Sembol Tanıma

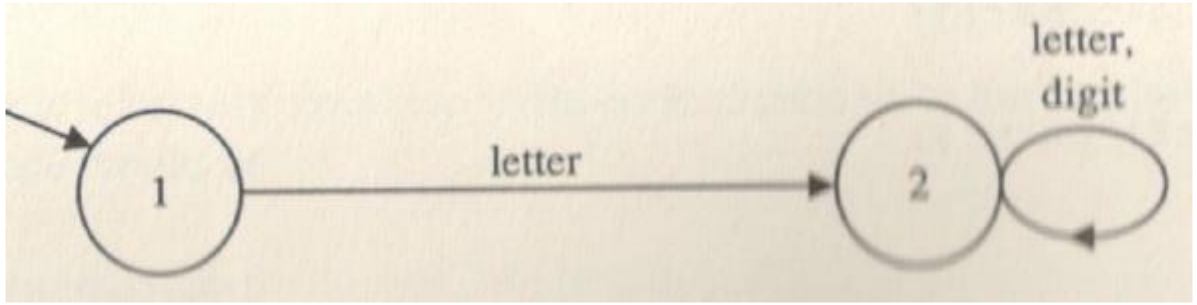
Örnekler

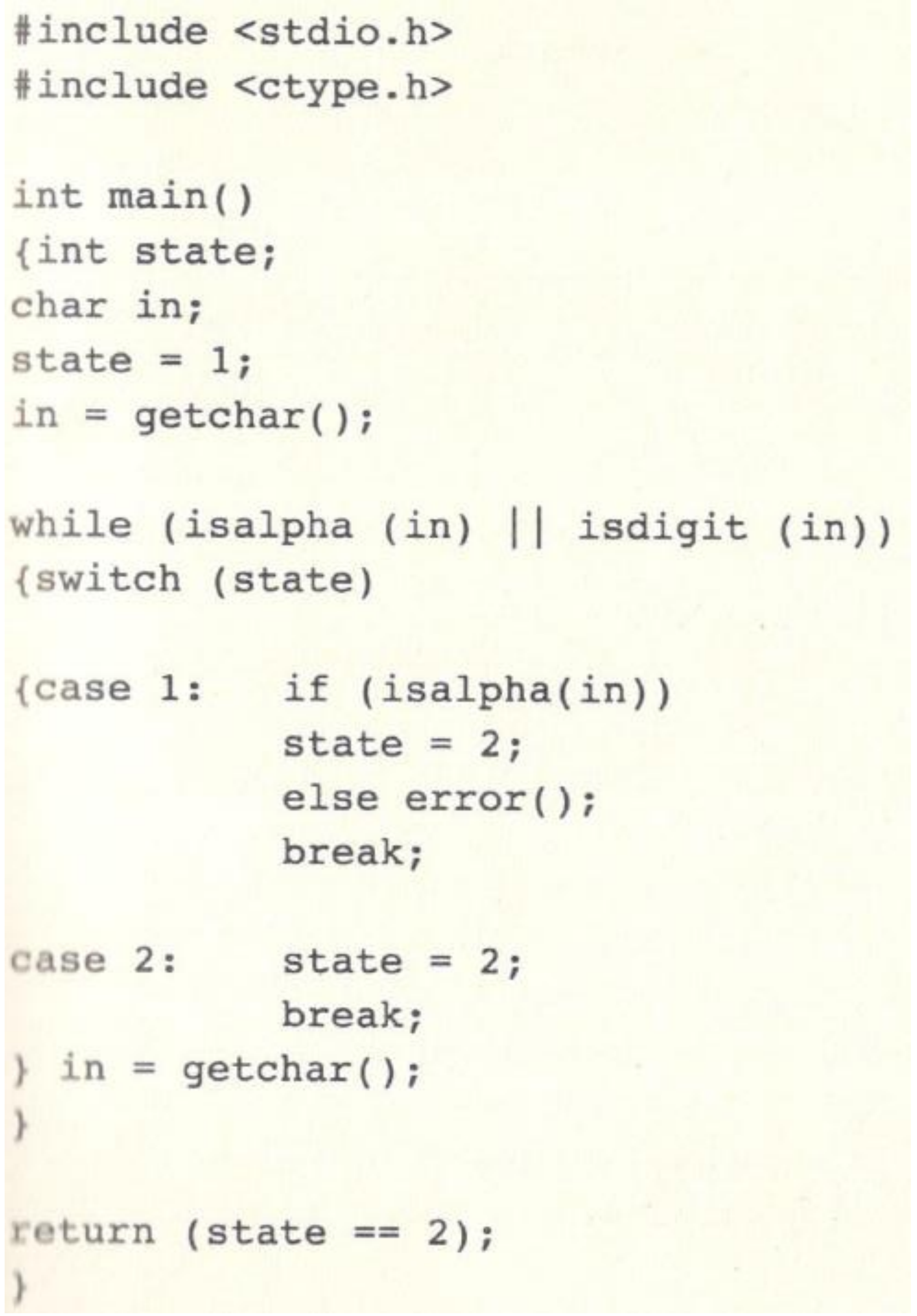
*letter(letter | digit)**

```
#include <stdio.h>
#include <ctype.h>

main()
{char in;
in = getchar();
if (isalpha(in))
in = getchar();
else error();
while (isalpha(in) || isdigit(in))
in = getchar();
}
```

