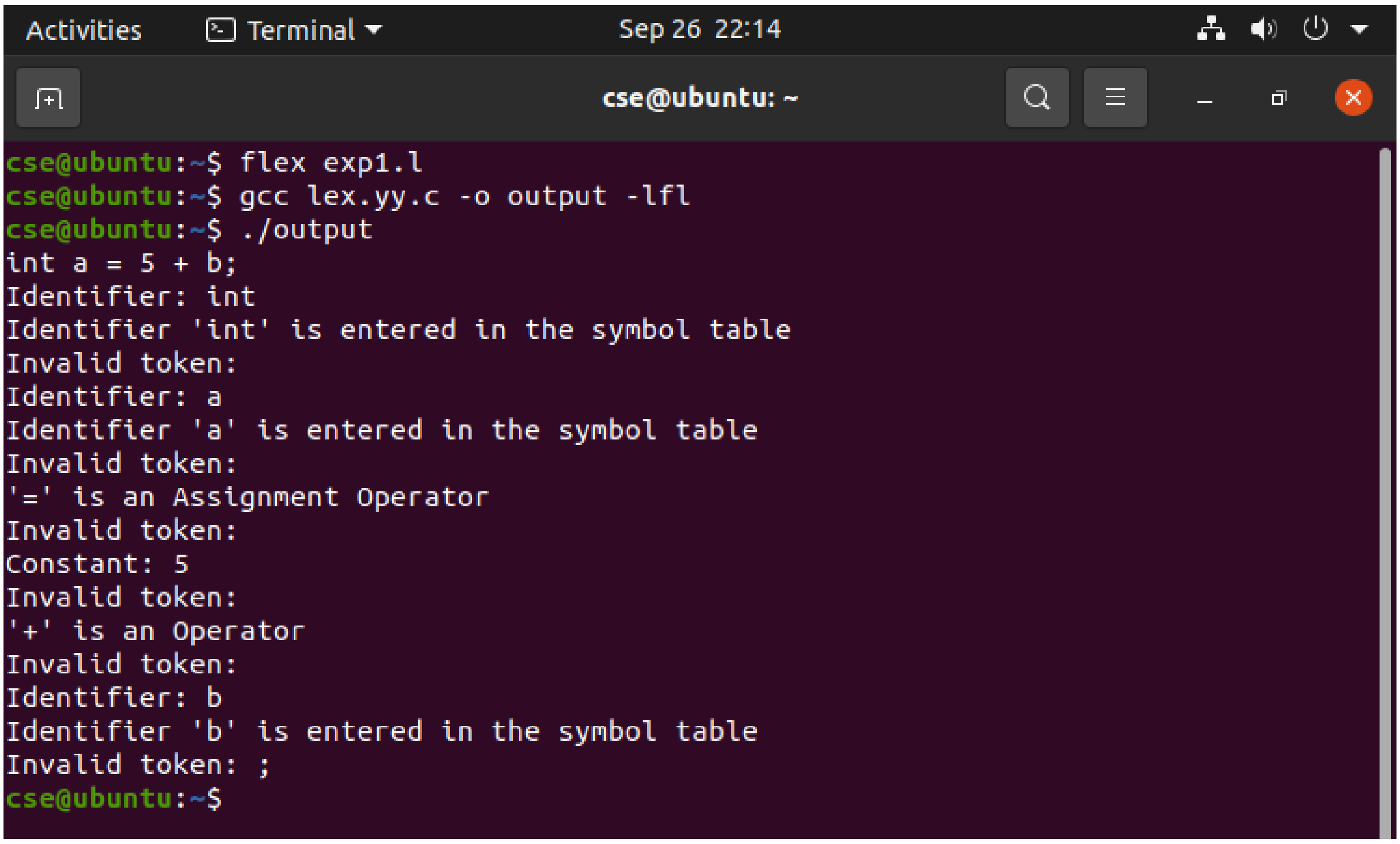
Program:

Exp1.l:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_IDENTIFIER_LENGTH 50

typedef struct {
    char name[MAX_IDENTIFIER_LENGTH];
} Symbol;

Symbol symbol_table[100];
int symbol_count = 0;

void addtosymboltable(const char* identifier){
    if(symbol_count < 100){
        strncpy(symbol_table[symbol_count].name, identifier, MAX_IDENTIFIER_LENGTH-1);
        symbol_table[symbol_count].name[MAX_IDENTIFIER_LENGTH-1] = '\0';
        symbol_count++;
        printf("Identifier '%s' is entered in the symbol table\n", identifier);
    } else {
        printf("Symbol table is full. Cannot add more identifiers.\n");
        exit(0);
    }
}
%}

%option noyywrap
%option yylineno

%%
```

```
[\t]+           ;  /* ignore tabs */
\n              ;  /* ignore newline */
\/\*([^*]|\*+[^*/])*\*+\/  ;  /* ignore comments */

[0-9]+           { printf("Constant: %s\n", yytext); }
=               { printf("'%s' is an Assignment Operator\n", yytext); }
[\*\+\-]         { printf("'%s' is an Operator\n", yytext); }
[a-zA-Z][a-zA-Z0-9]* { printf("Identifier: %s\n", yytext); addtosymboltable(yytext); }
.               { printf("Invalid token: %s\n", yytext); }

%%

int main() {
   yylex();
   return 0;
}
```

Output:



```
cse@ubuntu:~$ flex exp1.l
cse@ubuntu:~$ gcc lex.yy.c -o output -lfl
cse@ubuntu:~$ ./output
int a = 5 + b;
Identifier: int
Identifier 'int' is entered in the symbol table
Invalid token:
Identifier: a
Identifier 'a' is entered in the symbol table
Invalid token:
'=' is an Assignment Operator
Invalid token:
Constant: 5
Invalid token:
'+' is an Operator
Invalid token:
Identifier: b
Identifier 'b' is entered in the symbol table
Invalid token: ;
cse@ubuntu:~$
```

Program:

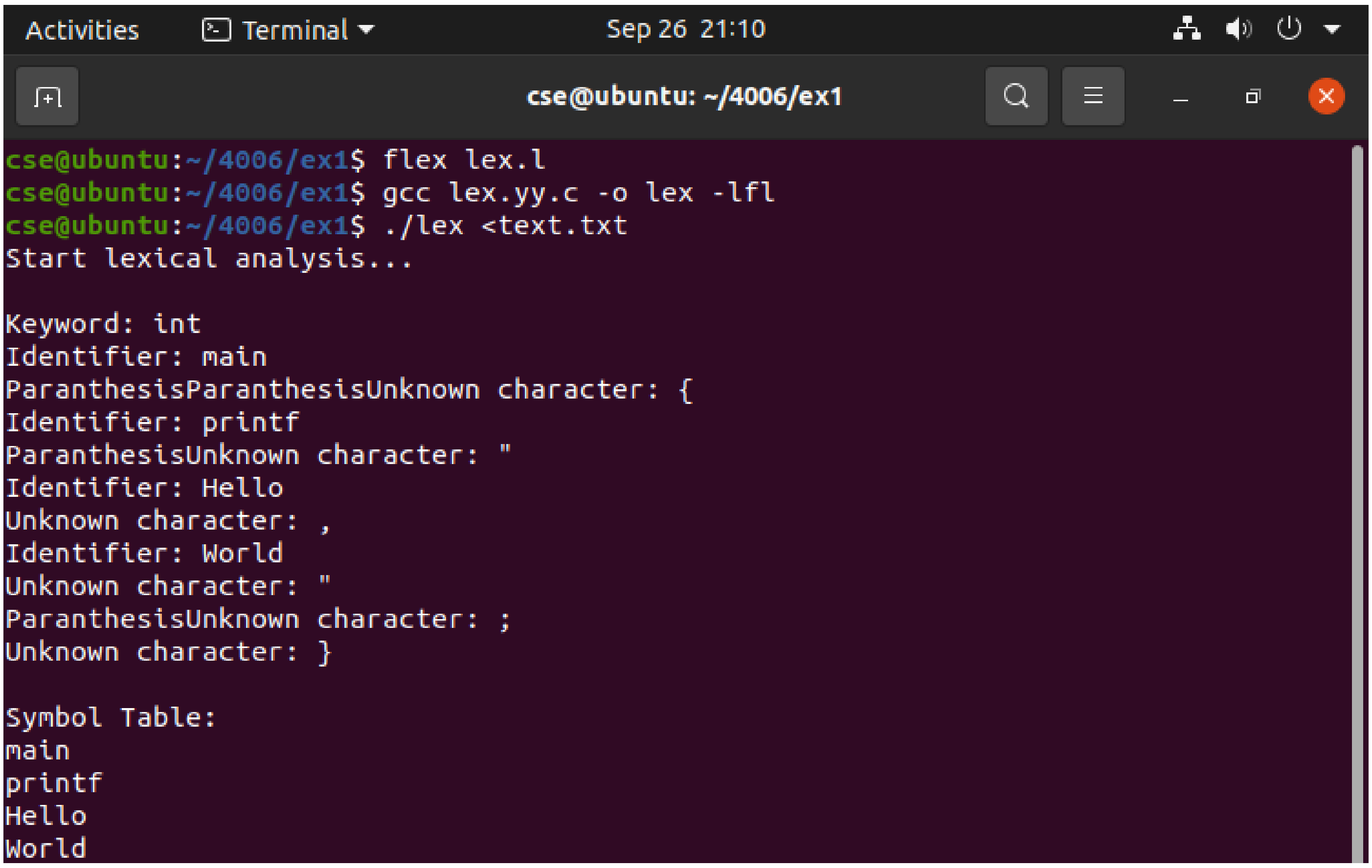
Exp2.l:

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_SYMBOLS 100
char* symbol_table[MAX_SYMBOLS];
int symbol_count = 0;

void add_symbol(char* id) {
    for (int i = 0; i < symbol_count; i++) {
        if (strcmp(symbol_table[i], id) == 0)
            return;
    }
    if (symbol_count < MAX_SYMBOLS)
        symbol_table[symbol_count++] = strdup(id);
    else
        fprintf(stderr, "Symbol table full: %s\n", id);
}
%}

%%
[ \t\n]+                    ;
"/*"([^*]*|\*+[^*/])*"*"+"/"  ;
"//".*                      ;
[0-9]+                      { printf("Constant: %s\n", yytext); }
[_a-zA-Z][_a-zA-Z0-9]*      {
    if (
        strcmp(yytext, "auto") == 0 ||
        strcmp(yytext, "break") == 0 ||
        strcmp(yytext, "case") == 0 ||
        strcmp(yytext, "char") == 0 ||
        strcmp(yytext, "const") == 0 ||
        strcmp(yytext, "continue") == 0 ||
        strcmp(yytext, "default") == 0 ||
        strcmp(yytext, "do") == 0 ||
        strcmp(yytext, "double") == 0 ||
        strcmp(yytext, "else") == 0 ||
        strcmp(yytext, "enum") == 0 ||
        strcmp(yytext, "extern") == 0 ||
        strcmp(yytext, "float") == 0 ||
        strcmp(yytext, "for") == 0 ||
        strcmp(yytext, "goto") == 0 ||
        strcmp(yytext, "if") == 0 ||
        strcmp(yytext, "inline") == 0 ||
        strcmp(yytext, "int") == 0 ||
        strcmp(yytext, "long") == 0 ||
        strcmp(yytext, "register") == 0 ||
        strcmp(yytext, "restrict") == 0 ||
        strcmp(yytext, "return") == 0 ||
        strcmp(yytext, "short") == 0 ||
        strcmp(yytext, "signed") == 0 ||
        strcmp(yytext, "sizeof") == 0 ||
        strcmp(yytext, "static") == 0 ||
        strcmp(yytext, "struct") == 0 ||
        strcmp(yytext, "switch") == 0 ||
        strcmp(yytext, "typedef") == 0 ||
        strcmp(yytext, "union") == 0 ||
        strcmp(yytext, "unsigned") == 0 ||
        strcmp(yytext, "void") == 0 ||
        strcmp(yytext, "volatile") == 0 ||
        strcmp(yytext, "while") == 0
    ) {
        printf("Keyword: %s\n", yytext);
    } else {
        printf("Identifier: %s\n", yytext);
        add_symbol(yytext);
    }
}
"+"|"-"|"*"|"/"|"="|"=="|"<"|">"|"!="|"<="|">=" { printf("Operator: %s\n", yytext); }
"("|")" {printf("Paranthesis");}
.                        { printf("Unknown character: %s\n", yytext); }
%%

int main() {
    printf("Start lexical analysis...\n\n");
    yylex();
    printf("\nSymbol Table:\n");
    for (int i = 0; i < symbol_count; i++)
        printf("%s\n", symbol_table[i]);
    return 0;
}
```

Text.txt:

```
int main() {
    printf("Hello, World");
}
```

Output:



```
cse@ubuntu:~/4006/ex1$ flex lex.l
cse@ubuntu:~/4006/ex1$ gcc lex.yy.c -o lex -lfl
cse@ubuntu:~/4006/ex1$ ./lex <text.txt
Start lexical analysis...

Keyword: int
Identifier: main
ParanthesisParanthesisUnknown character: {
Identifier: printf
ParanthesisUnknown character: "
Identifier: Hello
Unknown character: ,
Identifier: World
Unknown character: "
ParanthesisUnknown character: ;
Unknown character: }

Symbol Table:
main
printf
Hello
World
```

Program:

Ex3a.l

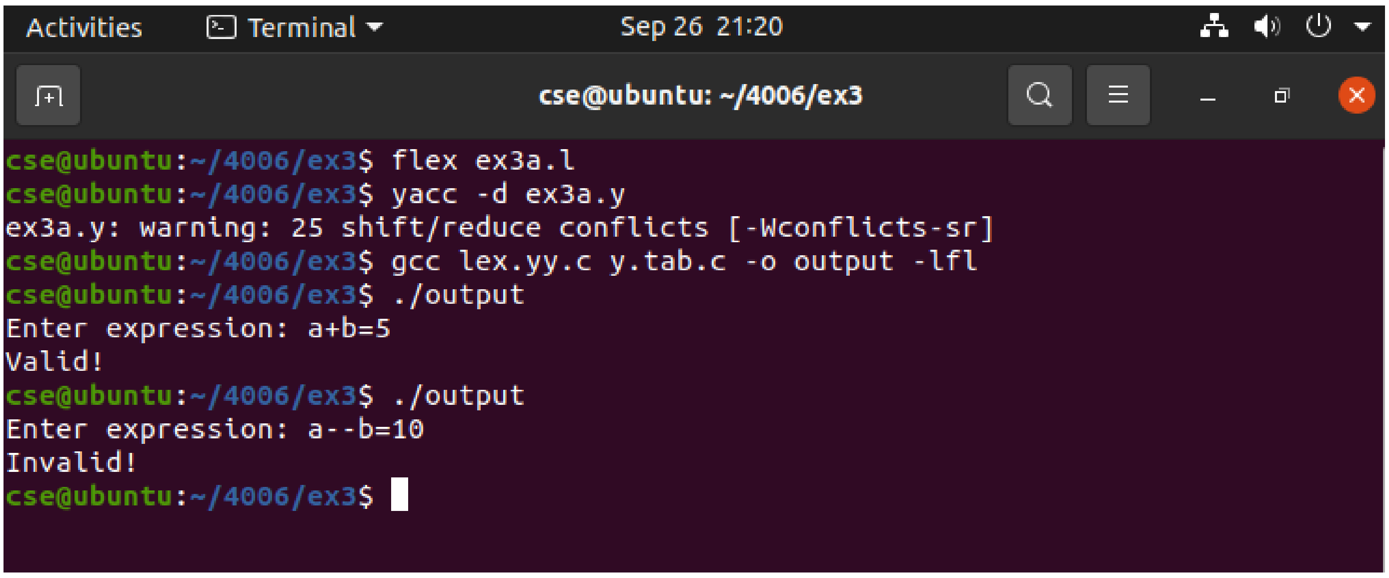
```
%{
#include "y.tab.h"
%}

%%

[0-9]+          { return NUMBER; }
[a-zA-Z][a-zA-Z0-9]*   { return ID; }
[+\-*/]         { return yytext[0]; }
[ \t\n]         ;   /* ignore spaces and newlines */
.               { return yytext[0]; }

%%

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

Ex3a.y

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

int yylex();
int yyerror(const char *s);
%}

%token NUMBER ID

%%
```

```
expr:
      expr '+' expr
    | expr '-' expr
    | expr '*' expr
    | expr '/' expr
    | expr '=' expr
    | NUMBER
    | ID
    ;

%%

int main() {
    printf("Enter expression: ");
    if (yyparse() == 0) {    // parsing success
        printf("Valid!\n");
    }
    return 0;
}

int yyerror(const char *s) {
    printf("Invalid!\n");
    exit(1);
}
```

Output:



```
cse@ubuntu:~/4006/ex3$ flex ex3a.l
cse@ubuntu:~/4006/ex3$ yacc -d ex3a.y
ex3a.y: warning: 25 shift/reduce conflicts [-Wconflicts-sr]
cse@ubuntu:~/4006/ex3$ gcc lex.yy.c y.tab.c -o output -lfl
cse@ubuntu:~/4006/ex3$ ./output
Enter expression: a+b=5
Valid!
cse@ubuntu:~/4006/ex3$ ./output
Enter expression: a--b=10
Invalid!
cse@ubuntu:~/4006/ex3$ 
```

Program:

Ex3b.l:

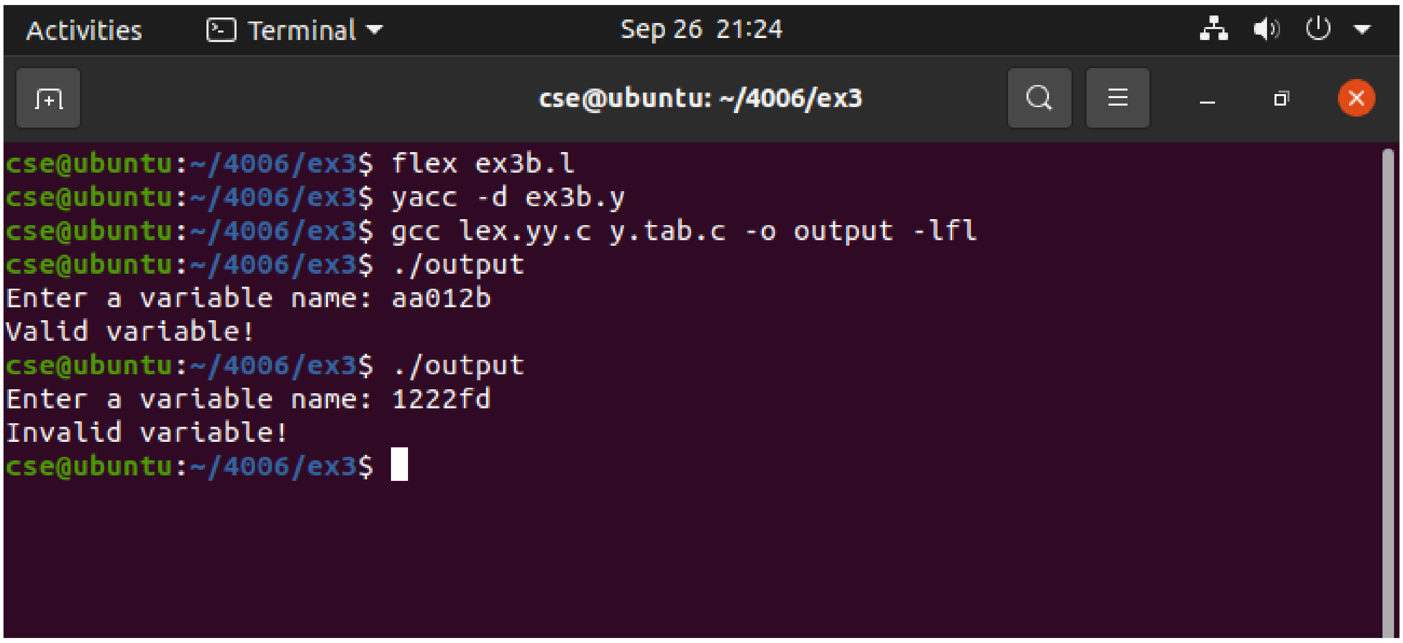
```
%{
#include "y.tab.h"
%}

%%

[a-zA-Z][a-zA-Z0-9]*   { return ID; }
\n                     { return '\n'; }
.                      { return yytext[0]; }

%%

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

Ex3b.y

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

int yylex();
int yyerror(const char *s);
%}

%token ID

%%

line:
    ID '\n'    { printf("Valid variable!\n"); }
```

```
 | '\n'     { /* ignore empty line */ }
 ;

%%

int main() {
   printf("Enter a variable name: ");
   yyparse();
   return 0;
}

int yyerror(const char *s) {
   printf("Invalid variable!\n");
   return 0;
}
```

