# Practical - 6

2CS701 – Compiler Construction

Harshit Gajipara
19BCE059

## Aim:

Intermediate Code Generation: To generate Three Address code for assignment statement.

## Code:

practical6.l

```
%{
    #include "y.tab.h"
%}

%%
[0-9]+? {yylval.sym=(char)yytext[0]; return NUMBER;}
[a-zA-Z]+? {yylval.sym=(char)yytext[0]; return LETTER;}
\n {return 0;}
. {return yytext[0];}

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

practical6.y

```
%{
    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>
    void ThreeAddressCode();
    void triple();
    void qudraple();
    char AddToTable(char, char, char);
    int ind = 0;      // count number of lines
    char temp = '1'; // for t1,t2,t3.....
    struct incod
    {
        char opd1;
        char opd2;
        char opr;
    };
%}

%union
{
    char sym;
}

%token <sym> LETTER NUMBER
%type <sym> expr
%left '+'
%left '*' '/'
%left '-'

%%
statement :
LETTER'='expr';'{AddToTable((char)$1,(char)$3,'=');}
        | expr';'
;

expr : expr'+'expr{$$ = AddToTable((char)$1,(char)$3,'+');}
    | expr'-'expr {$$ = AddToTable((char)$1,(char)$3,'-');}
    | expr'*'expr {$$ = AddToTable((char)$1,(char)$3,'*');}
    | expr'/'expr {$$ = AddToTable((char)$1,(char)$3,'/');}
    | '('expr')' {$$ = (char)$2;}
    | NUMBER {$$ = (char)$1;}
    | LETTER {$$ = (char)$1;}
    | '-'expr {$$ = AddToTable((char)$2,(char)'\t','-');}
    ;

%%

yyerror(char *s)
```

```
{
    printf("%s", s);
    exit(0);
}

struct incod code[20];

char AddToTable(char opd1, char opd2, char opr)
{
    code[ind].opd1 = opd1;
    code[ind].opd2 = opd2;
    code[ind].opr = opr;
    ind++;
    return temp++;
}

void ThreeAddressCode()
{
    int cnt = 0;
    char temp = '1';
    printf("\n\n\t THREE ADDRESS CODE\n\n");
    while (cnt < ind)
    {
        if (code[cnt].opr != '=')
            printf("t%c : = \t", temp++);
        if (isalpha(code[cnt].opd1))
            printf(" %c\t", code[cnt].opd1);
        else if (code[cnt].opd1 >= '1' && code[cnt].opd1 <=
'9')
            printf("t%c\t", code[cnt].opd1);
        printf(" %c\t", code[cnt].opr);
        if (isalpha(code[cnt].opd2))
            printf(" %c\n", code[cnt].opd2);
        else if (code[cnt].opd2 >= '1' && code[cnt].opd2 <=
'9')
            printf("t%c\n", code[cnt].opd2);
        cnt++;
    }
}

void quadraple()
{
    int cnt = 0;
    char temp = '1';
    printf("\n\n\t QUADRAPLE CODE\n\n");
    while (cnt < ind)
    {
        printf(" %c\t", code[cnt].opr);
        if (code[cnt].opr == '=')
        {
            if (isalpha(code[cnt].opd2))
                printf(" %c\t \t", code[cnt].opd2);
            else if (code[cnt].opd2 >= '1' && code[cnt].opd2
<= '9')
                printf("t%c\t \t", code[cnt].opd2);
            printf(" %c\n", code[cnt].opd1);
            cnt++;
            continue;
        }
        if (isalpha(code[cnt].opd1))
            printf(" %c\t", code[cnt].opd1);
        else if (code[cnt].opd1 >= '1' && code[cnt].opd1 <=
'9')
            printf("t%c\t", code[cnt].opd1);

