

Compiler Lab Report

Tanvir Hasan
Reg:12101005

August 12, 2015

Contents

0.1	Assignment 1: Token	2
0.1.1	Description	2
0.1.2	code	3
0.1.3	Input	6
0.1.4	Output	6
0.2	Assignment 2: Postfix,Infix,Prefix	7
0.2.1	Description	7
0.2.2	Code	7
0.2.3	Output:	10
0.3	Assignment 3: Flex	11
0.3.1	Description	11
0.3.2	Identification of C Token:	11
0.3.3	Convert Roman numbers to Decimal numbers	14
0.3.4	Identification of various number formats	16

0.1 Assignment 1: Token

0.1.1 Description

Token: In every Natural languages, there are some grammatical rules to write something. Like human language, C has some rules to write code properly. These rules are known as token.

tokens are of six types-

1. **Keywords:** A variable is a meaningful name of data storage location in computer memory. When using a variable you refer to memory address of computer. Example: do,while,for
2. **Identifiers:** The term identifier is usually used for variable names. Example: main,A,AB
3. **Constants::** Constants are expressions with a fixed value. Example: 1,23,0
4. **Strings:** Sequence of characters. Example: Ok,
5. **Special symbols:** Symbols other than the Alphabets and Digits and white-spaces. Example: `() ,`
6. **Operators:** A symbol that represent a specific mathematical or non-mathematical action. Example: `+, /,-,*`

0.1.2 code

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 template <class T> void print_vector(T v)
4 {
5     int sz=v.size()-1;
6     for (int i=0;i<=sz;i++)cout<<v[i]<<" \n" [ i==sz ];
7 }
8 class Token
9 {
10     private:
11         map<string, bool>token;
12         map<char, bool>operators;
13         string code;
14         set<string>Operators, keywords, Identifiers, strings, constants;
15         set<string>specialSymbol;
16         void init();
17         void process();
18         bool isKeyWord(string str);
19         bool isOperator(char str);
20         bool isSpecialSymbol(char str);
21     public:
22         Token(string file_name);
23         vector<string>getKeywords();
24         vector<string>getIdentifiers();
25         vector<string>getStrings();
26         vector<string>getConstants();
27         vector<string>getOperators();
28         vector<string>getSpecialSymbol();
29 };
30 void Token::init()
31 {
32     FILE *fp=fopen("keyword.txt", "r");
33     while (!feof(fp))
34     {
35         char tk[100];
36         fscanf(fp, "%s", tk);
37         token[tk]=true;
38     }
39     fclose(fp);
40     fp=fopen("operator.txt", "r");
41     while (!feof(fp))
42     {
43         char c[3];
44         fscanf(fp, "%s", c);
45         operators[c[0]]=true;
46     }
47     fclose(fp);
48 }
49 bool Token::isKeyWord(string str)
50 {
51     return token.find(str)!=token.end();
52 }
53 bool Token::isOperator(char str)
54 {
```

```

55     return operators.find(str)!=operators.end() and str!=' ';
56 }
57 void Token::process()
58 {
59     for(int i=0; i<code.size(); i++)
60     {
61         string str="";
62         if(isdigit(code[i]))
63         {
64             while(isdigit(code[i]))
65             {
66                 str+=code[i];
67                 i++;
68             }
69             constants.insert(str);
70             i--;
71         }
72         else if(code[i]=='\"')
73         {
74             str+='\\"';
75             i++;
76             while(code[i]!='\"')
77             {
78                 str+=code[i];
79                 i++;
80             }
81             str+='\\"';
82             strings.insert(str);
83         }
84         else if(isalpha(code[i]) || code[i]=='_')
85         {
86             while(isalpha(code[i]) || isdigit(code[i]) || code[i]=='_')
87             {
88                 str+=code[i];
89                 i++;
90             }
91             if(isKeyWord(str))keywords.insert(str);
92             else Identifiers.insert(str);
93             i--;
94         }
95         else
96         {
97             if(isOperator(code[i]))
98             {
99                 while(isOperator(code[i]))
100                 {
101                     str+=code[i];
102                     i++;
103                 }
104                 Operators.insert(str);
105                 i--;
106             }
107             else if(code[i]!=' ' and code[i]!='\n')
108             {
109                 str=" ";
110                 str+=code[i];

```

