Chapter 10.
변수 사용 시 일반적인 문제점

IT개발팀 조영일

목차

- 변수 선언을 쉽게 만드는 방법
- 변수의 초기화에 대한 지침
- 범위
- 지속성
- 바인딩 시간
- 데이터 형과 제어 구조 간의 관계
- 변수를 한 목적으로만 사용하는 방법

Declaration

- 암시적 선언
  - 프로그래밍 언어가 지원하는 가장 위험한 기능

Declaration

- 암시적 선언을 방지하는 방법
  - Visual Basic
    - Option Explicit
  - Perl
    - use strict ‘vars’;
  - C
    - -pendantic-errors / -Werror
  - Javascript
    - jslint / jshint
    - “use strict”;

Declaration

- 실수를 피하기 위한 방법
- 모든 변수를 선언
- Naming convention을 사용
- 변수의 이름을 검사

Initialization

- 부적절한 초기화의 사례
  - 미 할당
  - 더 이상 유효하지 않음
  - 변수의 부분에만 값을 대입

Initialization

- 초기화 문제를 피하는 방법
  - 선언 시 초기화

```
char  studentName [ NAME_LENGTH + 1 ] = {'\0'};   // full name of student
```

Initialization

- 초기화 문제를 피하는 방법
  - 첫 사용 위치 근처에서 초기화

```
' declare all variables
Dim accountIndex As Integer
Dim total As Double
Dim done As Boolean

' initialize all variables
accountIndex = 0
total = 0.0
done = False
...

' code using accountIndex
...

' code using total
...

' code using done
While Not done
   ...
```

```
Dim accountIndex As Integer
accountIndex = 0
' code using accountIndex
...

Dim total As Double
total = 0.0
' code using total
...

Dim done As Boolean
done = False
' code using done
While Not done
   ...
```

Initialization

- 초기화 문제를 피하는 방법
  - 첫 사용 위치 근처에서 선언하고 정의

```
int accountIndex = 0;
// code using accountIndex
...

double total = 0.0;
// code using total
...

boolean done = false;
// code using done
while ( ! done ) {
   ...
```

Initialization

- 초기화 문제를 피하는 방법
  - final이나 const 활용
  - 카운터와 누산기(accumulator) 재사용 주의
  - 클래스 멤버는 생성자에서 초기화할 것
  - 필요하다면 재 초기화
  - 매개변수 유효성 확인

Scope

- "Scope" or "Visibility"
  - 변수가 보이는 범위

Scope

- 언어마다 변수의 scope이 다름
  - 원시적인 언어: 모든 변수가 전역 변수
  - C++ like: block >> routine >> class >> global
  - Java/C#: package or namespace

