Project 3

Semantic Analysis

Virtual Machine & Optimization Laboratory
Dept. of Electrical and Computer Engineering
Seoul National University

중요 공지

Deprecation of "void" token

- 프로젝트의 핵심과 거리가 멀며, 문제를 일으킬 수 있어 제거하기로 결정
- void 와 관련된 grammar rule 모두 제거 바람

- `type_specifier:`

```
…
| VOID
…
```

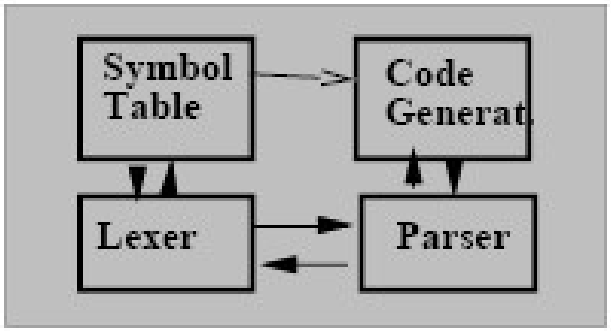
- `func_decl:`

```
…
| type_specifier pointers ID '(' VOID ')'
…
```

Projects

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

Phase Ordering of Compiler Front-Ends



Lexical analysis (lexer)

- Break input string into “words” (lexeme) called *tokens*

Project 1

Syntactic analysis (parser)

- Recover structure from the text and put it in a parse tree

Project 2

Semantic Analysis

- Discover “meaning” (e.g., type-checking)
- Prepare for code generation
- Works with a symbol table

Project 3

Project 3

Project 3

- Semantic Analysis를 통해 Semantic Error 체크 및 에러 메시지 출력

Example

```
int main() {
  int a;
  a = 5;        // Syntactic : OK. Semantic : OK
  a = 'a';      // Syntactic : OK. Semantic : Error => Print message
}
```

TODO

- 수업시간에 배운 내용을 토대로 **Scoped Symbol Table** 구현 `CH8 ppt`
- subc.y 문법의 각 terminal과 nonterminal 사이 적절한 위치에 action(C코드)을 삽입해서 Symbol Table을 이용해 **Semantic Error 체크** `Project3_slides.pdf`
- 에러가 발견될 경우 **메시지 출력** `project3.pdf`

SEMANTIC CHECK

Semantic Check

- Undeclared Variables & Functions
- Re-declaration
- Type Checking
- Structure & Structure pointer Declaration
- Function Declaration

Undeclared Variables & Functions

Defining variables or function call which is not declared makes error

*Implicit declaration, recursive functions & structs 은 존재하지 않는다고 가정

- variable (undeclared)

```
// int a;
a = 0;      /* error: use of undeclared identifier */
```

- variable (out of scope)

```
{ int a; }
a = 0;      /* error: use of undeclared identifier */
```

- function call (undeclared)

```
// void foo();
foo();      /* error: use of undeclared identifier */
```

Re-declaration

Re-declaration of variable, struct, function makes error

* Forward declaration, function overloading은 없다고 가정
* Variable, Struct, Function 등 서로 다른 종류끼리 이름이 겹치는 경우는 고려하지 않음

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

Type Checking (Assignment Operation)