```

111         specialSymbol.insert(str);
112     }
113 }
114 }
115 }
116 Token::Token(string filename)
117 {
118     init();
119     FILE *fp=fopen(filename.c_str(),"r");
120     code="";
121     while(!feof(fp))
122     {
123         char c;
124         fscanf(fp,"%c",&c);
125         code+=c;
126     }
127     fclose(fp);
128     process();
129 }
130 vector<string>Token::getKeywords()
131 {
132     vector<string>ret;
133     for(auto x:keywords)ret.push_back(x);
134     return ret;
135 }
136 vector<string>Token::getIdentifiers()
137 {
138     vector<string>ret;
139     for(auto x:Identifiers)ret.push_back(x);
140     return ret;
141 }
142 vector<string>Token::getConstants()
143 {
144     vector<string>ret;
145     for(auto x:constants)ret.push_back(x);
146     return ret;
147 }
148 vector<string>Token::getStrings()
149 {
150     vector<string>ret;
151     for(auto x:strings)ret.push_back(x);
152     return ret;
153 }
154 vector<string>Token::getOperators()
155 {
156     vector<string>ret;
157     for(auto x:Operators)ret.push_back(x);
158     return ret;
159 }
160 vector<string>Token::getSpecialSymbol()
161 {
162     vector<string>ret;
163     for(auto x:specialSymbol)ret.push_back(x);
164     return ret;
165 }
166 int main()

```

```

167 {
168     Token t("input.cpp");
169     printf("Keywords (%d): ", t.getKeywords().size());
170     print_vector(t.getKeywords());
171     printf("Identifiers (%d): ", t.getIdentifiers().size());
172     print_vector(t.getIdentifiers());
173     printf("Constants (%d): ", t.getConstants().size());
174     print_vector(t.getConstants());
175     printf("Strings (%d): ", t.getStrings().size());
176     print_vector(t.getStrings());
177     printf("Special symbols (%d): ", t.getSpecialSymbol().size());
178     print_vector(t.getSpecialSymbol());
179     printf("Operators (%d): ", t.getOperators().size());
180     print_vector(t.getOperators());
181     return 0;
182 }

```

0.1.3 Input

```

1 int main()
2 {
3     int A,B;
4     printf("Enter 1st Number: ");
5     scanf("%d",&A);
6     printf("Enter 2nd Number ");
7     scanf("%d",&B);
8     int result=A+B;
9     A++;
10    printf("result %d\n",result);
11    return 0;
12 }

```

0.1.4 Output

```

"G:\OTHER\all prb solve file\Dropbox\Dropbox\CSE 403\Assignment 1\Token.exe"
Keywords (2): int return
Identifiers (6): A B main printf result scanf
Constants (1): 0
Strings (4): "%d" "Enter 1st Number: " "Enter 2nd Number" "result %d\n"
Special symbols (8): & < > , ; = < >
Operators (2): + ++

Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.

```

0.2 Assignment 2: Postfix, Infix, Prefix

0.2.1 Description

Infix, Postfix and Prefix notations are three different but equivalent ways of writing expressions. It is easiest to demonstrate the differences by looking at examples of operators that take two operands.

Infix notation: $X + Y$: Operators are written in-between their operands. This is the usual way we write expressions. An expression such as $A * (B + C) / D$ is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer."

Postfix notation (also known as "Reverse Polish notation"): $X Y +$: Operators are written after their operands. The infix expression given $A * (B + C) / D$ is equivalent to $A B C + * D /$.

Prefix notation (also known as "Polish notation"): $+ X Y$: Operators are written before their operands. The expressions $A * (B + C) / D$ are equivalent to $/ * A + B C D$.

0.2.2 Code

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int get_operator_priority(char c)
4 {
5     int p = 0;
6     switch (c)
7     {
8         case '*':
9         case '/':
10            return 2;
11         case '+':
12         case '-':
13            return 1;
14         default:
15            return 0;
16     }
17 }
18 class PolishNotation
19 {
20     private:
21         string Expre;
22         string prefixToInfix();
23     public:
24         PolishNotation(string str);
25         string infixToPostfix();
26         string infixToPrefix();
27         string prefixToPostfix();
28 };
29
```



```

30 PolishNotation::PolishNotation(string str)
31 {
32     Expre=str;
33 }
34
35 string PolishNotation::infixToPostfix()
36 {
37     stack<char>Stk;
38     string ret="";
39     for(int i=0;i<Expre.size();i++)
40     {
41         char curr=Expre[i];
42         if(isalpha(curr))ret+=curr;
43         else if(curr=='(')Stk.push(curr);
44         else if(curr==')')
45         {
46             char d=Stk.top();
47             Stk.pop();
48             while(d!='(')
49             {
50                 ret+=d;
51                 d=Stk.top();
52                 Stk.pop();
53             }
54         }
55         else
56         {
57             while(!Stk.empty() and get_operator_priority(Stk.top())>=
get_operator_priority(curr))
58             {
59                 ret+=Stk.top();
60                 Stk.pop();
61             }
62             Stk.push(curr);
63         }
64     }
65     while(!Stk.empty())
66     {
67         ret+=Stk.top();
68         Stk.pop();
69     }
70     return ret;
71 }
72
73 string PolishNotation::infixToprefix()
74 {
75     stack<char>Stk;
76     string ret="";
77     for(int i=Expre.size()-1;i>=0;i--)
78     {
79         char curr=Expre[i];
80         if(isalpha(curr))ret+=curr;
81         else if(curr==')')Stk.push(curr);
82         else if(curr=='(')
83         {
84             char d=Stk.top();

