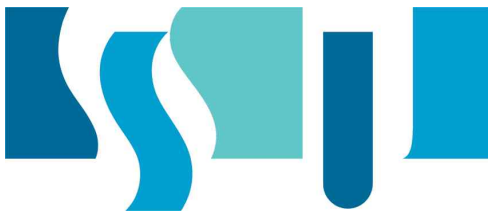


# 컴파일러 과제 4

| | |
|---|---|
| **과목명** | 컴파일러 |
| **교수명** | 유재우 교수님 |
| **학 과** | IT대학 컴퓨터학부 |
| **학 번** | 20172655 |
| **이 름** | 이강산 |
| **제출일** | 2021.11.12. |

## < 목차 >

# 1. 신택스 분석기 구현

## 1) 구현 개요

신택스 분석기는 원시프로그램이 문법적으로 올바른지 분석하여 신택스 트리를 생성하는 일을 한다. 이 과정에서 현재 토큰으로 넘겨받은 명칭 등을 사용할 수 있는지 스코프 규칙에 따라 판단하며, 기초적 수준의 시멘틱 분석을 동시에 진행한다.

자세한 과정은 다음과 같다.

- 원시프로그램 중에서 선언되는 모든 명칭을 저장하고 중복 선언되는지를 검사한다.
- 명칭의 종류와 타입 정보 등을 계산하고 연결한다.
- 함수 선언문에서 선언자의 형태, 리턴 타입의 종류 및 파라미터 선언들이 올바르게 되어 있는지 검사한다.
- 전방 참조를 포함하여 선언하지 않고 사용한 명칭이 있는 지 검사한다.
- 원시 프로그램 중에 명칭이 올바른 종류로 참조되고 사용되는지 검사한다.

## 2) 주요 기능 설명

### 2-1) lex 명세

| 1. 토큰 | |
|---|---|
| 예약어 | auto break case continue default do else enum for if return sizeof struct switch typedef union while int float char void |
| 특수기호 | ~ << >> ^ \| ? ++ -- -> < > <= >= == != && \|\| ... ( ) [ ] { } : . , ! * / % & ; + - = |
| 상수 | 정수형 상수, 실수형 상수, 문자형 상수, 스트링 리터럴 |
| 명칭 | identifier |
| lex 명세서에 작성된 예약어, 특수기호, 상수, 리터럴은 다음과 같다. 위 어휘들을 토큰으로 사용하는 신택스 분석기를 구현한다. | |

2. yylval의 타입 재지정

```
typedef union{
        char cval;
        int ival;
        char* pval;
        A_NODE* anode;
        A_TYPE* atype;
        A_SPECIFIER* aspec;
        A_ID* aid;
        S_KIND skind;
        T_KIND tkind;
} YYSTYPE;
```

yylval은 현재 토큰들의 값을 저장하는 변수로 사용된다. 기본 타입은 정수형이지만, 모든 토큰이 정수형은 아니므로, 여러 타입에 대응할 수 있도록 yylval 변수의 타입인 YYSTYPE 을 union{}으로 재지정했다. 수정된 yylval 변수는 정수 이외에도 문자, 스트링 등 다양한 타입의 토큰에 대응할 수 있다.

3. makeString 함수

```
char *makeString(char *s) {
        char *t;
        t=malloc(strlen(s)+1);
        strcpy(t, s);
        return(t);
}
```

yytext에 저장된 스트링을 꼭 맞는 메모리에 저장한 후 리턴한다.

4. checkIdentifier 함수

```
int checkIdentifier(char *s) {
        //printf("checkIdentifier called %s\n", s);
        A_ID *id=0;
        char *t;
        id=current_id;
        while(id) {
                if(strcmp(id->name, s)==0)
                        break;
                id=id->prev;
        }
        if(id==0) {
                yylval.pval=makeString(s);
                return(IDENTIFIER);
        }
        else if(id->kind==ID_TYPE) {
                yylval.atype=id->type;
                return(TYPE_IDENTIFIER);
        }
        else {
                yylval.pval=id->name;
                return(IDENTIFIER);
        }
}
```

현재 토큰이 스트링일 때, 일반  명칭인지 앞서 선언된 타입명인지 구분한다. typedef 로 선언된 이름은 TYPE_IDENTIFIER 로 리턴하고 (이때 yylval 값은 그 타입 테이블 주소), 일반 이름은 IDENTIFIER 로 리턴한다. (이때 yylval 은 이름의 문자열)

- lex 명세서 전문은 3-1에 있습니다.

## 2-2) yacc 명세

| 1. 보조함수 프로토타입 선언부 |
| --- |
| A_NODE* makeNode(NODE_NAME, A_NODE*, A_NODE*, A_NODE*);<br>A_NODE* makeNodeList(NODE_NAME, A_NODE*, A_NODE*);<br>A_ID* makeIdentifier(char*);<br>A_ID* makeDummyIdentifier();<br>A_TYPE* makeType(T_KIND);<br>A_SPECIFIER* makeSpecifier(A_TYPE*, S_KIND);<br>A_ID* searchIdentifier(char*, A_ID*);<br>A_ID* searchIdentifierAtCurrentLevel(char*, A_ID*);<br>A_SPECIFIER* updateSpecifier(A_SPECIFIER*, A_TYPE*, S_KIND);<br>void checkForwardReference();<br>void setDefaultSpecifier(A_SPECIFIER*);<br>A_ID* linkDeclaratorList(A_ID*, A_ID*);<br>A_ID* getIdentifierDeclared(char*);<br>A_TYPE* getTypeOfStructOrEnumRefIdentifier(T_KIND, char*, ID_KIND);<br>A_ID* setDeclaratorInit(A_ID*, A_NODE*);<br>A_ID* setDeclaratorKind(A_ID*, ID_KIND);<br>A_ID* setDeclaratorType(A_ID*, A_TYPE*);<br>A_ID* setDeclaratorElementType(A_ID*, A_TYPE*);<br>A_ID* setDeclaratorTypeAndKind(A_ID*, A_TYPE*, ID_KIND);<br>A_ID* setDeclaratorListSpecifier(A_ID*, A_SPECIFIER*);<br>A_ID* setFunctionDeclaratorSpecifier(A_ID*, A_SPECIFIER*);<br>A_ID* setFunctionDeclaratorBody(A_ID*, A_NODE*);<br>A_ID* setParameterDeclaratorSpecifier(A_ID*, A_SPECIFIER*);<br>A_ID* setStructDeclaratorListSpecifier(A_ID*, A_TYPE*);<br>A_TYPE* setTypeNameSpecifier(A_TYPE*, A_SPECIFIER*);<br>A_TYPE* setTypeElementType(A_TYPE*, A_TYPE*);<br>A_TYPE* setTypeField(A_TYPE*, A_ID*);<br>A_TYPE* setTypeExpr(A_TYPE*, A_NODE*);<br>A_TYPE* setTypeAndKindOfDeclarator(A_TYPE*, ID_KIND, A_ID*);<br>A_TYPE* setTypeStructOrEnumIdentifier(T_KIND, char*, ID_KIND);<br>BOOLEAN isNotSameFormalParameters(A_ID*, A_ID*);<br>BOOLEAN isNotSameType(A_TYPE*, A_TYPE*);<br>BOOLEAN isPointerOrArrayType(A_TYPE*); |
| 입력된 원시 프로그램을 분석하며 필요한 경우 호출되는 보조함수들이다. 제일 기본이 되는 함수들은 makeNode로 이름과 자식이 될 노드의 주소 3개를 받아 새로운 노드를 생성한다. 새로운 명칭 입력 시 searchIdentifier, searchIdentifierAtCurrentLevel 등의 함수를 통해 중복 여부를 판단한다 |

2. 신택스 트리 출력 함수 프로토타입 선언부

```
void print_ast(A_NODE *);
void prt_program(A_NODE *, int);
void prt_initializer(A_NODE *, int);
void prt_arg_expr_list(A_NODE *, int);
void prt_statement(A_NODE *, int);
void prt_statement_list(A_NODE *, int);
void prt_for_expression(A_NODE *, int);
void prt_expression(A_NODE *, int);
void prt_A_TYPE(A_TYPE *, int);
void prt_A_ID_LIST(A_ID *, int);
void prt_A_ID(A_ID *, int);
void prt_A_ID_NAME(A_ID *, int);
void prt_STRING(char *, int);
void prt_integer(int, int);
void print_node(A_NODE *,int);
void print_space(int);
```

신택스 분석 완료 후 생성된 신택스 트리를 출력하기 위한 함수들이다. print_ast 함수를 통해 호출할 수 있으며, 올바르게 생성되지 못한 경우 중도에 출력을 중단한다.

3. syntax_error, initialize, main 함수

```
void syntax_error(int, char*);
void initialize();
int main() {
        initialize();
        yyparse();
        print_ast(root);
        return 0;
}
```

syntax_error 함수는 보조함수 내부에서 잘못된 문법이 인식되었을 때 호출된다. 선언되지 않은 명칭 호출 등 13여 가지 오류 상황에 대해 switch-case 문으로 대응하여 출력한다. initialize 함수는 원시 프로그램을 입력 받기 전, int, float, char, void 와 같은 기본적인 타입이나 printf, scanf 등의 기본 함수들에 대한 테이블을 미리 생성한다. 따라서 main 함수의 흐름은 위와 같다.

- yacc 명세서 전문은 3-2에 있습니다.

3) 원시 프로그램 입력 결과

1.c

```
#include<stdio.h>
int a=10;
int b;
int main() {
        float b=2.0;
        char c1='c';
        return 0;
}
```

```
san@linux:~/21-2/Compiler/assign4$ ./a.out < 1.c
=======  syntax tree  ==========
N_PROGRAM (0,0)
|  (ID="a") TYPE:0x5638450a32f0 KIND:VAR SPEC=AUTO LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  (int)
|  |  INIT
|  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  10
|  (ID="b") TYPE:0x5638450a32f0 KIND:VAR SPEC=AUTO LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  (int)
|  (ID="main") TYPE:0x5638450a8b20 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  |  PARAMETER
|  |  |  |  TYPE
|  |  |  |  |  (int)
|  |  |  |  BODY
|  |  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  |  (ID="b") TYPE:0x5638450a3380 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  |  (float)
|  |  |  |  |  |  |  INIT
|  |  |  |  |  |  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_FLOAT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  2.0
|  |  |  |  |  |  (ID="c1") TYPE:0x5638450a3410 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  |  (char 1)
|  |  |  |  |  |  |  INIT
|  |  |  |  |  |  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_CHAR_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  99
|  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  N_STMT_RETURN (0,0)
|  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  0
|  |  |  |  |  |  |  N_STMT_LIST_NIL (0,0)
```

