Department of Computer Science & Engineering
IIT Guwahati, India, March 12, 2018
CS347 Compilers Lab - Assignment 3

Ayush soni(150101014)
Mayank agrawal (150101033)
Abhishek kumar (150101003)

PARSER CODE:

```
%{

%}

%token          NUM INT FLOAT FLOAT_CONST STRING STRING_CONST
BOOL DEL ID COMPARATOR CLOSESQ OPENSQ OPENBR %token    CLOSEBR
OPENPR VAR
%token    CLOSEPR EQUAL QUOTES COMMA COLON DOT
%token          FOR WHILE IF ELSE
%token    IS_RUNNING SUBMIT_JOBS GET_CLOCK_SPEED DISCARD_JOB
JOB_ID
%token    GET_AVAILABLE_MEMORY GET_MEMORY MEM_SIZE
%token    PROCESSOR ISA CLOCK_SPEED L1_MEM ARM AMD CDC MIPS
MEMORY MEMORY_TYPE MEMORY_SIZE
%token    PRIMARY SECONDARY CACHE LINK START_POINT END_POINT
BANDWIDTH JOB FLOPS_REQUIRED
%token    DEADLINE MEM_REQUIRED AFFINITY RUN WAIT
%token    CLUSTER TOPOLOGY NAME STAR RING BUS
%token    SUM
%token          MULT
%token          UNARY_OP
%start          program
%expect    3

%%
program : statement_list {printf("program --> statement_list\n");}
          ;
statement_list  : statement statement_list {printf("statement_list --> statement
statement_list\n");}
                    |
                    ;
statement : var_decl DEL{printf("statement --> var_decl DEL\n");}
           | loop {printf("statement --> loop\n");}
           | ifstmnt {printf("statement --> ifstmnt \n");}
           |func_call DEL {printf("statement --> func_call DEL\n");}
           | expr DEL {printf("statement --> expr DEL\n");}
           | obj_constr DEL {printf("statement --> obj_constr DEL\n");}
           ;
```

```
var_decl : var_type VAR {printf("var_decl --> var_type VAR\n");}
                    ;

var_type : INT{printf("var_type --> INT\n");}
                    | STRING{printf("var_type --> STRING\n");}
                    | BOOL{printf("var_type --> BOOL\n");}
                    | FLOAT{printf("var_type --> FLOAT\n");}
                    ;

loop : for_loop{printf("loop --> for_loop\n");}
          | while_loop{printf("loop --> while_loop\n");}
          ;

for_loop : FOR OPENPR expr DEL condition DEL expr CLOSEPR OPENBR
statement_list CLOSEBR {printf("for_loop\n");}

while_loop : WHILE OPENPR condition CLOSEPR OPENBR statement_list CLOSEBR
{printf("while_loop\n");}

ifstmnt : IF condition OPENBR statement_list CLOSEBR ELSE OPENBR statement_list
CLOSEBR {printf("ifstmnt\n");}

func_call : proc_func_call {printf("func_call --> proc_func_call\n");}
                     | mem_func_call {printf("func_call --> mem_func_call\n");}
                     | job_func_call {printf("func_call --> job_func_call\n");}
                     | wait_call{printf("func_call --> wait_call\n");}
                     ;

wait_call : WAIT OPENPR NUM CLOSEPR {printf("wait_call --> WAIT OPENPR NUM
CLOSEPR\n");}
                     ;

proc_func_call : VAR EQUAL VAR DOT IS_RUNNING OPENPR CLOSEPR
{printf("proc_func_call --> VAR EQUAL VAR DOT IS_RUNNING OPENPR
CLOSEPR\n");}
                                | VAR DOT SUBMIT_JOBS OPENPR parameter CLOSEPR
{printf("proc_func_call --> VAR DOT SUBMIT_JOBS OPENPR parameter
CLOSEPR\n");}
                                | VAR DOT GET_CLOCK_SPEED OPENPR CLOSEPR
{printf("proc_func_call --> VAR DOT GET_CLOCK_SPEED OPENPR CLOSEPR\n");}
                                | RUN OPENPR parameter CLOSEPR {printf("proc_func_call -->
RUN OPENPR parameter CLOSEPR\n");}
                                | DISCARD_JOB OPENPR VAR CLOSEPR {printf("proc_func_call
--> DISCARD_JOB OPENPR VAR CLOSEPR\n");}
                                ;
```

```
mem_func_call : VAR DOT  GET_AVAILABLE_MEMORY OPENPR CLOSEPR
{printf("mem_func_call --> VAR DOT  GET_AVAILABLE_MEMORY OPENPR
CLOSEPR\n");}
                              ;

job_func_call : VAR DOT GET_MEMORY OPENPR CLOSEPR {printf("job_func_call -->
VAR DOT GET_MEMORY OPENPR CLOSEPR\n");}
                              ;

obj_constr : VAR EQUAL class_obj {printf("obj_constr --> VAR EQUAL class_obj\n");}
                     ;