```

```

85         Stk.pop();
86         while(d!='')
87         {
88             ret+=d;
89             d=Stk.top();
90             Stk.pop();
91         }
92     }
93     else
94     {
95         while(!Stk.empty() and get_operator_priority(Stk.top())>=
get_operator_priority(curr))
96         {
97             ret+=Stk.top();
98             Stk.pop();
99         }
100         Stk.push(curr);
101     }
102 }
103 while(!Stk.empty())
104 {
105     ret+=Stk.top();
106     Stk.pop();
107 }
108 reverse(ret.begin(),ret.end());
109 return ret;
110 }
111
112 string PolishNotation::prefixToInfix()
113 {
114     stack<string>Stk;
115     for(int i=Expre.size()-1;i>=0;i--)
116     {
117         char curr=Expre[i];
118         if(isalpha(curr))Stk.push(string("")+curr);
119         else
120         {
121             string A=Stk.top();
122             Stk.pop();
123             string B=Stk.top();
124             Stk.pop();
125             string C="";
126             if(curr=='+' or curr=='-')C=A+curr+B;
127             else if(curr=='/')C="("+A+")"+curr+"("+B+")";
128             else if(curr=='*')C="("+A+")"+curr+B;
129             Stk.push(C);
130         }
131     }
132     return Stk.top();
133 }
134
135 string PolishNotation::prefixToPostfix()
136 {
137     Expre=prefixToInfix();
138     cout<<Expre<<endl;
139     return infixToPostfix();

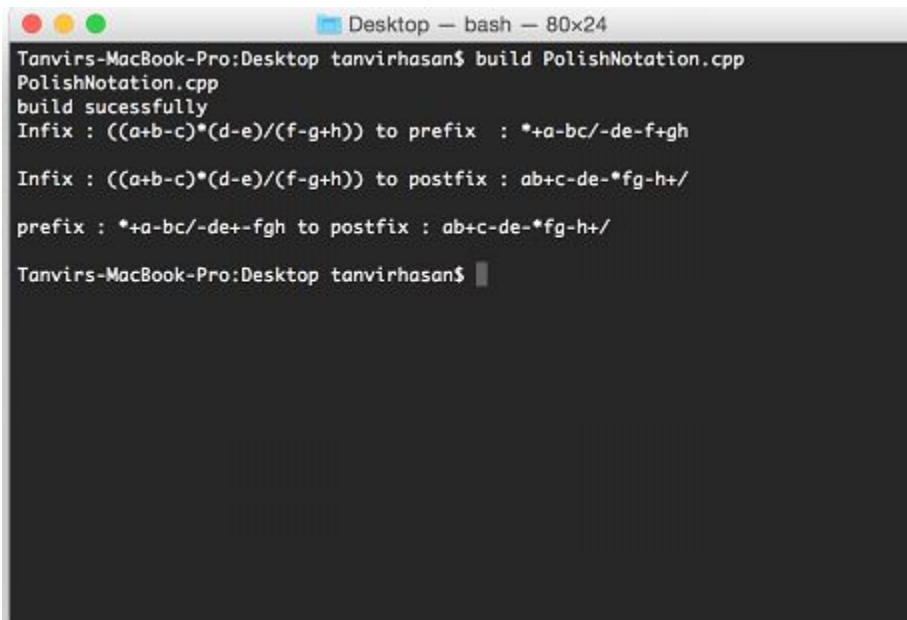
```

```

140 }
141 int main()
142 {
143     string infix="((a+b-c)*(d-e)/(f-g+h))";
144     string prefix="*+a-bc/-de+-fgh";
145     PolishNotation Infix(infix);
146     cout<<"Infix : "<<infix<<" to prefix : "<<Infix.infixToPrefix()<<endl<<endl;
147     cout<<"Infix : "<<infix<<" to postfix : "<<Infix.infixToPostfix()<<endl<<endl;
148     PolishNotation Prefix("AB+CD+*");
149     cout<<"prefix : "<<prefix<<" to postfix : "<<Prefix.prefixToPostfix()<<endl<<
    endl;
150     return 0;
151 }

```

0.2.3 Output:



```

Tanvirs-MacBook-Pro:Desktop tanvirhasan$ build PolishNotation.cpp
PolishNotation.cpp
build successfully
Infix : ((a+b-c)*(d-e)/(f-g+h)) to prefix : *+a-bc/-de-f+gh

Infix : ((a+b-c)*(d-e)/(f-g+h)) to postfix : ab+c-de-*fg-h+/

prefix : *+a-bc/-de+-fgh to postfix : ab+c-de-*fg-h+/

Tanvirs-MacBook-Pro:Desktop tanvirhasan$

```

0.3 Assignment 3: Flex

0.3.1 Description

Flex is a tool for generating scanners. A scanner, sometimes called a tokenizer, is a program which recognizes lexical patterns in text. The flex program reads user-specified input files, or its standard input if no file names are given, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called rules. Flex generates a C source file named, "lex.yy.c", which defines the function yylex(). The file "lex.yy.c" can be compiled and linked to produce an executable. When the executable is run, it analyzes its input for occurrences of text matching the regular expressions for each rule. Whenever it finds a match, it executes the corresponding C code.