전역 및 지역변수의 선언문, 배정문 예시

2.c

```
struct my_s{
        int b[3];
        float c;
}str;
typedef union{ int b; float c; }my_union;
```

```
san@linux:~/21-2/Compiler/assign4$ ./a.out < 2.c
=======  syntax tree  ==========
N_PROGRAM (0,0)
|  (ID="str") TYPE:0x55e192e88960 KIND:VAR SPEC=AUTO LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  STRUCT
|  |  |  |  FIELD
|  |  |  |  |  (ID="b") TYPE:0x55e192e88a50 KIND:FIELD SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  ARRAY
|  |  |  |  |  |  |  |  INDEX
|  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  3
(none)
|  |  |  |  |  |  |  |  ELEMENT_TYPE
|  |  |  |  |  |  |  |  |  (int)
|  |  |  |  |  (ID="c") TYPE:0x55e192e83380 KIND:FIELD SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  (float)
|  (ID="my_union") TYPE:0x55e192e88b90 KIND:TYPE SPEC=TYPEDEF LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  UNION
|  |  |  |  FIELD
|  |  |  |  |  (ID="b") TYPE:0x55e192e832f0 KIND:FIELD SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  (int)
|  |  |  |  |  (ID="c") TYPE:0x55e192e83380 KIND:FIELD SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  (float)
san@linux:~/21-2/Compiler/assign4$
```

구조체, 공용체 정의, 변수 선언 예시

3.c

```
#include<stdio.h>
enum color { white, red=10, green=11, blue, black } c1;
int main() {
        c1=white;
        return 0;
}
```

```
san@linux:~/21-2/Compiler/assign4$ ./a.out < 3.c
=======  syntax tree  ==========
N_PROGRAM (0,0)
|  (ID="c1") TYPE:0x55cf5f785960 KIND:VAR SPEC=AUTO LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  ENUM
|  |  |  |  ENUMERATORS
|  |  |  |  |  (ID="white") TYPE:(nil) KIND:ENUM_LITERAL SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  |  |  |  (ID="red") TYPE:(nil) KIND:ENUM_LITERAL SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  |  |  |  |  INIT
|  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  10
|  |  |  |  |  (ID="green") TYPE:(nil) KIND:ENUM_LITERAL SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  |  |  |  |  INIT
|  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  11
|  |  |  |  |  (ID="blue") TYPE:(nil) KIND:ENUM_LITERAL SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  |  |  |  (ID="black") TYPE:(nil) KIND:ENUM_LITERAL SPEC=NULL LEV=0 VAL=0 ADDR=0
|  (ID="main") TYPE:0x55cf5f785d70 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  |  PARAMETER
|  |  |  |  TYPE
|  |  |  |  |  (int)
|  |  |  |  BODY
|  |  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  N_STMT_EXPRESSION (0,0)
|  |  |  |  |  |  |  |  N_EXP_ASSIGN (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="c1") TYPE:0x55cf5f785960 KIND:VAR SPEC=AUTO LEV=0 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="white") TYPE:(nil) KIND:ENUM_LITERAL SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  N_STMT_RETURN (0,0)
|  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  0
|  |  |  |  |  |  N_STMT_LIST_NIL (0,0)
san@linux:~/21-2/Compiler/assign4$
```

열거형 정의, 변수 선언 예시

4.c

```
#include<stdio.h>
int main() {
        int *a;
        float **b;
        return 0;
}
```

```
san@linux:~/21-2/Compiler/assign4$ ./a.out < 4.c
=======  syntax tree  ==========
N_PROGRAM (0,0)
|  (ID="main") TYPE:0x556f42e3d980 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  |  PARAMETER
|  |  |  |  TYPE
|  |  |  |  |  (int)
|  |  |  |  BODY
|  |  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  |  (ID="a") TYPE:0x556f42e3da00 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  |  POINTER
|  |  |  |  |  |  |  |  |  ELEMENT_TYPE
|  |  |  |  |  |  |  |  |  |  (int)
|  |  |  |  |  |  |  (ID="b") TYPE:0x556f42e3dad0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  |  |  POINTER
|  |  |  |  |  |  |  |  |  |  ELEMENT_TYPE
|  |  |  |  |  |  |  |  |  |  |  POINTER
|  |  |  |  |  |  |  |  |  |  |  |  ELEMENT_TYPE
|  |  |  |  |  |  |  |  |  |  |  |  |  (float)
|  |  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  |  N_STMT_RETURN (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  0
|  |  |  |  |  |  |  N_STMT_LIST_NIL (0,0)
san@linux:~/21-2/Compiler/assign4$
```

포인터 변수의 선언 예시

5.c

```
#include<stdio.h>
void func1(char, int);
int main() {
        func1('a', 3);
        return 0;
}
void func1(char p1, int p2) { ; }
```

```
san@linux:~/21-2/Compiler/assign4$ ./a.out < 5.c
=======  syntax tree  ==========
N_PROGRAM (0,0)
|  (ID="func1") TYPE:0x560d6c3f5a60 KIND:FUNC SPEC=AUTO LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  |  PARAMETER
|  |  |  |  |  (ID="") TYPE:0x560d6c3f0410 KIND:PARM SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  TYPE
|  |  |  |  |  |  (char 1)
|  |  |  |  |  (ID="") TYPE:0x560d6c3f02f0 KIND:PARM SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  TYPE
|  |  |  |  |  |  (int)
|  |  |  |  TYPE
|  |  |  |  |  (void)|  (ID="main") TYPE:0x560d6c3f5b30 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  |  PARAMETER
|  |  |  |  TYPE
|  |  |  |  |  (int)
|  |  |  |  BODY
|  |  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  N_STMT_EXPRESSION (0,0)
|  |  |  |  |  |  |  |  N_EXP_FUNCTION_CALL (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="func1") TYPE:0x560d6c3f5a60 KIND:FUNC SPEC=AUTO LEV=0 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  N_ARG_LIST (0,0)
|  |  |  |  |  |  |  |  |  |  N_EXP_CHAR_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  97
|  |  |  |  |  |  |  |  |  |  N_ARG_LIST (0,0)
|  |  |  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  3
|  |  |  |  |  |  |  |  |  |  N_ARG_LIST_NIL (0,0)
|  |  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  |  N_STMT_RETURN (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  0
|  |  |  |  |  |  |  |  N_STMT_LIST_NIL (0,0)
|  (ID="func1") TYPE:0x560d6c3f6080 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  |  PARAMETER
|  |  |  |  |  (ID="p1") TYPE:0x560d6c3f0410 KIND:PARM SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  TYPE
|  |  |  |  |  |  (char 1)
|  |  |  |  |  (ID="p2") TYPE:0x560d6c3f02f0 KIND:PARM SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  TYPE
|  |  |  |  |  |  (int)
|  |  |  |  TYPE
|  |  |  |  |  (void)|  |  |  BODY
|  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  N_STMT_EMPTY (0,0)
|  |  |  |  |  N_STMT_LIST_NIL (0,0)
san@linux:~/21-2/Compiler/assign4$
```

함수 선언부, 함수 호출, 함수 정의부 예시

6.c

```
#include<stdio.h>
int main() {
        int a=3;
        char value;
        if(a>5) value='A';
        else value='a';
        return 0;
}
```

```
san@linux:~/21-2/Compiler/assign4$ ./a.out < 6.c
=======  syntax tree  ==========
N_PROGRAM (0,0)
|  (ID="main") TYPE:0x55d67726e980 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  PARAMETER
|  |  |  TYPE
|  |  |  |  (int)
|  |  |  BODY
|  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  (ID="a") TYPE:0x55d6772692f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  (int)
|  |  |  |  |  |  INIT
|  |  |  |  |  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  3
|  |  |  |  |  (ID="value") TYPE:0x55d677269410 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  (char 1)
|  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  N_STMT_IF_ELSE (0,0)
|  |  |  |  |  |  |  N_EXP_GTR (0,0)
|  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  (ID="a") TYPE:0x55d6772692f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  5
|  |  |  |  |  |  |  N_STMT_EXPRESSION (0,0)
|  |  |  |  |  |  |  |  N_EXP_ASSIGN (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="value") TYPE:0x55d677269410 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  N_EXP_CHAR_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  65
|  |  |  |  |  |  |  N_STMT_EXPRESSION (0,0)
|  |  |  |  |  |  |  |  N_EXP_ASSIGN (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="value") TYPE:0x55d677269410 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  N_EXP_CHAR_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  97
|  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  N_STMT_RETURN (0,0)
|  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  0
|  |  |  |  |  |  N_STMT_LIST_NIL (0,0)
san@linux:~/21-2/Compiler/assign4$
```

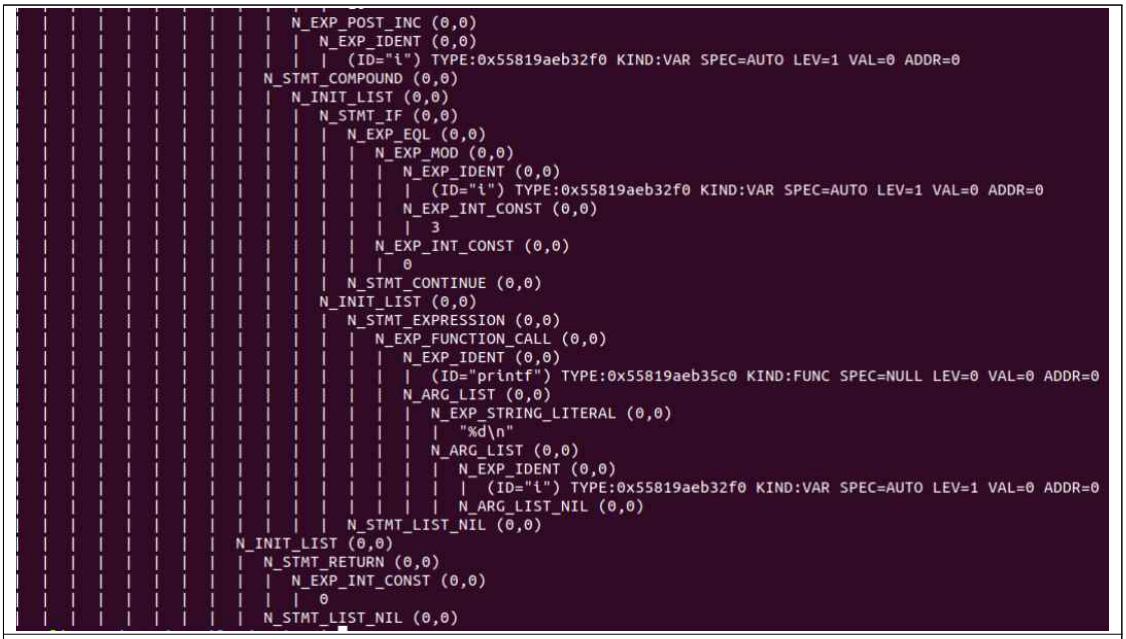
조건문(if) 예시

7.c

```
#include<stdio.h>
int main() {
        int a=3, i;
        while(a>0) {
                a++;
```

```
                if(a==25) break;
        }
        for(i=0; i<10; i++) {
                if(i%3==0) continue;
                printf("%d\n", i);
        }
        return 0;
}
```