class_obj : processor_obj {printf("class_obj --> processor_obj\n");}
                    | mem_obj {printf("class_obj --> mem_obj\n");}
                    | job_obj {printf("class_obj --> job_obj\n");}
                    | link_obj {printf("class_obj --> link_obj\n");}
                    | cluster_obj {printf("class_obj --> cluster_obj\n");}
                    ;

processor_obj : PROCESSOR OPENPR ISA EQUAL QUOTES isa_type QUOTES
COMMA CLOCK_SPEED COLON FLOAT_CONST COMMA L1_MEM EQUAL ID
CLOSEPR {printf("processor_obj\n");}

isa_type : ARM {printf("isa_type --> ARM\n");}
                   | AMD {printf("isa_type --> AMD\n");}
                   | CDC {printf("isa_type --> CDC\n");}
                   | MIPS OPENPR STRING_CONST CLOSEPR {printf("isa_type --> MIPS
OPENPR STRING_CONST CLOSEPR\n");}
                   ;


mem_obj : MEMORY OPENPR MEMORY_TYPE EQUAL QUOTES mem_type
QUOTES COMMA MEM_SIZE EQUAL NUM CLOSEPR {printf("mem_obj -->
MEMORY OPENPR MEMORY_TYPE EQUAL QUOTES mem_type QUOTES\n");}
             | MEMORY OPENPR MEMORY_TYPE EQUAL QUOTES mem_type QUOTES
COMMA MEM_SIZE EQUAL NUM COMMA NAME EQUAL STRING_CONST
CLOSEPR {printf("mem_obj --> MEMORY OPENPR MEMORY_TYPE EQUAL
QUOTES mem_type QUOTES some_extras\n");}
             ;

mem_type : PRIMARY {printf("mem_type --> PRIMARY\n");}
                   | SECONDARY {printf("mem_type --> SECONDARY\n");}
                   | CACHE OPENPR STRING_CONST CLOSEPR {printf("mem_type -->
CACHE OPENPR STRING_CONST CLOSEPR\n");}
                   ;
```

```
link_obj : LINK OPENPR START_POINT STRING_CONST COMMA END_POINT
EQUAL QUOTES COMMA BANDWIDTH EQUAL FLOAT_CONST COMMA
FLOAT_CONST CLOSEPR {printf("link_obj\n");}
                    ;

job_obj : JOB OPENPR JOB_ID EQUAL NUM COMMA FLOPS_REQUIRED EQUAL n
COMMA DEADLINE EQUAL n COMMA MEM_REQUIRED EQUAL NUM COMMA
AFFINITY EQUAL OPENSQ float_arr CLOSESQ CLOSEPR  {printf("job_obj");}
                   ;

float_arr : FLOAT_CONST COMMA float_arr {printf("float_arr --> FLOAT_CONST
COMMA float_arr\n");}
               | FLOAT_CONST COMMA FLOAT_CONST {printf("float_arr -->
FLOAT_CONST COMMA FLOAT_CONST");}
               ;

n : FLOAT_CONST {printf("n --> FLOAT_CONST\n");}
  | NUM {printf("n --> NUM\n");}
  ;


cluster_obj : CLUSTER OPENPR PROCESSOR EQUAL parameter COMMA
TOPOLOGY EQUAL QUOTES top_type QUOTES COMMA FLOAT_CONST COMMA
FLOAT_CONST NAME EQUAL STRING_CONST {printf("cluster_obj\n");}
                       |

top_type : STAR {printf("top_type --> STAR\n");}
                    | RING {printf("top_type -> RING\n");}
                    | BUS {printf("top_type --> BUS\n");}
                    ;

parameter : VAR {printf("parameter --> VAR\n");}
         | OPENSQ var_list CLOSESQ {printf("parameter --> OPENSQ var_list
CLOSESQ\n");}
         ;

var_list : VAR more_var {}
                    ;

more_var :  COMMA VAR {printf("more_var -->  COMMA VAR\n");}
                     | COMMA VAR more_var {printf("more_var -->  COMMA VAR
more_var\n");}
                     ;

expr : ID EQUAL condition {printf("expr --> ID EQUAL condition\n");}
          | ID EQUAL  arithmatic_op {printf("expr --> ID EQUAL  arithmatic_op\n");}
          | condition {printf("expr --> condition\n");}
          ;
```

```
condition : arithmatic_op COMPARATOR arithmatic_op {printf("condition -->
arithmatic_op COMPARATOR arithmatic_op\n");}
                    | UNARY_OP arithmatic_op {printf("condition --> UNARY_OP
arithmatic_op\n");}
                    | arithmatic_op UNARY_OP {printf("condition --> arithmatic_op
UNARY_OP\n");}
                    ;

arithmatic_op : mul SUM arithmatic_op {printf("mul --> factor MULT mul\n");}
                       | mul {printf("mul --> factor MULT mul\n");}
                       ;

mul : factor MULT mul {printf("mul --> factor MULT mul\n");}
         | factor {printf("mul --> factor\n");}
         ;

factor : ID {printf("factor --> ID \n");}
            | OPENPR arithmatic_op OPENPR {printf("factor --> OPENPR arithmatic_op