Output:



```
cse@ubuntu:~/4006/ex3$ flex ex3b.l
cse@ubuntu:~/4006/ex3$ yacc -d ex3b.y
cse@ubuntu:~/4006/ex3$ gcc lex.yy.c y.tab.c -o output -lfl
cse@ubuntu:~/4006/ex3$ ./output
Enter a variable name: aa012b
Valid variable!
cse@ubuntu:~/4006/ex3$ ./output
Enter a variable name: 1222fd
Invalid variable!
cse@ubuntu:~/4006/ex3$ 
```

Program:

Exp3c.l:

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

%%

"for"           { return FOR; }
"while"         { return WHILE; }
"if"            { return IF; }
"else"          { return ELSE; }
"switch"        { return SWITCH; }
"case"          { return CASE; }
"default"       { return DEFAULT; }

"("             { return LPAREN; }
")"             { return RPAREN; }
"{"             { return LBRACE; }
"}"             { return RBRACE; }
";"             { return SEMICOLON; }
":"             { return COLON; }

[a-zA-Z_][a-zA-Z0-9_]*  { return ID; }
[0-9]+                  { return NUMBER; }

[ \t\n]+        ; /* ignore spaces */
.               { return yytext[0]; }

%%
```

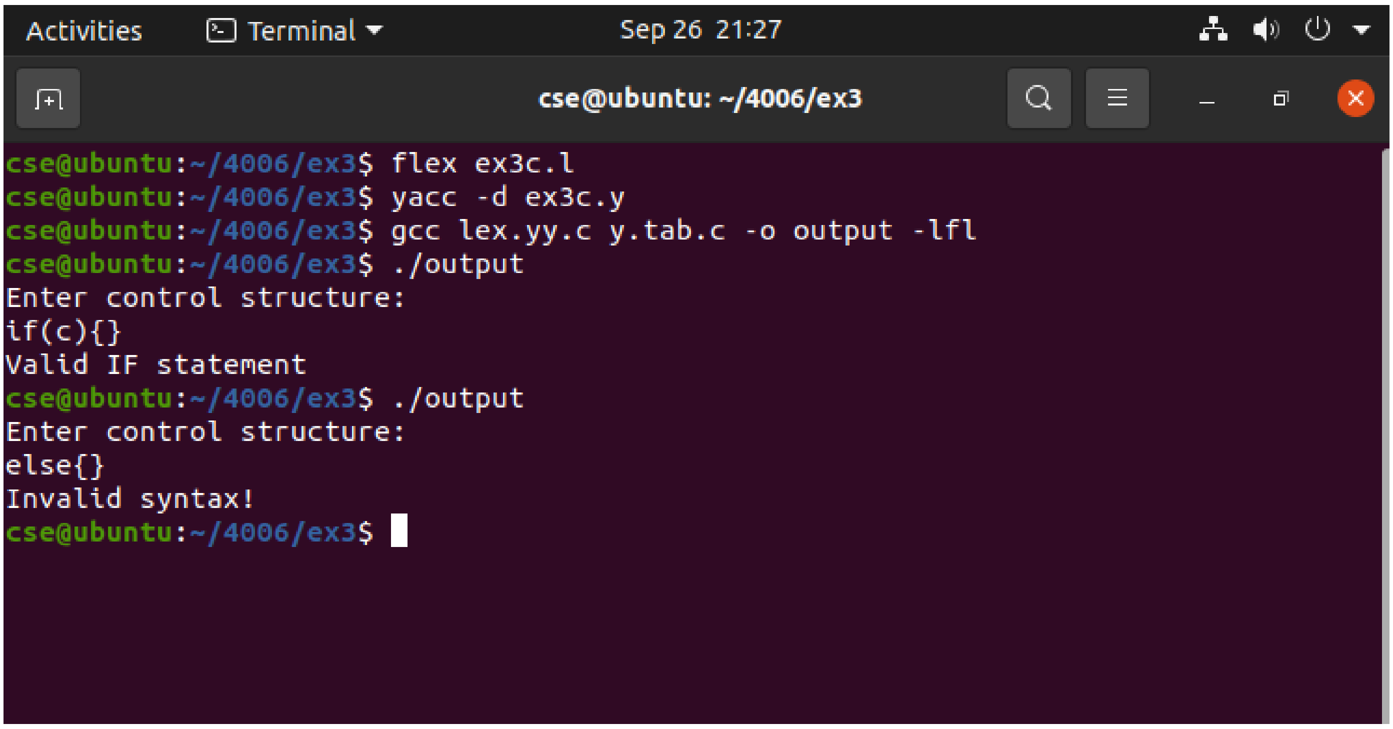
```
int yywrap() { return 1; }
```

Exp3c.y:

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

int yylex();
int yyerror(const char *s);
%}

%token FOR WHILE IF ELSE SWITCH CASE DEFAULT
%token ID NUMBER LPAREN RPAREN LBRACE RBRACE SEMICOLON COLON

%%

stmt:
      FOR LPAREN ID SEMICOLON ID SEMICOLON ID RPAREN LBRACE RBRACE
        { printf("Valid FOR loop\n"); }
    | WHILE LPAREN opt_id RPAREN LBRACE RBRACE
        { printf("Valid WHILE loop\n"); }
    | IF LPAREN opt_id RPAREN LBRACE RBRACE
        { printf("Valid IF statement\n"); }
    | IF LPAREN opt_id RPAREN LBRACE RBRACE ELSE LBRACE RBRACE
        { printf("Valid IF-ELSE statement\n"); }
    | SWITCH LPAREN opt_id RPAREN LBRACE CASE NUMBER COLON DEFAULT COLON
RBRACE
        { printf("Valid SWITCH statement\n"); }
    ;

opt_id:
    | ID
    ;
```

```
%%

int main() {
    printf("Enter control structure:\n");
    yyparse();
    return 0;
}

int yyerror(const char *s) {
    printf("Invalid syntax!\n");
    return 0;
}
```

