1 Sample 1

- Same token returned for all Keywords
- Same token returned for all Punctuators
- Specific token ID (for Keywords and Punctuators) hard-coded in .y file
- Token ID set to yylval
- Value of constant not computed
- Start condition in Flex to process *comment*
- Newline (\n) treated as white space – line cannot be counted

.l File

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

char com[1001];
%}

SGN [-+]
NZD [1-9]
D [0-9]
ES \\[\'\"\\\?abfnrtv]

IDND [a-zA-Z]
ID {IDND}({IDND}|{D})*

C [^ \'\\\n]|{ES}
CS {C}+
CRC \'{CS}\'

DS {D}+

IC {NZD}|{NZD}{DS}|[0]+

EXP [eE]{SGN}?{DS}
FRC {DS}?\.{DS}|{DS}\.
FC {FRC}{EXP}?|{DS}{EXP}

EC {ID}

CONSTANT {IC}|{FC}|{EC}|{CRC}

SC [^\"\\\n]|{ES}
SCS {SC}+
SL \"{SCS}?\"

WS [ \t\n]

SLC [/][/][^\n]*

%x INCOM
```

```
%%
<INITIAL>{
"/*"    { strcat(com,yytext); BEGIN(INCOM); }
}
<INCOM>{
"*/"    { strcat(com,yytext); printf("< Multi Line Comment , %s >\n",com); BEGIN(INITIAL); }
[^*\n]+ { strcat(com,yytext); }
"*"        { strcat(com,yytext); }
[\n]    { strcat(com,yytext); }
}


{SLC}        { printf("< Single Line Comment , %s >\n",yytext); }

"auto"        { yylval.ival = 1; return KEYWORD; }
"enum"        { yylval.ival = 2; return KEYWORD; }
"restrict"    { yylval.ival = 3; return KEYWORD; }
"unsigned"    { yylval.ival = 4; return KEYWORD; }
"break"        { yylval.ival = 5; return KEYWORD; }
"extern"    { yylval.ival = 6; return KEYWORD; }
"return"    { yylval.ival = 7; return KEYWORD; }
"void"        { yylval.ival = 8; return KEYWORD; }
"case"        { yylval.ival = 9; return KEYWORD; }
"float"        { yylval.ival = 10; return KEYWORD; }
"short"        { yylval.ival = 11; return KEYWORD; }
"volatile"    { yylval.ival = 12; return KEYWORD; }
"char"        { yylval.ival = 13; return KEYWORD; }
"for"        { yylval.ival = 14; return KEYWORD; }
"signed"    { yylval.ival = 15; return KEYWORD; }
"while"        { yylval.ival = 16; return KEYWORD; }
"const"        { yylval.ival = 17; return KEYWORD; }
"goto"        { yylval.ival = 18; return KEYWORD; }
"sizeof"    { yylval.ival = 19; return KEYWORD; }
"_Bool"        { yylval.ival = 20; return KEYWORD; }
"continue"    { yylval.ival = 21; return KEYWORD; }
"if"        { yylval.ival = 22; return KEYWORD; }
"static"    { yylval.ival = 23; return KEYWORD; }
"_Complex"    { yylval.ival = 24; return KEYWORD; }
"default"    { yylval.ival = 25; return KEYWORD; }
"inline"    { yylval.ival = 26; return KEYWORD; }
"struct"    { yylval.ival = 27; return KEYWORD; }
"_Imaginary"    { yylval.ival = 28; return KEYWORD; }
"do"        { yylval.ival = 29; return KEYWORD; }
"int"        { yylval.ival = 30; return KEYWORD; }
"switch"    { yylval.ival = 31; return KEYWORD; }
"double"    { yylval.ival = 32; return KEYWORD; }
"long"        { yylval.ival = 33; return KEYWORD; }
"typedef"    { yylval.ival = 34; return KEYWORD; }
"else"        { yylval.ival = 35; return KEYWORD; }
"register"    { yylval.ival = 36; return KEYWORD; }
"union"        { yylval.ival = 37; return KEYWORD; }
```

```
"["         { yylval.ival = 1; return PUNCTUATOR; }
"]"         { yylval.ival = 2; return PUNCTUATOR; }
"("         { yylval.ival = 3; return PUNCTUATOR; }
")"         { yylval.ival = 4; return PUNCTUATOR; }
"{"         { yylval.ival = 5; return PUNCTUATOR; }
"}"         { yylval.ival = 6; return PUNCTUATOR; }
"."         { yylval.ival = 7; return PUNCTUATOR; }
"->"         { yylval.ival = 8; return PUNCTUATOR; }
"++"         { yylval.ival = 9; return PUNCTUATOR; }
"--"         { yylval.ival = 10; return PUNCTUATOR; }
"&"         { yylval.ival = 11; return PUNCTUATOR; }
"*"         { yylval.ival = 11; return PUNCTUATOR; }
"+"         { yylval.ival = 12; return PUNCTUATOR; }
"-"         { yylval.ival = 13; return PUNCTUATOR; }
"~"         { yylval.ival = 14; return PUNCTUATOR; }
"!"         { yylval.ival = 15; return PUNCTUATOR; }
"/"         { yylval.ival = 16; return PUNCTUATOR; }
"%"         { yylval.ival = 17; return PUNCTUATOR; }
"<<"         { yylval.ival = 18; return PUNCTUATOR; }
">>"         { yylval.ival = 19; return PUNCTUATOR; }
"<"         { yylval.ival = 20; return PUNCTUATOR; }
">"         { yylval.ival = 21; return PUNCTUATOR; }
"<="         { yylval.ival = 22; return PUNCTUATOR; }
">="         { yylval.ival = 23; return PUNCTUATOR; }
"=="         { yylval.ival = 24; return PUNCTUATOR; }
"!="         { yylval.ival = 25; return PUNCTUATOR; }
"^"         { yylval.ival = 26; return PUNCTUATOR; }
"|"         { yylval.ival = 27; return PUNCTUATOR; }
"&&"         { yylval.ival = 28; return PUNCTUATOR; }
"||"         { yylval.ival = 29; return PUNCTUATOR; }
"?"         { yylval.ival = 30; return PUNCTUATOR; }
":"         { yylval.ival = 31; return PUNCTUATOR; }
";"         { yylval.ival = 32; return PUNCTUATOR; }
"..."         { yylval.ival = 33; return PUNCTUATOR; }
"="         { yylval.ival = 34; return PUNCTUATOR; }
"*="         { yylval.ival = 35; return PUNCTUATOR; }
"/="         { yylval.ival = 36; return PUNCTUATOR; }
"%="         { yylval.ival = 37; return PUNCTUATOR; }
"+="         { yylval.ival = 38; return PUNCTUATOR; }
"-="         { yylval.ival = 39; return PUNCTUATOR; }
"<<="         { yylval.ival = 40; return PUNCTUATOR; }
">>="         { yylval.ival = 41; return PUNCTUATOR; }
"&="         { yylval.ival = 42; return PUNCTUATOR; }
"^="         { yylval.ival = 43; return PUNCTUATOR; }
"|="         { yylval.ival = 44; return PUNCTUATOR; }
","         { yylval.ival = 45; return PUNCTUATOR; }
"#"         { yylval.ival = 46; return PUNCTUATOR; }

