 200이고, 각 원소의 크기는 4바이트라 가정

    base + (i − a) × size
    = 200 + (0 − (−2)) × 4
    = 208

- 2차원 배열
  - 예: A[1][2], A(1, 2)
  - C, C++, Java에서의 배열 선언

```
int A[3][2];                    ··· C
int[ ][ ] A = new int [3][2];   ··· Java
```

07_배열 타입

- FORTRAN, Ada에서의 배열 선언

```
INTEGER A(3, 2)                 ··· FORTRAN
A: array (1..3, 1..2) of integer;   ··· Ada
```

- 예

```
int A[3][2];
```

	1열	2열
1행	A[0][0]	A[0][1]
2행	A[1][0]	A[1][1]
3행	A[2][0]	A[2][1]

- 메모리에 저장되는 방식
  - 행 중심 저장 방식 : 먼저 첫째 행, 다음에 둘째 행, … 식으로 저장
    - 대부분의 언어에서 사용
  - 열 중심 저장 방식 : 먼저 첫째 열, 다음에 둘째 열, … 식으로 저장
    - FORTRAN에서 사용

07_배열 타입

- int A[3][2];

(a) 행 중심 저장 방식

(b) 열 중심 저장 방식

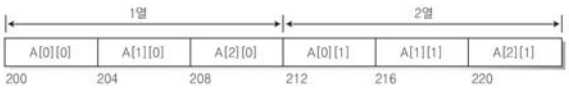
07_배열 타입

- 2차원 배열 원소의 주소
  - A[n][m]의 첫 번째 원소의 행과 열의 첨자는 a, 시작 주소 base, 원소의 크기는 size일 때, A[i][j]의 주소는?

행 중심 저장 방식 : A[i][j]의 주소 = base + (m×(i-a) + (j-a)) × size
열 중심 저장 방식 : A[i][j]의 주소 = base + (n×(j-a) + (i-a)) × size

  - 예 : A[3][2] 배열의 A[2][0]의 주소 값은?
    - 시작 주소 200, 각 원소의 크기를 4로 가정

(a) 행 중심 저장 방식



(b) 열 중심 저장 방식

07_배열 타입

- 행 중심 저장 방식의 A[2][0] 주소
  base + (m×(i-a) + (j-a)) × size
  = 200 + (2×(2-0) + (0-0)) × 4
  = 216
- 열 중심 저장 방식의 A[2][0] 주소
  base + (n×(j-a) + (i-a)) × size
  = 200 + (3×(0-0) + (2-0)) × 4
  = 208

- C/C++에서 배열의 초기화
  - int intArray[5] = {1, 2, 3, 4, 5};
  - int matrix[2][3] = {1, 2, 3, 4, 5, 6};
  - int intArray[] = {1, 2, 3, 4, 5};
    ➔ 배열 크기 생략 시 초기 값의 개수에 따라 배열 크기 설정
  - int matrix[][3] = {1, 2, 3, 4, 5, 6};
    ➔ 행의 크기만 생략 가능

08_레코드 타입

- 레코드
  - 집합체의 원소를 이름으로 식별하는 이질형 데이터의 모임
  - 1960년대 초기에 COBOL에 도입
  - 레코드 정의와 선언 예 : Pascal
    - name, number, address 데이터를 하나의 레코드로 묶음

```
type
  student = record
    name: packed array[1..20] of char;
    number: integer;
    address: packed array[1..30] of char
  end;
```

- 정의된 레코드인 student 타입의 변수 A 선언

```
var A: student;
```

08_레코드 타입

- 레코드 정의와 선언 예 : C

```
struct student {
   char name[20];
   int number;
   char address[30];
};

struct student A;
```

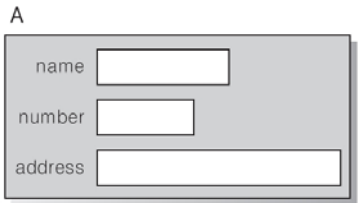
- 레코드 정의와 선언 예 : Ada

```
type student is record
   name: string (1..20);
   number: integer;
   address: string (1..30);
end record;

A: student;
```

08_레코드 타입

- 레코드 타입 변수 A의 구조도



  - 필드(field): 레코드를 이루고 있는 데이터
  - 필드 접근 시, C/Pascal/Ada는 변수 이름.필드이름 사용

- 예 : 변수 A의 number 필드에 10 저장하기
  - A.number = 10; … C
  - A.number := 10; … Pascal, Ada
  - NUMBER OF A … COBOL