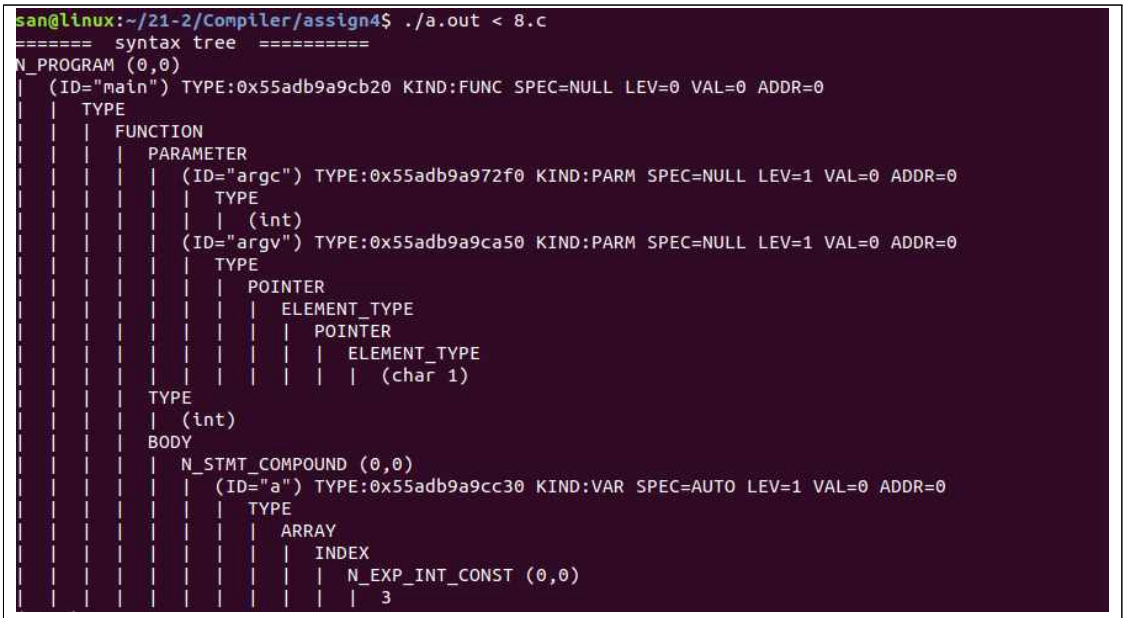
```
san@linux:~/21-2/Compiler/assign4$ ./a.out < 7.c
=======  syntax tree  ==========
N_PROGRAM (0,0)
|  (ID="main") TYPE:0x55819aeb8980 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  |  PARAMETER
|  |  |  |  TYPE
|  |  |  |  |  (int)
|  |  |  |  BODY
|  |  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  |  (ID="a") TYPE:0x55819aeb32f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  |  (int)
|  |  |  |  |  |  |  INIT
|  |  |  |  |  |  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  3
|  |  |  |  |  |  (ID="i") TYPE:0x55819aeb32f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  |  (int)
|  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  N_STMT_WHILE (0,0)
|  |  |  |  |  |  |  |  N_EXP_GTR (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="a") TYPE:0x55819aeb32f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  0
|  |  |  |  |  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  |  |  |  N_STMT_EXPRESSION (0,0)
|  |  |  |  |  |  |  |  |  |  |  N_EXP_POST_INC (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  |  (ID="a") TYPE:0x55819aeb32f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  |  |  |  N_STMT_IF (0,0)
|  |  |  |  |  |  |  |  |  |  |  N_EXP_EQL (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  |  (ID="a") TYPE:0x55819aeb32f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  |  25
|  |  |  |  |  |  |  |  |  |  |  N_STMT_BREAK (0,0)
|  |  |  |  |  |  |  |  |  |  N_STMT_LIST_NIL (0,0)
|  |  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  |  |  N_STMT_FOR (0,0)
|  |  |  |  |  |  |  |  |  N_FOR_EXP (0,0)
|  |  |  |  |  |  |  |  |  |  N_EXP_ASSIGN (0,0)
|  |  |  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  (ID="i") TYPE:0x55819aeb32f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  0
|  |  |  |  |  |  |  |  |  |  N_EXP_LSS (0,0)
|  |  |  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  (ID="i") TYPE:0x55819aeb32f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  10
```




단항연산 ++, 다항연산 %, 비교연산 <, ==, for문, while문 예시

8.c

```
#include <stdio.h>
int main(int argc, char** argv) {
	int a[3]={1,2,3};
	int i=3;
	for(i; i<10; i++)
		printf("HELLO\n");
}
```



```
san@linux:~/21-2/Compiler/assign4$ ./a.out < 8.c
=======  syntax tree  ==========
N_PROGRAM (0,0)
|  (ID="main") TYPE:0x55adb9a9cb20 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  TYPE
|  |  |  FUNCTION
|  |  |  |  PARAMETER
|  |  |  |  |  (ID="argc") TYPE:0x55adb9a972f0 KIND:PARM SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  (int)
|  |  |  |  |  (ID="argv") TYPE:0x55adb9a9ca50 KIND:PARM SPEC=NULL LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  POINTER
|  |  |  |  |  |  |  |  ELEMENT_TYPE
|  |  |  |  |  |  |  |  |  POINTER
|  |  |  |  |  |  |  |  |  |  ELEMENT_TYPE
|  |  |  |  |  |  |  |  |  |  |  |  (char 1)
|  |  |  |  TYPE
|  |  |  |  |  (int)
|  |  |  |  BODY
|  |  |  |  |  N_STMT_COMPOUND (0,0)
|  |  |  |  |  |  (ID="a") TYPE:0x55adb9a9cc30 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  |  ARRAY
|  |  |  |  |  |  |  |  |  INDEX
|  |  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  3
```

```
|  |  |  |  |  |  |  |  |  |  ELEMENT_TYPE
|  |  |  |  |  |  |  |  |  |  |  (int)
|  |  |  |  |  |  |  INIT
|  |  |  |  |  |  |  |  N_INIT_LIST_NIL (0,0)
|  |  |  |  |  |  |  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  1
|  |  |  |  |  |  |  |  |  N_INIT_LIST_NIL (0,0)
|  |  |  |  |  |  |  |  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  2
|  |  |  |  |  |  |  |  |  N_INIT_LIST_NIL (0,0)
|  |  |  |  |  |  |  |  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  |  |  3
|  |  |  |  |  |  |  |  |  |  (null) (0,0)
|  |  |  |  |  |  (ID="i") TYPE:0x55adb9a972f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  TYPE
|  |  |  |  |  |  |  |  (int)
|  |  |  |  |  |  |  INIT
|  |  |  |  |  |  |  |  N_INIT_LIST_ONE (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  3
|  |  |  |  |  |  N_INIT_LIST (0,0)
|  |  |  |  |  |  N_STMT_FOR (0,0)
|  |  |  |  |  |  |  N_FOR_EXP (0,0)
|  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  (ID="i") TYPE:0x55adb9a972f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  N_EXP_LSS (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="i") TYPE:0x55adb9a972f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  N_EXP_INT_CONST (0,0)
|  |  |  |  |  |  |  |  |  |  10
|  |  |  |  |  |  |  |  N_EXP_POST_INC (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="i") TYPE:0x55adb9a972f0 KIND:VAR SPEC=AUTO LEV=1 VAL=0 ADDR=0
|  |  |  |  |  |  |  N_STMT_EXPRESSION (0,0)
|  |  |  |  |  |  |  |  N_EXP_FUNCTION_CALL (0,0)
|  |  |  |  |  |  |  |  |  N_EXP_IDENT (0,0)
|  |  |  |  |  |  |  |  |  |  (ID="printf") TYPE:0x55adb9a975c0 KIND:FUNC SPEC=NULL LEV=0 VAL=0 ADDR=0
|  |  |  |  |  |  |  |  |  N_ARG_LIST (0,0)
|  |  |  |  |  |  |  |  |  |  N_EXP_STRING_LITERAL (0,0)
|  |  |  |  |  |  |  |  |  |  |  "HELLO\n"
|  |  |  |  |  |  |  |  |  N_ARG_LIST_NIL (0,0)
|  |  |  |  |  |  |  N_STMT_LIST_NIL (0,0)
san@linux:~/21-2/Compiler/assign4$
```

함수 파라미터, 배열, printf 함수 예시

## 2. 과제를 통해 배운 점

신택스 트리를 구현해 봄으로써 시멘틱 분석을 제외한 컴파일러 동작의 전단부에 해당하는 내용에 대해 깊은 이해를 할 수 있게 되었다. 신택스트리는 N_PROGRAM 노드 밑에 수많은 노드들이 매달린 형태이며 말단에는 터미널 심볼들의 심볼테이블 및 타입테이블이 존재한다. 신택스 트리에서 하나의 노드에는 최대 3개의 자식 노드들이 연결될 수 있으며, 이들의 조합을 통해 원시 프로그램을 텍스트 형태가 아닌, 하나의 자료구조로서 다룰 수 있게 한다. FOR문의 경우 다른 구문들과 달리 좌측에 반복 조건 및 증감을 위한 3개의 노드를 가지므로 보다 더 복잡한 구조를 갖는다. 신택스 트리를 역으로 조사해 올라가면 원시 프로그램을 얻을 수 있다.

