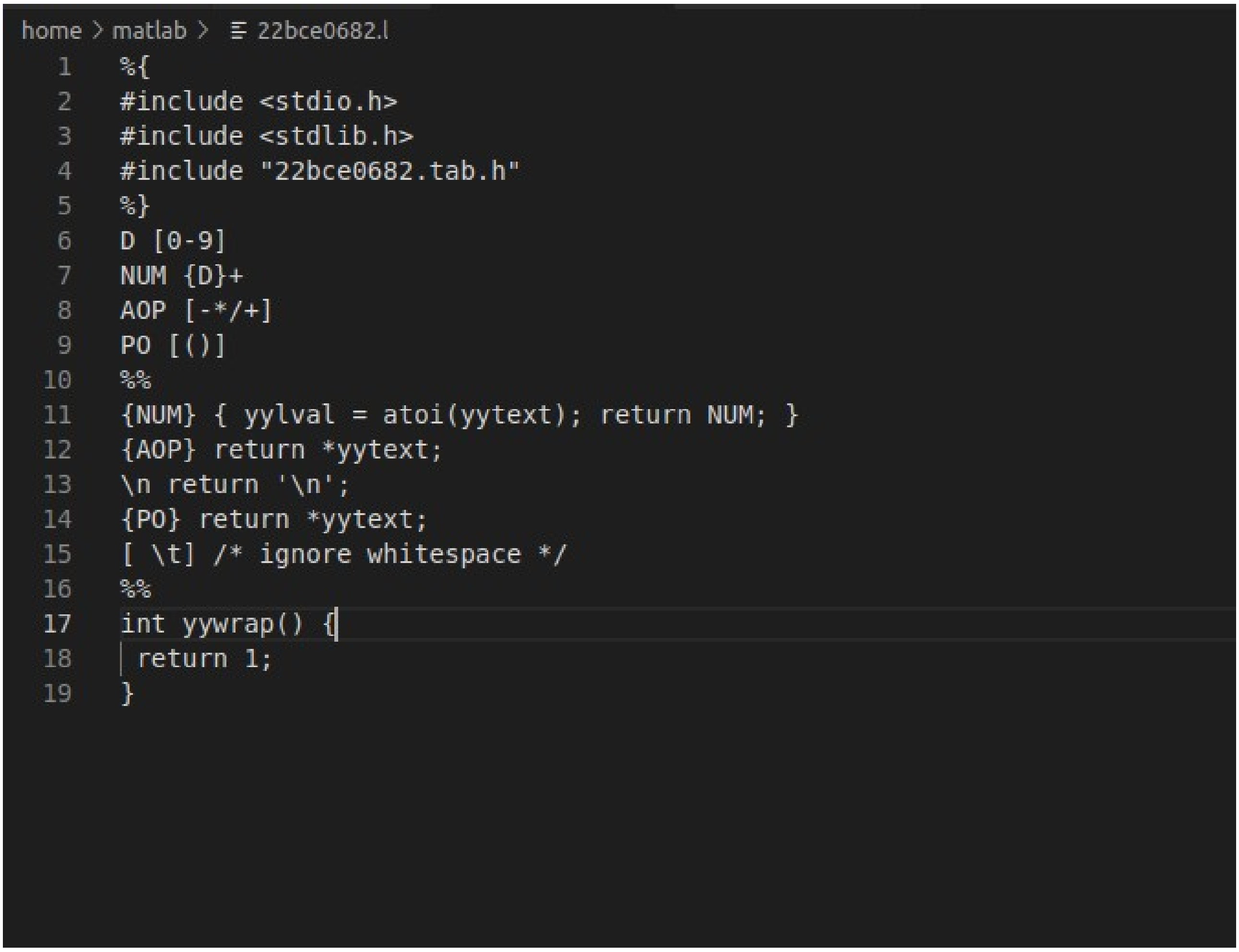
22BCE0682
Siddhant Bhagat

COMPILER-LAB

Q1) calculator

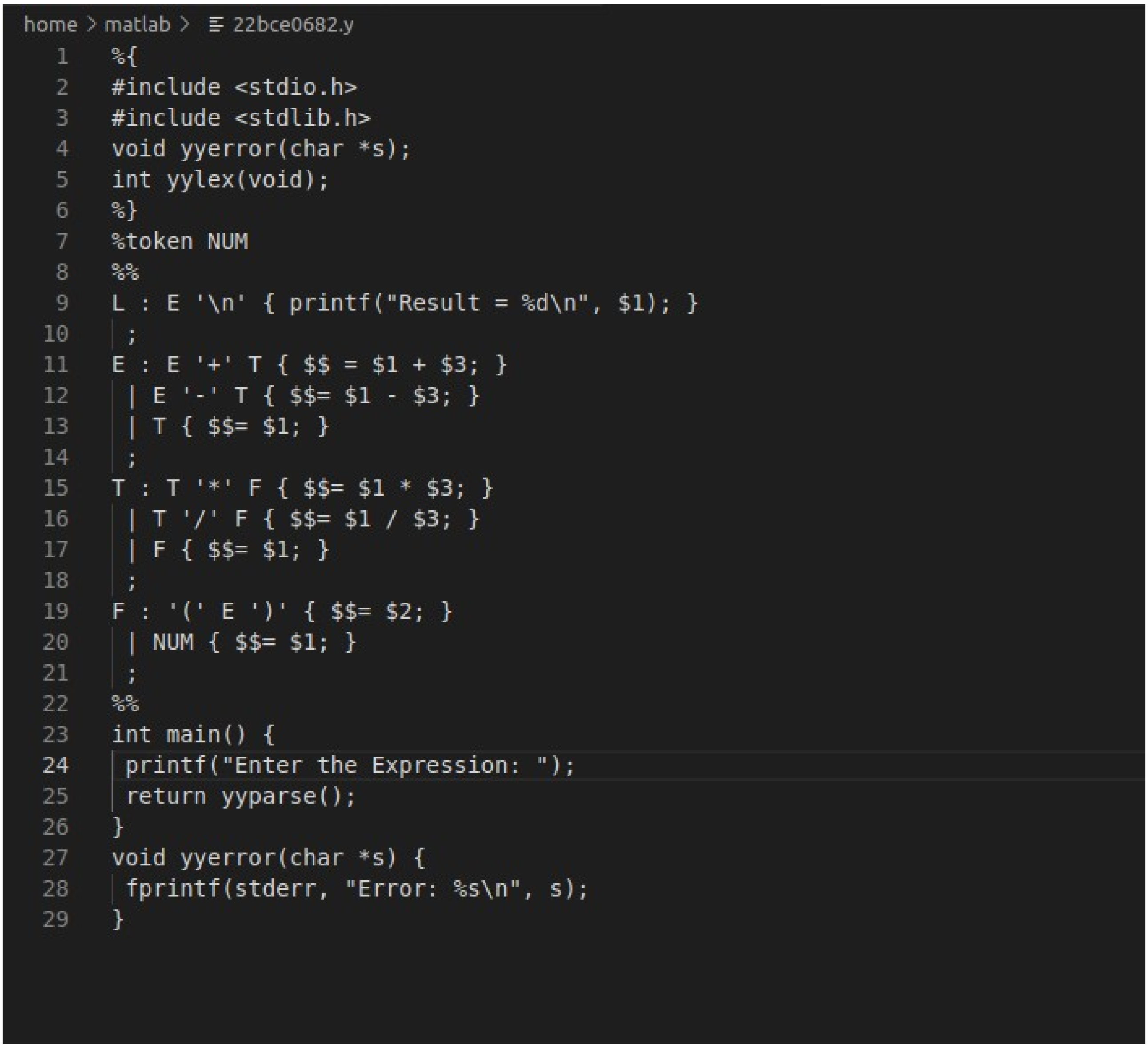


```
%{
#include <stdio.h>
#include <stdlib.h>
#include "22bce0682.tab.h"
%}
D [0-9]
NUM {D}+
AOP [-*/+]
PO [()]
%%
{NUM} { yylval = atoi(yytext); return NUM; }
{AOP} return *yytext;
\n return '\n';
{PO} return *yytext;
[ \t] /* ignore whitespace */
%%
int yywrap() {
 return 1;
}
```