Scope

- Span

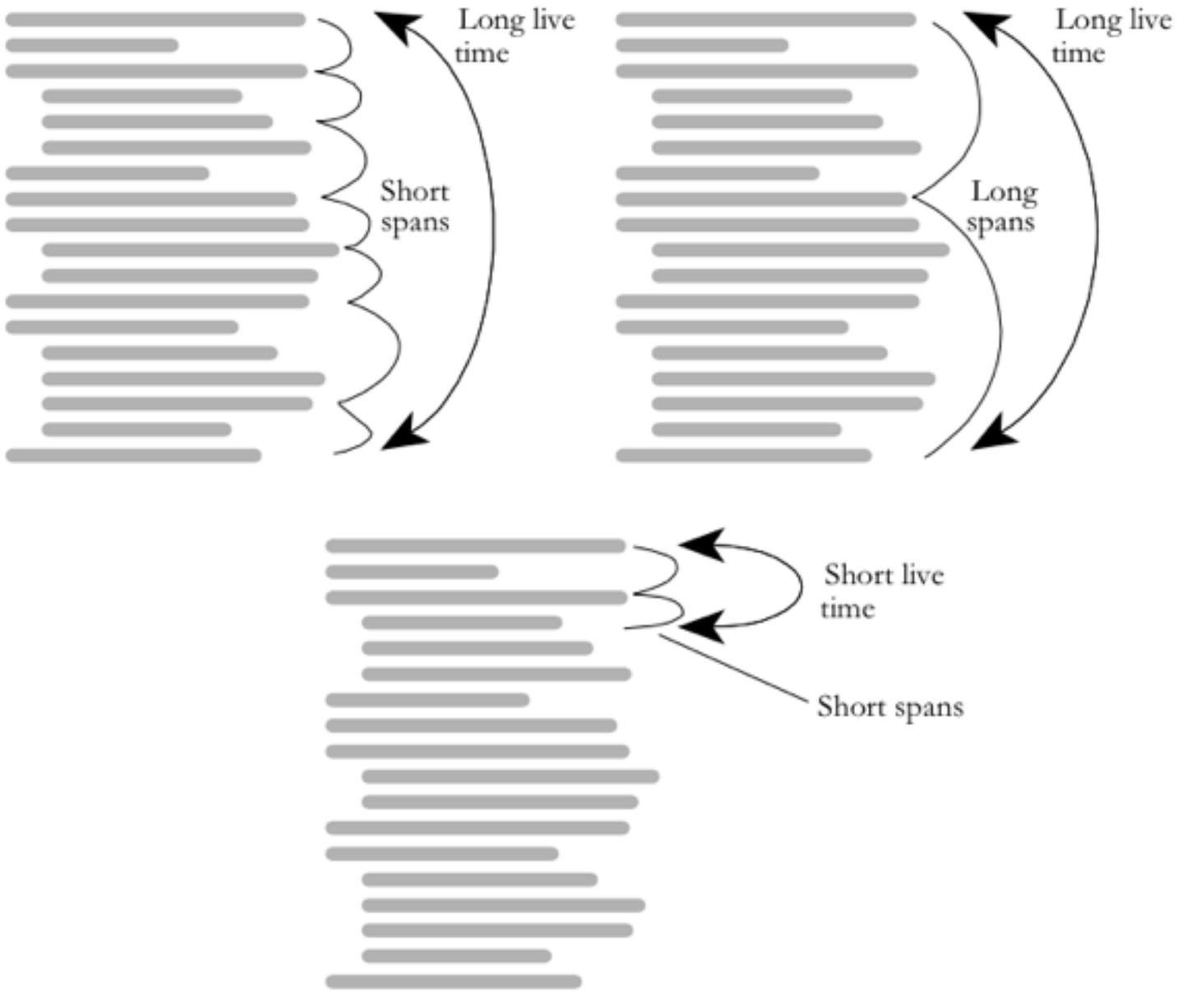
```
a = 0;
b = 0;
c = 0;
a = b + c;
```

```
a = 0;
b = 0;
c = 0;
b = a + 1;
b = b / c;
```

- a의 폭은 2, b의 폭은 1, c의 폭은 0
- a의 폭은 2, b의 평균 폭은 0.5, c의 폭은 1

Scope

- Live time



Scope

```
1   // initialize all variables
2   recordIndex = 0;
3   total = 0;
4   done = false;
    ...
26  while ( recordIndex < recordCount ) {
27  ...
28     recordIndex = recordIndex + 1;
       ...

64  while ( !done ) {
       ...
69     if ( total > projectedTotal ) {
70        done = true;
```

```
    ...
25  recordIndex = 0;
26  while ( recordIndex < recordCount ) {
27  ...
28     recordIndex = recordIndex + 1;
       ...

62  total = 0;
63  done = false;
64  while ( !done ) {
       ...
69     if ( total > projectedTotal ) {
70        done = true;
```

Scope

- 최소화 지침
  - 루프 변수는 루프 직전에서 초기화
  - 변수를 사용하기 전까지는 값을 대입하지 말 것
  - 연관된 명령문들을 그룹화 >> 루틴화
  - 가시도를 최대한 제한
    - global >> class
    - public >> protected >> private
    - loop >> routine

Persistence

- Persistence Types
  - 특정한 코드 블럭이나 루틴에서만 살아남는 경우
  - 허용한 동안 살아남는 경우
  - 프로그램이 종료할 때까지 살아남는 경우
  - 영원히 살아남는 경우