        if (isalpha(code[cnt].opd2))
            printf(" %c\t", code[cnt].opd2);
        else if (code[cnt].opd2 >= '1' && code[cnt].opd2 <=
'9')
            printf("t%c\t", code[cnt].opd2);
        else
            printf(" %c", code[cnt].opd2);
        printf("t%c\n", temp++);
        cnt++;
    }
}
```

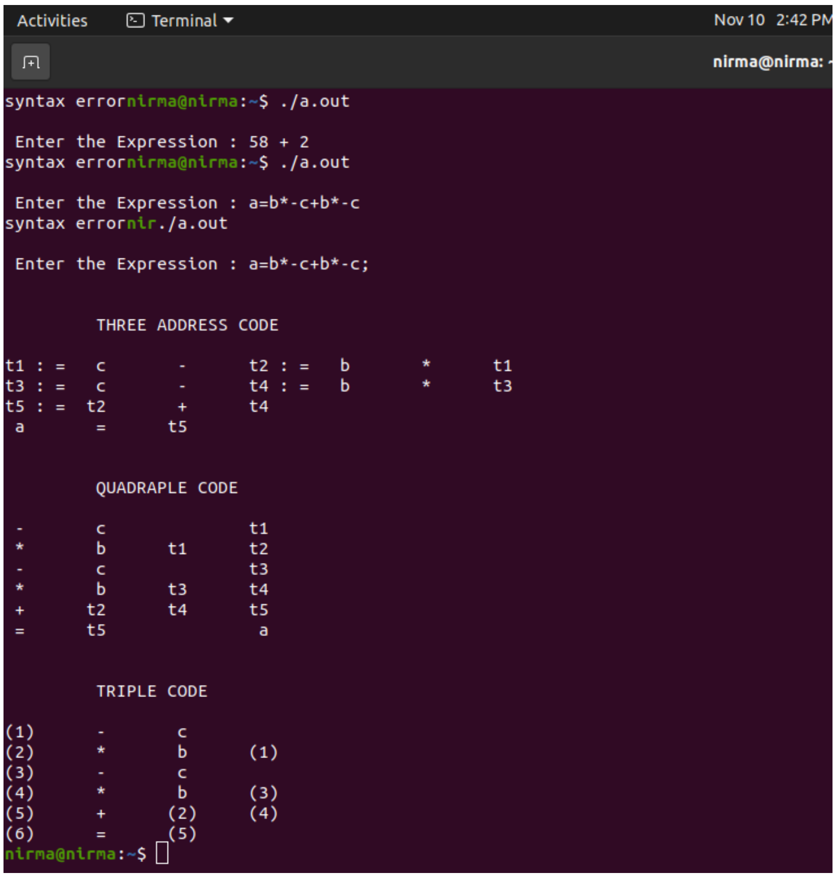
```c
void triple()
{
    int cnt = 0;
    char temp = '1';
    printf("\n\n\t TRIPLE CODE\n\n");
    while (cnt < ind)
    {
        printf("(%c) \t", temp);
        printf(" %c\t", code[cnt].opr);
        if (code[cnt].opr == '=')
        {
            if (isalpha(code[cnt].opd2))
                printf(" %c \t \t", code[cnt].opd2);
            else if (code[cnt].opd2 >= '1' && code[cnt].opd2
<= '9')
                printf("(%c)\n", code[cnt].opd2);
            cnt++;
            temp++;
            continue;
        }
        if (isalpha(code[cnt].opd1))
            printf(" %c \t", code[cnt].opd1);
        else if (code[cnt].opd1 >= '1' && code[cnt].opd1 <=
'9')
            printf("(%c)\t", code[cnt].opd1);

        if (isalpha(code[cnt].opd2))
            printf(" %c \n", code[cnt].opd2);
        else if (code[cnt].opd2 >= '1' && code[cnt].opd2 <=
'9')
            printf("(%c)\n", code[cnt].opd2);
        else
            printf(" %c\n", code[cnt].opd2);
        cnt++;
        temp++;
    }
}
```