(+ | – |)digit.digit digit**

```
#include <stdio.h>
#include <ctype.h>

main()
{char in;
in = getchar();

if (in=='+'||in=='-')
in = getchar();

while (isdigit(in))
in = getchar();
if (in=='.')
in = getchar();
else error();

if (isdigit(in))
in = getchar();
else error();

while (isdigit(in))
in = getchar();

printf("ok\n");
}
```

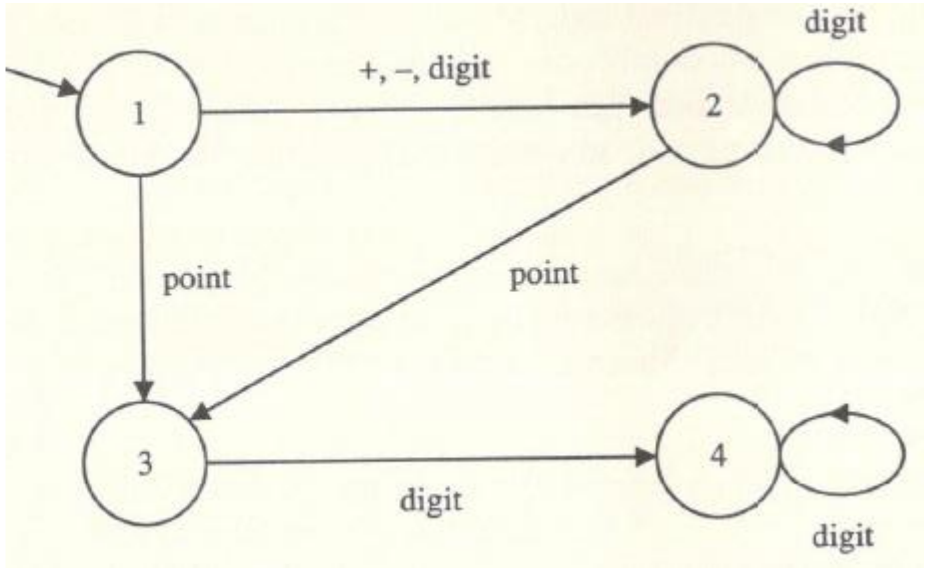




```
#include <stdio.h>
#include <ctype.h>

int main()
{int state;
char in;
state = 1;
in = getchar();

while (isalpha (in) || isdigit (in))
{switch (state)

{case 1:   if (isalpha(in))
           state = 2;
           else error();
           break;

case 2:    state = 2;
           break;
} in = getchar();
}

return (state == 2);
}
```

```
#include <stdio.h>
#include <ctype.h>
int issign (char sign)
{return (sign == '+' || sign == '-');
}

int main()

{int state;
char in;

state = 1;
in = getchar();

while (isdigit(in)||issign(in)|| in == '.')
{switch (state)
{
case 1:    if (isdigit (in)|| issign (in))
           state = 2;
           else if (in == '.')
           state = 3;
           break;
case 2:    if (isdigit(in))
           state = 2;
           else if (in == '.')
           state = 3;
           else error();
           break;
case 3:    if (isdigit(in))
           state = 4;
```

```
case 4:    if (isdigit(in))
           state = 4;
           else error();
           break;
}
  in = getchar();
}

return(state == 4);
}
```



LEX

```
letter      [a-z]
digit       [0-9]
identifier  {letter}({letter}|{digit})*
%%
{identifier}    {printf("identifier recognised\n");}
%%
```

