

제 6 장 자료형 Data Type

6.1 자료형 개요

자료형의 의미

- 다음 변수 선언을 생각해보자

- `int x;`

- 이 변수 선언에서 자료형의 의미는?

- 그 타입의 변수가 가질 수 있는 값들의 집합

- 따라서 위 선언의 의미는 $x \in \text{Integer}$

"member of" 기호 대신 사용할 수 있는 문장

`x: int;`

`int x;`

- [정의 1]

자료형은 값들의 집합이다(A data type is a set of values).

자료형의 의미

- [정의 2]
 - 값들의 집합(a set of values)
 - 그 값에 적용 가능한 연산들의 모임(a set of operations)
- A data type is a set of values +
a set of operations on those values.
- 데이터 타입의 수학적 의미
 - an algebra (S, Op) where
 - S is a set of values
 - Op is a set of operations on S

기본 자료형

- 사전 정의된 기본 자료형
 - boolean, char, int, float, double, ...
- 사용자 정의 기본 자료형
 - 열거형 : 열거된 값들로 이루어진 타입
- C/C++ 열거형 예
 - `enum` Day {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};
 - `enum` Day today = Monday;
- Ada 열거형 예
 - `type` Day is (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
 - `today : Day;`

부분 타입

Set의 subset처럼

- 부분 타입(subtype)
 - 기반 타입(base type)의 일부분 값들로 정의된 자료형
 - 기반 타입의 모든 연산을 부분 타입에 적용 가능하다.
 - Ada, Pascal의 부분범위형
- Ada 예
 - `subtype Digit is Integer range 0 .. 9;`
 - `subtype Weekday is Day range Monday .. Friday;`
- 부분범위형 없음(C, C++, Java)
 - `byte digit; // -128 .. 127`
 - `if (digit > 9 || digit < 0) throw new DigitException();`

6.2 복합 타입

타입 구성자란 무엇인가?

- 기본 자료형으로부터 더 복잡한 자료형(복합 타입)을 구성하는 방법
 - 배열 타입(Array type)
 - 리스트 타입(List type)
 - 레코드 타입(Record type)
 - 공용체(Union type)
 - 포인터 타입(Pointer type)
- 사용자-정의 타입

구조체(레코드) 타입

- 구조체
 - 여러 개의 (필드) 변수들을 묶어서 구성하는 자료형
 - `struct` in C
 - `record` in Pascal, Ada, ...
- C++의 `class`
 - C의 `struct`를 확장
 - 데이터뿐 아니라 연산(함수)들을 포함하도록 확장한 자료형

구조체(레코드) 타입

- C의 예

```
struct Employee
{
    int age;
    char name[10];
    float salary;
}
```

- 구조체에 새로운 이름

```
typedef struct
{
    int age;
    char name[10];
    float salary;
} Employee
```

Ada의 예

```
type Employee is record
    age: INTEGER;
    name: STRING(1..10)
    salary: FLOAT;
end record;
```

구조체 사용 예

```
void main() {  
    struct Employee person;  
    strcpy(person.name, "kim");  
    person.age = 25;  
    person.salary = 30000.00;  
    printf("person.name = %s\n", person.name); → 여기서 strcpy 사용  
    printf("person.age = %d\n", person.age);  
    printf("person.salary = %f\n", person.salary);  
}
```

배열 타입

- 배열 타입

- 같은 타입의 연속된 변수들로 구성하는 자료형
- 배열 요소(element), 인덱스

- C의 배열

- 배열 크기가 정적 결정

```
int A[10];
```

```
int B[] = {1,2,3,4};
```

→ 컴파일할 때 결정됨



- 배열 A의 원소의 주소

- $A[i]$ 의 주소 = $\text{base} + i * \text{sizeof}(\text{int})$

base는 배열의 시작주소

이차원 배열

- 이차원 배열 선언

자료형 배열이름[M][N];

- 예 int a[3][4];

- 개념적으로 3x4 행렬

a[0][0] a[0][1] a[0][2] a[0][3]
a[1][0] a[1][1] a[1][2] a[1][3]
a[2][0] a[2][1] a[2][2] a[2][3]

- 행 우선(row major order) 배치

- 첫 번째 행부터 배치하고 이어서 두 번째 행을 배치하는 방식
- C/C++ 언어에서 사용됨.
- a[i][j]의 주소 = $\text{base} + (i*4 + j)*\text{sizeof}(\text{int})$

$\text{base} + (i*N + j)*\text{sizeof}(\text{int})$

- 열 우선(column major order) 배치

- 첫 번째 열부터 배치하고 이어서 두 번째 열을 배치하는 방식
- Fortran 언어에서 사용됨.

