Exercise 9

Expression to 3 address code (all operators)

.l

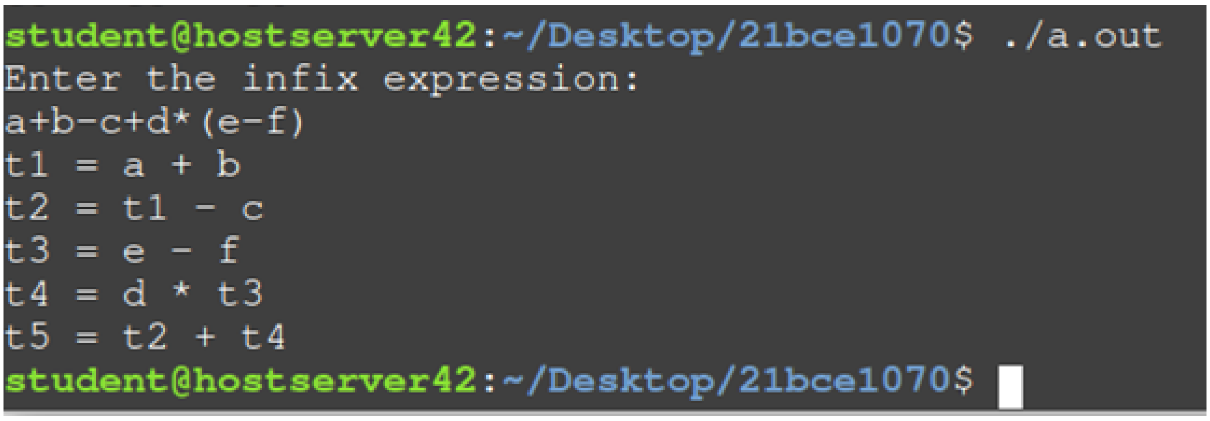
```
%{

#include<string.h>

#include "y.tab.h"

%}

%%

[a-z] {yylval.val=yytext;return NUM;}

"\n" {return NL;}

. {return yytext[0];}

%%

int yywrap(){}
```

.y

```
%{

#include<stdio.h>

#include<string.h>

char temp[3]="t1";

char st[10][10];

int top=-1;

int yylex(void);

int yyerror(char *s);

void codegen(char);

void push(char*);
%}

%union

{

char *val;

}

%token<val>NUM

%token NL

%type<val>E

%type<val>T

%type<val>F

%type<val>G

%type<val>H


%left '-'

%%

S: E NL{return 0;}

E: E '-' T {codegen('-');}
| E '+' T {codegen('+');}
| T
;

T: T '*' F {codegen('*');}
| T '/' F {codegen('/');}
| F
;

F: G '^' F {codegen('^');}
| G
;

G : '(' E ')' {$$ = $2;}
| H
;

H: NUM {push($1);}

;
```

```
}

%%
int main()
{
printf("Enter the infix expression:\n");
yyparse();
return 0;
}
int yyerror(char* s){
printf("\n Expression is invalid\n");
}

void push(char *ch)
{
top=top+1;
strcpy(st[top],ch);
}
void codegen(char a)
{
printf("%s = %s %c %s\n",temp,st[top-1],a,st[top]);
top-=1;
strcpy(st[top],temp);
temp[1]++;
}
```

Output



```
student@hostserver42:~/Desktop/21bce1070$ ./a.out
Enter the infix expression:
a+b-c+d*(e-f)
t1 = a + b
t2 = t1 - c
t3 = e - f
t4 = d * t3
t5 = t2 + t4
student@hostserver42:~/Desktop/21bce1070$
```

Expression to 3 address code (minus operator)

