

Programming Language Principle: Types

4 Types

Type เป็นชนิดข้อมูลที่กำหนดคุณลักษณะในการจัดเก็บในหน่วยความจำไว้แล้ว ว่าใช้มีขนาดกี่บิตแต่ละบิตให้ความหมายอย่างไร ขอบเขตของข้อมูลเป็นอย่างไร (maximum and minimum values) มีรูปแบบเป็นอย่างไร และสามารถถูกดำเนินการอย่างไรได้บ้าง ตัวอย่างชนิดข้อมูล เช่น integer, floating point, character(ตัวอักษร), string, pointer เป็นต้น

Type จะเป็นบริบทหนึ่งในการกำหนดการดำเนินการต่าง ๆ โดยปริยายว่าจะต้องดำเนินการอย่างไรโดยไม่ต้องระบุเพิ่มเติม ตัวอย่างเช่น นิพจน์ $a + b$ ในการดำเนินการของเครื่องหมายบวกจะดำเนินการอย่างไรนั้น ก็ขึ้นอยู่กับชนิดของตัวแปร a และ b ถ้าหากเป็น integer ก็จะการคำนวณทางคณิตศาสตร์บวก 2 จำนวนนี้เข้าด้วยกันหรือถ้าหากเป็น string ก็จะกลายเป็นการต่อข้อความเข้าด้วยกัน

หรือกรณีการสร้างคลาสด้วยคำสั่ง `new MyType()` โดยเป็นอ็อบเจกต์โดยในการการประกาศคลาสจะมีโครงสร้างซึ่งประกอบด้วยแอตทริบิวต์และเมธอดต่าง ๆ มีลักษณะดังนี้

```
class MyType {
    att1;
    att2;
    method1;
    method2;
}
```

โดยจากคำสั่งก็จะดำเนินการจองเนื้อที่ในหน่วยความจำตามโครงสร้างของคลาสที่ได้กำหนดไว้ ซึ่งในรายละเอียดการจองขนาดเนื้อที่เท่าใดหรือตำแหน่งไม่จำเป็นต้องระบุไว้ในข้อความสั่ง

Type ยังช่วยจำกัดการดำเนินการต่าง ๆ ที่จะถูกเรียกใช้งานให้ทำงานได้เฉพาะที่ถูกต้องตามหลักการทำงานของภาษาโปรแกรม เช่น ป้องกันบวกข้อมูลชนิด `char` กับ `struct` ซึ่งต่างชนิดได้ หรือ ป้องกันการส่งข้อมูลชนิดไฟล์ให้กับโปรแกรมย่อยที่รับอาร์กิวเมนต์เป็นชนิด `integer` เป็นต้น

4.1 Common Types

Common Type ในภาษาโปรแกรมจะแบ่งประเภทได้เป็นดังนี้

4.1.1 Discrete types

คือ ชนิดข้อมูลมีลักษณะไม่ต่อเนื่องกัน โดยแต่ละค่าในกลุ่มมีการแบ่งแยกระหว่างกันของแต่ละค่าอย่างสิ้นเชิง โดยจะมีลักษณะนับหรือระบุได้ที่มีการบ่งบอกถึงค่าก่อนหน้า (predecessor) และค่าตามหลัง (successor) ตัวอย่างชนิดข้อมูลประเภทนี้ เช่น integer, Boolean, char, enumeration (ค่าแจงนับ), subrange

4.1.2 Scalar types

คือ ชนิดข้อมูลที่มีค่าแบบเดี่ยว (single-value) เช่น discrete, real เป็นต้น

4.1.3 Composite types

คือ ค่า non-scalar หมายความว่าไม่ใช่ค่าแบบเดี่ยว เป็นการนำซึ่งเป็นการนำชนิดข้อมูลหนึ่งหรือหลาย ๆ ที่ซับซ้อนน้อยกว่าชนิดมาสร้างเป็นชนิดข้อมูลใหม่ เช่น array ที่มีการเก็บข้อมูลชนิดเดียวกันที่สามารถเก็บได้หลาย ๆ ค่า หรือ string เป็นการต่อกันของข้อมูลชนิด character หรือ ข้อมูลชนิดอื่น ๆ เช่น record (struct), set, pointer, list, file ฯลฯ

4.2 Type System

ในแต่ละภาษาโปรแกรมจะมี Type System แตกต่างกันไป โดยจะประกอบไปด้วย

4.2.1 กลไกในการนิยามชนิดข้อมูลและการนำไปใช้ตามโครงสร้างของภาษา (A mechanism to define types and associate them with certain language constructs)

ส่วนของกลไกการนิยามก็มีวิธีการให้โปรแกรมเมอร์สามารถนิยามชนิดข้อมูลใหม่ (composite type) หรือจะเป็นแบบที่ภาษาโปรแกรมเตรียมไว้ให้ (predefined types)

ส่วนของการนำไปใช้ภาษาโปรแกรมก็จะกำหนดวิธีการต่าง ๆ ที่จะนำชนิดข้อมูลที่นิยามไว้ไปใช้ในการเขียนโปรแกรม เช่น การนำไปสร้างตัวแปร การใช้เป็นชนิดผลลัพธ์ของฟังก์ชัน การใช้เป็นอาร์กิวเมนต์ของฟังก์ชัน เป็นต้น ต่อไปจะแสดงตัวอย่างของภาษาต่าง ๆ

ต่อไปนี้เป็นตัวอย่างของภาษา C

```
1: //C struct
2: typedef struct pnt {
3:     int x, y;
4: } Point;
5: Point point_new (int x, int y) { ... }
```

จากตัวอย่าง ตั้งแต่บรรทัดที่ 2 – 4 เป็นกลไกการประกาศชนิดข้อมูลชนิด struct ประกอบด้วย ค่า integer 2 ตัว คือ x และ y โดยให้มีชื่อว่า Point และในบรรทัดที่ 5 แสดงการนำชนิดข้อมูลที่ประกาศไปใช้โดย เป็นผลลัพธ์ของฟังก์ชัน point_new

ต่อไปนี้เป็นตัวอย่างกลไกการประกาศชนิดข้อมูลของภาษา Pascal

```
1: (* Pascal array, subrange *)
2: type
3:     ch_array = array[char] of 1..26;
4:     test_score = 0..100;
5: var
6:     alphabet: ch_array;
7:     score: test_score;
```

จากตัวอย่างในบรรทัดที่ 2 – 4 เป็นการประกาศชนิดข้อมูลชื่อ `ch_array` โดยมีโครงสร้างเป็น array ของ `char` มีค่าตั้งแต่ 1 ถึง 26 และชื่อ `test_score` มีโครงสร้างเป็น `subrange` มีค่าตั้งแต่ 0 ถึง 100 และบรรทัดที่ 5 – 7 เป็นการนำชนิดข้อมูลที่สร้างไว้มาใช้โดยประกาศเป็นตัวแปร

