```
/* table.h */

#define         BREAK                        0
#define   CHAR                               1
#define   CONTINUE                           2
#define   ELSE                               3
#define   FLOAT                              4
#define   FOR                                5
#define   IF                                 6
#define   INT                                7
#define   RETURN                             8
#define   VOID                               9
#define   WHILE                              10
#define   PROC                               11
#define   LNK                                12
#define   JB                                 13
#define   CLUST                              14
#define   CLUSTER                            15
#define   PROCESSOR                          16
#define   ISA                                17
#define   PROC_TYPE                          18
#define   CLOCK_SPEED                        19
#define   MEM1                               20
#define   MEM2                               21
#define   NAME                               22
#define   TOPOLOGY                           23
#define   LINK_BANDWIDTH                     24
#define   LINK_CAPACITY                      25
#define   LINK                               26
#define   START_POINT                        27
#define   END_POINT                          28
```

```
#define    MEMORY_TYPE                 29
#define    MEM_TYPE                    30
#define    MEMORY_SIZE                 31
#define    JOB                         32
#define    JOB_ID                      33
#define    FLOPS_REQUIRED              34
#define    DEADLINE                    35
#define    MEM_REQUIRED                36
#define    AFFINITY                    37
#define    RUN                         38
#define    WAIT                        39
#define    DISCARD_JOB                 40
#define    STOP                        41
#define    GET_AVAILABLE_MEMORY        42
#define    GET_JOB_AFFINITY            43
#define    GET_JOB_MEMORY              44
#define    GET_FLOPS                   45
#define    GET_DEADLINE                46
#define    IS_RUNNING                  47
#define    SUBMIT_JOBS                 48
#define    GET_FLOPS_SPEED             49
#define    GET_PROC_TYPE               50
#define    IS_PROCESSOR                51
#define    GET_PROCESSOR               52
#define    MEM                         53
#define    IDENTIFIER                  54
#define    CONSTANT                    55
#define    STRING_LITERAL              56
#define    RIGHT_OP                    57
#define    LEFT_OP                     58
#define    INC_OP                      59
#define    DEC_OP                      60
```

```
#define    DREF_OP                        61
#define    AND_OP                         62
#define    OR_OP                          63
#define    LE_OP                          64
#define    GE_OP                          65
#define    EQ_OP                          66
#define    NE_OP                          67
#define    SEMI_COLON                     68
#define    LEFT_CURLY                     69
#define    RIGHT_CURLY                    70
#define    COMMA                          71
#define    ASGN_OP                        72
#define    LEFT_PARENTEHSIS               73
#define    RIGHT_PARENTHESIS              74
#define    LEFT_BRACKET                   75
#define    RIGHT_BRACKET                  76
#define    DOT                            77
#define    AMPERSAND                      78
#define    NOT_OP                         79
#define    BTW_NOT                        80
#define    MINUS                          81
#define    PLUS                           82
