Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

T.Y.B.Tech (CSE)

System Software and Compilers(SSC)

Lab Assignment No – 5

Name: Aniruddha Shende

Roll number: PD-05

Batch: D1

Panel: D

Name:- Aniruddha Arun Shende
Roll no:- PD-05
Batch:-D1
Panel:- D

Subject:- SSC

Lab Assignment No-5

Title:- Generate lexical analyzer for Java language using LEX tool.

Aim:- Write a program using LEX specification to implement lexical analysis phase of compiler to generate tokens of a subset of JAVA program

Objective:

① To understand the lexical analysis phase of the compiler.

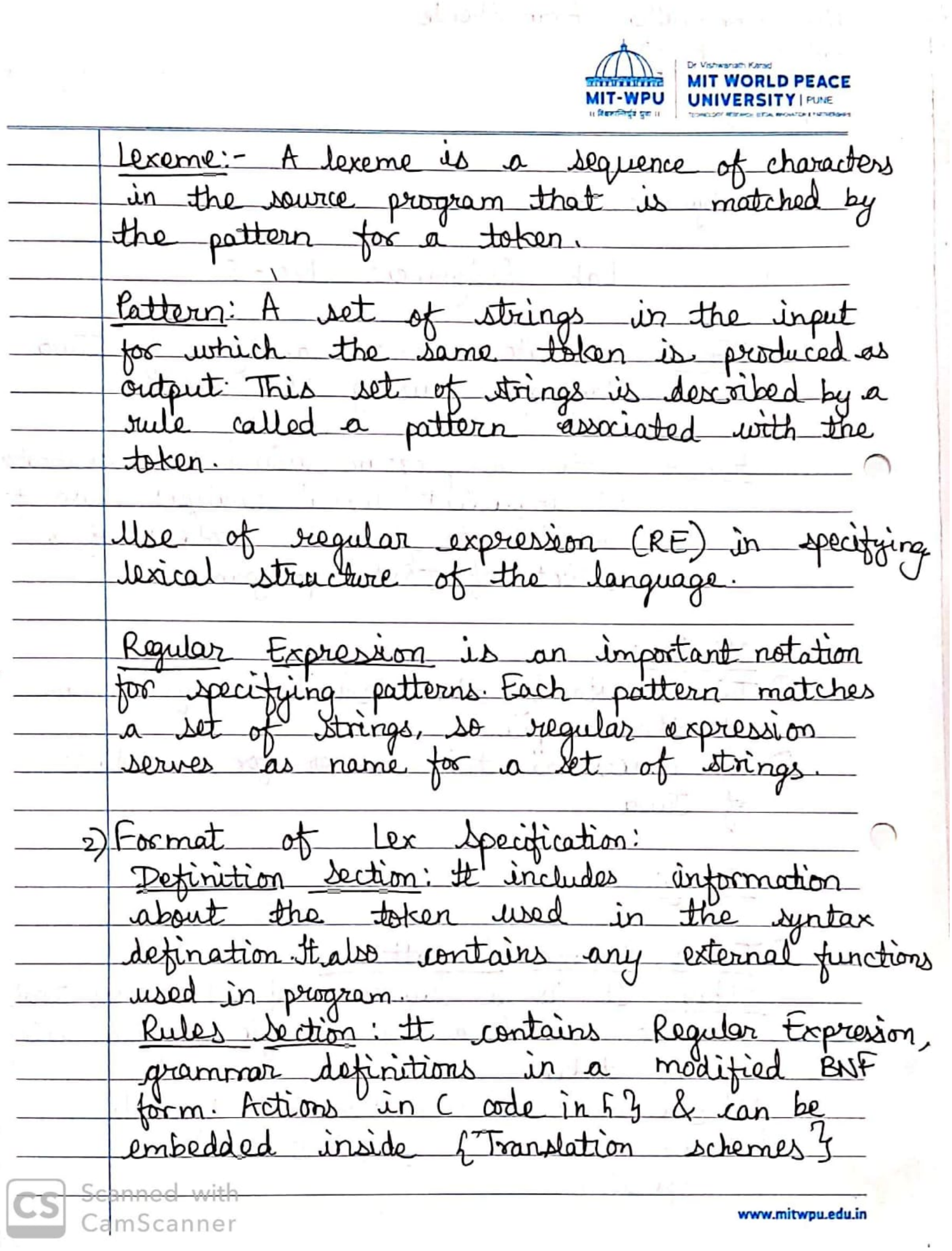
② To understand the scanner for a subset of Java.