08_레코드 타입

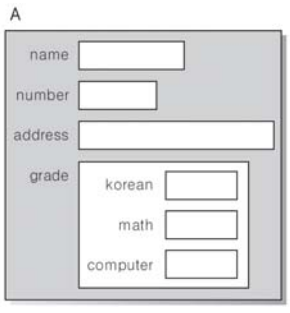
- 다른 레코드를 필드로 지정할 수 있음
  - Ada) student 레코드의 grade 필드가 score 레코드 타입

```
type score is record
   korea: integer;
   math: integer;
   computer: integer;
end record;
type student is record
   name: string (1..20);
   number: integer;
   address: string (1..30);
   grade: score;
end record;

A: student;
```

08_레코드 타입

- 중첩된 레코드 타입 변수 A의 구조도



  - 예) math 필드 접근 방법 : A.grade.math

08_레코드 타입

- Ada) 한 번에 레코드 타입 변수의 모든 필드에 값 배정 가능

```
type score is record
  korean: integer;
  math: integer;
  computer: integer;
end record;
student: score;
student := (70, 80, 90);
```

- Ada) 동등 연산자 적용 가능

```
type score is record
  math: integer;
  computer: integer;
end record;
student1, student2: score;
student1 := (70, 80);
student2 := (70, 80);
if student1 = student2 then
  put("equal");
else
  put("not equal");
end if;
```

09_공용체 타입

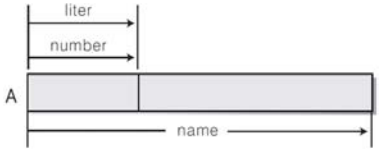
- 공용체
  - 레코드와 형식이 유사
  - 구조체와 달리 모든 필드가 같은 메모리를 공유하면서 필요에 따라 한 필드만을 사용할 수 있음
  - 개념 설명 예
    - 음료수는 float 타입의 liter(용량), 과일은 int 타입의 number(개수), 꽃은 char 배열의 name(이름)으로 가정
      - liter, number, name은 동시에 사용되는 경우는 없고 상황에 따라 하나만 사용되는데 이러한 경우에 공용체를 사용하는 것이 바람직
  - union을 사용한 C/C++ 공용체 표현

```
union product {
  float liter;
  int number;
  char name[10];
};

union product A;
```

08_레코드 타입

- 공용체 타입 변수 A의 구조도



- 예) liter 필드 접근 방법 : A.litter
- 프로그래밍 예

```
switch(type) {
  case DRINK:
    printf("%f\n", A.liter);
    break;
  case FRUIT:
    printf("%d\n", A.number);
    break;
  case FLOWER:
    printf("%s\n", A.name);
    break;
  default:
    printf("bad type\n");
}
```

08_레코드 타입

- Ada) 가변 레코드 – 판별자 이용
  - 판별자 kind에 따라 사용되는 필드가 달라짐

```
type Class is (DRINK, FRUIT,
FLOWER);
type product(kind: Class) is record
  name: string(1..5);
  no: integer;
  case kind is
    when DRINK =>
      liter: float;
    when FRUIT =>
      size: integer;
      number: integer;
    when FLOWER =>
      bunch: integer;
  end case;
end record;
```

08_레코드 타입

- 정의된 가변 레코드 product 타입의 변수 선언

```
goods1: product(DRINK);
goods2: product(FRUIT);
```

  - goods1 : name, no, liter 필드로 이루어짐
  - goods2 : name, no, size, numbe 필드로 이루어짐

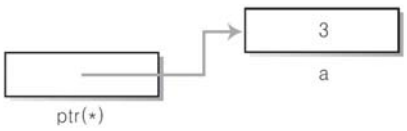
- 각 필드에 값을 배정하는 문장

```
goods1.name := "cider";
goods1.no := 1;
goods1.liter := 1.5;
goods2:= (kind=>FRUIT, name=>"apple", no=>2, size=>3, number=>5);
```

10_포인터 타입

- 포인터의 개요
  - 포인터 타입
    - 변수가 임의의 객체를 참조하기 위해 메모리 주소를 값으로 갖는 타입
    - Pascal, C, C++, Ada 등 최근 언어들은 포인터 개념을 제공
  - 포인터 개념이 도입된 큰 이유
    - 기억장소의 동적 관리
    - 동적으로 할당받는 메모리 공간 : heap
  - 포인터 개념

```
01  int a=3;
02  int *ptr;
03  ptr = &a;
04  printf("%d", *ptr);
```



10_포인터 타입

- 포인터 변수 선언 시, 변수 이름 앞에 * 를 붙임
- ptr : 포인터 변수
  - int 타입의 데이터를 저장하고 있는 메모리 주소를 저장
  - 포인터 변수에 *를 붙이면, ptr이 가리키는 곳을 의미

- 포인터를 이용한 동적 기억 장소 관리
  - 예: 구조체 선언, 포인터 타입 변수 선언 후 동적 메모리 할당

```
struct list {
   int data;
   struct list *next;
};
```

