CS347 Assignment 2 : Extension of C language

Lex code:

```
%option noyywrap
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void comment(void);
%}

D           [0-9]
L           [a-zA-Z_]

%%
"/*"              { comment(); }
" //".*$              { /* eat single line comments */}

"break"           { ECHO; printf("\t<--BREAK\n"); }
"char"            { ECHO; printf("\t<--CHAR\n"); }
"const"           { ECHO; printf("\t<--CONST\n"); }
"continue"        { ECHO; printf("\t<--CONTINUE\n"); }
"do"              { ECHO; printf("\t<--DO\n"); }
"double"          { ECHO; printf("\t<--DOUBLE\n"); }
"else"            { ECHO; printf("\t<--ELSE\n"); }
"float"           { ECHO; printf("\t<--FLOAT\n"); }
"for"             { ECHO; printf("\t<--FOR\n"); }
"if"              { ECHO; printf("\t<--IF\n"); }
"int"             { ECHO; printf("\t<--INT\n"); }
"long"            { ECHO; printf("\t<--LONG\n"); }
"return"          { ECHO; printf("\t<--RETURN\n"); }
"short"           { ECHO; printf("\t<--SHORT\n"); }
"signed"          { ECHO; printf("\t<--SIGNED\n"); }
"unsigned"        { ECHO; printf("\t<--UNSIGNED\n"); }
"while"           { ECHO; printf("\t<--WHILE\n"); }

    /*-------------------extended language keywords----------*/
name                  {ECHO; printf("\t<--P_NAME\n");}

    /*--------------------------PROCESSOR CLASS-----------------------------*/
Processor             {ECHO; printf("\t<--C_PROC\n");}
isa                        {ECHO; printf("\t<--P_ISA\n");}
```

```
clock_speed           {ECHO; printf("\t<--P_CLOCK_SPEED\n");}
l1_memory        {ECHO; printf("\t<--P_MEM_OBJ_1\n");}
l2_memory        {ECHO; printf("\t<--P_MEM_OBJ_2\n");}

is_running            {ECHO; printf("\t<--MF_IS_RUNNING\n");}
submit_jobs           {ECHO; printf("\t<--MF_SUBMIT_JOBS\n");}
get_clock_speed       {ECHO; printf("\t<--MF_GET_CS\n");}
run                   {ECHO; printf("\t<--MF_RUN\n");}
discard_job           {ECHO; printf("\t<--MF_DISCARD_JOB\n");}

    /*---------------------LINK CLASS ------------------------*/
Link                  {ECHO; printf("\t<--C_LINK\n");}
start_point      {ECHO; printf("\t<--P_START_POINT\n");}
end_point             {ECHO; printf("\t<--P_END_POINT\n");}
bandwidth             {ECHO; printf("\t<--P_BANDWIDTH\n");}
channel_capacity     {ECHO; printf("\t<--P_CHANNEL_CAP\n");}

    /*--------------------MEMORY CLASS-------------------*/
Memory                {ECHO; printf("\t<--C_Memory\n");}
memory_type           {ECHO; printf("\t<--P_MEM_TYPE\n");}
mem_size              {ECHO; printf("\t<--P_MEM_SIZE\n");}

    /*--------------------JOB CLASS---------------------*/
Job                   {ECHO; printf("\t<--C_JOB\n");}
job_id                {ECHO; printf("\t<--P_JOB_ID\n");}
flops_required        {ECHO; printf("\t<--P_FLOPS_REQ\n");}
deadline              {ECHO; printf("\t<--P_DEADLINE\n");}
mem_required     {ECHO; printf("\t<--P_MEM_REQ\n");}
affinity              {ECHO; printf("\t<--P_AFFINITY\n");}

get_memory            {ECHO; printf("\t<--MF_GET_MEMORY\n");}

    /*---------------CLUSTER CLASS----------------*/
Cluster               {ECHO; printf("\t<--C_CLUSTER\n");}
processors            {ECHO; printf("\t<--P_PROCESSORS\n");}
topology              {ECHO; printf("\t<--C_TOPOLOGY\n");}
link_bandwidth        {ECHO; printf("\t<--C_LINK_BANDW\n");}
link_capacity         {ECHO; printf("\t<--C_LINK_CAP\n");}

{L}({L}|{D})*         { ECHO; printf("\t<--IDENTIFIER\n"); }

{D}+        { ECHO; printf("\t<--INTEGER\n"); }
({D}+"."{D}*)|({D}*"."{D}+)      { ECHO; printf("\t<--FLOAT\n"); }

L?\"(\\.|[^\\"\n])*\"    { ECHO; printf("\t<--STRING_LITERAL\n"); }
L?\'(\\.|[^\\"\n])*\'    { ECHO; printf("\t<--STRING_LITERAL\n"); }
```

```
"+="            { ECHO; printf("\t<--ADD_ASSIGN\n"); }
"-="            { ECHO; printf("\t<--SUB_ASSIGN\n"); }
"*="            { ECHO; printf("\t<--MUL_ASSIGN\n"); }
"/="            { ECHO; printf("\t<--DIV_ASSIGN\n"); }
"%="            { ECHO; printf("\t<--MOD_ASSIGN\n"); }
"&="            { ECHO; printf("\t<--AND_ASSIGN\n"); }
"^="            { ECHO; printf("\t<--XOR_ASSIGN\n"); }
"|="            { ECHO; printf("\t<--OR_ASSIGN\n"); }
"++"            { ECHO; printf("\t<--INC_OP\n"); }
"--"            { ECHO; printf("\t<--DEC_OP\n"); }
"->"            { ECHO; printf("\t<--PTR_OP\n"); }
"&&"            { ECHO; printf("\t<--AND_OP\n"); }
"||"            { ECHO; printf("\t<--OR_OP\n"); }
"<="            { ECHO; printf("\t<--LE_OP\n"); }
">="            { ECHO; printf("\t<--GE_OP\n"); }
"=="            { ECHO; printf("\t<--EQ_OP\n"); }
"!="            { ECHO; printf("\t<--NE_OP\n"); }
";"             { ECHO; printf("\t<--SEMICOLON\n"); }
"{"             { ECHO; printf("\t<--LCB\n"); /*LEFT CURLY BRACKETS*/}
"}"             { ECHO; printf("\t<--RCB\n"); }
","             { ECHO; printf("\t<--COMMA\n"); }
":"             { ECHO; printf("\t<--COLON\n"); }
"="             { ECHO; printf("\t<--ASSIGN\n"); }
"("             { ECHO; printf("\t<--LP\n"); }
")"             { ECHO; printf("\t<--RP\n"); }
"["             { ECHO; printf("\t<--LSB\n"); /*LEFT SQUARE BRACKETS*/}
"]"             { ECHO; printf("\t<--RSB\n"); }
"."             { ECHO; printf("\t<--DOT\n"); }
"&"             { ECHO; printf("\t<--BIT_AND\n"); }
"!"             { ECHO; printf("\t<--NOT\n"); }
"-"             { ECHO; printf("\t<--MINUS\n"); }
"+"             { ECHO; printf("\t<--PLUS\n"); }
"*"             { ECHO; printf("\t<--TIMES\n"); }
"/"             { ECHO; printf("\t<--DIV\n"); }
"%"             { ECHO; printf("\t<--PERCENT\n"); }
"<"             { ECHO; printf("\t<--LT\n"); }
">"             { ECHO; printf("\t<--GT\n"); }
"^"             { ECHO; printf("\t<--CARET\n"); }
"|"             { ECHO; printf("\t<--BIT_OR\n"); }
"?"             { ECHO; printf("\t<--QUESTION_MARK\n"); }

[ \t\v\n\f]     { ; }
.           { printf("%s <--Unmatched character(s)\n",yytext);}
```

```
%%

void comment(void)
{
    char c, prev = 0;

    while ((c = input()) != 0)        /* (EOF maps to 0) */
    {
      if (c == '/' && prev == '*')
           return;
      prev = c;
    }
    printf("ERROR>> unterminated comment!\n");
}
```