Theory:

1) Token, lexeme & pattern:

Token:- It is a sequence of characters that can be treated as a single logical entry. Typical tokens are:

(i) Identifiers
(ii) Keywords
(iii) Operators
(iv) Special symbols
(v) Constants.



Lexeme:- A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

Pattern: A set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token.

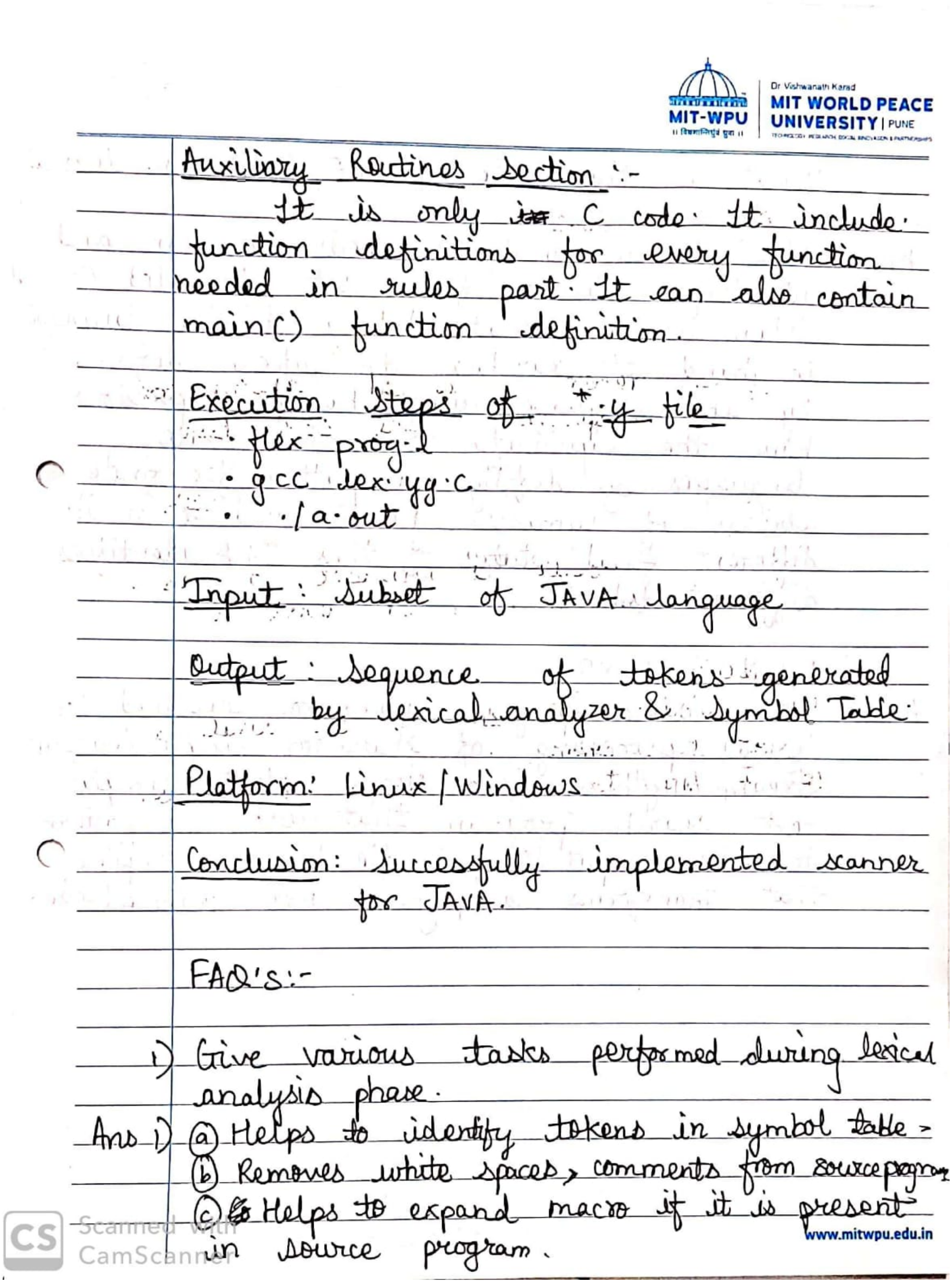
Use of regular expression (RE) in specifying lexical structure of the language.