ต่อไปนี้เป็นตัวอย่างกลไกการประกาศชนิดข้อมูลของภาษา Java

```
1: //Java enumeration, class
2: public enum Day {
3:     SUNDAY, MONDAY, TUESDAY,
4:     WEDNESDAY, THURSDAY, FRIDAY,
5:     SATURDAY
6: }
7: public class EnumTest {
8:     Day day;
9:     ... }
10: ...
11: EnumTest firstDay;
```

จากตัวอย่างเป็นการประกาศชนิดข้อมูลใหม่โดยภาษา Java แสดงเป็น 2 แบบ โดยในบรรทัดที่ 2 – 6 แสดงการประกาศชนิดข้อมูลชื่อ `Day` เป็นชนิด `enum` เป็น `enumeration` ที่ประกอบไปด้วยข้อมูลชื่อวันต่าง ๆ และถูกนำไปใช้ไปประกาศเป็นแอตทริบิวต์ของคลาสในบรรทัดที่ 8 ขณะที่บรรทัดที่ 7 – 9 เป็นการประกาศคลาสชื่อ `EnumTest` ซึ่งถูกนำไปประกาศเป็นตัวแปรในบรรทัดที่ 11

4.2.2 Type equivalence rules

ในกรณีที่ต้องการเปรียบเทียบข้อมูล 2 ค่า ภาษาโปรแกรมต้องกำหนดกฎเพื่อที่จะแสดงได้ว่า 2 ค่านั้นเหมือนกันหรือไม่ ภาษาโปรแกรมที่มี `equivalence rule` (ส่วนใหญ่จะเป็นภาษารุ่นเก่า) จะบอกว่าชนิดข้อมูลหนึ่งมีโครงสร้างตรงกันกับอีกชนิดหนึ่งซึ่งถือว่าเป็นชนิดเดียวกัน สร้างใช้แทนกันได้

4.2.3 Type compatibility rules

`compatibility rule` จะเป็นตัวกำหนดว่าข้อมูลชนิดไหนสามารถอนุโลมให้สามารถดำเนินการกับข้อมูลชนิดไหนได้อย่างไร อาจมีการแปลงข้อมูลบางอย่างเพื่อให้สามารถทำงานร่วมกันได้ อาจจะต้องมีการแปลงข้อมูลบางอย่างเพื่อให้ตรงตามบริบทที่นำไปใช้

4.2.4 Type inference rules

การอนุมาน คือ เมื่อทราบข้อเท็จจริงใด ๆ แล้ว สามารถนำมาใช้เพื่อประมวลผลเป็นองค์ความรู้ใหม่ ซึ่งจะใช้ในการอนุมานชนิดของผลลัพธ์ของนิพจน์จากองค์ประกอบของนิพจน์ หรือบริบทของนิพจน์นั้น

4.3 Type Checking

เป็นกระบวนการเพื่อให้แน่ใจว่าชนิดข้อมูลต่าง ๆ ที่ถูกใช้ในโปรแกรมนั้นเป็นไปตาม compatibility rule ของภาษาที่กำหนดไว้ โดยการใช้นิยามชนิดข้อมูลที่ฝ่าฝืนกฎนั้นจะถูกรเรียกว่า type clash ตัวอย่างเช่น การ assign ค่าข้อมูลชนิด string ไปยังตัวแปรที่ประกาศไว้เป็นชนิด int ของภาษาซี ซึ่งเป็นการฝ่าฝืนกฎของภาษาที่จะส่งผลให้การ compile ไม่ผ่าน ดังนี้

```
1: //c
2: int a;
3: a = "xyz";    //clash
```

4.3.1 Static vs. Dynamic Typing

ในภาษาโปรแกรมแบ่งตามกลไกการนิยามชนิดข้อมูลจะแบ่งเป็น 2 ประเภท คือ statically-typed language และ dynamically-typed language

โดยใน statically-typed language ชนิดข้อมูลจะถูกผูกกำหนดไว้ให้กับตัวแปรนั้นคือในการประกาศตัวแปรจะต้องระบุชนิดข้อมูลด้วยซึ่งจะไม่มีการเปลี่ยนแปลงอีกหมายความว่าตัวแปรจะไม่สามารถอ้างถึงชนิดข้อมูลที่ไม่ใช่ชนิดที่ถูกประกาศเอาไว้ (เช่นในภาษา Pascal, Java, C, C# เป็นต้น) และกระบวนการ type checking จะถูกทำในช่วงของการ compile เช่นตัวอย่างการประกาศตัวแปรดังนี้

```
1: //Java
2: String s = "abcd";    //s will forever be a string
```

จากตัวอย่างเป็นการประกาศตัวแปร s เป็นชนิด String ซึ่งต้องรับข้อมูลชนิด String หรือ compatible เท่านั้น

การมี Statically-typed language จะได้ประโยชน์ในการ compile เพราะทราบชนิดข้อมูลของตัวแปรแล้วทำให้สามารถที่จะทำ type checking ได้ก่อนที่จะนำไปประมวลผล ทำให้มี efficient ตอนทีรันเพราะการตรวจสอบการใช้งาน type ที่ผิดเกือบทั้งหมดก่อนแล้ว แต่ก็ยังมีส่วนที่ต้องทำ type checking ในช่วงการรัน เช่นตัวอย่างโค้ดภาษา C ดังต่อไปนี้

```
1: //C
2: int n;  intiarr[3];
3: ...
4: iarr[n] = 5;
```

จากตัวอย่างมีการอ้าง index ของตัวแปร array ในบรรทัดที่ 4 ถูกอ้างด้วยตัวแปร n ซึ่งเรายังไม่ทราบค่าที่แท้จริงของตัวแปร n ณ ช่วง compile การตรวจสอบ type checking ว่าละเมิดการใช้ชนิดข้อมูลว่าเกินขอบเขต array หรือไม่นั้นจึงจำเป็นต้องดำเนินการในช่วงที่โปรแกรมรัน ว่าค่า n ณ ขณะนั้นไม่ได้อยู่ในช่วง 0 - 2 หรือไม่

ส่วนสำหรับ Dynamically-typed language นั้นชนิดข้อมูลของตัวแปรจะขึ้นอยู่กับชนิดของข้อมูลที่ถูก assign ให้กับตัวแปร นั่นคือ ชนิดของตัวแปรหนึ่งจะเปลี่ยนไปตามข้อมูลที่ถูกนำมา assign ให้ ทำให้การประกาศตัวแปรไม่ต้องมี type กำกับ ดังตัวอย่างภาษา Python ต่อไปนี้

```
1: #Python
2: s = "abcd"    # s is a string
```

```

3: s = 123          # s is now an integer
4: p = s - 1

