

## Model Practical Examination

Reg No: 39110636

Name: Mohnish Devraj

Branch: CSE

Semester: IV Sem

Subject Code: SCSA2602

Subject Name: Compiler Design Lab

Date of Exam: 19-04-2022

Batch-id: CD-7

Duration: 3 Hours

No. of pages: 9

① Implement a Desktop Calculator Using LEX and YACC tool.

Grammar:

$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T * F \mid T / F \mid F$

$E \rightarrow (E) \mid id$

Aim:

To write a program using LEX and YACC to implement Desktop Calculator

Algorithm:

Lex:

step 1: Start

step 2: Include the necessary header files and declare the necessary variables

step 3: Define the keywords and the identifiers with the constant and operator

step 4: Get the input for analysis from user

- step 5: check each and every element in the statement with number
- step 6: check each and every element in the statement with small alphabet
- step 7: check each and every element for the operator
- step 8: Else print Invalid token
- step 9: return the value
- step 10: Stop

### YACC:

- step 1: Start
- step 2: Include the necessary header files and declare the necessary variables
- step 3: Define the keywords and the identifiers with the constant and operator.
- step 4: Take the value which was taken from user and implement the respective operator.
- step 5: return the value and print it
- step 6: Stop

### Program

### Lex:

```
% {
```

```
#include <stdlib.h>
```

```
#include "y.tab.h"
```

```
void yyerror(char *s);
```

```
extern int yylval;
```

```
% }
```

```
% .
```

```
[0-9]+ {yylval = atoi(yytext); return INT;}
```

```
[a-z]+ {yylval = toascii(*yytext) - 97; return ID;}
```

```
[A-Z]+ {yylval = toascii(*yytext) - 65; return ID;}
```

```
[-+*/\n] {return *yytext;}
```

```
\( {return *yytext;}
```

```
\) {return *yytext;}
```

```
[\t] ;
```

```
. {yyerror("Invalid Token!!");}
```

```
% .
```

```
int yywrap()
```

```
{
```

```
return 1;
```

```
}
```

```
%acc:
```

```
% {
```

```
#include <stdio.h>
```

```
extern int yylex(void);
```

```
void yyerror(char *s);
```



```
int x=0;
```

```
int val = [26];
```

```
%}
```

```
% token INT ID
```

```
% %
```

```
mohrish:
```

```
mohrish expr 'ln'.
```

```
{x=$2; printf ("%d\n", $2);}
```

```
|mohrish ID '=' expr 'ln'
```

```
{val[$2]=$4;}
```

```
|mohrish DE
```

```
;
```

```
expr:
```

```
expr '+' T
```

```
{ $$ = $1 + $3;}
```

```
|expr '-' T
```

```
{ $$ = $1 - $3;}
```

```
|T
```

```
{ $$ = $1;}
```

```
| '+' T
```

```
{ $$ = 2 + $2;}
```

```
| '-' T
```

```
{ $$ = x - $2;}
```

```
;
```

```
T:
```

```
F
```

```
{ $$ = $1;}
```

```
|T '*' F
```

```
{ $$ = $1 * $3;}
```

```
|T '/' F
```

```
{ $$ = $1 / $3;}
```

```
| '*' F
```

```
{ $$ = 2 * $2;}
```

```
| '/' F
```

```
{ $$ = 2 / $2;}
```

```
;
```

```
F:
```

INT

{ \$\$ = \$1; }

| ID

{ \$\$ = val [\$1]; }

| '(' expr ')'

{ \$\$ = \$2; }

;

%. %.

void yyerror(char \* s)

{

printf("x.s", s);

}

int main()

{

yy parse();

return 0;

}

Output:

The output is attached below

Result:

The above program is executed and the output is verified.

② Write a C program to generate intermediate code in three address code format for the given input string.

Input: ~~a~~ a := b + c - d \* e / f

Output: z := e / f   a := b + c - d \* z   y := d \* z   a := b + c - y   x := b + c   a := x - y   w := x - y

a := w   a := w

Algorithm:

step 1: start

step 2: Accept the choice from the user (1. assignment 2. arithmetic  
3. relational 4. Exit)

step 3: if choice = 1

step 3.1: Find the string length

step 3.2: from the end of the string, till = symbol, copy the  
expression and store it in a temp variable

step 3.3: the LHS of the expression is stored in the first

step 4: if choice = 2

step 4.1: check the operator for precedence

step 4.2: Evaluate the expression based on the precedence

step 5: if choice = 3

step 5.1: Check the operator for precedence

step 5.2: Repeat the code with appropriate statement

step 6: if choice = 4

step 6.1: Exit

step 7: Stop

Program:

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <stdlib.h>
```



```

int i = 1, j = 0, no = 0, tmpch = 90;
char str[100], left[15], right[15];

void findopr();
void explore();
void fleft(int);
void fright(int);

struct exp
{
    int pos;
    int op;
} k[15];

int main()
{
    scanf("%s", str);
    findopr();
    explore();
    return 0;
}

void findopr()
{
    for(i=0; str[i] != '\0'; i++)
        if(str[i] == '=')
            k[j].pos = i;
            k[j++].op = '=';

    for(i=0; str[i] != '\0'; i++)
        if(str[i] == '(')
            k[j].pos = i;
            k[j++].op = '(';
}

```