## 3. 원시 프로그램 전문

1) kim.l

kim.l

```
digit           [0-9]
letter          [A-Za-z_]
delim           [ \t]
line            [\n]
ws              {delim}+

%{
#include "type.h"
typedef union {
        char cval;
        int ival;
        char* pval;
        A_NODE* anode;
        A_TYPE* atype;
        A_SPECIFIER* aspec;
        A_ID* aid;
        S_KIND skind;
        T_KIND tkind;
} YYSTYPE;
#include "y.tab.h"

extern YYSTYPE yylval;

int yywrap(void);
extern int line_no;
```

```
extern A_ID *current_id;
char *makeString();
int checkIdentifier();
%}

%%

"#include"        { }
"<stdio.h>"       { }
{ws}              { }
{line}            { line_no++; }
auto              { return(AUTO_SYM); }
break             { return(BREAK_SYM); }
case              { return(CASE_SYM); }
continue          { return(CONTINUE_SYM); }
default  { return(DEFAULT_SYM); }
do                { return(DO_SYM); }
else              { return(ELSE_SYM); }
enum              { return(ENUM_SYM); }
for               { return(FOR_SYM); }
if                { return(IF_SYM); }
return            { return(RETURN_SYM); }
sizeof            { return(SIZEOF_SYM); }
static            { return(STATIC_SYM); }
struct            { return(STRUCT_SYM); }
switch            { return(SWITCH_SYM); }
typedef { return(TYPEDEF_SYM); }
union             { return(UNION_SYM); }
while             { return(WHILE_SYM); }

"\+\+"            { return(PLUSPLUS); }
"\-\-"            { return(MINUSMINUS); }
"\->"             { return(ARROW); }
"<"               { return(LSS); }
">"               { return(GTR); }
"<="              { return(LEQ); }
">="              { return(GEQ); }
"=="              { return(EQL); }
"!="              { return(NEQ); }
"&&"              { return(AMPAMP); }
```

```
"||"                    { return(BARBAR); }
"\.\.\."    { return(DOTDOTDOT); }
"("                     { return(LP); }
")"                     { return(RP); }
"\["                    { return(LB); }
"\]"                    { return(RB); }
"\{"                    { return(LR); }
"\}"                    { return(RR); }
"\:"                    { return(COLON); }
"\."                    { return(PERIOD); }
"\,"                    { return(COMMA); }
"\!"                    { return(EXCL); }
"\*"                    { return(STAR); }
"\/"                    { return(SLASH); }
"\%"                    { return(PERCENT); }
"\&"                    { return(AMP); }
"\;"                    { return(SEMICOLON); }
"\+"                    { return(PLUS); }
"\-"                    { return(MINUS); }
"\="                    { return(ASSIGN); }


{digit}+                        { yylval.ival=atoi(yytext); return(INTEGER_CONSTANT); }
{digit}+\.{digit}+              {  yylval.pval=makeString(yytext);  return(FLOAT_CONSTANT);
}
{letter}({letter}|{digit})*     { return(checkIdentifier(yytext)); }
\"([^"\n]|\\["\n])*\"                   {                               yylval.pval=makeString(yytext);
return(STRING_LITERAL); }
\'([^'\n]|\'\')\'               { yylval.cval=*(yytext+1); return(CHARACTER_CONSTANT); }
"//"[^\n]*                              { }

%%

char *makeString(char *s) {
        //printf("makeString called\n");
        char *t;
        t=malloc(strlen(s)+1);
        strcpy(t, s);
        return(t);
}
```

```
int checkIdentifier(char *s) {
	//printf("checkIdentifier called %s\n", s);
	A_ID *id=0;
	char *t;
	id=current_id;
	while(id) {
		if(strcmp(id->name, s)==0)
			break;
		id=id->prev;
	}
	if(id==0) {
		yylval.pval=makeString(s);
		return(IDENTIFIER);
	}
	else if(id->kind==ID_TYPE) {
		yylval.atype=id->type;
		return(TYPE_IDENTIFIER);
	}
	else {
		yylval.pval=id->name;
		return(IDENTIFIER);
	}
}
```