Assignment Operation Semantic Check

다음과 같은 순서로 Semantic Check

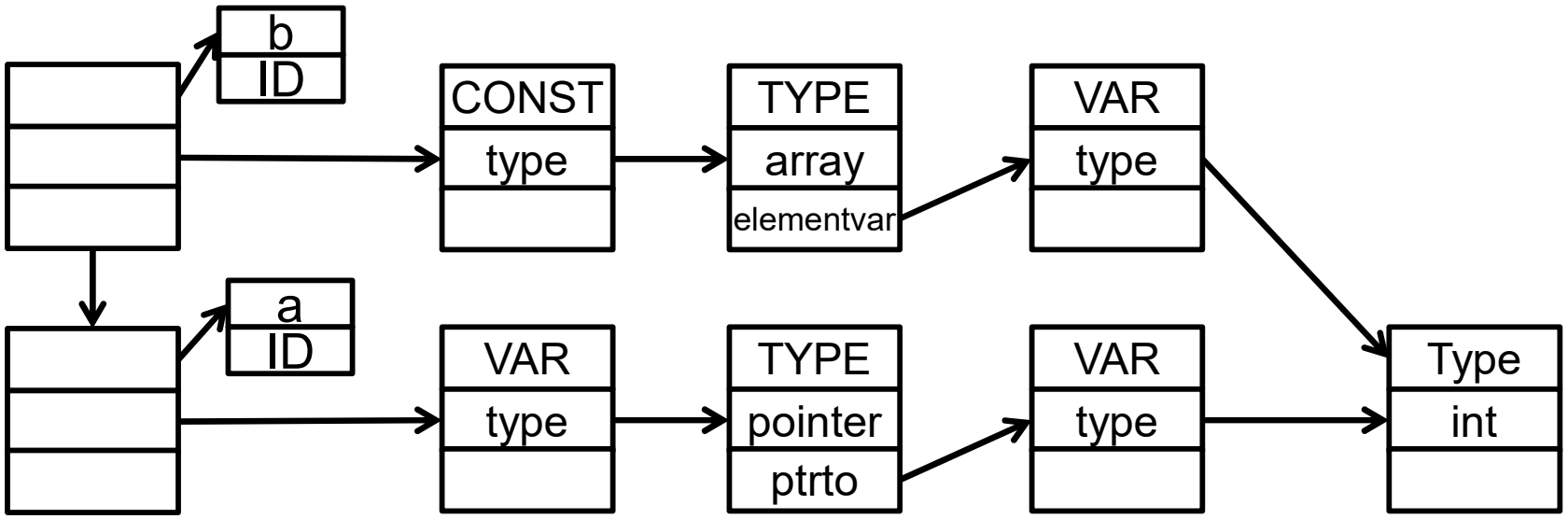
1. LHS가 variable인지 체크
2. LHS와 RHS의 타입이 같은지 체크
3. RHS가 NULL인지 체크

```
int a;
char b;
a = b; /* error: incompatible types for assignment operation*/
5 = a; /* error: lvalue is not assignable */
a = NULL; /* error: cannot assign 'NULL' to non-pointer type */
a = 5; /* legal */
```

Type Checking (Assignment Operation)

Example2

```
int *a;
int b[10];
a = b; /* error: incompatible types for assignment operation*/
b = a; /* error: lvalue is not assignable */
```



Type Checking (Assignment Operation)

Example3

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

a = b; /* error: lvalue is not assignable */
b = c; /* error: incompatible types for assignment operation */
s1 = s3; /* error: incompatible types for assignment operation */
s1 = s2; /* error: incompatible types for assignment operation */
s1 = &s2; /* legal */
```

Type Checking (Binary +, -)

Legal operand

- Only for integer
- int $\pm$ int

Error operand

- Array $\pm$ int
- int + Array
- pointer $\pm$ int
- int + pointer
- …

Type Checking (Unary -)

Only for integer

```
int a;
char b;

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

Type Checking (Logical Operators)

&&, ||, !

Only for integer

- int && int
- int || int
- ! int

Input test file

- int types 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 */

Type Checking (INCOP, DECOP)

For char, int

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

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

Type Checking (Relop)

>=, >, <=, <

char OP char

int OP int

return int type as a result

```
int result;
int a;
int b;
result = (a > 5) || ( a <= b );
```

Type Checking (Equop)

==, !=

char OP char
int OP int
pointer OP pointer (only same type pointer)

Return int type as a result

Pointer == array, array == array 등은 고려하지 않음

```
int result;
int *a;
int *b;
char *c;
result = ( a == b );
result = ( a == c ); /* error */
```

Type Checking (Pointer Operators)

- **Indirection, address-of operator : *, &**
  - '*' must have pointer type operand right
  - '&' must have variable type operand right
- NULL
  - You have you implement it properly.
  - **`0` cannot be used as NULL**

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

a = 0; /* error: incompatible types for assignment operation */
a = NULL; /* legal */
a = &b; /* legal */
a = *b; /* error: indirection requires pointer operand */
&b = a; /* error: lvalue is not assignable */
b = &c; /* error: cannot take the address of an rvalue */
b = 0; /* legal */
b = *a; /* legal */
```

Type Checking (Struct Operators)

- **Struct operator : ., ->**
  - '.' must have structure type operand left.
  - '->' must have structure pointer type operand left
  - An identifier followed by '.', '->' must be defined as the structure type