$\text{base} + (j*M + i)*\text{sizeof}(\text{int})$

Java의 배열

- Java의 배열 변수 선언

자료형[] 변수이름;

자료형 변수이름[];

- 주의: 배열 변수를 선언했다고 배열이 자동적으로 만들어지는 것이 아님!
- 배열 변수는 단지 참조 변수임

- 동적 배열

Java 배열은 배열 크기가 실행 시간에 결정되는 일종의 객체

배열 크기가 배열의 일부

`int[] x;` → 선언 (참조변수)

`x = new int[10];`

↓
생성



리스트

- 리스트(list)

- 항목들의 모음으로 다수의 항목을 집합적으로 처리하는데 유용하다.
- 파이썬의 대표적인 자료구조이며 []를 이용하여 정의한다.
- 파이썬에서는 배열 대신 리스트를 사용한다.

- Python 예

```
>>> Week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

```
>>> Week[0]
```

```
'Monday'
```

```
>>> primes = [2, 3, 5, 7, 11, 13, 17]
```

```
>>> primes[2]
```

```
5
```

리스트

+ 크기 지정 필요없음

- 리스트는 배열과 달리 원소가 같은 타입일 필요가 없다.

```
>>> b = [1, 'two', 3, 'four', 5, 'six']
```

```
>>> b[1]
```

```
'two'
```

- 리스트의 원소로 리스트도 가능하다.

```
>>> a = [1, 2, 3, ['a', 'b', 'c']]
```

```
>>> a[0]
```

```
1
```

```
>>> a[3]
```

```
['a', 'b', 'c']
```


부분 리스트(slicing)

- 하나의 리스트를 인덱스를 사용하여 부분 리스트로 분리

- `list[start : end]`

- 예

```
>>> months = ['jan', 'feb', 'mar', 'apr', 'may']
```

```
>>> months[1 : 4]
```

```
['feb', 'mar', 'apr']
```

```
>>> months[1 : 2]
```

```
['feb']
```

```
>>> months[1 : 1]
```

```
[]
```

```
>>> months[: 2]
```

```
['jan', 'feb']
```

```
>>> months[2 :]
```

```
['mar', 'apr', 'may']
```

Java의 ArrayList

- 일반적인 배열
 - 배열의 크기가 한번 결정되면 그 크기가 고정된다.
- 리스트 혹은 동적 배열
 - 배열의 크기가 동적으로 변하는 배열
 - Java의 ArrayList가 대표적인 예.

- 예

- ArrayList는 다양한 타입의 객체를 저장할 수 있다.

```
ArrayList list = new ArrayList();
```

```
list.add("ONE");
```

```
list.add(2);
```

```
list.add(new Float(3.0));
```

```
System.out.println(list);
```

```
//[One, 2, 3.0]
```

→ 어떻게 가능할까?

ArrayList 안에는 Integer(2), Float(3.0) 등으로 저장됨

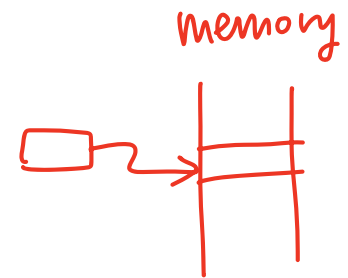
↳ autoboxing : 자동적으로 new Integer(2); 와 같이 실행됨

포인터 타입

- 지금까지 타입들은 위치(주소)를 값으로 사용할 수 없었다!
- 포인터 타입(pointer type)
 - 메모리의 위치(주소)를 값으로 사용하는 자료형이다.

- 포인터 변수 관련 구문

- | | |
|---------------|------------------|
| (1) $T^*p;$ | 포인터 변수 선언 |
| (2) $p = E;$ | 포인터 변수에 대입 |
| (3) $*p = E;$ | 포인터가 참조하는 변수에 대입 |



- 포인터 관련 연산

- | | |
|----------------------------|-------------------------|
| (1) <code>malloc(n)</code> | 크기 n의 메모리 할당 및 시작 주소 반환 |
| (2) <code>&x</code> | 변수 x의 포인터(주소) |
| (3) <code>*p</code> | 포인터 변수 p에 저장된 포인터 주소참조 |

예제 1

