

6. 데이터 타입

데이터 타입

✎ 핵심 개념

데이터 타입은 값들의 모임과 이러한 값들에 대한 미리 정의된 연산들의 집합으로, 프로그래밍 언어의 기본 구성 요소입니다.

1. 서론

- **데이터 타입이란?**
 - 값들의 모임과 이러한 값들에 대한 미리 정의된 연산들의 집합으로 정의된다
 - 모든 프로그래밍 언어는 데이터 타입을 제공
 - 언어에서 제공하는 데이터 타입이 실제 문제를 얼마나 잘 표현할 수 있나?
- **데이터 타입의 용도**
 - 오류 감지: 타입 검사
 - 프로그램 모듈화 지원: 프로그램을 구성하는 단위로 (클래스 또는 패키지)
 - 문서화

2. 기본 데이터 타입

🔗 기본 데이터 타입의 특징

- 하드웨어의 직접적인 반영
 - 다른 데이터 타입으로 정의되지 않는 기본적인 타입
 - 대부분의 프로그래밍 언어에서 공통적으로 제공
-
- **정의**
 - 다른 데이터 타입을 이용해서 정의되지 않는 타입
 - 하드웨어의 반영
 - **기본 데이터 타입 종류**
 - 수치 타입: 정수, 부동 소수점, 십진수, 복소수
 - 불리언 타입
 - 문자 타입

2.1 정수

- 하드웨어의 정확한 반영
- 다양한 크기의 정수 지원
 - 예: Java의 부호 정수 (byte, short, int, long)

2.2 부동 소수점

⚠ 주의사항

- 무한 소수는 정확하게 표현할 수 없음
 - 부동 소수점 연산에서 정확성 문제 발생 가능
 - 금융 계산에는 부동 소수점 사용을 피해야 함
-
- 실수를 근사 값으로 모델링
 - 특징:
 - π , e 와 같은 무한 소수 → 유한한 메모리로 표현할 수 없음
 - 일부 유한 소수는 유한 개의 이진수로 표현 불가능 (예: 0.1)
 - 표현 방식:

- 소수점 이하 부분과 지수 부분으로 표현
- IEEE 부동-소수점 표준 754 형식
- 2가지 실수 타입: float/double

2.3 복소수

- 복소수는 부동 소수점 수의 쌍(실수부, 허수부)으로 표현
 - 예: $a+bi$, $i = \sqrt{-1}$
 - 리터럴 형식: $(7 + 3j)$ in Python
 - 복소수 산술 연산 지원
- 지원 언어: Fortran, Python, C99

2.4 십진수

🔗 십진수의 특징

- 정확한 십진수 표현이 가능
- 금융 계산에 적합
- 메모리 사용량이 많음
- 십진수 숫자를 위한 이진수 코드를 이용하여 문자열 스트링과 유사하게 저장
- BCD(binary coded decimal)라고도 함
- 특징:
 - 십진수 한자리 수를 표현하기 위해 4비트 필요
 - 소수 3자리를 표현하기 위해 12비트 필요
- 장점: 정확성(십진수 값을 정확하게 표현)
- 단점: 제한된 범위, 메모리 낭비
- 지원 언어: COBOL, C#, Basic

2.5 불리안

- 값들의 범위는 단지 참, 거짓의 2가지
- 구현: 흔히 бай트로 구현
- 장점: 판독성 향상
- 지원 언어: C99, C++, Java, C#, VB, Python
- 예외: C89 (0은 거짓, 0이 아니면 참)

2.6 문자

📄 문자 인코딩의 발전

- ASCII → Unicode로의 발전
- Unicode는 전 세계 문자를 지원
- UTF-8이 가장 널리 사용되는 인코딩
- 문자 데이터는 수치 코딩으로 저장
- 코딩 기법:
 - ASCII (8bits): American Standard Code for Information Interchange
 - 16-bit Unicode (USC-2)
 - 1991년 Unicode 컨소시엄에서 발표
 - 세계 자연 언어 문자 대부분 포함
 - Java에서 처음으로 도입
 - C#, JavaScript, Python 지원
 - 32-bit Unicode (USC-4, UTF-32)

3. 문자 스트링 타입

🔗 스트링 타입의 중요성

- 대부분의 프로그램에서 가장 많이 사용되는 데이터 타입
- 언어마다 구현 방식이 다름
- 메모리 관리와 성능에 큰 영향을 미침

- 문자 스트링 타입(character string type)은 값이 일련의 문자들로 구성
- 설계 고려사항:
 - 기본 타입인가? 아니면 문자 배열인가?
 - 스트링의 길이가 정적인가? 아니면 동적인가?

