Group 10
Compilers Lab
12 March 2018

150101009 Anup Agarwal
150101078 Surabhi Gupta
150101059 Satti Sai Chandan Reddy

Assignment 2 Submission

---------- lexer.l ----------

```
D                               [0-9]
L                               [a-zA-Z_]
WS                              [ \t\n\v\f]

%{
#include "y.tab.h"
%}

%%
Processor                       { return(PROCESSOR); }
Link                            { return(LINK); }
Cluster                         { return(CLUSTER); }
Job                             { return(JOB); }
Memory                          { return(MEMORY); }
Scheduler                       { return(SCHEDULER); }
sched                           { return(SCHED); }
job_id                          { return(JOB_ID); }
flops_required                  { return(FLOPS_REQUIRED); }
deadline                        { return(DEADLINE); }
affinity                        { return(AFFINITY); }
algo                            { return(ALGO); }
isa                             { return(ISA); }
clock_speed                     { return(CLOCK_SPEED); }
mem_required                    { return(MEM_REQUIRED); }
l1_memory                       { return(L1_MEMORY); }
l2_memory                       { return(L2_MEMORY); }
memory_size                     { return(MEM_SIZE); }
name                            { return(NAME); }
start_point                     { return(START_POINT); }
end_point                       { return(END_POINT); }
bandwidth                       { return(BANDWIDTH); }
channel_capacity                { return(CHANNEL_CAPACITY); }
topology                        { return(TOPOLOGY); }
link_capacity                   { return(LINK_CAPACITY); }
link_bandwidth                  { return(LINK_BANDWIDTH); }
get_available_memory            { return(GET_AVAILABLE_MEMORY); }
get_memory                      { return(GET_MEMORY); }
is_running                      { return(IS_RUNNING); }
submit_jobs                     { return(SUBMIT_JOBS); }
get_clock_speed                 { return(GET_CLOCK_SPEED); }
discard_job                     { return(DISCARD_JOB); }
run                             { return(RUN); }
char                            { return(CHAR); }
else                            { return(ELSE); }
float                           { return(FLOAT); }
for                             { return(FOR); }
if                              { return(IF); }
int                             { return(INT); }
return                          { return(RETURN); }
void                            { return(VOID); }
while                           { return(WHILE); }
do                              { return(DO); }
bool                            { return(BOOL); }
mem_size                        { return(MEM_SIZE); }
```

```
memory_type                     { return(MEMORY_TYPE); }
"->"                            { return(PTR_OP); }
"&"                             { return(AMP); }
"~"                             { return(TILDE); }
"+"                             { return(PLUS); }
"-"                             { return(MINUS); }
"*"                             { return(STAR); }
"/"                             { return(DIVIDE); }
"%"                             { return(MODULUS); }
"++"                            { return(INC_OP); }
"--"                            { return(DEC_OP); }
"&&"                            { return(AND_OP); }
"||"                            { return(OR_OP); }
"<"                             { return(LT); }
">"                             { return(GT); }
"<="                            { return(LE_OP); }
">="                            { return(GE_OP); }
"=="                            { return(EQ_OP); }
"!="                            { return(NE_OP); }
"!"                             { return(NOT); }
"="                             { return(ASSIGN); }
