A5 Syntax Tree Printer: print.c

```
#include "type.h"
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_STMT_LIST_NIL",
        "N_INIT_LIST",
        "N_INIT_LIST_ONE",
        "N_INIT_LIST_NIL"};
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);
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);
```

```
}
char
*type_kind_name[]={"NULL","ENUM","ARRAY","STRUCT","UNION","FUNC","POINTER","V
OID"};

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:%x)₩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:%x 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:%x 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);  }
}
```