0.3.2 Identification of C Token:

Code

```
1 %{
2  #include <stdio.h>
3  #include <string.h>
4
5  int TotalKeyword=0;
6  int TotalIdentifier=0;
7  int TotalOperator=0;
8  int TotalConstant=0;
9  int TotalPunctuation=0;
10 int TotalParenthesis=0;
11
12 char KeywordList[1000];
13 char IdentifierList[1000];
14 char OperatorList[1000];
15 char ConstantList[1000];
16 char PunctuationList[1000];
17 char ParenthesisList[1000];
18 %}
19
20 token auto|break|c(char|co(nst|ntinue))|do(uble)?|e(lse|num|xtern)|f(float|or)|i(f|nt)
    |long|re(gister|turn)|s(i(gned|eof)|t(atic|ruct)|witch)|typedef|unsigned|vo(id|
    latile)|while|_Packed
21 identifier [a-zA-Z][a-zA-Z0-9]*
22 op [-|+|\*|/|^|=]
23 constant [0-9]+
24 paranthesis [{|}|\\[|\\]|\\(|\\)|]
25 punctuation [;|:|,]
26
27 %%
28 {token} {
29     TotalKeyword++;
30     strcat(KeywordList, yytext);
31     strcat(KeywordList, ",");
32 }
33 {identifier} {
34     TotalIdentifier++;
35     strcat(IdentifierList, yytext);
```

```

36     strcat(IdentifierList , ",");
37 }
38 {constant} {
39     TotalConstant++;
40     strcat(ConstantList , yytext);
41     strcat(ConstantList , ",");
42 }
43 {parenthesis} {
44     TotalParenthesis++;
45     strcat(ParenthesisList , yytext);
46     strcat(ParenthesisList , ",");
47 }
48 {op} {
49     TotalOperator++;
50     strcat(OperatorList , yytext);
51     strcat(OperatorList , ",");
52 }
53 {punctuation} {
54     TotalPunctuation++;
55     strcat(PunctuationList , yytext);
56     strcat(PunctuationList , ",");
57 }
58
59
60 %%
61
62 int main(int argc , char **argv){
63     printf("%s",argv[1]);
64     FILE *fp=fopen(argv[1] ,"r");
65     yyin = fp;
66     yylex();
67     printf("\n\nKeywords (%d): %s \n", TotalKeyword , KeywordList);
68     printf("Identifiers (%d): %s \n", TotalIdentifier , IdentifierList);
69     printf("Constants (%d): %s \n", TotalConstant , ConstantList);
70     printf("Operators (%d): %s \n", TotalOperator , OperatorList);
71     printf("Parenthesis (%d): %s \n", TotalParenthesis , ParenthesisList);
72     printf("Punctuation (%d): %s \n", TotalPunctuation , PunctuationList);
73     return 0;
74 }
75
76 int yywrap(void){
77     return 1;
78 }
79
80 int yyerror(void){
81     printf("Error\n");
82     exit(1);
83 }

```

Input

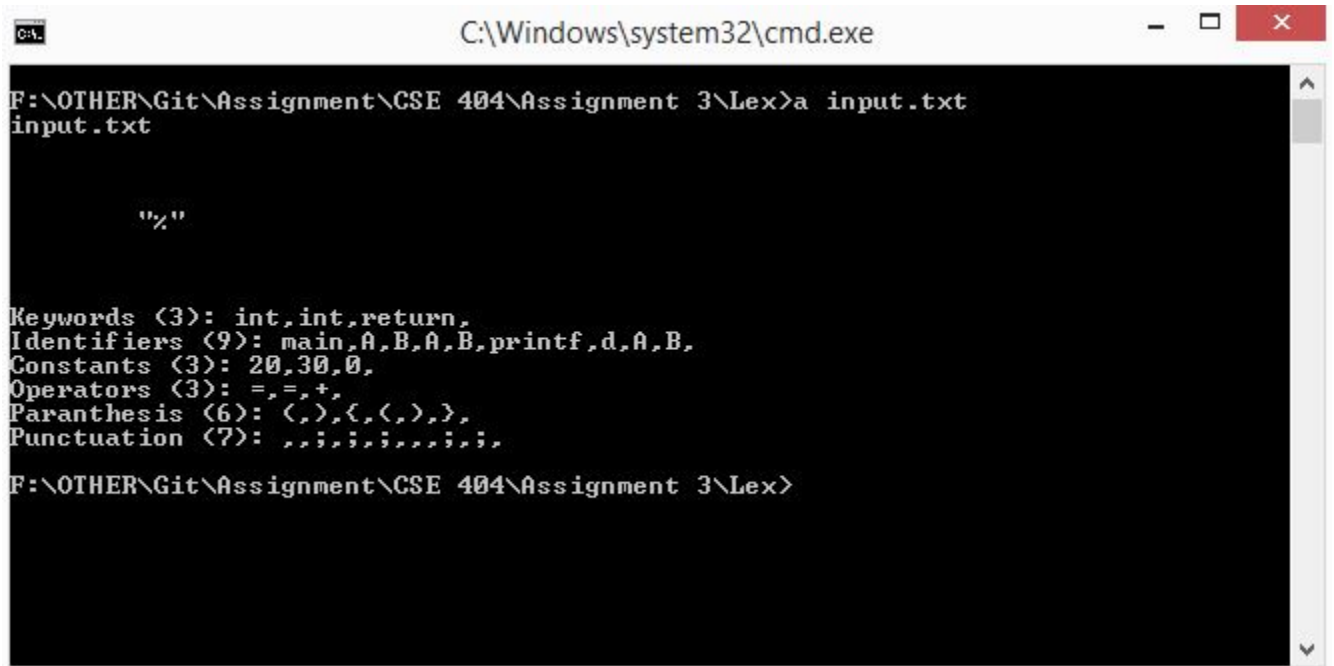
```

1 int main(){
2     int A,B;
3     A=20;
4     B=30;
5     printf("%d",A+B);