Output:



```
cse@ubuntu:~/4006/ex3$ flex ex3c.l
cse@ubuntu:~/4006/ex3$ yacc -d ex3c.y
cse@ubuntu:~/4006/ex3$ gcc lex.yy.c y.tab.c -o output -lfl
cse@ubuntu:~/4006/ex3$ ./output
Enter control structure:
if(c){}
Valid IF statement
cse@ubuntu:~/4006/ex3$ ./output
Enter control structure:
else{}
Invalid syntax!
cse@ubuntu:~/4006/ex3$
```

Program:

Exp3d.l:

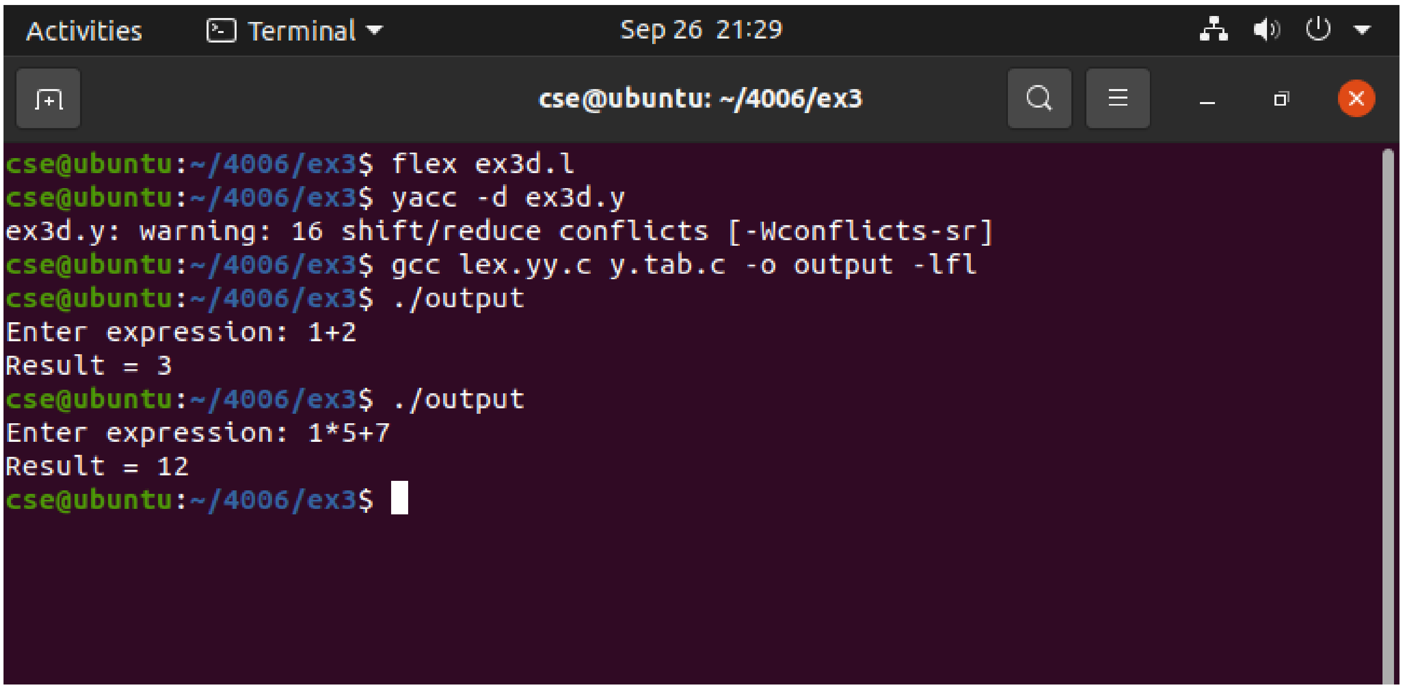
```
%{
#include "y.tab.h"
%}

%%

[0-9]+          { yylval = atoi(yytext); return NUMBER; }
[+\-*/()]       { return yytext[0]; }
[ \t]           ;      /* ignore spaces */
\n              { return '\n'; }
.               { return yytext[0]; }

%%

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

Exp3d.y:

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

int yylex();
int yyerror(const char *s);
%}

%token NUMBER

%%
```

```
input:
      /* empty */
    | input expr '\n'   { printf("Result = %d\n", $2); }
    ;

expr:
      expr '+' expr   { $$ = $1 + $3; }
    | expr '-' expr   { $$ = $1 - $3; }
    | expr '*' expr   { $$ = $1 * $3; }
    | expr '/' expr   { $$ = $1 / $3; }
    | '(' expr ')'    { $$ = $2; }
    | NUMBER          { $$ = $1; }
    ;

%%

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

int yyerror(const char *s) {
    printf("Invalid expression!\n");
    return 0;
}
```

Output:



```
cse@ubuntu:~/4006/ex3$ flex ex3d.l
cse@ubuntu:~/4006/ex3$ yacc -d ex3d.y
ex3d.y: warning: 16 shift/reduce conflicts [-Wconflicts-sr]
cse@ubuntu:~/4006/ex3$ gcc lex.yy.c y.tab.c -o output -lfl
cse@ubuntu:~/4006/ex3$ ./output
Enter expression: 1+2
Result = 3
cse@ubuntu:~/4006/ex3$ ./output
Enter expression: 1*5+7
Result = 12
cse@ubuntu:~/4006/ex3$ 
```

Program:

Exp4.l:

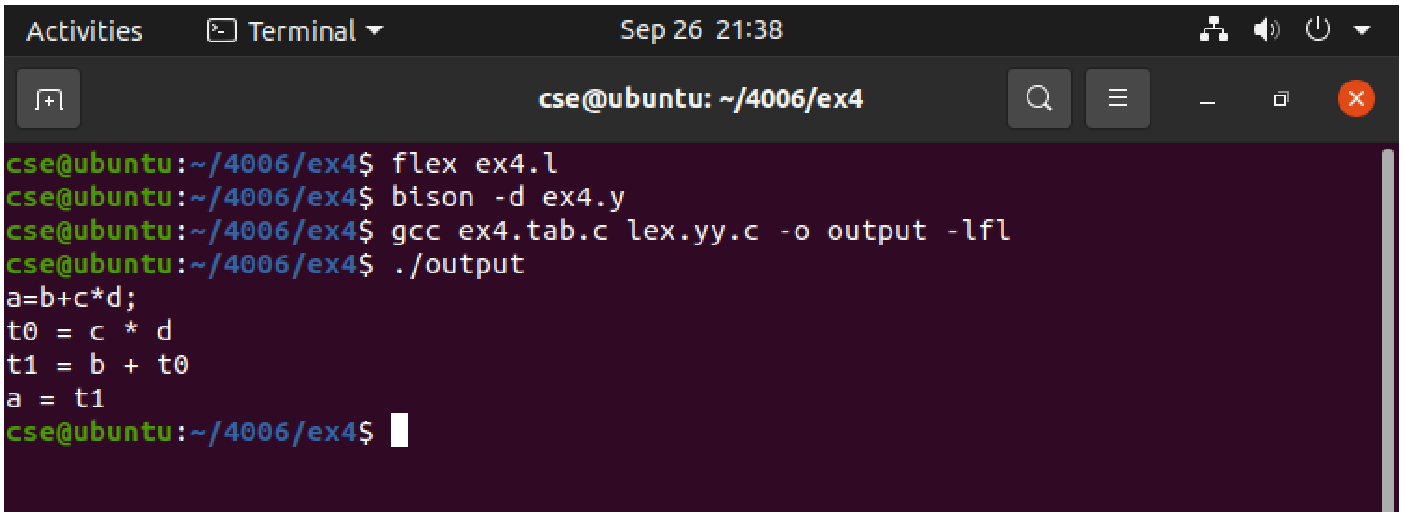
```
%{
#include "y.tab.h"
#include <stdlib.h>
%}

%%

[0-9]+                  { yylval.ival = atoi(yytext); return NUMBER; }
[a-zA-Z][a-zA-Z0-9]*    { yylval.sval = strdup(yytext); return ID; }
"="                     { return '='; }
"+"                     { return '+'; }
"*"                     { return '*'; }
";"                     { return ';'; }
[ \t\n]                 ;   // ignore spaces
.                       { return yytext[0]; }

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

Exp4.y:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int tempCount = 0;

char* newTemp() {
    static char buf[32];
    sprintf(buf, "t%d", tempCount++);
    return strdup(buf);
}

extern char* yytext;
int yylex(void);
int yyerror(const char *s);
%}

%union {
    int ival;
    char* sval;
}

%token <ival> NUMBER
%token <sval> ID
%type <sval> expr term factor

%left '+'
%left '*'

%%

stmt: ID '=' expr ';' {
    printf("%s = %s\n", $1, $3);
}
;

expr: expr '+' term {
    char* t = newTemp();
    printf("%s = %s + %s\n", t, $1, $3);
    $$ = t;
}
```

```
| term { $$ = $1; }
;

term: term '*' factor {
    char* t = newTemp();
    printf("%s = %s * %s\n", t, $1, $3);
    $$ = t;
}
| factor { $$ = $1; }
;

factor: ID     { $$ = $1; }
      | NUMBER {
          char buf[20];
          sprintf(buf, "%d", $1);
          $$ = strdup(buf);
      }
;
%%

int main() {
    return yyparse();
}

int yyerror(const char *s) {
    printf("Error: %s\n", s);
    return 0;
}
```

Output:



```
cse@ubuntu:~/4006/ex4$ flex ex4.l
cse@ubuntu:~/4006/ex4$ bison -d ex4.y
cse@ubuntu:~/4006/ex4$ gcc ex4.tab.c lex.yy.c -o output -lfl
cse@ubuntu:~/4006/ex4$ ./output
a=b+c*d;
t0 = c * d
t1 = b + t0
a = t1
cse@ubuntu:~/4006/ex4$ 
```

Program:

Exp5.l:

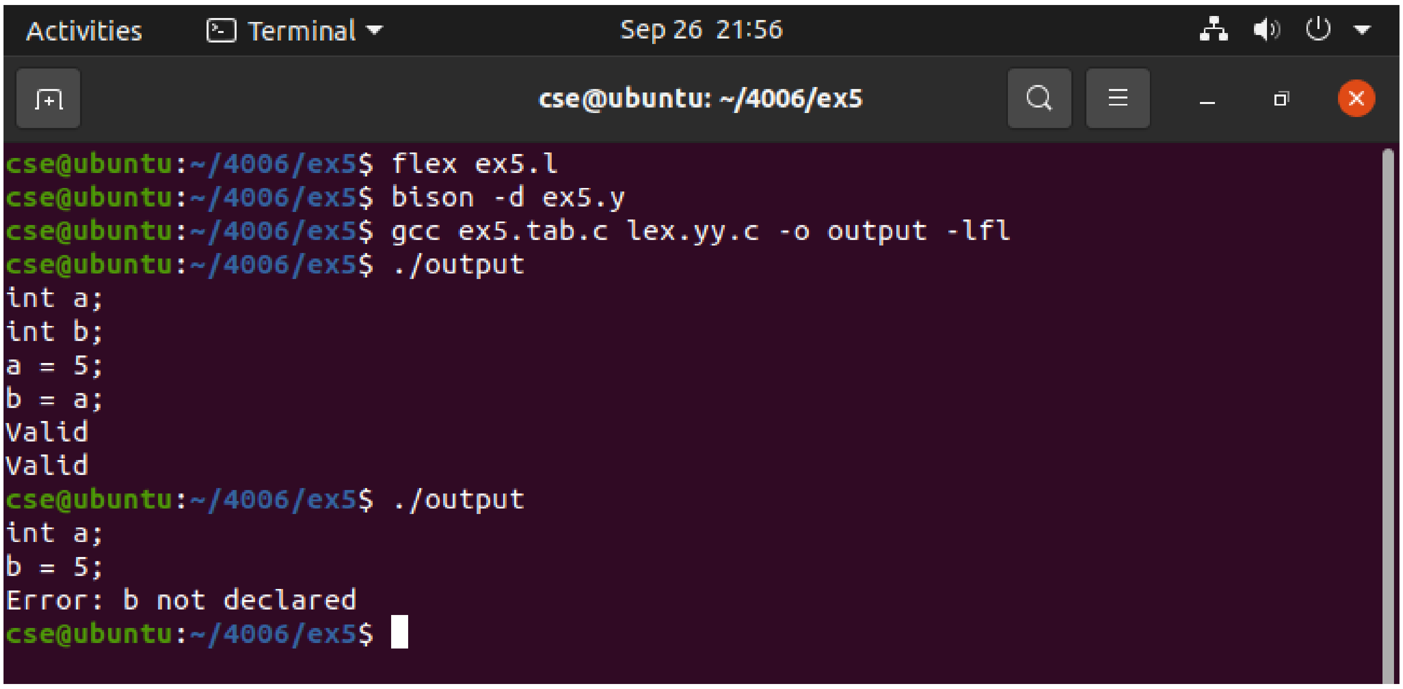
```
%{
#include "y.tab.h"
#include <stdlib.h>
%}

%%
int             { return INT; }
[a-zA-Z][a-zA-Z0-9]*  { yylval.sval = strdup(yytext); return ID; }
[0-9]+          { yylval.sval = strdup(yytext); return NUMBER; }
"="             { return '='; }
";"             { return ';'; }
[ \t\n]         ;  // ignore whitespace
.               { return yytext[0]; }



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

Exp5.y:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    char* name;
    char* type;
} sym;

sym table[100];
int table_index = 0;
void insert(char* name, char* type) {
    table[table_index].name = strdup(name);
    table[table_index].type = strdup(type);
    table_index++;
}
char* lookup(char* name) {
    for(int i=0;i<table_index;i++)
        if(strcmp(table[i].name,name)==0) return table[i].type;
    return NULL;
}

extern char* yytext;
int yylex(void);
int yyerror(const char *s) { printf("Error: %s\n", s); return 0; }
%}

%union { char* sval; }
%token <sval> ID NUMBER
%token INT
%type <sval> expr

%%

program: declarations statements ;

declarations:
    | declarations decl
    ;

decl: INT ID ';' {
    insert($2,"int");
}
;
```

```
statements:
    | statements stmt
    ;

stmt: ID '=' expr ';' {
    char* t = lookup($1);
    if(t==NULL) printf("Error: %s not declared\n",$1);
    else if(strcmp(t,$3)!=0) printf("Type Error: %s and %s mismatch\n",$1,$3);
}
;

expr: NUMBER { $$ = "int"; }
    | ID {
        char* t = lookup($1);
        if(t==NULL) { printf("Error: %s not declared\n",$1); $$ = "int"; }
        else $$ = t;
    }
;
%%

int main() { return yyparse(); }
```