Persistence

- 냉장고에 넣어둔 우유와 같아서 수명 예측이 어려움
- 지침
  - debug나 assertion 활용 —> test case
  - 끝나면 nullify
  - 변수의 값이 지속적이지 않을 거라고 가정할 것
  - 사용 직전에 선언하고 초기화하는 습관

Binding Time

- Binding: 변수와 값을 연결
- Binding time을 가급적 늦출 것

```
titleBar.color = 0xFF; // 0xFF is hex value for color blue
```

```
private static final int COLOR_BLUE = 0xFF;
private static final int TITLE_BAR_COLOR = COLOR_BLUE;
...
titleBar.color = TITLE_BAR_COLOR;
```

```
titleBar.color = ReadTitleBarColor();
```

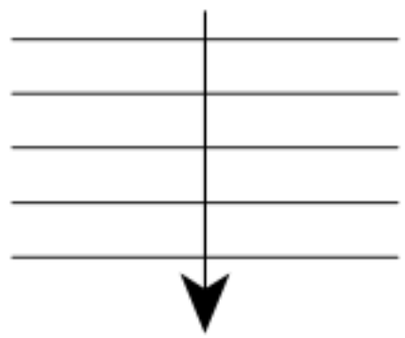
코드 작성 시 >> 컴파일 타임 >> 로딩 시 >> 객체 생성 시 >> Just In Time

Data Type & Control Structure

- 데이터 타입과 제어 구조는 관련이 있음
  - Jackson, 1975
- 지극히 당연한 내용

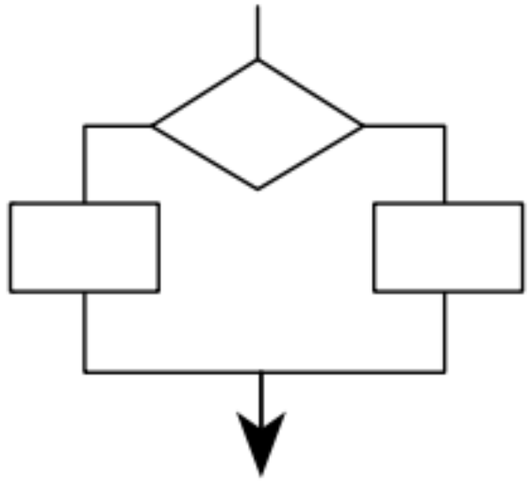
Data Type & Control Structure

- Sequential Data >> Sequential Statements



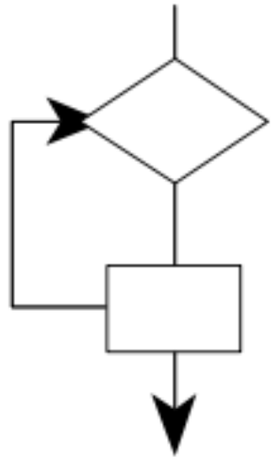
Data Type & Control Structure

- Selective Data >> If/Case Statement



Data Type & Control Structure

- Iterative Data >> Loop Statement



Exactly One Purpose

- 각 변수를 한 가지 목적만을 위해 사용

```
// Compute roots of a quadratic equation.
// This code assumes that (b*b-4*a*c) is positive.
temp = Sqrt( b*b - 4*a*c );
root[0] = ( -b + temp ) / ( 2 * a );
root[1] = ( -b - temp ) / ( 2 * a );
...
// swap the roots
temp = root[0];
root[0] = root[1];
root[1] = temp;
```