OPENPR \n");}
            ;
```

LEXER CODE:

```
DIGIT                [0-9]
QUOTES               """
STRING               [^ ]*
TEXT_NUMBERS [a-zA-Z0-9_]
NUM          {DIGIT}+
ID                   [a-zA-Z]{TEXT_NUMBERS}*
VAR                  {ID} | {ID}"["{NUM}"]" | {ID}"["{ID}"]"
FLOATCONST           {NUM}"."{NUM}
BOOLCONST            "true"|"false"
SUM          "+"|"-"
MUL          "*"|"/"
LOGICAL_OP           "&"|"\|"
UNARY_OP             "!"|"++"|"--"
COMPARATOR           ">"|"<"|">="|"<="|"=="|"!="



%%
"("                  { return (OPENPR);}
")"                          { return (CLOSEPR);         }
"{"                          { return (OPENBR);          }
"}"                          { return (CLOSEBR);         }
```

```
"["                             { return (OPENSQ);          }
"]"                             { return (CLOSESQ);         }
"."                             { return (DOT);     }
";"                             { return (DEL);             }
":"                             { return (COLON);  }
"="                             { return (EQUAL);  }
{QUOTES}            { return (QUOTES);          }
{BOOLCONST}                     { return (BOOLCONST);}
{NUM}                           { return (NUM);     }
{FLOATCONST}    { return (FLOAT_CONST);         }
{SUM}                           { return (SUM);             }
{MUL}                           { return (MUL);     }
{UNARY_OP}                      { return (UNARY_OP);                }
{COMPARATOR}  { return (COMPARATOR);}
"if"                { return (IF);          }
"while"                         { return (WHILE);   }
"else"              { return (ELSE);     }
"for"               { return (FOR);                 }
"int"               { return (INT);                 }
"float"             { return (FLOAT);   }
"bool"              { return (BOOL);    }
"string"            { return (STRING); }
"Processor"         { return (PROCESSOR);}
"Job"               { return (JOB);         }
"Cluster"           { return (CLUSTER);         }
"Memory"            { return (MEMORY);          }
"Link"              { return (LINK);     }
"is_running"  { return (IS_RUNNING);}
"primary"           { return (PRIMARY);         }
"secondary"         { return (SECONDARY); }
"cache"                         { return (CACHE);  }
"flops_required"{ return (FLOPS_REQUIRED);}
"submit_jobs"       { return (SUBMIT_JOBS);         }
"get_clock_speed"{ return (GET_CLOCK_SPEED);}
"discard_job"       { return (DISCARD_JOB);}
"job_id"            { return (JOB_ID);  }
"get_available_memory" { return (GET_AVAILABLE_MEMORY);         }
"get_memory"        { return (GET_MEMORY);          }
"mem_size"          { return (MEM_SIZE);     }
"isa"               { return (ISA);         }
"clock_speed"       { return (CLOCK_SPEED);         }
"l1_mem"            { return (L1_MEM);          }
"ARM"                           { return (ARM);     }
"AMD"                           { return (AMD);     }
"CDC"                           { return (CDC);     }
"MIPS"                          { return (MIPS);    }
"memory_type"       { return (MEMORY_TYPE);         }
"memory_size"       { return (MEMORY_SIZE);         }
```

```
"start_point"  { return (START_POINT); }
"end_point"           { return (END_POINT);    }
"bandwidth"           { return (BANDWIDTH);  }
"deadline"            { return (DEADLINE);      }
"mem_required"      { return (MEM_REQUIRED);}
"affinity"            { return (AFFINITY);        }
"topology"            { return (TOPOLOGY);     }
"name"                        { return (NAME);    }
"star"                { return (STAR);      }
"bus"                 { return (BUS);                 }
{ID}                  { return (ID);                  }
\"{STRING}\"          { return (STRING_CONST);        }
\'{STRING}\'          { return (STRING_CONST);        }
{VAR}                         { return (VAR);                 }

%%



int main(int argc, char *argv[]) {
yylex();
return 0;
}
```