{SL}         { yylval.text = strdup(yytext); return STRING_LITERAL; }
{ID}         { yylval.text = strdup(yytext); return IDENTIFIER; }
{CONSTANT}     { return CONSTANT; }
{WS}         ;
%%
```

.y File

```
%{ /* C Declarations and Definitions */
extern int yylex();
void yyerror();

#define AUTO 1
#define ENUM 2
#define RESTRICT 3
#define UNSIGNED 4
#define BREAK 5
#define EXTERN 6
#define RETURN 7
#define VOID 8
#define CASE 9
#define FLOAT 10
#define SHORT 11
#define VOLATILE 12
#define CHAR 13
#define FOR 14
#define SIGNED 15
#define WHILE 16
#define CONST 17
#define GOTO 18
#define SIZEOF 19
#define _BOOL 20
#define CONTINUE 21
#define IF 22
#define STATIC 23
#define _COMPLEX 24
#define DEFAULT 25
#define INLINE 26
#define STRUCT 27
#define _IMAGINARY 28
#define DO 29
#define INT 30
#define SWITCH 31
#define DOUBLE 32
#define LONG 33
#define TYPEDEF 34
#define ELSE 35
#define REGISTER 36
#define UNION 37

#define LSB 1
#define RSB 2
#define LRB 3
#define RRB 4
#define LCB 5
#define RCB 6
#define DOT 7
#define ARR 8
#define PLPL 9
#define MIMI 10
```

```
#define AMP 11
#define STAR 12
#define ADD 13
#define SUB 14
#define TILDE 15
#define EXCL 16
#define DIV 17
#define MOD 18
#define LS 19
#define RS 20
#define LT 21
#define GT 22
#define LTE 23
#define GTE 24
#define EQ 25
#define NEQ 26
#define XOR 27
#define PIPE 28
#define AND 29
#define OR 30
#define QT 31
#define COLON 32
#define SCOLON 33
#define ECLIPSIS 34
#define ASS 35
#define MASS 36
#define DASS 37
#define MODASS 38
#define AASS 39
#define SASS 40
#define LSASS 41
#define RSASS 42
#define ANDASS 43
#define XORASS 44
#define ORASS 45
#define COMMA 46
#define HASH 47

%}

%union{
    int ival;
    float fval;
    char *text;
}

%token <ival> KEYWORD
%token <string> IDENTIFIER
%token <ival> CONSTANT
%token <string> STRING_LITERAL
%token <ival> PUNCTUATOR

%type <ival> statement
```

```
%%
statement: {;}
%%

void yyerror()
{
    printf("HEY\n");
}
```

2 Sample 2

- Same token returned for all Keywords
- Same token returned for all Punctuators
- Same token returned for all Constants – value not computed – lexeme returned in yylval (why not yytext?)
- Specific tokens returned as lexeme through yylval
- Specific token IDs not defined and used
- yylval.string = strdup(yytext);
  does the job of
  yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
  strncpy(yylval.string, yytext, yyleng + 1);
- Comment definitions complicated

.l File

```
/* scanner for tinyC*/


%{
   #include <stdlib.h>
   #include <string.h>
   #include "y.tab.h"
   int line_count =0;

%}
DIG               [0-9]
LETTER             [a-zA-Z_]
ENUM             [Ee][+-]?{DIG}+
%option noyywrap
%%


{DIG}                { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                       strncpy(yylval.string, yytext, yyleng + 1);/*char constants*/
                      return(CONSTANT); }
[1-9]{DIG}*            {  yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                     strncpy(yylval.string, yytext, yyleng + 1);
                     return(CONSTANT); }
{DIG}+{ENUM}         {  yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                     strncpy(yylval.string, yytext, yyleng + 1);
                     return(CONSTANT); }
{DIG}*"."{DIG}+({ENUM})?    { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                     strncpy(yylval.string, yytext, yyleng + 1);
                      return(CONSTANT); }
{DIG}+"."{DIG}*({ENUM})?    {  yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                     strncpy(yylval.string, yytext, yyleng + 1);
                     return(CONSTANT); }
"auto"             {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                     strncpy(yylval.string, yytext, yyleng + 1);
                     return(KEYWORD);}
"break"             {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
```