```
1 { int *p = malloc(4);
2   int y = 10;
3   *p = y;
4   *p = *p / 2;
5   printf("%d : %d\n", p, *p);
6   p = &y;
7   printf("%d : %d\n", p, *p);
8 }
```

* linked list 만들때 많이 사용!



● 실행결과

27041808 : 5

944757364 : 10

재귀 타입(Recursive Type)

- 타입 정의에 자신의 이름을 사용하는 경우

```
struct CharList
{
    char data;
    struct CharList next; /* c에서 위법 */
};
```

- 이 정의의 문제는 무엇인가?

```
struct CharListNode
{
    char data;
    struct CharListNode* next; /* c에서 적법 */
};
typedef struct CharListNode* CharList;
```

6.3 사례 연구

사례연구: C 언어 자료형

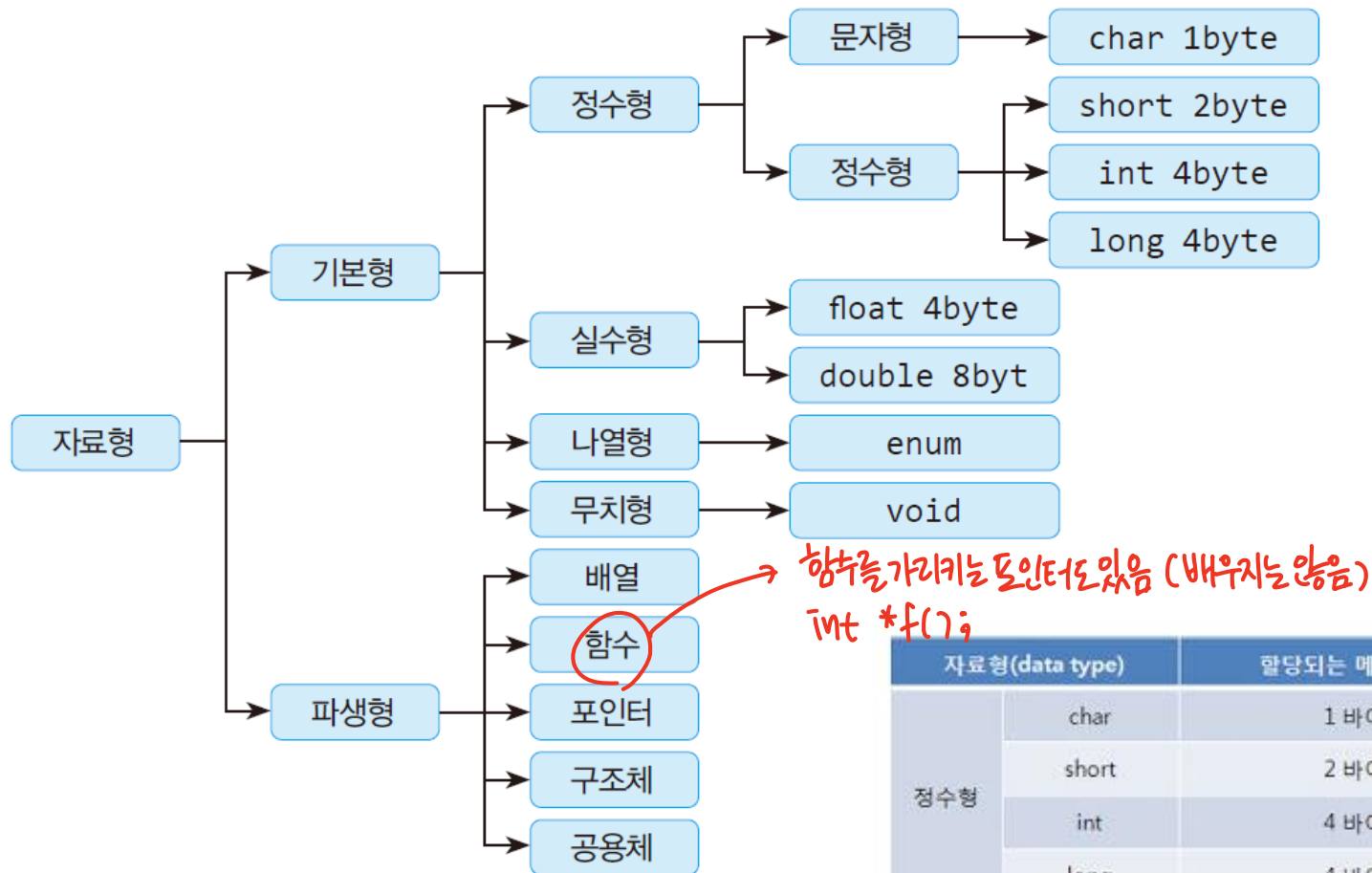


그림 6.1 C 언어 자료형

자료형(data type)		할당되는 메모리 크기	표현 가능한 데이터의 범위
정수형	char	1 바이트	-128 ~ +127
	short	2 바이트	-32768 ~ + 32767
	int	4 바이트	-2147483648 ~ + 2147483647
	long	4 바이트	-2147483648 ~ + 2147483647
실수형	float	4 바이트	$3.4 \times 10^{-37} \sim 3.4 \times 10^{+38}$
	double	8 바이트	$1.7 \times 10^{-307} \sim 1.7 \times 10^{+308}$
	long double	8 바이트 혹은 그 이상	차이를 많이 보임

사례연구: Java 자료형

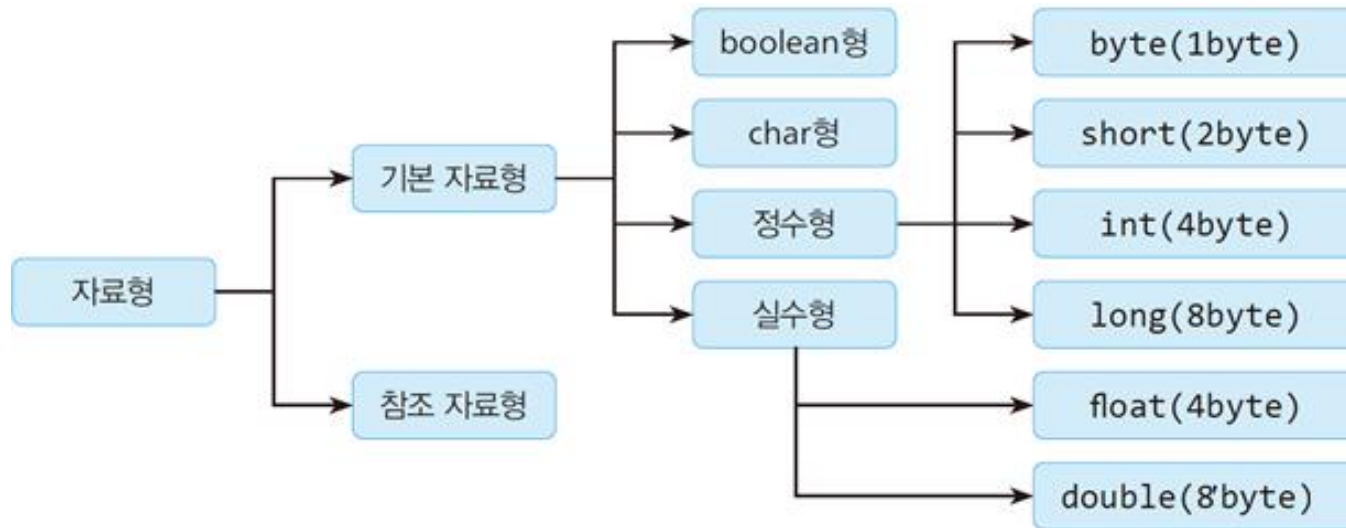


그림 6.2 Java 언어 자료형

- 참조 자료형
 - 클래스
 - 인터페이스
 - 배열

