컴파일러의 기초 Project 3

전기정보공학부 이상민(2016-17069) - functions.c 구현, subc.y 구현, 보고서 작성

전기정보공학부 박찬정(2017-17088) – check.c 구현, subc.y 구현, 보고서 작성

수업시간에 배운 내용을 기반으로 해서 scoped symbol table을 만들었다. scope를 관리하는 scope stack element을 만들고 이들이 ste를 관리하도록 하였다.

ppt에 나열된 semantic check를 기준으로 구현을 설명하고자 한다.

1. undeclared variable & function의 경우,   
ID가 unary로 reduce 될 때, 전체 scope에서 해당 ID로 같은 ste가 있는지 찾아본다. 같은 ID가 있다면 전에 선언된 것을 의미하지만 없다면 선언되지 않은 경우이며, 지금 선언되지 않고 쓰이는 상황이므로 에러메세지 “not declared”를 출력한다. 찾은 경우, 해당하는 ste의 decl을 unary가 declptr이므로 unary가 같은 decl을 가지도록 한다. 함수의 선언의 경우 ID가 unary로 reduce 되고 나서 unary '(' args ')' 가 unary로 reduce 됨으로 unary: ID 문법에서 마찬가지로 확인된다.

2. Redeclaration

declare 관련된 문법들을 살펴 보아야 한다. 선언하는 문법의 경우, 재선언 됐는지를 확인하는 과정을 거친다.

ext\_def : type\_specifier pointers ID ';' 에서 type\_specifier 와 ID가 값이 존재하는 경우(둘다 NULL)이 아닌 경우, ID로 전체scope(ext\_def이므로)를 검색해서 값이 있는 경우 에러메세지 redeclaration을 출력한다. 만약 find되지않는 경우 새로 decl을 생성해서 declare해주는데 이때 포인터가 있는지 없는지에 따라 vardecl을 만드거나 ptrdecl을 만든다.

ext\_def : type\_specifier pointers ID '[' const\_expr ']' ';' 의 경우도 위와 동일하다.

struct\_specifier: STRUCT ID '{' def\_list '}'

의 경우는 STRUCT ID '{' 와 def\_list '}' 가운데 sematic action으로 field를 모으기 위해 가짜 스코프를 push 하고 find를 했을 때 만약 찾아진다면 이미 선언된 것이므로 에러메세지 redeclartion을 출력한다. 해당문법 끝의 semantic action으로는 가짜 스코프를 pop해서 ste\* fields가 필드들을 가르키도록 하고 선언된 것이 아니면 structdecl을 만들어서 declare 한다.

func\_decl: type\_specifier pointers ID '(' ')'이 경우에 현재 스코프를 find해서 있는 경우, 에러메세지 redeclartion을 출력한다. 없는 경우에 procdecl을 생성해서 declare한다.

func\_decl: type\_specifier pointers ID '(' param\_list ')'

의 경우도 위와 마찬가지이며, formals들을 모으기 위해서 중간 semantic action으로 가짜스코프를 push하고 이를 pop한다. 수업 ppt 34page 참조.

param\_decl : type\_specifier pointers ID

param\_decl : type\_specifier pointers ID '[' const\_expr ']' 파라미터의 declaration 또한 마찬가지이다. 이에 대한 내용은 위의 내용과 크게 다르지 않다.

def : type\_specifier pointers ID ';'

def : type\_specifier pointers ID '[' const\_expr ']' ';' 의 경우에도 맨 첫번째의 ext\_def와 다르지 않으나 ext\_def는 전체 스코프를 찾았던 것에 반해 그냥 def의 경우는 현재 스코프만 찾는다.

3. type checking 의 경우

ppt에 있는 3가지 순서대로 assignment operation의 type check를 하였으며,

expr : unary '=' expr

위의 문법에서 그 순서에 맞게 타입 체크를 해주었다. 해당 nonterminal 이 가르키고 있는 decl의 declclass를 보고 판단을 해주었다. 그런 기능을 편하게 하기 위해서 각종 체크함수들을 만들었다.   
  
4. structure & structure pointer declaration 의 경우

struct\_specifier: STRUCT ID 문법에서 ID에 해당하는 declaration이 존재하는지 확인한 뒤, 존재하지 않는 경우 해당 struct가 정의되지 않은 상태에서 접근하는 경우이므로 incomplete type 에러를 내보낸다.

