http://plac.dongguk.ac.kr

컴파일러 구성

제 13 강

U-Code 변환(2)

Programming Language Lab
Dongguk University

Control statement [1/11]

- **Control Statements**
  1. conditional statement - **if**, **case**, **switch**
  2. iteration statement - **for**, **while**, **do-while, loop, repeat-until**
  3. branch statement - **goto**

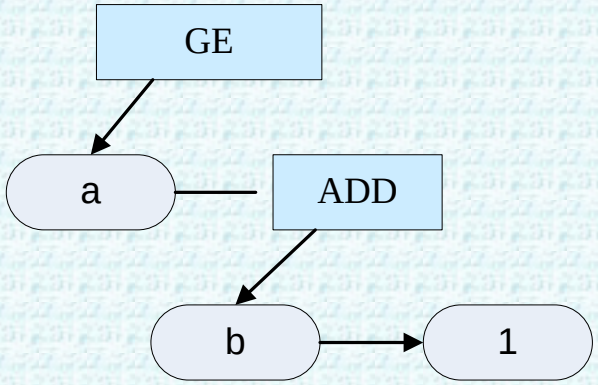
- **Logical expression**
  1. use calculation of logical values
  2. use control expression in control statements

- **Expression of logical values**
  1. true와 false를 숫자로 변환, 산술식의 연산과 유사한 방법으로 계산
  2. 값에 따라 선택적인 실행이 가능

Control statement [2/11]

【예제 10.12】 관계식 a >= b + 1에 대한 AST와 U-코드는 다음과 같다.

- AST 형태 :



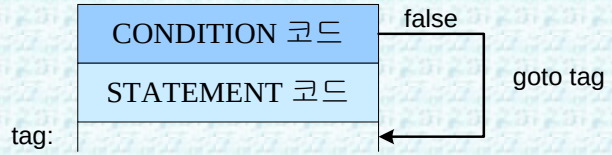
- U-코드 :

```
lod    Ba  Oa    //  Ba: 변수 a의 base, Oa: 변수 a의 offset
lod    Bb  Ob    //  Bb: 변수 b의 base, Ob: 변수 b의 offset
loc    1
add
ge
```

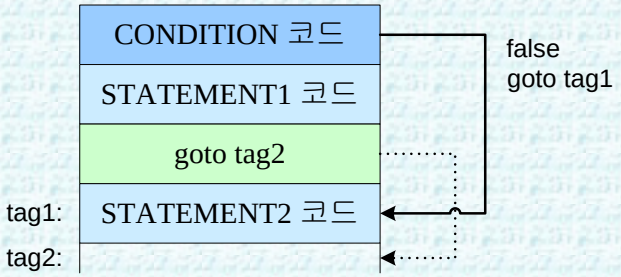
Control statement [3/11]

Scheme for control statements

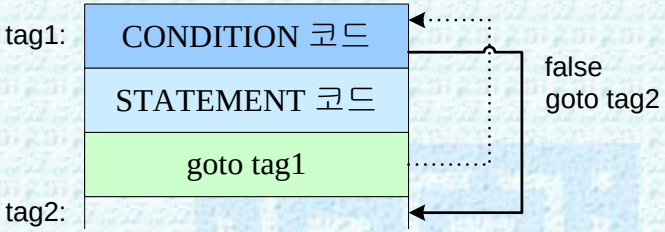
if 구조



if – else 구조



while 구조



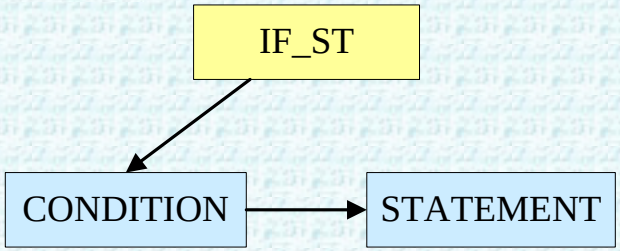
Control statement [4/11]

- **Grammar**

if_st	→	'if' '(' expression ')' statement	=> IF_ST;
	→	'if' '(' expression ')' statement 'else' statement	=> IF_ELSE_ST;
while_st	→	'while' '(' expression ')' statement	=> WHILE_ST;

Control statement [5/11]