.l

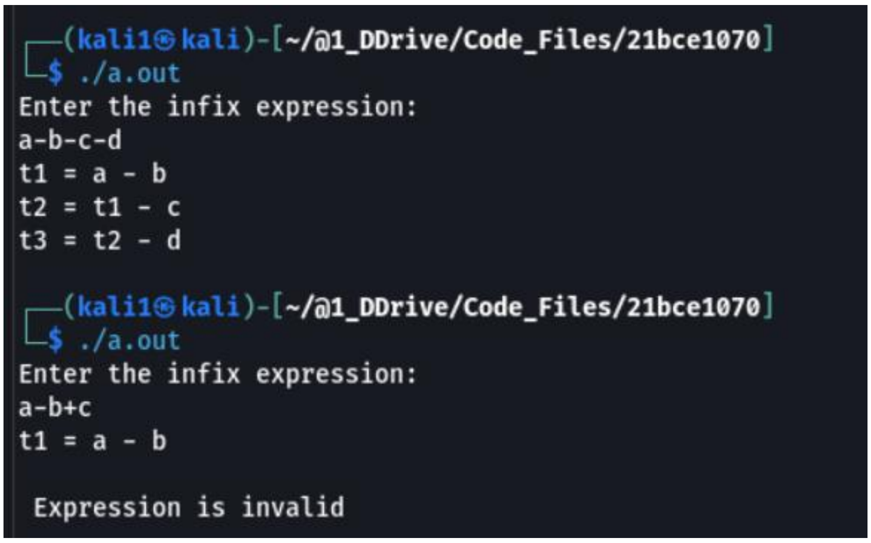
```
%{

#include<string.h>

#include "y.tab.h"

%}

%%

[a-z] {yylval.val=yytext;return NUM;}

"\n" {return NL;}

. {return yytext[0];}

%%

int yywrap(){}
```

.y

```
%{

#include<stdio.h>

#include<string.h>

char temp[3]="t1";
char st[10][10];

int top=-1;

int yylex(void);

int yyerror(char *s);

void codegen(char);

void push(char*);

%}

%union

{

char *val;

}

%token<val>NUM

%token NL

%type<val>E

%type<val>T

%left '-'

%%

S: E NL{return 0;}

E: E '-' T {codegen('-');}

| T

;

T: NUM {push($1);}

;


%%

int main()
```

```
{

printf("Enter the infix expression:\n");

yyparse();

return 0;

}

int yyerror(char* s){

printf("\n Expression is invalid\n");

}


void push(char *ch)

{

top=top+1;

strcpy(st[top],ch);

}

void codegen(char a)

{

printf("%s = %s %c %s\n",temp,st[top-1],a,st[top]);

top-=1;

strcpy(st[top],temp);

temp[1]++;

}
```

Output


```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ ./a.out
Enter the infix expression:
a-b-c-d
t1 = a - b
t2 = t1 - c
t3 = t2 - d

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ ./a.out
Enter the infix expression:
a-b+c
t1 = a - b

 Expression is invalid
```