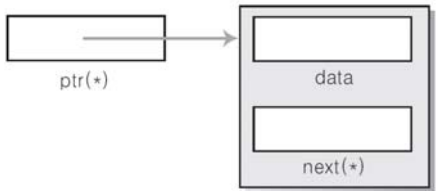
```
struct list *ptr;
```

  - 생성되는 ptr은 struct list 타입 데이터를 저장하고 있는 메모리 주소를 저장할 수 있음
  - malloc 함수 사용, 동적으로 메모리를 할당한 후 ptr이 이 영역을 가리킴

10_포인터 타입

```
ptr = (struct list *)malloc(sizeof(struct list));
```

- malloc 함수 사용, 동적으로 메모리를 할당한 후 ptr이 이 영역을 가리킴

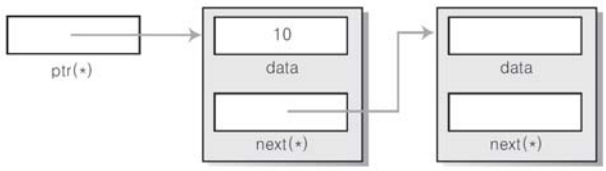


- ptr이 가리키는 영역의 data에 접근하는 표현 형식 : ptr->data
  - data에 10을 저장하는 방법 : ptr->data = 10;

10_포인터 타입

- 또 다른 메모리 영역을 동적으로 할당받고 ptr->next가 이 영역을 가리키게 하는 코드

```
ptr->next = (struct list *)malloc(sizeof(struct list));
```



- 새롭게 생성된 영역의 data에 20 저장

```
ptr->next->data = 20;
```

- 다른 영역을 가리키고 있지 않음을 의미하는 NULL을 저장

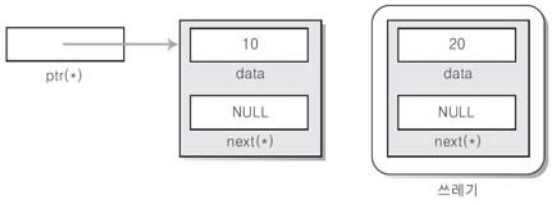
```
ptr->next->next = NULL;
```

10_포인터 타입

- 이 상태에서 다음을 실행

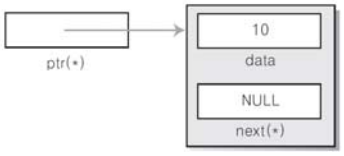
```
ptr->next = NULL;
```

- 쓰레기 발생 : 20을 저장한 영역은 더 이상 접근 불가



- 동적으로 할당된 영역을 회수하기 위한 free 함수

```
free(ptr->next);
ptr->next = NULL;
```



10_포인터 타입

- Ada를 이용한 동적 메모리 관리

```
01  with TEXT_IO;
02  use TEXT_IO;
03  procedure pointer is
04  package INT_IO is new TEXT_IO.INTEGER_IO (integer);
05  use INT_IO;
06  type list;
07  type ptrList is access list;
08  type list is record
09                                          data: integer;
10                                          next: ptrList;
11  end record;
12  ptr: ptrList;
13  begin
14  ptr := new list'(10, null);
15  ptr.next := new list'(20, null);
16  put(ptr.data);
17  put(ptr.next.data);
18  end pointer;
```

10_포인터 타입

- 참조 타입
  - C++는 참조 타입이라는 포인터 타입을 추가적으로 제공
    - 참조 타입 변수는 변수 선언과 동시에 반드시 초기화되어야 함
    - 그 후에는 값을 변경할 수 없음
    - 참조 타입 변수는 선언할 때 변수 이름 앞에 &를 붙임
    - C++ 예

```
01  int val = 10;
02  int &ref = val;
      :
03  ref = 20;
```

      - 참조타입 변수 ref 선언 후, val과 ref는 이름만 다른 같은 변수가 됨
      - 3행 실행 후 val도 20이 됨

10_포인터 타입

- 부프로그램의 형식 매개 변수로 사용할 때 매우 유용
  - 포인터를 이용해서 main의 count 변수의 값을 1 증가시키는 예

```
void incr (int *ptr)
{
  (*ptr)++;
}
int main (void)
{
   ⋮
  incr (&count);
   ⋮
}
```

10_포인터 타입

- 참조 타입 변수를 이용해서 main의 count 변수의 값을 1 증가시키는 예

```
void incr (int &ref)
{
  ref++;
}
int main (void)
{
   ⋮
  incr (count);
   ⋮
}
```

  - 판독하기가 쉽고, 프로그램의 안정성이 높다.