```
main()
{
    printf("\nEnter the Expression : ");
    yyparse();
    ThreeAddressCode();
    quadraple();
    triple();
}
```

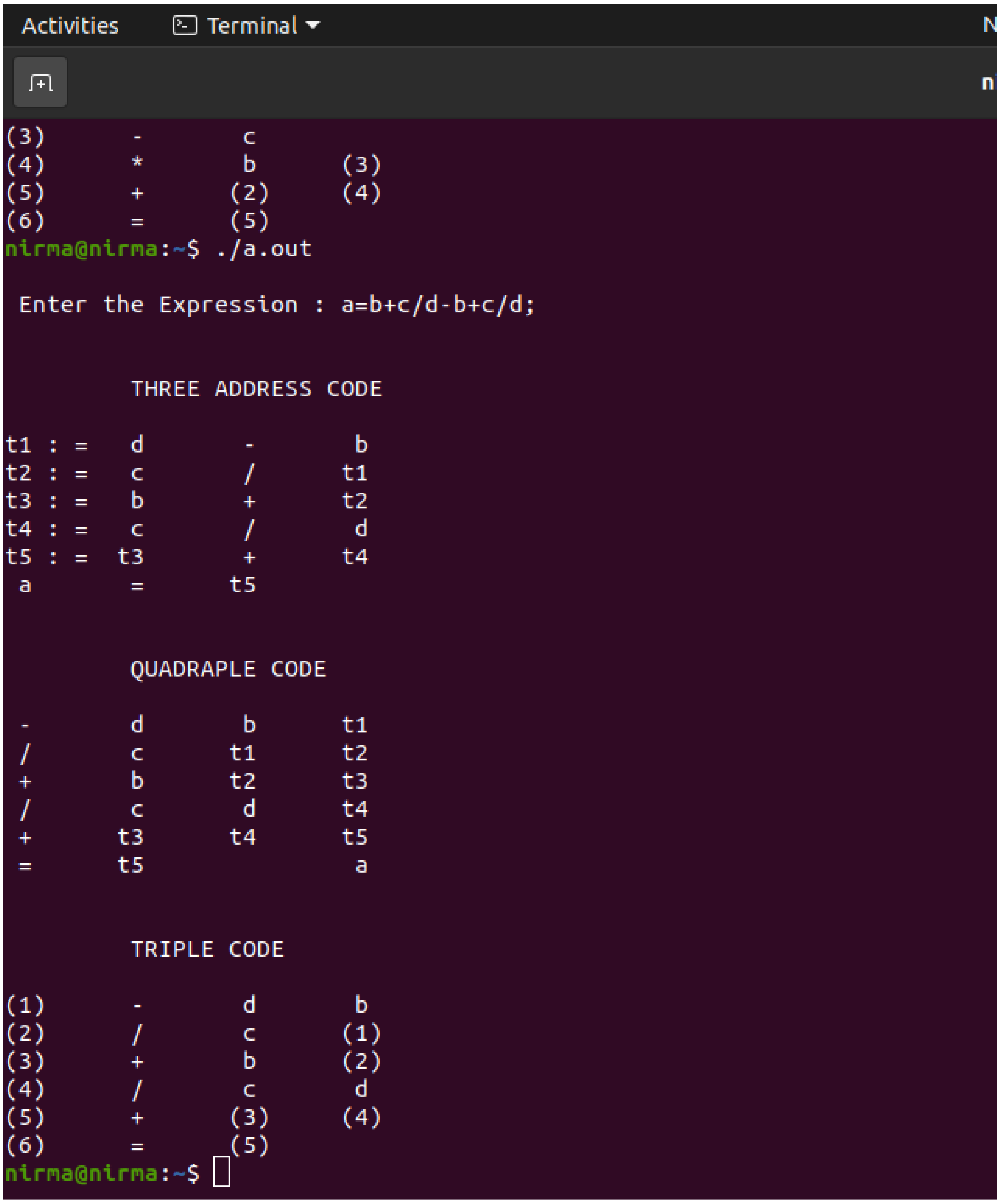
Commands:

- yacc –d practical6.y
- lex practical6.l
- gcc lec.yy.c y.tab.c –ll
- ./a.out