":"                             { return(COLON); }
"("                             { return(OPEN_BRACKET); }
")"                             { return(CLOSED_BRACKET); }
"{"                             { return(OPEN_CURLY); }
"}"                             { return(CLOSED_CURLY); }
L?\"(\\.|[^\\"])*\"             { return(STRING_LITERAL); }
L?\'(\\.|[^\\'])*\'             { return(STRING_LITERAL); }
"]"                             { return(CLOSED_SQUARE); }
"["                             { return(OPEN_SQUARE); }
[0-9]*"."[0-9]+                 { return(REAL); }
[0-9]+                          { return(NUM);}
{L}({L}|{D})*                   { return(ID); }
";"                             { return(SEMI);}
","                             { return(COMMA); }
"."                             { return(DOT); }
"^"                             { return(XOR); }
"|"                             { return(PIPE); }
"?"                             { return(QUES); }
{WS}+                           {}


%%

int yywrap(void)
{
    return(1);
}

----------- scanner.c -----------

#include <stdio.h>
#include "y.tab.h"

extern int yylex();
extern int yylineno;
extern char *yytext;
extern int yyleng;

int main(void)
{
    FILE *fp = fopen("token.txt" ,"w");
    int ntoken, vtoken ,i = 0;

    ntoken = yylex();
    while(ntoken)
```

```
    {
        fprintf(fp, "%s", yytext);
        for(i = 0;i<18-yyleng; i++)
            fprintf(fp, "%c", ' ');
        fprintf(fp, "%d\n", ntoken);
        ntoken = yylex();
    }
    return 0;
}


---------- grammar.y ----------

%token CHAR ELSE FLOAT FOR IF INT RETURN VOID WHILE PTR_OP INC_OP DEC_OP AND_OP
OR_OP LT GT LE_OP GE_OP EQ_OP NE_OP BOOL DO
%token NOT AMP TILDE STAR ASSIGN PROCESSOR LINK CLUSTER JOB MEMORY JOB_ID
FLOPS_REQUIRED DEADLINE
%token AFFINITY ALGO ISA CLOCK_SPEED MEM_REQUIRED ID MEMORY_TYPE SCHEDULER
%token NUM REAL STRING_LITERAL L1_MEMORY L2_MEMORY MEM_SIZE NAME START_POINT
END_POINT BANDWIDTH
%token CHANNEL_CAPACITY TOPOLOGY LINK_CAPACITY LINK_BANDWIDTH GET_AVAILABLE_MEMORY
GET_MEMORY IS_RUNNING SUBMIT_JOBS
%token GET_CLOCK_SPEED DISCARD_JOB RUN OPEN_BRACKET CLOSED_BRACKET OPEN_CURLY
CLOSED_CURLY OPEN_SQUARE CLOSED_SQUARE
%token SEMI COMMA DOT PLUS MINUS DIVIDE MODULUS PIPE XOR QUES COLON SCHED


%{
  extern int yylex();
  extern int yyerror(char *);
  int yydebug=1;
%}

%%

statement_list
: statement statement_list
| /* EPSILON */
;

statement
: expression_statement
| selection_statement
| iteration_statement
| OPEN_CURLY statement_list CLOSED_CURLY
| declaration
;

declaration
: type_specifier declarator ASSIGN assignment_expression SEMI
;

declarator
: ID
| declarator OPEN_SQUARE CLOSED_SQUARE
;

type_specifier
: VOID
| CHAR
| INT
| FLOAT
| BOOL
| PROCESSOR
| LINK
| CLUSTER
```

```
| JOB
| MEMORY
;

primary_expression
: ID
| NUM
| REAL
| STRING_LITERAL
| OPEN_BRACKET expression CLOSED_BRACKET
| constructor
| mem_func
;

postfix_expression
: primary_expression
| postfix_expression OPEN_SQUARE expression CLOSED_SQUARE
| postfix_expression OPEN_BRACKET CLOSED_BRACKET
| postfix_expression OPEN_BRACKET argument_expression_list CLOSED_BRACKET
| postfix_expression DOT ID
| ID DOT mem_func
| postfix_expression PTR_OP ID
| postfix_expression INC_OP
| postfix_expression DEC_OP
| OPEN_CURLY initializer_list CLOSED_CURLY
| OPEN_SQUARE initializer_list CLOSED_SQUARE
;

initializer
: OPEN_CURLY initializer_list CLOSED_CURLY
| OPEN_SQUARE initializer_list CLOSED_SQUARE
| assignment_expression
;

initializer_list
: initializer
| initializer_list COMMA initializer
;

argument_expression_list
: assignment_expression
| argument_expression_list COMMA assignment_expression
;

unary_expression
: postfix_expression
| INC_OP unary_expression
| DEC_OP unary_expression
| unary_operator unary_expression
;

unary_operator
: AMP
| STAR
| PLUS
| MINUS
| TILDE
| NOT
;

multiplicative_expression