22bce0682.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "22bce0682.tab.h"
%}

D [0-9]
NUM {D}+
AOP [-*/+]
PO [()]
%%
{NUM} { yylval = atoi(yytext); return NUM; }
{AOP} return *yytext;
\n return '\n';
```

```
{PO} return *yytext;
[ \t] /* for ignoring whitespace */
%%
int yywrap() {
 return 1;
}
```



home > matlab > 22bce0682.y

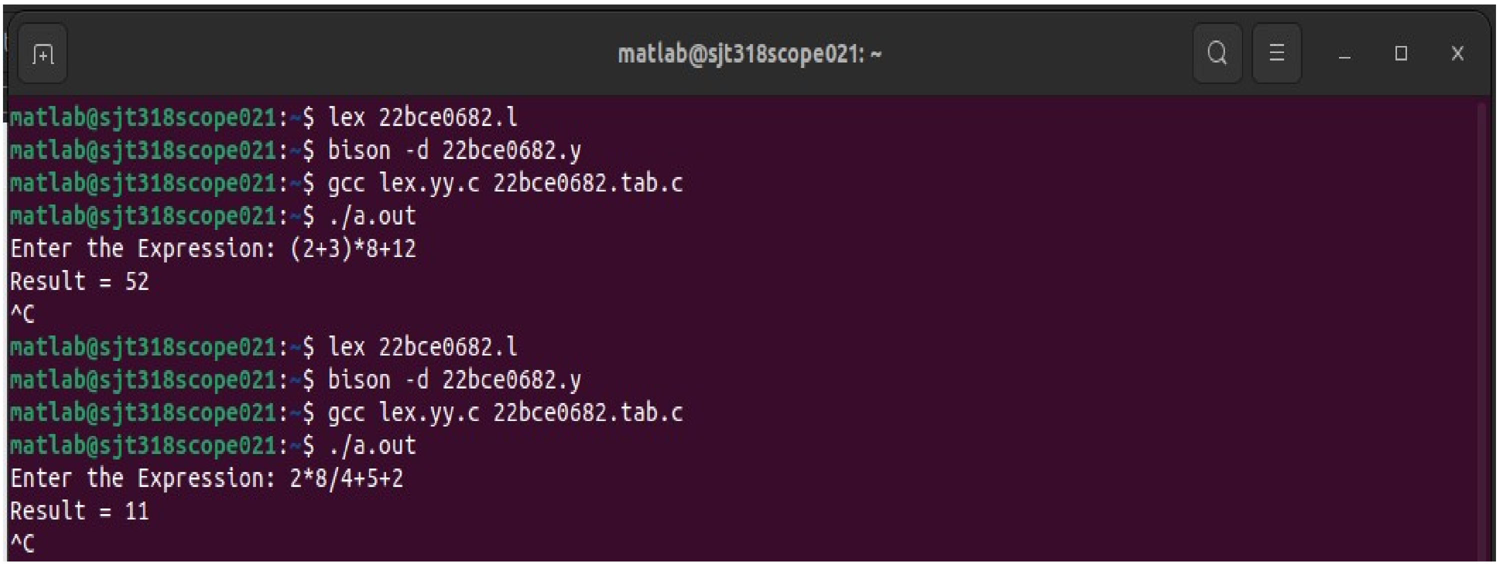
```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(char *s);
int yylex(void);
%}
%token NUM
%%
L : E '\n' { printf("Result = %d\n", $1); }
  ;
E : E '+' T { $$ = $1 + $3; }
  | E '-' T { $$= $1 - $3; }
  | T { $$= $1; }
  ;
T : T '*' F { $$= $1 * $3; }
  | T '/' F { $$= $1 / $3; }
  | F { $$= $1; }
  ;
F : '(' E ')' { $$= $2; }
  | NUM { $$= $1; }
  ;
%%
int main() {
 printf("Enter the Expression: ");
 return yyparse();
}
void yyerror(char *s) {
 fprintf(stderr, "Error: %s\n", s);
}
```

<u>22bce0682.y</u>

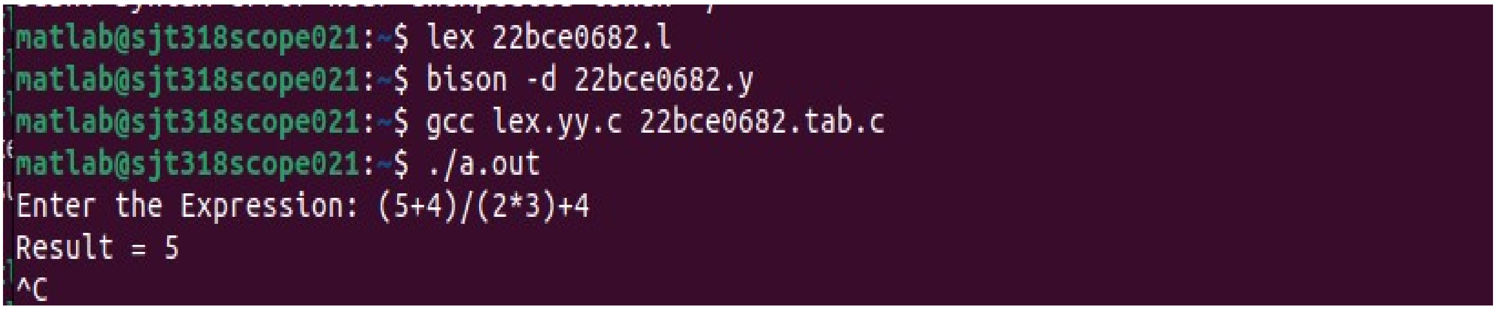
```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(char *s);
int yylex(void);
%}
%token NUM
%%
```