- if statement
  - AST

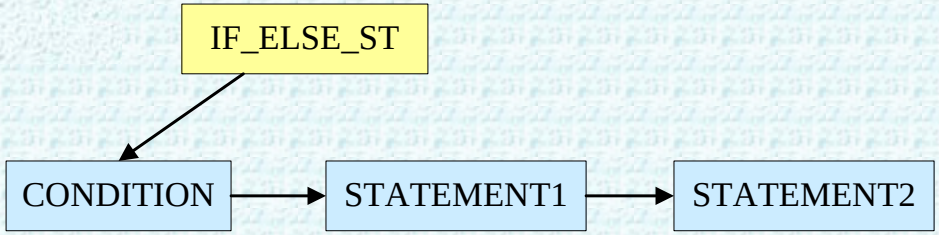


  - Code segment

```
void processStatement(Node *ptr)
{
    //…
    case IF_ST:
    {
        char label[LABEL_SIZE];
        genLabel(label);
        processCondition(ptr->son);      // condition part
        emitJump(fjp, label);
        processStatement(ptr->son->brother); // true part
        emitLabel(label);
    }
    //…
}
```

Control statement [6/11]

- **if-else statement**
  - AST

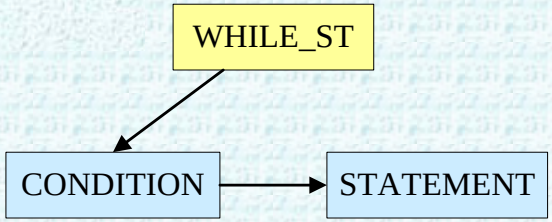


  - Code segment

```
void processStatement(Node *ptr)
{
    //…
    case IF_ELSE_ST:
    {
        char label1[LABEL_SIZE], label2[LABEL_SIZE];
        genLabel(label1); genLabel(label2);
        processCondition(ptr->son);                    // condition part
        emitJump(fjp, label1);
        processStatement(ptr->son->brother);           // true part
        emitJump(ujp, label2);
        emitLabel(label1);
        processStatement(ptr->son->brother->brother);  // false part
        emitLabel(label2);
    }
    //…
}
```

Control statement [7/11]

- **while statement**
  - AST



  - Code segment

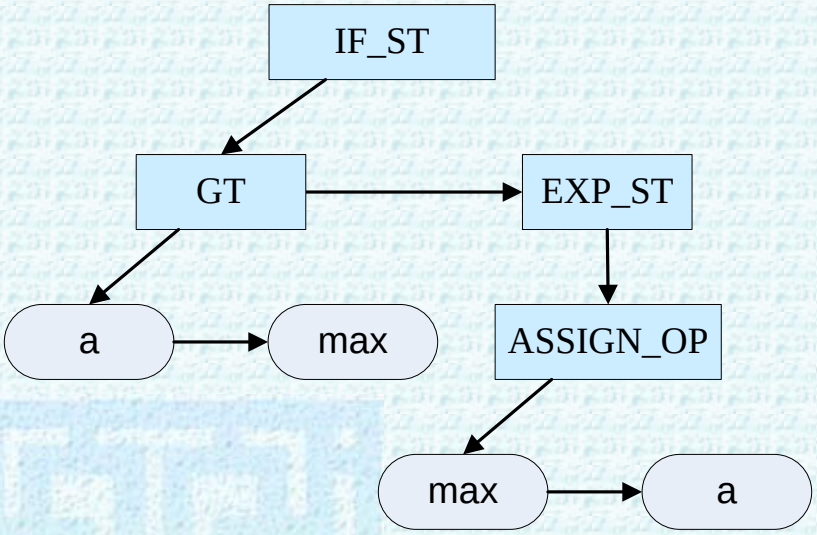
```
void processStatement(Node *ptr)
{
    //…
    case WHILE_ST:
    {
        char label1[LABEL_SIZE], label2[LABEL_SIZE];

        genLabel(label1); genLabel(label2);
        emitLabel(label1);
        processCondition(ptr->son);              // condition part
        emitJump(fjp, label2);
        processStatement(ptr->son->brother);     // loop body
        emitJump(ujp, label1);
        emitLabel(label2);
    }
    //…
}
```

Control statement [8/11]

- Example 1
  - code

```
if (a > max) max = a;
```

  - AST



Control statement [9/11]