```
struct str1 {int i; char c;};

struct str1 st1;
struct str1 *pst1;

int main() {
  int i;
  i = st1.i;
  i = st1.i2; /* error: no such member in struct */
  i = st1->i; /* error: member reference base type is not a struct pointer */
  i = pst1->i;
  i = pst1.i; /* error: member reference base type is not a struct */
}
```

Type Checking (Array Operator)

- **Array operator : [ ]**
  - A[i] : *A* must be an array type

```
int a[5];
int b;

b = a[1];
a[1] = b;
a[1] = b[1]; /* error: subscripted value is not an array */
```

Structure & Structure pointer Declaration

Structure

- Structure type must be defined before declaration of the structure type instance
- Structure declaration is always regarded as a global declaration
- Redefining structure type is illegal
  - ✓ scope is not applied to struct type
  - ✓ remember this is against C/C++ standard

Structure & Structure pointer Declaration

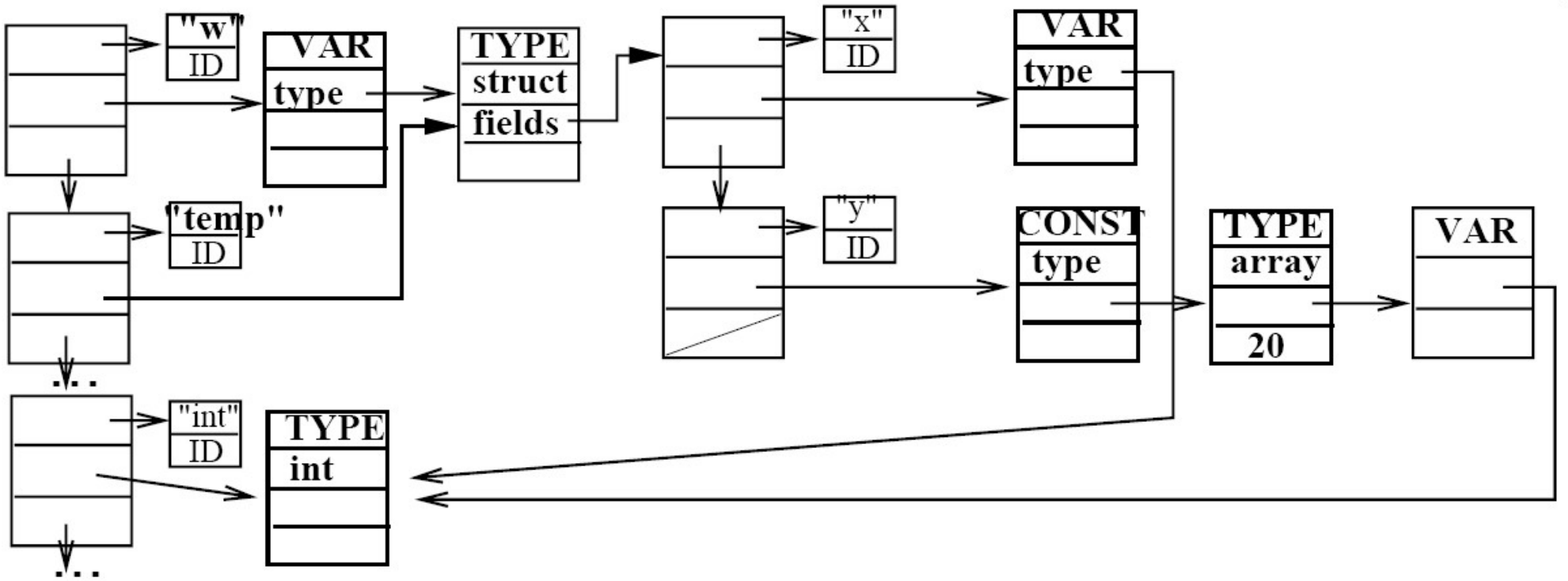
Structure Pointer

- 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 & Structure pointer Declaration

ex)

```
struct temp { int x; int y[20]; } w;
struct temp *w1;
```



Structure & Structure pointer Declaration

- Struct가 정의되지 않은 상태에서 사용하려 할 때 에러

