

Compilers Lab Assignment _3
Bison and Flex code

ANURAG RAMTEKE - 150101010
NEEL MITTAL - 150101042
PRASHANSI KAMDAR - 150101047

*/*bison code*/*

```
%{
#include <stdio.h>
void func(char * s);
%}
%define parse.error verbose
%token BREAK
%token CHAR
%token CONTINUE
%token ELSE
%token FLOAT
%token FOR
%token IF
%token INT
%token RETURN
%token VOID
%token WHILE
%token PROC
%token LNK
%token JB
%token COLON
%token CLUST
%token CLUSTER
%token PROCESSOR
%token ISA
%token PROC_TYPE
%token CLOCK_SPEED
%token MEM1
%token MEM2
%token NAME
%token TOPOLOGY
%token LINK_BANDWIDTH
%token LINK_CAPACITY
%token LINK
%token START_POINT
```

%token	END_POINT
%token	MEMORY_TYPE
%token	MEMORY_SIZE
%token	JOB
%token	JOB_ID
%token	FLOPS_REQUIRED
%token	DEADLINE
%token	MEM_REQUIRED
%token	AFFINITY
%token	RUN
%token	WAIT
%token	DISCARD_JOB
%token	STOP
%token	GET_AVAILABLE_MEMORY
%token	GET_JOB_AFFINITY
%token	GET_JOB_MEMORY
%token	GET_FLOPS
%token	GET_DEADLINE
%token	IS_RUNNING
%token	SUBMIT_JOBS
%token	GET_FLOPS_SPEED
%token	GET_PROC_TYPE
%token	IS_PROCESSOR
%token	GET_PROCESSOR
%token	MEM
%token	IDENTIFIER
%token	CONSTANT
%token	STRING_LITERAL
%token	RIGHT_OP
%token	LEFT_OP
%token	INC_OP
%token	DEC_OP
%token	AND_OP
%token	OR_OP
%token	LE_OP
%token	GE_OP
%token	EQ_OP
%token	NE_OP
%token	SEMI_COLON
%token	LEFT_CURLY
%token	RIGHT_CURLY
%token	COMMA
%token	ASGN_OP

%token LEFT_PARENTHESIS
%token RIGHT_PARENTHESIS
%token LEFT_BRACKET
%token RIGHT_BRACKET
%token DOT
%token AMPERSAND
%token NOT_OP
%token BTW_NOT
%token MINUS
%token PLUS
%token MUL_OP
%token DIV_OP
%token MOD_OP
%token LESS_THAN
%token GREATER_THAN
%token XOR_OP
%token BTW_OR
%token INVALID
%token MEMORY
%token PROCESSORS
%token Deref_OP
%token GET_CLOCK_SPEED
%token MEM_TYPES
%start Start
%%
Start

:Function_declaration Start {func("Start->Function_declaration Start\n");}
|Object_declaration Start {func("Start->Object_declaration Start\n");}
|Function_declaration {func("Start->Function_declaration\n");}
|Object_declaration {func("Start->Object_declaration\n");}
;

Object_declaration
:Dec_specifier IDENTIFIER ASGN_OP postfix_expression SEMI_COLON {func("Object_declaration->Dec_specifier IDENTIFIER ASGN_OP postfix_expression SEMI_COLON\n");}
|Dec_specifier IDENTIFIER ASGN_OP object_expression SEMI_COLON {func("Object_declaration->Dec_specifier IDENTIFIER ASGN_OP object_expression SEMI_COLON\n");}
|Dec_specifier IDENTIFIER SEMI_COLON {func("Object_declaration->Dec_specifier IDENTIFIER SEMI_COLON\n");}
|Dec_specifier array_expression ASGN_OP array_initialiser SEMI_COLON {func("Object_declaration->Dec_specifier array_expression ASGN_OP array_initialiser SEMI_COLON\n");}
|Dec_specifier array_expression SEMI_COLON {func("Object_declaration->Dec_specifier array_expression SEMI_COLON\n");}
;