## 2) kim.y

kim.y

```
%{
#include "type.h"
#define YYSTYPE_IS_DECLARED  1
typedef union{
	char cval;
	int ival;
	char* pval;
	A_NODE* anode;
```

```
        A_TYPE* atype;
        A_SPECIFIER* aspec;
        A_ID* aid;
        S_KIND skind;
        T_KIND tkind;
} YYSTYPE;

A_NODE* makeNode(NODE_NAME, A_NODE*, A_NODE*, A_NODE*);
A_NODE* makeNodeList(NODE_NAME, A_NODE*, A_NODE*);
A_ID* makeIdentifier(char*);
A_ID* makeDummyIdentifier();
A_TYPE* makeType(T_KIND);
A_SPECIFIER* makeSpecifier(A_TYPE*, S_KIND);
A_ID* searchIdentifier(char*, A_ID*);
A_ID* searchIdentifierAtCurrentLevel(char*, A_ID*);
A_SPECIFIER* updateSpecifier(A_SPECIFIER*, A_TYPE*, S_KIND);
void checkForwardReference();
void setDefaultSpecifier(A_SPECIFIER*);
A_ID* linkDeclaratorList(A_ID*, A_ID*);
A_ID* getIdentifierDeclared(char*);
A_TYPE* getTypeOfStructOrEnumRefIdentifier(T_KIND, char*, ID_KIND);
A_ID* setDeclaratorInit(A_ID*, A_NODE*);
A_ID* setDeclaratorKind(A_ID*, ID_KIND);
A_ID* setDeclaratorType(A_ID*, A_TYPE*);
A_ID* setDeclaratorElementType(A_ID*, A_TYPE*);
A_ID* setDeclaratorTypeAndKind(A_ID*, A_TYPE*, ID_KIND);
A_ID* setDeclaratorListSpecifier(A_ID*, A_SPECIFIER*);
A_ID* setFunctionDeclaratorSpecifier(A_ID*, A_SPECIFIER*);
A_ID* setFunctionDeclaratorBody(A_ID*, A_NODE*);
A_ID* setParameterDeclaratorSpecifier(A_ID*, A_SPECIFIER*);
A_ID* setStructDeclaratorListSpecifier(A_ID*, A_TYPE*);
A_TYPE* setTypeNameSpecifier(A_TYPE*, A_SPECIFIER*);
A_TYPE* setTypeElementType(A_TYPE*, A_TYPE*);
A_TYPE* setTypeField(A_TYPE*, A_ID*);
A_TYPE* setTypeExpr(A_TYPE*, A_NODE*);
A_TYPE* setTypeAndKindOfDeclarator(A_TYPE*, ID_KIND, A_ID*);
A_TYPE* setTypeStructOrEnumIdentifier(T_KIND, char*, ID_KIND);
BOOLEAN isNotSameFormalParameters(A_ID*, A_ID*);
BOOLEAN isNotSameType(A_TYPE*, A_TYPE*);
BOOLEAN isPointerOrArrayType(A_TYPE*);
```

```
void syntax_error(int, char*);
void initialize();

////////////////////////////////////////////////

void print_ast(A_NODE *);
void prt_program(A_NODE *, int);
void prt_initializer(A_NODE *, int);
void prt_arg_expr_list(A_NODE *, int);
void prt_statement(A_NODE *, int);
void prt_statement_list(A_NODE *, int);
void prt_for_expression(A_NODE *, int);
void prt_expression(A_NODE *, int);
void prt_A_TYPE(A_TYPE *, int);
void prt_A_ID_LIST(A_ID *, int);
void prt_A_ID(A_ID *, int);
void prt_A_ID_NAME(A_ID *, int);
void prt_STRING(char *, int);
void prt_integer(int, int);
void print_node(A_NODE *,int);
void print_space(int);

////////////////////////////////////////////////

void yyerror(char*);
int yywrap(void);
extern int yylex (void);
extern char *yytext;
A_TYPE *int_type, *char_type, *void_type, *float_type, *string_type;
A_NODE *root;
A_ID *current_id=NIL;
int syntax_err=0;
int line_no=1;
int current_level=0;
%}

%token<pval> FLOAT_CONSTANT STRING_LITERAL IDENTIFIER
%token<cval> CHARACTER_CONSTANT
%token<atype> TYPE_IDENTIFIER
%token<ival> INTEGER_CONSTANT
```

```
        PLUS MINUS PLUSPLUS
        MINUSMINUS BAR AMP BARBAR AMPAMP ARROW
        SEMICOLON LSS GTR LEQ GEQ EQL NEQ DOTDOTDOT
        LP RP LB RB LR RR PERIOD COMMA EXCL STAR SLASH PERCENT
ASSIGN
        COLON AUTO_SYM STATIC_SYM TYPEDEF_SYM
        STRUCT_SYM ENUM_SYM SIZEOF_SYM UNION_SYM
        IF_SYM ELSE_SYM WHILE_SYM DO_SYM FOR_SYM CONTINUE_SYM
        BREAK_SYM RETURN_SYM SWITCH_SYM CASE_SYM DEFAULT_SYM

%type<anode> program initializer initializer_list postfix_expression
primary_expression unary_expression cast_expression multiplicative_expression
additive_expression shift_expression relational_expression equality_expression
bitwise_and_expression bitwise_xor_expression bitwise_or_expression
logical_and_expression logical_or_expression conditional_expression
assignment_expression comma_expression expression constant_expression
constant_expression_opt arg_expression_list arg_expression_list_opt
jump_statement expression_opt for_expression iteration_statement
selection_statement expression_statement compound_statement labeled_statement
statement statement_list statement_list_opt
%type<atype> struct_specifier enum_specifier type_specifier type_name
direct_abstract_declarator abstract_declarator abstract_declarator_opt pointer
%type<aspec> declaration_specifiers
%type<aid> translation_unit declaration declaration_list init_declarator_list_opt
init_declarator_list init_declarator struct_declaration_list struct_declaration
struct_declarator_list parameter_declaration parameter_list parameter_type_list
parameter_type_list_opt direct_declarator declarator enumerator enumerator_list
struct_declarator declaration_list_opt external_declaration function_definition
%type<skind> storage_class_specifier
%type<tkind> struct_or_union

%start program

%%

program
        :translation_unit {root=makeNode(N_PROGRAM, NIL, $1, NIL);
checkForwardReference();}
        ;
translation_unit
```

```
        :external_declaration                                   {$$=$1;}
        |translation_unit external_declaration      {$$=linkDeclaratorList($1, $2);}
        ;
external_declaration
        :function_definition        {$$=$1;}
        |declaration                {$$=$1;}
        ;
function_definition
        :declaration_specifiers declarator
{$<aid>$=setFunctionDeclaratorSpecifier($2, $1);}
        compound_statement
{$$=setFunctionDeclaratorBody($<aid>3, $4);}
        |declarator
{$<aid>$=setFunctionDeclaratorSpecifier($1, makeSpecifier(int_type, 0));}
        compound_statement
{$$=setFunctionDeclaratorBody($<aid>2, $3);}
        ;
declaration_list_opt
        :/* empty */                {$$=NIL;}
        |declaration_list {$$=$1;}
        ;
declaration_list
        :declaration                        {$$=$1;}
        |declaration_list declaration       {$$=linkDeclaratorList($1, $2);}
        ;
declaration
        :declaration_specifiers init_declarator_list_opt SEMICOLON
{$$=setDeclaratorListSpecifier($2, $1);}
        ;
declaration_specifiers
        :type_specifier                                         {$$=makeSpecifier($1, 0);}
        |storage_class_specifier                                {$$=makeSpecifier(0, $1);}
        |type_specifier declaration_specifiers                  {$$=updateSpecifier($2,
$1, 0);}
        |storage_class_specifier declaration_specifiers     {$$=updateSpecifier($2,  0,
$1);}
        ;
storage_class_specifier
        :AUTO_SYM       {$$=S_AUTO;}
        |STATIC_SYM     {$$=S_STATIC;}
```

```
        |TYPEDEF_SYM  {$$=S_TYPEDEF;}
        ;
init_declarator_list_opt
        :/* empty */            {$$=makeDummyIdentifier();}
        |init_declarator_list    {$$=$1;}
        ;
init_declarator_list
        :init_declarator                              {$$=$1;}
        |init_declarator_list COMMA init_declarator       {$$=linkDeclaratorList($1,
$3);}
        ;
init_declarator
        :declarator                                   {$$=$1;}
        |declarator ASSIGN initializer    {$$=setDeclaratorInit($1, $3);}
        ;
initializer
        :constant_expression              {$$=makeNode(N_INIT_LIST_ONE, NIL, $1,
NIL);}
        |LR initializer_list RR    {$$=$2;}
        ;
initializer_list
        :initializer                                  {$$=makeNode(N_INIT_LIST,    $1,
NIL, makeNode(N_INIT_LIST_NIL, NIL, NIL, NIL));}
        |initializer_list COMMA initializer {$$=makeNodeList(N_INIT_LIST, $1, $3);}
        ;
type_specifier
        :struct_specifier {$$=$1;}
        |enum_specifier {$$=$1;}
        |TYPE_IDENTIFIER        {$$=$1;}
        ;
struct_specifier
        :struct_or_union IDENTIFIER
{$<atype>$=setTypeStructOrEnumIdentifier($1, $2, ID_STRUCT);}
        LR                                {$<aid>$=current_id; current_level++;}
        struct_declaration_list RR        { c h e c k F o r w a r d R e f e r e n c e ( ) ;
$$=setTypeField($<atype>3, $6); current_level--; current_id=$<aid>5;}
        |struct_or_union          {$<atype>$=makeType($1);}
        LR                                {$<aid>$=current_id; current_level++;}
        struct_declaration_list RR        { c h e c k F o r w a r d R e f e r e n c e ( ) ;
$$=setTypeField($<atype>2, $5); current_level--; current_id=$<aid>4;}
```

```
        |struct_or_union IDENTIFIER     {getTypeOfStructOrEnumRefIdentifier($1,
$2, ID_STRUCT);}
        ;
struct_or_union
        :STRUCT_SYM    {$$=T_STRUCT;}
        |UNION_SYM     {$$=T_UNION;}
        ;
struct_declaration_list
        :struct_declaration                                {$$=$1;}
        |struct_declaration_list struct_declaration{$$=linkDeclaratorList($1, $2);}
        ;
struct_declaration
        :type_specifier struct_declarator_list SEMICOLON
{$$=setStructDeclaratorListSpecifier($2, $1);}
        ;
struct_declarator_list
        :struct_declarator                                        {$$=$1;}
        |struct_declarator_list COMMA struct_declarator  {$$=linkDeclaratorList($1,
$3);}
        ;
struct_declarator
        :declarator       {$$=$1;}
        ;
enum_specifier
        :ENUM_SYM IDENTIFIER
{$<atype>$=setTypeStructOrEnumIdentifier(T_ENUM, $2, ID_ENUM);}
        LR enumerator_list RR  {$$=setTypeField($<atype>3, $5);}
        |ENUM_SYM              {$<atype>$=makeType(T_ENUM);}
        LR enumerator_list RR  {$$=setTypeField($<atype>2, $4);}
        |ENUM_SYM IDENTIFIER {$$=getTypeOfStructOrEnumRefIdentifier(T_ENUM,
$2, ID_ENUM);}
        ;
enumerator_list
        :enumerator                              {$$=$1;}
        |enumerator_list COMMA enumerator    {$$=linkDeclaratorList($1, $3);}
        ;
enumerator
        :IDENTIFIER            {$$=setDeclaratorKind(makeIdentifier($1),
ID_ENUM_LITERAL);}
        |IDENTIFIER            {$<aid>$=setDeclaratorKind(makeIdentifier($1),
```

```
ID_ENUM_LITERAL);}
        ASSIGN expression       {$$=setDeclaratorInit($<aid>2, $4);}
        ;
declarator
        :pointer direct_declarator          {$$=setDeclaratorElementType($2, $1);}
        |direct_declarator                  {$$=$1;}
        ;
pointer
        :STAR           {$$=makeType(T_POINTER);}
        |STAR pointer   {$$=setTypeElementType($2, makeType(T_POINTER));}

        ;
direct_declarator
        :IDENTIFIER                         {$$=makeIdentifier($1);}
        |LP declarator RP                   {$$=$2;}
        |direct_declarator LB constant_expression_opt RB
{$$=setDeclaratorElementType($1, setTypeExpr(makeType(T_ARRAY), $3));}
        |direct_declarator LP               {$<aid>$=current_id; current_level++;}
        parameter_type_list_opt RP      {checkForwardReference();
current_id=$<aid>3;         current_level--;        $$=setDeclaratorElementType($1,
setTypeField(makeType(T_FUNC), $4));}
        ;
parameter_type_list_opt
        :/* empty */              {$$=NIL;}
        |parameter_type_list      {$$=$1;}
        ;
parameter_type_list
        :parameter_list  {$$=$1;}
        |parameter_list COMMA DOTDOTDOT     {$$=linkDeclaratorList($1,
setDeclaratorKind(makeDummyIdentifier(), ID_PARM));}
        ;
parameter_list
        :parameter_declaration  {$$=$1;}
        |parameter_list COMMA parameter_declaration   {$$=linkDeclaratorList($1,
$3);}
        ;
parameter_declaration
        :declaration_specifiers declarator{$$=setParameterDeclaratorSpecifier($2,
$1);}
        |declaration_specifiers abstract_declarator_opt
```

```
{$$=setParameterDeclaratorSpecifier(setDeclaratorType(makeDummyIdentifier(), $2), $1);}
        ;
abstract_declarator_opt
        :/* empty */             {$$=NIL;}