Expression to Assembly

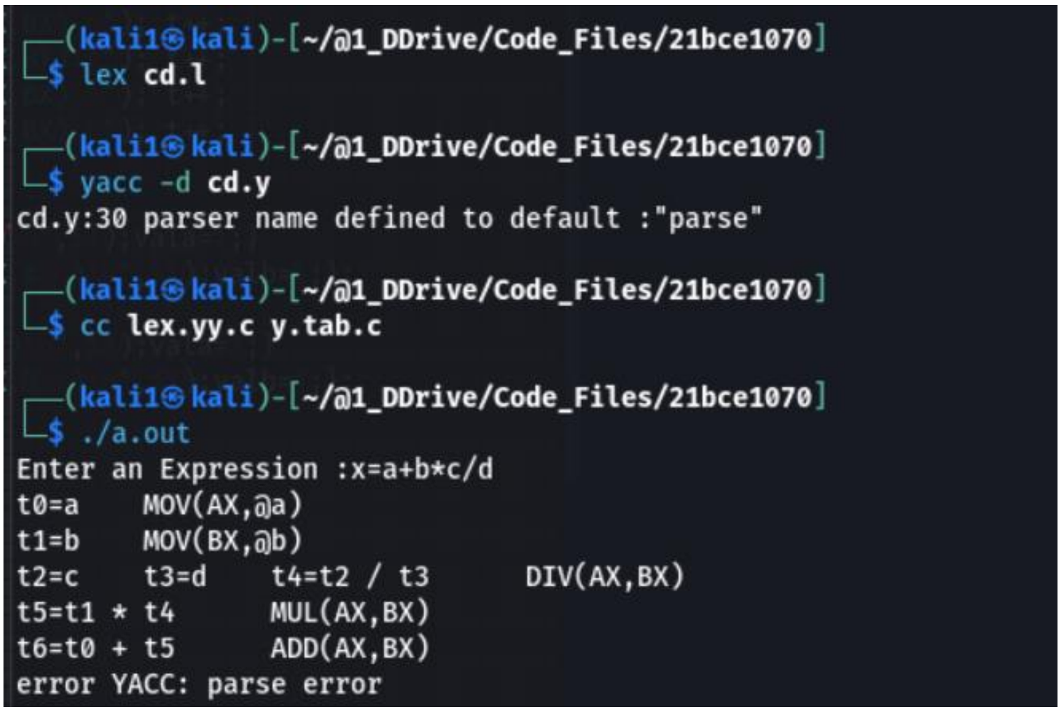
.l

```
%{
#include"y.tab.h"
extern int yylval;
%}

%%

[0-9]* { yylval = atoi(yytext); return NUM; }
[a-zA-Z]* { yylval = yytext[0]; return NAME; }
[ \t] { }
"\n" return 0;
. return yytext[0];

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

```
}


.y

%start GOAL
%token NUM NAME NL


%{
#include<stdlib.h>
#include<stdio.h>
int yylex(void);
int yyerror(char *s);
int vala=0,valb=0;
int t=0;
%}


%%
GOAL : NAME '=' EXP NL{ printf("Answer:t%d\t",$3); printf("MOV(@%c,AX)\n",$1); };


EXP : T '+' EXP { $$=t; printf("t%d=t%d + t%d\t",t,$1,$3); printf("ADD(AX,BX)\n"); t++; }
    | T '-' EXP { $$=t; printf("t%d=t%d - t%d\t",t,$1,$3); printf("SUB(AX,BX)\n"); t++; }
    | T '*' EXP { $$=t; printf("t%d=t%d * t%d\t",t,$1,$3); printf("MUL(AX,BX)\n"); t++; }
    | T '/' EXP { $$=t; printf("t%d=t%d / t%d\t",t,$1,$3); printf("DIV(AX,BX)\n"); t++; }
    | T { $$=$1;} ;


T : NUM { $$=t; printf("t%d=%d\t",t,$1); if(vala==0) {printf("MOV(AX,#%d)\n",$1);vala=1;}
```

```
                                        else if(valb==0) {printf("MOV(BX,#%d)\n",$1);valb=1;};
                                        t++;}
  | NAME { $$=t; printf("t%d=%c\t",t,$1); if(vala==0) {printf("MOV(AX,@%c)\n",$1);vala=1;}
                                        else if(valb==0) {printf("MOV(BX,@%c)\n",$1);valb=1;};
 t++;};


%%


int main()
{
  printf("Enter an Expression :");
  yyparse();
  return 0;
}


int yyerror (char *msg) {
    return printf ("error YACC: %s\n", msg);
}
```

Output



```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ lex cd.l

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ yacc -d cd.y
cd.y:30 parser name defined to default :"parse"

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ cc lex.yy.c y.tab.c

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ ./a.out
Enter an Expression :x=a+b*c/d
t0=a    MOV(AX,@a)
t1=b    MOV(BX,@b)
t2=c    t3=d    t4=t2 / t3      DIV(AX,BX)
t5=t1 * t4      MUL(AX,BX)
t6=t0 + t5      ADD(AX,BX)
error YACC: parse error
```