데이터 타입 1

Programming Languages

2018 1학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

목차

- ❑ 데이터 타입의 개념 및 목적
- ❑ 기본 데이터 타입
- ❑ 배열

데이터 타입(Data Types)

❑ 데이터 타입이란?

- 데이터 값들의 모임과 그들 값들에 대한 미리 정의된 연산들의 집합을 정의
- 예)
  - ✓ FORTRAN의 INTEGER, REAL, Arrays 등
  - ✓ C의 int, float, char, 포인터 …
  - ✓ Ada의 사용자 정의 타입

❑ 데이터 타입의 종류

- 기본 데이터 타입
  - ✓ 정수, 실수, 불리안(참거짓), 문자
- 유도된 데이터 타입
  - ✓ 순서 타입(enum), 배열(array), 레코드(record), 유니온(union), 포인터, …

데이터 타입(Data Types)

❑ 데이터 타입의 목적

- 실제 세계의 대상들을 표현
  - ✓ F, 홍길동, <김구, 821-2114>
  - ✓ 성적[1], 성적[2],… 성적[학번]

- 공간을 효과적으로 쓸 수 있도록
  - ✓ 컴파일러가 적절한 크기의 메모리를 할당하게 함

- 타입 검사로 오류를 사전에 알려줌
  - ✓ int x; -> x에 4byte 할당, char y; -> y에 1byte할당
  - ✓ int x; Boolean t; -> x + t // 오류

기본 데이터 타입(primitive data type)

❑ 기본 데이터 타입

- 다른 데이터 타입을 사용하지 않고도, 스스로 정의되는 데이터 타입
- 예)
  - ✓ 수 : 정수(integer), 실수(floating point), 십진수(decimal)
  - ✓ 기타 : 참거짓(boolean), 문자(character)

기본 데이터 타입(primitive data type)

❑ 정수 (integer)

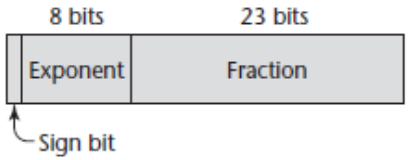
- 컴퓨터 하드웨어에서 직접 지원
- 한 언어 내에서도 서로 다른 여러 가지 정수 타입을 가질 수 있음
- 음의 정수 : 대부분의 컴퓨터에서 2의 보수법 사용하여 저장
- 예)
  - ✓JAVA는 4가지 크기의 부호 정수(signed integer) : byte, short, int, long
  - ✓C++, C# 등 : 부호 정수(signed integer), 비부호 정수(unsigned integer)

기본 데이터 타입(primitive data type)

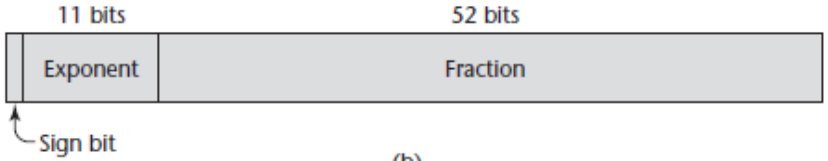
❑ 실수 (floating-point)

- 수학적 실수는 오직 근사치로만 표현 가능
  - ✓ π , e 등은 유한하게 표현 불가
  - ✓컴퓨터 2진수 표현 : 0.1도 표현이 불가
    - 예) 십진수의 0.1은 이진수로 0.0001100110011...

- 표현 방법 –> 부동 소수점(floating point)
  - ✓ 대부분의 언어 두 가지 부동 소수점 타입 : float(4byte), double(8byte)



(a)



(b)

IEEE floating point format : (a) 단정도 (b) 배정도

기본 데이터 타입(primitive data type)

❑ 복소수(complex)

- 어떤 프로그래밍 언어는 복소수 데이터 타입을 지원
- 복소수 값은 순서화된 부동 소수점 수의 값들의 쌍으로 표현
- 예) Fortran과 Python
  - ✓ Python : (7 + 3j)

기본 데이터 타입(primitive data type)

❑ 십진수(decimal)