```
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD);}
"case"              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD);}
"const"             {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"continue"          {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"default"           {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"do"                {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"double"            {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"else"              {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"enum"              {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"extern"            {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"float"             {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"for"               {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"goto"              {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"if"                {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"int"               {    yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"long"              {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"register"          {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"return"            {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"short"             {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
```

```
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"signed"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"sizeof"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"static"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"struct"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"switch"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"typedef"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"union"           {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"unsigned"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"void"            {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"volatile"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"while"           {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"_Bool"           {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"_COMPLEX"        {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }
"_IMAGINARY"    {   yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(KEYWORD); }

{LETTER}({LETTER}|{DIG})*   {
                yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                strncpy(yylval.string, yytext, yyleng + 1);
                return(IDENTIFIER);}
\"([^\\\"]|\\.)*\"  { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(STRING_LITERAL); }

"..."            {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
```

```
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
">>="               {      yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"<<="               {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"+="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"-="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                     return(PUNCTUATOR); }
"*="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"/="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"++"              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"--"              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"->"              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"&&"              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"||"              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"<="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
">="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"=="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"!="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"%="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"&="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"^="              {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
```

```
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"|="            {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
">>"            {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"<<"            {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
";"             {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"~"           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"-"           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"+"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
"*"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
"/"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
"%"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
"<"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
">"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
"^"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
"|"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
"?"           { yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR);  }
","           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
":"           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"="           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
```

```
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"("           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
")"           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"."           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"&"           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }
"!"           {     yylval.string=(char *)malloc((yyleng+1)*(sizeof(char)));
                    strncpy(yylval.string, yytext, yyleng + 1);
                    return(PUNCTUATOR); }

[ \t\v\f]    { /* ignore bad characters */ }

.            {}
[/]+.*      { /*ignore single line comments*/}
"/*"([^*]|\*+[^*/])*\*+"/"            {/*ignore multi-line comments*/}
\n           {
               ++line_count;
               return (int)'\n';
            }

%%
```

.y File

```
%{ /* C Declarations and Definitions */
#include <string.h>
#include <stdio.h>
extern int yylex();
void yyerror(char *s);
%}
%union {
int intval;
char* string;
}
%token <intval> KEYWORD
%token <intval> IDENTIFIER
%token <intval> CONSTANT
%token <intval> STRING_LITERAL
%token <intval> PUNCTUATOR
%type <intval> expression
%%
expression : KEYWORD;
%%
void yyerror(char *s) {
printf("%s",s);
}
```

3 Sample 3

- All Keywords clubbed in a single regular definition
- All Punctuators clubbed in a single regular definition
- Specific tokens returned as lexeme through yytext
- Same token returned for all Constants – value not computed – lexeme returned in yytext
- Specific token IDs not defined and used
- Comment handle by simple yet error-prone C function

.l File

```
%{
  #include "y.tab.h"
  void comment();
%}

WS                  [ \n\t]
SIGN                 [+-]
DIGIT                  [0-9]
NON_ZERO_DIGIT      [1-9]
DIGIT_SEQ            {DIGIT}+
ALPHABET            [_a-zA-Z]
ESCAPE_SEQ_C        [\"\?\\\a\b\f\r\t\v]
ESCAPE_SEQ_S        [\'\?\\\a\b\f\r\t\v]
C_CHAR                [^\'\\\n]|{ESCAPE_SEQ_C}
C_CHAR_SEQ            {C_CHAR}+
S_CHAR                [^\"\\\n]|{ESCAPE_SEQ_S}
S_CHAR_SEQ            {S_CHAR}+
EXPONENTIAL            [eE]({SIGN}?){DIGIT_SEQ}
INT_CONST            "0"|({NON_ZERO_DIGIT}({DIGIT}*))
FRACTION_CONST        ({DIGIT_SEQ}?\.{DIGIT_SEQ})|({DIGIT_SEQ}\.)
FLOAT_CONST            ({FRACTION_CONST}({EXPONENTIAL}?))|({DIGIT_SEQ}{EXPONENTIAL})
ENUM_CONST            {ID}
CHAR_CONST            \'{C_CHAR_SEQ}\'

KEY                 "auto"|"break"|"case"|"char"|"const"|"continue"|"default"|"do"|"double"|"else"|"enum"|"ex

CONST               {INT_CONST}|{FRACTION_CONST}|{FLOAT_CONST}|{ENUM_CONST}|{CHAR_CONST}

PUNCT                "["|"]"|"++"|"?"|"="|","|"("|")"|"{"|"}"|"."|"->"|"*"|"+"|"-"|"~"|"!"|"%"|"<<"|">>"|"<"|

ID                  ({ALPHABET}+)({DIGIT}|{ALPHABET})*

STRL                 \"{S_CHAR_SEQ}?\"



%%

"//"[^\n]*            {;}
"/*"                {comment();}
```

```
{WS}                    {if(strcmp(yytext,"\n")==0)
                           fprintf(yyout,"\n");}

{KEY}                   {fprintf(yyout,"<KEYWORD,%s> ",yytext);}

{ID}                    {fprintf(yyout,"<IDENTIFIER,%s> ",yytext);}

{CONST}                 {fprintf(yyout,"<CONSTANT,%s> ",yytext);}

{PUNCT}                 {fprintf(yyout,"<PUNCTUATOR,%s> ",yytext);}

{STRL}                  {fprintf(yyout,"<STRING-LITERAL,%s> ",yytext);}

%%

void comment() {
    char c, prev = 0;
    while((c = input()) != 0){
        if(c == '/' && prev == '*') return;
        prev = c;
    }

    error("Unterminated Comment\n");
}
```

.y File

```
%{
#include <string.h>
#include <stdio.h>
extern int yylex();
void yyerror(char *s);
%}