#define    MUL_OP                         83
#define    DIV_OP                         84
#define    MOD_OP                         85
#define    LESS_THAN                      86
#define    GREATER_THAN                   87
#define    XOR_OP                         88
#define    BTW_OR                         89
#define    INVALID                        90
#define    MEMORY                         91
#define    PROCESSORS                     92
```

```
/* grammar.lex*/


D                   [0-9]
L                   [a-zA-Z_]
H                   [a-fA-F0-9]
E                   [Ee][+-]?{D}+
FS                  (f|F|l|L)
IS                  (u|U|l|L)*

%{
#include <stdio.h>
#include "table.h"

void count();
%}

%%

"break"                                { printf("<"); count(); printf(",%s> ","BREAK"); return(BREAK);}
"char"                                 { printf("<"); count(); printf(",%s> ","CHAR"); return(CHAR);}
"continue"                             { printf("<"); count(); printf(",%s> ","CONTINUE"); return(CONTINUE);}
"else"                                 { printf("<"); count(); printf(",%s> ","ELSE"); return(ELSE);}
"float"                                { printf("<"); count(); printf(",%s> ","FLOAT"); return(FLOAT);}
"for"                                  { printf("<"); count(); printf(",%s> ","FOR"); return(FOR);}
"if"                                   { printf("<"); count(); printf(",%s> ","IF"); return(IF);}
"int"                                  { printf("<"); count(); printf(",%s> ","INT"); return(INT);}
"return"                               { printf("<"); count(); printf(",%s> ","RETURN"); return(RETURN);}
"void"                                 { printf("<"); count(); printf(",%s> ","VOID"); return(VOID);}
"while"                                { printf("<"); count(); printf(",%s> ","WHILE"); return(WHILE);}
"proc"                                 { printf("<"); count(); printf(",%s> ","PROC"); return(PROC);}
"lnk"                                  { printf("<"); count(); printf(",%s> ","LNK"); return(LNK);}
```

```
"jb"                              { printf("<"); count(); printf(",%s> ","JB"); return(JB);}
"clust"                           { printf("<"); count(); printf(",%s> ","CLUST"); return(CLUST);}
"Cluster"                        { printf("<"); count(); printf(",%s> ","CLUSTER"); return(CLUSTER);}
"Processor"                       { printf("<"); count(); printf(",%s> ","PROCESSOR"); return(PROCESSOR);}
"processors"             { printf("<"); count(); printf(",%s> ","PROCESSORS"); return(PROCESSORS);}
"isa"                             { printf("<"); count(); printf(",%s> ","ISA"); return(ISA);}
('ARM')|('AMD')|('CDC')|('MIPS')  { printf("<"); count(); printf(",%s> ","PROC_TYPE"); return(PROC_TYPE);}
"clock_speed"                     { printf("<"); count(); printf(",%s> ","CLOCK_SPEED"); return(CLOCK_SPEED);}
"l1_memory"              { printf("<"); count(); printf(",%s> ","MEM1"); return(MEM1);}
"l2_memory"              { printf("<"); count(); printf(",%s> ","MEM2"); return(MEM2);}
"name"                            { printf("<"); count(); printf(",%s> ","NAME"); return(NAME);}
"topology"                        { printf("<"); count(); printf(",%s> ","TOPOLOGY"); return(TOPOLOGY);}
"Link_bandwidth"                   { printf("<"); count(); printf(",%s> ","LINK_BANDWIDTH"); return(LINK_BANDWIDTH);}
"link_capacity"                    { printf("<"); count(); printf(",%s> ","LINK_CAPACITY"); return(LINK_CAPACITY);}
"Link"                            { printf("<"); count(); printf(",%s> ","LINK"); return(LINK);}
"start_point"                      { printf("<"); count(); printf(",%s> ","START_POINT"); return(START_POINT);}
"end_point"                       { printf("<"); count(); printf(",%s> ","END_POINT"); return(END_POINT);}
