Project 3

Semantic Analysis

Projects

1. Lexical analyzer
2. Yacc programming
3. **Semantic analysis**
4. Code generation

Overview

- Embedded action
- Grammar
- Semantic check
- Output format & Tips
- Submission

Embedded Action

Embedded Action

- thing: A B ;

- thing: A { printf(“seen an A”); } B ;
- thing: A fakename B ;

  fakename: /* empty */ { printf(“seen an A”); } ;

- thing: A { $$ = 17; } B { printf(“%d”, $2); } ;

  $$ $1 $2 $3 $4

Embedded Action

- Embedded action can cause shift/reduce conflict in otherwise acceptable grammars

thing: abcd | abcz ;
abcd: ‘A’ ‘B’ ‘C’ ‘D’ ;
abcz: ‘A’ ‘B’ ‘C’ ‘Z’ ;

thing: abcd | abcz ;
abcd: ‘A’ ‘B’ ‘C’ ‘D’ ;
abcz: ‘A’ ‘B’ { somefunc(); } ‘C’ ‘Z’ ;
fakename

Grammar

Grammar

- Cannot declare variable and initialize simultaneously
  - int a = 0; /* syntax error */

- No anonymous struct declaration
  - struct { int x;, int y; } w; /* syntax error */

Grammar Change

- No following ASSIGNOP
  - +=, -=, *=, /=, %=
- No multiply, divide, modulo operation on binary expression
  - binary ‘*’ binary
- Added Pre-increment/decrement operation
  - INCOP unary
  - DECOP unary

Semantic Check

Semantic Check

- If the variable and function are declared
- Re-declaration of same variables at same scope
- No Type Casting
- Pointer Operation
- Operation on Structure
- Structure Pointer Declaration
- Function
- LValue Checking
- Operand Checking

Undeclared Variables & Functions

- Defining variables or function call which is not declared makes error
  - variable

```
// int a;
a = 0;              /* error */
```

  - function call

```
// void foo();
foo();              /* error */
```

Redeclaration of Same Variables at Same Scope

```
{
        int a;
        int a;          /* error */
        char a;         /* error */
}
{
        int a;
        {
                int a;  /* OK */
        }
}
```

No Type Conversion

- No explicit type casting

  ex) char a;

  int b = (int) a; /* error */

- No implicit type casting

- 0 cannot be used as NULL
  - 0 != NULL

  ex) int* b = 0; /* error */

NULL

- 0 is always integer

```
int *a[2];
int *b;
int c;
char *d;
char e;

a = 0; /* error (int** != int) */
b = 0; /* error (int* != int) */
b = 1; /* error */
c = 0; /* legal */
d = 0; /* error */
e = 0; /* error */
```

Pointer Operation

- Only following operations are admitted
  - pointer operator

    *, &
  - increment / decrement operator

    ++, --
  - relation operator

    >, <, >=, ==, !=
  - pointer arithmetic

    pointer + integer, pointer - integer

Operation on Structures

- '.' must have structure type operand left.
- '->' must have pointer to structure type operand left
- ID following '.', '->' must be defined in the structure type
- Only pointer can be operator of unary '*'

Structure Declaration

- Structure type must be defined before declaration of the structure type instance

- Redefining structure type is illegal
  - scope is not applied to struct type
  - remember this is against C/C++ standard