```

จากตัวอย่าง ตลอดการทำงานของโปรแกรมตัวแปรชื่อ `s` ถูกกำหนดชนิดข้อมูล 2 ชนิด คือ `string` ในบรรทัดที่ 2 และเปลี่ยนเป็น `integer` ในบรรทัดที่ 3 โดยกระบวนการ `Type checking` ทั้งหมดจะถูกทำในช่วงของการรันโปรแกรม ดังนี้ เราจะทราบว่าในบรรทัดที่ 4 จะทราบได้ว่าใช้งานชนิดข้อมูลได้ถูกต้องหรือไม่ก็ต้องรอช่วงการรันโปรแกรมถึงบรรทัดนั้น ว่าตัวแปร `s` เป็นชนิดที่สามารถทำงานได้ตามกฎหรือเปล่า อย่างไรก็ตามนี้สามารถทำงานได้เพราะ `s` เป็นชนิดตัวเลขแล้วซึ่งภาษา Python อนุญาตให้ทำการลบกันได้ แต่ถ้าหากในกรณีที่ไม่มีคำสั่งบรรทัดที่ 3 โปรแกรมก็จะเกิด `Error` ขึ้นเพราะไม่อนุญาตให้นำ `string` ลบกับตัวเลข

4.3.2 Type Equivalence Rules

ในภาษาโปรแกรมที่อนุญาตให้ผู้ใช้สามารถประกาศชนิดข้อมูลได้นั้น เมื่อนำชนิดข้อมูลเหล่านั้นมาเทียบจะสามารถทำได้ 2 ลักษณะต่อไปนี้

4.3.2.1 Structural equivalence

ปรากฏอยู่ในภาษารุ่นเก่า ๆ โดยการเปรียบเทียบจะดูจากโครงสร้างของข้อมูลภายใน ว่ามีองค์ประกอบเหมือนกันหรือไม่ ถ้าเหมือนกันชนิดข้อมูลนั้นเทียบเท่ากัน ตัวอย่างภาษาโปรแกรมที่มีการเปรียบเทียบเชิงโครงสร้าง เช่น Algol-68, Modula-3, (to some extent) C, ML เป็นต้น

ตัวอย่างการเปรียบเทียบเชิงโครงสร้าง พิจารณาจากโค้ดตัวอย่างต่อไปนี้

```

1: type R1 = record
2:     a, b : integer
3: end;
4: type R2 = record
5:     a : integer;
6:     b : integer
7: end;
8: type R3 = record
9:     b : integer;
10:    a : integer
11: end;

```

จากโค้ดตัวอย่างมีการประกาศชนิดข้อมูล `record` โดยมีโครงสร้างประกอบด้วยตัวแปรสมาชิกชนิด `integer` 2 ตัวคือ `a` และ `b` โดยมีการสลับลำดับในชนิดข้อมูลชื่อ `R3` ดังนั้นในการเปรียบเทียบ ระหว่าง `R1`, `R2` และ `R3` เมื่อพิจารณาโครงสร้างที่ประกอบด้วยสมาชิกชนิด `integer` 2 ตัวเหมือนกัน เมื่อจัดเก็บในหน่วยความจำจะมีโดยโครงสร้างเดียวกัน เช่น การจองหน่วยความจำสำหรับเก็บค่า `integer` 2 ตำแหน่งติดกัน ถ้าในภาษาที่สนใจเฉพาะลำดับองค์ประกอบที่เป็นชนิดข้อมูลก็จะมองว่าทั้ง 3 `type` ที่ประกาศนั้นมีค่าชนิดข้อมูลเท่ากัน แต่สำหรับบางภาษาที่สนใจลำดับของชื่อตัวแปรด้วยก็จะมอง `R2` และ `R3` ไม่เท่ากัน เพราะลำดับชื่อตัวแปรต่างกันแบบนี้จะพบในภาษา ML

ตัวอย่างถัดไปจะแสดงประเด็นที่จะเป็นปัญหาในการเปรียบเทียบเชิงโครงสร้าง โดยพิจารณาจากโค้ดต่อไปนี้

```

1: type student = record
2:     name, address : string
3:     age: integer
4: type school = record
5:     name, address : string
6:     age : integer
7: x : student; y : school;
8: ...
9: x := y;

```

จากโค้ดจะเป็นว่าข้อมูลชนิด student กับ school มีโครงสร้างภายในที่เหมือนกันตามการเปรียบเทียบเชิงโครงสร้าง ทำให้การทำงานในบรรทัดที่ 9 ซึ่ง assign ข้อมูลชนิด school ให้กับตัวแปรชนิด student สามารถทำงานได้เพราะ type เทียบเท่ากัน แต่ในทางความหมายแล้ว เมื่ออ่านโค้ดการใช้ตัวแปรชนิด student เก็บข้อมูลชนิด school จะให้ค่าที่ไม่สมเหตุผล เช่น ค่าอายุ (age) ของ school ที่สามารถเกิน 100 ปีได้แต่อาจจะไม่ปกติสำหรับนักเรียนที่จะมีอายุเกิน 100 ปี แต่การที่ compiler อนุญาตให้ assign ซึ่งกันและกันอาจเกิดการ assign โดยความไม่ตั้งใจและทำให้เกิดการผิดพลาดในการทำงานของโปรแกรมได้

ในทำนองเดียวกันปัญหานี้ก็อาจเกิดกับชนิดข้อมูลในภาษาโปรแกรมที่อนุญาตให้ประกาศ type ประเภทค่าเดียวได้ และมี structural equivalent ด้วย เช่น โค้ดภาษา C ต่อไปนี้

```

1: //C structural equivalence for scalar types
2: typedef float celsius;
3: typedef float fahrenheit;
4: ...
5: celsius c; fahrenheit f;
6: ...
7: f = c;

```

จากโค้ดตัวอย่างจะเห็นว่าการประกาศชนิดข้อมูลโดยมีโครงสร้างเป็นชนิด float เพื่อเก็บข้อมูลต่างหน่วยนับอุณหภูมิกัน แต่ด้วย structure equivalence ของข้อมูล 2 ชนิด สามารถ assign กันได้โดยตรงไม่ได้กันให้เกิดข้อมูลชนิดเดียวกันความหมายของชื่อชนิดข้อมูล ทำให้อาจเกิดการผิดพลาดที่ใช้ชนิดข้อมูลเก็บค่าต่างหน่วยนับโดยที่ไม่ได้ทำการแปลงก่อนได้ทำให้ข้อมูลที่ได้ผิดเพี้ยนไปได้