5. function declaration 의 경우

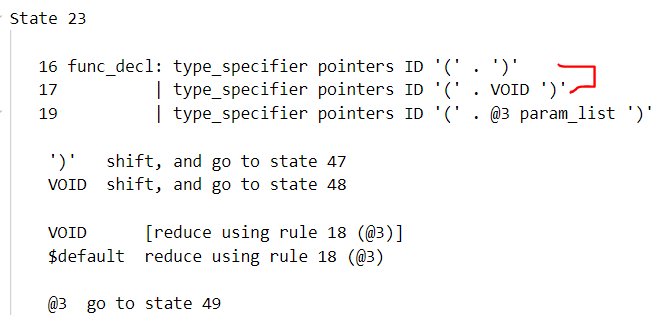
function의 return type 확인은 RETURN 토큰이 등장했을 때 체크하도록 하였다. 해당 scope에서 returnid를 확인하여 return하려는 type과 동일한지 확인하여 그렇지 않다면 incompatible return types 에러를 내보낸다. 또한 unary: unary ‘(‘ args ‘)’ 문법과 unary: unary ‘(‘ ‘)’ 문법에서는 unary의 type을 확인하고 function type이 맞는지 확인한다. 아닌 경우 not a function 에러를 내보낸다. function type이 맞은 경우, formal을 하나하나 확인하여 개수와 타입이 맞는지 확인한다. 틀린 경우 actual args ar not equal to formal args 에러를 내보낸다.

문법 conflict

make를 했을 경우 shift/reduce conflict가 나는 것을 확인하였다.







이는 해당 문법에서 필연적으로 발생하는 conflict라고 생각된다.  
그 이유는 VOID가 empty로 reduce 돼서 결국 16번 rule로 reduce 될 것인지 아니면 VOID를 shift해서 나중에 17번 rule로 reduce 될 것인지에 대한 conflict라고 볼 수 있다.

새로운 문법 설정

compound\_stmt의 경우 일반적으로 새로운 scope를 사용하는 것이 맞지만, 현재 문법에서는 이를 함수 정의나 if, for, while등에서 동일한 문법으로 사용하고 있기 때문에 이 경우 scope를 새로 열지 않아 처리가 곤란하다. 그래서 non\_pushing\_compound\_stmt 와 non\_pushing\_stmt nonterminal을 새로 만들었다. 기존의 stmt 문법에서 if, for, while문은 stmt가 아닌 non\_pushing\_stmt를 사용하게 되며, non\_pushing\_stmt는 compound\_stmt가 아닌 non\_pushing\_compound\_stmt를 사용한다. compound\_stmt는 새로운 스코프를 push하지만 non\_pushing\_compound\_stmt는 새로운 scope를 push하지 않는다.

Binary 연산 관련 midrule 설정

Binary OP binary 연산의 경우, 두 binary nonterminal 모두 에러가 발생하면 한 줄에 대해 여러 에러가 발생하게 된다. 따라서 midrule을 삽입하여, 잠시 error 출력을 멈추도록 설정하였다.

fuctions.c 함수 설명

void declare(id\* name, decl\* decl);

주어진 name 과 decl로 새로운 ste를 만들고

스택의 맨 위에 오도록 stetop으로 설정한다.(push)

void push\_scope();

새로운 sse(stack scope element)를 만들고

이 새로운 sse의 stetop이 기존의 ssetop의 stetop 을 가르키도록하고

이sse를 sse스택 맨위인 ssetop에 오도록 한다.(push)

ste\* pop\_scope();

현재 스코프를 팝할 것이며 현재 스코프에 해당하는 ste들을 pop한다. 즉,

현재 스코프의 ssetop의 stetop이 가르키고 있는 ste부터 ssetop의 prev의stetop이 가르키고 있는 ste(제외, 바로 위의 ste)까지 pop할 것이다. return하는 경우 기존에 스코프에서 가장 위에 있는 ste가 순서가 가장 밑으로 가게 되며 이렇게 순서를 뒤집어서 가장 위의 ste 포인터를 리턴한다.

void push\_stelist(ste\* stelist);

ste들이 연결된 stelist를 인풋으로 받게 되면, 이를 하나하나 위에서부터 차례대로 현재 스택의 맨위에 차례대로 push해서 올린다.