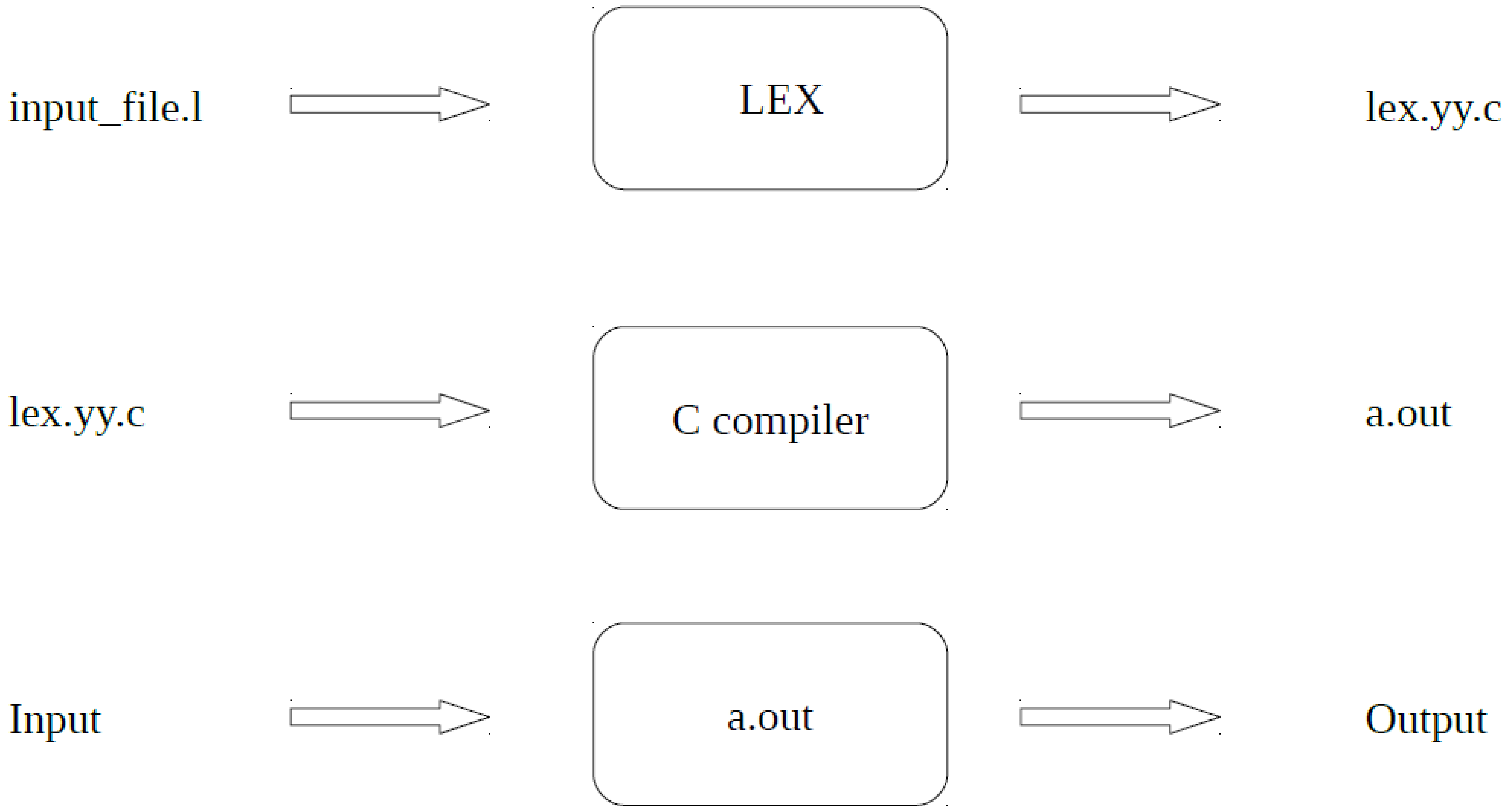
```
lex firstlex.l
```

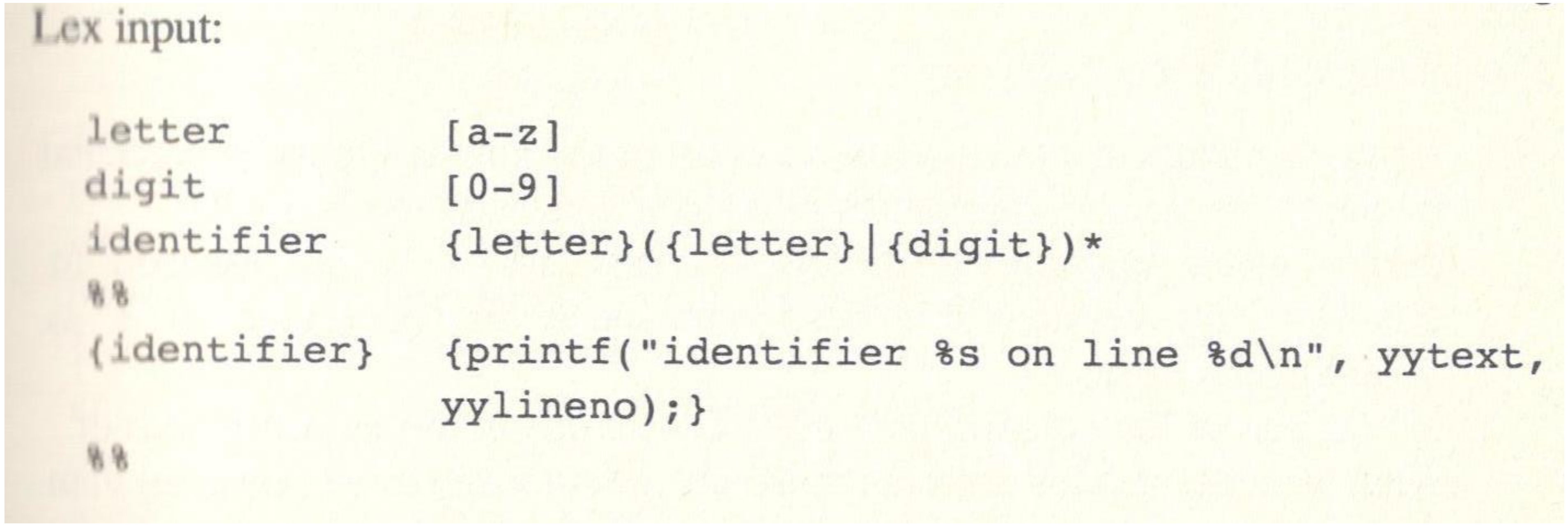
```
lex.yy.c
```

```
cc -o firstlex lex.yy.c -ll
```

```
firstlex <cprog
```