Grammar Description for the given language

Note: space character between non terminals is only to distinguish between their names, it is not as a terminal.

Along with the grammar for our subset of the C language, following will be the extension to it's grammar:

```
EXPRESSION → PROCESSOR | CLUSTER | LINK  | MEMORY | JOB | IS_RUNNING | SUBMIT_JOBS | GCS | GAV
| RUN | DJ | GM |  OTHER

Detailed grammar of our subset of C language is given at the bottom of this document.
```

Common Rules

```
ALPHA → [a-zA-Z]
DIGIT → [0-9]
ID -> ALPHA(ALPHA|DIGIT|_)*
ID_ARRAY → ID,ID_ARRAY | ID
OPTIONAL_SIGN → + | - | ε
INT → DIGIT INT | DIGIT
SIGNED_INT → OPTIONAL_SIGN INT
FLOAT → SIGNED_INT | SIGNED_INT . INT
FLOAT_ARRAY → FLOAT , FLOAT_ARRAY | FLOAT

// Escaping double quote inside STRINGDBL
inSTRINGDBL -> (\\.|[^\\"])
// Escaping single quote inside STRINGSGL
inSTRINGSGL -> (\\.|[^\\'])
STRINGDBL -> inSTRINGDBL STRINGDBL | inSTRINGDBL
STRINGSGL -> inSTRINGSGL STRINGSGL | inSTRINGSGL

//QSTRING=Quoted String
QSTRING → "STRINGDBL" | "STRINGSGL" | "" | ε
```

```
Comma Separated String Arguments-
CSTRINGS → ,QSTRING | ,name =QSTRING |,name = None | ε
```

For making the grammar, terminals were chosen such that they would not depend on the users input.
In Processor class, three main non-terminals and one extra non-terminals are there. PI(processor isa), PCS(processor clock speed)and PL1(processor l1_memory) are used to scan the corresponding parameters as input. The P_OPTIONAL non terminal handles the case where l2_memory maybe added. Also the CSTRING non-terminal is present at ending of all P_OPTIONAL productions which correspond to take the optional name which may be entered by the user. QSTRING is the non-terminal which produces all the strings inside double quotes(“”) or single quotes(‘’)(* special case added to handle some example present in the pdf). FLOAT is the non-terminal to produce all the floating point numbers(both signed and unsigned)