자료형	기억장소	최소값	최대값
byte	8 비트	-128	127
short	16 비트	-32768	32767
int	32 비트	-2,147,483,648	2,147,483,647
long	64 비트	약 -9×10^{18}	약 9×10^{18}
float	32 비트	약 -3.4×10^{38} (유효숫자 7자리)	약 3.4×10^{38} (유효숫자 7자리)
double	64 비트	약 -1.7×10^{308} (유효숫자 15자리)	약 1.7×10^{308} (유효숫자 15자리)

6.4 타입 변환

자동 형변환

이항연산에서 두 피연산자의 자료형이 다른 경우

- 자동 형변환(automatic type conversion)
 - 자동으로 형을 변환하는 묵시적 형변환(implicit type conversion)
- 확장 변환(widening conversion)
 - 표현 범위가 더 넓은 쪽으로 변환하는 상향 변환(promotion)
 - Java에서 자동 형변환은 거의 대부분 확장 변환이다.
- 예
 - `double y = 99; // 99.0`
- Java 확장 변환 순서
 - `byte(1) < short(2) < int(4) < long(8)`
 - `float(4) < double(8)`

축소 변환

- 축소 변환(narrowing conversion)
 - 확장 변환의 반대로 표현 범위가 더 작은 자료형으로 변환
 - 예컨대 int를 short로, double을 float로 형변환
- 대입 변환(assignment conversion)
 - 대입문의 경우에도 확장 변환인 경우에만 자동 수행
 - 축소 변환인 경우에는 자동으로 수행되지 않는다. → 정보 손실 우려
- 예외적으로 int 상수에 대해서만 자동 축소 변환됨
 - int 상수를 byte, short, char에 대입할 때 정보 손실이 없으면 자동 변환
 - `byte b = 123;` // 축소 변환
 - `short s = 456;` // 축소 변환
 - `int i = s;` // 확장 변환
 - `byte b2 = 456;` // 오류: byte 범위 밖