```

```
6 return 0;
7 }
```

Output



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the execution of a Lex program. The prompt is "F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>a input.txt". The output is as follows:

```
input.txt

    "%"
```

Keywords <3>: int,int,return,
Identifiers <9>: main,A,B,A,B,printf,d,A,B,
Constants <3>: 20,30,0,
Operators <3>: =,=,+
Paranthesis <6>: <,>,<,<,>,>,
Punctuation <7>: ,,;,;,;,;,;,;,;

The prompt returns to "F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>".

0.3.3 Convert Roman numbers to Decimal numbers

Code

```
1  %{
2      int sum=0;
3  %}
4
5  %%
6
7  I {
8      sum += 1;
9  }
10 IV {
11     sum += 4;
12 }
13 V {
14     sum += 5;
15 }
16 IX {
17     sum += 9;
18 }
19 X {
20     sum += 10;
21 }
22 XL {
23     sum += 40;
24 }
25 L {
26     sum += 50;
27 }
28 XC {
29     sum += 90;
30 }
31 C {
32     sum += 100;
33 }
34 CD {
35     sum += 400;
36 }
37 D {
38     sum += 500;
39 }
40 CM {
41     sum += 900;
42 }
43 M {
44     sum += 1000;
45 }
46
47 [\n] {
48     return sum;
49 }
50
51 %%
52 int main (void) {
```

```

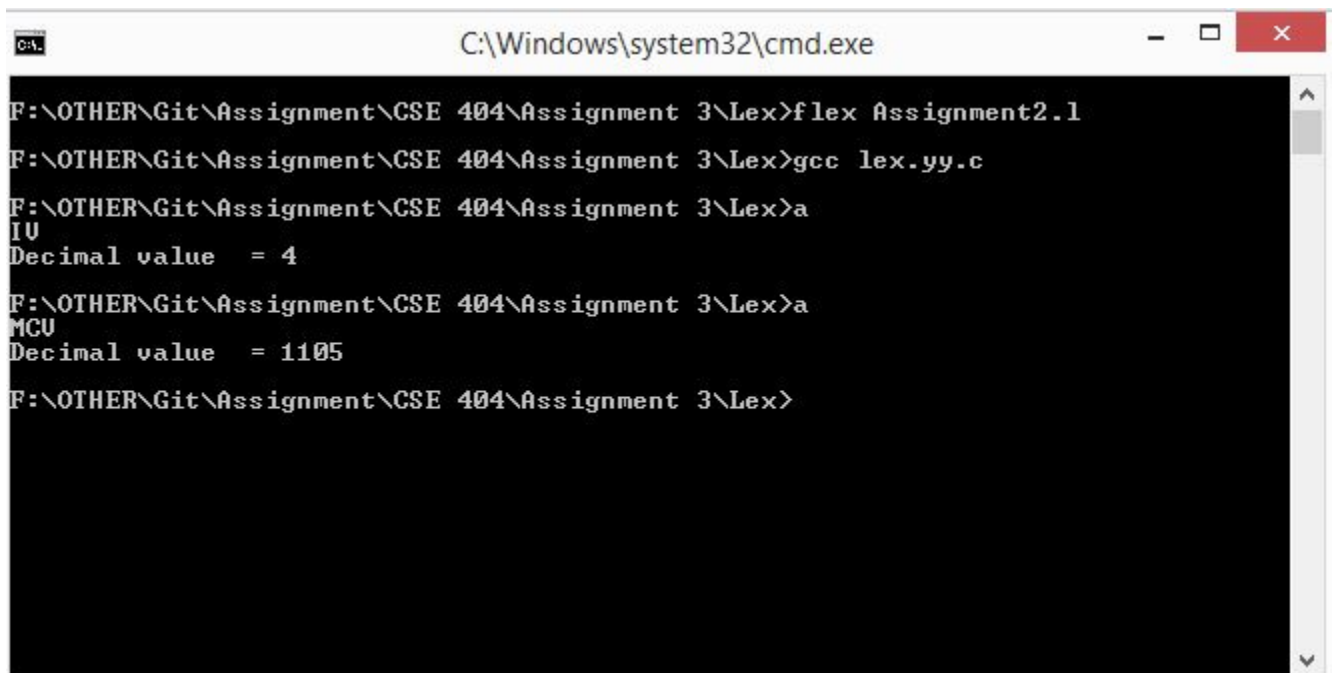
53
54 yylex ();
55
56 printf ("Decimal value = %d\n", sum);
57 return 0;
58 }
59
60 int yywrap(void){
61     return 0;
62 }
63
64 int yyerror(void){
65     printf("Error\n");
66     exit(1);
67 }

```

Input

IV
MCV

Output



```

C:\Windows\system32\cmd.exe
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>flex Assignment2.1
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>gcc lex.yy.c
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>a
IV
Decimal value = 4