Regular Expression is an important notation for specifying patterns. Each pattern matches a set of strings, so regular expression serves as name for a set of strings.

2) Format of Lex Specification:

Definition section: It includes information about the token used in the syntax defination. It also contains any external functions used in program.

Rules Section: It contains Regular Expression, grammar definitions in a modified BNF form. Actions in C code in {} & can be embedded inside {Translation schemes}



Auxiliary Routines section :-

It is only ~~in~~ C code. It include function definitions for every function needed in rules part. It can also contain main() function definition.

Execution Steps of *.y file

- flex prog.l
- gcc lex.yy.c
- ./a.out

Input : Subset of JAVA language

Output : Sequence of tokens generated by lexical analyzer & Symbol Table.

Platform: Linux / Windows

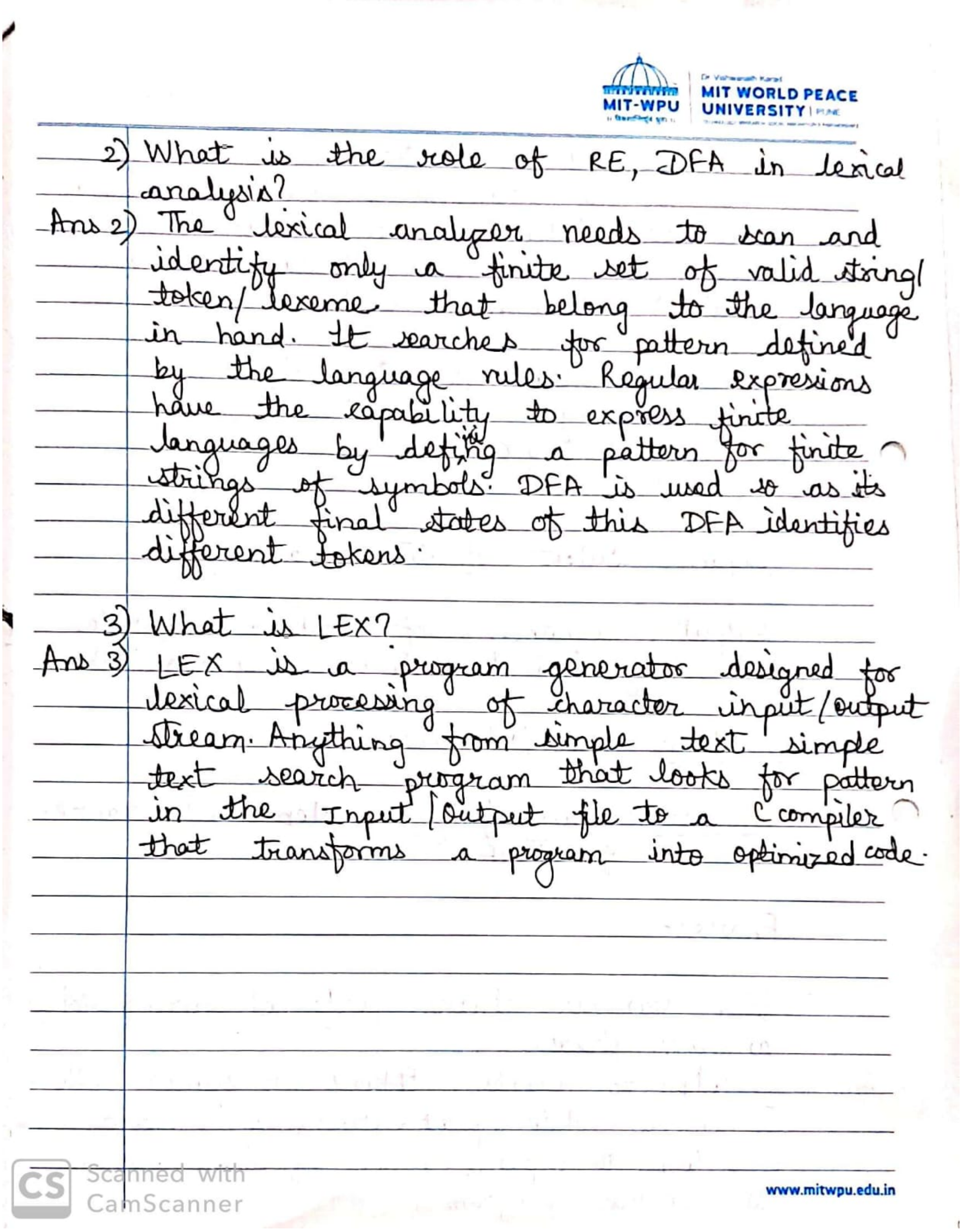
Conclusion: Successfully implemented scanner for JAVA.