```
struct a {
    struct b x;          /* error: incomplete type */
    struct b* p;         /* error: incomplete type */
    struct b { } y;      /* OK */
};

struct b {               /* error: redeclaration */
};

int func() {
    struct b { } x;      /* error: redeclaration */
}
```

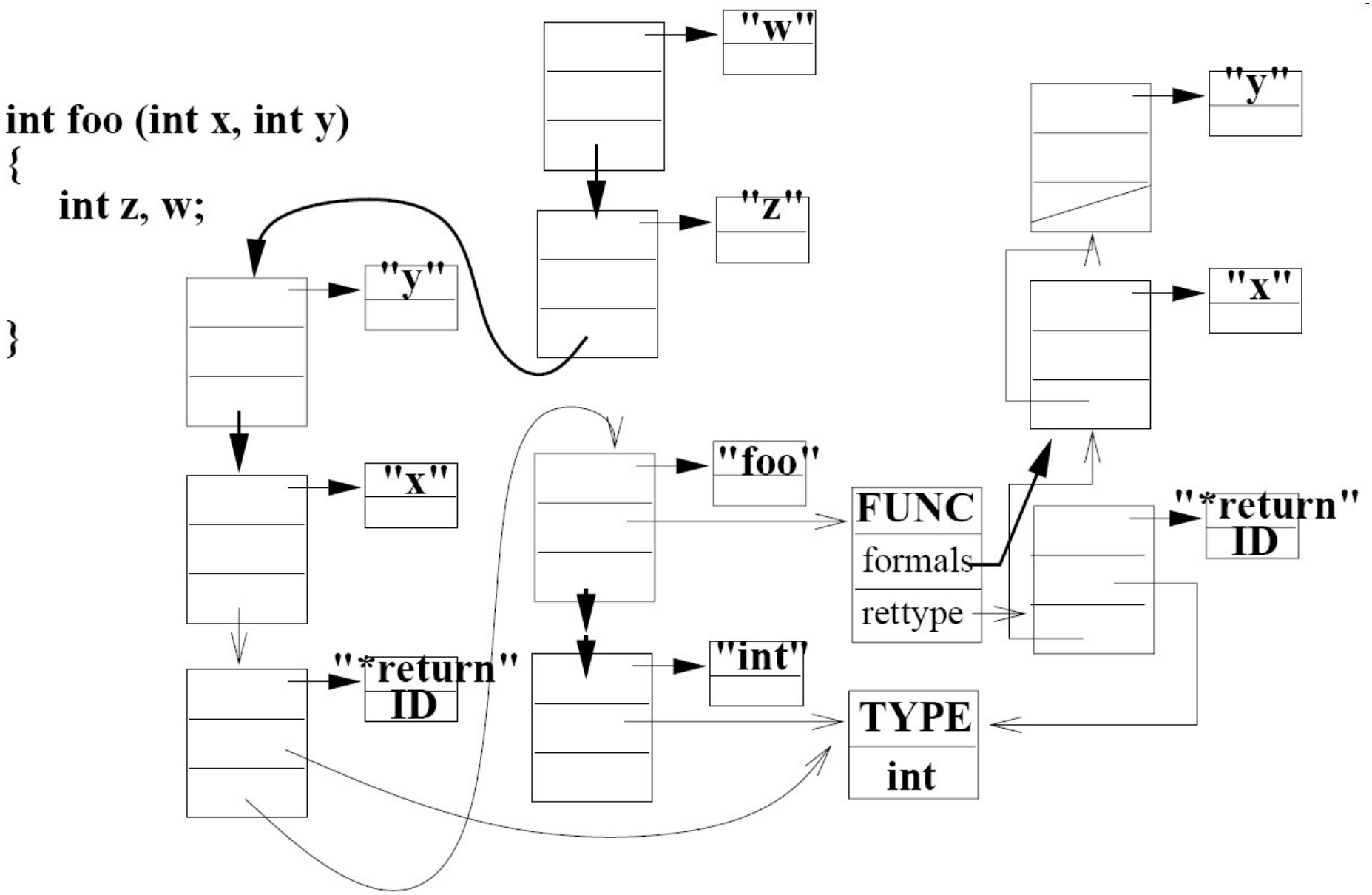
Function Declaration

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

