Session 2: Lexical Analysis

Assignment:

- Suppose, we have a C source program scanned and filtered as it was done in Session 1. We now take that modified file as input, and separate the lexemes first. We further recognize and mark the lexemes as different types of tokens like keywords, identifiers, operators, separators, parenthesis, numbers, etc.

Sample input:

char c; int x1, x_2; float y1, y2; x1=5; x_2= 10; y1=2.5+x1*45; y2=100.o5-x_2/3; if(y1<=y2) c='y'; else c='n';

Step 1: Lexemes are separated. Mark that two-character relational operators are also distinguished beside separators, one-character operators, parenthesis, number constants and alphanumeric strings with or without underscore.

char c ; int x1 , x_2 ; float y1 , y2 ; x1 = 5 ; x_2 = 10 ; y1 = 2.5 + x1 * 45 ; y2 = 100.o5 - x_2 / 3 ; if (y1 <= y2) c = ' y ' ; else c = ' n ' ;

Step 2. Lexemes are categorized under the categories kw for keyword, id for identifier, etc. Some may be labeled unkn (unknown).

[kw char] [id c] [sep ;] [kw int] [id x1] [sep ,] [id x_2] [sep ;] [kw float] [id y1] [sep ,] [id y2] [sep ;] [id x1] [op =] [num 5] [sep ;] [id x_2] [op =] [num 10] [sep ;] [id y1] [op =] [num 2.5] [op +] [id x1] [op *] [num 45] [sep ;] [id y2] [op =] [unkn 100.o5] [op -] [id x_2] [op /] [num 3] [sep ;] [kw if] [par (] [id y1] [op <=] [id y2] [par)] [id c] [op =] [sep '] [id y] [sep '] [sep ;] [kw else] [id c] [op =] [sep '] [id n] [sep '] [sep ;]

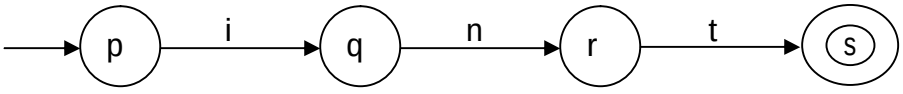
- Note that we need to request for generation of an error message for [unkn 100.o5]. And we will further modify some of the tokens (categorized/ recognized lexemes) for symbol table entry. For example, we may replace all [id c] in the same scope with the token, say, [id 1], all [id x1] with [id 2], and so on. Here, 1 and 2 refer to entry pointer (record number) in the Symbol Table.

Sample tools

DFAs for recognition of tokens

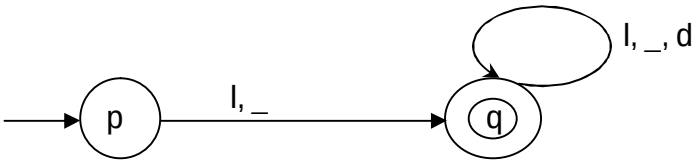
Recognition of keywords using DFA is easier than recognition of identifiers and numbers.

1. To recognize a keyword, int we can just use the following DFA:



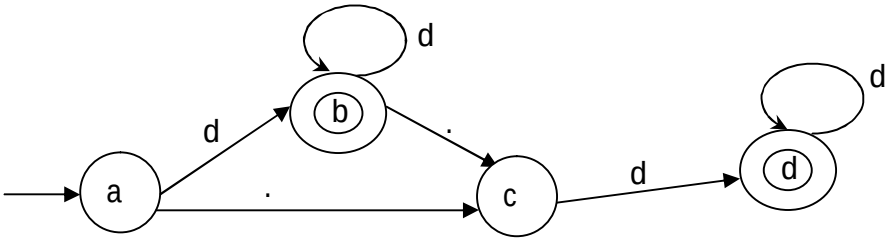
Regular expression is int.

2. But to recognize a valid C identifier the DFA might be as the one that follows:



Mark that the regular expression for valid C identifiers is l|_(l|_|d)*, where l stands for a|b|c|…|z|A|B|C|…|Z and d stands for 0|1|2|…|9.

3. And a DFA for simple floating point numbers might take the following form:



Mark that the regular expression is dd*|d*.dd*, where d stands for 0|1|2|…|9, and mark also that the DFA has two final states.

Sample implementation of the DFA:

```
int num_rec(char lex[20])

{       int i, l, s;
        i=0;
        if(isdigit(lex[i]))
             {s=1; i++;}
        else
           if(lex[i]=='.') {s=2; i++;}
           else s=0;

        l=strlen(lex);
```

```
if(s==1)
   for(;i<l;i++)
   {  if(isdigit(lex[i]))
         s=1;
      else
         if(lex[i]=='.')
                {s=2; i++; break;}
         else {s=0; break;}}
```

```
if(s==2)
    if(isdigit(lex[i]))
      {s=3; i++;}
    else s=0;

if(s==3)
   for(;i<l;i++)
    {  if(isdigit(lex[i]))
          s=3;
       else {s=0; break;}}

if(s==3) s=1;
return s; }
```

Some useful Library Functions from ctype.h:

S.N.	Function & Description
1	int isalnum(int c) This function checks whether the passed character is alphanumeric.
2	int isalpha(int c) This function checks whether the passed character is alphabetic.
3	int isdigit(int c) This function checks whether the passed character is a numeral.
4	int islower(int c) This function checks whether the passed character is a lowercase letter.