EXPERIMENT 2

CSE-304 Tokenization

2.1 OBJECTIVE(S):

- To understand the concept of Tokenization for designing Compiler
- To write a program in C++ to Tokenize a code stream written in C.
    - Scan the input program (code stream) and identify Tokens
    - Insert tokens into Symbol Table, print the whole symbol table in console for each insertion.
    - Generate different files for different Tokens mentioning the lexeme and its line number.
    - Generate lexical errors with the line number and print it in the console.

2.2 EQUIPMENT/SETUP:

- A Computer (PC) with C++ compiler and corresponding IDE

3.3 BACKGROUND:

Tokenization is a way of separating a piece of text into smaller units called tokens.

Token:
A token is a pair consisting of a token name and an optional attribute value. <TOKEN, ATTRIBUTE>

Lexeme:
A Lexeme is a sequence of characters (actual character set).

Pattern:
A pattern is a description of the form that the lexemes of a token may take.

Symbol Table:
A symbol-table is a data structure maintained by compilers in order to store information about the occurrence of various identifiers, functions, objects, etc.

Lexical error:
if any lexeme does not match with any pattern described.

2.4 PROBEM DESSCRIPTION IN DETAIL:

The stream of code in C and the file name are given below.

File name: sample_input2.txt

Code stream:

```
void prime(int n){
  int i=3,c+;
  auto count=0;

  for( count = 2 ; count <= n ;  )
  {
    for  ( c = 2 ; c <= i - 1 ; c++ )
    {
      if( i%c == 0 )
        break;
    }
    if( c == i )
    {
      printf("%d & %d\n ", i/2,i*2);
      count++;
    }
    else if  (){
       i+=1
    }
    els print("done");!
    //i++;
  }
}

int main()
{
  int n;
  float m=2.3;
  unsigned char b=2;
  printf("b<<2 = %d & b>>5 = %d %d\n",b<<2,b >>5,b^b);

  printf("Enter the number of prime numbers required\n");
  scanf("%d"*=,&n);

  if( n >= 1 )
  {
    printf("First %d prime numbers are :\n",n);
    printf("2\n");
  }
  else prime(n);

  return 0;
}
```

2.4.1 Instruction table

Instruction table is presented in Table 1.1.

Table 1.1: Instructions of the tokens to be handled

Serial	Token	Tokens to be handled
1	KEYWORD	Identify the following keywords **if, else, else if, for, while, do, break, int, char, float, double, unsigned, const, return, include**
2	FUNCTION	Identify functions: For all types of function calling and declarations.
3	IDENTIFIER	Identify identifiers
4	LITERAL	Identify literals: "Hello World!"
5	NUMBER	Identify numbers: 51,2.3
6	OPERATOR	Identify **arithmetic, logical, bitwise, and assignment operators**: Arithmetic operators: +, -, *, % Logical operators: &&, || Bitwise operators: &, |, <<, >> Assignment operators: =, +=, /=, %=

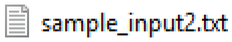
- Handle **only those mentioned** in the table.
- First check Keywords in the given code stream.
- Do not check the Function and Identifier before checking Keywords.
- If function name is checked before the keyword, "**if ( )**" will be detected as a function name.

2.4.2 Input-Output files

Implemented program code using C++, "**lab2_ID.cpp**".

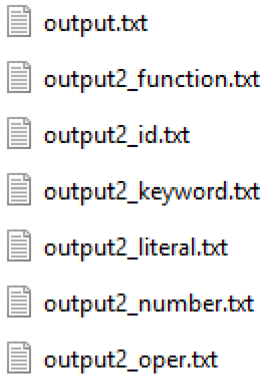
lab2_202300001.cpp

Sample input file "**sample_input1.txt**" containing the code stream written in C.



sample_input2.txt

Output files (7 files) as follows.



output.txt

output2_function.txt

output2_id.txt

output2_keyword.txt

output2_literal.txt

output2_number.txt

output2_oper.txt

The file “**output.txt**” contains the whole symbol table for each insertion.

The other **six different output files** contain different Tokens mentioning the lexeme and its line number.

Also print lexical errors with the line number in the **console**.

2.6 EXPERIMENTAL PROCEDURE:

- Implement the code by follow the instructions provided in this manual and ensure proper understanding.