- Example 1(계속)
  - U-Code

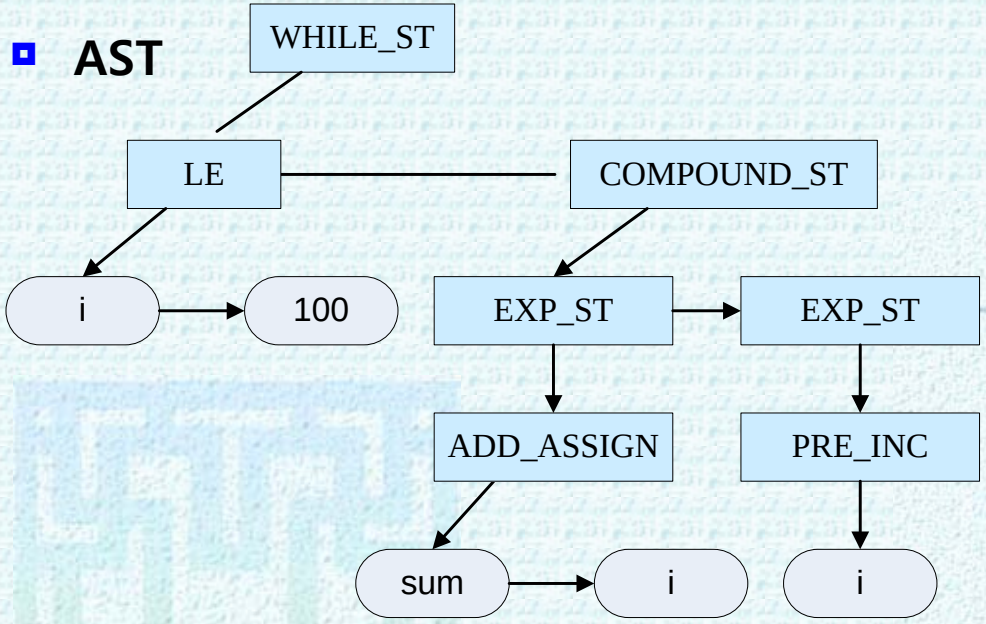
```
        lod     1   1       // a
        lod     1   2       // max
        gt                  // a > max
        fjp     $$1
        lod     1   1
        str     1   2       // max = a
$$1     nop
```

Control statement [10/11]

- Example 2
  - code

```
while (i <= 100) {
    sum += i;
    ++i;
}
```

  - AST



Control statement [11/11]

- Example 2(계속)
  - U-Code

```
$$1     nop
        lod     1   1
        loc     100
        le                  // i <= 100
        fjp     $$2
        lod     sum
        lod     i           // sum += i;
        add
        str     sum
        lod     i           // ++i;
        inc
        str     i
        ujp     $$1
$$2     nop
```

Function

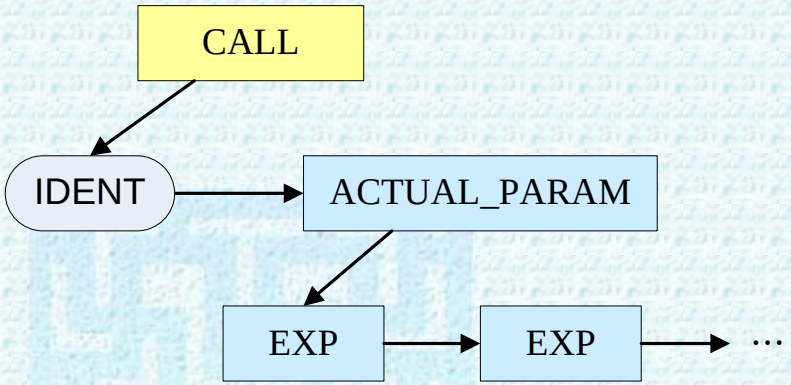
1. Function Call
2. Function Definition

Function – Function call [1/2]

Grammar

```
postfix_exp        →  primary_exp;
                   →  postfix_exp '(' opt_actual_param ')'   => CALL;
opt_actual_param   →  actual_param;
                   →  ;
actual_param       →  actual_param_list                      => ACTUAL_PARAM;
actual_param_list  →  assignment_exp;
                   →  actual_param_list ',' assignment_exp;
```

AST



Function – Function call [2/2]