4.3.2.2 Name Equivalence

วิธีการนี้จะสนใจที่ชื่อของชนิดข้อมูลหากเป็นชื่อเดียวกันก็หมายถึงเป็นชนิดเดียว และต่างชนิดกันในทางตรงกันข้าม นั่นคือถึงแม้ว่าองค์ประกอบที่ประกาศไว้ภายในชนิดข้อมูลจะเหมือนกันแต่จะถูกมองว่าเป็นต่างชนิดกันหากไม่ได้ใช้ชื่อเดียวกัน ภาษาโปรแกรมที่ใช้วิธีการนี้ เช่น Java, C#, Pascal, Ada เป็นต้น ในปัจจุบันภาษาโปรแกรมมีแนวโน้มว่าจะใช้วิธีการนี้มากกว่าแบบ structural equivalence

ตัวอย่างโค้ดต่อไปนี้จะแสดงความไม่เท่ากันของชนิดข้อมูลที่มีโครงสร้างเหมือนกันแต่คนละชื่อ ดังนี้

```

1: //C
2: typedef struct b1 {
3:     char title[20];
4:     char author[20];
5:     int book_id;
6: } Book1;
7: typedef struct b2 {
8:     char title[20];
9:     char author[20];
10:    int book_id;
11: } Book2;
12: ...
13: Book1 book1; Book2 book2;
14: ...
15: book2 = book1; //no match for operator =, operand types are Book2 and Book1

```

จากตัวอย่างมีการประกาศ type 2 ตัว คือ Book1 และ Book 2 โดยมีโครงสร้างเหมือนกันคือ ประกอบด้วย ชื่อ, ผู้แต่ง และ เลขที่หนังสือ แต่เนื่องจากทั้งสองมีชื่อไม่เหมือนกันและด้วยการเปรียบเทียบแบบ name equivalence ทั้งสอง type จึงถูกมองว่าเป็นคนละชนิดกัน ซึ่งจะทำให้การทำงานของโปรแกรมในบรรทัดที่ 15 เกิด compile error ขึ้น

Exercise 4.1 Structural vs. Name Equivalence

ให้ตอบคำถามโดยพิจารณาจากโค้ดภาษา Java ต่อไปนี้

```

1: //Java
2: class MyCard {
3:     public MyCard() { ... }
4:     public int suit() { ... }
5:     public int rank() { ... }
6:     private int suitValue;
7:     private int rankValue;
8: }
9: class YourCard {
10:    public YourCard() { ... }
11:    public int suit() { ... }
12:    public int rank() { ... }
13:    private int suitValue;
14:    private int rankValue;

```

15: }

จากการประกาศตัวแปร ชื่อ mc ดังนี้

`MyCard mc;`

จงอธิบายว่าแต่ละข้อความสั่งต่อไปนี้ใช้ type equivalence แบบใด? และผลของการทำ type checking ในแต่ละข้อความสั่งเป็นอย่างไร? (English version question: What kind of type equivalence is used in each statement? What happens to each statement when type checking is performed?)

1. `mc = new YourCard();`
2. `mc = (MyCard) new MyCardChild();`
3. `mc = new MyCardChild();`

เฉลย Exercise 4.1

1. `mc = new YourCard();`

คำตอบ: `compile.error.by.name.equivalence`

อธิบายคำตอบ:

เนื่องจาก mc มีชนิดเป็น MyCard แต่ถูก assign ด้วย YourCard ซึ่งเมื่อภาษา Java เป็นแบบ Name equivalence ก็จะถือว่าไม่เท่ากัน ถึงแม้ว่าโครงสร้างภายในจะเหมือนกันก็ตาม ดังนั้นก็จะเกิด compile error นั่นเอง

2. `mc = (MyCard) new MyCardChild();`

คำตอบ: `Compile.and.run.OK.by.name.equivalence.(cast.to.MyCard)`

อธิบายคำตอบ:

จากคำถามจะมีจุด type checking 2 จุด คือ ในจุดของการ assign ค่า จะสามารถทำได้เนื่องจากตัวดำเนินการทั้งสองข้างเป็นชนิดเดียวกันคือ MyCard นั่นคือ นิพจน์ด้านขวาถูกแปลงเป็น MyCard (type casting) ก่อนนั่นเอง

จุดที่ 2 คือ การแปลง MyChildCard เป็น MyCard (type casting) ซึ่งทำ ตาม compatibility rule ของภาษา Java ซึ่งจะทำให้โปรแกรมสามารถทำงานได้โดยไม่เกิด error อีกด้วย แต่ในบางครั้งการ compile อาจผ่านแต่โปรแกรมทำงานไม่ได้เนื่องจากสามารถทราบได้ในตอน compile ว่า type ไม่ใช่ชนิดเดียวกัน เช่น การ cast กับตัวแปรชนิด Object เป็นต้น

3. `mc = new MyCardChild();`

คำตอบ: `Compile.and.run.OK.by.name.equivalence.(cast.to.MyCard)`

อธิบายคำตอบ:

เป็นการลดรูปจากข้อ 2 คือ ไม่ต้องเขียนโค้ดส่วน casting เรียกว่าเป็น loose name equivalence, loose แปลว่าหลวม คือ อนุโลมสำหรับกรณี type 2 type ที่แม้ว่าจะชื่อไม่เหมือนกันแต่มีความสัมพันธ์กันอยู่ เช่นกรณีนี้ คือ superclass-subclass โดยถือว่า subclass compatible กัน superclass ทำให้สามารถจะมอง object ชนิด MyCardChild เป็น object ชนิด MyCard ได้

จบการเฉลย Exercise 4.1

4.3.3 Type Conversion and Cast

ในการเขียนโปรแกรมในนิพจน์ต่าง ๆ จะมีความคาดหวังว่า ชนิดข้อมูลของนิพจน์นั้นถูกต้องตรงกับความต้องการประมวลผลหรือไม่ และบ่อยครั้งที่มีการนำนิพจน์ต่างชนิดข้อมูลกันมาดำเนินการร่วมกัน เช่นนิพจน์

```
a = expression
```

เราก็คาดหมายว่านิพจน์ expression กับตัวแปร a จะต้องเป็นชนิดข้อมูลเดียวกัน แต่หาก expression นั้นไม่ได้มีชนิดข้อมูลเดียวกันกับตัวแปร a ซึ่งอาจจำเป็นต้องมีการเปลี่ยนชนิดข้อมูลให้ตรงตามความต้องการ โดยในภาษาโปรแกรมก็จะมีกลไกในการดำเนินการเช่นนี้ เรียกว่า explicit type conversion เพื่อให้สามารถแปลง expression ให้มีชนิดข้อมูลตรงกันที่ต้องการประมวลผล โดยวิธีการนี้คือการที่โปรแกรมเมอร์ต้องประกาศให้ชัดเจนว่าต้องการแปลงชนิดข้อมูล (type casting) ซึ่งทำให้ต้องมีการเขียนปรากฏอยู่ในโค้ด เช่น สมมติว่าตัวแปร a มีชนิดเป็น T1 แต่ expression ได้ผลลัพธ์เป็นชนิด T2 แล้วต้องการ cast ให้เป็น T1 ต้องมีการแสดงความจำนง ดังโค้ดต่อไปนี้

```
T1 a = (T1) expression
```

