프로그래밍 언어론

Binding

컴퓨터공학과
이만호

충남대학교
CHUNGNAM NATIONAL UNIVERSITY

Binding

학습목표

프로그래밍언어에서 사용되는 변수에 type과 같은 속성을 부여하는 binding 개념에 대해서 학습한다.

학습내용

- 변수의 Binding
- 변수의 Type Binding
- 변수의 Category(부류)
- Named Constant(이름상수)

목 차

알고가기 1

1. C 언어로 작성된 아래의 프로그램 코드를 보고 물음에 답하시오.

```
int foo(int fpar)
{   int   lvar = 24;                --- (1)
    static  int   slvar = 35;       --- (2)
    ...
    return  slvar;   }
```

문장 (1)에서 `lvar`의 값이 `24`로 저장되는 시점은 언제인가?

(a) 프로그램이 실행을 시작하기 직전
(b) foo가 호출되어 실행을 시작할 때
(c) foo가 종료되어 return할 때
(d) 프로그램이 종료할 때

알고가기 2

2. C 언어로 작성된 아래의 프로그램 코드를 보고 물음에 답하시오.

```
int foo(int fpar)
{  int  lvar = 24;                  --- (1)
   static  int  slvar = 35;         --- (2)
   ...
   return  slvar;  }
```

문장 (2)에서 slvar의 값이 35로 저장되는 시점은 언제인가?

(a) 프로그램이 실행을 시작하기 직전
(b) foo가 호출되어 실행을 시작할 때
(c) foo가 종료되어 return할 때
(d) 프로그램이 종료할 때

Binding의 개념

Binding

- 개체(entity)에 속성을 맺어주는 것.
- 변수의 Binding
  - 변수에 변수와 관련된 속성을 맺어주는 것

```
int  age; // 변수 age에 type(int)을 binding – type binding
age = 19; // 변수 age에   값(19)을 binding – value binding
```

- 연산자(operator)의 Binding
  - 기호(symbol)에 연산(operation)의 의미를 맺어주는 것

```
'+' : 더하기(add)
'*' : 곱하기(multiply)
```

Binding Time

- Binding이 이루어지는 시점
- Binding되는 속성에 따라 binding time이 다르다.

Binding Time

Static Binding

- 프로그래밍언어를 설계할 때
  - 연산자 기호 — 연산 (예: '+' — 더하기)
- 프로그래밍언어를 구현할 때 (compiler/interpreter를 작성할 때)
  - 정수형 값의 범위: 16-bit / 32-bit
  - 실수형의 표현
- 프로그램을 compile할 때
  - 변수의 type (C의 경우)
- 프로그램을 메모리에 load할 때
  - 전역변수의 주소 (C의 경우)
  - static 변수의 주소 (C의 경우)

```
int  gvar;   // 전역변수
...
int foo(int fa)
{
  int  n, *p;
  static int  svar;

  p = (int*)malloc(sizeof(int));
  n = 3;
  n = n + 5;
  ...
}
...
```

Dynamic Binding

- 프로그램을 실행할 때
  - static이 아닌 지역변수의 주소 (C의 경우)
  - 부프로그램의 parameter(인자)의 주소
  - 동적으로 할당된 기억장소의 주소

Static Binding vs. Dynamic Binding

Static Binding

- 최초의 binding이 프로그램 실행하기 전에 이루어진다.
- 한번 binding이 이루어진 속성은 프로그램이 종료될 때까지 변하지 않는다.

Dynamic Binding

- 최초의 binding이 프로그램 실행하는 동안 이루어진다.
- 한번 binding이 이루어진 속성이 프로그램 실행 과정에서 다시 발생하는 binding에 의해서 다른 속성으로 변할 수 있다.

변수의 Type Binding

Type Binding과 관련된 주제

Type이 지정되는 방법

- 명시적 선언 (explicit declaration)
- 묵시적 선언 (implicit declaration)
- 값(value)의 배정(assignment)
- Type Inference(추론)

Type Binding이 이루어지는 시점

Static Binding

> 대부분의 compiler 언어에서 프로그램을 compile할 때 type binding이 이루어진다.

> 대부분, 명시적 또는 묵시적 선언 방법에 의해서 type이 지정된다.

Dynamic Binding

> 대부분의 interpreter 언어에서 프로그램을 실행할 때 type binding이 이루어진다.

> 대부분, 값의 배정 방법에 의해서 type이 지정된다.

명시적 선언 vs. 묵시적 선언

명시적 선언 (explicit declaration)

- 변수의 type을 지정하는 선언문에 의해서 type이 지정된다.

예 C 언어에서,

```
int    age, score;
float  average;
```

```
float  SCORE, AVRGE;
...
SCORE = 98.5;
AVRGE = SCORE / 5;
```

장점	Reliability 향상

묵시적 선언 (implicit declaration)

- 언어 설계 시 정의된 규칙에 의해서 변수의 type이 지정된다.
- 프로그램에서 변수가 처음 나타나는 지점에서 type이 지정된다.