 - `short s2 = i;` // 오류

형변환 연산자

- 형변환 연산자

- (자료형) 수식
- 수식에 의해 계산된 값을 명시된 자료형의 값으로 변환함.
- 정보 손실이 발생할 수 있음.
- 타입 캐스팅 연산자(type casting operator) 또는 캐스트(cast)라고 함.

- 예

- `byte b2 = (byte) 456;` // 캐스팅: 정보 손실
- `short s2 = (short) b2;` // 캐스팅: 정보 손실 없음
- `int i2 = (int) 3.14;` // 캐스팅: 정보 손실

6.5 타입과 언어 분류

정적 타입 언어

- 정적 타입 언어(statically typed language)
 - 변수의 타입이 컴파일 시간에 결정되어 고정되는 언어로
 - 보통 컴파일 시간에 타입 검사를 한다.
 - Java, C, C++, FORTRAN, Pascal, Scala



● Java 예

- String 타입으로 선언된 변수 name은 한번 타입이 선언되면 그 타입이 변할 수 없다.

String name;

✓ name = "john";

✗ name = 34;

변수는 선언된 타입을 갖는다.

값은 타입을 갖는다

변수는 타입이 변경될 수 없다.

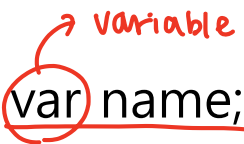
동적 타입 언어

- 동적 타입 언어

- 변수의 타입이 저장되는 값에 따라 실행 중에 바뀔 수 있는 언어로
- 보통 실행 시간에 타입 검사를 한다.
- Perl, Python, Scheme, JavaScript 등

- JavaScript 예

- 변수 name은 선언된 타입이 없고 대입된 값에 따라 변수의 타입이 변경된다.


var name;
✓ name = "john";
✓ name = 34;

변수는 선언된 타입이 없다.
값은 타입을 갖는다
변수는 타입이 변경될 수 있다.

동적 타입 언어: 사례

- Python

- 변수의 타입이 저장되는 값에 따라 실행 중에 바뀔 수 있으며
- 변수, 매개변수, 필드에 대한 타입 선언이 없음.
- 각 값은 타입을 나타내는 타입 태그를 갖는다.

- 예