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>a
MCV
Decimal value = 1105
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>

```


0.3.4 Identification of various number formats

Code

```
1 %{
2  #include<stdio.h>
3  #include<stdlib.h>
4  int pos = 0;
5  int result[100];
6  char Ans[][30]={ " ", "ODD", "DIV4", "Singed", "Decimal", "Scientefic", "HEX", "Overflow",
7  "String", "Identifier", "Unknown" };
8  %%
9  digit [0-9]{4,}
10 decimal [0-9]+\.[0-9]+
11 singed (-|\+)[0-9]+
12 hex [0-9A-F]+
13 string [0-9][0-9a-zA-Z]*
14 scientefic ({singed}(e|E){singed})
15 identifier [a-zA-Z][0-9a-zA-Z]*
16 unknown [^(-\+0-9a-zA-Z \n)]
17
18 %%
19 [ ] {}
20 {digit} {
21  int number=atoi(yytext);
22  int len=yylen;
23  if(number%2){
24    result[pos]=1;
25  }else if(number%4==0){
26    result[pos]=2;
27  }else if(len>4){
28    result[pos]=7;
29  }else{
30    result[pos]=10;
31  }
32  pos++;
33 }
34 {singed} {
35  result[pos]=3;
36  pos++;
37 }
38 {decimal} {
39  result[pos]=4;
40  pos++;
41 }
42 {scientefic} {
43  result[pos]=5; pos++;
44 }
45 {hex} {
46  int len=yylen;
47  if(len>4){
48    result[pos]=7;
49  }else{
50    result[pos]=6;
51  }
```

```

52     pos++;
53 }
54 {string} {
55     result[pos]=8; pos++;
56 }
57 {identifier} {
58     result[pos]=9; pos++;
59 }
60 {unknown} {
61     result[pos]=10; pos++;
62 }
63 [\n] {
64     return 0;
65 }
66 %%
67
68
69 int main(int argc, char **argv){
70     yyin = stdin;
71     yylex();
72     int i=0;
73     for(i=0;i<pos;i++){
74         printf("%s\n",Ans[result[i]]);
75     }
76     return 0;
77 }
78
79 int yywrap(void){
80     return 0;
81 }