%union {
  int integer;
  float real;
  char *string;
}

%token <integer> KEYWORD

%token <integer> IDENTIFIER

%token <integer> PUNCTUATOR

%token <integer> CONSTANT

%token <integer> STRING_LITERAL

%type <integer> exp
%%
  exp: {printf("NO GRAMMAR\n");}
%%

void yyerror(char *s) {
printf("%s\n",s);
}
```

4 Sample 4

- All Keywords clubbed in a single regular definition
- All Punctuators clubbed in a single regular definition
- Specific tokens returned as lexeme through yytext
- Same token returned for all Constants – value not computed – lexeme returned in yytext
- Specific token IDs not defined and used
- Comment definitions complicated

.l File

```
%{
    #include <math.h>
    #include "y.tab.h"
%}

multi_comment ("/*"([^*]|\*+[^*/])*\*+"/")

single_comment("//"[^\n]*)

keyword    (auto|enum|restrict|unsigned|break|extern|return|void|case|float|short|volatile|char|for|signed|wh

digit [0-9]

identifier-nondigit  [_a-zA-Z]

identifier [_a-zA-Z]([_a-zA-Z]|{digit})*

constant ({integer-constant}|{floating-constant}|{enumeration-constant}|{character-constant})

integer-constant {nonzero-digit}({digit})*

nonzero-digit  [1-9]

floating-constant  ({fractional-constant}|{fractional-constant}{exponent-part}|{digit-sequence}{exponent-part

fractional-constant ("."{digit-sequence}|{digit-sequence}"."{digit-sequence}|{digit-sequence}".")

exponent-part  ("e"{sign}{digit-sequence}|"e"{digit-sequence}|"E"{sign}{digit-sequence}|"E"{digit-sequence})

sign  ("+"|"-")

digit-sequence ({digit})+

enumeration-constant {identifier}

character-constant "'"{c-char-sequence}"'"

c-char-sequence ({c-char})+

c-char [^("\\"|"\'"|"\n")]
```

```
escape-sequence " "|"\\\'"|"\\\""|"\\\?"|"\\\\"|" \\a"|"\\b"|"\\f"|"\\n"|"\\r"|"\\t"|"\\v"

string-literal "\""{s-char-sequence}"\""|"\"\""

s-char-sequence ({s-char})+

s-char [^("\n"|"\\"|"\"")]

punctuator  "["|"]"|"("|")"|"{"|"}"|"."|"->"|"++"|"--"|"&"|"*"|"+"|"-"|"~"|"!"|"/"|"%"|"<<"|">>"|"<"|">"|"<="


%%
{multi_comment}    {
    ;
}
{single_comment}    {
    ;
}
{keyword}  {
        yylval.strval=strdup(yytext);
        return KEYWORD;
    }
{identifier}  {
        yylval.strval=strdup(yytext);
        return IDENTIFIER;
    }
{constant}  {
        yylval.strval=strdup(yytext);
        return CONSTANT;
    }
{string-literal}  {
        yylval.strval=strdup(yytext);
        return STRING_LITERAL;
    }
{punctuator}  {
        yylval.strval=strdup(yytext);
        return PUNCTUATOR;
    }
{escape-sequence}    ;
. ;
%%
```

.y File

```
%{
  #include <string.h>
  #include <iostream>
  extern int yylex();
 void yyerror(char *s);
%}

%union{
char *strval;
}

%token <strval> KEYWORD
%token <strval> IDENTIFIER
%token <strval> CONSTANT
%token <strval> STRING_LITERAL
%token <strval> PUNCTUATOR

%%
   statement:  KEYWORD
        {printf("%s keyword\n",$1);}
        |statement KEYWORD
        {printf("%s keyword\n",$2);}
        |IDENTIFIER
        {printf("%s identifier\n",$1);}
        |statement IDENTIFIER
        {printf("%s identifier\n",$2);}
        |CONSTANT
        {printf("%s constant",$1);}
        |statement CONSTANT
        {printf("%s constant\n",$2);}
         |STRING_LITERAL
        {printf("%s string_literal\n",$1);}
        |statement STRING_LITERAL
        {printf("%s string_literal\n",$2);}
        |PUNCTUATOR
        {printf("%s punctuator\n",$1);}
        |statement PUNCTUATOR
        {printf("%s punctuator\n",$2);}
%%

void yyerror(char *s) {
std::cout << s << std::endl;
}

int main(){
    yyparse();
}
```

5 Sample 5

- Lexems of Keywords printed through yytext - unnecessary
- Lexems of Punctuators printed through yytext - unnecessary
- Same token returned for all Constants – value not computed – lexeme printed from yytext
- Comment handling is wrong - only start and end of comments detected (possibly erroneously)

.l File