```
>>> a = 3
```

```
>>> type(a)
```

```
<class 'int'>
```

```
>>> a = 3.0
```

```
>>> type(a)
```

```
<class 'float'>
```

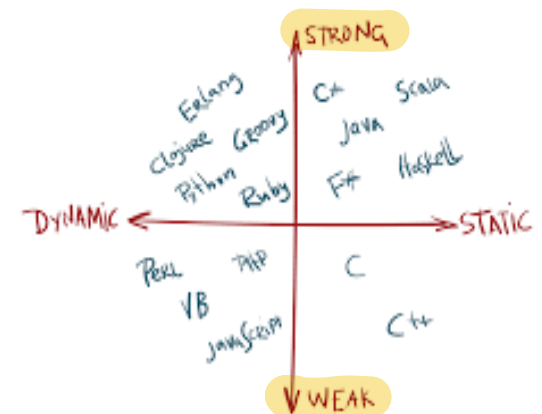
```
>>> a = 'this is a string'
```

```
>>> type(a)
```

```
<class 'str'>
```


강한 타입 언어

- 타입 규칙(typing rule)
 - 언어를 설계할 때 프로그램 구성요소의 타입 사용 규칙도 정한다.
 - 타입 규칙의 엄격성에 따라 강한/약한 타입 언어로 분류한다.
- 강한 타입 언어(strongly typed language)
 - 엄격한 타입 규칙을 적용하여 (모든) 타입 오류를 찾아 낼 수 있는 언어
 - Java, C#, Python
- 약한 타입 언어(weakly typed language)
 - 그렇지 않은 느슨한 타입 규칙을 적용한 언어
 - C/C++, PHP, Perl, JavaScript



사례 1: 강한 정적 타입 언어 Java

불변함

but 안정성↑

- Java

- 정적 타입 언어: 변수의 타입이 컴파일 시간에 결정되어 변할 수 없음
- 강한 타입: 강한 타입 규칙을 적용한다.

- 예

```
public class test {  
    public static void main() {  
        int x=0; float y=3.14f; char z='A';  
        x = y;    // 오류 축소변환  
        z = x;    // 오류 축소변환  
        y = z;  
    }  
}
```

↓
데이터유실위험
(자동적용 X, 에러발생시킴)

```
$ javac test.java
```

test.java:6: error: possible loss of precision

```
    x = y;
```

^

required: int

found: float

test.java:7: error: possible loss of precision

```
    z = x;
```

^

required: char

found: int

2 errors

사례 2: 약한 정적 타입 언어 C

편함

but! 에러발생가능성 ↑

- C 언어

- 정적 타입 언어: 변수의 타입이 컴파일 시간에 결정되어 변할 수 없음
- 약한 타입(weak typing): 약한 타입 규칙을 적용한다.

- 예

```
int main() {  
    int x=0; float y=3.14; char z='A';  
    x = y;  
    printf("%d\\n", x);  
    y = z;  
    printf("%f\\n", y);  
    z = x;  
    printf("%c\\n", z);  
}
```

→ 자동 축소변환

→ 자동 확장변환

→ 자동 축소변환

```
$ gcc test.c
```

```
$ a.out
```

```
3
```

```
65.000000
```

□ ← white space 출력됨

사례 3: 강한 동적 타입 언어 Python

- Python

- 동적 타입 언어:
변수의 타입이 저장되는 값에 따라 실행 중에 바뀔 수 있다.
- 강한 타입(Strong typing):
강한 타입 규칙을 적용한다.

- 예