예 Fortran에서,
정수형: 변수이름이 I,J,…,N으로 시작
실수형: 변수이름이 기타문자로 시작

```
...
SCORE = 98.5
AVRGE = SCORE / 5
```

장점	Writability 향상
단점	Reliability 하락

Dynamic Type Binding

- 배정문(assignment statement)을 실행할 때 type이 지정된다.
- 배정되는 값의 type에 따라 변수의 type이 지정된다.

예

JavaScript에서,

```
list = [2, 4.33, 6, 8]; // list: 1차원 배열
list = 17.3;            // list: scalar 변수
```

장점

- 유연성(flexibility) 향상
  - Generic subprogram(모든 type에 적용될 수 있는 부프로그램)을 작성하는 것이 가능하다.

단점

- 프로그램 실행 비용 증가
  - Type checking이 프로그램 실행 시 이루어져야 한다.
- Type오류를 발견하기가 어려움

```
n = 3;   d = 5;   s = 7.4;   // n,d:정수형, s:실수형
n = s;   // 입력 실수(원래 의도는 n = d;). 오류는 아님.
         // 변수 n의 type이 실수형으로 바뀜.
```

Type inference(추론)

- 변수가 사용된 주변 상황으로부터 type을 추론(inference)한다.
- 주로 함수형언어(functional languages)에서 적용된다.

예

ML 언어에서,

```
fun area(r) = 3.14 * r * r;        // r:실수형
fun times10(x) = 10 * x;           // x:정수형
fun square(x) = x * x;  // x의 type 지정 불능. Error 발생
```

해결 방법

```
fun square(x): int = x * x;
fun square(x:int) = x * x;
fun square(x) = (x:int) * x;
fun square(x) = x * (x:int);
```

기억장소 Binding

- 변수에 기억장소를 할당하는 것.
  - 개체의 주소가 결정됨 (address binding)
- 기억장소 할당 vs. 기억장소 반납

기억장소 할당(allocation)	기억장소 반환(de-allocation)
유휴 기억장소 집합으로부터 기억장소를 할당한다.	할당 받아 사용하던 기억장소를 유휴 기억장소 집합에 반환한다.

예 C 언어에서,

```
int foo(int fp)   // foo가 호출되면, fp용 기억장소가 할당됨
{ int  k;         // foo가 호출되면,  k용 기억장소가 할당됨
  int  *p;        // foo가 호출되면,  p용 기억장소가 할당됨
  p = (int *) malloc(sizeof(int));  // 기억장소 할당
  ...
  free(p);                          // 기억장소 반납
}
```

- 변수의 존속기간(lifetime)
  - 변수에 기억장소가 할당되는 시점부터 반환되는 시점까지의 시간

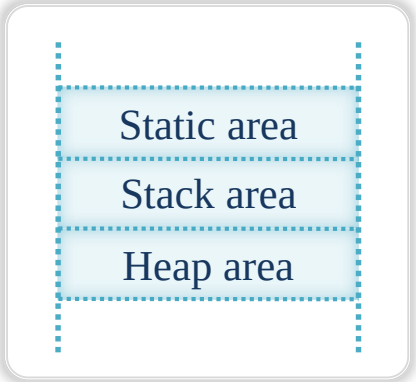
변수의 Category(부류)

변수의 종류

- Scalar 변수, pointer 변수, 구조체 변수, …
- Array 변수

변수의 존속기간(lifetime)에 따른 scalar 변수의 category(부류)

- Static 변수
- Stack-dynamic 변수
- Explicit heap-dynamic 변수
- Implicit heap-dynamic 변수



Static 변수

- 프로그램 실행 전에 static area의 기억장소가 binding된다.
- Binding된 기억장소는 프로그램이 종료될 때까지 유지된다.

예 C에서 `static`으로 선언된 모든 변수, Fortran 77에서 모든 변수

장점

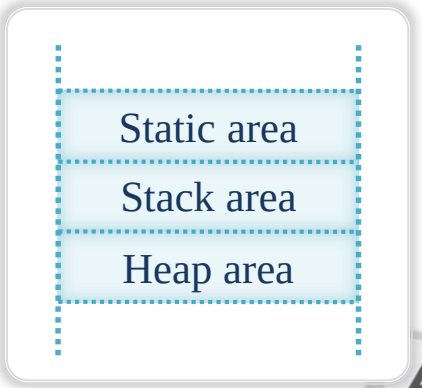
- 프로그램 실행의 효율성
  - 변수에 직접 접근할 수 있음. (direct addressing)
  - 기억장소의 할당(allocation) 및 반환(de-allocation) 시간이 필요 없음.
- Subprogram에서 history-sensitive(과거-민감) 변수 지원
  - Subprogram을 호출해서 실행을 시작할 때, 이전 호출에서 마지막으로 배정된 값을 갖는다.
  - 난수(random number)를 생성할 수 있는 부프로그램 작성에 사용됨,

```
int random() { static int seed;  ... }
```