"memory_type"                    { printf("<"); count(); printf(",%s> ","MEMORY_TYPE"); return(MEMORY_TYPE);}
('primary')|('secondary')|('cache') { printf("<"); count(); printf(",%s> ","MEM_TYPE"); return(MEM_TYPE);}
"mem_size"                        { printf("<"); count(); printf(",%s> ","MEMORY_SIZE"); return(MEMORY_SIZE);}
"Job"                             { printf("<"); count(); printf(",%s> ","JOB"); return(JOB);}
"job_id"                         { printf("<"); count(); printf(",%s> ","JOB_ID"); return(JOB_ID);}
"flops_required"                 { printf("<"); count(); printf(",%s> ","FLOPS_REQUIRED"); return(FLOPS_REQUIRED);}
"deadline"                        { printf("<"); count(); printf(",%s> ","DEADLINE"); return(DEADLINE);}
"mem_required"                   { printf("<"); count(); printf(",%s> ","MEM_REQUIRED"); return(MEM_REQUIRED);}
"affinity"                       { printf("<"); count(); printf(",%s> ","AFFINITY"); return(AFFINITY);}
"run"                             { printf("<"); count(); printf(",%s> ","RUN"); return(RUN);}
"wait"                            { printf("<"); count(); printf(",%s> ","WAIT"); return(WAIT);}
"discard_job"                     { printf("<"); count(); printf(",%s> ","DISCARD_JOB"); return(DISCARD_JOB);}
"stop"                           { printf("<"); count(); printf(",%s> ","STOP"); return(STOP);}
"Get_available_memory"            { printf("<"); count(); printf(",%s> ","GET_AVAILABLE_MEMORY"); return(GET_AVAILABLE_MEMORY);}
"get_job_affinity"                { printf("<"); count(); printf(",%s> ","GET_JOB_AFFINITY"); return(GET_JOB_AFFINITY);}
```

```
"get_memory"                { printf("<"); count(); printf(",%s> ","GET_JOB_MEMORY"); return(GET_JOB_MEMORY);}
"get_flops"                 { printf("<"); count(); printf(",%s> ","GET_FLOPS"); return(GET_FLOPS);}
"get_deadline"              { printf("<"); count(); printf(",%s> ","GET_DEADLINE"); return(GET_DEADLINE);}
"is_running"                { printf("<"); count(); printf(",%s> ","IS_RUNNING"); return(IS_RUNNING);}
"submit_jobs"               { printf("<"); count(); printf(",%s> ","SUBMIT_JOBS"); return(SUBMIT_JOBS);}
"get_flops_speed"           { printf("<"); count(); printf(",%s> ","GET_FLOPS_SPEED"); return(GET_FLOPS_SPEED);}
"get_proc_type"             { printf("<"); count(); printf(",%s> ","GET_PROC_TYPE"); return(GET_PROC_TYPE);}
"is_processor"              { printf("<"); count(); printf(",%s> ","IS_PROCESSOR"); return(IS_PROCESSOR);}
"get_processor"             { printf("<"); count(); printf(",%s> ","GET_PROCESSOR"); return(GET_PROCESSOR);}
"Memory"                    { printf("<"); count(); printf(",%s> ","MEMORY"); return(MEMORY);}
"mem"                       { printf("<"); count(); printf(",%s> ","MEM"); return(MEM);}

{L}({L}|{D})*               { printf("<"); count(); printf(",%s> ","IDENTIFIER"); return(IDENTIFIER);}

0[xX]{H}+{IS}?              { printf("<"); count(); printf(",%s> ","CONSTANT"); return(CONSTANT);}
0{D}+{IS}?                  { printf("<"); count(); printf(",%s> ","CONSTANT"); return(CONSTANT);}
{D}+{IS}?                   { printf("<"); count(); printf(",%s> ","CONSTANT"); return(CONSTANT);}
L?'(\\.|[^\\'])+'           { printf("<"); count(); printf(",%s> ","CONSTANT"); return(CONSTANT);}

{D}+{E}{FS}?                { printf("<"); count(); printf(",%s> ","CONSTANT"); return(CONSTANT);}
{D}*"."{D}+({E})?{FS}?      { printf("<"); count(); printf(",%s> ","CONSTANT"); return(CONSTANT);}
{D}+"."{D}*({E})?{FS}?      { printf("<"); count(); printf(",%s> ","CONSTANT"); return(CONSTANT);}

\"(\\.|[^\\"])*\"             { printf("<"); count(); printf(",%s> ","STRING_LITERAL"); return(STRING_LITERAL);}

">>"                        { printf("<"); count(); printf(",%s> ","RIGHT_OP"); return(RIGHT_OP);}