```

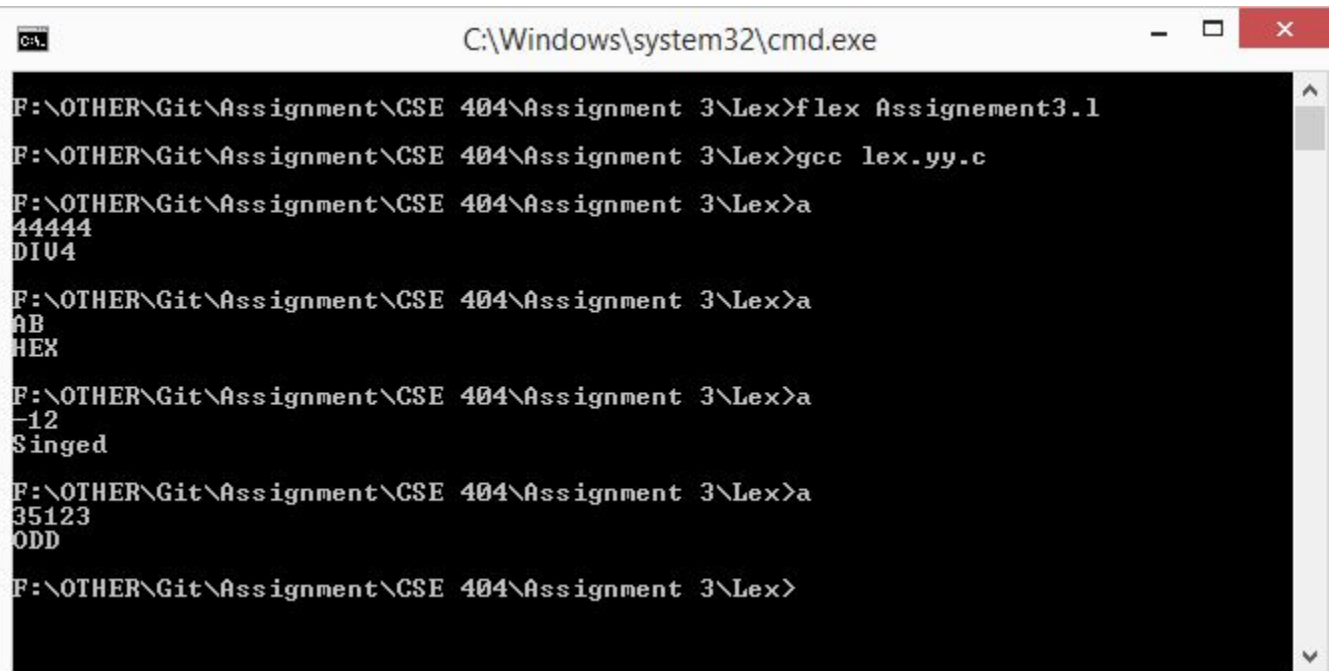
Input

```

4444
AB
-12
35123

```

Output



```
C:\Windows\system32\cmd.exe

F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>flex Assignement3.1
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>gcc lex.yy.c
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>a
44444
DIU4
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>a
AB
HEX
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>a
-12
Singed
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>a
35123
ODD
F:\OTHER\Git\Assignment\CSE 404\Assignment 3\Lex>
```