단점

- 유연성(flexibility) 부족
- Recursive subprogram작성이 불가능함.
- 지역변수들 사이에 기억장소를 공용할 수 없음.

```
int foo() { static int fa[1000]; ... }
Int goo() { static int ga[1000]; ... }
```



Stack-dynamic 변수

- 프로그램이 실행되는 과정에서, 변수 선언문이 활성화될 때 stack area의 기억장소가 binding된다.

```
int foo() { int n;  ... }
```

- Scalar 변수의 경우, 다른 속성(이름, type, scope)은 static binding되기도 한다.

예 C에서, 함수(function)의 지역변수

장점

- Recursive subprogram 작성이 가능함.

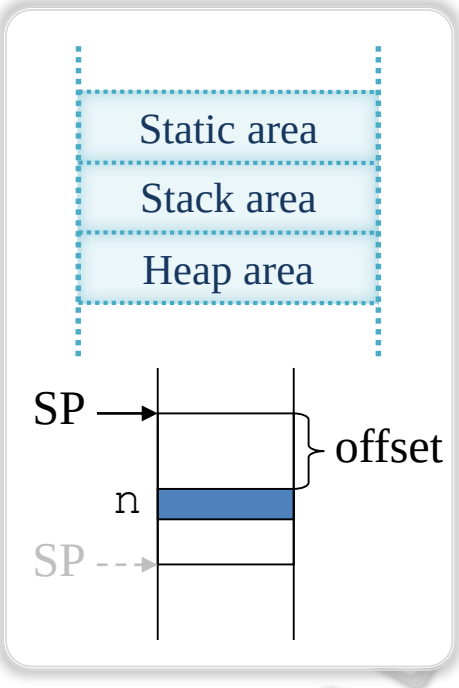
```
int roo() { int rv; ... roo() ... }
```

- 지역변수들 사이에 기억장소를 공용할 수 있음.

```
int foo() { int fa[1000]; ... }
Int goo() { int ga[1000]; ... }
```

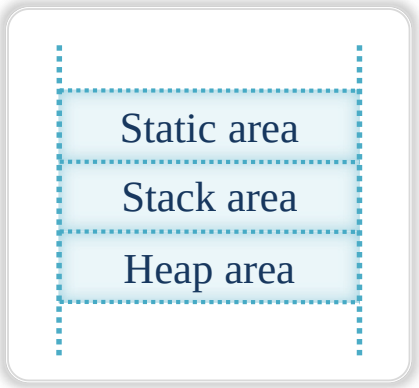
단점

- 프로그램 실행 시간의 증가
  - 기억장소의 할당 및 반환 시간이 추가적으로 필요함.
  - 변수에 간접적으로 접근해야 함. (in-direct addressing)
- Subprogram에서 history-sensitive(과거-민감) 변수를 이용할 수 없음



Explicit heap-dynamic 변수

- 기억장소를 할당하는 문장을 실행할 때 heap area의 기억장소가 binding되고, 반환하는 문장을 실행할 때 반환된다.
- Linked-list(연결리스트)나 tree와 같이 프로그램 실행 중에 크기가 커지거나 줄어드는 경우에 사용된다.



장점

- 필요에 따라 기억공간을 효율적으로 관리하며 사용할 수 있다

단점

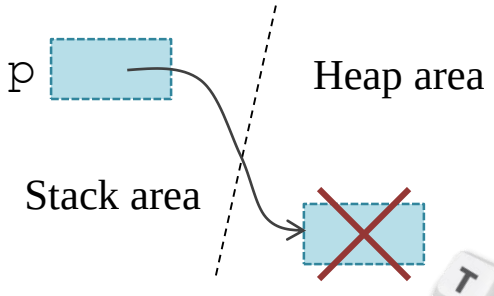
- Pointer나 참조변수를 올바르게 사용하기 어려움 → Reliability 저하
- 기억장소의 dynamic 할당 및 반환에 필요한 실행시간 증가

C

```
int   *p;
...
p = (int*) malloc(sizeof(int));
... *p ...
free(p);
```

C++

```
int   *p;
...
p = new int;
... *p ...
delete p;
```



Implicit heap-dynamic 변수

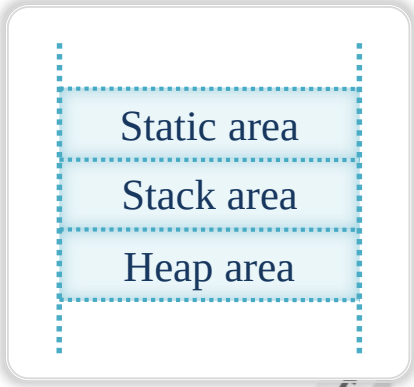
- 배정문(assignment statement)을 실행할 때 기억장소가 binding된다.
  - JavaScript의 string(문자열)

```
var  st = "Hong";    // 길이: 4
st += " Gil-Dong";   // 길이: 13으로 확장됨
```