- 근사치가 아니라 정확한 표현이 꼭 필요할 때 사용되는 타입
- 고정된 개수의 십진수를 저장, 소수점 위치도 고정
  - ✓ 장점 : 정확성
  - ✓ 단점 : 메모리 낭비, 표현 가능한 수의 제한
    - 한 개의 십진수 숫자를 표현하기 위해 4개의 비트 필요, 6개의 숫자로 구성된 십진수를 저장하기 위해서는 24비트 필요.
    - 하지만, 같은 수를 이진수로 저장하는데에는 20비트만 필요

- 십진수 타입은 십진수 숫자를 위한 이진수 코드를 사용하여 문자 스트링과 흡사하게 저장 -> BCD (binary coded decimal)

기본 데이터 타입(primitive data type)

❑ 불리안 타입(boolean)

- 참거짓 타입이라고도 함
- 조건 검사 등 프로그램에서 많이 사용됨
- 주로 1byte로 표현됨
  - ✓0이면 거짓, 0이 아니면 참
- 별도의 데이터 타입을 가지거나(대부분의 언어), 정수 타입을 빌려서 사용(C, C++)

기본 데이터 타입(primitive data type)

❑ 문자 타입 (char)

- 문자 데이터는 수치 코딩으로 컴퓨터에 저장
- ASCII(8bit)
  - ✓0~127 범위의 값을 사용하여 128개의 다른 문자에 대한 코드 제공
  - ✓ 영어 1문자, 특수 문자등 표현
  - ✓ ‘a’, ‘c’, ‘%’, …
- Unicode(16bit)
  - ✓ 16bit로 영문자 뿐아니라 한글, 중국어, 일본어 및 더 많은 특수 문자를 포함
  - ✓Unicode의 처음 128개 문자는 ASCII와 동일
  - ✓Java는 Unicode 문자 집합을 사용하는 처음으로 널리 사용되는 언어
  - ✓이후, JavaScript, Python, Perl, C#, F#에도 채용

배열 (Arrays)

- 동일한 데이터 타입을 가지는 값들의 모임

- 각 원소는 첨자에 의해 인식됨
  - 이때 첨자는 첫번째 원소를 기준으로 한 상대적인 위치를 의미
  - 예) int arr[5]; arr[**0**] = 1; arr[**1**] = 2;

배열의 인덱스(Index)

❑ 인덱스

- 인덱스(indexing) : 배열이름과 첨자를 가지고 특정 원소에 대응시키는 것
  - ✓ FORTRAN, PL/I, Ada는 ( ), 나머지는 [ ]를 사용

- 첨자의 데이터 타입
  - ✓ 정수(FORTRAN, C, Java)
  - ✓ integer, Boolean, char, enum (Pascal, Ada)

- 첨자 개수 (다차원 배열)
  - ✓ 3개로 한정(FORTRAN I), 7개로 한정(FORTRAN 77)
  - ✓ 그러나, 그외에는 제한 없음

배열의 유형

❑ 정적 배열(static array)

- 첨자의 최대 범위가 정적으로 바인딩
- 기억공간 할당도 정적으로 바인딩
- 장점 : 실행 효율성
- 단점 : 배열에 대한 기억공간이 프로그램 실행 전체에 고정
- 예) static 명세자를 포함하는 C와 C++ 함수에 선언된 배열

❑ 고정 스택 동적 배열(fixed stack-dynamic array)

- 첨자의 최대 범위는 정적으로 바인딩
- 배열의 메모리 할당은 실행중에 이루어짐 (스택)
- 장점 : 기억 공간의 효율성
- 예) C와 C++ 함수에 선언된 배열(static 명세자를 포함X)

배열의 유형

❑ 스택 동적 배열(stack-dynamic array)

- 첨자 범위, 메모리 공간은 수행 중 동적으로 결정됨 (스택)
- 그러나 시작할때 한번 결정이 되면 변수 존속기간 동안 고정
- 장점 : 융통성 (배열이 사용되기 직전에 첨자 범위와 배열 크기를 알 수 있을때 유용)
- 예) Ada

```
Get(List_Len);
declare
  List : array (1..List_Len) of Integer;
  begin
  ...
  end;
```

배열의 유형

❑ 고정 힙 동적 배열(fixed heap-dynamic array)