```
L : E '\n' { printf("Result = %d\n", $1); }
 ;
E : E '+' T { $$ = $1 + $3; }
 | E '-' T { $$= $1 - $3; }
 | T { $$= $1; }
 ;
T : T '*' F { $$= $1 * $3; }
 | T '/' F { $$= $1 / $3; }
 | F { $$= $1; }
 ;
F : '(' E ')' { $$= $2; }
 | NUM { $$= $1; }
 ;
%%
int main() {
 printf("Enter the Expression: ");
 return yyparse();
}
void yyerror(char *s) {
 fprintf(stderr, "Error: %s\n", s);
}
```

OUTPUT:



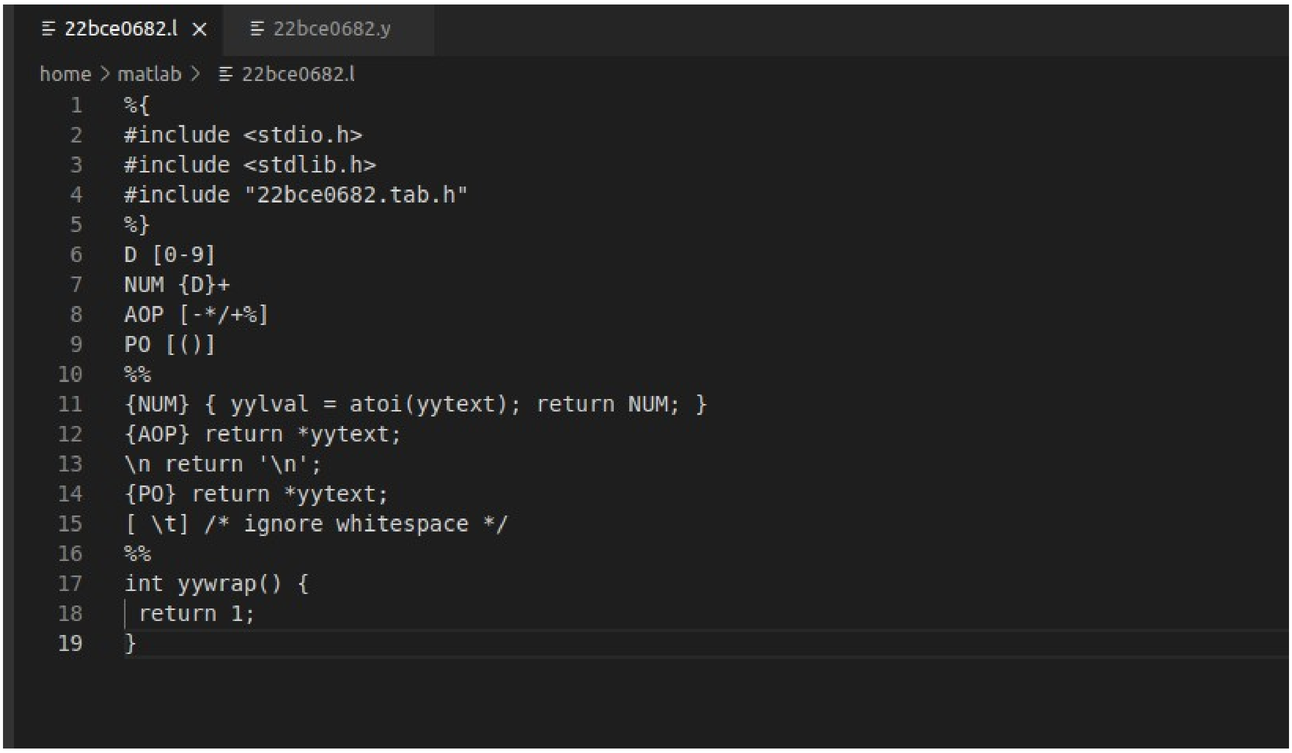
```
matlab@sjt318scope021:~$ lex 22bce0682.l
matlab@sjt318scope021:~$ bison -d 22bce0682.y
matlab@sjt318scope021:~$ gcc lex.yy.c 22bce0682.tab.c
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: (2+3)*8+12
Result = 52
^C
matlab@sjt318scope021:~$ lex 22bce0682.l
matlab@sjt318scope021:~$ bison -d 22bce0682.y
matlab@sjt318scope021:~$ gcc lex.yy.c 22bce0682.tab.c
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: 2*8/4+5+2
Result = 11
^C
```



```
matlab@sjt318scope021:~$ lex 22bce0682.l
matlab@sjt318scope021:~$ bison -d 22bce0682.y
matlab@sjt318scope021:~$ gcc lex.yy.c 22bce0682.tab.c
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: (5+4)/(2*3)+4
Result = 5
^C
```

22BCE0682
Siddhant Bhagat

calculator with %



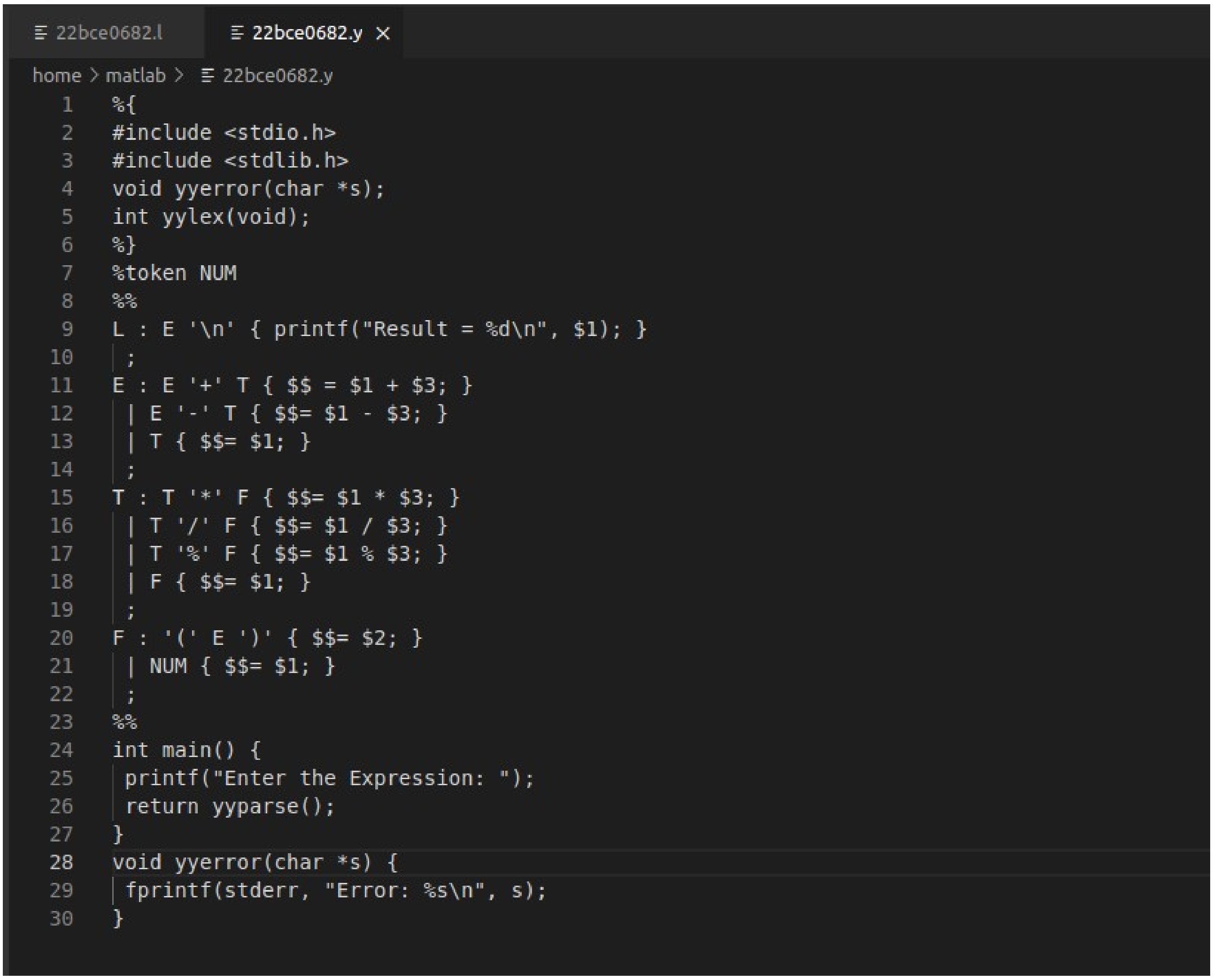
```
%{
#include <stdio.h>
#include <stdlib.h>
#include "22bce0682.tab.h"
%}
D [0-9]
NUM {D}+
AOP [-*/+%]
PO [()]
%%
{NUM} { yylval = atoi(yytext); return NUM; }
{AOP} return *yytext;
\n return '\n';
{PO} return *yytext;
[ \t] /* ignore whitespace */
%%
int yywrap() {
 return 1;
}
```

<u>22bce0682.l</u>

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "22bce0682.tab.h"
%}
D [0-9]
NUM {D}+
AOP [-*/+%]
PO [()]
%%
{NUM} { yylval = atoi(yyte%{
#include <stdio.h>
#include <stdlib.h>
#include "22bce0682.tab.h"
%}
D [0-9]
NUM {D}+
AOP [-*/+%]
PO [()]
%%
{NUM} { yylval = atoi(yytext); return NUM; }
{AOP} return *yytext;
\n return '\n';
{PO} return *yytext;
```

22BCE0682
Siddhant Bhagat

```
[ \t] /* ignore whitespace */
%%
int yywrap() {
 return 1;
} xt); return NUM; }
{AOP} return *yytext;
\n return '\n';
{PO} return *yytext;
[ \t] /* ignore whitespace */
%%
int yywrap() {
 return 1;
}
```