```

for(i=0; str[i] != '\0'; i++)
    if(str[i] == '*')
        k[j].pos = i;
        k[j++].op = '*';

for(i=0; str[i] != '\0'; i++)
    if(str[i] == '+')
        k[j].pos = i;
        k[j++].op = '+';

for(i=0; str[i] != '\0'; i++)
    if(str[i] == '-')
        k[j].pos = i;
        k[j++].op = '-';
}

void explore()
{
    i = 1;
    while(k[i].op != '\0')
    {
        fleft(k[i].pos);
        fright(k[i].pos);
        str[k[i].pos] = tmpch--;
        printf("\n c = %s %c %s", str[k[i].pos], left, k[i].op, right);
        for(j=0; j < strlen(str); j++)
            if(str[j] != '\0')
                printf("\n c", str[j]);
            i++;
    }
    fright(-1);
    if(no == 0)
    {
        fleft(strlen(str));
        printf("\n s = %s", right, left);
        exit(0);
    }
}

```

```

}
printf("x.s:=x.c", right, str[x--], pos);
}

```

```

void fleft(int x)
{

```

```

    int w=0, flag=0;
    x--;
    while(x!=-1 && str[x]!='+' && str[x]!='*'
    && str[x]!='=' && str[x]!='\0' && str[x]!='\n'
    && str[x]!='\r')

```

```

{
    if(str[x]!='\0' && flag==0)

```

```

    {
        left[w++] = str[x];

```

```

        left[w] = '\0';

```

```

        str[x] = 'k';

```

```

        flag = 1;

```

```

    }

```

```

    x--;

```

```

}

```

```

void fright(int x)
{

```

```

    int w=0, flag=0;

```

```

    x++;

```

```

    while(x!=-1 && str[x]!='*' && str[x]!='\0'
    && str[x]!='=' && str[x]!='\n' && str[x]!='\r'
    && str[x]!='/')

```

```

{
    if(str[x]!='\0' && flag==0)

```

```

    {
        right[w++] = str[x];

```

```

        right[w] = '\0';

```

```

    str[x] = 'k';

```

```

    flag = 1;

```

```

}

```

```

    x++;

```

```

}

```

```

}

```

Output:

The output is attached below.

Result:

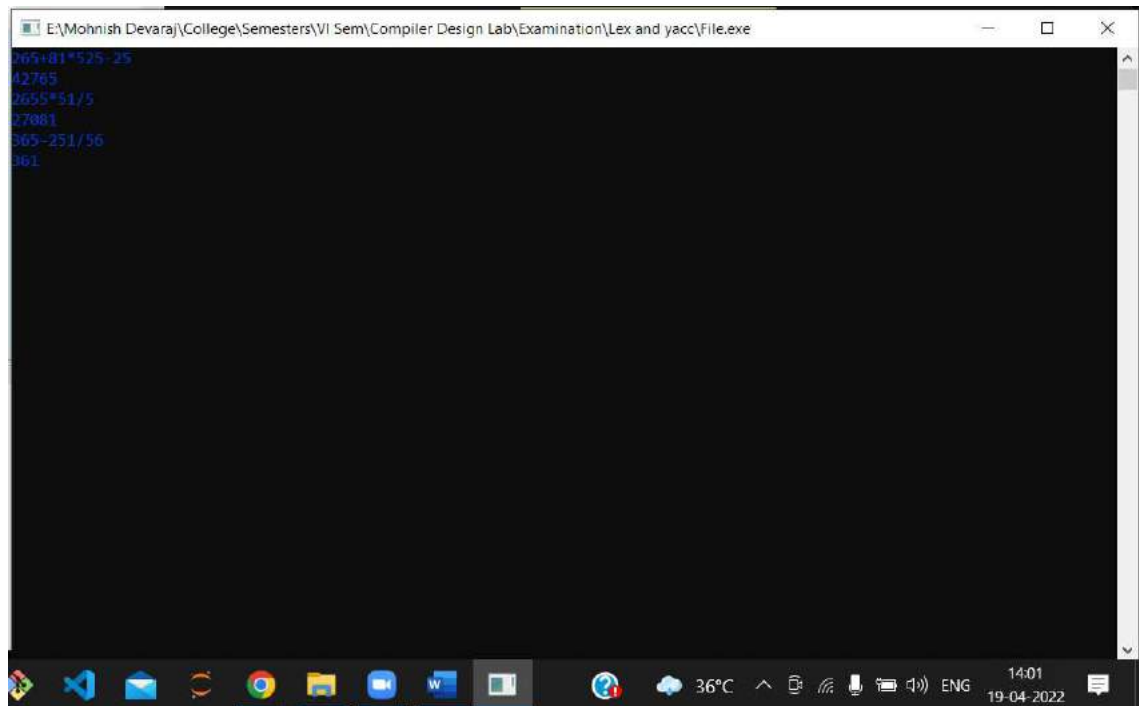
The above code is executed and successfully and the output is verified.



39110636

Output Screen shot:

1) Implement Desktop Calculator



2) Write a C program to generate intermediate code in three address code format for the given input string

	Test	Input	Expected	Got
✓	TestCase	a:=b+c-d*e/f	Z:=e/fa:=b+c-d*ZY:=d*Za:=b+c-YX:=b+ca:=X-YW:=X-Ya:=Wa:=W	Z:=e/f
Passed all tests! ✓				