: unary_expression
| multiplicative_expression STAR unary_expression
| multiplicative_expression DIVIDE unary_expression
| multiplicative_expression MODULUS unary_expression
;
```

```
additive_expression
: multiplicative_expression
| additive_expression PLUS multiplicative_expression
| additive_expression MINUS multiplicative_expression
;

relational_expression
: additive_expression
| relational_expression LT additive_expression
| relational_expression GT additive_expression
| relational_expression LE_OP additive_expression
| relational_expression GE_OP additive_expression
;

equality_expression
: relational_expression
| equality_expression EQ_OP relational_expression
| equality_expression NE_OP relational_expression
;

and_expression
: equality_expression
| and_expression AMP equality_expression
;

exclusive_or_expression
: and_expression
| exclusive_or_expression XOR and_expression
;

inclusive_or_expression
: exclusive_or_expression
| inclusive_or_expression PIPE exclusive_or_expression
;

logical_and_expression
: inclusive_or_expression
| logical_and_expression AND_OP inclusive_or_expression
;

logical_or_expression
: logical_and_expression
| logical_or_expression OR_OP logical_and_expression
;

conditional_expression
: logical_or_expression
| logical_or_expression QUES expression COLON conditional_expression
;

assignment_expression
: conditional_expression
| unary_expression ASSIGN assignment_expression
;

expression
: assignment_expression
| expression COMMA assignment_expression
;

iteration_statement
: WHILE OPEN_BRACKET expression CLOSED_BRACKET statement
| DO statement WHILE OPEN_BRACKET expression CLOSED_BRACKET SEMI
| FOR OPEN_BRACKET expression_statement expression_statement CLOSED_BRACKET
statement
```

```
| FOR OPEN_BRACKET expression_statement expression_statement expression
CLOSED_BRACKET statement
| FOR OPEN_BRACKET declaration expression_statement CLOSED_BRACKET statement
| FOR OPEN_BRACKET declaration expression_statement expression CLOSED_BRACKET
statement
;

selection_statement
: IF OPEN_BRACKET expression CLOSED_BRACKET statement ELSE statement
| IF OPEN_BRACKET expression CLOSED_BRACKET statement
;

expression_statement
: SEMI
| expression SEMI
;

assign_colon
: ASSIGN
| COLON
;

constructor
: PROCESSOR OPEN_BRACKET param_list_proc CLOSED_BRACKET
| MEMORY OPEN_BRACKET param_list_mem CLOSED_BRACKET
| JOB OPEN_BRACKET param_list_job CLOSED_BRACKET
| LINK OPEN_BRACKET param_list_link CLOSED_BRACKET
| CLUSTER OPEN_BRACKET param_list_cluster CLOSED_BRACKET
| SCHEDULER OPEN_BRACKET param_list_scheduler CLOSED_BRACKET
;

param_list_job
: job_param1 COMMA job_param2 COMMA job_param3 COMMA job_param4 COMMA job_param5
;

job_param1
: JOB_ID assign_colon assignment_expression
| assignment_expression
;

job_param2
: FLOPS_REQUIRED assign_colon assignment_expression
| assignment_expression
;

job_param3
: DEADLINE assign_colon assignment_expression
| assignment_expression
;

job_param4
: MEM_REQUIRED assign_colon assignment_expression
| assignment_expression
;

job_param5
: AFFINITY assign_colon assignment_expression
| assignment_expression
;

param_list_proc
: proc_param1 COMMA proc_param2 COMMA proc_param3 opt_proc_param4 opt_proc_param5
opt_proc_param6

;
```

```
proc_param1
: ISA assign_colon assignment_expression
| assignment_expression
;

proc_param2
: CLOCK_SPEED assign_colon assignment_expression
| assignment_expression
;

proc_param3
: L1_MEMORY assign_colon assignment_expression
| assignment_expression
| L1_MEMORY assign_colon assignment_expression COMMA
| assignment_expression COMMA
;

opt_proc_param4
: L2_MEMORY assign_colon assignment_expression
| L2_MEMORY assign_colon assignment_expression COMMA
| assignment_expression
| assignment_expression COMMA
| /* EPSILON */
;

opt_proc_param5
: SCHED assign_colon assignment_expression
| assignment_expression