  - JavaScript의 배열(array)

```
var  na = new Array(34, 52, 19); // 크기: 3
na[8] = 68;                      // 크기: 9로 확장됨
```

- 쓸모가 없어진 기억장소는 보통 시스템에 의한 쓰레기수집(garbage collection)을 통해 반환된다.



장점

- 유연성(flexibility) 향상
  - 기억장소를 필요할 때 할당하고, 필요가 없어졌을 때 반환.
  - 모든 type에 적용될 수 있는 generic code(포괄형 코드) 작성이 가능.

단점

- 기억장소의 dynamic allocation및 de-allocation에 필요한 실행시간 증가
- Compile할 때 오류를 검출할 수 없다. → Reliability 저하

Named constant(이름 상수)

- 변수 이름에 기억장소 binding이 이루어질 때, value binding이 동시에 이루어진다. 한번 binding된 값은 변경될 수 없다.
- Binding 방법에 따른 named constant의 분류
  - Static binding에 의한 이름상수: 정적으로 value binding이 이루어짐
    - Manifest constant(명확 상수)라고 부른다.
  - Dynamic binding에 의한 이름상수: 동적으로 값 binding이 이루어짐

```
int foo() {  // C++의 예
  ...     // width의 값이 미리 binding되어 있어야 함.
  const int Max = 2 * width;    // dynamic binding 발생
  ...    // 변수 Max 값을 변경할 수 없음.
}
```

- 사용 예
  - 특별한 의미를 갖는 상수 값

```
#define  pi  3.14159      // 원주율
#define  G   6.67E-11     // 중력상수
```

  - 프로그램의 여러 곳에서 자주 사용되는 상수 값

```
#define  ArrSize   100   // 배열의 크기
int    a[ArrAize], k;
for (k=0; k<ArrSize; k++) { ... }
```

장점 : Readability, Writability, Reliability 향상, Cost 절감

평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?

총 3문제가 있습니다.

START

평가하기 1

1. C언어에서, 선언문 "`int n;`"에서, 변수 `n`의 **type**으로 `int`가 **binding** 되는 시점은?

(a) 프로그래밍언어를 구현할 때
(b) 프로그램을 compile할 때
(c) 프로그램을 메모리에 load할 때
(d) 프로그램을 실행할 때

평가하기 2

2. C언어에서, 선언문 "static int n;"에서, 변수 n에 주소가 binding되는 시점은?

(a) 프로그래밍언어를 구현할 때
(b) 프로그램을 compile할 때
(c) 프로그램을 메모리에 load할 때
(d) 프로그램을 실행할 때

평가하기 3

3. 아래에는 C code와 scalar 변수의 존속기간에 따른 변수의 category가 나열되어 있다. 주어진 code에 사용된 각 변수와 scalar 변수의 존속기간에 따른 변수의 category가 잘 짝지어진 것은?

```
static  int  s;
...
void foo(int f);
{
  int  q, *p;
  static  int  r;
  p = (int *) malloc(sizeof(int));
  ...
}
```

(ㄱ) Static 변수
(ㄴ) Stack-dynamic 변수
(ㄷ) Explicit heap-dynamic 변수
(ㄹ) Implicit heap-dynamic 변수

(a) s:ㄴ, f:ㄹ, q:ㄹ, r:ㄴ, p:ㄷ, *p:ㄷ
(b) s:ㄱ, f:ㄴ, q:ㄷ, r:ㄱ, p:ㄷ, *p:ㄹ
(c) s:ㄱ, f:ㄴ, q:ㄴ, r:ㄱ, p:ㄴ, *p:ㄷ
(d) s:ㄴ, f:ㄹ, q:ㄱ, r:ㄴ, p:ㄴ, *p:ㄹ

정리하기

- **Binding 시점**
  - Static binding: 프로그램 실행 전에 binding이 이루어짐
  - Dynamic binding: 프로그램 실행 중에 binding이 이루어짐
- **변수의 Type binding 방법**
  - 명시적 선언 (explicit declaration)
  - 묵시적 선언 (implicit declaration)
  - 값(value)의 배정(assignment)
  - Type inference(추론)
- **변수의 존속기간(lifetime)에 따른 scalar 변수의 categories(부류)**
  - Static 변수
  - Stack-dynamic 변수
  - Explicit heap-dynamic)변수
  - Implicit heap-dynamic 변수