CS6109 – COMPILER DESIGN
LAB – 2 ASSIGNMENT

NAME: HEMANTH N
REG NO: 2019503519
BATCH 1

1. Consider a C program. Using the Lexical analyzer, collect the tokens, classify the tokens as identifiers, constants, keywords and operators. Store each category into a separate binary search tree and display the tokens from the respective binary search tree along with their count.

LEX FILE:

```
%{

#include<stdio.h>
#include<malloc.h>
#include<string.h>

struct node
{
  char info[20];
  struct node *left,*right;
};

int identc=0,keyc=0,consc=0,oprc=0;

struct node *iden=NULL,*cons=NULL,*opr=NULL,*key=NULL;
void insert(struct node **root,char str[20])
{
   struct node *p;
   p = malloc(sizeof(struct node));
   strcpy(p->info,str);
   p->left = NULL;
   p->right = NULL;

   if(*root==NULL)
         *root=p;
   if(strcmp(str,(*root)->info)==0)
         return;
   if(strcmp(str,((*root)->info))>0)
        insert(&((*root)->right),str);
   else
        insert(&((*root)->left),str);
}
void display(struct node *root)
{
   if(root!=NULL)
   {
         display(root->left);
         printf("%s ",root->info);
         display(root->right);
         }
}
%}
```

```
%option noyywrap

%%

#.* {}
["|,|;(|)|{|}|.|_] {}
[[]] {}
"#"|"@"|"$"|"^"|"%"|"^"|"&" {}
"["|"]" {}
"="|"+"|"-"|"*"|"/"|"&&"|"||"|"!"|"<="|">="|"++"|"!="|"=="|"<"|">"  {oprc++;insert(&opr,yytext);}
"auto"|"break"|"case"|"char"|"const"|"continue"|"default"|"do"|"double"|"else"|"enum"|"extern"|"float
"|"for"|"goto"|"if"|"int"|"long"|"register"|"return"|"short"|"signed"|"sizeof"|"static"|"struct"|"switch"|"
typedef"|"union"|"unsigned"|"void"|"volatile"|"while" {keyc++;insert(&key,yytext);}
[a-zA-Z_][a-zA-Z0-9_]*\( {}
[0-9]+|"-"[0-9]+|[0-9]."[0-9]+|[-][0-9]."[0-9]+|["]([^"\\\n]|\\.|\\\n)*["]|"NULL"
{consc++;insert(&cons,yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {identc++;insert(&iden,yytext);}

"//"." " {}
printf\([A-Za-z0-9%" "\"]* {}
scanf\([A-Za-z0-9%\"]* {}
["\t\n]+
\/\*([^*]|[\r\n]|(\*+([^*/]|[\r\n])))*\*+\/ {}

%%
int main(int argc, char* argv[])
{
  yyin = fopen(argv[1], "r");
  yylex();

  printf("\nDistinct Keywords : \n");
  display(key);
  printf("\ntotal Count : %d\n",keyc);

  printf("Distinct  Operators : \n");
  display(opr);
  printf("\ntotal Count : %d\n",oprc);

  printf("Distinct  Constants : \n");
  display(cons);
  printf("\nTotal Count : %d\n",consc);

  printf("Distinct  Identifiers : \n");
  display(iden);
  printf("\nTotal Count : %d\n",identc);

  return 0;
}
```

INPUT C FILE:

```
#include<stdio.h>
int main() {
double first, second, temp;
printf("first number ");
scanf("%lf", &first);
printf("second number ");
scanf("%lf", &second);
temp = first;
first = second;
second = temp;
printf("Swap1 %.2lf", first);
printf("swap2 %.2lf", second);
return 0;
}
```

```
#include<stdio.h>
int main()
{
float first, second, t;
printf("num1");
scanf("%f",&first);
printf("num2");
scanf("%f",&second);
t = first;
first = second;
second = temp;
printf("Swap1 %.2lf", first);
printf("swap2 %.2lf", second);
return 0;
}
```

OUTPUT:

```
hemsmit@hemsmit-2019503519:~/Downloads$ flex cbst.l
hemsmit@hemsmit-2019503519:~/Downloads$ gcc lex.yy.c
hemsmit@hemsmit-2019503519:~/Downloads$ ./a.out ctest.c

Distinct Keywords :
double float int return
total Count : 6
Distinct  Operators :
=
total Count : 6
Distinct  Constants :
"%f" "%lf" "Swap1 %.2lf" "first number " "num1" "num2" "secon
Total Count : 14
Distinct  Identifiers :
first second t temp
Total Count : 26
```

2. Create a lexical analyzer to identify Student register number, Aadhar number, PAN, Mobile Number, Date in dd/mm/yyyy, Time in HH:MM:SS:AM/PM, IP Address, MAC Address, email address, a vehicle registration number (Ex TN07AB1234) from the given text file. Store all the valid information in separate arrays and display. Ignore the contents which are not matching in any of the above said categories.