Processor Class

```
PROCESSOR → Processor(PI,PCS,PL1 P_OPTIONAL);
PI → isa=QSTRING
PCS → clock_speed : FLOAT
PL1 → l1_memory=L1
L1 → ID | MEMORY
P_OPTIONAL → ,l2_memory=L1 CSTRINGS | ,l2_memory=None CSTRINGS | CSTRINGS
```

The member functions include the productions for the member functions submit_jobs() , get_clock_speed() , run() and discard_jobs() where an id or an array of ids can be passed using the production of non-terminal ID_ARRAY.

Member Functions

```
IS_RUNNING → ID.is_running();
SUBMIT_JOBS → submit_jobs(ID_ARRAY);
//GCS=Get Clock Speed
GCS → ID.get_clock_speed();
// The Member function ‘run’
RUN → run(ID_ARRAY);
//DJ=discard job
DJ → ID.discard_job(ID);
```

Link Class

```
LINK → Link(LSP,LEP,LB, FLOAT CSTRINGS);
LSP → start_point=QSTRING
LEP → end_point=QSTRING
LB → bandwidth=FLOAT
Memory Class
MEMORY → Memory(memory_type=QSTRING, mem_size=INT CSTRINGS);
```

Member Functions

```
GAV → ID.get_available_memory();
```

Job Class

```
JOB → Job(JID , JFR , JD , JM , JAF);
JID → job_id=INT
JFR → flops_required=FLOAT
JD → deadline=FLOAT
JM → mem_required=INT
```

```
JAF → affinity=AF
// array of floats can also be passed as a variable
AF → ID |  [FLOAT, FLOAT, FLOAT, FLOAT]
```

Member Functions

```
//GM=get memory
GM → ID.get_memory();
```

Cluster Class

```
CLUSTER → Cluster(CP , CT , CLB , CLC CSTRINGS);
CP → processors=CLUSTER_ARRAY
CT → topology=QSTRING
CLB → link_bandwidth=FLOAT
CLC → link_capacity=FLOAT
CLUSTER_ARRAY → ID | [ID_ARRAY]
```

Grammar for subset of C language-

```
statement
    : labeled_statement
    | compound_statement
    | expression_statement
    | selection_statement
    | iteration_statement
    | jump_statement
    ;

selection_statement
    : IF '(' expression ')' statement
    | IF '(' expression ')' statement ELSE statement


iteration_statement
    : WHILE '(' expression ')' statement
    | FOR '(' expression_statement expression_statement ')' statement
    | FOR '(' expression_statement expression_statement expression ')' statement
    | FOR '(' declaration expression_statement ')' statement
    | FOR '(' declaration expression_statement expression ')' statement

expression
    : assignment_expression
    | expression ',' assignment_expression
    | PROCESSOR
    | CLUSTER
    | LINK
    | MEMORY
    | JOB
    | IS_RUNNING
```

```
    | SUBMIT_JOBS
    | GCS
    | GAV
    | RUN
    | DJ
    | GM
    ;

assignment_expression
    : conditional_expression
    | unary_expression assignment_operator assignment_expression
    ;

conditional_expression
    : logical_or_expression    ;


expression_statement
: ';'
| expression ';'
;

declaration
    : type_specifier ';'
    | type_specifier init_declarator_list ';'
    ;

type_specifier
    : VOID
    | CHAR
    | SHORT
    | INT
    | LONG
    | FLOAT
    | DOUBLE
    ;
init_declarator_list
    : init_declarator
    | init_declarator_list ',' init_declarator
    ;

init_declarator
    : declarator
    | declarator '=' initializer
    ;

argument_expression_list
    : assignment_expression
    | argument_expression_list ',' assignment_expression
    ;
```

```
primary_expression
    : ID
    | CONSTANT
    | STRING_LITERAL
    | '(' expression ')'
    ;


postfix_expression
    : primary_expression
    | postfix_expression '[' expression ']'
    | postfix_expression '(' ')'
    | postfix_expression '(' argument_expression_list ')'
    | postfix_expression '.' ID
    | postfix_expression INC_OP
    | postfix_expression DEC_OP
    ;

multiplicative_expression
    : postfix_expression
    | multiplicative_expression '*' postfix_expression
    | multiplicative_expression '/' postfix_expression
    | multiplicative_expression '%' postfix_expression
    ;

additive_expression
    : multiplicative_expression
    | additive_expression '+' multiplicative_expression
    | additive_expression '-' multiplicative_expression
    ;

relational_expression
    : additive_expression
    | relational_expression '<' additive_expression
    | relational_expression '>' 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 '&' equality_expression
    ;

inclusive_or_expression
```

```
    : and_expression
    | inclusive_or_expression '|' 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
    ;
```