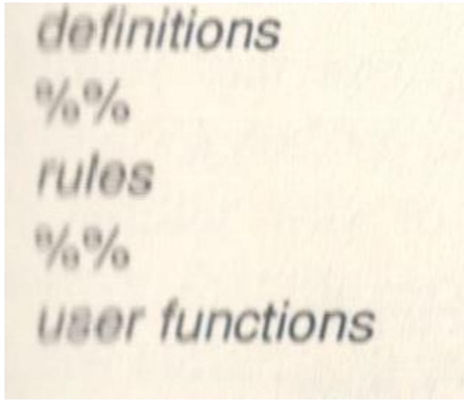
```
firstlex <cprog >idents
```


input_file.l
LEX
lex.yy.c
lex.yy.c
C compiler
a.out
Input
a.out
Output


Lex input:

```
letter          [a-z]
digit           [0-9]
identifier      {letter}({letter}|{digit})*
%%
{identifier}    {printf("identifier %s on line %d\n", yytext,
                yylineno);}
%%
```

yytext: Tanınan son sembolün metinsel gösterimi
yylineno: Karşılaşılan satırların sonunda bir sayı tutar ve değeri mevcut satır numarasını gösterir.

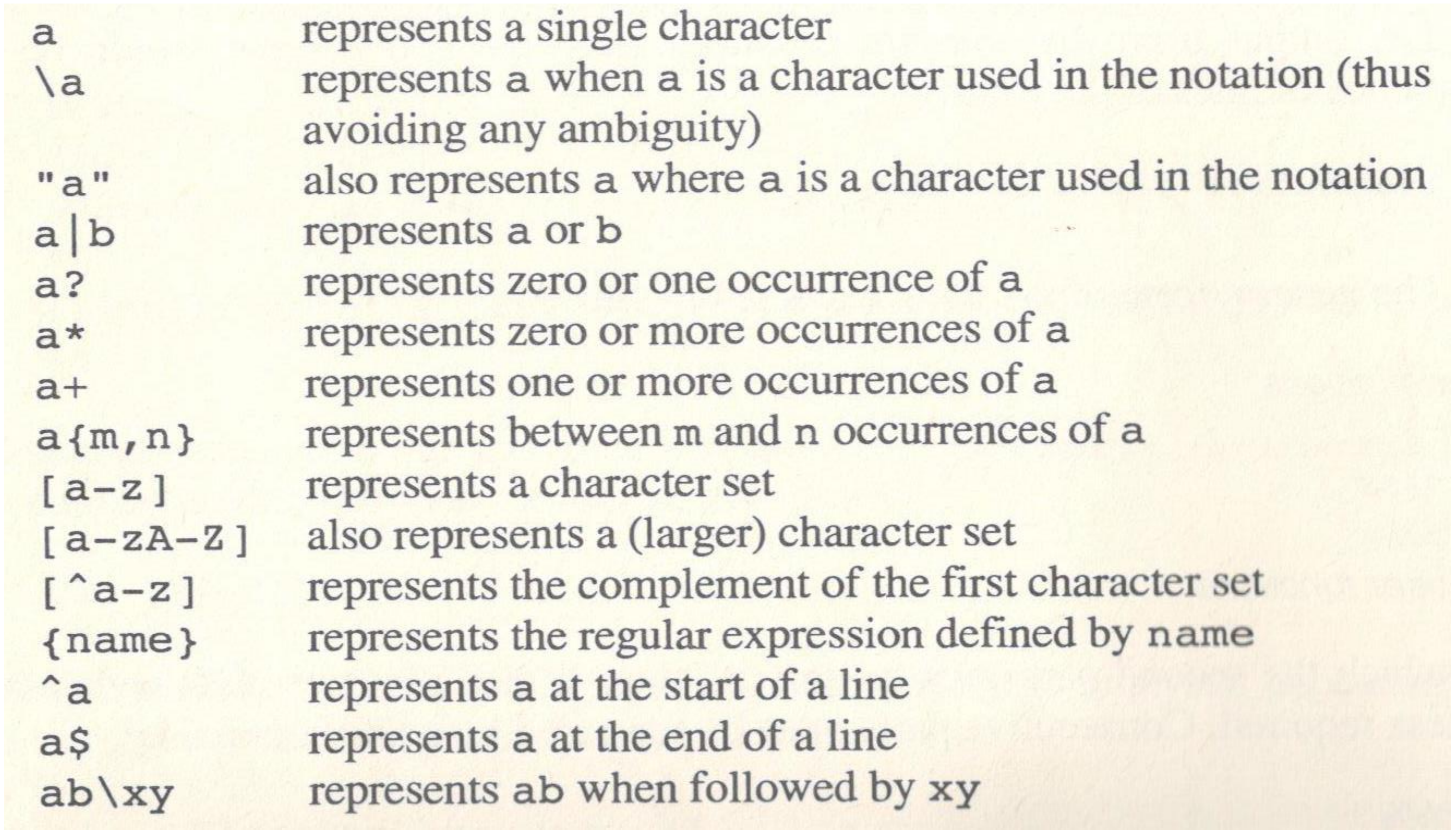


definitions
%%
rules
%%
user functions

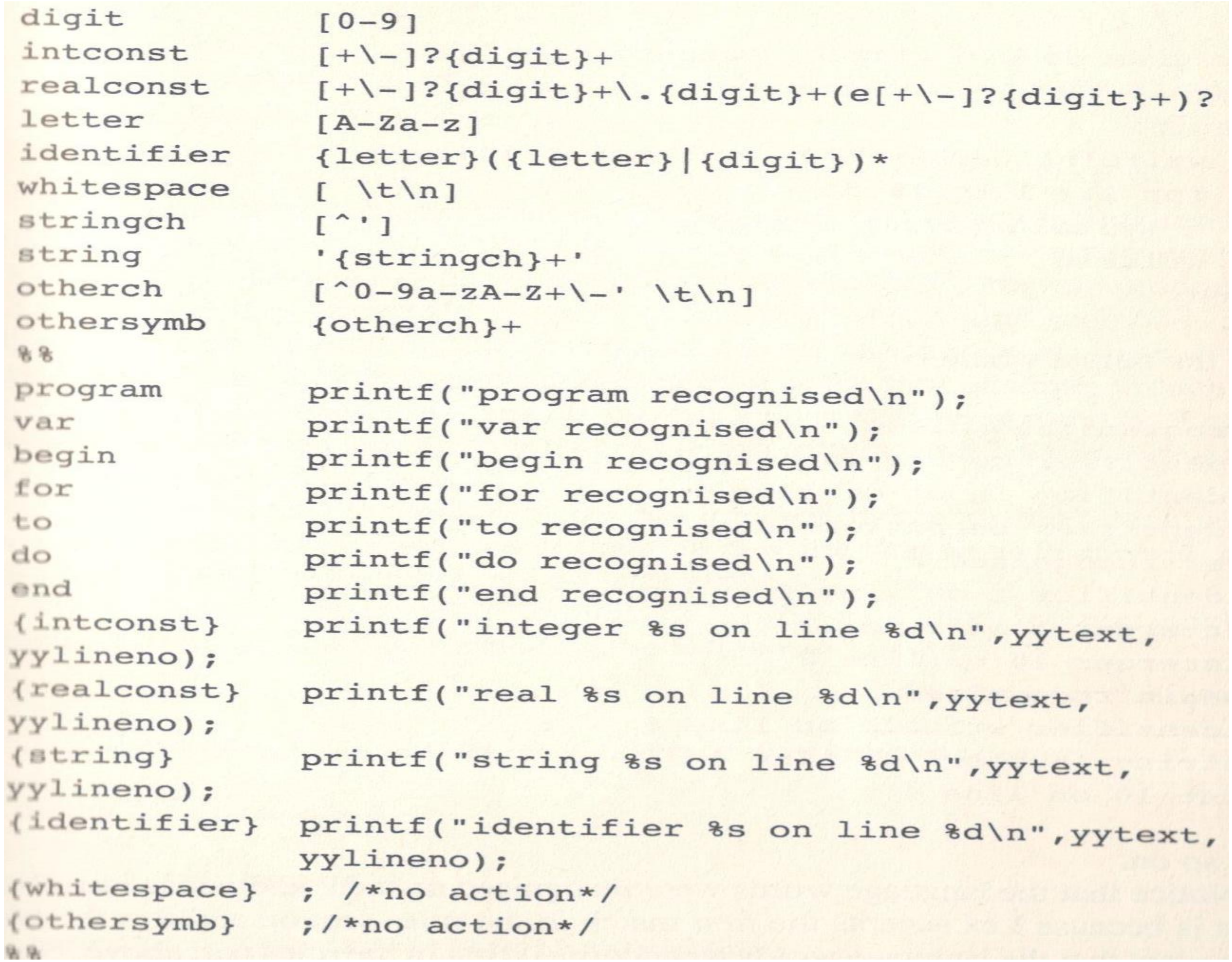
```
%{
    #include<stdio.h>
    int global_variable; //Auxiliary declarations
%}
number   [0-9]+           //Regular definitions
op       [-|+|*|/|^|=]

%%
    /* Rules */
%%
    /* Auxiliary functions */
```

(+ | − |)digit.digit digit**

```
digit       [0-9]
realno      [+\-]?{digit}*\.{digit}+
%%
{realno}    {printf("real%s on line %d\n",yytext,yylineno);}
```