void declare\_struct(id\* name, decl\* decl);

struct의 경우 항상 global로 관리해야하므로 전역변수인globalste로 스택을 관리하며, 인풋으로 주어진 name 과 decl로 새로운 ste를 만들고 이를 제일 밑으로 오게 넣어준다. globalste는 가장 밑의 ste를 포인팅하고 있다.

ste\* findste(id\* name);

주어진 인풋인 name으로 ste를 찾는다 ssetop->stetop에서부터 계속해서 prev로 이동해가면서 같은 이름을 가진 ste가 있으면 그 ste를 return하고 없으면 NULL을 리턴한다.

ste\* findste\_currentscope(id\* name);

주어진 인풋인 name으로 ste를 찾는데 현재 스코프에서만 찾는다.

ste\* findste\_field(ste\* field, id\* name);

주어진 field부터 시작해서 아래로 검색. 같은 이름을 가진 ste가 있으면 그 ste 를 return하고 없으면 NULL을 리턴한다.

decl\* finddecl(id\* name);

주어진 인풋인 name으로 ste를 찾는다 ssetop->stetop에서부터 계속해서 prev로 이동해가면서 같은 이름을 가진 ste가 있으면 그 ste->decl를 return하고 없으면 NULL을 리턴한다.

decl\* finddecl\_currentscope(id\* name);

주어진 인풋인 name으로 ste를 찾는데 현재 스코프에서만 찾는다. 같은 이름을 가진 ste가 있으면 그 ste->decl를 return하고 없으면 NULL을 리턴한다.

decl\* finddecl\_field(id\* name);

주어진 field부터 시작해서 아래로 검색.

같은 이름을 가진 ste가 있으면 그 ste->decl를 return하고 없으면 NULL을 리턴한다.

decl 만드는 함수들.

decl\* maketypedecl(int typeclass);

인풋으로 받은 typeclass로 typedecl을 만든다.

decl\* makevardecl(decl\* typedecl);

typedecl 을 입력으로 받아서 이를 decl의 두번째 변수인 decl\* type이 가르키게 하고

decl의 첫번째 변수인 declclass가 var인 vardecl을 만든다.

decl\* makeconstdecl(decl\* typedecl);

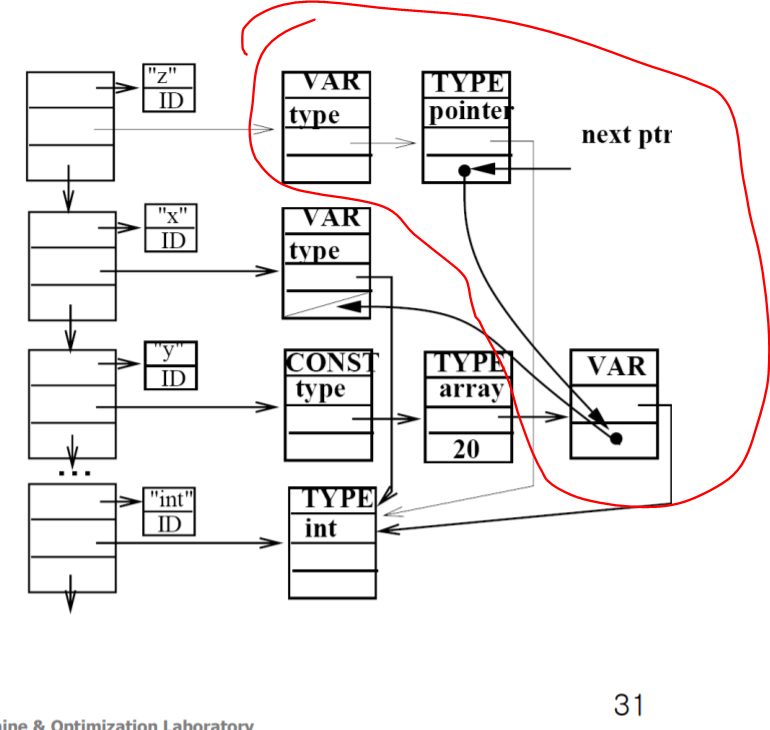
typedecl 을 입력으로 받아서 이를 decl의 두번째 변수인 decl\* type이 가르키게 하고

decl의 첫번째 변수인 declclass가 const인 constdecl을 만든다.