```
%{
#include "y.tab.h"
#include <stdio.h>
#include <math.h>
%}

nondigit     [a-zA-Z_]
digit         [0-9]
exp          [eE][+-]?[0-9]+

%%

"auto"        {printf("<Keyword,%s>",yytext);return AUTO;}
"break"        {printf("<Keyword,%s>",yytext);return(BREAK);}
"case"        {printf("<Keyword,%s>",yytext);return(CASE);}
"char"        {printf("<Keyword,%s>",yytext);return(CHAR);}
"const"        {printf("<Keyword,%s>",yytext);return(CONST);}
"continue"    {printf("<Keyword,%s>",yytext);return(CONTINUE);}
"default"    {printf("<Keyword,%s>",yytext);return(DEFAULT);}
"do"        {printf("<Keyword,%s>",yytext);return(DO);}
"double"    {printf("<Keyword,%s>",yytext);return(DOUBLE);}
"else"        {printf("<Keyword,%s>",yytext);return(ELSE);}
"enum"        {printf("<Keyword,%s>",yytext);return(ENUM);}
"extern"    {printf("<Keyword,%s>",yytext);return(EXTERN);}
"float"        {printf("<Keyword,%s>",yytext);return(FLOAT);}
"for"        {printf("<Keyword,%s>",yytext);return(FOR);}
"goto"        {printf("<Keyword,%s>",yytext);return(GOTO);}
"if"        {printf("<Keyword,%s>",yytext);return(IF);}
"inline"    {printf("<Keyword,%s>",yytext);return(INLINE);}
"int"        {printf("<Keyword,%s>",yytext);return(INT);}
"long"        {printf("<Keyword,%s>",yytext);return(LONG);}
"register"    {printf("<Keyword,%s>",yytext);return(REGISTER);}
"restrict"    {printf("<Keyword,%s>",yytext);return(RESTRICT);}
"return"    {printf("<Keyword,%s>",yytext);return(RETURN);}
"short"        {printf("<Keyword,%s>",yytext);return(SHORT);}
"size"        {printf("<Keyword,%s>",yytext);return(SIZE);}
"signed"    {printf("<Keyword,%s>",yytext);return(SIGNED);}
"sizeof"    {printf("<Keyword,%s>",yytext);return(SIZEOF);}
"static"    {printf("<Keyword,%s>",yytext);return(STATIC);}
"struct"    {printf("<Keyword,%s>",yytext);return(STRUCT);}
"switch"    {printf("<Keyword,%s>",yytext);return(SWITCH);}
"typedef"    {printf("<Keyword,%s>",yytext);return(TYPEDEF);}
"union"        {printf("<Keyword,%s>",yytext);return(UNION);}
"unsigned"    {printf("<Keyword,%s>",yytext);return(UNSIGNED);}
```

```
"volatile"    {printf("<Keyword,%s>",yytext);return(VOLATILE);}
"while"        {printf("<Keyword,%s>",yytext);return(WHILE);}
"_Bool"        {printf("<Keyword,%s>",yytext);return(_BOOL);}
"_Complex"    {printf("<Keyword,%s>",yytext);return(_COMPLEX);}
"_Imaginary"    {printf("<Keyword,%s>",yytext);return(_IMAGINARY);}

"\*"         {printf("<Comment_start,%s>",yytext);return(CMMNT_STRT);}
"*/"         {printf("<Comment_End>,%s",yytext);return(CMMNT_END);}
"//"          {printf("<Single_Comment>,%s",yytext);return(CMMNT_SNGLE);}

"*="         {printf("<punctuator,%s>",yytext);return(MUL_ASSIGN);}
"/="         {printf("<punctuator,%s>",yytext);return(DIV_ASSIGN);}
"%="         {printf("<punctuator,%s>",yytext);return(MOD_ASSIGN);}
"+="         {printf("<punctuator,%s>",yytext);return(ADD_ASSIGN);}
"-="         {printf("<punctuator,%s>",yytext);return(SUB_ASSIGN);}
"<<="         {printf("<punctuator,%s>",yytext);return(LEFT_ASSIGN);}
">>="         {printf("<punctuator,%s>",yytext);return(RIGHT_ASSIGN);}
"&="         {printf("<punctuator,%s>",yytext);return(AND_ASSIGN);}
"^="         {printf("<punctuator,%s>",yytext);return (XOR_ASSIGN);}
"|="         {printf("<punctuator,%s>",yytext);return(OR_ASSIGN);}
"->"         {printf("<punctuator,%s>",yytext);return(PTR_OP);}
"++"         {printf("<punctuator,%s>",yytext);return(INCREMENT);}
"--"         {printf("<punctuator,%s>",yytext);return(DECREMENT);}
"<<"         {printf("<punctuator,%s>",yytext);return(RSHIFT_OP);}
">>"         {printf("<punctuator,%s>",yytext);return(LSHIFT_OP);}
"<="         {printf("<punctuator,%s>",yytext);return(LTE_OP);}
">="         {printf("<punctuator,%s>",yytext);return(GTE_OP);}
"=="         {printf("<punctuator,%s>",yytext);return(EQ_OP);}
"!="         {printf("<punctuator,%s>",yytext);return(NEQ_OP);}
"&&"         {printf("<punctuator,%s>",yytext);return(AND);}
"||"         {printf("<punctuator,%s>",yytext);return(OR);}
"..."         {printf("<punctuator,%s>",yytext);return(ELLIPSIS);}
"["        {printf("<punctuator,%s>",yytext);return('[');}
"]"        {printf("<punctuator,%s>",yytext);return(']');}
"("        {printf("<punctuator,%s>",yytext);return('(');}
")"        {printf("<punctuator,%s>",yytext);return(')');}
"{"        {printf("<punctuator,%s>",yytext);return('{');}
"}"        {printf("<punctuator,%s>",yytext);return('}');}
"."        {printf("<punctuator,%s>",yytext);return('.');}
"&"        {printf("<punctuator,%s>",yytext);return('&');}
"*"        {printf("<punctuator,%s>",yytext);return('*');}
"+"        {printf("<punctuator,%s>",yytext);return('+');}
"-"        {printf("<punctuator,%s>",yytext);return('-');}
"~"        {printf("<punctuator,%s>",yytext);return('~');}
"!"        {printf("<punctuator,%s>",yytext);return('!');}
"/"        {printf("<punctuator,%s>",yytext);return('/');}
"%"        {printf("<punctuator,%s>",yytext);return('%');}
"<"        {printf("<punctuator,%s>",yytext);return('<');}
">"        {printf("<punctuator,%s>",yytext);return('>');}
"^"        {printf("<punctuator,%s>",yytext);return('^');}
"|"        {printf("<punctuator,%s>",yytext);return('|');}
"?"        {printf("<punctuator,%s>",yytext);return('?');}
":"        {printf("<punctuator,%s>",yytext);return(':');}
";"        {printf("<punctuator,%s>",yytext);return(';');}
```