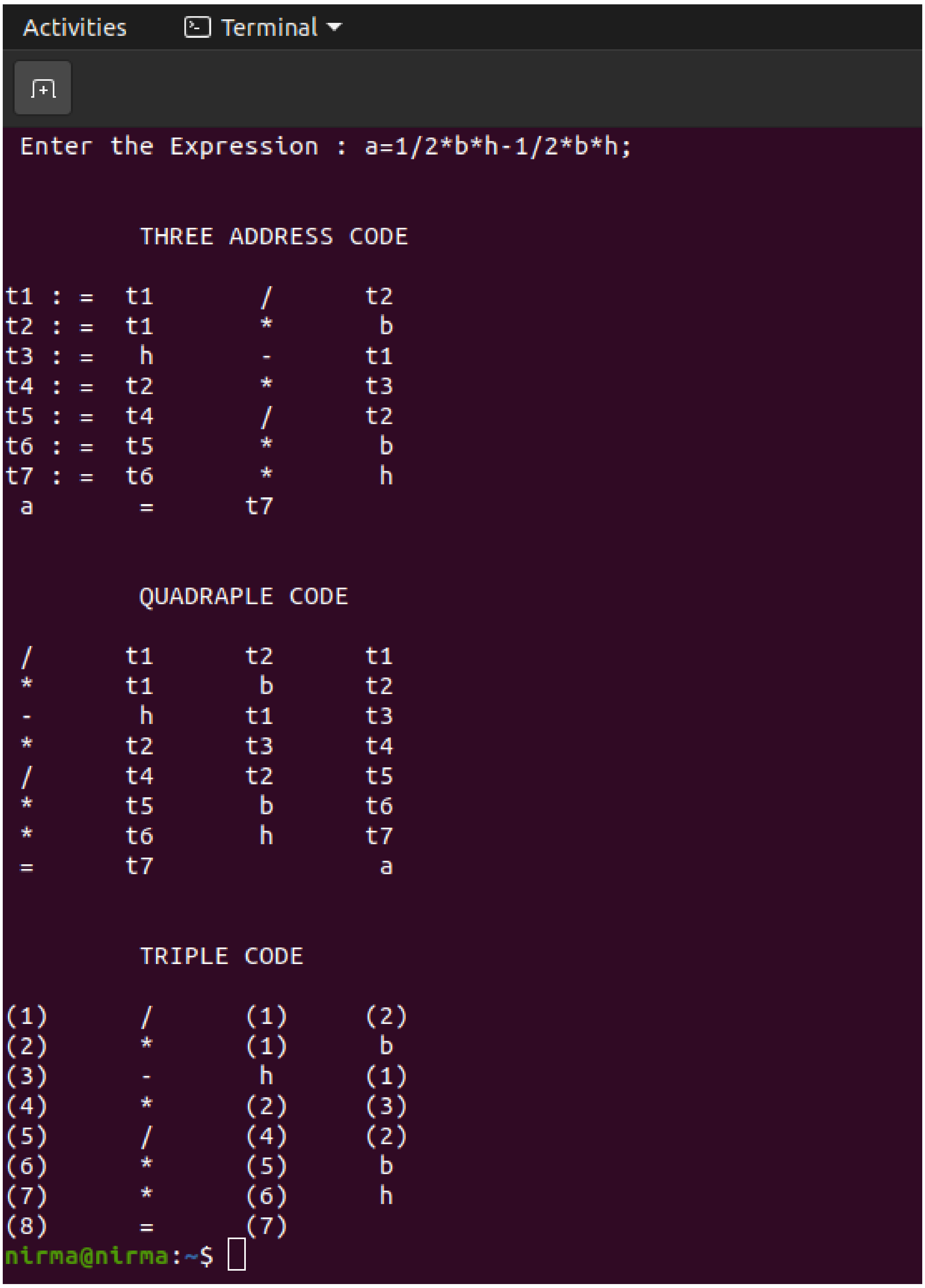
**Output:**



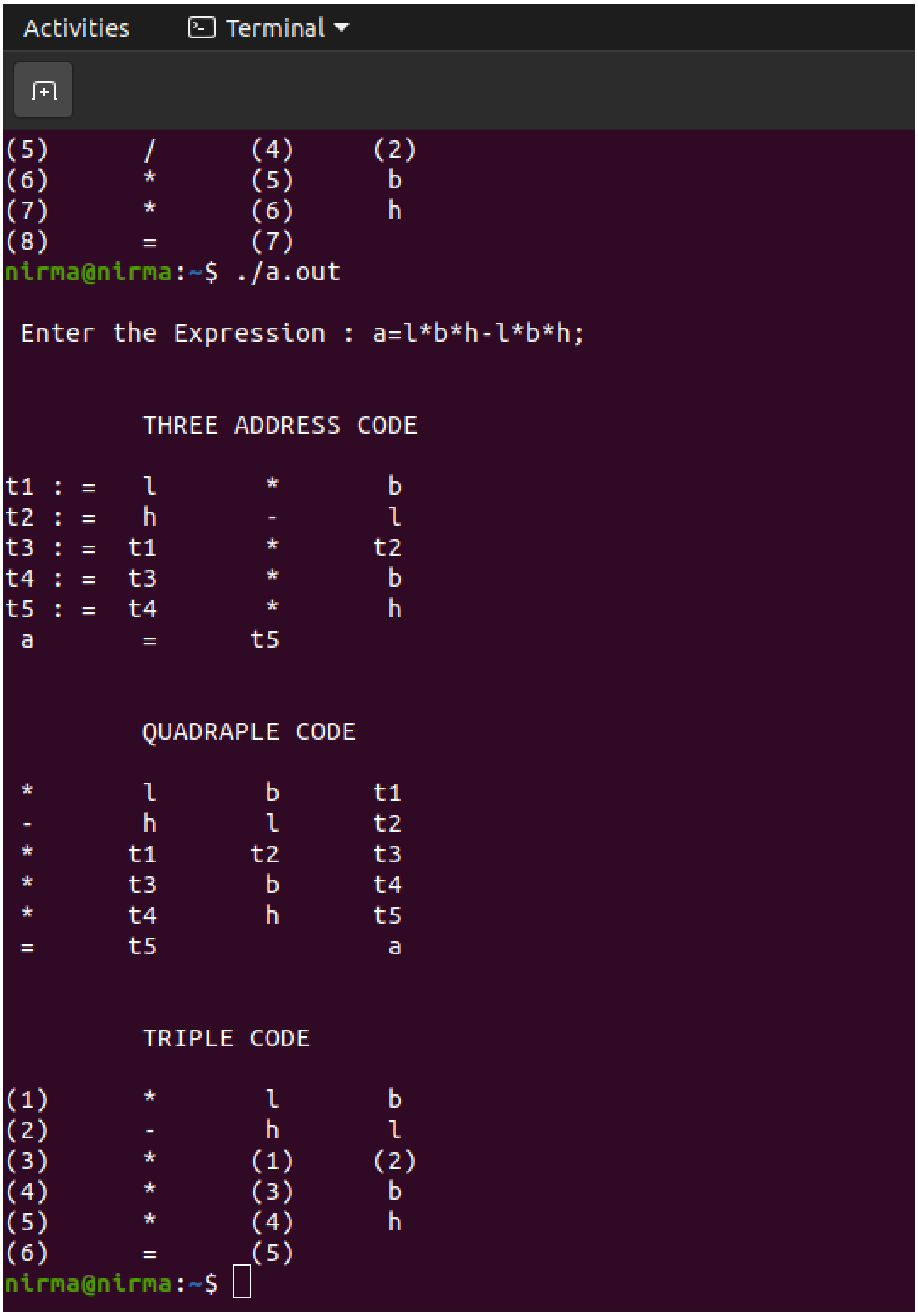
```
syntax errornirma@nirma:~$ ./a.out