        |abstract_declarator     {$$=$1;}
        ;
abstract_declarator
        :pointer                 {$$=makeType(T_POINTER);}
        |direct_abstract_declarator      {$$=$1;}
        |pointer direct_abstract_declarator      {$$=setTypeElementType($2, makeType(T_POINTER));}
        ;
direct_abstract_declarator
        :LP abstract_declarator RP                                      {$$=$2;}
        |LB constant_expression_opt RB
{$$=setTypeExpr(makeType(T_ARRAY), $2);}
        |direct_abstract_declarator LB constant_expression_opt RB
{$$=setTypeElementType($1, setTypeExpr(makeType(T_ARRAY), $3));}
        |LP parameter_type_list_opt RP
{$$=setTypeField(makeType(T_FUNC), $2);}
        |direct_abstract_declarator LP parameter_type_list_opt RP
{$$=setTypeElementType($1, setTypeField(makeType(T_FUNC), $3));}
        ;
statement_list_opt
        :/* empty */    {$$=makeNode(N_STMT_LIST_NIL, NIL, NIL, NIL);}
        |statement_list  {$$=$1;}
        ;
statement_list
        :statement                      {$$=makeNode(N_STMT_LIST, $1, NIL, makeNode(N_STMT_LIST_NIL, NIL, NIL, NIL));}
        |statement_list statement       {$$=makeNodeList(N_STMT_LIST, $1, $2);}
        ;
statement
        :labeled_statement      {$$=$1;}
        |compound_statement     {$$=$1;}
        |expression_statement   {$$=$1;}
        |selection_statement    {$$=$1;}
        |iteration_statement    {$$=$1;}
        |jump_statement{$$=$1;}
```

```
        ;
labeled_statement
        :CASE_SYM constant_expression COLON statement
{$$=makeNode(N_STMT_LABEL_CASE, $2, NIL, $4);}
        |DEFAULT_SYM COLON statement
{$$=makeNode(N_STMT_LABEL_DEFAULT, NIL, $3, NIL);}
        ;
compound_statement
        :LR {$<aid>$=current_id; current_level++;} declaration_list_opt
        statement_list_opt RR    {checkForwardReference();
        $$=makeNode(N_STMT_COMPOUND, $3, NIL, $4); current_id=$<aid>2;
        current_level--;}
        ;
expression_statement
        :SEMICOLON              {$$=makeNode(N_STMT_EMPTY, NIL, NIL, NIL);}
        |expression SEMICOLON {$$=makeNode(N_STMT_EXPRESSION,    NIL,    $1,
NIL);}
        ;
selection_statement
        :IF_SYM LP expression RP statement
{$$=makeNode(N_STMT_IF, $3, NIL, $5);}
        |IF_SYM LP expression RP statement ELSE_SYM statement
{$$=makeNode(N_STMT_IF_ELSE, $3, $5, $7);}
        |SWITCH_SYM LP expression RP statement
{$$=makeNode(N_STMT_SWITCH, $3, NIL, $5);}
        ;
iteration_statement
        :WHILE_SYM LP expression RP statement
{$$=makeNode(N_STMT_WHILE, $3, NIL, $5);}
        |DO_SYM statement WHILE_SYM LP expression RP SEMICOLON
{$$=makeNode(N_STMT_DO, $2, NIL, $5);}
        |FOR_SYM LP for_expression RP statement
{$$=makeNode(N_STMT_FOR, $3, NIL, $5);}
        ;
for_expression
        :expression_opt SEMICOLON expression_opt SEMICOLON expression_opt
{$$=makeNode(N_FOR_EXP, $1, $3, $5);}
        ;
expression_opt
        :/* empty */     {$$=NIL;}
```

```
        |expression       {$$=$1;}
        ;
jump_statement
        :RETURN_SYM expression_opt SEMICOLON
{$$=makeNode(N_STMT_RETURN, NIL, $2, NIL);}
        |CONTINUE_SYM SEMICOLON
{$$=makeNode(N_STMT_CONTINUE, NIL, NIL, NIL);}
        |BREAK_SYM SEMICOLON
{$$=makeNode(N_STMT_BREAK, NIL, NIL, NIL);}
        ;
arg_expression_list_opt
        :/* empty */              {$$=makeNode(N_ARG_LIST_NIL, NIL, NIL, NIL);}
        |arg_expression_list       {$$=$1;}
        ;
arg_expression_list
        :assignment_expression {$$=makeNode(N_ARG_LIST,          $1,          NIL,
makeNode(N_ARG_LIST_NIL, NIL, NIL, NIL));}
        |arg_expression_list COMMA assignment_expression
{$$=makeNodeList(N_ARG_LIST, $1, $3);}
        ;
constant_expression_opt
        :/* empty */     {$$=NIL;}
        |constant_expression     {$$=$1;}
        ;
constant_expression
        :expression       {$$=$1;}
        ;
expression
        :comma_expression       {$$=$1;}
        ;
comma_expression
        :assignment_expression {$$=$1;}
        ;
assignment_expression
        :conditional_expression  {$$=$1;}
        |unary_expression ASSIGN assignment_expression
{$$=makeNode(N_EXP_ASSIGN, $1, NIL, $3);}
        ;
conditional_expression
        :logical_or_expression    {$$=$1;}
```

```
        ;
logical_or_expression
        :logical_and_expression  {$$=$1;}
        |logical_or_expression BARBAR logical_and_expression
{$$=makeNode(N_EXP_OR, $1, NIL, $3);}
        ;
logical_and_expression
        :bitwise_or_expression   {$$=$1;}
        |logical_and_expression AMPAMP bitwise_or_expression
{$$=makeNode(N_EXP_AND, $1, NIL, $3);}
        ;
bitwise_or_expression
        :bitwise_xor_expression  {$$=$1;}
        ;
bitwise_xor_expression
        :bitwise_and_expression {$$=$1;}
        ;
bitwise_and_expression
        :equality_expression     {$$=$1;}
        ;
equality_expression
        :relational_expression    {$$=$1;}
        |equality_expression EQL relational_expression
{$$=makeNode(N_EXP_EQL, $1, NIL, $3);}
        |equality_expression NEQ relational_expression
{$$=makeNode(N_EXP_NEQ, $1, NIL, $3);}
        ;
relational_expression
        :shift_expression{$$=$1;}
        |relational_expression LSS shift_expression
{$$=makeNode(N_EXP_LSS, $1, NIL, $3);}
        |relational_expression GTR shift_expression
{$$=makeNode(N_EXP_GTR, $1, NIL, $3);}
        |relational_expression LEQ shift_expression
{$$=makeNode(N_EXP_LEQ, $1, NIL, $3);}
        |relational_expression GEQ shift_expression
{$$=makeNode(N_EXP_GEQ, $1, NIL, $3);}
        ;
shift_expression
        :additive_expression     {$$=$1;}
```

```
        ;
additive_expression
        :multiplicative_expression          {$$=$1;}
        |additive_expression PLUS multiplicative_expression
{$$=makeNode(N_EXP_ADD, $1, NIL, $3);}
        |additive_expression MINUS multiplicative_expression
{$$=makeNode(N_EXP_SUB, $1, NIL, $3);}
        ;
multiplicative_expression
        :cast_expression{$$=$1;}
        |multiplicative_expression STAR cast_expression
{$$=makeNode(N_EXP_MUL, $1, NIL, $3);}
        |multiplicative_expression SLASH cast_expression
{$$=makeNode(N_EXP_DIV, $1, NIL, $3);}
        |multiplicative_expression PERCENT cast_expression
{$$=makeNode(N_EXP_MOD, $1, NIL, $3);}
        ;
cast_expression
        :unary_expression        {$$=$1;}
        |LP type_name RP cast_expression      {$$=makeNode(N_EXP_CAST,   $2,
NIL, $4);}
        ;
unary_expression
        :postfix_expression      {$$=$1;}
        |PLUSPLUS unary_expression    {$$=makeNode(N_EXP_PRE_INC,  NIL,  $2,
NIL);}
        |MINUSMINUS unary_expression {$$=makeNode(N_EXP_PRE_DEC,  NIL,  $2,
NIL);}
        |AMP cast_expression          {$$=makeNode(N_EXP_AMP,    NIL,    $2,
NIL);}
        |STAR cast_expression         {$$=makeNode(N_EXP_STAR,   NIL,    $2,
NIL);}
        |EXCL cast_expression         {$$=makeNode(N_EXP_NOT, NIL, $2, NIL);}
        |MINUS cast_expression        {$$=makeNode(N_EXP_MINUS,  NIL,  $2,
NIL);}
        |PLUS cast_expression         {$$=makeNode(N_EXP_PLUS,   NIL,  $2,
NIL);}
        |SIZEOF_SYM unary_expression  {$$=makeNode(N_EXP_SIZE_EXP, NIL, $2,
NIL);}
        |SIZEOF_SYM LP type_name RP {$$=makeNode(N_EXP_SIZE_TYPE, NIL, $3,
```

```
NIL);}
        ;
postfix_expression
        :primary_expression      {$$=$1;}
        |postfix_expression LB expression RB    {$$=makeNode(N_EXP_ARRAY, $1,
NIL, $3);}
        |postfix_expression LP arg_expression_list_opt RP
{$$=makeNode(N_EXP_FUNCTION_CALL, $1, NIL, $3);}
        |postfix_expression PERIOD IDENTIFIER {$$=makeNode(N_EXP_STRUCT,
$1, NIL, $3);}
        |postfix_expression ARROW IDENTIFIER {$$=makeNode(N_EXP_ARROW,
$1, NIL, $3);}
        |postfix_expression PLUSPLUS            {$$=makeNode(N_EXP_POST_INC,
NIL, $1, NIL);}
        |postfix_expression MINUSMINUS{$$=makeNode(N_EXP_POST_DEC, NIL, $1,
NIL);}
        ;
primary_expression
        :IDENTIFIER     {$$=makeNode(N_EXP_IDENT,                    NIL,
getIdentifierDeclared($1), NIL);}
        |INTEGER_CONSTANT    {$$=makeNode(N_EXP_INT_CONST, NIL, $1, NIL);}
        |FLOAT_CONSTANT      {$$=makeNode(N_EXP_FLOAT_CONST,   NIL,   $1,
NIL);}
        |CHARACTER_CONSTANT{$$=makeNode(N_EXP_CHAR_CONST,    NIL,    $1,
NIL);}
        |STRING_LITERAL      {$$=makeNode(N_EXP_STRING_LITERAL, NIL, $1,
NIL);}
        |LP expression RP      {$$=$2;}
        ;
type_name
        :declaration_specifiers abstract_declarator_opt
{$$=setTypeNameSpecifier($2, $1);}
        ;

%%

#include <stdio.h>
#include "y.tab.h"

extern char *yytext;
```

```
void yyerror(char *s) {
	//printf("yyerror called\n");
	syntax_err++;
	printf("line %d: %s near %s \n", line_no, s, yytext);
}
int yywrap(void) {
	//printf("yywrap called\n");
	return(1);
}

int main()
{
	initialize();
	yyparse();
	print_ast(root);
	return 0;
}
////////////////////////////////////////////////////////////////////////
A_NODE* makeNode(NODE_NAME n, A_NODE* a, A_NODE* b, A_NODE* c) {
	//printf("makeNode called\n");
	A_NODE* m;
	m=(A_NODE*)malloc(sizeof(A_NODE));
	m->name=n;
	m->llink=a;
	m->clink=b;
	m->rlink=c;
	m->type=NIL;
	m->line=line_no;
	m->value=0;
	return(m);
}

A_NODE* makeNodeList(NODE_NAME n, A_NODE* a, A_NODE* b) {
	//printf("makeNodeList called\n");
	A_NODE *m, *k;
	k=a;
	while(k->rlink)
		k=k->rlink;
	m=(A_NODE*)malloc(sizeof(A_NODE));
```

```
        m->name=k->name;
        m->llink=NIL;
        m->clink=NIL;
        m->rlink=NIL;
        m->type=NIL;
        m->line=line_no;
        m->value=0;
        k->name=n;
        k->llink=b;
        k->rlink=m;
        return(a);
}

A_ID* makeIdentifier(char* s) {
        //printf("makeIdentifier called\n");
        A_ID *id;
        id=(A_ID*)malloc(sizeof(A_ID));
        id->name=s;
        id->kind=0;
        id->specifier=0;
        id->level=current_level;
        id->address=0;
        id->init=NIL;
        id->type=NIL;
        id->link=NIL;
        id->line=line_no;
        id->value=0;
        id->prev=current_id;
        current_id=id;
        return(id);
}

A_ID* makeDummyIdentifier() {
        //printf("makeDummyIdentifier called\n");
        A_ID *id;
        id=malloc(sizeof(A_ID));
        id->name="";
        id->kind=0;
        id->specifier=0;
        id->level=current_level;
```

```
	id->address=0;
	id->init=NIL;
	id->type=NIL;
	id->link=NIL;
	id->line=line_no;
	id->value=0;
	id->prev=0;
	return(id);
}

A_TYPE* makeType(T_KIND k) {
	//printf("makeType called\n");
	A_TYPE* t;
	t=malloc(sizeof(A_TYPE));
	t->kind=k;
	t->size=0;
	t->local_var_size=0;
	t->element_type=NIL;
	t->field=NIL;
	t->expr=NIL;
	t->check=FALSE;
	t->prt=FALSE;
	t->line=line_no;
	return(t);
}

A_SPECIFIER* makeSpecifier(A_TYPE* t, S_KIND s) {
	//printf("makeSpecifier called\n");
	A_SPECIFIER* p;
	p=malloc(sizeof(A_SPECIFIER));
	p->type=t;
	p->stor=s;
	p->line=line_no;
	return(p);
}

A_ID* searchIdentifier(char* s, A_ID* id) {
	//printf("searchIdentifier called\n");
	while(id) {
		if(strcmp(id->name, s)==0)
```

```
                        break;
                id=id->prev;
        }
        return(id);
}

A_ID* searchIdentifierAtCurrentLevel(char* s, A_ID* id) {
        //printf("searchIdentifierAtCurrentLevel called\n");
        while(id) {
                if(id->level<current_level)
                        return(NIL);
                if(strcmp(id->name, s)==0)
                        break;
                id=id->prev;
        }
        return(id);
}

A_SPECIFIER* updateSpecifier(A_SPECIFIER* p, A_TYPE* t, S_KIND s) {