LEX FILE:

```
%{
   #include<stdio.h>
   #include<string.h>
   char *phno[10], *time[10], *aadhar[10], *pan[10], *mac[10], *email[10], *regno[10],
*ipadd[10], *day[10], *veh[10];
   int ph,t,ad,p, m, em,reg, ip ,d,v;
%}

%%
[ \n\t]
([1-9][0-9]{3})[5][0-9]{2}[0 5][0-9]{2} {regno[reg++]=strdup(yytext);}
(([0-9]{4})" "([0-9]{4})" "([0-9]{4})) {aadhar[ad++]=strdup(yytext);}
[6-9][0-9]{9} {phno[ph++]=strdup(yytext);}
([A-Z]{5})([0-9]{4}[A-Z]) {pan[p++]=strdup(yytext);}
([0-2][0-9]|[3][0-1])\/((0(1|3|5|7|8))|(10|12))\/([1-2][0-9][0-9][-0-9])|([0-2][0-
9]|30)\/((0(4|6|9))|11)\/([1-2][0-9][0-9][0-9])|([0-1][0-9]|2[0-8])\/02\/([1-2][0-9][0-9][0-9])
{day[d++]=strdup(yytext);}
([0-1][0-9]):([0-5][0-9]):([0-5][0-9])(" AM"|" PM")  {time[t++]=strdup(yytext);}
```

```
[a-z.0-9_]+@[a-z]+".com"|".in" {email[em++]=strdup(yytext);}
TN" "[0-9][0-9]" "([A-Z ]|"")[A-Z]" "([0-9]{4}) {veh[v++]=strdup(yytext);}
([0-9][a-f]|[a-f][0-9]|[0-9][0-9]|[a-f][a-f]):([0-9][a-f]|[a-f][0-9]|[0-9][0-9]|[a-f][a-f]):([0-9][a-f]|[a-
f][0-9]|[0-9][0-9]|[a-f][a-f]):([0-9][a-f]|[a-f][0-9]|[0-9][0-9]|[a-f][a-f]):([0-9][a-f]|[a-f][0-9]|[0-9][0-
9]|[a-f][a-f]):([0-9][a-f]|[a-f][0-9]|[0-9][0-9]|[a-f][a-f]) {mac[m++]=strdup(yytext);}

([0-9]|[1-9][0-9]|[1][0-9][0-9]|[2][0-5][0-5])"."([0-9]|[1-9][0-9]|[1][0-9][0-9]|[2][0-5][0-5])"."([0-
9]|[1-9][0-9]|[1][0-9][0-9]|[2][0-5][0-5])"."([0-9]|[1-9][0-9]|[1][0-9][0-9]|[2][0-5][0-5])
{ipadd[ip++]=strdup(yytext);}
"//".*" "*|" " {}
[\t\n]+
[a-zA-Z0-9]
%%

int yywrap() {
   return 1;
}

int main(int argc, char* argv[])
{
   yyin = fopen(argv[1], "r");
   yylex();


   printf("\nREG NO:\n");
   for(int i=0;i<reg;i++)
      printf("%d %s\n", i+1, regno[i]);

   printf("\nPHONE NUMBER:\n");
   for(int i=0;i<ph;i++)
      printf("%d %s\n", i+1, phno[i]);
   printf("\nDAY:\n");
   for(int i=0;i<d;i++)
      printf("%d %s\n", i+1,day[i]);
  printf("\nTIME:\n");
   for(int i=0;i<t;i++)
      printf("%d %s\n", i+1, time[i]);

   printf("\nVEHICLE REG NO:\n");
   for(int i=0;i<v;i++)
      printf("%d %s\n", i+1, veh[i]);

  printf("\nAADHAR:\n");
   for(int i=0;i<ad;i++)
      printf("%d %s\n", i+1, aadhar[i]);


  printf("\nPAN:\n");
   for(int i=0;i<p;i++)
      printf("%d %s\n", i+1, pan[i]);
  printf("\nMAC:\n");
   for(int i=0;i<m;i++)
```

```
        printf("%d %s\n", i+1, mac[i]);

    printf("\nEMAIl:\n");
     for(int i=0;i<em;i++)
        printf("%d %s\n", i+1, email[i]);

     printf("\nIP ADDRESS:\n");
     for(int i=0;i<ip;i++)
        printf("%d %s\n", i+1, ipadd[i]);


     return 0;
}
```