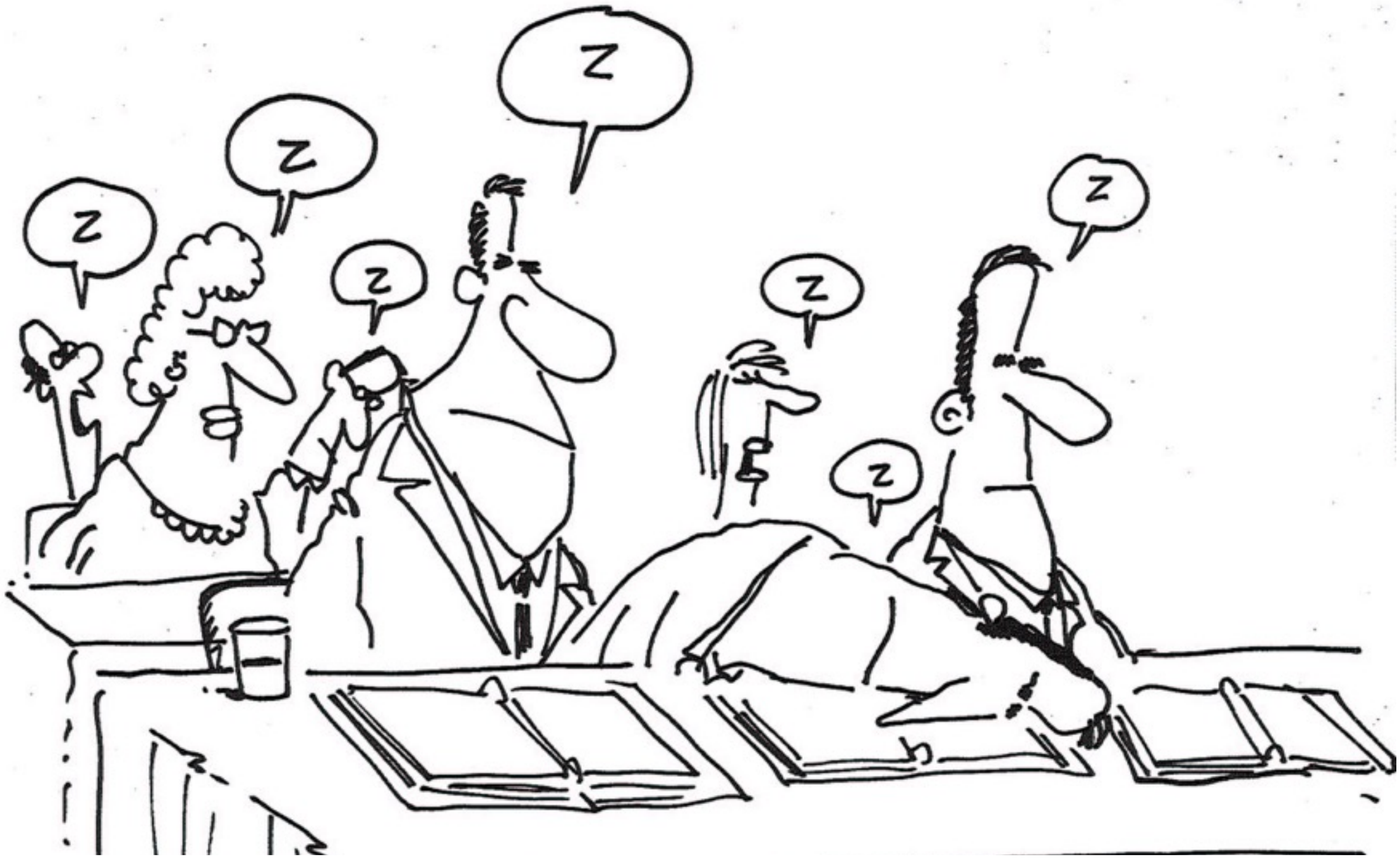
```
// Compute roots of a quadratic equation.
// This code assumes that (b*b-4*a*c) is positive.
discriminant = Sqrt( b*b - 4*a*c );
root[0] = ( -b + discriminant ) / ( 2 * a );
root[1] = ( -b - discriminant ) / ( 2 * a );
...
// swap the roots
oldRoot = root[0];
root[0] = root[1];
root[1] = oldRoot;
```

Exactly One Purpose

- 숨겨진 의미를 갖지 않도록
  - if (pageCount == -1) { throw Exception(msg); }
  - if (bytesWritten < 0) { numDiskDrives = -bytesWritten; }

Exactly One Purpose

- 선언된 모든 변수를 사용
- 참조되지 않은 변수는 높은 오류 발생률과 상관 관계가 있음
    - Card, Church, Agresti, 1986



Thank You