- 첨자 범위와 기억공간의 바인딩이 기억공간이 할당된 후에 고정된다는 점에서 스택 동적 배열과 유사
- 차이점
  - ✓첨자 범위와 기억공간의 바인딩 : 실행 중에 사용자 프로그램이 요청할 때
  - ✓기억공간은 스택이 아닌 힙
- 예) C의 malloc/free, C++의 new/delete로 조작되는 배열

배열의 유형

❑ 힙 동적 배열(heap-dynamic array)

- 첨자 범위와 기억공간이 수행중에 동적으로 결정됨
- 수행하다가 변할 수 있음
- 예) C#, Java의 ArrayList
  - ✓ ArrayList int List = new ArrayList(); //빈 배열을 생성
  - ✓ intArray.Add(nextone);

- 예) 패턴매칭 (Perl, JavaScript, PHP)
  - ✓ @list = {1, 2, 4, 7, 10};
  - ✓ push(@list, 13,17); // @list 배열에 두 원소 추가

배열의 초기화

❑ 배열 초기화

- 어떤 언어는 배열의 기억공간이 할당되는 시점에 초기화하는 수단을 제공
- 예) C

```
int list [] = {4, 5, 7, 83};
```

- 예) Ada

```
List : array (1..5) of Integer := (1, 3, 5, 7, 9);
Bunch : array (1..5) of Integer := (1 => 17, 3 => 34,
                                    others => 0);
```

- 예) FORTRAN

```
Integer, Dimension (3) :: List = (/0, 5, 5/)
```

❑ 배열의 연산

- C언어 기반 언어들 Java, C++, C#의 메소드를 통하는 경우를 제외하면 배열 연산을 제공하지 않음

- 예) Ada
  - ✓ 배열 지정(assignment)(:=), 이어붙이기(&), 동등(equality)과 비동등 (inequality) 비교

- 예) FORTRAN
  - ✓ 행렬 벡터 연산 라이브러리가 풍부함

- 예) APL
  - ✓ 이제까지의 언어중 가장 강력한 배열 처리 언어
  - ✓ 배열에 대한 4칙연산, 벡터의 원소 역수, 행렬의 행/열의 역순, 전치행렬, 역행렬
  - ✓ A × B (행렬 곱셈), A +.× B (내적의 합)

정방형/비정방형 배열

❑ 정방형배열 (rectangle array)

- 모든 행이 동일 개수의 원소들을 포함, 모든 열이 동일 개수의 원소들을 포함
  - ✓ myArray [3, 7] //in FORTRAN, Ada, C#

❑ 비정방형배열 (jagged array)

- 행, 열의 크기가 동일하지 않은 다차원 배열
  - ✓ 행렬이 3개의 행을 포함하고, 각각 5개의 원소를 갖고, 7개의 원소를 갖고, 12개의 원소를 갖는 행으로 구성될 수 있음
  - ✓ 비정방형배열은 다차원 배열이 배열들로 구성된 배열일 때 가능해짐
  - ✓ 예) myArray [3][7] // array of arrays in C, C++, Java
  - ✓ C#, F#은 비정방형배열도 지원

배열의 슬라이스(Slice)

❑ 슬라이스

- 배열의 슬라이스는 배열의 어떤 부분 구조
- 예) FORTRAN 95
  - ✓ Integer vector(1:10), Mat(1:3, 1:3), Cube(1:3, 1:3, 1:4)

…

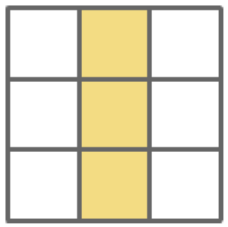
Vector(3:6)

Mat(1:3, 2)

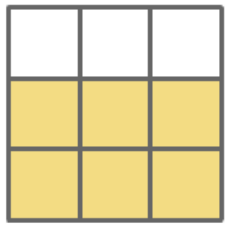
Mat(3, 1:3)

Cube(1:3, 1:3, 2)