decl\* makeptrdecl(decl\* ptrtodecl);

ptrtodecl 을 입력으로 받아서 이를 decl의 변수인 decl\* ptrto가 가르키게 하고

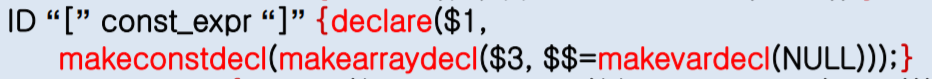
decl의 declclass가 TYPE이고 decl의 typeclass가 POINTER인 ptrdecl을 만든다. 단 주의할 점은 decl을 만드는 경우에 declare($\*,makevardecl($$=makeptrdecl(NULL)))처럼 vardecl도 같이 만들어 줘야한다. ppt30, ppt31 참고.  

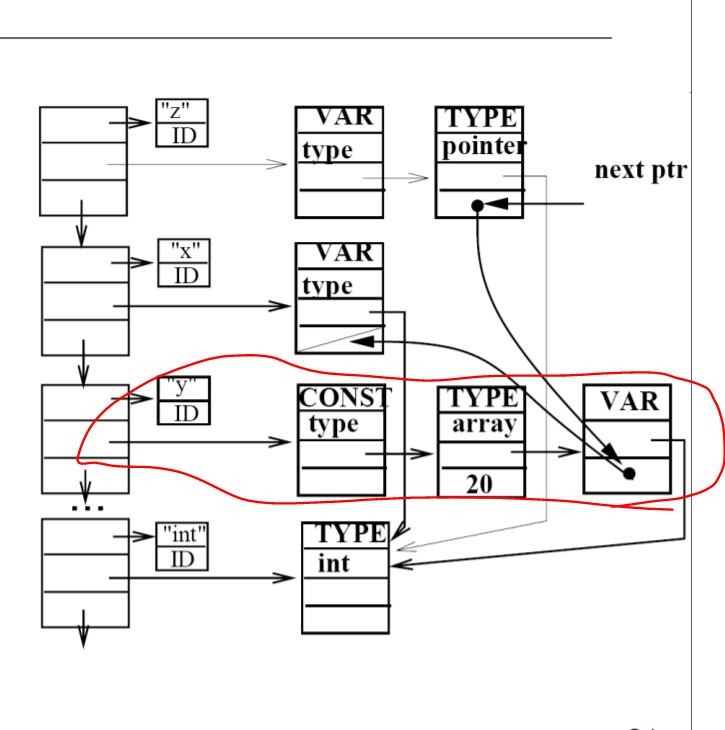



decl\* makearraydecl(decl\* const\_expr\_decl, decl\* vardecl);

이 경우, ppt30의 사용용법과 같이 첫번째 인자로 const\_expr\_decl을 받아서 이 값이 가지고 있는 int\_value 값을 element 개수를 알려주는 num\_index 변수로 넣어준다.

그리고 두번째 인자인 vardecl의 값으로 decl의 elementvar가 vardecl을 가르키게 한다. 그리고 사용할 시에 밑의 그림과 같이 constdecl을 또 만들어줘야한다.





decl\* makestructdecl(ste\* fields);

defclass가 TYPE이고 typeclass가 STRUCT이고 fielist가 인풋인 fields인 decl을 만들고 이를 return한다.

decl\* makeprocdecl();

defclass가 FUNC인 decl을 만들고 이를 return 한다.

check.c 함수 설명

int check\_compatible\_type(decl\* dec1, decl\* dec2)

dec1과 dec2의 type을 비교한다. 같은 type이거나, pointer이면서 ptrto의 type이 같거나, array이면서 element의 type이 같으면 1을 반환한다.

int check\_is\_var(decl\* dec1)

dec1이 CONST인 int 혹은 char인지 확인한다.

int check\_is\_expr(decl\* dec1)

dec1이 expression인지 확인한다.

int check\_is\_pointer(decl\* dec1)

dec1이 pointer인지 확인한다.

int check\_is\_array (decl\* dec1)

dec1이 array인지 확인한다.

int check\_is\_struct (decl\* dec1)

dec1이 struct인지 확인한다.

int check\_is\_struct\_type (decl\* dec1)

dec1의 type이 struct인지 확인한다.