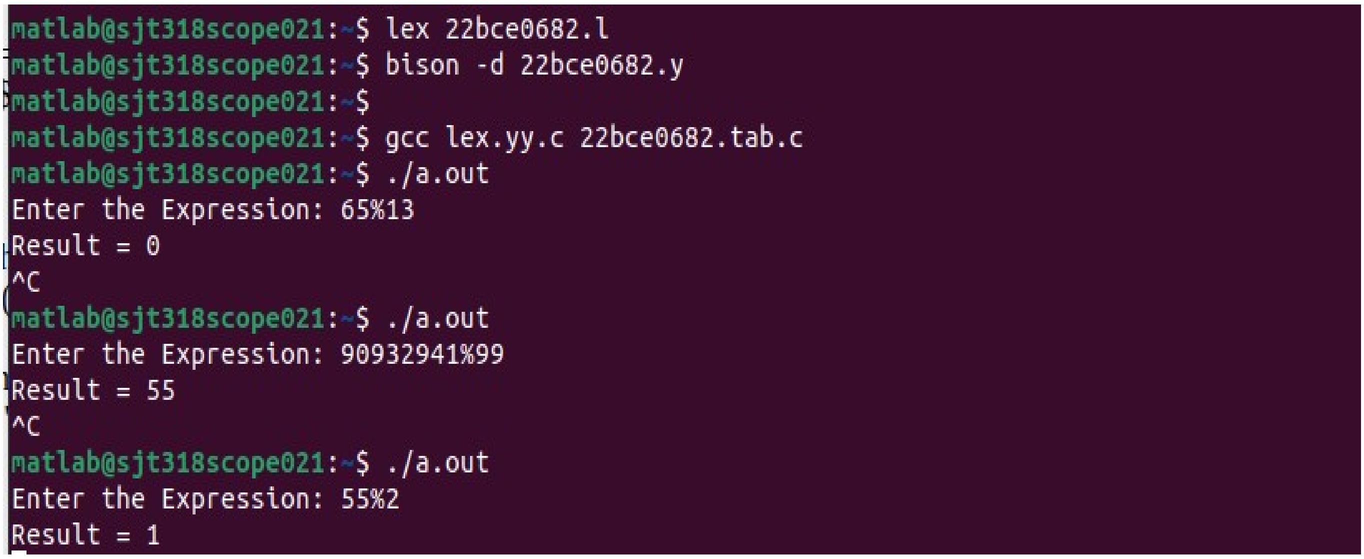


```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(char *s);
int yylex(void);
%}
%token NUM
%%
L : E '\n' { printf("Result = %d\n", $1); }
 ;
E : E '+' T { $$ = $1 + $3; }
 | E '-' T { $$= $1 - $3; }
 | T { $$= $1; }
 ;
T : T '*' F { $$= $1 * $3; }
 | T '/' F { $$= $1 / $3; }
 | T '%' F { $$= $1 % $3; }
 | F { $$= $1; }
 ;
F : '(' E ')' { $$= $2; }
 | NUM { $$= $1; }
 ;
%%
int main() {
 printf("Enter the Expression: ");
 return yyparse();
}
void yyerror(char *s) {
 fprintf(stderr, "Error: %s\n", s);
}
```

22bce0682.y

```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(char *s);
int yylex(void);
%}
%token NUM
%%
L : E '\n' { printf("Result = %d\n", $1); }
 ;
E : E '+' T { $$ = $1 + $3; }
 | E '-' T { $$= $1 - $3; }
 | T { $$= $1; }
 ;
T : T '*' F { $$= $1 * $3; }
 | T '/' F { $$= $1 / $3; }
 | T '%' F { $$= $1 % $3; }
 | F { $$= $1; }
 ;
F : '(' E ')' { $$= $2; }
 | NUM { $$= $1; }
 ;
%%
int main() {
 printf("Enter the Expression: ");
 return yyparse();
}
void yyerror(char *s) {
 fprintf(stderr, "Error: %s\n", s);
}
```

OUTPUT:



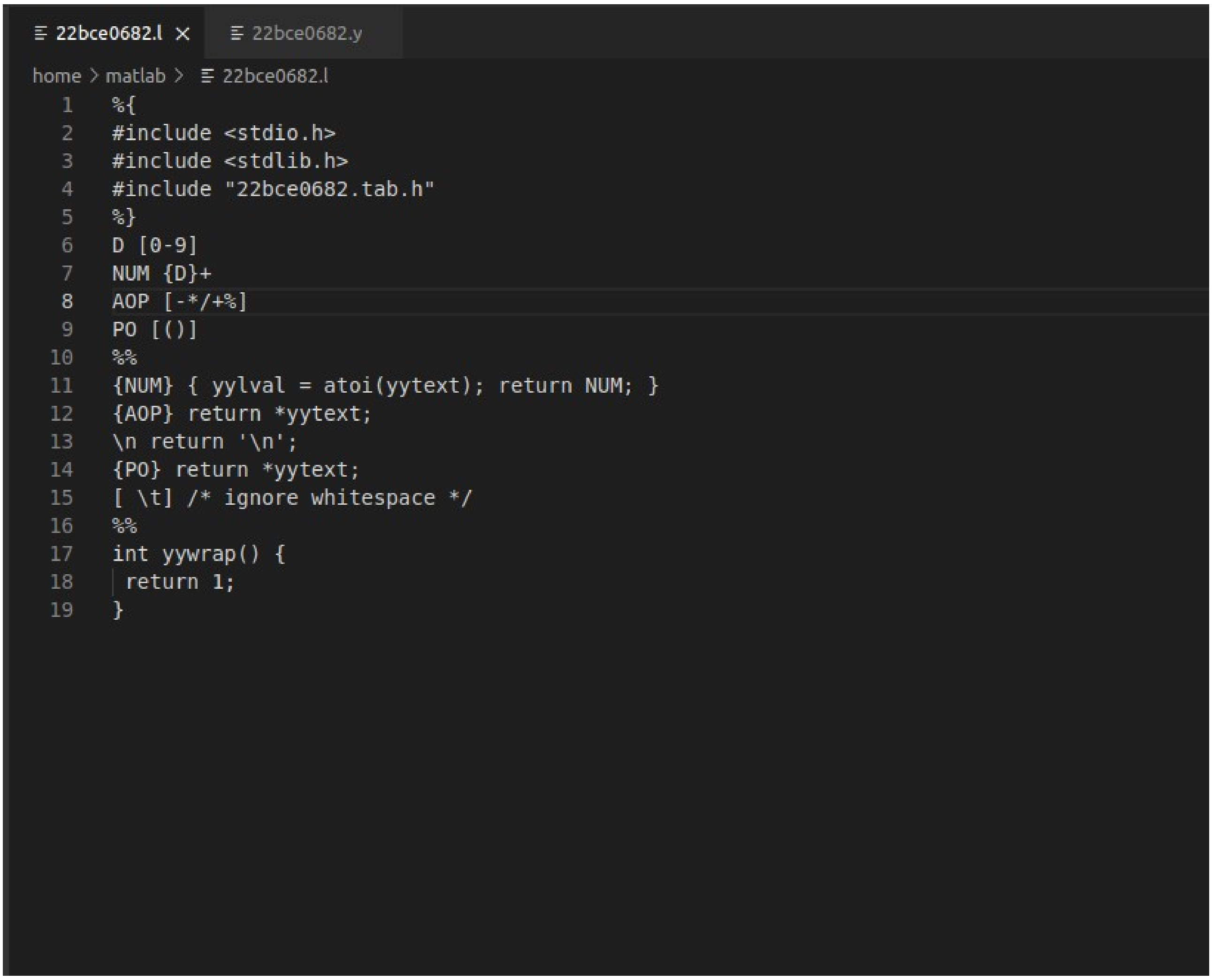
```
matlab@sjt318scope021:~$ lex 22bce0682.l
matlab@sjt318scope021:~$ bison -d 22bce0682.y
matlab@sjt318scope021:~$
matlab@sjt318scope021:~$ gcc lex.yy.c 22bce0682.tab.c
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: 65%13
Result = 0
^C
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: 90932941%99
Result = 55
^C
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: 55%2
Result = 1
```