INPUT TEXT FILE:

2019503519 Register Num this is hemanth

1134 5541 5264 Aadhar thjois is aadhar 9080568795

AAAAA1234A Pan

9858021412 Phone

07/11/2021 Date

11:11:11 AM Time

07:00:01 AM Time

hems@gmail.com email

sharms@gmail.com email sample

TN 03 CN 2562 Vehicle

TN 07 HS 0701 Vehicle

2c:0a:73:b1:c1:7e Mac Address

0c:00:53:c0:b0:8c Mac Address

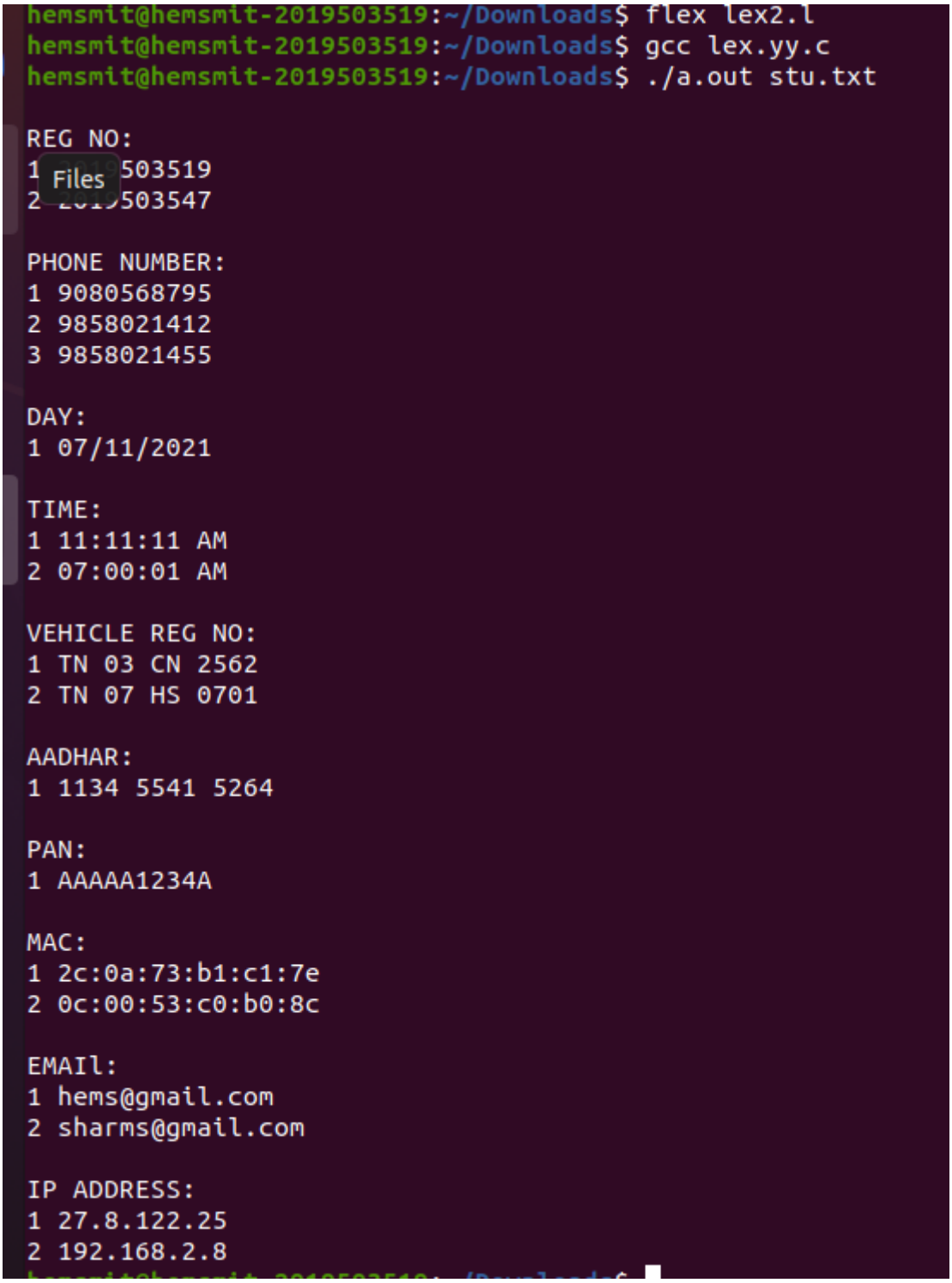
27.8.122.25 IP Address

192.168.2.8 ip sample

2019503547 Register No

9858021455 Phone

OUTPUT:


hemsmit@hemsmit-2019503519:~/Downloads$ flex lex2.l
hemsmit@hemsmit-2019503519:~/Downloads$ gcc lex.yy.c
hemsmit@hemsmit-2019503519:~/Downloads$ ./a.out stu.txt
REG NO:
1 503519
2 503547
PHONE NUMBER:
1 9080568795
2 9858021412
3 9858021455
DAY:
1 07/11/2021
TIME:
1 11:11:11 AM
2 07:00:01 AM
VEHICLE REG NO:
1 TN 03 CN 2562
2 TN 07 HS 0701
AADHAR:
1 1134 5541 5264
PAN:
1 AAAAA1234A
MAC:
1 2c:0a:73:b1:c1:7e
2 0c:00:53:c0:b0:8c
EMAIl:
1 hems@gmail.com
2 sharms@gmail.com
IP ADDRESS:
1 27.8.122.25
2 192.168.2.8

3. Construct a LEX file for recognizing tokens in a structured query language.

LEX FILE:

```
%{
   #include<stdio.h>
%}

%%
[ \n\t;']+
(?i:add)          {printf("%s : ADD \n", yytext);}
(?i:alter)        {printf("%s     : ALTER \n", yytext);}
(?i:all) {printf("%s : ALL \n", yytext);}
(?i:and)          {printf("%s   : AND \n", yytext);}
(?i:any)          {printf("%s     : ANY \n", yytext);}
(?i:as)  {printf("%s     :AS \n", yytext);}
(?i:asc) {printf("%s    : ASC \n", yytext);}
(?i:between)    {printf("%s     :BETWEEN \n", yytext);}
(?i:case)         {printf("%s     : CASE \n", yytext);}
(?i:check)        {printf("%s     :CHECK \n", yytext);}
(?i:column)     {printf("%s    :COLUMN \n", yytext);}
(?i:constraint) {printf("%s : CONSTRAINT \n", yytext);}
(?i:create)       {printf("%s     : CREATE \n", yytext);}
(?i:count)        {printf("%s     : COUNT \n", yytext);}
(?i:delete)       {printf("%s     : DELETE KEYWORD \n", yytext);}
(?i:database)   {printf("%s :     DATABASE \n", yytext);}
(?i:default)      {printf("%s     : DEFAULT \n", yytext);}
(?i:desc)         {printf("%s      :DESC \n", yytext);}
(?i:distinct)     {printf("%s     : DISTINCT \n", yytext);}
(?i:drop)         {printf("%s      :DROP \n", yytext);}
(?i:exec)         {printf("%s      :EXEC \n", yytext);}
(?i:exists)       {printf("%s     :EXISTS \n", yytext);}
(?i:forign)        {printf("%s    : FORIGN \n", yytext);}
(?i:from)         {printf("%s      :FROM \n", yytext);}
(?i:group" "by) {printf("%s    : GROUP BY \n", yytext);}
(?i:having)       {printf("%s     : HAVING \n", yytext);}
(?i:index)        {printf("%s      : INDEX \n", yytext);}
(?i:in)  {printf("%s      : IN \n", yytext);}
(?i:inner)        {printf("%s      : INNER \n", yytext);}
(?i:join)         {printf("%s     : JOIN \n", yytext);}
(?i:into)         {printf("%s     : INTO \n", yytext);}
(?i:is)  {printf("%s     :IS \n", yytext);}
(?i:right)        {printf("%s      : RIGHT \n", yytext);}
(?i:left)         {printf("%s     : LEFT \n", yytext);}
(?i:like)         {printf("%s     : LIKE \n", yytext);}
(?i:not)          {printf("%s     : NOT \n", yytext);}
(?i:order)        {printf("%s     :ORDER \n", yytext);}
(?i:primary)    {printf("%s     : PRIMARY \n", yytext);}
(?i:key)           {printf("%s     : KEY \n", yytext);}
```

```
(?i:procedure) {printf("%s     : PROCEDURE \n", yytext);}
(?i:select)       {printf("%s     : SELECT \n", yytext);}
(?i:where)       {printf("%s      : WHERE \n", yytext);}
(?i:unique)      {printf("%s     :UNIQUE \n", yytext);}
(?i:update)      {printf("%s     : UPDATE \n", yytext);}
(?i:view)        {printf("%s      :VIEW \n", yytext);}
(?i:values)       {printf("%s    : VALUES \n", yytext);}

(char\([0-9]+\)|char|varchar|varchar\([0-9]+\)|bool|int|bigint|smallint|float) {printf("%s     ==>
DATATYPE \n", yytext);}
"*"                {printf("%s     :SELECT ALL \n", yytext);}
[\(\)]    {printf("%s      : BRACKETS \n", yytext);}
[0-9]+  {printf("%s      : NUMBER \n", yytext);}
("<"|">"|"<="|">="|"<>"|"!=") {printf("%s    : RELATIONAL OPERATOR \n", yytext);}
"=" {printf("%s          : EQUALITY OPERATOR \n", yytext);}
[A-Za-z][A-Za-z0-9_]*         {printf("%s     : IDENTIFER \n", yytext);}
%%

int main(int argc, char** argv)
{
   yyin = fopen(argv[1], "r");
   yylex();

   return 0;
}

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

INPUT TEXT FILE:

SELECT * FROM STUDENTS;

DELETE FROM Customers;

DROP table STUDENTS;

SELECT SupplierName FROM Suppliers WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

SELECT COUNT(CustomerID), Country FROM Customers m GROUP BY Country ORDER BY COUNT(CustomerID) DESC;

OUTPUT:

```
hemsmit@hemsmit-2019503519:~/Downloads$ gcc lex.yy.c
hemsmit@hemsmit-2019503519:~/Downloads$ ./a.out sql.txt
SELECT  : SELECT
*       :SELECT ALL
FROM    :FROM
STUDENTS        : IDENTIFER
DELETE  : DELETE KEYWORD
FROM    :FROM
Customers       : IDENTIFER
DROP    :DROP
table   : IDENTIFER
STUDENTS        : IDENTIFER
SELECT  : SELECT
SupplierName    : IDENTIFER
FROM    :FROM
Suppliers       : IDENTIFER
WHERE   : WHERE
EXISTS  :EXISTS
(       : BRACKETS
SELECT  : SELECT
ProductName     : IDENTIFER
FROM    :FROM
Products        : IDENTIFER
WHERE   : WHERE
Products        : IDENTIFER
.SupplierID     : IDENTIFER
=       : EQUALITY OPERATOR
Suppliers       : IDENTIFER
.supplierID     : IDENTIFER
AND   : AND
Price   : IDENTIFER
=       : EQUALITY OPERATOR
22       : NUMBER
)       : BRACKETS
SELECT  : SELECT
COUNT   : COUNT
(       : BRACKETS
CustomerID      : IDENTIFER
)       : BRACKETS
,Country        : IDENTIFER
FROM    :FROM
Customers       : IDENTIFER
n       : IDENTIFER
GROUP BY        : GROUP BY
```

```
hemsmit@hemsmit-2019503519:~/Downloads$ flex sql.l
hemsmit@hemsmit-2019503519:~/Downloads$ gcc lex.yy.c
hemsmit@hemsmit-2019503519:~/Downloads$ ./a.out sql.txt
SELECT   : SELECT
*        :SELECT ALL
FROM     :FROM
STUDENTS         : IDENTIFER
DELETE   : DELETE KEYWORD
FROM     :FROM
Customers        : IDENTIFER
DROP     :DROP
table    : IDENTIFER
STUDENTS         : IDENTIFER
FOREIGN : FORIGN
WHERE    : WHERE
SELECT   : SELECT
SupplierName     : IDENTIFER
FROM     :FROM
Suppliers        : IDENTIFER
WHERE    : WHERE
EXISTS   :EXISTS
(        : BRACKETS
SELECT   : SELECT
ProductName      : IDENTIFER
FROM     :FROM
Products         : IDENTIFER
WHERE    : WHERE
Products         : IDENTIFER
.SupplierID      : IDENTIFER
=        : EQUALITY OPERATOR
Suppliers        : IDENTIFER
.supplierID      : IDENTIFER
AND    : AND
Price    : IDENTIFER
=        : EQUALITY OPERATOR
22       : NUMBER
)        : BRACKETS
SELECT   : SELECT
COUNT    : COUNT
(        : BRACKETS
CustomerID       : IDENTIFER
)        : BRACKETS
```

4. Consider the Hall Ticket issued for an End Semester Examination for a student. Count the number of Hard Core courses, Mathematical Soft Cores courses, Professional Elective Courses registered by the student.

LEX FILE:

```
%{

   #include<stdio.h>
   int hc, psc, pe,msc;
%}

%%
CS61(0[1-9]|10) hc++;
CS60(0[1-9]|[1-2][0-9]|35) pe++;
CS63(0[1-9]|[1-2][0-9]|08) psc++;
MA6201|CS6201|EC6201|CS6202 msc++;
[-a-zA-Z0-9]|\n|\t|" " ;

%%

int main(int ac, char **av)
{
   FILE *fd;

   if(ac==2)
   {
      if(!(fd=fopen(av[1],"r")))
      {
         perror("Error: ");
         return (-1);
      }
      yyset_in(fd);
      yylex();
      printf("O/P BY HEMANTH N | 2019503519\nHARDCORE: %d \nPROFESSIONAL
ELECTIVE: %d \nPROFESSIONAL SOFTCORE: %d \nMATHS SOFT CORE : %d \n",
hc,pe,psc, msc);
      fclose(fd);
   }
   else
      printf("Usage: a.out filename\n");
   return (0);
}
int yywrap(){return(1);}
```

INPUT TEXT HALL TICKET FILE:

1 25-JUN-21 Fore Noon CS6104 DATA STRUCTURES AND ALGORITHMS
2 30-JUN-21 Fore Noon MA6351 PROBABILITY AND STATISTICS
3 12-JUL-21 Fore Noon CS6202 THEORY OF COMPUTATION
4 13-JUL-21 Fore Noon CS6111 COMPUTER NETWORKS
5 14-JUL-21 Fore Noon CS6301 MACHINE LERANING
6 15-JUL-21 Fore Noon CS6001 DATA MINING

// 2 Hardcore courses , 1 prof elec , 1 math softcore, 1 prof softcore, and maths doesn't come under any of the following, (it comes under science and humanities so it is discarded.

OUTPUT:

```
hemsmit@hemsmit-2019503519:~/Downloads$ flex lex.l
hemsmit@hemsmit-2019503519:~/Downloads$ gcc lex.yy.c
hemsmit@hemsmit-2019503519:~/Downloads$ ./a.out hall.txt
O/P BY HEMANTH N | 2019503519
HARDCORE: 2
PROFESSIONAL ELECTIVE: 1
PROFESSIONAL SOFTCORE: 1
MATHS SOFT CORE : 1
hemsmit@hemsmit-2019503519:~/Downloads$
```

5. Assume you are supposed to design a language for the preparation of the invoice for any super market. List down the patterns for the important fields in the invoice.

LEX FILE:

```
%{

%}

%option noyywrap

%%
([A-Z]{4})([0-9]{5}[A-Z])    { printf("Transaction Id"); }  //4letters+5numbers+1letter
[A-Za-Z]* {printf("Product / Company name "); }
```

```
[1-9][0-9]+[.]?[0-9]*% { printf("Total GST or Tax Percentage"); }
[a-zA-Z0-9\/],* { printf("Address"): }
[+][0-9]{2}-[0-9]{10} {printf("Phonenumber"); }
[$][1-9][0-9]+[.]?[0-9]*  {printf("Total Amount"); }
([0-2][0-9]|[3][0-1])\/((0(1|3|5|7|8))|(10|12))\/([1-2][0-9][0-9][-0-9])|([0-2][0-
9]|30)\/((0(4|6|9))|11)\/([1-2][0-9][0-9][0-9])|([0-1][0-9]|2[0-8])\/02\/([1-2][0-9][0-9][0-9])
{printf("Issue Date"); }
([0-1][0-9]):([0-5][0-9]):([0-5][0-9])(" AM"|" PM") {printf("Issue Time"); }
[#][0-9]* { printf("Invoice number"); } // Numbers followed by a hash }
[a-z.0-9_]+@[a-z]+".com"|".in"   { printf("Contact email for queries"); }

%%

int main(int argc, char* argv[])
{
    yyin = fopen(argv[1], "r");
    yylex();

    return 0;
}
```