Dec_specifier
:type_specifier pointer {func("Dec_specifier->Type_specifier pointer\n");}
|type_specifier {func("Dec_specifier->Type_specifier\n");}
;

array_initialiser

```
:LEFT_CURLY array_initialiser RIGHT_CURLY {func("array_initialiser->LEFT_CURLY array_initialiser RIGHT_CURLY\n");}  
|LEFT_CURLY array_initialiser RIGHT_CURLY COMMA array_initialiser {func("array_initialiser->array_initialiser COMMA LEFT_CURLY array_initialiser RIGHT_CURLY\n");}  
|expression {func("array_initialiser->expression\n");}  
;
```

Function_declaration

```
:Dec_specifier IDENTIFIER LEFT_PARENTHESIS argument_list RIGHT_PARENTHESIS body_or_proto {func("Function_declaration->Dec_specifier IDENTIFIER LEFT_PARENTHESIS argument_list  
RIGHT_PARENTHESIS body_or_proto\n");}  
|Dec_specifier IDENTIFIER LEFT_PARENTHESIS RIGHT_PARENTHESIS body_or_proto {func("Function_declaration->Dec_specifier IDENTIFIER LEFT_PARENTHESIS  
body_or_proto\n");}  
;
```

body_or_proto

```
:LEFT_CURLY statement_list RIGHT_CURLY {func("body_or_proto->LEFT_CURLY statement_list RIGHT_CURLY\n");}  
|SEMI_COLON {func("body_or_proto->SEMI_COLON\n");}  
;
```

argument_list

```
:Dec_specifier IDENTIFIER COMMA argument_list {func("argument_list->Dec_specifier IDENTIFIER COMMA argument_list\n");}  
|Dec_specifier IDENTIFIER {func("argument_list->Dec_specifier IDENTIFIER\n");}  
;
```

type_specifier

```
:INT {func("type_specifier->INT\n");}  
|VOID {func("type_specifier->VOID\n");}  
|FLOAT {func("type_specifier->FLOAT\n");}  
|CHAR {func("type_specifier->CHAR\n");}  
|PROC {func("type_specifier->PROC\n");}  
|CLUST {func("type_specifier->CLUST\n");}  
|JB {func("type_specifier->JB\n");}  
|MEM {func("type_specifier->MEM\n");}  
|LNK {func("type_specifier->LNK\n");}  
;
```

pointer

```
:MUL_OP pointer {func("pointer->MUL_OP pointer\n");}  
|MUL_OP {func("pointer->MUL_OP\n");}  
;
```

statement_list

```
:Statement statement_list {func("statement_list->Statement statement_list\n");}  
|Statement {func("statement_list->Statement\n");}  
;
```

Statement

```
:LEFT_CURLY statement_list RIGHT_CURLY      {func("Statement->LEFT_CURLY statement_list RIGHT_CURLY\n");}  
|Expression_statement      {func("Statement->Expression_Statement\n");}  
|Selection_statement       {func("Statement->Selection_statement\n");}  
|Iteration_statement       {func("Statement->Iteration_statement\n");}  
|Jump_statement           {func("Statement->Jump_statement\n");}  
|Object_declaration       {func("Statement->Object_declaration\n");}  
;
```

Jump_statement

```
:RETURN expression {func("Jump_statement->RETURN expression\n");}  
|CONTINUE SEMI_COLON {func("Jump_statement->CONTINUE SEMI_COLON\n");}  
|BREAK SEMI_COLON    {func("Jump_statement->BREAK SEMI_COLON\n");}  
;
```

