Model Practical Examination

Reg No: 39110636	Name: Mohnish Devaraj
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:

step1: Start

step2: Include the necessary header files and declare the necessary variables

step3: Define the keywords and the identifiers with the constant and operator

step4: 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 token 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:
}
```

<u>Yacc:</u>

```
%{
#include <stdio.h>
extern int yylex(void);
void yyerror(char *);
int x=0;
int val [26];
%}
%token INT ID
%%
mohrish:
mohrish expr '\n'          {x=$2; printf ("%d\n", $2);}
|mohrish ID '=' expr '\n'  {val [$2] = $4;}
|~~mohrish IDe~~
;
expr:
expr '+' T                 {$$ = $1 + $3;}
|expr '-' T                {$$ = $1 - $3;}
|T                         {$$ = $1;}
|'+' T                     {$$ = x + $2;}
|'-' T                     {$$ = x - $2;}
;
T:
F                          {$$ = $1;}
|T '*' F                   {$$ = $1 * $3;}
|T '/' F                   {$$ = $1 / $3;}
|'*' F                     {$$ = x * $2;}
|'/' F                     {$$ = x / $2;}
;
F:
```

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

|ID                          {$$=val [$1]; }

|'(' expr')'                 {$$ =$2; }

;

%%

void yyerror(char* s)
 {
  printf ("%s", s);
  }

int main()
{
yyparse();
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:= b+c-d*e/f

Output: z:=e/f a: b+c-d*z Y:= d* za:=b+c-Yx:= b+ca :=x-YW: =X-Y

a:=Wa:=W

Algorithm:

```
step 1: start
step 2: Accept the choice from the user (1. assignment 2. arithematic
        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
                  expresion 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 expresion based on the Precedence
step 5: if choice = 3
        step 5.1 : Check the operator for precedence
        steps5.2 : 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("%c:=%s%c%s", str[k[i].pos], left,
  k[i].op, right);
 for(j=0;j<strlen(str);j++)
 if(str[j]!='$')
 printf("%c", str[j]);
 i++;
 }
 fright(-1);
 If (no==0)
 {
 fleft(strlen(str));
 printf("%s:=%s", right, left);
  exit(0);
}
printf("%s:=%c",right, str[k[--i].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]!='-'
&& str[x]!=',')
{
if(str[x]!='$'&& flag==0)
 {
 left[w++]=str[x];
 left[w]='\0';
 str[x]='$';
 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]!=':'&& str[x]!='-'
&& str[x]!='/')
{
if(str[x] != '$'&& flag= =0)
{
 right[w++] = str[x];
 right[w]='\0';
 str[x]='$';
 flag=1;
 }
 x++;
 }
}
```

Output:

The output is attached below.

Result:

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