| SCHED assign_colon assignment_expression COMMA
| assignment_expression COMMA
| /* EPSILON */
;

opt_proc_param6
: NAME assign_colon assignment_expression
| assignment_expression
| /* EPSILON */
;

param_list_mem
: mem_param1 COMMA mem_param2 opt_mem_param3
;

mem_param1
: MEMORY_TYPE assign_colon assignment_expression
| assignment_expression
;

mem_param2
: MEM_SIZE assign_colon assignment_expression
| assignment_expression
| MEM_SIZE assign_colon assignment_expression COMMA
| assignment_expression COMMA
;

opt_mem_param3
: NAME assign_colon assignment_expression
| assignment_expression
| /* EPSILON */
;

param_list_link
: link_param1 COMMA link_param2 COMMA link_param3 COMMA link_param4 opt_link_param5
;

link_param1
```

```
: START_POINT assign_colon assignment_expression
| assignment_expression
;

link_param2
: END_POINT assign_colon assignment_expression
| assignment_expression
;

link_param3
: BANDWIDTH assign_colon assignment_expression
| assignment_expression
;

link_param4
: CHANNEL_CAPACITY assign_colon assignment_expression
| assignment_expression
| CHANNEL_CAPACITY assign_colon assignment_expression COMMA
| assignment_expression COMMA
;

opt_link_param5
: NAME assign_colon assignment_expression
| assignment_expression
| /* EPSILON */
;

param_list_cluster
: cluster_param1 COMMA cluster_param2 COMMA cluster_param3 COMMA cluster_param4
opt_cluster_param5 opt_cluster_param6
;

cluster_param1
: assignment_expression;

cluster_param2
: TOPOLOGY assign_colon assignment_expression
| assignment_expression
;

cluster_param3
: LINK_BANDWIDTH assign_colon assignment_expression
| assignment_expression
;

cluster_param4
: LINK_CAPACITY assign_colon assignment_expression
| assignment_expression
| LINK_CAPACITY assign_colon assignment_expression COMMA
| assignment_expression COMMA
;

opt_cluster_param5
: SCHED assign_colon assignment_expression
| assignment_expression
| SCHED assign_colon assignment_expression COMMA
| assignment_expression COMMA
| /* EPSILON */
;

opt_cluster_param6
: NAME assign_colon assignment_expression
| /* EPSILON */
;

param_list_scheduler
```

```
: schedule_param1 opt_schedule_param2
;

schedule_param1
: ALGO assign_colon assignment_expression
| assignment_expression
| ALGO assign_colon assignment_expression COMMA
| assignment_expression COMMA
;

opt_schedule_param2
: NAME assign_colon assignment_expression
| assignment_expression
| /* EPSILON */
;

mem_func
: GET_AVAILABLE_MEMORY OPEN_BRACKET CLOSED_BRACKET
| GET_MEMORY OPEN_BRACKET CLOSED_BRACKET
| IS_RUNNING OPEN_BRACKET CLOSED_BRACKET
| SUBMIT_JOBS OPEN_BRACKET assignment_expression CLOSED_BRACKET
| GET_CLOCK_SPEED OPEN_BRACKET CLOSED_BRACKET
| RUN OPEN_BRACKET assignment_expression CLOSED_BRACKET
| DISCARD_JOB OPEN_BRACKET assignment_expression CLOSED_BRACKET
;

%%
#include <stdio.h>

extern char yytext[];

int yyerror(char *s){
    fflush(stdout);
    printf("%s\n", s);
}

int main(){
    return yyparse();
}


---------- sample_test.c ----------

job_1 = Job(job_id=1, flops_required = 100, deadline = 200,
            mem_required = 1024, affinity = [0.2,0.5,1,2]);

job_2 = Job(job_id=2, flops_required = 5, deadline = 20,
            mem_required = 64, affinity = [0.2,0.5,1,2]);

mem1 = Memory(memory_type= 'cache', mem_size=1);
mem2 = Memory(memory_type= 'cache', mem_size=2);
mem3 = Memory(memory_type= 'cache', mem_size=2);

proc_1 = Processor(isa = 'ARM', clock_speed : 40, l1_memory = mem1,
                   sched = Scheduler(algo = "SJF", name = "my_sched_sjf"));

proc_2 = Processor(isa = 'AMD', clock_speed : 78, l1_memory = mem2);
proc_3 = Processor(isa = 'AMD', clock_speed : 78, l1_memory = mem3);

mono_sched = Scheduler(algo = "Monolithic");
cluster_1 = Cluster(processors={proc_2, proc_3},
                    topology = "star", 100, 80, sched = mono_sched, name =
"cluster1");

run(proc_1);
run(cluster_1);
```