```
"="         {printf("<punctuator,%s>",yytext);return('=');}

","         {printf("<punctuator,%s>",yytext);return(',');}
"#"         {printf("<punctuator,%s>",yytext);return('#');}

[_a-zA-Z]+[a-zA-Z_0-9]*          {printf("<id,%s>",yytext);return IDENTIFIER;}

[0-9]*'.'[0-9]+({exp})?             {printf("<floating_const,%s>",yytext);return(CONSTANT);}
[0-9]+'.'({exp})?             {printf("<floating_const,%s>",yytext);return(CONSTANT);}
[0-9]+({exp})                 {printf("<floating_const,%s>",yytext);return(CONSTANT);}

[1-9]+[0-9]*                  {printf("<int_const,%s>",yytext);return CONSTANT;}

\'(\\[abfnrtv\"\'?\\]|[^\\'\n])*\'    {printf("<char_const,");printf(yytext);printf(">");return(CONSTANT);};
\"(\\[abfnrtv\"\'?\\]|[^\\'\n])*\"    {printf("<strng_literal,");printf(yytext);printf(">");return(CONSTANT);

.                     {printf("%s",yytext);};
```

.y File

```
%{
#include <stdio.h>

extern int yylex();
void yyerror(char *s);
%}

%union{
    int intval;
}

%token KEYWORD
%token IDENTIFIER
%token CONSTANT
%token STRING_LITERAL
%token PUNCTUATOR


%token AUTO
%token BREAK
%token CASE
%token CHAR
%token CONST
%token CONTINUE
%token DEFAULT
%token DO
%token DOUBLE
%token ELSE
%token ENUM
%token EXTERN
%token FLOAT
%token FOR
%token GOTO
%token IF
%token INLINE
%token INT
%token LONG
%token REGISTER
%token RESTRICT
%token RETURN
%token SHORT
%token SIZE
%token SIGNED
%token SIZEOF
%token STATIC
%token STRUCT
%token SWITCH
%token TYPEDEF
%token UNION
%token UNSIGNED
%token VOLATILE
%token WHILE
%token _BOOL
```

```
%token _COMPLEX
%token _IMAGINARY

%token PTR_OP
%token INCREMENT
%token DECREMENT
%token ADD_ASSIGN
%token MUL_ASSIGN
%token DIV_ASSIGN
%token MOD_ASSIGN
%token SUB_ASSIGN
%token RSHIFT_OP
%token LSHIFT_OP
%token LTE_OP
%token GTE_OP
%token EQ_OP
%token NEQ_OP
%token AND
%token OR
%token ELLIPSIS
%token LEFT_ASSIGN
%token RIGHT_ASSIGN
%token AND_ASSIGN
%token OR_ASSIGN
%token XOR_ASSIGN

%token CMMNT_STRT
%token CMMNT_END
%token CMMNT_SNGLE

%token SNGLE_QUOTE
%token DBLE_QUOTE
%token QUE_MARK
%token BACK_SLSH
%token BEEP
%token CARRIAGE_RTRN
%token  FEED_FORM
%token  VRTCL_TAB
%token  HZNTL_TAB
%token  BACK_SPACE
%token  NEW_LINE
%token DUMMY
%%
s:DUMMY;
%%

void yyerror(char *s){
    printf("%s",s);
}
```

6 Sample 6

- For Constants – value not computed
- For Identifiers – lexem is not set
- Comment handle by simple yet error-prone C function

.l File

```
%{
#include "y.tab.h"
#include <math.h>
extern void yyerror(const char *);  /* prints grammar violation message */
void count(void);
void comment (void);
%}

/* Regular Expression Definitions */

D                    [0-9]
L                      [a-zA-Z_]
ID                     {L}({L}|{D})*
nonzero-digit        [1-9]
E                    ([Ee][+-]?{D}+)
IC                     {nonzero-digit}{D}*
FC                     ({FRC}{E}?)|({D}+{E})
FRC                 (({D}+)?\.{D}+)|(({D}+)\.)

EC                     {ID}
ESC_SEQ                \\['"?\\abfnrtv]
cchar                 [^'\\\n]|{ESC_SEQ}
CC                     '({cchar})+'

schar                  [^"\\\n]|{ESC_SEQ}
SC                     \"({schar}+)\"

WS                   [ \t\v\n\f]
%%

"/*"            { comment(); return COMMENT;}
"//".*             { return COMMENT; /* consume //-comment */ }

"auto"           { return AUTO; }
"break"         { return BREAK; }
"case"          { return CASE; }
"char"          { return CHAR; }
"const"         { return CONST; }
"continue"      { return CONTINUE; }
"default"       { return DEFAULT; }
"do"            { return DO; }
"double"        { return DOUBLE; }
"else"          { return ELSE; }
"enum"          { return ENUM; }
"extern"        { return EXTERN; }
"float"         { return FLOAT; }
```