โดยการใส่ casting เป็นการบอก compiler ว่าโปรแกรมเมอร์เองทราบอยู่แล้วว่า type นั้นไม่ตรงกันแล้วต้องการให้มีการแปลง type หากไม่มีการใส่ก็จะเกิด compile error หาก type ไม่ตรงกันนั่นเอง

จากข้อความข้างต้นในการทำงานเบื้องหลังจะมีโอกาสเกิดผลการประมวลผลในช่วง run-time ได้ 3 กรณีหลัก ในการแปลงชนิดข้อมูลว่า compiler จะต้องทำการสร้างโค้ดในส่วนของการประมวลเพื่อแปลงข้อมูล (conversion) หรือไม่นั้นก็ขึ้นอยู่กับชนิดข้อมูลที่เข้ามาเกี่ยวข้อง โดยในแต่ละกรณี ได้แก่

1. กรณี type 2 ชนิด ซึ่งมีโครงสร้างเหมือนกัน (Structure Equivalent) แต่ภาษาโปรแกรมเป็นแบบ name equivalence จะไม่มีการกระทำการแปลงข้อมูลใด ๆ เพิ่มเติมในช่วง runtime ตัวอย่างเช่น

```
1:   type student = record
2:     name, address : string
3:     age: integer
4:   type school = record
5:     name, address : string
6:     age : integer
7:   x : student; y : school;
8:   ...
9:   x := y;
```

จากตัวอย่างมีการประกาศชนิดข้อมูล 2 ชนิดที่มีโครงสร้างเหมือนกัน คือ school กับ structure แต่ในภาษาโปรแกรมแบบ name equivalence จะแปลความว่าเป็นชนิดข้อมูลที่ต่างกัน ดังนั้น บรรทัดที่ 9 จะเกิด compile error จากการ assign ค่าต่างชนิดกัน แต่มีวิธีการที่จะทำให้ข้อความสั้นนี้ผ่านการตรวจสอบโดย compiler ได้ โดยเพิ่ม type cast ลงไปได้เป็น

```
x := (student) y
```

แต่สิ่งที่เกิดขึ้นในการคอมไพล์จะเป็นเพียง compiler ไม่สนใจชนิดที่ต่างกัน และสร้างโค้ด assignment ตามปกติ เพียงแต่ไม่เกิด compile error แล้ว นั่นคือ จะไม่ต้องการการสร้างโค้ดใด ๆ เพิ่มเติมเลยเพื่อจะแปลงชนิดข้อมูลในกรณีนี้

แต่ก็ไม่เสมอไปที่จะสามารถทำการ cast ระหว่างชนิดข้อมูล 2 ต่างชนิดที่มีโครงสร้างเหมือนกัน อย่างเช่นใน ภาษา C ที่ไม่อนุญาตให้ cast ข้อมูลชนิด struct ไปยังต่างชนิดกันแม้โครงสร้างเหมือนกันและเบื้องหลังสามารถจะทำได้ก็ตาม

- กรณีที่มีชนิดข้อมูล 2 ชนิดที่มีโครงสร้างต่างกัน แต่มีโครงสร้างบางส่วนที่เหมือนกัน (intersect) เช่น ชนิดข้อมูลที่เป็น subrange ของอีกชนิดข้อมูลหนึ่ง ยกตัวอย่างเช่นมีชนิดข้อมูล 2 ชนิด คือ integer และ test_score ซึ่งมีค่าระหว่าง 0 ถึง 100 มีลักษณะเป็น subrange ของ integer หากมีการประกาศตัวแปร i เป็นชนิด integer และ j เป็นชนิด test_score และเมื่อต้องการ assign ค่าตัวแปร i ให้กับตัวแปร j ผ่านการ cast ดังนี้

```
j = (test_score) i;
```

การ cast นี้อาจจะสามารถทำได้หรือไม่จะขึ้นอยู่กับค่าของตัวแปร i ในขณะนั้น หากค่าอยู่ในช่วงที่ชนิด test_score รับได้ (0 ถึง 100) ก็จะสามารถได้และจะทำได้การณียุ่่นนอกช่วงซึ่งจะเกิดเป็น error ขึ้น โดยการจะทราบผลการ cast นี้ได้ก็ต้องทำในช่วงของ run-time แล้ว

กล่าวคือ ในกรณีที่ทำการ cast จากชนิดข้อมูลหนึ่งที่สามารถเก็บบางค่าที่ชนิดข้อมูลปลายทางไม่สามารถรองรับได้ ในช่วง run-time จึงจำเป็นต้องมีการตรวจสอบว่าค่าของตัวแปร ณ ขณะที่กำลังจะ cast นั้นเป็นค่าที่ถูกต้องตามที่ชนิดปลายทางสามารถรับได้หรือไม่ ซึ่งเป็นหน้าที่ compiler ที่ต้องสร้างโค้ดส่วนนี้เพิ่มเติมนั่นเอง

- กรณีที่ระหว่างชนิดข้อมูลที่จะ cast มี โครงสร้างการเก็บข้อมูลแตกต่างกันอย่างสิ้นเชิง เช่น ระหว่าง integer กับ floating-point แต่สามารถที่จะกำหนดวิธีการแปลงค่าระหว่างกันได้ เช่น การแปลง floating-point เป็น integer อาจทำได้โดยการตัดค่าส่วนทศนิยมออกเหลือเพียงค่าของจำนวนเต็ม เป็นต้น โดยวิธีการแบบนี้ส่วนใหญ่จะมีให้แล้วอยู่ในรูปของชุดคำสั่งภาษาเครื่อง (machine instruction)

จากที่กล่าวมาสามารถสรุปกรณีต่าง ๆ ได้สั้น ๆ ดังนี้

กรณีที่ 1 cast โดยไม่ต้องดำเนินการใด ๆ เพิ่มเติม เพราะโครงสร้างข้อมูลเหมือนกัน และภาษาโปรแกรมนั้นที่ยินยอมให้ทำได้

กรณีที่ 2 cast โดยทำการตรวจสอบความถูกต้องของค่าก่อนในตอน run-time ว่าสามารถทำได้หรือไม่