Q2) Increment Decrement

22bce0682.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "22bce0682.tab.h"
%}
D [0-9]
NUM {D}+
AOP [-*/+%]
PO [()]
%%
{NUM} { yylval = atoi(yytext); return NUM; }
{AOP} return *yytext;
\n return '\n';
{PO} return *yytext;
[ \t] /* ignore whitespace */
%%
int yywrap() {
 return 1;
}
```

22BCE0682
Siddhant Bhagat



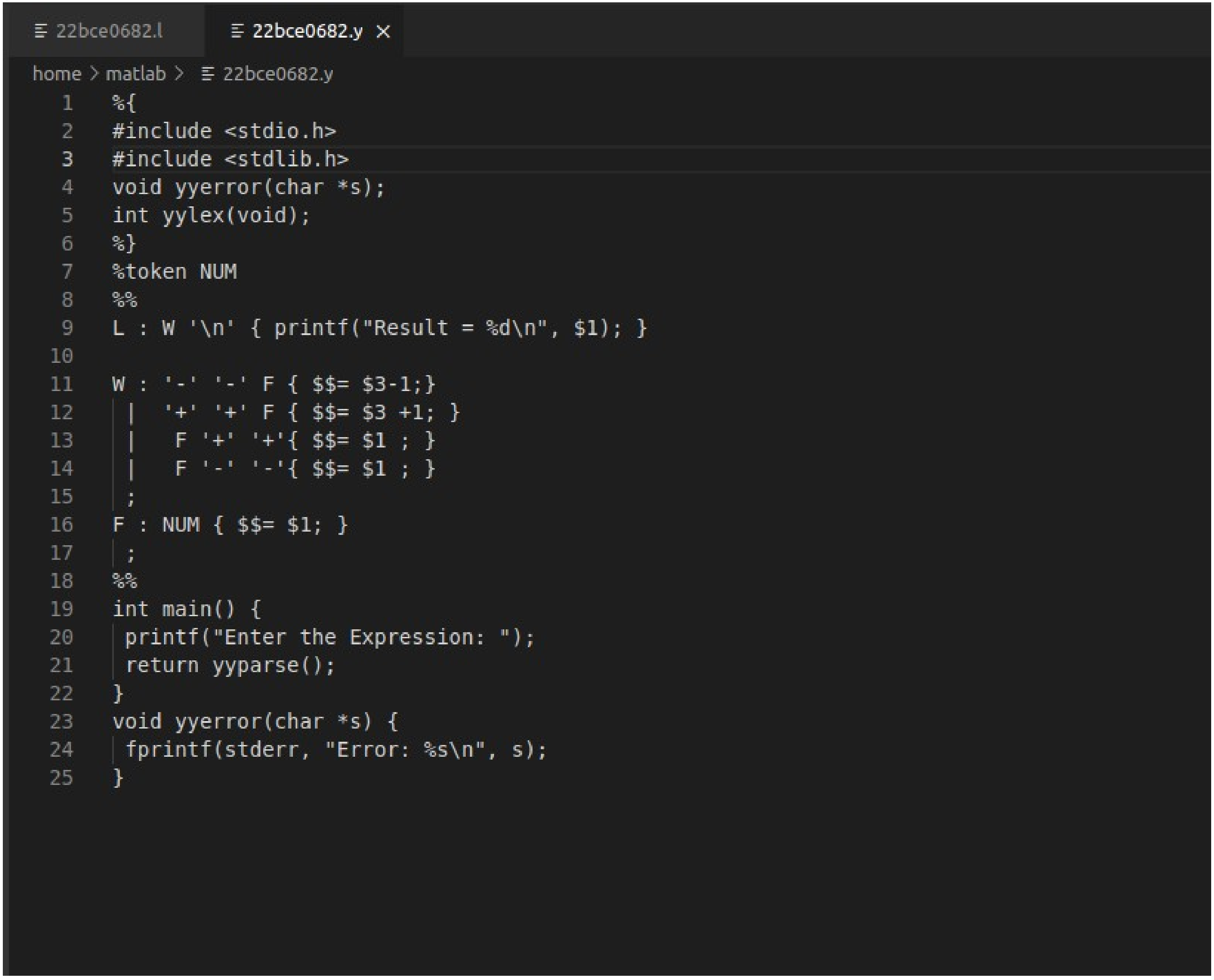
```
%{
#include <stdio.h>
#include <stdlib.h>
#include "22bce0682.tab.h"
%}
D [0-9]
NUM {D}+
AOP [-*/+%]
PO [()]
%%
{NUM} { yylval = atoi(yytext); return NUM; }
{AOP} return *yytext;
\n return '\n';
{PO} return *yytext;
[ \t] /* ignore whitespace */
%%
int yywrap() {
  return 1;
}
```

22bce0682.y

```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(char *s);
int yylex(void);
%}
%token NUM
%%
L : W '\n' { printf("Result = %d\n", $1); }

W : '-' '-' F { $$= $3-1;}
 |  '+' '+' F { $$= $3 +1; }
 |   F '+' '+'{ $$= $1 ; }
 |   F '-' '-'{ $$= $1 ; }
 ;
F : NUM { $$= $1; }
 ;
```

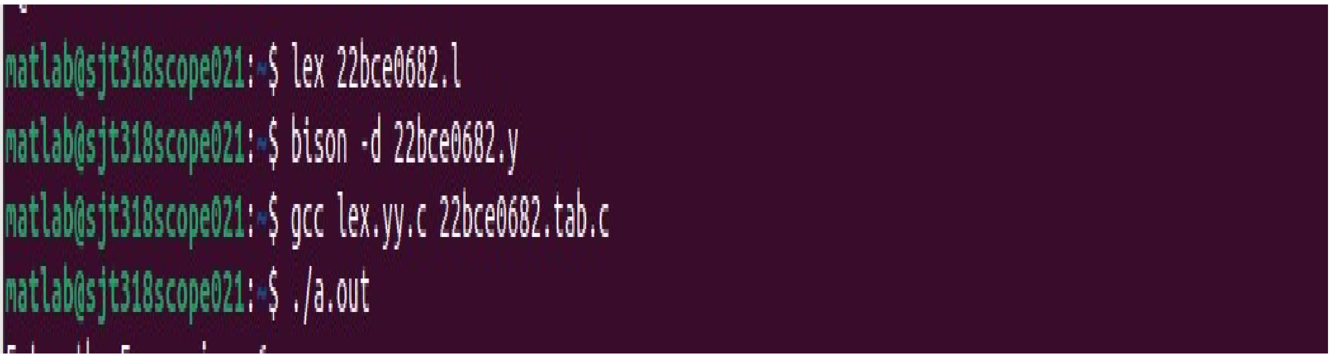
22BCE0682
Siddhant Bhagat