CS 347
Assignment 2
Compilers Lab

Anup Agarwal (150101009)
Chandan Satti (150101059)
Surabhi Gupta (150101078)

Department Of Computer Science And Engineering
IIT Guwahati
12 March 2018

Introduction:

Language Supports:

- Iteration statements (for, while, do-while)
- conditional statements (if else)
- nested loops, expressions and conditionals
- arrays
- Type Specifiers : {void, bool, int, float, char, Processor, Link, Cluster, Job, Memory, **Scheduler**}

- Operators:
  - Unary operators (*, +, -, ~, !)
  - Arithmetic (+, -, *, /, %)
  - Relational (<=, >=, >, <, ==, !=)
  - Bitwise Operators (&, |, ^)
  - Logical Operators (&&, ||)
  - Conditional Operator (? :)

Basic Production Rules:

```
statement_list: statement statement_list
| /* EPSILON */

statement: var_decl;
| expression_statement
| selection_statement
| iteration_statement
| ‘{‘ statement_list ‘}’
| declaration
```

The program is essentially a statement_list. Each statement can be either a variable declaration, expression, selection , iteration or a declaration.

statement -> ‘{‘ statement_list ’}’ allows provision for nesting and compound statements.

```
iteration_statement
: WHILE OPEN_BRACKET expression CLOSED_BRACKET statement
| DO statement WHILE OPEN_BRACKET expression CLOSED_BRACKET SEMI
| FOR OPEN_BRACKET expression_statement expression_statement CLOSED_BRACKET statement
| FOR OPEN_BRACKET expression_statement expression_statement expression CLOSED_BRACKET
statement
| FOR OPEN_BRACKET declaration expression_statement CLOSED_BRACKET statement
| FOR OPEN_BRACKET declaration expression_statement expression CLOSED_BRACKET
statement

selection_statement
: IF OPEN_BRACKET expression CLOSED_BRACKET statement ELSE statement
| IF OPEN_BRACKET expression CLOSED_BRACKET statement
```

Iteration_statements deals with loops in which we have done the while, do while and the for loop. Multiple declarations for the FOR loops are cases in which FOR() has 2 or 3 arguments and the first argument can also be a declaration. (2nd argument is the conditional and third is the iteration)
Also there are the selection productions i.e. the if else statements.

```
declaration: type_specifier declarator ASSIGN assignment_expression
SEMI

declarator: ID
     | declarator OPEN_SQUARE CLOSED_SQUARE
```

These productions provide for declarations for variables and arrays.
type_specifier takes one of the types described in the introduction slide.