Selection_statement

```
:IF LEFT_PARENTHESIS expression RIGHT_PARENTHESIS LEFT_CURLY statement_list RIGHT_CURLY ELSE LEFT_CURLY statement_list RIGHT_CURLY {func("Select_statement->IF LEFT_PARENTEHSIS  
expression RIGHT_PARENTHESIS LEFT_CURLY statement_list RIGHT_CURLY else LEFT_CURLY statement_list RIGHT_CURLY\n");}  
|IF LEFT_PARENTHESIS expression RIGHT_PARENTHESIS LEFT_CURLY statement_list RIGHT_CURLY {func("Select_statement->IF LEFT_PARENTHESIS expression RIGHT_PARENTHESIS LEFT_CURLY  
statement_list RIGHT_CURLY\n");}  
;
```

Iteration_statement

```
:WHILE LEFT_PARENTHESIS expression RIGHT_PARENTHESIS Statement {func("Iteration_Statement->WHILE LEFT_PARENTEHSIS expression RIGHT_PARENTHESIS Statement\n");}  
|FOR LEFT_PARENTHESIS Expression_statement Expression_statement RIGHT_PARENTHESIS Statement {func("Iteration_Statement->FOR LEFT_PARENTHESIS Expression_statement  
Expression_statement RIGHT_PARENTHESIS Statement\n");}  
|FOR LEFT_PARENTHESIS Expression_statement Expression_statement expression RIGHT_PARENTHESIS Statement {func("Iteration_Statement->FOR LEFT_PARENTHESIS Expression_statement  
Expression_statement expression RIGHT_PARENTHESIS Statement\n");}  
;
```

Expression_statement

```
:expression SEMI_COLON {func("Expression_Statement->expression SEMI_COLON\n");}  
|SEMI_COLON {func("Expression_Statement->SEMI_COLON\n");}  
;
```

expression

```
:assignment_expression {func("expression->assignment_expression\n");}  
|assignment_expression COMMA expression {func("expression->assignment_expression COMMA expression\n");}  
;
```

assignment_expression

```
:unary_expression ASGN_OP assignment_expression {func("assignment_expression->unary_expression ASGN_OP assignment_expression\n");}  
|conditional_expression {func("assignment_expression->conditional_expression\n");}  
|object_expression {func("assignment_expression->object_expression\n");}  
;
```

conditional_expression

```
:logical_or_expression {func("conditional_expression->logical_or_expression\n");}
```

```

;
logical_or_expression
:logical_and_expression OR_OP logical_or_expression {func("logical_or_expression->logical_or_expression OR_OP logical_and_expression\n");}
|logical_and_expression {func("logical_or_expression->logical_and_expression\n");}
;

logical_and_expression
:bitwise_or_expression AND_OP logical_and_expression {func("logical_and_expression->logical_and_expression AND_OP bitwise_or_expression\n");}
|bitwise_or_expression {func("logical_and_expression->bitwise_or_expression\n");}
;

bitwise_or_expression
:bitwise_xor_expression BTW_OR bitwise_or_expression {func("bitwise_or_expression->bitwise_or_expression BTW_OR bitwise_xor_expression\n");}
|bitwise_xor_expression {func("bitwise_or_expression->bitwise_xor_expression\n");}
;

bitwise_xor_expression
:and_expression {func("bitwise_xor_expression->and_expression\n");}
|and_expression XOR_OP bitwise_xor_expression {func("bitwise_xor_expression->bitwise_xor_expression XOR_OP and_expression\n");}
;

and_expression
:equality_expression {func("and_expression->equality_expression\n");}
|equality_expression AMPERSAND and_expression {func("and_expression->equality_expression AMPERSAND and_expression\n");}
;

equality_expression
:relational_expression {func("equality_expression->relational_expression\n");}
|relational_expression EQ_OP equality_expression {func("equality_expression->equality_expression EQ_OP relational_expression\n");}
|relational_expression NE_OP equality_expression {func("equality_expression->equality_expression NE_OP relational_expression\n");}
;

relational_expression
:shift_expression {func("relational_expression->shift_expression\n");}
|shift_expression LESS_THAN relational_expression {func("relational_expression->relational_expression LESS_THAN shift_expression\n");}
|shift_expression GREATER_THAN relational_expression {func("relational_expression->relational_expression GREATER_THAN shift_expression\n");}
|shift_expression LE_OP relational_expression {func("relational_expression->relational_expression LE_OP shift_expression\n");}
|shift_expression GE_OP relational_expression {func("relational_expression->relational_expression GE_OP shift_expression\n");}
;

shift_expression
:additive_expression {func("shift_expression->additive_expression\n");}
|additive_expression LEFT_OP shift_expression {func("shift_expression->shift_expression LEFT_OP additive_expression\n");}
|additive_expression RIGHT_OP shift_expression {func("shift_expression->shift_expression RIGHT_OP additive_expression\n");}
;

additive_expression
:multiplicative_expression {func("additive_expression->multiplicative_expression\n");}
|multiplicative_expression PLUS additive_expression {func("additive_expression->additive_expression PLUS multiplicative_expression\n");}
|multiplicative_expression MINUS additive_expression {func("additive_expression->additive_expression MINUS multiplicative_expression\n");}