FAQ's:-

1) Give various tasks performed during lexical analysis phase.

Ans 1) (a) Helps to identify tokens in symbol table.
(b) Removes white spaces, comments from source program.
(c) Helps to expand macro if it is present in source program.

www.mitwpu.edu.in



2) What is the role of RE, DFA in lexical analysis?

Ans 2) The lexical analyzer needs to scan and identify only a finite set of valid string/ token/ lexeme that belong to the language in hand. It searches for pattern defined by the language rules. Regular expressions have the capability to express finite languages by defining a pattern for finite strings of symbols. DFA is used so as its different final states of this DFA identifies different tokens.

3) What is LEX?

Ans 3) LEX is a program generator designed for lexical processing of character input/output stream. Anything from simple text simple text search program that looks for pattern in the Input/Output file to a C compiler that transforms a program into optimized code.

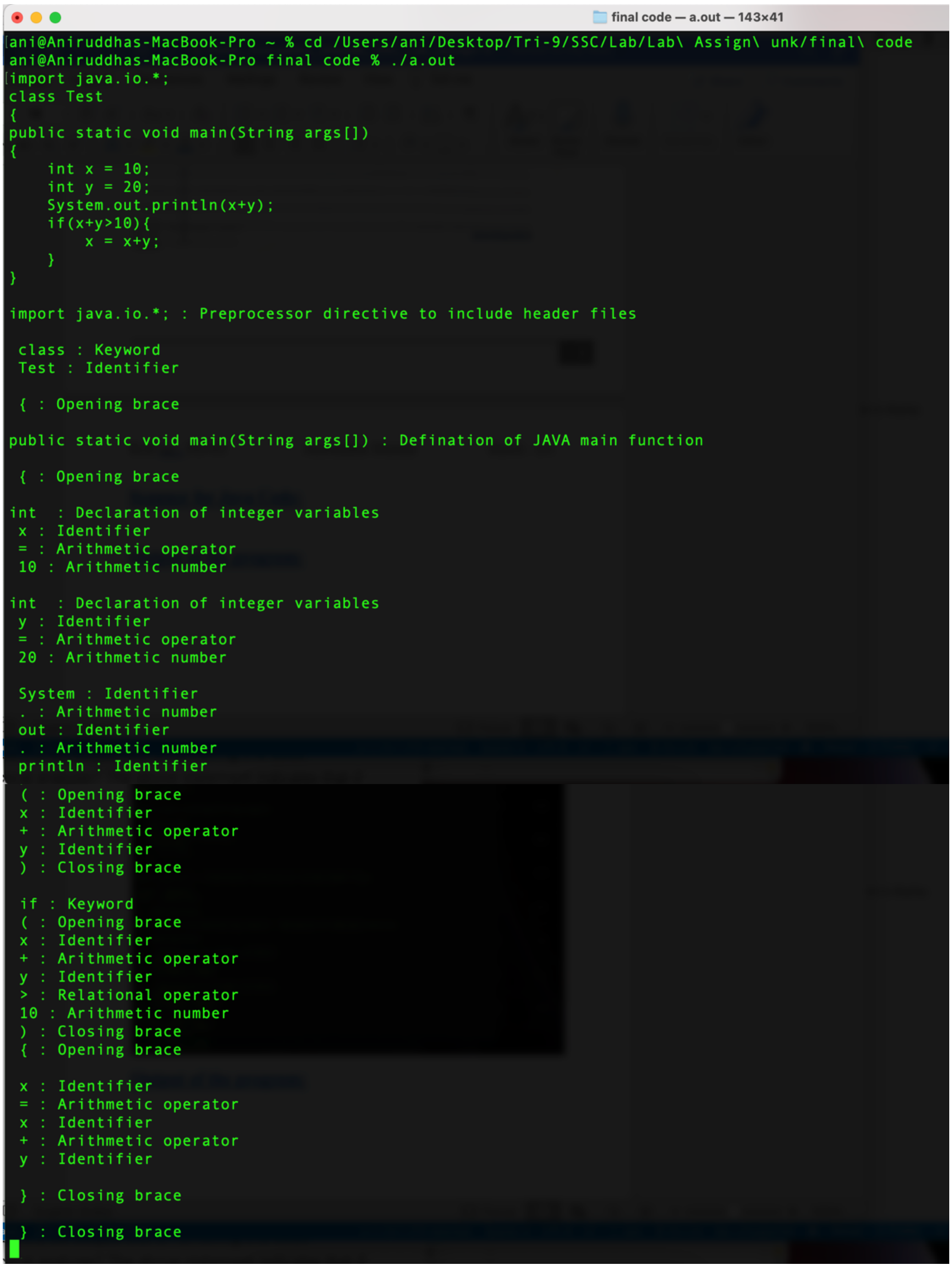
www.mitwpu.edu.in

Scanner for Java Code:

```
%{
#include<string.h>
   int i,m;
   struct symtab
   {
     char name[200];
     char type[200];
   }sym[20];
            FILE* yyin;
            int checksymtab(char *temp);
%}


ws       [\t ]
digit    [0-9]
alpha    [a-zA-Z]
id       {alpha}({alpha}|{digit})*
aspec "public"|"private"|"protected"
stat "static"
sc ";"


%%

import{ws}{alpha}+(.{alpha}+)*(.*?){sc}    {printf("\n%s : Preprocessor directive to include
header files",yytext);}

if |
case |
else |
while |
do |
switch{ws}*\(.*\) |
for |
static |
class    {printf("\n %s : Keyword",yytext);}

{aspec}  {printf("\n %s : Access Specifier", yytext);}


System\.out\.println{ws}*\(\".*\"\){sc} |
System\.out\.print{ws}*\(\".*\"\){sc}  {printf("\n %s : Library Function",yytext);}

\/\/{ws}*{alpha}*({ws}*{alpha}*)*  {printf("\n %s : This is single line comment", yytext);}
```

```
public{ws}+static{ws}+void{ws}+main\(String{ws}+{id}"[]"\) {printf("\n%s : Defination
of JAVA main function",yytext);}

{aspec}{ws}+{stat}*{ws}+void{ws}+{id}{ws}*\(({ws}*{id}{ws}*,)*({ws}*{id}{ws}*)?\)
          {
                                                                        printf("\n%s : Definition
of function with return type void" ,yytext);
                                                                        if(checksymtab(yytext)
== 0) {

          strcpy(sym[m].name,yytext);

          strcpy(sym[m].type,"Function");
                                                                                m++;
                                                                        }
                                                                }
{aspec}{ws}+{stat}*{ws}+int{ws}+{id}{ws}*\(({ws}*{id}{ws}*,)*({ws}*{id}{ws}*)?\)
{printf("\n%s : Definition of function with return type int" ,yytext);}
{aspec}{ws}+{stat}*{ws}+float{ws}+{id}{ws}*\(({ws}*{id}{ws}*,)*({ws}*{id}{ws}*)?\)
          {printf("\n%s : Definition of function with return type char" ,yytext);}