```
%%
int main() {
 printf("Enter the Expression: ");
 return yyparse();
}
void yyerror(char *s) {
 fprintf(stderr, "Error: %s\n", s);
}
```

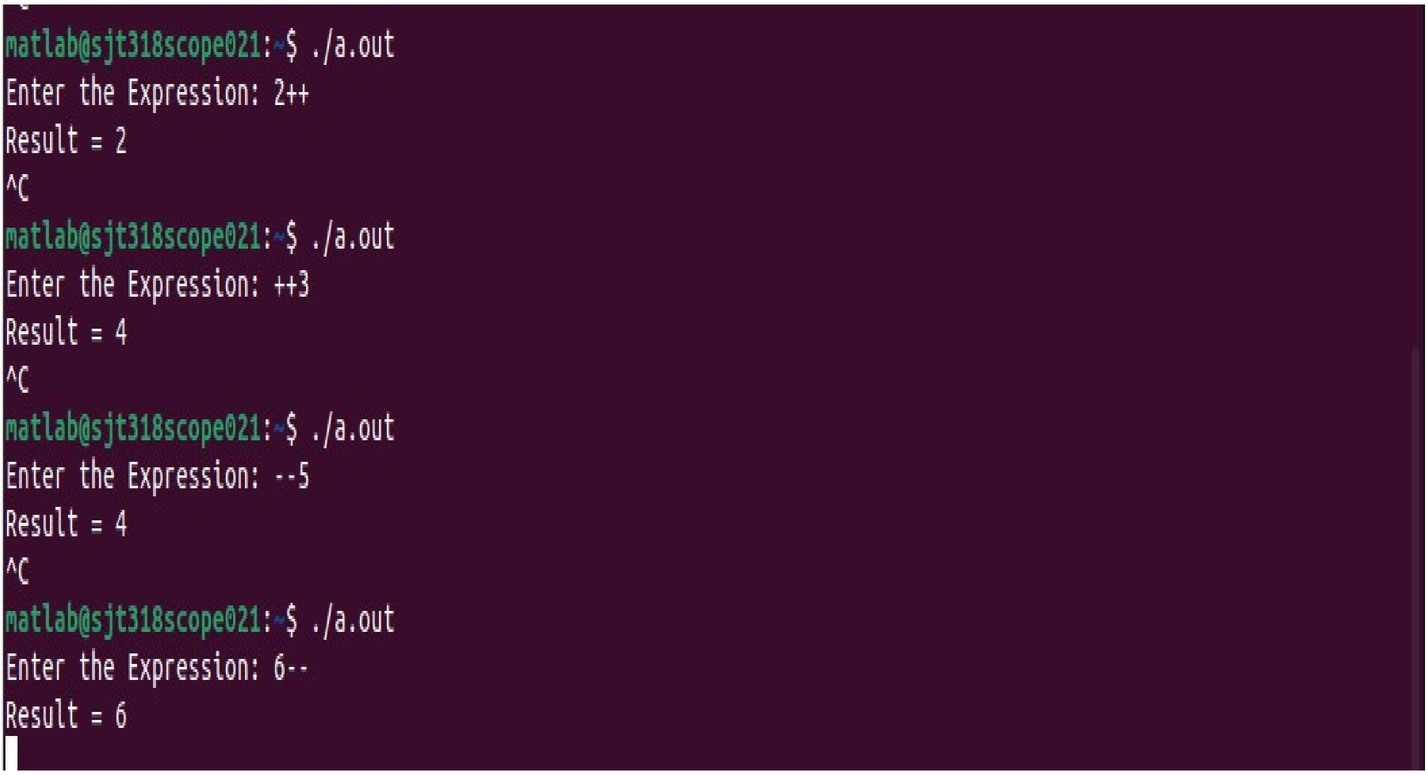


```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(char *s);
int yylex(void);
%}
%token NUM
%%
L : W '\n' { printf("Result = %d\n", $1); }

W : '-' '-' F { $$= $3-1;}
  |  '+' '+' F { $$= $3 +1; }
  |   F '+' '+'{ $$= $1 ; }
  |   F '-' '-'{ $$= $1 ; }
  ;
F : NUM { $$= $1; }
  ;
%%
int main() {
 printf("Enter the Expression: ");
 return yyparse();
}
void yyerror(char *s) {
 fprintf(stderr, "Error: %s\n", s);
}
```

OUTPUT:



```
matlab@sjt318scope021:~$ lex 22bce0682.l
matlab@sjt318scope021:~$ bison -d 22bce0682.y
matlab@sjt318scope021:~$ gcc lex.yy.c 22bce0682.tab.c
matlab@sjt318scope021:~$ ./a.out
```



```
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: 2++
Result = 2
^C
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: ++3
Result = 4
^C
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: --5
Result = 4
^C
matlab@sjt318scope021:~$ ./a.out
Enter the Expression: 6--
Result = 6
```

Q1)

PREDICTIVE PARSER

CODE:

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

int main() {

char fin[10][20], st[10][20], ft[20][20], fol[20][20];

char terminals[20][10], non_terminals[20];

int a = 0, e, i, t, b, c, n, k, l = 0, j, s, m, p;

int num_terminals, num_non_terminals;

printf("Enter the number of non-terminals:\n");

scanf("%d", &num_non_terminals);

printf("Enter the non-terminals (single characters):\n");

for(i = 0; i < num_non_terminals; i++) {

scanf(" %c", &non_terminals[i]);

}

non_terminals[num_non_terminals] = '\0';

printf("Enter the number of terminals:\n");

scanf("%d", &num_terminals);

printf("Enter the terminals (use # for epsilon):\n");

for(i = 0; i < num_terminals; i++) {

scanf("%s", terminals[i]);

}

printf("Enter the number of productions:\n");

scanf("%d", &n);


printf("Enter the productions in the grammar (use # for epsilon):\n");
```

```
for(i = 0; i < n; i++)

scanf("%s", st[i]);

for(i = 0; i < n; i++)

fol[i][0] = '\0';
for(s = 0; s < n; s++) {

for(i = 0; i < n; i++) {

j = 3;

l = 0;

a = 0;

l1:
int is_terminal = 0;

for(t = 0; t < num_terminals; t++) {

if(strncmp(&st[i][j], terminals[t], strlen(terminals[t])) == 0) {

is_terminal = 1;

break;

}

}

if(is_terminal) {

for(m = 0; m < l; m++) {

if(strncmp(&ft[i][m], terminals[t], strlen(terminals[t])) == 0)

goto s1;

}

strcpy(&ft[i][l], terminals[t]);

l += strlen(terminals[t]);

s1:

j += strlen(terminals[t]);
} else {

if(s > 0) {

while(st[i][j] != st[a][0]) {

a++;

}

b = 0;

while(ft[a][b] != '\0') {

for(m = 0; m < l; m++) {

if(ft[i][m] == ft[a][b])

goto s2;

}

ft[i][l] = ft[a][b];

l++;

s2:

b++;

}

}

}

while(st[i][j] != '\0') {

if(st[i][j] == '|') {

j++;

goto l1;

}

j++;

}
```

```
ft[i][l] = '\0';
}
}
printf("FIRST sets:\n");
for(i = 0; i < n; i++) {
printf("FIRST[%c] = ", st[i][0]);
for(j = 0; ft[i][j] != '\0'; j++) {
printf("%c", ft[i][j]);
if(ft[i][j+1] != '\0')
printf(", ");
}
printf("\n");
}
fol[0][0] = '$';
for(i = 0; i < n; i++) {
k = 0;
j = 3;
if(i == 0)
l = 1;
else
l = 0;
k1:
while((st[i][0] != st[k][j]) && (k < n)) {
if(st[k][j] == '\0') {
```

```
k++;

j = 2;

}

j++;

}

j++;

if(st[i][0] == st[k][j-1]) {

if((st[k][j] != '|') && (st[k][j] != '\0')) {

a = 0;

if(!((st[k][j] >= 'A') && (st[k][j] <= 'Z'))) {

for(m = 0; m < l; m++) {

if(fol[i][m] == st[k][j])

goto q3;

}

fol[i][l] = st[k][j];

l++;

q3:

p++;

} else {

while(st[k][j] != st[a][0]) {

a++;

}

p = 0;

while(ft[a][p] != '\0') {

if(ft[a][p] != '#') {
```