```
>>> x = 1                # x bound to an integer.
```

```
>>> y = "23"            # y bound to a string .
```

```
>>> print(x+y) → error!
```

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module>

print(x+y)

TypeError: unsupported operand type(s) for +: 'int' and 'str'

사례 4: 약한 동적 타입 언어 JavaScript

↓
web 프로그래밍은 주로 String을 다룸
∴ String의 + 연산 필요

- JavaScript

- 동적 타입 언어:

- 변수의 타입이 저장되는 값에 따라 실행 중에 바뀔 수 있다.

- 약한 타입:

- 약한 타입 규칙을 적용한다.

- 예

```
var x = 1;
```

```
var y = "23";
```

```
var z = x + y; // 123
```

int를 String으로 자동형변환
→ String의 + 연산 : 집합

실습문제 #4

1. 다음과 같이 언어 S에 **배열 선언 및 사용 기능**을 추가하고 이를 **인터프리터에 구현**하시오.

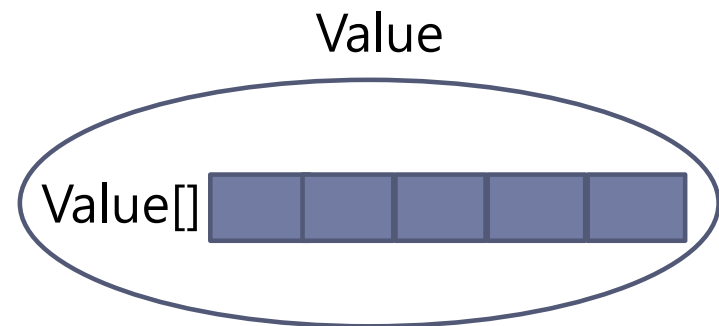
- Parser.java

```
<decl> → ... | <type> id;  
                | <type> id[n];           // 배열 선언  
  
<stmt> → ... | id = <expr>;  
                | id[<expr>] = <expr>;    // 배열 요소에 대입  
  
<factor> → ... | id  
                | id[<expr>]              // 배열 요소 사용  
  
<type> → int | bool | string
```

실습문제 #4

- AST.java(제공함)

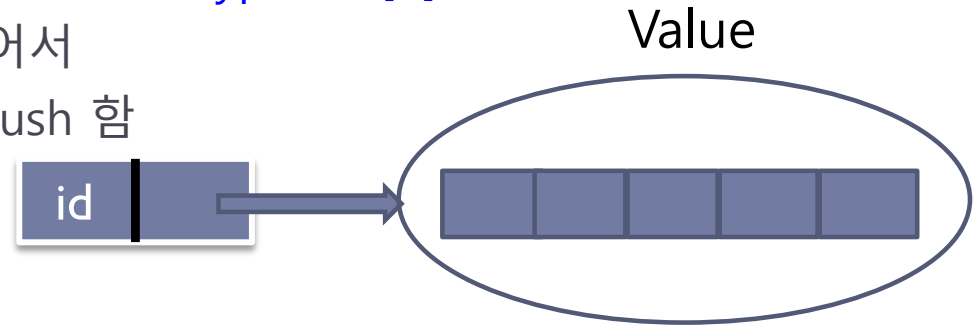
- class Decl extends Command { ... } // <type> id[n];
// 배열 선언(배열 이름, 크기) 추가
- class Array extends Expr { ... } // id[<expr>]
// 배열 요소 접근 즉 배열[인덱스]에 대한 표현
- class Assignment extends Stmt { ... } // id[<expr>] = <expr>;
// 배열 요소에 대입을 추가
- class Value extends Expr { ... }
// 값의 하나로 배열 Value[]이 추가됨
// arrValue()를 사용하여 배열 추출!



실습문제 #4

- Sint.java

- State allocate (Decl ds, State state) { // <type> id[n];
 // 배열 선언에 대해서 배열을 만들어서
 // (id, new Value(배열))을 state에 push 함
}



- Value V(Expr e, State state) {
 if (e instanceof Array) { // id[<expr>]
 // 배열 이름이 나타내는 배열에서
 // 배열 요소 id[<expr>]의 값 리턴
 }
}
- State Eval(Assignment a, State state) { // id[<expr>] = <expr>;
 // 배열 이름이 나타내는 배열의
 // 배열 요소에 대입
}