Addition of Non-Terminals in Grammar from Assignment 4

- Non terminal M has been augmented, with reduction to epsilon. To make backpatching possible by putting the index of quad where required.

- Non terminal N has been augmented, which implements an unconditional goto when reduced to epsilon

- Non terminal F is augmented, for function declaration computation handling.

Instances:

- logical_AND_expression → logical_AND_expression BINARY_AND M inclusive_OR_expression
M augmented so that if first expression is true then it jumps to M

Code:
logical_AND_expression :
logical_AND_expression BINARY_AND M inclusive_OR_expression

```
{
        backpatch($1.TL,$3);
        $$.FL = merge($1.FL, $4.FL);
        $$.TL = $4.TL;
        $$.type = new_node(BOOL_,-1);
}
```

- logical_OR_expression → logical_OR_expression BINARY_OR M logical_AND_expression
Here, M augmented so that if $1 is false then it jumps to $3
Code:

```
{
        backpatch($1.FL,$3);
        $$.TL = merge($1.TL,$4.TL);
        $$.FL = $4.FL;
        $$.type = new_node(BOOL_,-1);
}
```

- conditional_expression → logical_OR_expression N '?' M expression N ':' M conditional_expression
The details are complex, and have been discussed in class. N is used for unconditional goto depending on truth value of logical_OR_expression.

The code is lengthy and is omitted here.

- block_item_list → block_item_list M block_item
M used for backpathcing to block_item_list ($1)’s next list

- function_definition → declaration_specifiers declarator F compound_statement
Here F is used to change the current symbol table pointer to point to the symbol table of apt

function as we enter its scope.

Codes for reduction of these three non-terminals to epsilon is as follows:

M:

```
$$ = next_instr;
```

N:

```
$$ = makelist(next_instr);
fields_quad x(0,0,0,GOTO_,0,0,0);
quad_array->emit(x);
```

F :

```
current = temp_use;
int i;
char *t;
for(i=0;i<=global->curr_length;i++){
    if((((global->table)[i]).nestedTable) == current){
        t = strdup(((global->table)[i]).name);
        break;
    }
}
fields_quad x(t,0,0,Function,0,0,0);
quad_array->emit(x);
flag1 = 0;
flag2 = 0;
c = 0;
```

Attributes

I) attribute_exp (struct) → used for :
primary_expression, expression postfix_expression, constant_expression statement, compound_statement, selection_statement, iteration_statement ,jump_statement, block_item_list, block_item, expression_statement, unary_expression, cast_expression multiplicative_express additive_expression shift_expression relational_expression equality_expression

II) attribute_variable_declaration (struct) → used for:
type_specifier, declaration_specifiers, direct_declarator, declarator, parameter_declaration, init_declarator, init_declarator_list, pointer

III) attribute_unary (struct) → used for:
unary_operator

IV) instr (int) → used for:
M

V) attribute_N → used for:
N

VI) intval, charval, floatval and strval (char*)

Design of structures:

Stucture	Type/ Composition	Use
data_type	enum	Variable data types used in the assignment, along with BOOL.
quad_data_type	enum	Enum for op fields of quad
tnode	struct down – data_type l – int * r – tnode*	An expansion tree is used with tnode as the structure for its nodes
lnode	struct index_list – int next – lnode*	List nodes
symbol_table_fields	class name – char * type – tnode* initial_value – void* size – int offset – int nestedTable - symbolTable*	Storing data related to fields of symbol table at each row.
parameter_list	struct parameter – symboltablefiedls * next – parameter_list *	List of parameters
quad_array_fields	struct arg1 – char* arg2 – char* res – char* op – quadEnum arg1_loc – symboltablefields* arg2_loc – symboltablefields* arg3_loc – symboltablefields*	Storing data related to fields of quad table at each row.
initializer_attr_struct	Struct int_data – int double_data – double char_data - char	attribute of initializer
attribute_expression	struct loc – symboltablefields * TL- lnode * FL- lnode * NL- lnode *	Data type for exp_attr

	type – tnode* array – symboltablefields* loc1 – symboltablefields* val - initializer_attr_struct	
attribute_variable_declaration	Struct type – tnode* width – int var – char *	Data type for var_decl_attr
id_attr_struct	Struct loc – symboltablefields* var – char *	Data type for id_attr
SymbolTable	Class table – symboltablefields * curr_length – int	Data type for implementing symbol table
QuadArray	quad_Table – quadArrayRow*	Data type for implementing quad array

Symbol Table

- Implemented by SymbolTable class whose data members have been shown in the table above.
- Member Methods

– ***symboltablefields *lookup(char *)*** : looks up an entry in the table and returns a pointer to the row if found, null otherwise

- void insert(symboltablefields &) : Inserts a row in the symbolTable-

-***symboltablefields *gentemp(typeEnum)*** : generates a temporary

- void print_table() : prints the table by printing each row