"<<"                        { printf("<"); count(); printf(",%s> ","LEFT_OP"); return(LEFT_OP);}
"++"                        { printf("<"); count(); printf(",%s> ","INC_OP"); return(INC_OP);}
"--"                        { printf("<"); count(); printf(",%s> ","DEC_OP"); return(DEC_OP);}
"->"                        { printf("<"); count(); printf(",%s> ","DREF_OP"); return(DREF_OP);}
"&&"                       { printf("<"); count(); printf(",%s> ","AND_OP"); return(AND_OP);}
"||"                        { printf("<"); count(); printf(",%s> ","OR_OP"); return(OR_OP);}
```

```
"<="                        { printf("<"); count(); printf(",%s> ","LE_OP"); return(LE_OP);}
">="                        { printf("<"); count(); printf(",%s> ","GE_OP"); return(GE_OP);}
"=="                        { printf("<"); count(); printf(",%s> ","EQ_OP"); return(EQ_OP);}
"!="                        { printf("<"); count(); printf(",%s> ","NE_OP"); return(NE_OP);}
";"                         { printf("<"); count(); printf(",%s> ","SEMI_COLON"); return(SEMI_COLON);}
("{"|"<%")                  { printf("<"); count(); printf(",%s> ","LEFT_CURLY"); return(LEFT_CURLY);}
("}"|"%>")                  { printf("<"); count(); printf(",%s> ","RIGHT_CURLY"); return(RIGHT_CURLY);}
","                         { printf("<"); count(); printf(",%s> ","COMMA"); return(COMMA);}
"="                         { printf("<"); count(); printf(",%s> ","ASGN_OP"); return(ASGN_OP);}
":"                         { printf("<"); count(); printf(",%s> ","ASGN_OP"); return(ASGN_OP);}
"("                         { printf("<"); count(); printf(",%s> ","LEFT_PARENTEHSIS"); return(LEFT_PARENTEHSIS);}
")"                         { printf("<"); count(); printf(",%s> ","RIGHT_PARENTHESIS"); return(RIGHT_PARENTHESIS);}
("["|"<:")                  { printf("<"); count(); printf(",%s> ","LEFT_BRACKET"); return(LEFT_BRACKET);}
("]"|":>")                  { printf("<"); count(); printf(",%s> ","RIGHT_BRACKET"); return(RIGHT_BRACKET);}
"."                         { printf("<"); count(); printf(",%s> ","DOT"); return(DOT);}
"&"                         { printf("<"); count(); printf(",%s> ","AMPERSAND"); return(AMPERSAND);}
"!"                         { printf("<"); count(); printf(",%s> ","NOT_OP"); return(NOT_OP);}
"~"                         { printf("<"); count(); printf(",%s> ","BTW_NOT"); return(BTW_NOT);}
"-"                         { printf("<"); count(); printf(",%s> ","MINUS"); return(MINUS);}
"+"                         { printf("<"); count(); printf(",%s> ","PLUS"); return(PLUS);}
"*"                         { printf("<"); count(); printf(",%s> ","MUL_OP"); return(MUL_OP);}
"/"                         { printf("<"); count(); printf(",%s> ","DIV_OP"); return(DIV_OP);}
"%"                         { printf("<"); count(); printf(",%s> ","MOD_OP"); return(MOD_OP);}
"<"                         { printf("<"); count(); printf(",%s> ","LESS_THAN"); return(LESS_THAN);}
">"                         { printf("<"); count(); printf(",%s> ","GREATER_THAN"); return(GREATER_THAN);}
"^"                         { printf("<"); count(); printf(",%s> ","XOR_OP"); return(XOR_OP);}
"|"                         { printf("<"); count(); printf(",%s> ","BTW_OR"); return(BTW_OR);}

[ \t\v\n\f]                 { count();}
.                           { printf("<"); count(); printf(",%s> ","INVALID"); return(INVALID);}

%%
```

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


int column = 0;

void count()
{
    int i;

    for (i = 0; yytext[i] != '\0'; i++)
            if (yytext[i] == '\n')
                    column = 0;
            else if (yytext[i] == '\t')
                    column += 8 - (column % 8);
            else
                    column++;

    ECHO;
}
```