{aspec}{ws}+{stat}*{ws}+char{ws}+{id}{ws}*\(({ws}*{id}{ws}*,)*({ws}*{id}{ws}*)?\)
          {printf("\n%s : Definition of function with return type float" ,yytext);}
{aspec}{ws}+{stat}*{ws}+double{ws}+{id}{ws}*\(({ws}*{id}{ws}*,)*({ws}*{id}{ws}*)
?\)       {printf("\n%s : Definition of function with return type double" ,yytext);}

{id}{ws}*\(({ws}*{id}{ws}*,)*({ws}*{id}{ws}*)?\)                {printf("\n%s : Function
call" ,yytext);}

int{ws}+        {printf("\n%s : Declaration of integer variables",yytext);}
char{ws}+       {printf("\n%s : Declaration of character variables",yytext);}
float{ws}+      {printf("\n%s : Declaration of float variables",yytext);}
double{ws}+ {printf("\n%s : Declaration of double variables",yytext);}
boolean{ws}+ {printf("\n%s : Declaration of boolean variables",yytext);}
String{ws}+     {printf("\n%s : Declaration of string variables",yytext);}
byte{ws}+ {printf("\n%s : Declaration of byte variables",yytext);}
short{ws}+ {printf("\n%s : Declaration of short variables",yytext);}

[;,]      {}
[{(]      {printf("\n %s : Opening brace",yytext);}
[})]      {printf("\n %s : Closing brace",yytext);}


[+\-\*\\%=]     {printf("\n %s : Arithmetic operator",yytext);}
[<>]=?                  {printf("\n %s : Relational operator",yytext);}

{id}      {
                printf("\n %s : Identifier",yytext);
                if(checksymtab(yytext) == 0) {
                        strcpy(sym[m].name,yytext);
                        strcpy(sym[m].type,"Identifier");
                        m++;
                }
        }

{digit}*(\.{digit}*)?   {
                                printf("\n %s : Arithmetic number",yytext);
                                if(checksymtab(yytext) == 0) {
                                        strcpy(sym[m].name,yytext);
                                        strcpy(sym[m].type,"Number");
                                        m++;
                                }
                        }

{id}"["{digit}*"]"      {
                                printf("\n %s : Array",yytext);
                                if(checksymtab(yytext) == 0) {
                                        strcpy(sym[m].name,yytext);
                                        strcpy(sym[m].type,"Array");
                                        m++;
                                }
                        }

%%

int checksymtab(char *temp) {
        for(int i = 0; i < 20; i++)
                if(strcmp(sym[i].name, temp) == 0) return 1;
        return 0;
}
int main(int argc,char* argv[])
{

/*      printf("Enter the String: ");
        yylex();*/

        yyin=fopen(argv[1],"r");
        yylex();
        fclose(yyin);


        printf("\n\n\n\t--------------------\n");
        printf("\tIndex\tSymbol name\t\t\tSymbol Type");
        printf("\t\t\n--------------------------\n");

        for(i=0;i<m;i++)
        {
           printf("\n\t%d",i+1);
                printf("\t\t%s",sym[i].name);
                printf("\t\t\t%s\n",sym[i].type);
        }
        yywrap();
}

extern int yywrap()
{
        return 1;
}
```

