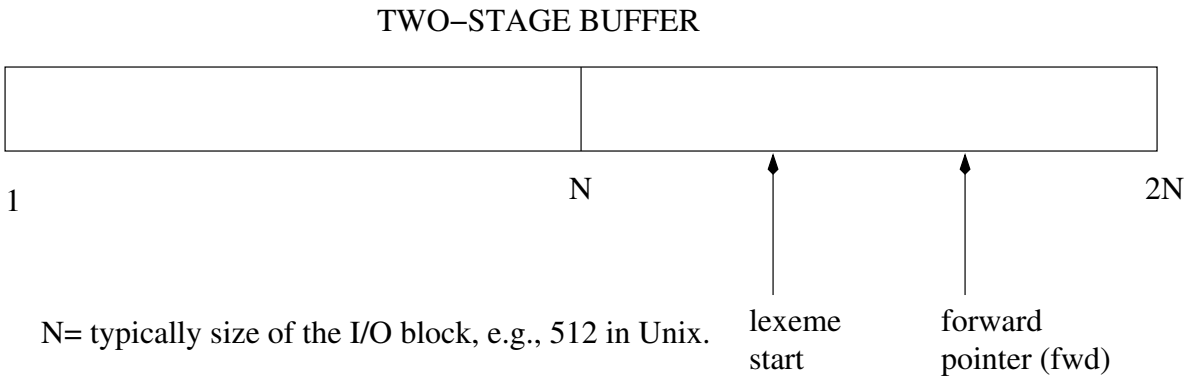
Lexical Analysis

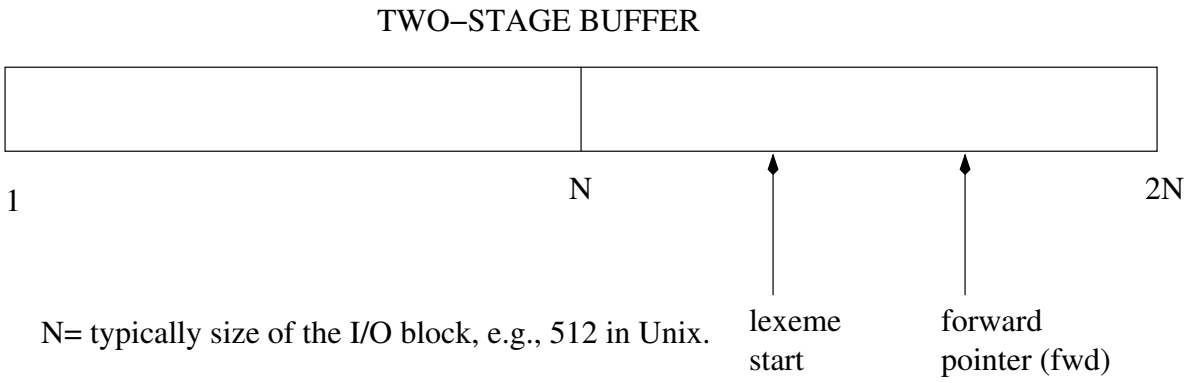
- Converting a character stream into a token stream (token recognition).
- 1. Ad hoc (customized) lex analyzers

  2. Lexical analyzer generators (e.g., `lex`)
- Lexical analysis is the only I/O-bound stage in compiling; need efficient I/O handling.
- Customized lexical analysis:

Take care of I/O efficiently: buffering



Advance `fwd` until a delimiter. Anything between `lexeme start` and `fwd` is a token.



fwd $>$ 2N ? Load first half and reset fwd

fwd $>$ N ? Load second half and advance

1 $<$ fwd $<$ 2N and no more characters : use sentinels.

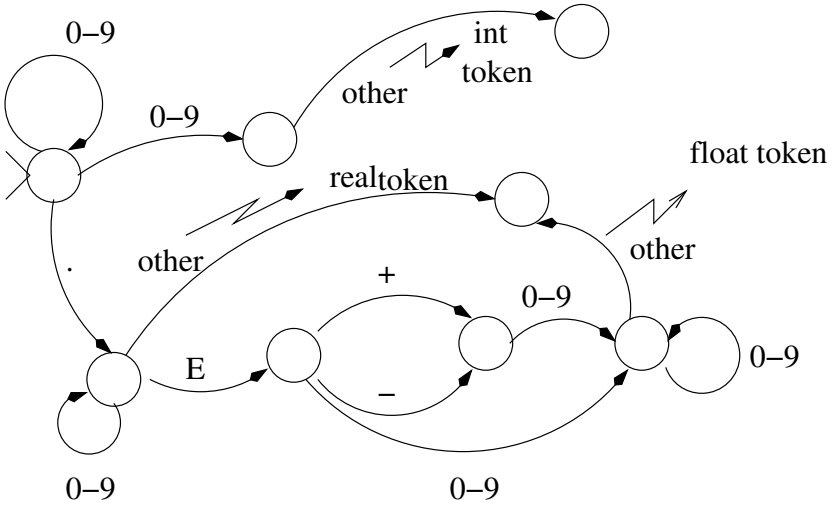
what if token size can be greater than N ?

- LEX ANALYSIS USING FSMs

  automated tools for I/O handling and pattern recognition.

  Patterns for tokens in most PLs are regular; FSMs can be used for efficient recognition.

- Design a grammar for token; write a NFA for it; convert to DFA; then minimize the DFA.



A FINITE–STATE TRANSDUCER FOR NUMBERS

(A Mealy Machine)

USE OF EXTENDED REG. EXP. NOTATION in LeX

- It is only for notational convenience; does not extend the power above type-3 languages.

ex: digits, non-digits, and letters

```
[0-9]      [^0-9]     [A-Za-z]
```

ex: decimal with up to 5 digits in fraction.

```
{digit}+\.{digit}{1,5}
```

ex: `IF` in Fortran `IF=3`

```
IF(I,J)=3   IF(I+J,3)=4     IF(I)=4
```

```
IF(A.EQ.B) A=3
```

```
IF/(\(({num}|{id}|{op})(,|{num}|{id}|{op}*\))?=
```

- But this is only an approximation; you need to know *expression syntax* of FORTRAN.

- Ambiguity in pattern match: more than one pattern is satisfied

Use the longest match: `"//".*` matches till the end of line.

- OPEN HASH TABLE: no overflow in hash table. In case of collision, form a chain of items with the same hash value.

```
/* open hash for symbol table */
#define HASHSIZE 997
#define EOS '\0'

int hash(s)
char *s;
{
  int hv = 0;
  int i;
  for (i=0; s[i] != EOS; i++)
  {
    int v = (hv>>28) ^ (s[i] & 0xf);
```

```
    hv = (hv << 4) | v; }
  hv = hv & 0x7fffffff;
  return hv % HASHSIZE; }
main ()
{
  char s[50];
  while (1)
  {
    printf("enter a string of chars\n");
    scanf("%s",s);
    printf("hash index of the string: %d\n",hash(s)); } }
  ..
symtab *symbols[HASHSIZE];
  ..
if (symbols[hash("temp")]=NULL){ insert...};
```