Structure Pointer Declaration

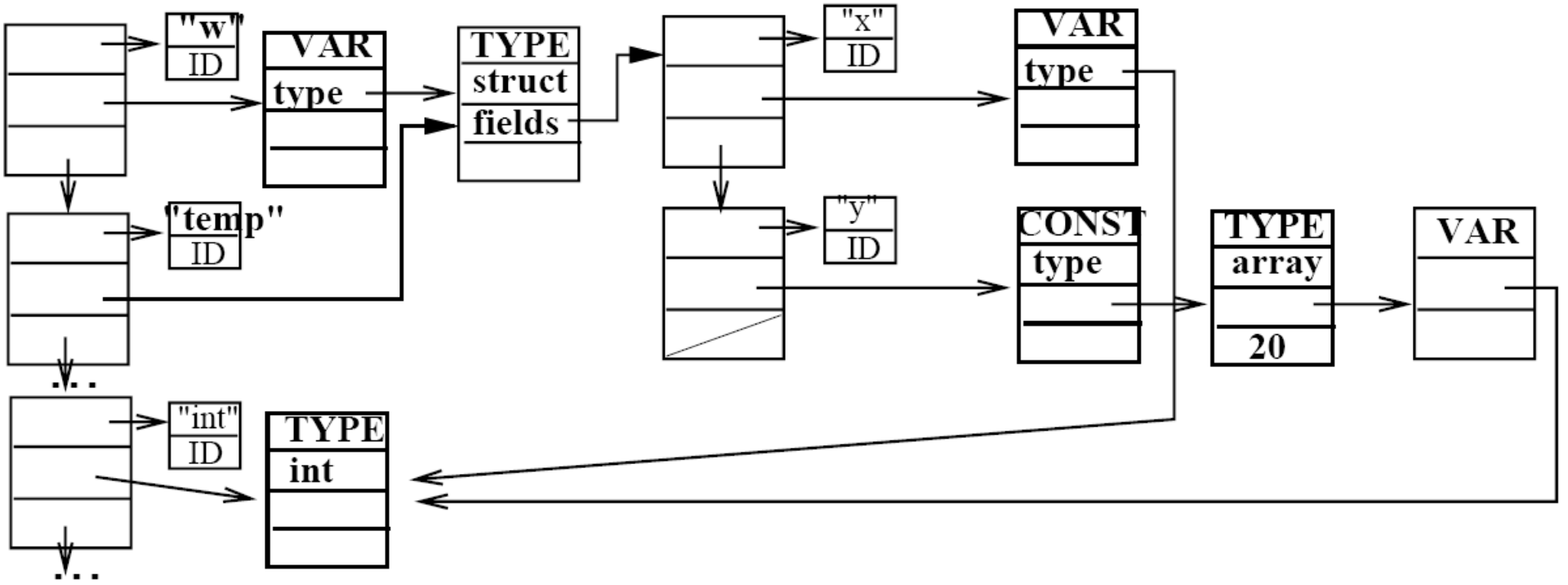
- When structure pointer type variable is declared, lookup structure type
- Link if the structure type is defined
- Otherwise, generate incomplete type error
  - this is also against ANSI C/C++ standard

Structure Pointer Declaration

- ex)

  struct temp { int x; int y[20]; } w;

  struct temp *w1;



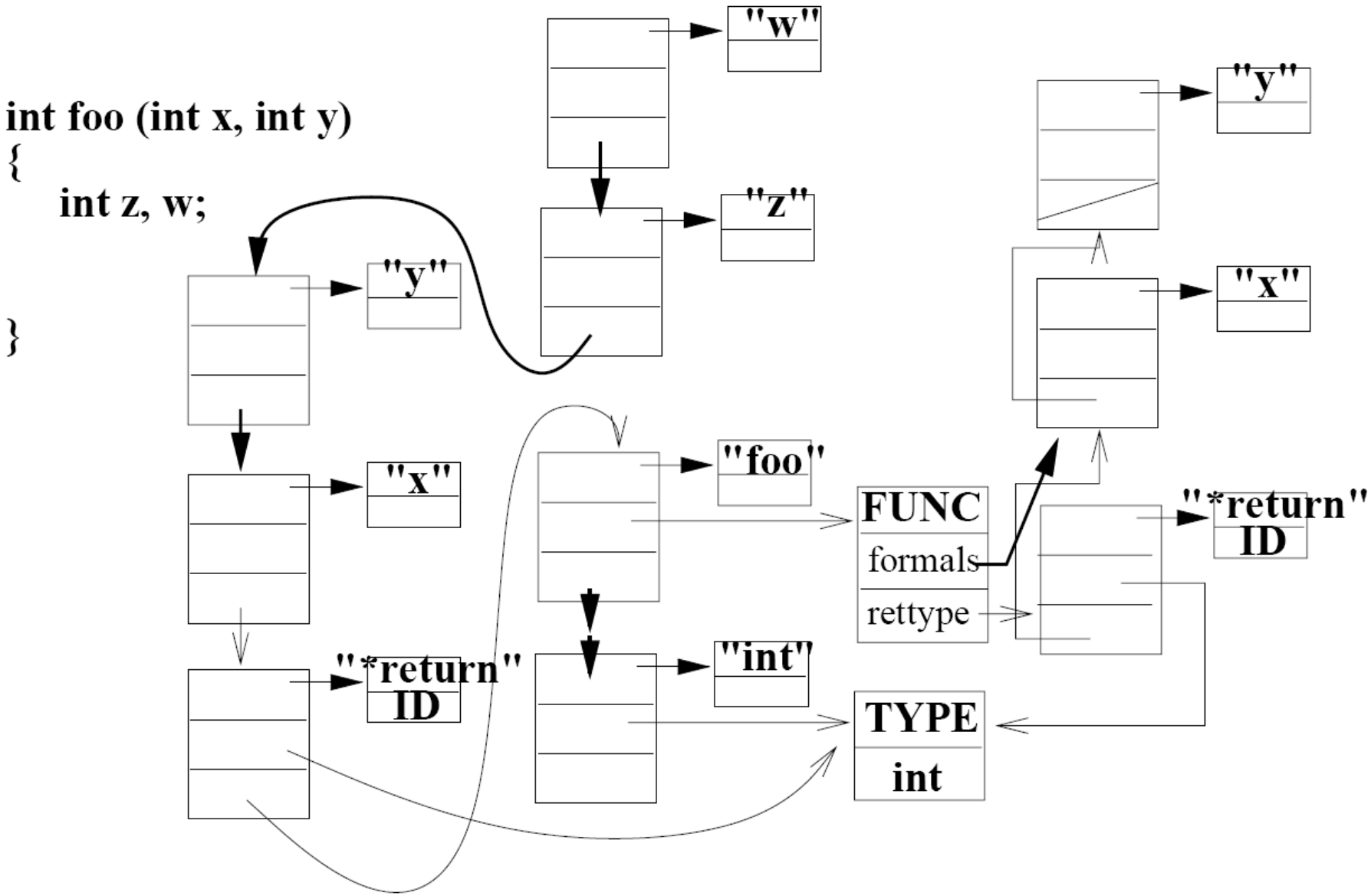
Structure Declaration

```
struct a {
        struct b x;          /* incomplete type error */
        struct b* p;         /* incomplete type error */
        struct b { } y;      /* OK!! */
};
struct b {                   /* error */
};
int func() {
        struct b { } x;      /* error */
}
```