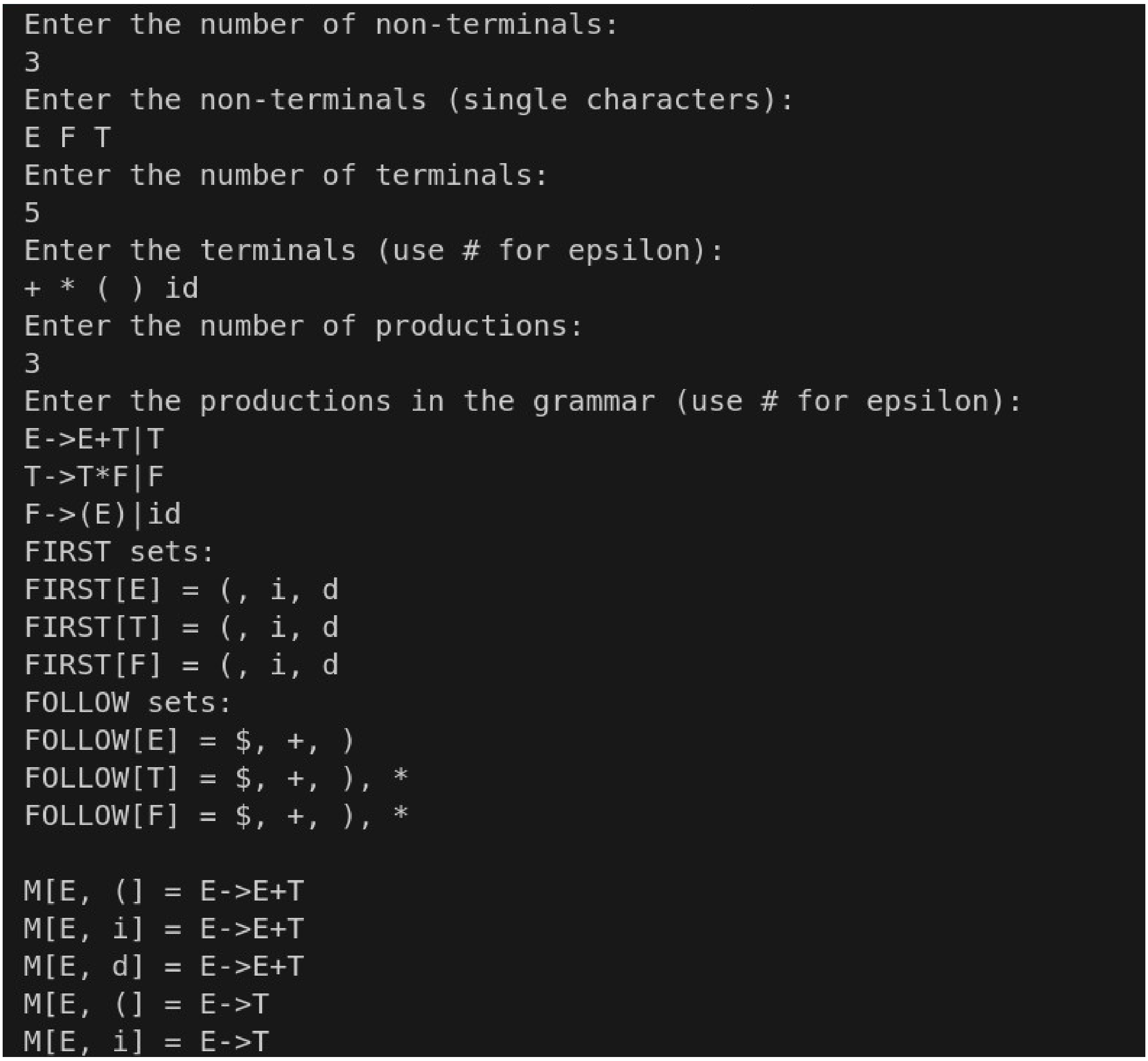
```
for(m = 0; m < l; m++) {

if(fol[i][m] == ft[a][p])

goto q2;

}

fol[i][l] = ft[a][p];

l++;

} else {

e = 1;

}

q2:

p++;

}

if(e == 1) {

e = 0;

goto a1;

}

}

} else {

a1:

c = 0;

a = 0;

while(st[k][0] != st[a][0]) {

a++;

}
```

```
while((fol[a][c] != '\0') && (st[a][0] != st[i][0])) {

for(m = 0; m < l; m++) {

if(fol[i][m] == fol[a][c])

goto q1;

}

fol[i][l] = fol[a][c];

l++;

q1:

c++;

}

}

goto k1;

}

fol[i][l] = '\0';

}

printf("FOLLOW sets:\n");

for(i = 0; i < n; i++) {

printf("FOLLOW[%c] = ", st[i][0]);

for(j = 0; fol[i][j] != '\0'; j++) {

printf("%c", fol[i][j]);

if(fol[i][j+1] != '\0')

printf(", ");

}

printf("\n");

}
```

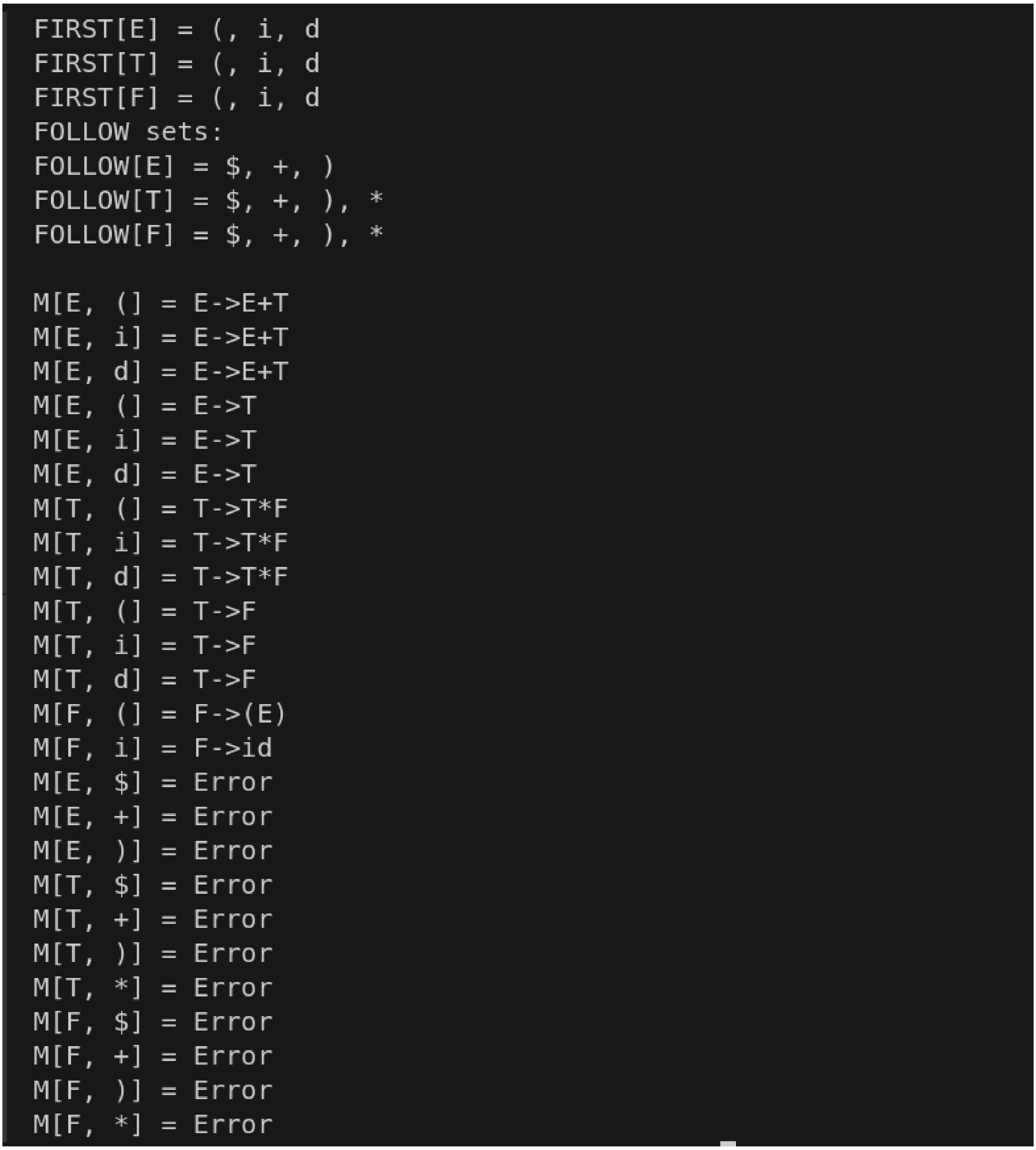
```
printf("\n");
s = 0;
for(i = 0; i < n; i++) {
j = 3;
while(st[i][j] != '\0') {
if((st[i][j-1] == '|') || (j == 3)) {
for(p = 0; p <= 2; p++) {
fin[s][p] = st[i][p];
}
t = j;
for(p = 3; ((st[i][j] != '|') && (st[i][j] != '\0')); p++) {
fin[s][p] = st[i][j];
j++;
}
fin[s][p] = '\0';

if(st[i][t] == '#') {
b = 0;
a = 0;
while(st[a][0] != st[i][0]) {
a++;
}
while(fol[a][b] != '\0') {
printf("M[%c, %c] = %s\n", st[i][0], fol[a][b], fin[s]);
b++;
}
} else if(!((st[i][t] >= 'A') && (st[i][t] <= 'Z'))) {
printf("M[%c, %c] = %s\n", st[i][0], st[i][t], fin[s]);
} else {
b = 0;
a = 0;
while(st[a][0] != st[i][3]) {
a++;
}
while(ft[a][b] != '\0') {
printf("M[%c, %c] = %s\n", st[i][0], ft[a][b], fin[s]);
b++;
}
}
s++;
}
if(st[i][j] == '|') {
j++;
}
}
}
for(i = 0; i < n; i++) {
for(j = 0; fol[i][j] != '\0'; j++) {
int found = 0;
```

```
for(s = 0; s < n; s++) {

if(fin[s][0] == st[i][0] && fin[s][2] == fol[i][j]) {

found = 1;

break;

}

}

if(!found) {

printf("M[%c, %c] = Error\n", st[i][0], fol[i][j]);

}

}

}

return 0;

}
```