```

```

;
multiplicative_expression
:unary_expression    {func("multiplicative_expression->unary_expression\n");}
|multiplicative_expression MUL_OP unary_expression    {func("multiplicative_expression->multiplicative_expression MUL_OP unary_expression\n");}
|multiplicative_expression DIV_OP unary_expression    {func("multiplicative_expression->multiplicative_expression DIV_OP unary_expression\n\n");}
|multiplicative_expression MOD_OP unary_expression    {func("multiplicative_expression->multiplicative_expression MOD_OP unary_expression\n");}
;

unary_expression
:postfix_expression  {func("unary_expression->postfix_expresso\n");}
|INC_OP unary_expression  {func("unary_expression->INC_OP unary_expresso\n");}
|DEC_OP unary_expression {func("unary_expression->DEC_OP unary_expression\n");}
|unary_operator unary_expression  {func("unary_expression->unary_operator unary_expression\n");}
;

unary_operator
:PLUS {func("unary_operator->PLUS\n");}
|MINUS      {func("unary_operator->MINUS\n");}
|BTW_NOT    {func("unary_operator->BTW_NOT\n");}
|NOT_OP     {func("unary_operator->NOT_OP\n");}
|MUL_OP     {func("unary_operator->MUL_OP\n");}
|AMPERSAND  {func("unary_operator->AMPERSAND\n");}
;

postfix_expression
:primary_expression  {func("postfix_expression->primary_expression\n");}
|array_expression     {func("postfix_expression->array_expression\n");}
|function_expression {func("postfix_expression->function_expression\n");}
|postfix_expression INC_OP {func("postfix_expression->postfix_expression INC_OP\n");}
|postfix_expression DEC_OP      {func("postfix_expression->postfix_expression DEC_OP\n");}
;

array_expression
:array_expression LEFT_BRACKET expression RIGHT_BRACKET      {func("array_expression->array_expression LEFT_BRACKET expression RIGHT_EXPRESSION\n");}
|primary_expression LEFT_BRACKET expression RIGHT_BRACKET    {func("array_expression->Primary_expression LEFT_BRACKET expression RIGHT_BRACKET\n");}
;

primary_expression
:IDENTIFIER Deref_OP IDENTIFIER      {func("primary_expression->IDENTIFIER Deref_OP IDENTIFIER\n");}
|IDENTIFIER {func("primary_expression->IDENTIFIER\n");}
|CONSTANT {func("primary_expression->CONSTANT\n");}
|STRING_LITERAL {func("primary_expression->STRING_LITERAL\n");}
|LEFT_PARENTHESIS expression RIGHT_PARENTHESIS      {func("primary_expression->LEFT_PARENTHESIS expression RIGHT_PARENTHESIS\n");}
;

object_expression