Function

- Check return type with the previous function declaration
  - check strictly, not using implicit rules
- Check actual arguments with formal arguments
  - check strictly, not using implicit rules
- Check type of the expression following return type of the function

Function


int foo (int x, int y)
{
int z, w;
}
"w"
"z"
"y"
"x"
"foo"
"int"
"*return"
ID
FUNC
formals
rettype
TYPE
int
"y"
"x"
"*return"
ID

LValue (Assignment) Checking

- Should have same type
- Exception : pointer = array

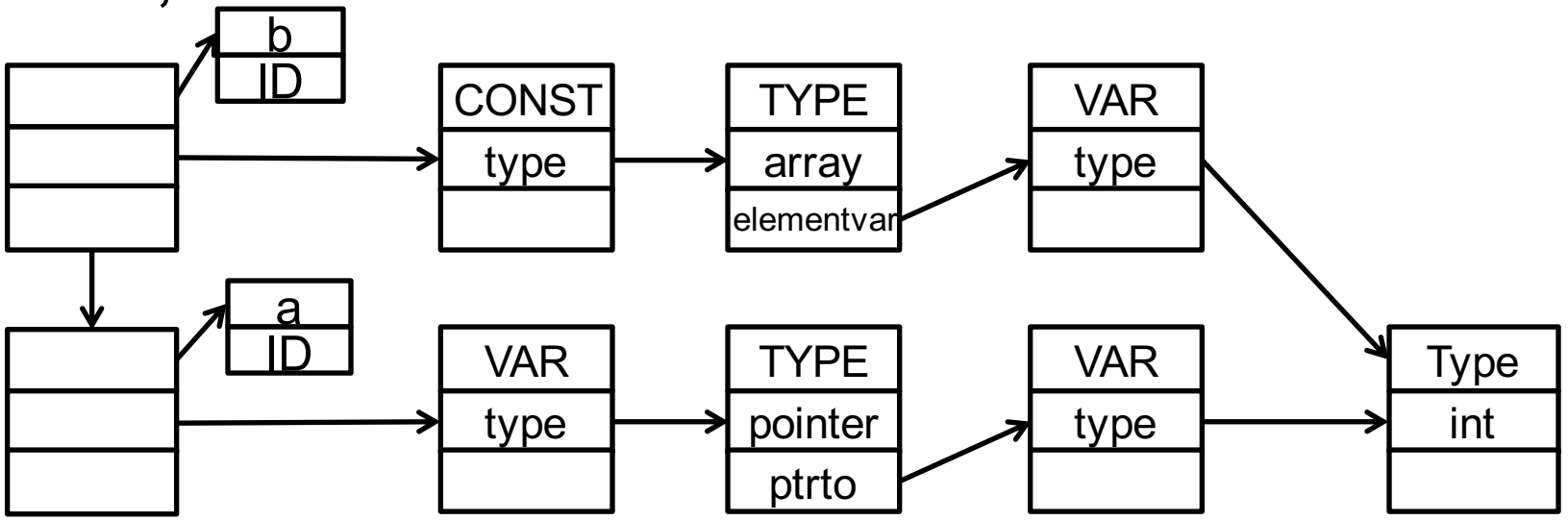
```
int *a[5];
int *b;
int c[10];
struct temp1 { int a; } *s1;
struct temp1 s2;
struct temp2 { int b; } *s3;

a = b; /* error (int**) != (int*) */
b = c; /* legal (int*) == (int array) */
s1 = s3; /* error (*temp1 != *temp2) */
s1 = s2; /* error */
s1 = &s2; /* legal */
```