กรณีที่ 3 convert โดยใช้ machine instruction

ไม่ต้องใช้ cast โปรแกรมทำใน

Exercise 4.2 Type Conversion and Cast

กำหนดให้มีการประกาศตัวแปรชนิดต่าง ๆ ดังต่อไปนี้

```
1: --Ada
2: n : integer;           --assume 32 bits
3: r : long_float;       --assume IEEE double-precision
4: t : test_score;       --type test_score is new integer range 0..100;
5: c : celsius_temp;     --type celsius_temp is new integer;
6: ...
```

ให้อธิบายการ cast (ภาษา Ada จะมีรูปแบบการ cast เหมือนการเรียกฟังก์ชัน คือใช้ชื่อ type ตามด้วยค่าที่ต้องการแปลงอยู่ในวงเล็บ) ที่เกิดขึ้นว่าผลในตอน compile เป็นอย่างไร และในตอน run-time จะเกิดการทำงานอย่างไรจาก 3 กรณีที่ได้กล่าวมาแล้วในแต่ละข้อต่อไปนี้

```
1. t := test_score(n); .....
2. n := integer(t); .....
3. r := long_float(n); .....
4. n := integer(r); .....
5. n := integer(c); .....
6. c := celsius_temp(n); .....
```

เฉลย Exercise 4.2

1. t := test_score(n);

คำตอบ: compile ผ่าน และตอน run-time ต้องตรวจสอบความถูกต้องของค่า n ก่อน

อธิบายคำตอบ: เป็นการแปลงค่า integer ให้เป็น test_score ซึ่งเป็น subrange กัน ซึ่งเป็นไปตามกรณีที่ 2 ที่ integer สามารถมีค่าที่นอกเหนือจากที่ test_score รับได้ จึงต้องตรวจสอบก่อนว่าค่า integer ขณะ assign นั้นอยู่ในช่วงที่ test_score รับได้หรือไม่

2. n := integer(t);

คำตอบ: compile ผ่านและตอน run-time ไม่ต้องดำเนินการใด ๆ เพิ่มเติม

อธิบายคำตอบ: แม้ว่าเงื่อนไขระหว่างชนิดข้อมูลตรงกับกรณีที่ 2 แต่ว่าชนิดข้อมูลปลายทางสามารถรับได้ทุกค่าจากชนิดต้นทาง นั่นคือ integer สามารถเก็บค่า 0 ถึง 100 ได้ จึงไม่ต้องการตรวจสอบความถูกต้องอีก

3. r := long_float(n);

คำตอบ: compile ผ่านและตอน run-time แปลงข้อมูลผ่าน machine instruction

อธิบายคำตอบ: การแปลง integer ให้เป็น floating-point ซึ่งมีโครงสร้างข้อมูลต่างกันจะตรงกับกรณีที่ 3 ที่ทำการแปลงด้วย machine instruction

4. `n := integer(r);`

คำตอบ: compile ผ่านและตอน run-time ต้องตรวจสอบความถูกต้องของค่า r ก่อน

อธิบายคำตอบ: การแปลงตรงตามเงื่อนไขในกรณีที่ 3 ที่ต้องทำด้วย machine instruction และมีส่วนของกรณีที่ 2 ด้วยนั่นคือ integer เป็น subrange ของ long-float นั่นคือสามารถเก็บค่าจำนวนเต็มทีเกินกว่า integer จะรับได้ (64 bit vs 32 bit) จึงต้องมีการตรวจสอบความถูกต้องของค่าก่อน

5. `n := integer(c);`

คำตอบ: compile ผ่านและตอน run-time ไม่ต้องดำเนินการใด ๆ เพิ่มเติม

อธิบายคำตอบ: c เป็นชนิด celsius_temp ซึ่งเก็บข้อมูลแบบเดียวกัน integer จึงตรงกันกับกรณี 1 คือ โครงสร้างข้อมูลเหมือนกันจึงไม่ต้องดำเนินการใด ๆ เพิ่มเติม

6. `c := celsius_temp(n);`

คำตอบ: compile ผ่านและตอน run-time ไม่ต้องดำเนินการใด ๆ เพิ่มเติม

อธิบายคำตอบ: เช่นเดียวกับคำตอบข้อ 5 เพียงแต่เป็นการ cast ในทางตรงกันข้าม

จบการเฉลย Exercise 4.2

4.3.4 Type Compatibility Rules

ภาษาโปรแกรมส่วนใหญ่ไม่ได้เข้มงวดกับการดำเนินการระหว่างชนิดข้อมูล 2 ชนิดว่าต้องมีความเหมือนกัน แต่จะอนุญาตให้สามารถข้อมูลต่างชนิดสามารถทำงานร่วมกันได้ขึ้นอยู่กับบริบทที่โปรแกรมภาษานั้นยินยอมให้ โดยในการอนุญาตให้ใช้ชนิดข้อมูลต่างจากที่คาดหวังไว้ก็จะเกิดการแปลงข้อมูลแต่จะเป็นแบบ **automatic implicit conversion** คือ แปลงโดยอัตโนมัติในเบื้องหลังโดยไม่ต้องมีการเขียนสั่งไว้ในโค้ด โดยวิธีการนี้เรียกว่า type coercion

เช่นเดียวกันกับ explicit type conversion ใน machine code จาก compiler จะมีการสร้างโค้ดต่าง ๆ สำหรับการแปลงชนิดข้อมูลอย่างการตรวจสอบความถูกต้องของข้อมูลและการเปลี่ยนโครงสร้างข้อมูลระหว่างกันเป็นต้น

4.3.4.1 Type Coercion

ตัวอย่างของ type coercion เช่น ระหว่าง primitive type ในภาษา C ที่สามารถใช้ดำเนินการร่วมกันได้โดยไม่ต้องทำการ cast โดยโปรแกรมเมอร์ แต่จะมีข้อเสียคือในบางครั้งข้อมูลที่แปลงอาจจะไม่ครบถ้วนตามข้อมูลต้นฉบับ เช่น การแปลงค่าชนิดทศนิยมเป็นจำนวนเต็มอาจต้องเสียค่าส่วนทศนิยมไป ซึ่งเป็นส่วนที่ต้องคำนึงถึงด้วยในการเขียนโปรแกรม

พิจารณาได้จากตัวอย่างโค้ดภาษา C ที่สามารถ assign ค่าต่างชนิดกันโดยไม่ต้อง cast ดังต่อไปนี้

```
1: //C
2: short int s;                //16 bits
3: unsigned long int l;        //32 bits
4: char c;                     //8 bits
```

```

5: float f;      //32 bits, IEEE single-precision
6: double d;     //64 bits, IEEE double-precision
7: ...          //something may be interpreted differently,
8:              //or some precision may be lost
9: s = l;
10: l = s;
11: s = c;
12: f = l;
13: d = f;
14: f = d;