Operator Precedence Explanation:

For each precedence level a non terminal is made:

- Operators in precedence:
  - Unary operators (*, +, -, ~, !) : unary_expression
  - Multiplicative (*, /, %) : multiplicative_expression
  - Additive (+, -) : additive_expression
  - Relational (<=, >=, >, <) : relational_expression
  - Equality (==, !=) : equality_expression
  - Bitwise Operators (&, ^, |) : {and, exclusive_or, inclusive_or}_expression
  - Logical Operators (&&, ||) : logical_{and, or}_expression
  - Conditional Operator (? :) : conditional_expression

Operator Precedence Explanation:

For each non terminal productions are added as:

```
lower
: higher | lower OPERATOR higher
```

For e.g:

```
additive_expression
: multiplicative_expression
| additive_expression PLUS multiplicative_expression
| additive_expression MINUS multiplicative_expression
```

Some other important productions:

```
postfix_expression
: postfix_expression OPEN_SQUARE expression CLOSED_SQUARE (array accesses)
| postfix_expression OPEN_BRACKET argument_expression_list CLOSED_BRACKET
(function calls)
| postfix_expression DOT ID (class data member access)
| ID DOT mem_func (inbuilt member function call)
| postfix_expression PTR_OP ID (pointer dereferencing)
| OPEN_CURLY initializer_list CLOSED_CURLY
| OPEN_SQUARE initializer_list CLOSED_SQUARE
;

initializer
: OPEN_CURLY initializer_list CLOSED_CURLY
| OPEN_SQUARE initializer_list CLOSED_SQUARE
| assignment_expression
;
```

```
initializer_list
: initializer
| initializer_list COMMA
initializer
;

argument_expression_list
: assignment_expression
| argument_expression_list
COMMA assignment_expression
;
```

Overview of the Classes (Jobs, Links…)

Each of the custom type specifier is interpreted as a class, having a constructor, member functions and parameters.

The constructor support optional parameters as well as parameters passed by referencing the name of the formal parameter (i.e. `func(name = “xyz”))`

The constructor is of the form:

```
constructor: PROCESSOR OPEN_BRACKET param_list_proc
CLOSED_BRACKET
```

NOTE: ‘primary_expression’ derives ‘constructor’ which allows for in place declarations in the function calls and constructors.

Implementing Optional Parameters

```
param_list_cluster : cluster_param1 COMMA
cluster_param2 COMMA cluster_param3 COMMA
cluster_param4 opt_cluster_param5 opt_cluster_param6

cluster_param1: assignment_expression

cluster_param2: TOPOLOGY assign_colon
assignment_expression
| assignment_expression

cluster_param3
: LINK_BANDWIDTH assign_colon assignment_expression
| assignment_expression
```

Have a non terminal for each parameter, the ones which are optional have EPSILON productions and the ones before optional parameters have optional COMMAS

NOTE: all type checks will be done at time of Semantic Analysis

```
cluster_param4: LINK_CAPACITY assign_colon
assignment_expression
| assignment_expression
| LINK_CAPACITY assign_colon
assignment_expression COMMA
| assignment_expression COMMA

opt_cluster_param5
: SCHED assign_colon assignment_expression
| assignment_expression
| SCHED assign_colon assignment_expression
COMMA
| assignment_expression COMMA
| /* EPSILON */

opt_cluster_param6
: NAME assign_colon assignment_expression
| /* EPSILON */
```

Parsing for Scheduler

The schedulers are of 2 types: distributed schedulers, individual processor scheduler.

User defines an object of class Scheduler as:

```
Scheduler(algo = (string or function pointer), name = (string))
```

If algo is a string then it is a well known scheduler (eg: SJF, FCFS, RR; Monolithic, YARN, Mesos etc.)

Based on the string value or the prototype of the function the type of the scheduler is determined.
NOTE: The constructor is overloaded, viz.

```
Scheduler(string, void nextJobAction(proc_list, job_list))
Scheduler(string, void nextJobAction(cluster_list))
```

This scheduler is an optional parameter for the classes: Processor and Cluster.

When run will be invoked, the appropriate schedulers will be used.