        //printf("updateSpecifier called\n");
        if(t) {
                if(p->type) {
                        if(p->type==t) ;
                        else syntax_error(24, 0);
                }
                else
                        p->type=t;
        }
        if(s) {
                if(p->stor) {
                        if(s==p->stor) ;
                        else syntax_error(24, 0);
                }
                else
                        p->stor=s;
        }
        return(p);
}

void checkForwardReference() {
```

```
	//printf("checkForwardReference called\n");
	A_ID* id;
	A_TYPE* t;
	id=current_id;
	while(id) {
		if(id->level<current_level)
			break;
		t=id->type;
		if(id->kind==ID_NULL)
			syntax_error(31, id->name);
		else if((id->kind==ID_STRUCT||id->kind==ID_ENUM)&&t->field==NIL)
			syntax_error(32, id->name);
		id=id->prev;
	}
}

void setDefaultSpecifier(A_SPECIFIER* p) {
	//printf("setDefaultSpecifier called\n");
	A_TYPE* t;
	if(p->type==NIL)
		p->type=int_type;
	if(p->stor==S_NULL)
		p->stor=S_AUTO;
}

A_ID* linkDeclaratorList(A_ID* id1, A_ID* id2) {
	//printf("linkDeclaratorList called\n");
	A_ID* m=id1;
	if(id1==NIL)
		return(id2);
	while(m->link)
		m=m->link;
	m->link=id2;
	return(id1);
}

A_ID* getIdentifierDeclared(char* s) {
	//printf("getIdentifierDeclared called\n");
	A_ID* id;
	id=searchIdentifier(s, current_id);
```

```
	if(id==NIL)
		syntax_error(13, s);
	return(id);
}

A_TYPE* getTypeOfStructOrEnumRefIdentifier(T_KIND k, char* s, ID_KIND kk) {
	//printf("getTypeOfStructOrEnumRefIdentifier called\n");
	A_TYPE* t;
	A_ID* id;
	id=searchIdentifier(s, current_id);
	if(id) {
		if(id->kind==kk && id->type->kind==k)
			return(id->type);
		else
			syntax_error(11, s);
	}
	t=makeType(k);
	id=makeIdentifier(s);
	id->kind=kk;
	id->type=t;
	return(t);
}

A_ID* setDeclaratorInit(A_ID* id, A_NODE* n) {
	//printf("setDeclaratorInit called\n");
	id->init=n;
	return(id);
}

A_ID* setDeclaratorKind(A_ID* id, ID_KIND k) {
	//printf("setDeclaratorKind called\n");
	A_ID* a;
	a=searchIdentifierAtCurrentLevel(id->name, id->prev);
	if(a)
		syntax_error(12, id->name);
	id->kind=k;
	return(id);
}

A_ID* setDeclaratorType(A_ID* id, A_TYPE* t) {
```

```
	//printf("setDeclaratorType called\n");
	id->type=t;
	return(id);
}

A_ID* setDeclaratorElementType(A_ID* id, A_TYPE* t) {
	//printf("setDeclaratorElementType called\n");
	A_TYPE* tt;
	if(id->type==NIL)
		id->type=t;
	else {
		tt=id->type;
		while(tt->element_type)
			tt=tt->element_type;
		tt->element_type=t;
	}
	return(id);
}

A_ID* setDeclaratorTypeAndKind(A_ID* id, A_TYPE* t, ID_KIND k) {
	//printf("setDeclaratorTypeAndKind called\n");
	id=setDeclaratorElementType(id, t);
	id=setDeclaratorKind(id, k);
	return(id);
}

A_ID* setDeclaratorListSpecifier(A_ID* id, A_SPECIFIER* p) {
	//printf("setDeclaratorListSpecifier called\n");
	A_ID* a;
	setDefaultSpecifier(p);
	a=id;
	while(a) {
		if(strlen(a->name) && searchIdentifierAtCurrentLevel(a->name,
a->prev))
			syntax_error(12, a->name);
		a=setDeclaratorElementType(a, p->type);
		if(p->stor==S_TYPEDEF)
			a->kind=ID_TYPE;
		else if(a->type->kind==T_FUNC)
			a->kind=ID_FUNC;
```

```
                else
                        a->kind=ID_VAR;
                a->specifier=p->stor;
                if(a->specifier==S_NULL)
                        a->specifier=S_AUTO;
                a=a->link;
        }
        return(id);
}

A_ID* setFunctionDeclaratorSpecifier(A_ID* id, A_SPECIFIER* p) {
        //printf("setFunctionDeclaratorSpecifier called\n");
        A_ID* a;
        if(p->stor)
                syntax_error(25, 0);
        setDefaultSpecifier(p);
        if(id->type->kind!=T_FUNC) {
                syntax_error(21, 0);
                return(id);
        }
        else {
                id=setDeclaratorElementType(id, p->type);
                id->kind=ID_FUNC;
        }
        a=searchIdentifierAtCurrentLevel(id->name, id->prev);
        if(a) {
                if(a->kind!=ID_FUNC || a->type->expr)
                        syntax_error(12, id->name);
                else {
                        if(isNotSameFormalParameters(a->type->field,
id->type->field))
                                syntax_error(22, id->name);
                        if(isNotSameType(a->type->element_type,
id->type->element_type))
                                syntax_error(26, a->name);
                }
        }
        a=id->type->field;
        while(a) {
                if(strlen(a->name))
```

```
                        current_id=a;
                else if(a->type)
                        syntax_error(23, 0);
                a=a->link;
        }
        return(id);
}

A_ID* setFunctionDeclaratorBody(A_ID* id, A_NODE* n) {
        //printf("setFunctionDeclaratorBody called\n");
        id->type->expr=n;
        return(id);
}

A_ID* setParameterDeclaratorSpecifier(A_ID* id, A_SPECIFIER* p) {
        //printf("setParameterDeclaratorSpecifier called\n");
        if(searchIdentifierAtCurrentLevel(id->name, id->prev))
                syntax_error(12, id->name);
        if(p->stor || p->type==void_type)
                syntax_error(14, 0);
        setDefaultSpecifier(p);
        id=setDeclaratorElementType(id, p->type);
        id->kind=ID_PARM;
        return(id);
}

A_ID* setStructDeclaratorListSpecifier(A_ID* id, A_TYPE* t) {
        //printf("setStructDeclaratorListSpecifier called\n");
        A_ID *a;
        a=id;
        while(a) {
                if(searchIdentifierAtCurrentLevel(a->name, a->prev))
                        syntax_error(12, a->name);
                a=setDeclaratorElementType(a, t);
                a->kind=ID_FIELD;
                a=a->link;
        }
        return(id);
}
```

```
A_TYPE* setTypeNameSpecifier(A_TYPE* t, A_SPECIFIER* p) {
	//printf("setTypeNameSpecifier called\n");
	if(p->stor)
		syntax_error(20, 0);
	setDefaultSpecifier(p);
	t=setTypeElementType(t, p-> type);
	return(t);
}

A_TYPE* setTypeElementType(A_TYPE* t, A_TYPE* s) {
	//printf("setTypeElementType called\n");
	A_TYPE* q;
	if(t==NIL)
		return(s);
	q=t;
	while(q->element_type)
		q=q->element_type;
	q->element_type=s;
	return(t);
}

A_TYPE* setTypeField(A_TYPE* t, A_ID* n) {
	//printf("setTypeField called\n");
	t->field=n;
	return(t);
}

A_TYPE* setTypeExpr(A_TYPE* t, A_NODE* n) {
	//printf("setTypeExpr called\n");
	t->expr=n;
	return(t);
}

A_TYPE* setTypeAndKindOfDeclarator(A_TYPE* t, ID_KIND k, A_ID* id) {
	//printf("setTypeAndKindOfDeclarator called\n");
	if(searchIdentifierAtCurrentLevel(id->name, id->prev))
		syntax_error(12, id->name);
	id->type=t;
	id->kind=k;
	return(t);
```

```
}

A_TYPE* setTypeStructOrEnumIdentifier(T_KIND k, char* s, ID_KIND kk) {
	//printf("setTypeStructOrEnumIdentifier called\n");
	A_TYPE* t;
	A_ID *id, *a;
	a=searchIdentifierAtCurrentLevel(s, current_id);
	if(a) {
		if(a->kind==kk && a->type->kind==k) {
			if(a->type->field)
				syntax_error(12, s);
			else
				return(a->type);
		}
		else
			syntax_error(12, s);
	}
	id=makeIdentifier(s);
	t=makeType(k);
	id->type=t;
	id->kind=kk;
	return(t);
}

BOOLEAN isNotSameFormalParameters(A_ID* a, A_ID* b) {
	//printf("isNotSameFormalParameters called\n");
	if(a==NIL)
		return(FALSE);
	while(a) {
		if(b==NIL || isNotSameType(a->type, b->type))
			return(TRUE);
		a=a->link;
		b=b->link;
	}
	if(b)
		return(TRUE);
	else
		return(FALSE);
}
```

```
BOOLEAN isNotSameType(A_TYPE* t1, A_TYPE* t2) {
	//printf("isNotSameType called\n");
	if(isPointerOrArrayType(t1) || isPointerOrArrayType(t2))
		return(isNotSameType(t1->element_type, t2->element_type));
	else
		return(t1!=t2);
}

BOOLEAN isPointerOrArrayType(A_TYPE* t) {
	//printf("isPointerOrArrayType called\n");
	if(t->kind==T_POINTER || t->kind==T_ARRAY)
		return(TRUE);
	else
		return(FALSE);
}

void initialize() {
	int_type=setTypeAndKindOfDeclarator(makeType(T_ENUM), ID_TYPE,
makeIdentifier("int"));
	float_type=setTypeAndKindOfDeclarator(makeType(T_ENUM), ID_TYPE,
makeIdentifier("float"));
	char_type=setTypeAndKindOfDeclarator(makeType(T_ENUM), ID_TYPE,
makeIdentifier("char"));
	void_type=setTypeAndKindOfDeclarator(makeType(T_VOID), ID_TYPE,
makeIdentifier("void"));
	string_type=setTypeElementType(makeType(T_POINTER), char_type);
	int_type->size=4;int_type->check=TRUE;
	float_type->size=4;	float_type->check=TRUE;
	char_type->size=1;	char_type->check=TRUE;
	void_type->size=0;	void_type->check=TRUE;
	string_type->size=4;	int_type->check=TRUE;
	setDeclaratorTypeAndKind(
		makeIdentifier("printf"),
		setTypeField(
			setTypeElementType(makeType(T_FUNC), void_type),
			linkDeclaratorList(
				setDeclaratorTypeAndKind(makeDummyIdentifier(),
string_type, ID_PARM),
				setDeclaratorKind(makeDummyIdentifier(),
ID_PARM))),
```

```
                ID_FUNC);
        setDeclaratorTypeAndKind(
                makeIdentifier("scanf"),
                setTypeField(
                        setTypeElementType(makeType(T_FUNC), void_type),
                        linkDeclaratorList(
                                setDeclaratorTypeAndKind(makeDummyIdentifier(),
string_type, ID_PARM),
                                setDeclaratorKind(makeDummyIdentifier(),
ID_PARM))),
                ID_FUNC);
        setDeclaratorTypeAndKind(
                makeIdentifier("malloc"),
                setTypeField(
                        setTypeElementType(makeType(T_FUNC), string_type),
                        setDeclaratorTypeAndKind(makeDummyIdentifier(), int_type,
ID_PARM)),
                ID_FUNC);
}

void syntax_error(int i, char* s) {
        syntax_err++;
        printf("line %d: syntax error: ", line_no);
        switch(i) {
                case 11: printf("illegal referencing struct or union identifier %s",
s); break;
                case 12: printf("redeclaration of identifier %s", s); break;
                case 13: printf("undefined identifier %s", s); break;
                case 14: printf("illegal type specifier in formal parameter"); break;
                case 20: printf("illegal storage class in type specifiers"); break;
                case 21: printf("illegal function declarator"); break;
                case 22: printf("conflicting parameter type in prototype function
%s", s); break;
                case 23: printf("empty parameter name"); break;
                case 24: printf("illegal declaration specifiers"); break;
                case 25: printf("illegal function specifiers"); break;
                case 26: printf("illegal or conflicting return type in function %s",
s); break;
                case 31: printf("undefined type for identifier %s", s); break;
                case 32: printf("incomplete forward reference for identifier %s", s);
```

```
break;
                default: printf("unknown"); break;
        }
        if(strlen(yytext)==0)
                printf(" at end\n");
        else
                printf(" near %s\n", yytext);
}
char * node_name[] = {
        "N_NULL",
        "N_PROGRAM",
        "N_EXP_IDENT",
        "N_EXP_INT_CONST",
        "N_EXP_FLOAT_CONST",
        "N_EXP_CHAR_CONST",
        "N_EXP_STRING_LITERAL",
        "N_EXP_ARRAY",
        "N_EXP_FUNCTION_CALL",
        "N_EXP_STRUCT",
        "N_EXP_ARROW",
        "N_EXP_POST_INC",
        "N_EXP_POST_DEC",
        "N_EXP_PRE_INC",
        "N_EXP_PRE_DEC",
        "N_EXP_AMP",
        "N_EXP_STAR",
        "N_EXP_NOT",
        "N_EXP_PLUS",
        "N_EXP_MINUS",
        "N_EXP_SIZE_EXP",
        "N_EXP_SIZE_TYPE",