```

ตัวอย่างต่อไปนี้เป็นอีกตัวอย่างที่สามารถใช้ตัวแปรชนิด pointer ได้เหมือนตัวแปรชนิด array

```

1: //C
2: //array and pointer can be mixed
3: int n;
4: int *a;
5: int b[10];
6: a = b;
7: n = a[3];

```

ตัวแปรชนิด pointer เป็นตัวแปรที่เก็บที่อยู่อ้างอิงถึงตำแหน่งในหน่วยความจำ ดังนั้นในกรณีนี้จะชี้ไปยังตำแหน่งเริ่มต้นข้อมูลของตัวแปร b ที่เป็นชนิด array ดังนั้นทำให้วิธีการเข้าถึงหน่วยความจำแบบ array จึงสามารถทำได้กับตัวแปรชนิด pointer ที่ชี้ไปยัง array นั้นเอง

4.3.4.2 Universal Reference Type

Compatibility Rule อีกลักษณะหนึ่งที่สามารถพบได้บ่อย คือ universal reference type เป็นลักษณะของชนิดตัวแปรที่สามารถจะรับค่าข้อมูลชนิดใดก็ได้ อย่างเช่นที่ปรากฏในภาษา C ที่เป็นภาษาแบบ value model ที่มีการกำหนดชนิดข้อมูลพิเศษลักษณะนี้ขึ้น คือ pointer ถึงแม้จะเป็น value model แต่มีโครงสร้างข้อมูลที่เก็บค่าที่อยู่ในหน่วยความจำ เพื่อใช้ในการอ้างอิงเพื่อเข้าถึงข้อมูลต่าง ๆ ตัวอย่างการใช้งาน pointer เช่น

```

1: //C
2: int x;  char y;  void *z;
3: z = &x;
4: z = &y;

```

จากโค้ดสมมติว่าในหน่วยความจำมีการเก็บข้อมูลดังนี้

Data Segment: x = 0, y = 0, z = NULL

จากข้อมูลค่าที่อยู่ของ x และ y คือ 1 และ 2 ตามลำดับ ดังนั้นค่าของ pointer ชื่อ z เมื่อการทำงานบรรทัดที่ 3 สำเร็จจะเป็นดังนี้

Data Segment: x = 0, y = 0, z = 1

และเมื่อการทำงานผ่านบรรทัดที่ 4 ผลลัพธ์จะเป็นดังนี้

Data Segment: x = 0, y = 0, z = 2

และเมื่อมีการนำตัวแปร z ไปกระทำการใด ๆ ก็จะเป็นเหมือนการกระทำกับค่าของตัวแปรที่ถูกชี้อยู่

ตัวอย่างถัดไปเป็นตัวอย่างภาษา Java ซึ่งเป็นภาษาแบบ reference model โดยชนิดข้อมูลที่เป็น user define class จะเป็นแบบ reference model ทั้งหมด โดยทุกคลาสจะเป็น subclass ของคลาส Object จึงถือได้ว่าเป็น Universal Reference Type พิจารณาจาก โค้ด ดังต่อไปนี้

```
1: //Java
2: Object a;
3: Cat c = new Cat(); Dog d = new Dog();
4: a = c;
5: a = d;
```

จากโค้ดตัวอย่างโปรแกรมสามารถทำงานได้โดยไม่เกิดข้อผิดพลาด ที่เป็นเช่นนี้เนื่องจากทุกคลาสในภาษา Java จะมีคลาส Object ของทุกคลาส ถึงแม้จะไม่ได้กำกับบอกไว้ในโค้ดก็ตาม เช่นหากประกาศคลาส Cat โดยละส่วนของการ extend ดังนี้

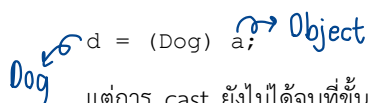
```
public class Cat {};
```

จะมีค่าเท่ากับการประกาศคลาสโดยกำกับ extension ดังนี้

```
public class Cat extends Object {};
```

ดังนั้น ตาม compatibility rule ของภาษา Java ที่ให้ super class สามารถรับการ assign จาก subclass ได้ ทำให้ไม่ว่าจะนำตัวแปรชนิดคลาส Object ไปรับข้อมูลคลาสชนิดใดก็สามารถทำได้โดยไม่ต้องทำการ cast ใด ๆ เลย

จากที่กล่าวมาแล้วว่า Universal Reference Type สามารถรับ assign จะชนิดข้อมูลใดก็ได้ แต่หากต้องการ assign จาก Universal Reference ไปเป็นชนิดที่เฉพาะเจาะจง (particular reference) นั้น ไม่สามารถทำได้ทันทีเนื่องจากความเป็นสากลนั้น เราจึงไม่สามารถแน่ใจได้ว่าขณะที่ทำการ assign นั้น ค่าที่ reference อยู่เป็นชนิดเดียวกับชนิดปลายทางที่ต้องการหรือไม่ เช่น หากต้องการ assign Object ไปเป็น class โดยไม่มีการ cast จะไม่ผ่านการคอมไพล์ ทำให้จำเป็นต้องมีการสั่งให้ cast ก่อนจึงจะผ่านขั้นตอนคอมไพล์ไปยัง run-time เช่น



```
d = (Dog) a;
```

แต่การ cast ยังไม่ได้จบที่ขั้นตอนการคอมไพล์ เพราะความที่ไม่สามารถแน่ใจได้ว่าค่านั้นเป็นชนิดที่ต้องการหรือไม่จึงจำเป็นต้องมีการตรวจสอบความถูกต้องของค่าก่อนว่าเป็นชนิดเดียวกันหรือไม่ และเพื่อให้สามารถตรวจสอบได้สำหรับภาษาเชิงวัตถุแต่ละอ็อบเจกต์ต้องมีการอธิบายชนิดของตัวเองเรียกว่า type tag (type information) โดยในโครงสร้างข้อมูลของแต่ละอ็อบเจกต์จะประกอบด้วย

- Class Info
- Method Table
- Attribute

โดย tag นี้จะถูกสร้างและแปะไว้กับทุกอ็อบเจกต์ที่ถูกสร้างขึ้น ซึ่งจะมีข้อมูลที่สามารถนำมาใช้ในการประมวลผลด้านชนิดข้อมูลอยู่

Exercise 4.3 Universal Reference Type

กำหนดให้คลาส Stack มีเมธอดที่เกี่ยวข้องกับโจทย์ ดังนี้

```
1: class Stack {
2:     Object push(Object item) {...}
3:     Object pop() {...}
4:     ...
5: }
```

พิจารณาโค้ดตัวอย่างภาษา Java ดังต่อไปนี้

```
1: import java.util.* //library containing Stack container class
2: ...
3: Stack my_stack = new Stack();
4: String s = "Hi, Mom";
5: foo f = new foo();
6: ...
7: Object aString = my_stack.push(s);
8: Object aFoo = my_stack.push(f);
9: ...
10: s = my_stack.pop();
```

จากโค้ดข้างต้นนี้ให้อธิบายว่าข้อความสั่งในบรรทัดสุดท้ายจะเกิดปัญหาหรือทำงานได้หรือไม่? แล้วจะสามารถทำให้มันใจว่าการ assign ค่าจาก universal reference จะไม่ทำให้โปรแกรมเกิดการผิดพลาดขึ้นได้อย่างไร?