LValue (Assignment) Checking

```
int a[10];
int *b;
a = 0;          /* error */
a[0] = 0;       /* legal */
b = a;          /* legal */
b = &a;         /* error */
b = &a[10];     /* legal */
b = &b;         /* error */
b = &*(a+5);    /* error */
b = &(b++);     /* error */
b = &*(b++);    /* legal */
```

LValue (Assignment) Checking

```
int *a;
int b[10];
a = b;
```



Operand Check: Unary -

- Only for integer

```
int a;
char b;

a = 10;
b = 'a';
a = -a;  /* legal */
b = -b;  /* error */
```

Operand Check: INCOP, DECOP

- For char, int, pointer

```
int a;
char b;
int* c;
char d[10];
struct temp { int a;} e;

a++;
--a;
b++;
c++;
--d;      /* error */
++e;      /* error */
```

Operand Check: Binary +, -

- Legal operand
  - int ± int
  - pointer ± int
  - int + pointer

- Error operand
  - Array ± int
  - int + Array
  - …

Operand Check: Relop, Equop

- >=, >, <=, <, ==, !=

- char OP char
- int OP int
- pointer OP pointer (pointers must point same type)
- return int variable as result

```
int *a;
int *b;
(a > b) || ( a == b )
 1 or 0      1 or 0
```

Operand Check: Logical Operators

- &&, ||, !
- Only for int variable
  - int && int
  - int || int
  - ! int
- Input test file
  - int variables are derived from Relop, Equop, Logical op
  - Don't need to check whether it is derived from relop/equop/logical op or not
    - ex) a = 5 * (b == 0) /* OK */

NULL

- No null in input file
- 0 is always integer

```
int *a[2];
int *b;
int c;
char *d;
char e;

a = 0; /* error (int** != int) */
b = 0; /* error (int* != int) */
b = 1; /* error */
c = 0; /* legal */
d = 0; /* error */
e = 0; /* error */
```

Output & Tips

Output

- Output format
  - filename:line: error: description
  - Use read_line() function to get line number
  - ex)
    - test.c:5 : error: ‘i’ undeclared (first use in this function)
    - test.c:11: error: invalid initializer

Skip error code

- Should be able to proceed to next step when error occurs
- Return null when error occurs

```
int a;
char a; /* error */
a = 1;     /* legal (int) = (int) */
a = 'c';   /* error */

Var_decl
         : pointers ID {
            if($1==0){  // Not pointer
                  if(check_is_declared($2))
                   declare($2,$$=makevardecl(NULL));
                  else
                   $$ = NULL;
            }
```

Tips

- Carefully follows the implementation in class handout.
  - Grammars are almost same.
- Implement your own type check functions for data structures.
  - Improves code readability
  - Faster debugging
  - Be careful for segmentation fault(accessing NULL pointer)
- Always beware of how information flows while reduce occurs.

Submission

- 제출 기한
  - 11월 23일 23시 55분
- 제출 방법
  - newetl.snu.ac.kr을 통해서 제출
- 제출 파일
  - subc.l, subc.y, subc.h, hash.c, hash.h 등 소스파일과 Makefile, readme 파일, 결과 보고서을 압축해서 zip파일로 제출
  - 결과 보고서에는 구현방법, 구현내용,각 제한사항에 대한 문제점에 대해서 작성한다
  - 파일명: project3_학번.zip
  - readme 파일에는 이름, 학번, 이메일, 실행방법(Makefile을 변경하였을 경우)을 적는다.

Notice

- 수업 게시판 확인
  - 수정 또는 추가되는 사항은 항상 게시판을 통하여 공지
  - 제출 마지막날까지 공지된 사항을 반영해서 제출
- 소스코드에 자세히 주석달기
- Cheating 금지 (F처리, 모든 코드 철저히 검사)
- TA
  - 정인창 (301동 851호)
  - E-mail: jic0729@altair.snu.ac.kr