 Enter the Expression : 58 + 2
syntax errornirma@nirma:~$ ./a.out

 Enter the Expression : a=b*-c+b*-c
syntax errornir./a.out

 Enter the Expression : a=b*-c+b*-c;


         THREE ADDRESS CODE

t1 : =   c       -      t2 : =   b       *      t1
t3 : =   c       -      t4 : =   b       *      t3
t5 : =  t2       +      t4
 a       =      t5


         QUADRAPLE CODE

 -       c              t1
 *       b      t1      t2
 -       c              t3
 *       b      t3      t4
 +      t2      t4      t5
 =      t5               a


         TRIPLE CODE

(1)      -       c
(2)      *       b      (1)
(3)      -       c
(4)      *       b      (3)
(5)      +      (2)     (4)
(6)      =      (5)
nirma@nirma:~$ 
```



```
(3)       -       c
(4)       *       b       (3)
(5)       +      (2)      (4)
(6)       =      (5)
nirma@nirma:~$ ./a.out

 Enter the Expression : a=b+c/d-b+c/d;


         THREE ADDRESS CODE

t1 : =   d       -        b
t2 : =   c       /       t1
t3 : =   b       +       t2
t4 : =   c       /        d
t5 : =  t3       +       t4
 a       =       t5


         QUADRAPLE CODE

 -       d       b       t1
 /       c      t1       t2
 +       b      t2       t3
 /       c       d       t4
 +      t3      t4       t5
 =      t5                a


         TRIPLE CODE

(1)       -       d       b
(2)       /       c      (1)
(3)       +       b      (2)
(4)       /       c       d
(5)       +      (3)     (4)
(6)       =      (5)
nirma@nirma:~$
```



```
 Enter the Expression : a=1/2*b*h-1/2*b*h;


          THREE ADDRESS CODE

t1 : =  t1       /       t2
t2 : =  t1       *        b
t3 : =   h       -       t1
t4 : =  t2       *       t3
t5 : =  t4       /       t2
t6 : =  t5       *        b
t7 : =  t6       *        h
 a        =      t7


          QUADRAPLE CODE

 /      t1      t2      t1
 *      t1       b      t2
 -       h      t1      t3
 *      t2      t3      t4
 /      t4      t2      t5
 *      t5       b      t6
 *      t6       h      t7
 =      t7               a


          TRIPLE CODE

(1)      /      (1)     (2)
(2)      *      (1)      b
(3)      -       h      (1)
(4)      *      (2)     (3)
(5)      /      (4)     (2)
(6)      *      (5)      b
(7)      *      (6)      h
(8)      =      (7)
nirma@nirma:~$ 
```



```
(5)       /       (4)       (2)
(6)       *       (5)       b
(7)       *       (6)       h
(8)       =       (7)
nirma@nirma:~$ ./a.out

 Enter the Expression : a=l*b*h-l*b*h;


         THREE ADDRESS CODE

t1 : =   l        *         b
t2 : =   h        -         l
t3 : =  t1        *        t2
t4 : =  t3        *         b
t5 : =  t4        *         h
 a        =      t5


         QUADRAPLE CODE

 *       l        b        t1
 -       h        l        t2
 *      t1       t2        t3
 *      t3        b        t4
 *      t4        h        t5
 =      t5                  a


         TRIPLE CODE

(1)       *       l         b
(2)       -       h         l
(3)       *      (1)       (2)
(4)       *      (3)        b
(5)       *      (4)        h
(6)       =      (5)
nirma@nirma:~$
```