```

```

:link_object    {func("object_expression->link_object\n");}
|memory_object  {func("object_expression->memory_object\n");}
|job_object     {func("object_expression->job_object\n");}
|cluster_object {func("object_expression->cluster_object\n");}
;
cluster_object
:CLUSTER LEFT_PARENTHESIS proc_arr_arg COMMA topology_arg COMMA link_band_arg COMMA link_cap_arg narp {func("cluster_object->CLUSTER LEFT_PARENTHESIS proc_arr_arg COMMA
topology_arg COMMA link_band_arg COMMA link_cap_arg narp\n");}
|processor_object {func("cluster_object->processor_object\n");}
;
proc_arr_arg
:PROCESSORS ASGN_OP IDENTIFIER {func("proc_arr_arg->PROCESSORS ASGN_OP IDENTIFIER\n");}
|PROCESSORS ASGN_OP LEFT_BRACKET cluster_list RIGHT_BRACKET {func("proc_arr_arg->PROCESSORS ASGN_OP LEFT_BRACKET cluster_list RIGHT BRACKET\n");}
|IDENTIFIER {func("proc_arr_arg->IDENTIFIER\n");}
|LEFT_BRACKET cluster_list RIGHT_BRACKET {func("proc_arr_arg->LEFT BRACKET cluster_list RIGHT_BRACKET\n");}
;
cluster_list
:cluster_object COMMA cluster_list {func("cluster_list->cluster_object COMMA cluster_list\n");}
|cluster_object {func("cluster_list->cluster_object\n");}
;
topology_arg
:TOPOLOGY ASGN_OP STRING_LITERAL {func("topology_arg->TOPOLOGY ASGN_OP STRING_LITERAL\n");}
|STRING_LITERAL {func("topology_arg->STRING_LITERAL\n");}
;
link_band_arg
:LINK_BANDWIDTH ASGN_OP conditional_expression {func("link_band_arg->LINK_BANDWIDTH ASGN_OP conditional_expression\n");}
|conditional_expression {func("link_band_arg->conditional_expression\n");}
;
link_cap_arg
:LINK_CAPACITY ASGN_OP conditional_expression {func("link_cap_arg->LINK_CAPACITY ASGN_OP conditional_expression\n");}
|conditional_expression {func("link_cap_arg->conditional_expression\n");}
;
narp
:COMMA conditional_expression RIGHT_PARENTHESIS {func("narp->COMMA conditional_expression RIGHT_PARENTHESIS\n");}
|COMMA NAME ASGN_OP conditional_expression RIGHT_PARENTHESIS {func("narp->COMMA NAME ASGN_OP conditional_expression\n");}
|RIGHT_PARENTHESIS {func("narp->RIGHT PARENTHESIS\n");}
;
processor_object
:PROCESSOR LEFT_PARENTHESIS isa_args COMMA Clock_args COMMA mem_args narp {func("processor_object->PROCESSOR LEFT_PARENTHESIS isa_args COMMA Clock_args COMMA mem_args
narp\n");}
;
isa_args
:ISA ASGN_OP PROC_TYPE {func("isa_args->ISA ASGN_OP PROC_TYPE\n");}