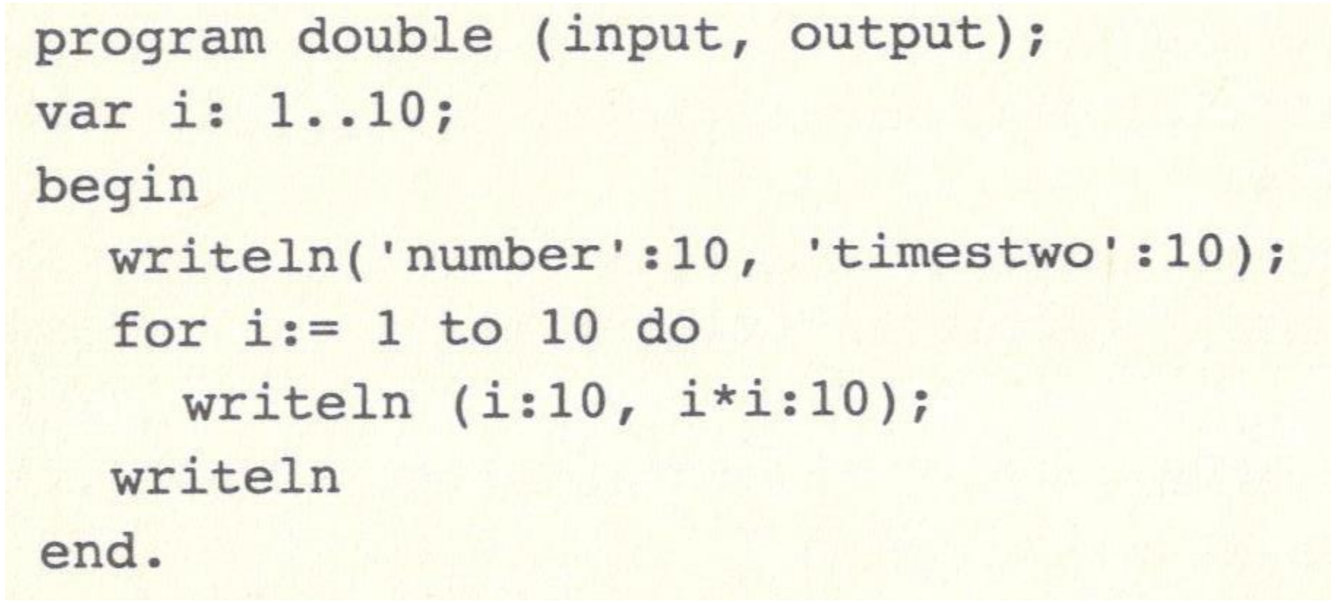


`a`	represents a single character
`\a`	represents `a` when `a` is a character used in the notation (thus avoiding any ambiguity)
`"a"`	also represents `a` where `a` is a character used in the notation
`a|b`	represents `a` or `b`
`a?`	represents zero or one occurrence of `a`
`a*`	represents zero or more occurrences of `a`
`a+`	represents one or more occurrences of `a`
`a{m,n}`	represents between `m` and `n` occurrences of `a`
`[a-z]`	represents a character set
`[a-zA-Z]`	also represents a (larger) character set
`[^a-z]`	represents the complement of the first character set
`{name}`	represents the regular expression defined by `name`
`^a`	represents `a` at the start of a line
`a$`	represents `a` at the end of a line
`ab\xy`	represents `ab` when followed by `xy`



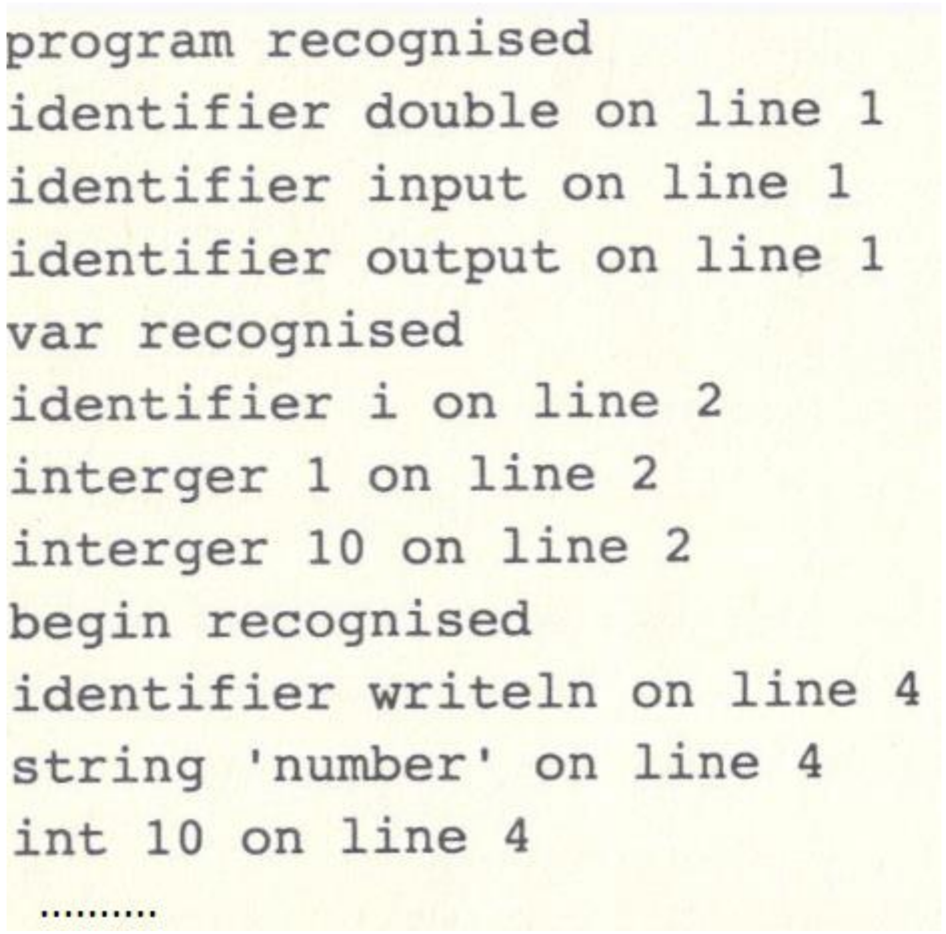
```
digit          [0-9]
intconst       [+\-]?{digit}+
realconst      [+\-]?{digit}+\.{digit}+(e[+\-]?{digit}+)?
letter         [A-Za-z]
identifier     {letter}({letter}|{digit})*
whitespace     [ \t\n]
stringch       [^']
string         '{stringch}+'
otherch        [^0-9a-zA-Z+\-' \t\n]
othersymb      {otherch}+
%%
program        printf("program recognised\n");
var            printf("var recognised\n");
begin          printf("begin recognised\n");
for            printf("for recognised\n");
to             printf("to recognised\n");
do             printf("do recognised\n");
end            printf("end recognised\n");
{intconst}     printf("integer %s on line %d\n",yytext,
yylineno);
{realconst}    printf("real %s on line %d\n",yytext,
yylineno);
{string}       printf("string %s on line %d\n",yytext,
yylineno);
{identifier}   printf("identifier %s on line %d\n",yytext,
               yylineno);
{whitespace}   ; /*no action*/
{othersymb}    ;/*no action*/
%%
```

