意味解析

計算機科学実験及び演習3（ソフトウェア）2016

構文的には正しいが変なプログラム

パラメータ名が被っている

```
int f(int x, int x) {
    return x;
}
```

配列の添字が int 型に
なっていない

```
int g(int *x, int *y) {
    return x[y];
}
```

…変なプログラムその２

```
int f(int a) {
  if (a) {
    int x;
    x = 1;
  }
  print(x);
}
```

変数を宣言されたブロックの外で使っている

```
int g(int a) {
  if (a) {
    int x;
    x = 1;
  } else {
    print(x);
  }
}
```

同じレベルに入り直してももちろんダメ

課題9（意味解析）で作る関数は

1. 抽象構文木を受け取って
2. さっきみたいな変なプログラムに対応するものなら、エラーメッセージを表示する
3. そうでないなら、変数名を適切なdecl構造体で置き換えた抽象構文木を返す

どういうものをエラーにすべきなのかが実験資料（6節）で定義されている

エラーにならない場合の動作例

```
(fun-def 'int 'f
  (list (param-decl 'int 'x))
  (return-stmt 'x))
```

入力


```
(fun-def  'int (decl 'f 0 'fun (fun 'int 'int))
  (list (param-decl  'int (decl 'x 1 'para 'int)))
  (return-stmt  (decl 'x 1 'para 'int)))
```

出力

実装技法

抽象構文木を操作する関数の例

```
(define (replace-var-ast ast)
  (cond ...
        ((stx:if-stmt? ast)
         (stx:if-stmt (replace-var-ast (stx:if-stmt-test ast))
                      (replace-var-ast (stx:if-stmt-tbody ast))
                      (replace-var-ast (stx:if-stmt-ebody ast))))
        ((stx:var-stmt? ast)
         (stx:lit-exp 10))
        ...
```

ノードの種類で場合分け

再帰で呼び出して

ばらして

また組み立てる

この関数では変数参照式を定数式の10で置き換える

デルタ関数（環境）

変数に割り当てられた情報を管理するためのもの

```
(define initial-delta (lambda (x) #f))
(define (extend-delta delta x data)
  (lambda (y) (if (equal? x y) data (delta y))))
```

デルタ関数の使用例

```
> (define d initial-delta)
> (d 'x) ;'xの情報を探そうとするがないので
#f
> (define new-d (extend-delta d 'x 10)) ;'xに関する情報（10）を追加
> (new-d 'x)
10
> (new-d 'y)
#f
```

デルタ関数の代替（連想配列）

```
(define initial-env '())
(define (extend-env env x data)
  (cons (cons x data) env))
(define (lookup-env env x)
  (if (null? env) #f
      (if (equal? x (car (car env))) (cdr (car env))
          (lookup-env (cdr env)))))
```

意味解析の実装例

変数宣言の部分の一部

とりあえずノードを分解する

```
(define (sem-ast env cur-lv ast)
  (cond ...
        ((stx:var-decl? ast)
         (let* ((type (stx:var-decl-ty ast))
                (name (stx:var-decl-name ast))
                (pos (stx:var-decl-pos ast))
                (obj (env name)))
           (if obj (...) (...))))
        ...
```

環境から obj を探す

obj が見つからなかった場合
（次のスライド）

obj が見つかった場合
（次の次のスライド）

obj が見つからなかった場合

```
...
(let ((new-obj (decl name cur-lv 'var type)))
  (cons (extend-delta env name new-obj)
        (stx:var-decl type new-obj pos)))
...
```

抽象構文木の type が
意味解析で使用する
type と異なる場合は
適当に変換すること

新しい情報を
追加した環境

変数名をオブジェクトで
置き換えた抽象構文木

obj が見つかった場合

```
...
(let ((lv (decl-lev obj)
      (kind (decl-kind obj))
  (cond ((or (equal? kind 'fun) (equal? kind 'proto))
         (if (equal? cur-lv 0)
             (print-error “error message”)
             (...)))
        ...))
...
```

実験資料 6.2 の変数宣言の場合の 2番目に対応

obj が見つかった場合と同じ処理