        "N_EXP_CAST",
        "N_EXP_MUL",
        "N_EXP_DIV",
        "N_EXP_MOD",
        "N_EXP_ADD",
        "N_EXP_SUB",
        "N_EXP_LSS",
        "N_EXP_GTR",
        "N_EXP_LEQ",
```

```
            "N_EXP_GEQ",
            "N_EXP_NEQ",
            "N_EXP_EQL",
            "N_EXP_AND",
            "N_EXP_OR",
            "N_EXP_ASSIGN",
            "N_ARG_LIST",
            "N_ARG_LIST_NIL",
            "N_STMT_LABEL_CASE",
            "N_STMT_LABEL_DEFAULT",
            "N_STMT_COMPOUND",
            "N_STMT_EMPTY",
            "N_STMT_EXPRESSION",
            "N_STMT_IF",
            "N_STMT_IF_ELSE",
            "N_STMT_SWITCH",
            "N_STMT_WHILE",
            "N_STMT_DO",
            "N_STMT_FOR",
            "N_STMT_RETURN",
            "N_STMT_CONTINUE",
            "N_STMT_BREAK",
            "N_FOR_EXP",
            "N_STMT_LIST",
            "N_INIT_LIST",
            "N_STMT_LIST_NIL",
            "N_INIT_LIST_NIL",
            "N_INIT_LIST_ONE"};

extern A_TYPE *int_type, *float_type, *char_type, *void_type, *string_type;
void print_node(A_NODE *node, int s) {
            print_space(s);
            printf("%s (%x,%d)\n", node_name[node->name],node->type,node->value);
}
void print_space(int s)
{
            int i;
            for(i=1; i<=s; i++) printf("|   ");
}
void print_ast(A_NODE *node)
```

```
{
        printf("=======  syntax tree  ==========\n");
        prt_program(node,0);
}
void prt_program(A_NODE *node, int s)
{
        print_node(node,s);
        switch(node->name) {
           case N_PROGRAM:
                prt_A_ID_LIST(node->clink, s+1);
                break;
           default :
                printf("****syntax tree error******");
        }
}
void prt_initializer(A_NODE *node, int s)
{
        print_node(node,s);
        switch(node->name) {
           case N_INIT_LIST:
                prt_initializer(node->llink, s+1);
                prt_initializer(node->rlink, s+1);
                break;
           case N_INIT_LIST_ONE:
                prt_expression(node->clink, s+1);
                break;
           case N_INIT_LIST_NIL:
                break;
           default :
                printf("****syntax tree error******");
        }
}
void prt_expression(A_NODE *node, int s)
{
        print_node(node,s);
        switch(node->name) {
           case N_EXP_IDENT :
                prt_A_ID_NAME(node->clink, s+1);
                break;
           case N_EXP_INT_CONST :
```

```
            prt_integer(node->clink, s+1);
            break;
        case N_EXP_FLOAT_CONST :
            prt_STRING(node->clink, s+1);
            break;
        case N_EXP_CHAR_CONST :
            prt_integer(node->clink, s+1);
            break;
        case N_EXP_STRING_LITERAL :
            prt_STRING(node->clink, s+1);
            break;
        case N_EXP_ARRAY :
            prt_expression(node->llink, s+1);
            prt_expression(node->rlink, s+1);
            break;
        case N_EXP_FUNCTION_CALL :
            prt_expression(node->llink, s+1);
            prt_arg_expr_list(node->rlink, s+1);
            break;
        case N_EXP_STRUCT :
        case N_EXP_ARROW :
            prt_expression(node->llink, s+1);
            prt_STRING(node->rlink, s+1);
            break;
        case N_EXP_POST_INC :
        case N_EXP_POST_DEC :
        case N_EXP_PRE_INC :
        case N_EXP_PRE_DEC :
        case N_EXP_AMP :
        case N_EXP_STAR :
        case N_EXP_NOT :
        case N_EXP_PLUS :
        case N_EXP_MINUS :
        case N_EXP_SIZE_EXP :
            prt_expression(node->clink, s+1);
            break;
        case N_EXP_SIZE_TYPE :
            prt_A_TYPE(node->clink, s+1);
            break;
        case N_EXP_CAST :
```

```
                prt_A_TYPE(node->llink, s+1);
                prt_expression(node->rlink, s+1);
                break;
            case N_EXP_MUL :
            case N_EXP_DIV :
            case N_EXP_MOD :
            case N_EXP_ADD :
            case N_EXP_SUB :
            case N_EXP_LSS :
            case N_EXP_GTR :
            case N_EXP_LEQ :
            case N_EXP_GEQ :
            case N_EXP_NEQ :
            case N_EXP_EQL :
            case N_EXP_AND :
            case N_EXP_OR :
            case N_EXP_ASSIGN :
                prt_expression(node->llink, s+1);
                prt_expression(node->rlink, s+1);
                break;
            default :
                printf("****syntax tree error******");
        }
}
void prt_arg_expr_list(A_NODE *node, int s)
{
        print_node(node,s);
        switch(node->name) {
            case N_ARG_LIST :
                prt_expression(node->llink, s+1);
                prt_arg_expr_list(node->rlink, s+1);
                break;
            case N_ARG_LIST_NIL :
                break;
            default :
                printf("****syntax tree error******");
        }
}
void prt_statement(A_NODE *node, int s)
{
```

```
print_node(node,s);

switch(node->name) {
    case N_STMT_LABEL_CASE :
        prt_expression(node->llink, s+1);
        prt_statement(node->rlink, s+1);
        break;
    case N_STMT_LABEL_DEFAULT :
        prt_statement(node->clink, s+1);
        break;
    case N_STMT_COMPOUND:
        if(node->llink)
            prt_A_ID_LIST(node->llink, s+1);
        prt_statement_list(node->rlink, s+1);
        break;
    case N_STMT_EMPTY:
        break;
    case N_STMT_EXPRESSION:
        prt_expression(node->clink, s+1);
        break;
    case N_STMT_IF_ELSE:
        prt_expression(node->llink, s+1);
        prt_statement(node->clink, s+1);
        prt_statement(node->rlink, s+1);
        break;
    case N_STMT_IF:
    case N_STMT_SWITCH:
        prt_expression(node->llink, s+1);
        prt_statement(node->rlink, s+1);
        break;
    case N_STMT_WHILE:
        prt_expression(node->llink, s+1);
        prt_statement(node->rlink, s+1);
        break;
    case N_STMT_DO:
        prt_statement(node->llink, s+1);
        prt_expression(node->rlink, s+1);
        break;
    case N_STMT_FOR:
        prt_for_expression(node->llink, s+1);
```

```
                prt_statement(node->rlink, s+1);
                break;
            case N_STMT_CONTINUE:
                break;
            case N_STMT_BREAK:
                break;
            case N_STMT_RETURN:
                if(node->clink)
                        prt_expression(node->clink, s+1);
                break;
            default :
                printf("****syntax tree error******");
        }
}
void prt_statement_list(A_NODE *node, int s)
{
        print_node(node,s);
        switch(node->name) {
        case N_STMT_LIST:
                prt_statement(node->llink, s+1);
                prt_statement_list(node->rlink, s+1);
                break;
        case N_STMT_LIST_NIL:
                break;
        default :
                printf("****syntax tree error******");

        }
}
void prt_for_expression(A_NODE *node, int s)
{
        print_node(node,s);

        switch(node->name) {

            case N_FOR_EXP :
                if(node->llink)
                        prt_expression(node->llink, s+1);
                if(node->clink)
                        prt_expression(node->clink, s+1);
```

```
                    if(node->rlink)
                            prt_expression(node->rlink, s+1);
                    break;
            default :
                    printf("****syntax tree error******");
        }
}
void prt_integer(int a, int s)
{
        print_space(s);
        printf("%d\n", a);
}
void prt_STRING(char *str, int s) {
        print_space(s);
        printf("%s\n", str);
}
c                       h                       a                       r
*type_kind_name[]={"NULL","ENUM","ARRAY","STRUCT","UNION","FUNC","POINTER","VOI
D"};

void prt_A_TYPE(A_TYPE *t, int s)
{
        print_space(s);
        if (t==int_type)
                printf("(int)\n");
        else if (t==float_type)
                printf("(float)\n");
        else if (t==char_type)
                printf("(char %d)\n",t->size);
        else if (t==void_type)
                printf("(void)");
        else if (t->kind==T_NULL)
                printf("(null)");
        else if (t->prt)
                printf("(DONE:%p)\n",t);
        else
            switch (t->kind) {
                case T_ENUM:
                        t->prt=TRUE;
                        printf("ENUM\n");
```

```
                print_space(s); printf("|   ENUMERATORS\n");
                prt_A_ID_LIST(t->field,s+2);
                break;
        case T_POINTER:
                t->prt=TRUE;
                printf("POINTER\n");
                print_space(s); printf("|   ELEMENT_TYPE\n");
                prt_A_TYPE(t->element_type,s+2);
                break;
        case T_ARRAY:
                t->prt=TRUE;
                printf("ARRAY\n");
                print_space(s); printf("|   INDEX\n");
                if (t->expr)
                        prt_expression(t->expr,s+2);
                else
                        print_space(s+2); printf("(none)\n");
                print_space(s); printf("|   ELEMENT_TYPE\n");
                prt_A_TYPE(t->element_type,s+2);
                break;
        case T_STRUCT:
                t->prt=TRUE;
                printf("STRUCT\n");
                print_space(s); printf("|   FIELD\n");
                prt_A_ID_LIST(t->field,s+2);
                break;
        case T_UNION:
                t->prt=TRUE;
                printf("UNION\n");
                print_space(s); printf("|   FIELD\n");
                prt_A_ID_LIST(t->field,s+2);
                break;
        case T_FUNC:
                t->prt=TRUE;
                printf("FUNCTION\n");
                print_space(s); printf("|   PARAMETER\n");
                prt_A_ID_LIST(t->field,s+2);
                print_space(s); printf("|   TYPE\n");
                prt_A_TYPE(t->element_type,s+2);
                if (t->expr) {
```

```
                                print_space(s); printf("|   BODY\n");
                                prt_statement(t->expr,s+2);}
            }
}
void prt_A_ID_LIST(A_ID *id, int s)
{
        while (id) {
                prt_A_ID(id,s);
                id=id->link;
        }
}
char            *id_kind_name[]={"NULL","VAR","FUNC","PARM","FIELD","TYPE","ENUM",
"STRUCT","ENUM_LITERAL"};
char *spec_name[]={"NULL","AUTO","STATIC","TYPEDEF"};
void prt_A_ID_NAME(A_ID *id, int s)
{
        print_space(s);
        printf("(ID=\"%s\") TYPE:%p KIND:%s SPEC=%s LEV=%d VAL=%d ADDR=%d
\n",
                id->name,               id->type,               id_kind_name[id->kind],
spec_name[id->specifier], id->level, id->value, id->address);
}
void prt_A_ID(A_ID *id, int s)
{
        print_space(s);
        printf("(ID=\"%s\") TYPE:%p KIND:%s SPEC=%s LEV=%d VAL=%d ADDR=%d
\n",
                id->name,                               id->type,id_kind_name[id->kind],
spec_name[id->specifier],id->level, id->value, id->address);
        if (id->type) {
                print_space(s);
                printf("|   TYPE\n");
                prt_A_TYPE(id->type,s+2);}
        if (id->init) {
                print_space(s);
                printf("|   INIT\n");
                if (id->kind==ID_ENUM_LITERAL)
                        prt_expression(id->init,s+2);
                else
                        prt_initializer(id->init,s+2); }}
```