3.1 문자 스트링 타입 연산

- 전형적인 연산들:
 - 배정
 - 비교(=, >, 등)
 - 집합
 - 부분 스트링 참조(substring reference)
 - 패턴 매칭(pattern matching)

3.2 언어별 구현

C, C++

⚠ C/C++ 스트링 주의사항

- 버퍼 오버플로우 위험
- 메모리 관리 필요
- 널 문자 처리 필수

- 기본 타입이 아니고, char 배열로 제공
- 스트링은 null 문자('\0')로 끝남
- 스트링 연산을 표준 라이브러리 string.h로 제공
 - strcpy, strcat, strcmp, strlen
- C++은 string 클래스 제공

예제:

```
char src[] = "Hello World!";
char dest[5];
strcpy(dest, src);
strncpy(dest, src, sizeof(dest)-1);
dest[4] = '\0'; // 널문자 처리

std::string s = "Hello";
s += " world";
std::string original = "Hello, world!";
std::string copy = original;
```

Java

🔗 Java String 특징

- String은 불변(immutable)
- StringBuffer는 가변(mutable)
- 문자열 연산 시 성능 고려 필요

- String 클래스: 불변
- StringBuffer 클래스

예제:

```
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World"); // 문자열이 수정됨 (새 문자열 객체가 아님)
```

Python

- 기본 타입 스트링 지원
- Java의 String 클래스처럼 값은 불변
- 다양한 스트링 연산 제공(탐색, 대체, 부분 스트링 참조, 접합 등)

Perl, JavaScript, PHP

- 정규식 기반 패턴 매칭 연산 제공
 - C++, Java, C#, Python에서 클래스 라이브러리로 지원
- 정규식 예시:
 - `/[A-Za-z][A-Za-z\d]+/`
 - `/\d+\.\d*|\.\d+/`

3.3 스트링 길이 선택 사항

🔗 스트링 길이 선택 시 고려사항

- 정적 길이: 메모리 효율적, 제한된 유연성
- 동적 길이: 유연성 높음, 메모리 관리 필요
- 제한된 동적 길이: 중간적인 접근

1. 정적 길이 스트링(static length string)

- 스트링 생성시 그 길이가 설정되고 고정
- 예: Python, Java

2. 제한된 동적 길이 스트링(limited dynamic length string)

- 스트링 선언 시 고정된 최대 길이까지의 가변적인 길이를 갖는 것을 허용
- 예: C, C++의 C 스타일 스트링

3. 동적 길이(dynamic length string)

- 최대 길이 제한 없이 가변 길이를 갖는 것을 허용
- 최대의 유연성, 동적 할당/회수 부담
- 예: Perl, JavaScript, C++

3.4 스트링 타입 평가

- 작성력 향상
 - 스트링이 문자 배열로 지원되고, strcpy를 위한 함수가 제공되지 않은 경우 고려
- 기본 타입으로 스트링(정적 길이) 제공 필요
 - 동적 길이 스트링은 유연하지만 비용 부담
- 단순 패턴 매칭이나 접합과 같은 연산은 필수적

4. 열거 타입

🔗 열거 타입의 장점

- 코드의 가독성 향상
- 타입 안전성 제공
- 유지보수 용이성

- 열거 타입(enumeration type)은 모든 가능한 값들이 그 정의에서 제공되는 타입
- 값은 열거 상수(enumeration constants)라 불리는 이름 상수로 표현

4.1 예제

```
// in C
enum days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

```
enum colors {red, blue, green, yellow, black}; // 디폴트 내부 값은 0, 1, ...

// 정수 문맥에서 int로 강제변환
int main() {
    enum colors myColor = blue, yourColor = red;
    // ...
    myColor = yourColor + 1; // 적법한가?
    // ...
    myColor = 4; // 적법한가?
}
```

4.2 지원 언어

- 지원: C, C++, Java, C#, Python3.4
- 미지원: Perl, JavaScript

4.3 평가

⚠ 열거 타입 사용 시 주의사항

- 산술 연산의 의미 있는 사용 여부 검토
 - 범위를 벗어난 값 할당 가능성 확인
 - 타입 변환 시 주의 필요
- **장점:**
 - 판독성 향상: 이름 상수가 코딩된 값보다 쉽게 인식
 - 신뢰성 향상
 - **고려사항:**
 - 열거 타입에 대한 산술 연산이 의미가 있는가?
 - 열거 타입 변수에 범위를 벗어난 값을 할당 가능한가?
 - C, C++, Java, C#의 열거 타입 비교