LEX tarafından üretilen tarayıcıya aşağıdaki gibi bir PASCAL programı giriş olarak sunulsun.



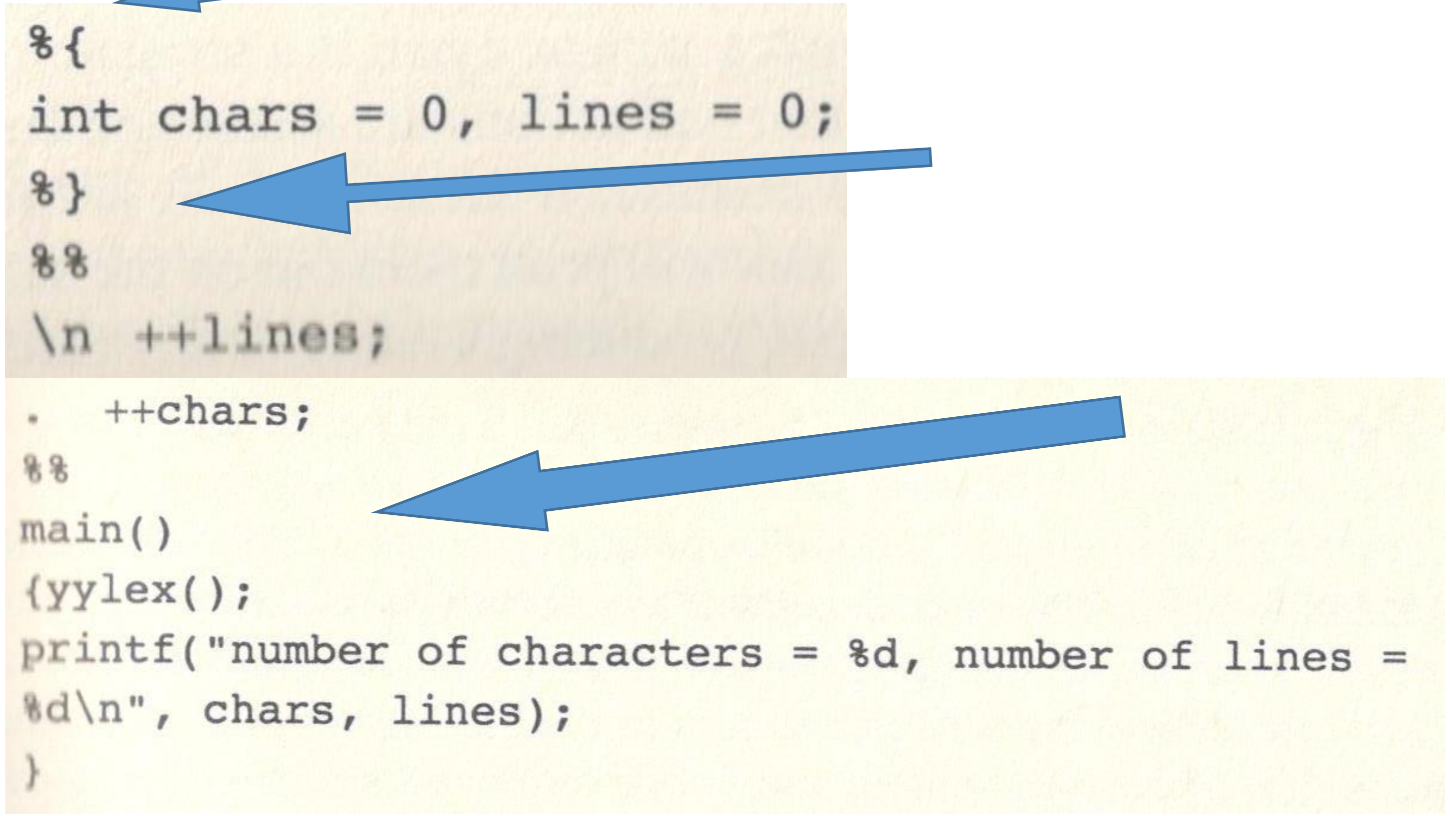
```
program double (input, output);
var i: 1..10;
begin
  writeln('number':10, 'timestwo':10);
  for i:= 1 to 10 do
    writeln (i:10, i*i:10);
  writeln
end.
```