Output of the program:



```
ani@Aniruddhas-MacBook-Pro ~ % cd /Users/ani/Desktop/Tri-9/SSC/Lab/Lab\ Assign\ unk/final\ code
ani@Aniruddhas-MacBook-Pro final code % ./a.out
import java.io.*;
class Test
{
public static void main(String args[])
{
    int x = 10;
    int y = 20;
    System.out.println(x+y);
    if(x+y>10){
        x = x+y;
    }
}

import java.io.*; : Preprocessor directive to include header files

 class : Keyword
 Test : Identifier

 { : Opening brace

public static void main(String args[]) : Defination of JAVA main function

 { : Opening brace

int  : Declaration of integer variables
 x : Identifier
 = : Arithmetic operator
 10 : Arithmetic number

int  : Declaration of integer variables
 y : Identifier
 = : Arithmetic operator
 20 : Arithmetic number

 System : Identifier
 . : Arithmetic number
 out : Identifier
 . : Arithmetic number
 println : Identifier
 ( : Opening brace
 x : Identifier
 + : Arithmetic operator
 y : Identifier
 ) : Closing brace

 if : Keyword
 ( : Opening brace
 x : Identifier
 + : Arithmetic operator
 y : Identifier
 > : Relational operator
 10 : Arithmetic number
 ) : Closing brace
 { : Opening brace

 x : Identifier
 = : Arithmetic operator
 x : Identifier
 + : Arithmetic operator
 y : Identifier

 } : Closing brace

 } : Closing brace
```