Output:



```
cse@ubuntu:~/4006/ex5$ flex ex5.l
cse@ubuntu:~/4006/ex5$ bison -d ex5.y
cse@ubuntu:~/4006/ex5$ gcc ex5.tab.c lex.yy.c -o output -lfl
cse@ubuntu:~/4006/ex5$ ./output
int a;
int b;
a = 5;
b = a;
Valid
Valid
cse@ubuntu:~/4006/ex5$ ./output
int a;
b = 5;
Error: b not declared
cse@ubuntu:~/4006/ex5$
```

Program:

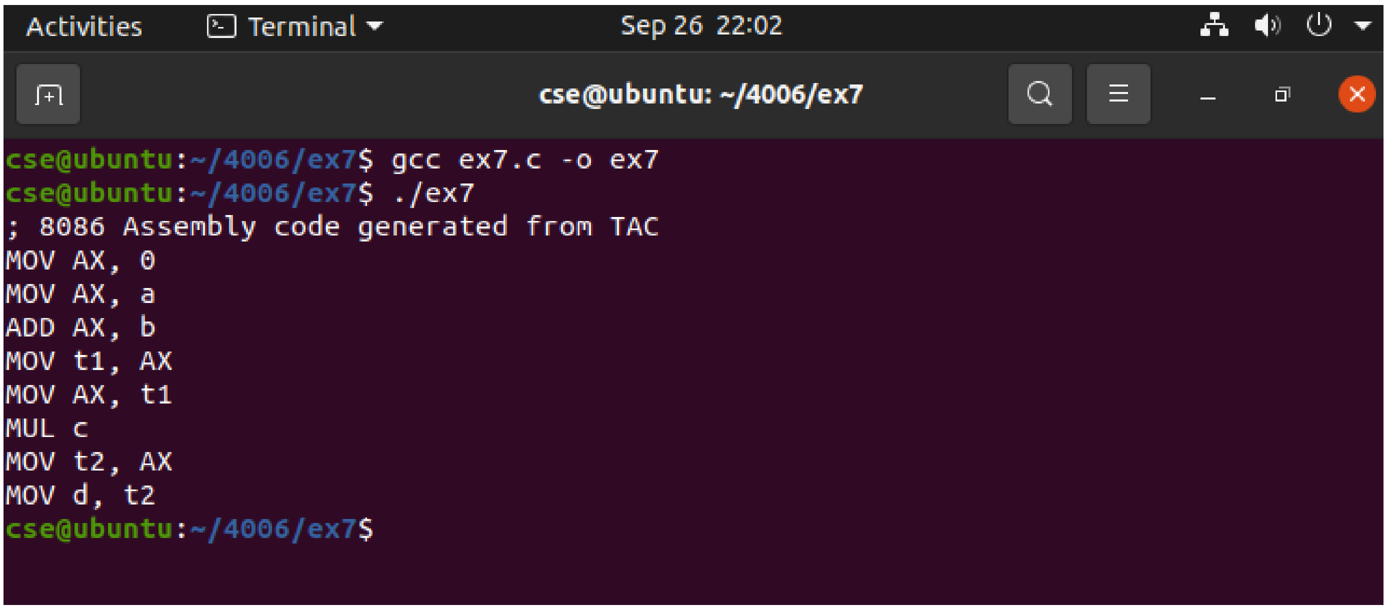
Exp7.c:

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

int main() {
    FILE *tacFile;
    char line[100], op[10], arg1[10], arg2[10], result[10];

    // Open TAC input file
    tacFile = fopen("tac.txt", "r");
    if (tacFile == NULL) {
        printf("Error: Cannot open TAC file\n");
        return 1;
    }

    printf("; 8086 Assembly code generated from TAC\n");
    printf("MOV AX, 0\n"); // initialize AX

    // Read TAC line by line
    while (fgets(line, sizeof(line), tacFile) != NULL) {
        if (sscanf(line, "%s = %s + %s", result, arg1, arg2) == 3) {
            printf("MOV AX, %s\n", arg1);
            printf("ADD AX, %s\n", arg2);
            printf("MOV %s, AX\n", result);
        } else if (sscanf(line, "%s = %s - %s", result, arg1, arg2) == 3) {
            printf("MOV AX, %s\n", arg1);
            printf("SUB AX, %s\n", arg2);
            printf("MOV %s, AX\n", result);
        } else if (sscanf(line, "%s = %s * %s", result, arg1, arg2) == 3) {
            printf("MOV AX, %s\n", arg1);
            printf("MUL %s\n", arg2);
            printf("MOV %s, AX\n", result);
        } else if (sscanf(line, "%s = %s / %s", result, arg1, arg2) == 3) {
            printf("MOV AX, %s\n", arg1);
            printf("DIV %s\n", arg2);
            printf("MOV %s, AX\n", result);
        } else if (sscanf(line, "%s = %s", result, arg1) == 2) {
            printf("MOV %s, %s\n", result, arg1);
        }
    }

    fclose(tacFile);
    return 0;
}
```

Tac.txt:

```
t1 = a + b
t2 = t1 * c
d = t2
```

Output:

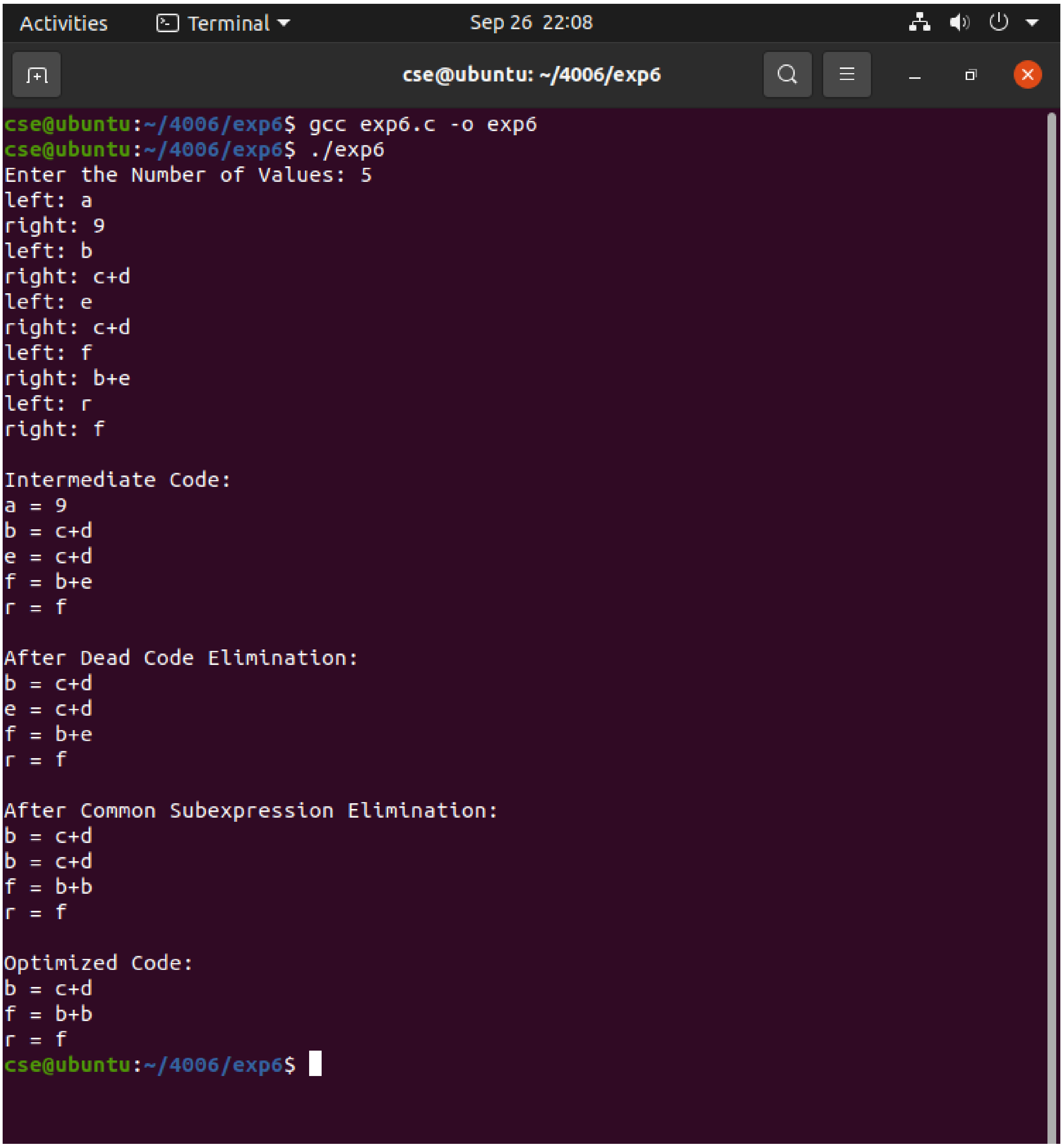


```
cse@ubuntu:~/4006/ex7$ gcc ex7.c -o ex7
cse@ubuntu:~/4006/ex7$ ./ex7
; 8086 Assembly code generated from TAC
MOV AX, 0
MOV AX, a
ADD AX, b
MOV t1, AX
MOV AX, t1
MUL c
MOV t2, AX
MOV d, t2
cse@ubuntu:~/4006/ex7$
```

Program:

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

struct op {
    char l;
    char r[20];
} op[10], pr[10];

int main() {
    int a, i, k, j, n, z = 0, m, q;
    char *p, *l;
    char temp, t;
    char *tem;

    printf("Enter the Number of Values: ");
    scanf("%d", &n);

    for(i = 0; i < n; i++) {
        printf("left: ");
        scanf(" %c", &op[i].l);
        printf("right: ");
        scanf("%s", op[i].r);
    }

    printf("\nIntermediate Code:\n");
    for(i = 0; i < n; i++) {
        printf("%c = %s\n", op[i].l, op[i].r);
    }

    // Dead Code Elimination
    for(i = 0; i < n-1; i++) {
        temp = op[i].l;
        for(j = 0; j < n; j++) {
            p = strchr(op[j].r, temp);
            if(p) {
                pr[z].l = op[i].l;
                strcpy(pr[z].r, op[i].r);
                z++;
            }
        }
    }
    pr[z].l = op[n-1].l;
    strcpy(pr[z].r, op[n-1].r);
    z++;

    printf("\nAfter Dead Code Elimination:\n");
    for(k = 0; k < z; k++) {
        printf("%c = %s\n", pr[k].l, pr[k].r);
    }

    // Common Subexpression Elimination
    for(m = 0; m < z; m++) {
        tem = pr[m].r;
        for(j = m+1; j < z; j++) {
            p = strstr(tem, pr[j].r);
            if(p) {
                t = pr[j].l;
                pr[j].l = pr[m].l;
                for(i = 0; i < z; i++) {
                    l = strchr(pr[i].r, t);
                    if(l) {
                        a = l - pr[i].r;
                        pr[i].r[a] = pr[m].l;
                    }
                }
            }
        }
    }

    printf("\nAfter Common Subexpression Elimination:\n");
    for(i = 0; i < z; i++) {
        printf("%c = %s\n", pr[i].l, pr[i].r);
    }

    // Remove duplicate expressions (simple version)
    for(i = 0; i < z; i++) {
        for(j = i+1; j < z; j++) {
            q = strcmp(pr[i].r, pr[j].r);
            if((pr[i].l == pr[j].l) && !q) {
                pr[i].l = '\0';
            }
        }
    }

    printf("\nOptimized Code:\n");
    for(i = 0; i < z; i++) {
        if(pr[i].l != '\0') {
            printf("%c = %s\n", pr[i].l, pr[i].r);
        }
    }

    return 0;
}
```

Output:



```
cse@ubuntu:~/4006/exp6$ gcc exp6.c -o exp6
cse@ubuntu:~/4006/exp6$ ./exp6
Enter the Number of Values: 5
left: a
right: 9
left: b
right: c+d
left: e
right: c+d
left: f
right: b+e
left: r
right: f

Intermediate Code:
a = 9
b = c+d
e = c+d
f = b+e
r = f

After Dead Code Elimination:
b = c+d
e = c+d
f = b+e
r = f

After Common Subexpression Elimination:
b = c+d
b = c+d
f = b+b
r = f

Optimized Code:
b = c+d
f = b+b
r = f
cse@ubuntu:~/4006/exp6$
```