2.7 LAB PRACTICE/EXPERIMENT/IMPLEMENTATION:

Please write and execute the following code. **Modify where necessary.**

2.7.1 Include headers of your program

```
#include<bits/stdc++.h>
using namespace std;

/// mod val 12
```

2.7.2 Define “output.txt” as output file containing the Symbol Table

```
ofstream fileout("output.txt", ios_base :: out);
```

2.7.3 Define a function to check patterns for Identifier or Keywords

```
bool idOrKey(char ch)
{
   if((ch>='0' && ch<='9') || (ch>='a' && ch<='z') || (ch>='A' && ch<='Z') || ch=='_')
   {
      return true;
   }
   else return false;
}
```

2.7.4 *Define* a function to check patterns for Digits

```
bool digit(string a)
{
   int cnt=0;
   int n=a.size();
   if(a[0]!='-' && !(a[0]>='0' && a[0]<='9') && a[0]!='.') return false;
   if(a[0]=='.') cnt++;
   for (int i=1; i<n; i++)
   {
      if(!(a[i]>='0' && a[i]<='9') && a[i]!='.') return false;
      if(a[i]=='.') cnt++;
   }
   if(cnt>1) return false;
   return true;
}
```

2.7.5 *Define* a function to check patterns for Identifiers

```
bool id(string s)
{
   for (int i=0; i<s.size(); i++)
   {
      if(!idOrKey(s[i])) return false;
   }
   if(s.size()>0) if(s[0]>='0' && s[0]<='9') return false;
   if(s.empty()) return false;
   return true;
}
```

2.7.5 *Define* a function to check patterns for Operators

```
bool op(string ch)
{
   vector<string>op= {"+", "-", "*", "%","&&","||","&", "|", "<<", ">>","=", "+=", "/=",
"%=","!","!=","-=","==",">","<","!"};
   if ( find(op.begin(), op.end(), ch) != op.end() )
      return true;
   else
      return false;
}
```

2.7.6 *Define* a function to check patterns for White Space

```
bool space(char ch)
{
   if(ch==' ' || ch=='\n' || ch=='\t') return true;
   return false;
}
```

2.7.7 *Define* a function to check patterns for Punctuation

```
/*
bool punc(char ch)
{
   vector<char>punc= {',', '.', ';','?',':'};
   if ( find(punc.begin(), punc.end(), ch) != punc.end() )
      return true;
   else
      return false;
}
*/
```

2.7.8 *Define* a function to check patterns for Keywords

```
bool keyword(string a)
{
  vector<string>key= {"if", "else", "else if", "for", "while", "do", "break", "int",
"char","float", "double", "unsigned", "const", "return", "include"};
  if ( find(key.begin(), key.end(), a) != key.end() )
    return true;
  else
    return false;
}
```

2.7.9 *Define* a function to check patterns for Interrupt

```
bool interrupt(char ch)
{
  vector<char>bracket= {'(',')','{','}','[',']',',', '.', ';','?',':'};
  if ( find(bracket.begin(), bracket.end(), ch) != bracket.end() )
    return true;
  else
    return false;
}
```

2.7.10 *Define* a Class to handle Symbol Info.

```
class SymbolInfo
{
  string symbol, symbolType;

public:
  SymbolInfo(string symbol, string symbolType)
  {
    this->symbol = symbol;
    this->symbolType = symbolType;
  }
  string getSymbol()
  {
    return symbol;
  }
  string getSymbolType()
  {
    return symbolType;
  }
  void setSymbol(string symbol)
  {
    this->symbol = symbol;
  }
  void setSymbolType(string symbolType)
  {
    this->symbolType = symbolType;
  }
};
```