```



```

        |PROC_TYPE {func("isa_args->PROC_TYPE\n");}
        ;
Clock_args
    :CLOCK_SPEED ASGN_OP CONSTANT {func("Clock_args->CLOCK_SPEED ASGN_OP CONSTANT\n");}
    |CONSTANT {func("Clock_args->CONSTANT\n");}
    ;

mem_args
    :MEM1 ASGN_OP memory COLON MEM2 ASGN_OP memory {func("mem_args->MEM1 ASGN_OP memory COMMA MEM2 ASGN_OP Memory\n");}
    |MEM1 ASGN_OP memory COLON memory {func("mem_args->MEM1 ASGN_OP memory COMMA memory\n");}
    |MEM1 ASGN_OP memory {func("mem_args->MEM1 ASGN_OP memory\n");}
    |memory COLON MEM2 ASGN_OP memory {func("mem_args->memory COMMA MEM2 ASGN_OP memory\n");}
    |memory COLON memory {func("mem_args->memory COMMA memory\n");}
    |memory {func("mem_args->memory\n");}
    ;
memory
    :memory_object {func("memory->memory_object\n");}
    |IDENTIFIER {func("memory->IDENTIFIER\n");}
    ;
link_object
    :LINK LEFT_PARENTHESIS start_args COMMA end_args COMMA link_band_arg COMMA link_cap_arg narp {func("link_object->LINK LEFT_PARENTHESIS start_args COMMA end_args COMMA link_band_arg
COMMA link_cap_arg narp\n");}
    ;
start_args
    :START_POINT ASGN_OP conditional_expression {func("start_args->START_POINT ASGN_OP conditional expression\n");}
    |conditional_expression {func("start_args->conditional_expression\n");}
    ;
end_args
    :END_POINT ASGN_OP conditional_expression {func("end_args->END_POINT ASGN_OP conditional_expression\n");}
    |conditional_expression {func("end_args->conditional_expression\n");}
    ;
memory_object
    :MEMORY LEFT_PARENTHESIS mem_type_args COMMA mem_size_args narp {func("memory_object->MEMORY LEFT_PARENTHESIS mem_type_args COMMA mem_size_args narp\n");}
    ;
mem_type_args
    :MEMORY_TYPE ASGN_OP MEM_TYPES {func("mem_type_args->MEMORY_TYPE ASGN_OP MEM_TYPES\n");}
    |MEM_TYPES {func("mem_type_args->MEM_TYPES\n");}
    ;
mem_size_args
    :MEMORY_SIZE ASGN_OP conditional_expression {func("mem_size_args->MEMORY_SIZE ASGN_OP conditional_expression\n");}

```

```

        |conditional_expression {func("mem_size_args->conditional_expression\n");}
    ;
job_object
    :JOB LEFT_PARENTHESIS job_id_args COMMA flops_args COMMA deadline_args COMMA mem_required_args COMMA affinity_args RIGHT_PARENTHESIS {func("job_object->JOB LEFT_PARENTHESIS
job_id_args COMMA flop_args COMM deadline_args COMMA mem_required_args COMMA affinity_args RIGHT_PARENTHESIS\n");}
    ;
job_id_args
    :JOB_ID ASGN_OP conditional_expression {func("job_id_args->JOB_ID ASGN_OP conditional_expression\n");}
    |conditional_expression {func("job_id_args->conditional_expression\n");}
    ;
flops_args
    :FLOPS_REQUIRED ASGN_OP conditional_expression {func("flops_args->JOB_ID ASGN_OP conditional_expression\n");}
    |conditional_expression {func("flops_args->conditional_expression\n");}
    ;
deadline_args
    :DEADLINE ASGN_OP conditional_expression {func("deadline_args->DEADLINE ASGN_OP conditional_expression\n");}
    |conditional_expression {func("deadline_args->conditional_expression\n");}
    ;
mem_required_args
    :MEM_REQUIRED ASGN_OP conditional_expression {func("mem_required_args->MEM_REQUIRED ASGN_OP conditional_expression\n");}
    |conditional_expression {func("mem_required_args->conditional_expression\n");}
    ;
affinity_args
    :AFFINITY ASGN_OP LEFT_BRACKET list RIGHT_BRACKET {func("affinity_args->AFFINITY ASGN_OP LEFT_BRACKET list RIGHT_BRACKET\n");}
    |LEFT_BRACKET list RIGHT_BRACKET {func("affinity_args->LEFT_BRACKET list RIGHT_BRACKET\n");}
    ;
list
    :CONSTANT {func("list->CONSTANT\n");}
    |CONSTANT COMMA list {func("list->CONSTANT COMMA list\n");}
    ;
function_expression
    :RUN LEFT_PARENTHESIS cluster_list RIGHT_PARENTHESIS {func("function_expression->RUN LEFT_PARENTHESIS cluster_list RIGHT_PARENTHESIS\n");}
    |WAIT LEFT_PARENTHESIS expression RIGHT_PARENTHESIS {func("function_expression->WAIT LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
    |DISCARD_JOB LEFT_PARENTHESIS job_list RIGHT_PARENTHESIS {func("function_expression->DISCARD_JOB LEFT_PARENTHESIS job_list RIGHT_PARENTHESIS\n");}
    |STOP LEFT_PARENTHESIS IDENTIFIER RIGHT_PARENTHESIS {func("function_expression->STOP LEFT_PARENTHESIS IDENTIFIER RIGHT_PARENTHESIS\n");}
    |IDENTIFIER LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("function_expression->IDENTIFIER LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
    |IDENTIFIER LEFT_PARENTHESIS expression RIGHT_PARENTHESIS {func("function_expression->IDENTIFIER LEFT_PARENTHESIS expression RIGHT_PARENTHESIS\n");}
    |IDENTIFIER DOT object_function {func("function_expression->IDENTIFIER DOT object_function\n");}
    ;
object_function
    :processor_function {func("object_function->processor_function\n");}
    |job_function {func("object_function->job_function\n");}
    |memory_function {func("object_function->memory_function\n");}