- **Process function call**

```
void processStatement(Node *ptr)
{
        //…
        case CALL:
        {
            //…
            // predefined(Library) functions
            //…
            // handle for user function
            functionName = p->token.value.id;
            stIndex = lookup(functionName);
            if (stIndex == -1) break; // undefined function !!!
            noArguments = symbolTable[stIndex].width;

            emit0(ldp);
            p = p->brother;        // ACTUAL_PARAM
            while (p) {            // processing actual arguments
                    if (p->noderep == nonterm) processOperator(p);
                    else rv_emit(p);
                    noArguments--;
                    p = p->brother;
            }
            //…
            emitJump(call, ptr->son->token.value.id);
            break;
        }
        //…
}
```

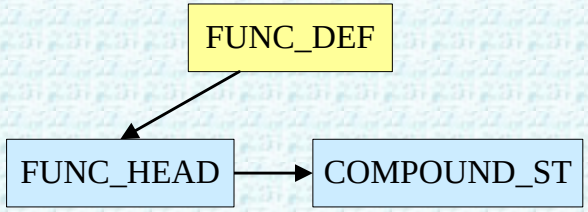
Function – Function definition [1/4]

Grammar

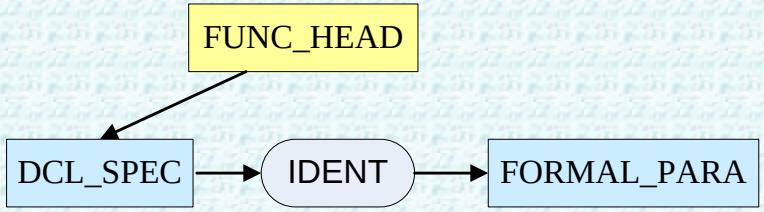
function_def	→	function_header compound_st	=> FUNC_DEF;
function_header	→	dcl_spec function_name formal_param	=> FUNC_HEAD;
function_name	→	'%ident';	
formal_param	→	'(' opt_formal_param ')'	=> FORMAL_PARA;
opt_formal_param	→	formal_param_list;	
	→	;	
formal_param_list	→	param_dcl;	
	→	formal_param_list ',' param_dcl;	
param_dcl	→	dcl_spec declarator	=> PARAM_DCL;

Function – Function definition [2/4]

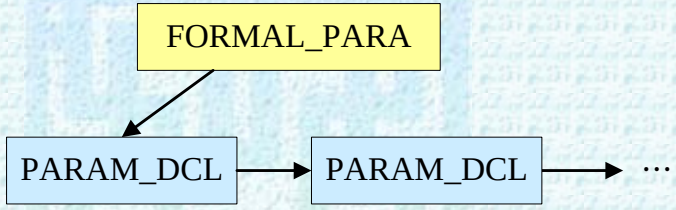
- **AST**
  - Function definition



  - Function head



  - Formal parameter



Function – Function definition [3/4]

- **Process function header**

```
void processFuncHeader(Node *ptr)
{
    //…
    // step 1: determine return type
    p = ptr->son->son;
    while (p) {
        if (p->token.number == INT_NODE) returnType = INT_TYPE;
        else if (p->token.number == VOID_NODE) returnType = VOID_TYPE;
        else printf("invalid function return type\n");
        p = p->brother;
    }
    // step 2: count the number of formal parameters
    p = ptr->son->brother->brother;          // FORMAL_PARA
    p = p->son;                              // PARAM_DCL
    noArguments = 0;
    while (p) {
        noArguments++;
        p = p->brother;
    }
    // step 3: insert function name
    stIndex = insert(ptr->son->brother->token.value.id, returnType, FUNC_TYPE,
                     1/*base*/, 0/*offset*/, noArguments/*width*/, 0/*initialValue*/);
    //if (!strcmp("main", functionName)) mainExist = 1;
}
```

Function – Function definition [4/4]

- **Main routine for processing a function definition**

```
void processFunction(Node *ptr)
{
    // …
    // step 1: process function header          // already explained
    // step 2: process function body
    // …
}

void processFunctionBody(Node *ptr)
{
    // …
    // step 1: process the declaration part in function body
    // step 2: emit the function start code
    // step 3: process the statement part in function body
    // step 4: check if return type and return value
    // step 5: generate the ending codes
    // …
}
```