```
"for"           { return FOR; }
"goto"          { return GOTO; }
"if"            { return IF; }
"inline"        { return INLINE; }
"int"           { return INT; }
"long"          { return LONG; }
"register"      { return REGISTER; }
"restrict"      { return RESTRICT; }
"return"        { return RETURN; }
"short"         { return SHORT; }
"signed"        { return SIGNED; }
"sizeof"        { return SIZEOF; }
"static"        { return STATIC; }
"struct"        { return STRUCT; }
"switch"        { return SWITCH; }
"typedef"       { return TYPEDEF; }
"union"         { return UNION; }
"unsigned"      { return UNSIGNED; }
"void"          { return VOID; }
"volatile"      { return VOLATILE; }
"while"         { return WHILE; }
"_Bool"         { return BOOL; }
"_Complex"      { return COMPLEX; }
"_Imaginary"    { return IMAGINARY; }

{SC}             { return STRING; }
{ID}            { return IDENTIFIER; }
{IC}             { return INT_CONSTANT; }
{FC}             { return FLOAT_CONSTANT; }
{EC}             { return ENU_CONSTANT; }
{CC}             { return CHAR_CONSTANT; }

"["               { return '['; }
"]"               { return ']'; }
"("               { return '('; }
")"               { return ')'; }
"{"               { return '{'; }
"}"               { return '}'; }
"."               { return '.'; }
"->"            { return PTR_OP; }

"++"            { return INC_OP; }
"--"            { return DEC_OP; }
"&"               { return '&'; }
"*"               { return '*'; }
"+"               { return '+'; }
"-"               { return '-'; }
"~"               { return '~'; }
"!"               { return '!'; }

"/"               { return '/'; }
"%"               { return '%'; }
"<<"            { return LEFT_OP; }
">>"            { return RIGHT_OP; }
```

```
"<"                { return '<'; }
">"                { return '>'; }
"<="            { return LE_OP; }
">="            { return GE_OP; }
"=="            { return EQ_OP; }
"!="            { return NE_OP; }
"^"                { return '^'; }
"|"                { return '|'; }
"&&"            { return AND_OP; }
"||"            { return OR_OP; }

"?"                { return '?'; }
":"                { return ':'; }
";"                { return ';'; }
"..."            { return ELLIPSIS; }

"="                { return '='; }
"*="            { return MUL_ASSIGN; }
"/="            { return DIV_ASSIGN; }
"%="            { return MOD_ASSIGN; }
"+="            { return ADD_ASSIGN; }
"-="            { return SUB_ASSIGN; }
"<<="            { return LEFT_ASSIGN; }
">>="            { return RIGHT_ASSIGN; }
"&="            { return AND_ASSIGN; }
"^="            { return XOR_ASSIGN; }
"|="            { return OR_ASSIGN; }

","                { return ','; }
"#"                { return '#'; }

{WS}            { /* whitespace separates tokens */ }
%%

int column = 0;
void count(void) {
    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;
}
void comment(void) {
    char c, prev = 0;
    while ((c = input()) != 0) {     /* (EOF maps to 0) */
        if (c == '/' && prev == '*')
            return;
        prev = c;
    }
    error("unterminated comment");
}
```

.y File

```
%{ /* C Declarations and Definitions */
#include <string.h>
#include <stdio.h>
extern int yylex();
void yyerror(char *s);
%}

%union {
int intval;
}

%token TYPEDEF EXTERN STATIC AUTO REGISTER INLINE RESTRICT
%token CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE CONST VOLATILE VOID
%token BOOL COMPLEX IMAGINARY
%token STRUCT UNION ENUM
%token BREAK CASE CONTINUE DEFAULT DO IF ELSE FOR GOTO WHILE SWITCH SIZEOF
%token RETURN

%token ELLIPSIS RIGHT_ASSIGN LEFT_ASSIGN ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN
%token DIV_ASSIGN MOD_ASSIGN AND_ASSIGN XOR_ASSIGN OR_ASSIGN RIGHT_OP LEFT_OP
%token INC_OP DEC_OP PTR_OP AND_OP OR_OP LE_OP GE_OP EQ_OP NE_OP

%token IDENTIFIER STRING PUNCTUATORS COMMENT
%token INT_CONSTANT FLOAT_CONSTANT ENU_CONSTANT CHAR_CONSTANT

%%
dummy:AUTO
%%
void yyerror(char *s) {
    printf ("ERROR: %s",s);
}
```

7 Sample 7

- Comment handled erroneously
- Newline (\n) treated as white space – line cannot be counted

.l File

```
%{
#include <iostream>
#include "y.tab.h"
#include <math.h>
#include "parser.h"
using namespace std;
%}

DIGIT          [0-9]
NON_ZERO_DIGIT [1-9]
ID_NON_DIGIT   [_a-zA-Z]
ID             {ID_NON_DIGIT}({ID_NON_DIGIT}|{DIGIT})*
INT_CONST      0|{NON_ZERO_DIGIT}({DIGIT})*
DIGIT_SEQ      {DIGIT}+
FRAC_CONST     ({DIGIT_SEQ}?\.{DIGIT_SEQ})|({DIGIT_SEQ}\.)
EXP            (e|E)[+-]?{DIGIT}+
FLOAT_CONST    ({FRAC_CONST}{EXP}?)|({DIGIT}+{EXP})
STRING_CONST   (\".*\")
COMMENTS       "//".*\n


%%
"auto"        { return AUTO; }
"break"       { return BREAK; }
"case"        { return CASE; }
"char"        { return CHAR; }
"const"       { return CONST; }
"continue"    { return CONTINUE; }
"default"     { return DEFAULT; }
"do"          { return DO; }
"double"      { return DOUBLE; }
"else"        { return ELSE; }
"enum"        { return ENUM; }
"extern"      { return EXTERN; }
"float"       { return FLOAT; }
"for"         { return FOR; }
"goto"        { return GOTO; }
"if"          { return IF; }
"inline"      { return INLINE; }
"int"         { return INT; }
"long"        { return LONG; }
"register"    { return REGISTER; }
"restrict"    { return RESTRICT; }
"return"      { return RETURN; }
"short"       { return SHORT; }
"signed"      { return SIGNED; }
"sizeof"      { return SIZEOF; }
```