2.7.11 *Define* a Class to implement Symbol Table

```
class SymbolTable
{
    vector<SymbolInfo>table[12];

public:
    void insertVal(string symbol, string symbolType)
    {
        bool b=lookup(symbol);
        if(b==false)
        {
            SymbolInfo obj = SymbolInfo(symbol, symbolType);
            int hashVal = hashFunc(symbol);
            table[hashVal].push_back(obj);
            int pos = table[hashVal].size();
            fileout<<"Inserted at position "<<hashVal<<","<<pos-1<<endl;
            print();
        }
        else
        {
            fileout<<"Value already exists"<<endl;
        }
    }

    bool lookup(string symbol)
    {
        int hashVal = hashFunc(symbol);
        bool b = false;

        for (int j = 0; j < table[hashVal].size(); j++)
        {
            if(table[hashVal][j].getSymbol()==symbol)
            {
                fileout<<"Found at "<<hashVal<<","<<j<<endl;
                b = true;
            }
        }
        return b;
        /// if(b==false) cout<<"Symbol not found in the Table."<<endl;
    }

    void del(string symbol)
    {
        int pos = 0;
        int hashVal = hashFunc(symbol);
        bool b = false;

        for(auto it = table[hashVal].begin(); it!= table[hashVal].end(); it++)
        {
            if(it->getSymbol()==symbol)
            {
                fileout<<"Deleted from "<<hashVal<<","<<pos<<endl;
                table[hashVal].erase(it);
                b = true;
                break;
            }
            pos++;
        }
        if(b==false)
            fileout<<"Symbol not found in the Table."<<endl;
    }

    void print()
    {
        for(int i=0; i<12; i++)
        {
            fileout<<i<<" -> ";
            for(int j=0; j<table[i].size(); j++)
            {
                fileout<<"<"<<table[i][j].getSymbol()<<","
                                        <<table[i][j].getSymbolType()<<"> ";
            }
            fileout<<endl;
        }
    }

    int hashFunc(string symbol)
    {
        /*
        int a = ((symbol[0]+symbol[1]+symbol[2])*2)% 12;
        return a;
        */
        int sum = 0;
        if(symbol.length()>=1)sum+=symbol[0];
        if(symbol.length()>=2)sum+=symbol[1];
        if(symbol.length()>=3)sum+=symbol[2];
        return ((sum*2)%12);
    }
};
```

2.7.12 Implement the main function

```
int main()
{
    string symbol, symbolType;
    SymbolTable ob;

    ifstream input;
    ofstream key("output1_keyword.txt");
    ofstream func("output1_function.txt");
    ofstream identifier("output1_id.txt");
    ofstream operat("output1_oper.txt");
    ofstream num("output1_number.txt");
    ofstream liter("output1_literal.txt");

    input.open("sample_input1.txt");

    string s="";
    int line=1;
    while(getline(input,s))
    {
```

```
int n=s.size();
string token="", temp="", lit="";
bool literal =false, op_pattern=false, str_pattern=false;

for (int i=0; i<n; i++)
{
   temp="";
   if(s[i]=='"')
   {
      if(literal)
      {
         lit+=s[i];
         liter<<lit<<" "<<line<<endl;
         literal=false;
      }
      else
      {
         literal=true;
         ///continue;
      }
   }

   if(literal)
   {
      lit+=s[i];
   }
   if(idOrKey(s[i]) && !literal)
   {
      token+=s[i];
   }
   else
   {
      if(!interrupt(s[i]) && !literal && !space(s[i]))
      {
         temp=s[i];
         if(op(temp))
         {
            string y=temp+s[i+1];
            if(op(y))
            {
               operat<<y<<" "<<line<<endl;
               i++;
               ob.insertVal(y,"Operator");
            }
            else
            {
               operat<<temp<<" "<<line<<endl;
               ob.insertVal(temp,"Operator");
            }
            op_pattern=true;
         }
         else op_pattern=false;
      }
      if(keyword(token))
      {
```

```
               key<<token<<" "<<line<<endl;
               ob.insertVal(token,"Keyword");
               str_pattern=true;
            }
            else if(id(token))
            {

                  if(s[i]=='(')
                  {
                     func<<token<<" "<<line<<endl;
                     ob.insertVal(token,"Function");
                  }

                  else
                  {
                     identifier<<token<<" "<<line<<endl;
                     ob.insertVal(token,"Identifier");
                  }

               str_pattern=true;
            }
            else if(digit(token))
            {
               num<<token<<" "<<line<<endl;
               ob.insertVal(token,"Number");
               str_pattern=true;
            }
            else
            {
               str_pattern=false;
            }
            if(((token.size() && str_pattern==false) || (temp.size() && op_pattern==false)) && !literal || interrupt(s[i]))
            {
               cout<<"Lexical error at line "<<line<<" and error is: "<<s[i]<<endl;
            }
            token="";
         }
         str_pattern=op_pattern=false;
      }
      line++;
   }

   input.close();

   return 0;
}
```

2.8 RESULTS:

Run all the codes at a time in a single block and provide input as follows. Please test the program based on the sample input-output format as given.

TASKS/EVALUATION:

1. Show your code to the instructor.
2. Explain your understanding.
3. Answer the questions asked.

REFERENCES:

NA

REPORT ON
LAB EXPERIMENT 2

EXPERIMENT NO.: 02
TITLE: TOKENIZATION

DATE:
Batch & Section:

Student ID: ________________________ Student Name: ___________________________________

Objectives of the experiment:

Some Outputs/Graphs with appropriate title and explanation:

Comments based on your overall understanding:

To fill by the instructor:

Marks from the instructor: □ X □ Y □ Z

Instructor's Name and Signature: ____