```

```

|cluster_function {func("object_function->cluster_function\n");}
;
memory_function
:GET_AVAILABLE_MEMORY LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("memory_function->GET_AVAILABLE_MEMORY LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
;

job_function
:GET_JOB_AFFINITY LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("job_function->GET_JOB_AFFINITY LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
|GET_JOB_MEMORY LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("job_function->GET_JOB_MEMORY LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
|GET_FLOPS LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("job_function->GET_FLOPS LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
|GET_DEADLINE LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("job_function->GET_DEADLINE LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
;

processor_function
:IS_RUNNING LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("processor_function->IS_RUNNING LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
|SUBMIT_JOBS LEFT_PARENTHESIS job_list RIGHT_PARENTHESIS {func("processor_function->SUBMIT_JOBS LEFT_PARENTHESIS job_list RIGHT_PARENTHESIS\n");}
|GET_CLOCK_SPEED LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("processor_function->GET_CLOCK_SPEED LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
|GET_PROC_TYPE LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("processor_function->GET_PROC_TYPE LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
|IS_PROCESSOR LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("processor_function->IS_PROCESSOR LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
;

cluster_function
:GET_PROCESSOR LEFT_PARENTHESIS RIGHT_PARENTHESIS {func("cluster_function->GET_PROCESSOR LEFT_PARENTHESIS RIGHT_PARENTHESIS\n");}
|GET_PROCESSOR LEFT_PARENTHESIS primary_expression RIGHT_PARENTHESIS {func("cluster_function->GET_PROCESSOR LEFT_PARENTHESIS primary_expression RIGHT_PARENTHESIS\n");}
;

job_list
:IDENTIFIER COMMA job_list {func("job_list->IDENTIFIER COMMA job_list\n");}
|IDENTIFIER {func("job_list->IDENTIFIER\n");}
|job_object COMMA job_list {func("job_list->job_object COMMA job_list\n");}
|job_object {func("job_list->job_object\n");}
;

%%

extern char yytext[];
extern int column;
extern int yylineno;
extern int yyleng;
void func(char * s)
{

```

```
        printf("%s",s);
    }
    int yyerror(s)
    char *s;
    {
        fflush(stdout);
        printf("At Line No. %d Column No. %d : %s\n",yylineno,column-yyldeng,s);
        return 0;
    }
    FILE * fp;
    int main()
    {
        fp=fopen("lex_file","w");
        yyparse();
        fclose(fp);
        return 0;
    }
}
```