Çıkış aşağıdaki gibi olacaktır.



```
program recognised
identifier double on line 1
identifier input on line 1
identifier output on line 1
var recognised
identifier i on line 2
interger 1 on line 2
interger 10 on line 2
begin recognised
identifier writeln on line 4
string 'number' on line 4
int 10 on line 4
 ..........
```

Aşağıdaki LEX girişi örneği, C kodunun LEX tarafından üretilen tarayıcının içerisien nasıl entegre edileceğini göstermektedir.



```
%{
int chars = 0, lines = 0;
%}
%%
\n ++lines;
.  ++chars;
%%
main()
{yylex();
printf("number of characters = %d, number of lines =
%d\n", chars, lines);
}
```

Örnek1

```
%{
/*
1.Request input of an even and an odd number
2.indicate input characteristic : Even/Odd [digit_length]
3.check for input's correctness and print result
*/

#include<stdlib.h>
#include<stdio.h>

int number_1;
int number_2;
%}

number_sequence [0-9]*
```

Örnek1 (devam)

```
%%

{number_sequence}[0|2|4|6|8]        {
                                    printf("Even number [%d]",yyleng);
                                    return atoi(yytext);
                               }

{number_sequence}[1|3|5|7|9]        {
                                    printf("Odd number [%d]",yyleng);
                                    return atoi(yytext);
                               }
%%
```

Örnek1 (devam)

```
int yywrap
{
     return 1;
}

int main()
{
     printf("\nInput an even number and an odd number\n");
     number_1 = yylex();
     number_2 = yylex();
     int diff = number_1 - number_2;
     if(diff%2!=0)
          printf("\nYour inputs were checked for correctness,
\nResult : Correct\n");
     else
          printf("\nYour inputs were checked for correctness,
\nResult : You do not know how to read\n");


     return 1;
}
```

Örnek 2

```
%{
/* Scan and return a token for identifiers of the format :
          (string)(number)
     Note : strings are not case sensitive
     examples : a0 , A1 , ab2 , AB4 , aBc5
*/
#include<stdio.h>

#define ID 1  //Identifier token
#define ER 2  //Error token

%}
```

```
low_case  [a-z]
upp_case  [A-Z]
number    [0-9]

%option noyywrap
```

```
I: Var9
O: Acceptable
```

```
%%
({low_case}|{upp_case})({low_case}|{upp_case})*({number})
                                          return ID;
(.)*                                      return ER;

%%
int main()
{
     int token = yylex();
     if(token==ID)
          printf("Acceptable\n");
     else if(token==ER)
          printf("Unacceptable\n");
     return 1;

}
```

```
%{
int letters = 0, words = 0, len = 0, length;
%}
word        [a-zA-Z]+
space       [ \n]
ws          {space}+
%%
{word}      {++words; length = yyleng;
            letters = letters+length;
            if (length > len) len = length;}
ws          ;/*do nothing*/
.           ;/*do nothing*/
%%
main()
{yylex();
printf("maximum word length = %d,average word length = %f\n",
  len, letters/words);
}
```

```
%{
int letters = 0, words = 0, len = 0, length;
%}
word        [a-zA-Z]+
space       [ \n]
ws          {space}+
eos         [!?.]
%%
{word}      {++words; length = yyleng;
            letters = letters+length;
            if (length > len) len = length;}
{eos}       yywrap();
ws          ;/*do nothing*/
.           ;/*do nothing*/
%%
main()
{yylex();
printf("maximum word length = %d,average word length = %f\n",
  len, letters/words);
}
```

```
%{
int lineno = 1;
%}

line [^\n]*\n
%%
{line}     {printf("%d %s", lineno++, yytext);}
%%
main()
{yylex();
}
```

```
%%
"/*" {char in;
  for (;;)
  {
  while ((in = getchar()) ! = '*');
  /* do nothing more */
  while ((in = getchar()) =='*');
  /* consume *'s */
  if (in == '/')
  break;
  /* end of commentary*/
  }
  }
```

```
COMMENT   "/*""/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"*/"
```

```
"/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"
```

```
"/*""/"*   ⇐   ⇒   "*"*"*/"
```

```
([^*/]|[^*]"/"|"*"[^/])*
```

```
COMMENT   "/*""/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"*/"
```

```
%{
int ncloc =0, count =0;
%}
comment         "/*""/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"*/"
space           [ \t]
newline         \n
%%
{comment}       ;\*do nothing*\
{space}         ;\*do nothing*\
{newline}       {if (count > 0) ncloc = ncloc+1; count = 0;}
.               count = count + 1;
%%
main()
{yylex();
printf("number of non-comment lines of code (NCLOC)= %d", ncloc);
}
```

```
%{
int nochars = 0, ncloc = 0, count = 0;
%}
comment       "/*""/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"*/"
space         [ \t]
newline       \n
%%
{comment}     ;\*do nothing*\
{space}       ;\*do nothing*\
{newline}     {if (count > 0) ncloc = ncloc+1; count = 0;}
.             {count = count+1; nochars = nochars+1;}
main()
{yylex();
printf("number of characters per non-comment lines of code
                                  (NCLOC) = %f", ncloc/nochars);
}
```