```
int func1(int a, char b) { return 0; }
int func2(int a, char b) { return 'c'; } /* error:incompatible return types */
int func1() {} /* error: redeclaration */

int main() {
  int a;
  int b;
  char c;
  b = func1(a, b); /* error: incompatible arguments in function call */
  b = func1(a, c);
  c = func1(a, c); /* error: incompatible types for assignment operation */
  b = a(); /* error: not a function */
}
```

Function Declaration



GRAMMAR

Grammar

*** Syntax Error가 발생하는 코드는 채점 시 테스트 케이스로 들어가지 않음**

e.g.

Cannot declare variable and initialize simultaneously

- int a = 0; /* syntax error */

No anonymous struct declaration

- struct { int x; int y; } w; /* syntax error */

Cannot declare variable after other statement(stmt) in a scope

- int a;
  int b;
  a = 5; /* stmt */
  int c; /* syntax error */
  { int a; } /* OK */

Grammar

*** Syntax Error는 아니지만 채점 시 테스트 케이스로 들어가지 않는 경우**

- 자기 자신을 call하는 함수
- 자기 자신을 멤버로 갖는 구조체
- 리턴문이 없는 함수 (e.g. int foo() {};)
- Struct, Function의 Forward Declaration
- Function의 Implicit Declaration, Overloading
- String Constant (e.g. char* a = "Hello";)
- 배열과 포인터, 배열과 배열간 비교연산
- Variable Length Array

Output & Tips

Virtual Machine & Optimization Laboratory
Dept. of Electrical and Computer Engineering
Seoul National University

Output

Output format

- <filename>:<line_num>:(SPACE)<error>(SPACE)<description>
- (SPACE) is a space character.
- Use read_line() function to get line number (Project #3 *subc.l* skeleton code)
- ex)
  - `test.c:5: error: an error`

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;
             }
```

Multiple Errors

여러 에러가 동시에 발생하는 경우, 소스코드의 라인마다 에러는 1개씩만 출력
(파싱할 때 먼저 찾을 수 있는 에러를 출력)

e.g.
아래 예제의 경우 `expr->unary '=' expr` 를 통한 REDUCE가 일어나기 전
`unary->unary '[' expr ']'` 이 먼저 REDUCE되므로,
`subscripted value is not an array` 에러만 출력

```
int func() {return 1;}
int main() {
  int a;
  func = a[1]; /* error: lvalue is not assignable,
                  error: subscripted value is not an array */
  return 1;
}
```

Multiple Errors

한 Production에서 여러 에러가 발생해서 Semantic check에 우선순위를 정해야 하는 경우가 있다면, 본인이 생각했을 때 더 나은 방향으로 구현한 뒤 보고서에 작성

(채점 시에는 아예 잘못된 에러가 출력되는 경우가 아니라면 맞게 채점할 것)

e.g.

아래 예제의 경우 `expr->unary '=' expr` 에서 REDUCE가 일어날 때,

1) LHS가 variable인 것을 먼저 검사하는지 2) LHS, RHS가 same type 인지를 먼저 검사하는지에 따라 출력되는 메시지가 달라질 것이다.

참고로 이 예제의 경우 9p 에서 따로 순서를 지정해 놓았기에 **`lvalue is not assignable`** 에러가 출력되는 것이 맞다.

```
int func() {return 1;}
int main() {
  int a;
  func = a;      /* error: lvalue is not assignable
                    error: incompatible types for assignment operation*/
  return 1;
}
```

Tips

Carefully follow the implementation in class handout.

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.

Check test code in `open_test` directory.

```
Ex) ./subc open_test/func_op.c > result
```

Submission

제출 기한

- December 4, 2024

제출 방법

- etl.snu.ac.kr을 통해서 제출

제출 파일

- 'src' directory 안의 파일들과 'report.pdf' 를 제출

- report.pdf 를 project3 디렉토리 안에 복사한 후 submit.sh 로 압축
  - project3 의 subdirectory 도 인식할 수 있으니 아무 곳에나 넣어도 됨.
  - Project container 안에서 ./submit.sh xxxx-xxxxx 실행

- Archive 의 파일 이름 확인
  - project3_학번.zip (학번 format은 20xx-xxxxx)

Notice

수업 게시판 확인

- eTL 공지 및 질문 게시판 항상 확인할 것
- 스펙이 수정 또는 추가되는 사항은 항상 게시판을 통하여 공지
- 제출 마지막날까지 공지된 사항을 반영해서 제출

제출 형식 지키기 (파일 이름 및 출력)

- **지키지 않을 시 감점**

Cheating 금지 (F처리, 모든 코드 철저히 검사)

TA

- 이상현
- E-mail : sanghyeon@snu.ac.kr