เฉลย Exercise 4.3

คำตอบแรก บรรทัดสุดท้ายจะเกิด compilation error จากการ assign ค่าจาก universal reference ไปยังที่ชนิดที่เฉพาะเจาะจงซึ่งไม่อนุญาตโดยภาษา เนื่องจาก my_stack.pop() ให้ผลลัพธ์เป็นชนิด Object แต่ s เป็นชนิด String หากต้องการทำเช่นนี้จำเป็นต้องมีการสั่ง cast ดังนั้น เพื่อแก้ error นี้จึงทำการเพิ่ม type casting เป็นดังนี้

```
s = (String) my_stack.pop();
```

ด้วยการแก้ไขนี้จะทำให้โปรแกรมผ่าน compile error ได้ แต่หากพิจารณาจากโค้ดแล้ว ก่อนที่ Stack จะถูก pop มา assign ให้กับตัวแปร s ในบรรทัดที่ 8 ชนิดของอ็อบเจกต์ที่ถูก push ลงใน Stack ไม่ใช่ String ดังนั้น ในตอน run-time เมื่อโปรแกรมทำงาน ถึงบรรทัดสุดท้ายก็จะเกิด error ขึ้นเนื่องจากค่าที่ได้ออกมาไม่ใช่ String ทำให้ไม่สามารถ cast ได้เนื่องจากค่าที่ได้ออกเป็นชนิดคลาส Foo นั่นเอง

หากต้องการเพิ่มความปลอดภัยให้กับโปรแกรมไม่ให้เกิด error ควรต้องมีการตรวจสอบชนิดของข้อมูลก่อนที่จะทำการ cast ว่าเป็นชนิดที่ต้องการหรือไม่ และให้ cast เฉพาะชนิดที่สามารถรับได้ โดยภาษา Java ได้เตรียมคำสั่งเปรียบเทียบชนิดข้อมูลไว้ให้คือ instanceof โดยเมื่อแก้ไขแล้วจะเป็นดังนี้

```
10: Object obj = my_stack.pop();
11: if(obj instanceof String)
12:   s = (String) obj;
```

จากการแก้ไขเมื่อชนิดข้อมูลที่ pop จาก Stack ไม่ใช่ชนิด String ก็จะไม่เกิดการ cast และ assign ทำให้โปรแกรมจะไม่มีโอกาสเกิด error แล้ว

จบการเฉลย Exercise 4.3

4.3.5 Type Inference Rules

ในบางกรณีชนิดผลลัพธ์ของนิพจน์ (expression) หนึ่ง ๆ นั้นจำเป็นต้องมีการอนุมานชนิดข้อมูลจากนิพจน์ย่อย (subexpressions) ว่าจะมีผลลัพธ์เป็นชนิดใดเพื่อนำไปใช้สำหรับกลไกของ type checking สมมติว่า มีข้อความสั่งดังนี้

```
a = b + c
```

ในส่วนของการนิพจน์ $b + c$ จำเป็นต้องอนุมานว่าเป็นชนิดใด ในกรณีนี้จะทำการอนุมานได้อย่างชัดเจน เช่น โดยทั่วไปแล้วชนิดข้อมูลก็จะเป็นชนิดเดียวกันกับชนิดของตัวดำเนินการ เช่น b และ c เป็นชนิด int ก็จะสามารถได้ว่า นิพจน์ $b + c$ เป็นชนิด int หรือหาก b เป็นชนิด int แต่ c เป็นชนิด float ก็จะสามารถได้จาก type coercion ได้ว่า $b + c$ เป็นชนิด float เป็นต้น จึงสามารถจะใช้ในการตรวจสอบเทียบกับตัวแปร a ต่อไปว่าเป็นชนิดเดียวกันหรือไม่

นอกจากนี้การอนุมานยังอนุมานได้จาก ชนิดข้อมูลของ assignment มีค่าเดียวกันกับชนิดของข้อมูลฝั่งซ้ายของเครื่องหมาย หรือชนิดผลลัพธ์จากการเรียกฟังก์ชันอนุมานจากชนิดที่ประกาศไว้กับฟังก์ชัน เป็นต้น

แต่ก็จะมีในบางภาษาโปรแกรมที่ไม่ได้มีความชัดเจนในการอนุมาน ยกตัวอย่างเช่น ภาษา Pascal ดังนี้

```
1: (* Pascal *)
2: type Atype = 0..20;
3:   Btype = 10..20;
4: var a: Atype; b: Btype; c: integer;
5: c = a + b; (* subrange base type (integer), not the type sometype = 10..40 *)
```

จากโค้ด มีนิพจน์บวกระหว่างตัวแปร a และ b ซึ่งเป็นชนิด Atype ที่เป็น subrange มีค่าตั้ง 0 ถึง 20 และ Btype ที่เป็น subrange มีค่าตั้ง 10 ถึง 20 ตามลำดับ หากอนุมานตรงไปตรงมาผลลัพธ์ที่ได้ความจะเป็นชนิดข้อมูลแบบ subrange ชนิดหนึ่งที่มี

ค่าระหว่าง 10 ถึง 40 แต่การจะตรวจสอบเพื่อสร้าง subrange ใหม่มีความยุ่งยาก ดังนั้นภาษา Pascal จึงใช้วิธีการอนุมานไปหาชนิดข้อมูลฐาน (base type) แทน ซึ่งในกรณีนี้คือ integer

สรุปแล้วในการเขียนโปรแกรมควรพึงพิจารณาว่าในภาษาโปรแกรมมีกลไกเกี่ยวกับชนิดข้อมูลเหล่านี้อยู่ เพื่อที่การเขียนการดำเนินการต่าง ๆ จะได้รับผลกระทบเหล่านี้อยู่ซึ่งหากเลือกได้ไม่เหมาะสมก็จะเป็นสาเหตุของข้อผิดพลาดต่าง ๆ ของโปรแกรมได้