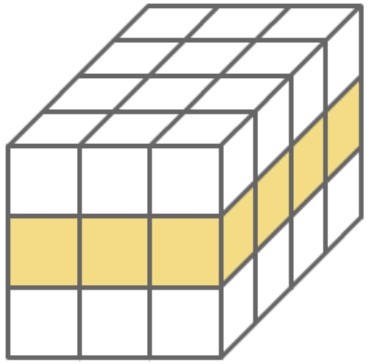
Vector(2:10:2) //2, 4, 6, 8, 10번째 원소



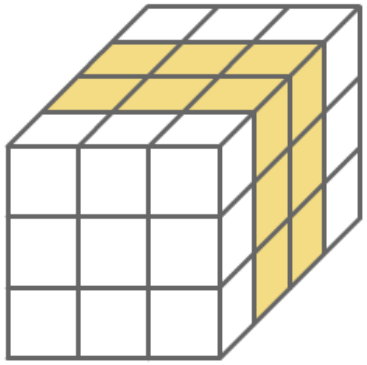
Mat(1:3, 2)



Mat(2:3, 1:3)



Cube(2, 1:3, 1:4)

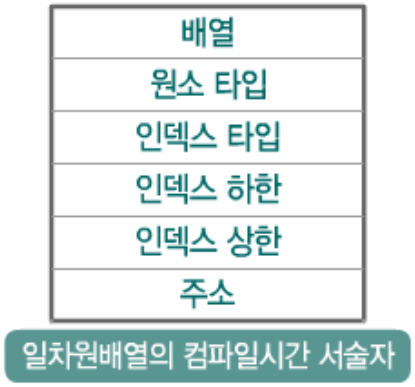


Cube(1:3, 1:3, 2:3)

배열 타입의 구현

❑ 배열의 구현

- 배열 원소의 접근에 대한 코드를 컴파일러가 생성
- 접근 함수를 통해서 첨자 식을 배열의 한 주소에 사상
  - ✓ 접근 함수 구성, 인덱스 범위 검사
  - ✓ 주소(list[k]) = 주소(list[하한]) + ((k-하한) * 원소 크기)
  - ✓ 예) c언어 :하한=>0
- 서술자(descriptor)
  - ✓접근 함수 구성, 인덱스 범위 검사에 필요한 정보를 포함
  - ✓컴파일시간/실행시간 서술자



배열 타입의 구현

❑ 배열의 구현

– 다차원 배열의 저장

✓열-우선 순서(column major order) : FORTRAN

✓행-우선 순서(row major order) : 다른 언어

예

3	5	7
6	2	5
1	3	8

- 3 5 7 6 2 5 1 3 8 .. (행–우선 순서)
- 3 6 1 5 2 3 7 5 8 .. (열–우선 순서)

배열 타입의 구현

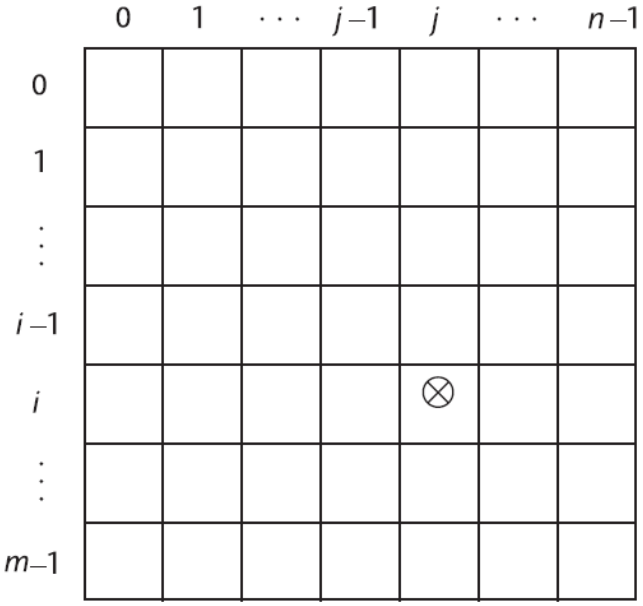
❑ 배열의 구현

- 다차원 배열의 주소(row major)
- a[i, j] = 주소(a[0,0]) + (i번째 행보다 위에 위치한 행의 개수* 행의 크기 + j번째 열보다 왼쪽에 위치한 원소의 개수) * 원소 크기

  = 주소(a[0,0]) + ((i-1)*n +(j-1)) * 원소크기

  = 주소(a[0,0]) –(n+1) * 원소크기 + (i * n+ j) * 원소크기

상수 부분 변수 부분



Q and A