```
"static"      { return STATIC; }
"struct"      { return STRUCT; }
"switch"      { return SWITCH; }
"typedef"     { return TYPEDEF; }
"union"       { return UNION; }
"unsigned"    { return UNSIGNED; }
"void"        { return VOID; }
"volatile"    { return VOLATILE; }
"while"       { return WHILE; }
"_Bool"       { return _BOOL; }
"_Complex"    { return _COMPLEX; }
"_Imaginary"  { return _IMAGINARY; }

"("           { return LPARAN; }
")"           { return RPARAN; }
"{"           { return LBRACE; }
"}"           { return RBRACE; }
"["           { return LBRACKET; }
"]"           { return RBRACKET; }

"+"           { return ADD_OP; }
"-"           { return SUB_OP; }
"*"           { return MULT_OP; }
"/"           { return DIV_OP; }
"%"           { return MODULO_OP; }

"<<"          { return LSHIFT_OP; }
">>"          { return RSHIFT_OP; }

"<"           { return LESS_OP; }
">"           { return GREATER_OP; }
"<="          { return LEQ_OP; }
">="          { return GEQ_OP; }
"=="          { return EQ_OP; }
"!="          { return NEQ_OP; }

"++"          { return INC_OP; }
"--"          { return DEC_OP; }

"!"           { return LOGICAL_NEG_OP; }
"&&"          { return LOGICAL_AND_OP; }
"||"          { return LOGICAL_OR_OP; }

"~"           { return BIT_NOT_OP; }
"&"           { return BIT_AND_OP; }
"|"           { return BIT_OR_OP; }
"^"           { return BIT_XOR_OP; }

"="           { return ASSIGN_OP; }
"+="          { return ADD_ASSIGN_OP; }
"-="          { return SUB_ASSIGN_OP; }
"*="          { return MULT_ASSIGN_OP; }
"/="          { return DIV_ASSIGN_OP; }
"%="          { return MODULO_ASSIGN_OP; }
```

```
"&="            { return BIT_AND_ASSIGN_OP; }
"|="            { return BIT_OR_ASSIGN_OP; }
"^="            { return BIT_XOR_ASSIGN_OP; }
"<<="           { return BIT_LSHIFT_ASSIGN_OP; }
">>="           { return BIT_RSHIFT_ASSIGN_OP; }

"?"             { return QUESTIONMARK_OP; }
":"             { return COLON; }
","             { return COMMA; }

";"             { return SEMICOLON; }
"..."           { return ELLIPSES; }
"#"             { return HASH; }
"."             { return DOT_OP; }
"->"            { return STRUCT_REFERENCE; }

{INT_CONST}     { yylval.intval = atoi(yytext); return INT_CONST;}

{FLOAT_CONST}   { yylval.dval = atof(yytext); return FLOAT_CONST;}

{STRING_CONST} { yylval.sval = strdup(yytext); return STRING_CONST;}

{COMMENTS}      ;/* ignore comment */

[ \t\n]         ;          /* ignore white space */


{ID}            { /* return symbol pointer */
                  yylval.symp = symlook(yytext);
                  return ID;
                }
%%
/* int main() {
    int rVal = 0;
    while (rVal = yylex()) {
        printf("%d %s\n", rVal, yytext);
    }
    return 0;
} */
```

.y File

```
%{
#include <string.h>
#include <iostream>
using namespace std;
#include "parser.h"
extern int yylex();
void yyerror(const char *s);
#define NSYMS 20    /* maximum number of symbols */
symboltable symtab[NSYMS];
%}

%union {
    int intval;
    double dval;
    char *sval;
    struct symtab *symp;
}

%token <symp> ID
%token <intval> INT_CONST
%token <dval> FLOAT_CONST
%token <sval> STRING_CONST
%token ENUM_CONST


%token AUTO BREAK CASE CHAR CONST CONTINUE DEFAULT DO
%token DOUBLE ELSE ENUM EXTERN FLOAT FOR GOTO IF INLINE
%token INT LONG REGISTER RESTRICT RETURN SHORT SIGNED
%token SIZEOF STATIC STRUCT SWITCH TYPEDEF UNION UNSIGNED
%token VOID VOLATILE WHILE _BOOL _COMPLEX _IMAGINARY

%token LPARAN RPARAN LBRACKET RBRACKET LBRACE RBRACE
%token ADD_OP SUB_OP MULT_OP DIV_OP MODULO_OP
%token LSHIFT_OP RSHIFT_OP
%token LESS_OP GREATER_OP LEQ_OP GEQ_OP EQ_OP NEQ_OP
%token INC_OP DEC_OP
%token ASSIGN_OP ADD_ASSIGN_OP SUB_ASSIGN_OP MULT_ASSIGN_OP DIV_ASSIGN_OP MODULO_ASSIGN_OP
%token QUESTIONMARK_OP COLON COMMA
%token LOGICAL_NEG_OP LOGICAL_AND_OP LOGICAL_OR_OP
%token BIT_NOT_OP BIT_AND_OP BIT_OR_OP BIT_XOR_OP
%token BIT_AND_ASSIGN_OP BIT_OR_ASSIGN_OP BIT_XOR_ASSIGN_OP BIT_LSHIFT_ASSIGN_OP BIT_RSHIFT_ASSIGN_OP
%token SEMICOLON ELLIPSES HASH DOT_OP STRUCT_REFERENCE

%nonassoc UMINUS_OP UPLUS_OP ADDRESS_OF_OP CONTENT_OF_OP

%start start
%%
```