OUTPUT:
TEST-CASE 1:

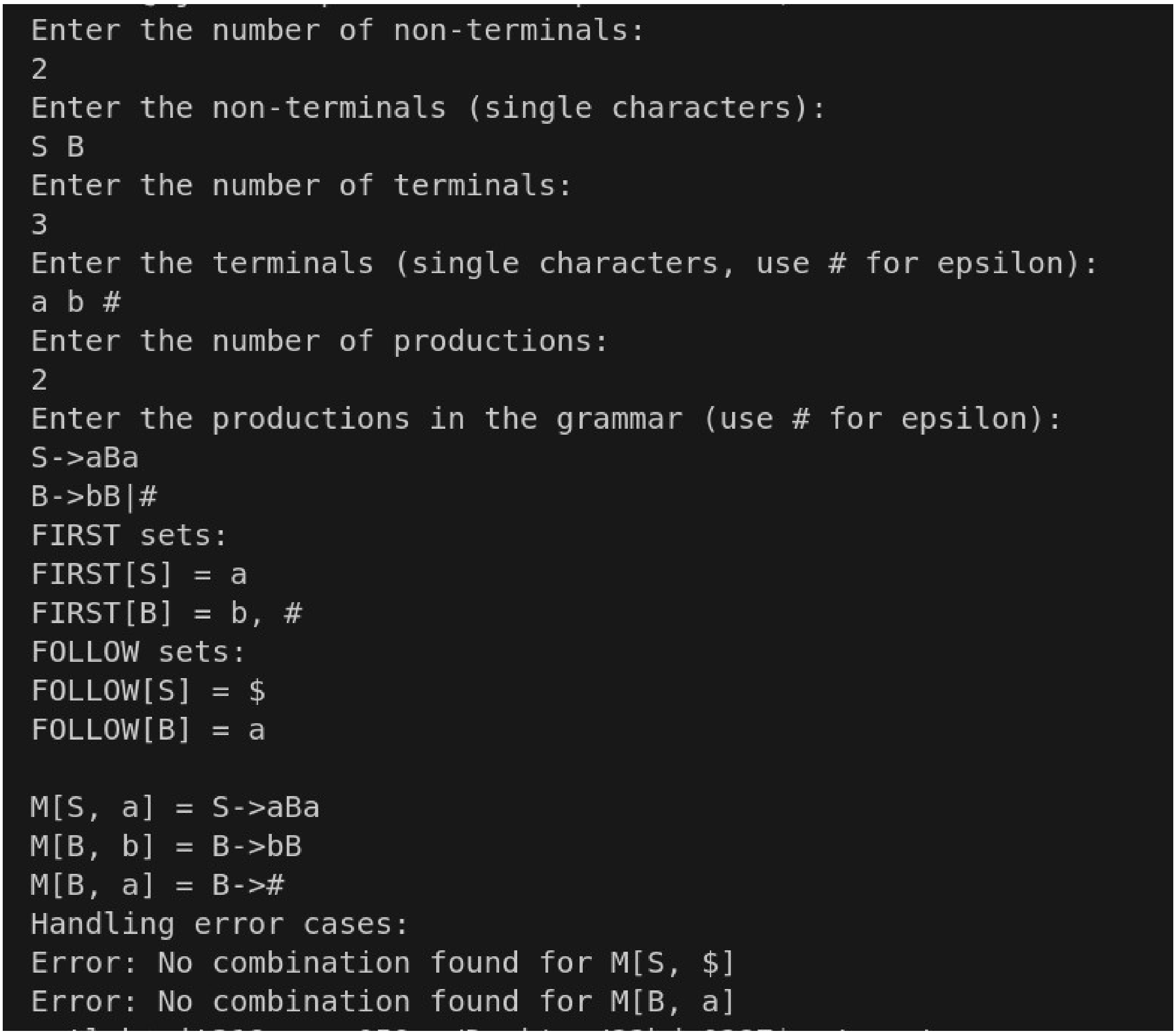


```
Enter the number of non-terminals:
3
Enter the non-terminals (single characters):
E F T
Enter the number of terminals:
5
Enter the terminals (use # for epsilon):
+ * ( ) id
Enter the number of productions:
3
Enter the productions in the grammar (use # for epsilon):
E->E+T|T
T->T*F|F
F->(E)|id
FIRST sets:
FIRST[E] = (, i, d
FIRST[T] = (, i, d
FIRST[F] = (, i, d
FOLLOW sets:
FOLLOW[E] = $, +, )
FOLLOW[T] = $, +, ), *
FOLLOW[F] = $, +, ), *

M[E, (] = E->E+T
M[E, i] = E->E+T
M[E, d] = E->E+T
M[E, (] = E->T
M[E, i] = E->T
```



```
FIRST[E] = (, i, d
FIRST[T] = (, i, d
FIRST[F] = (, i, d
FOLLOW sets:
FOLLOW[E] = $, +, )
FOLLOW[T] = $, +, ), *
FOLLOW[F] = $, +, ), *

M[E, (] = E->E+T
M[E, i] = E->E+T
M[E, d] = E->E+T
M[E, (] = E->T
M[E, i] = E->T
M[E, d] = E->T
M[T, (] = T->T*F
M[T, i] = T->T*F
M[T, d] = T->T*F
M[T, (] = T->F
M[T, i] = T->F
M[T, d] = T->F
M[F, (] = F->(E)
M[F, i] = F->id
M[E, $] = Error
M[E, +] = Error
M[E, )] = Error
M[T, $] = Error
M[T, +] = Error
M[T, )] = Error
M[T, *] = Error
M[F, $] = Error
M[F, +] = Error
M[F, )] = Error
M[F, *] = Error
```

TEST-CASE 2:



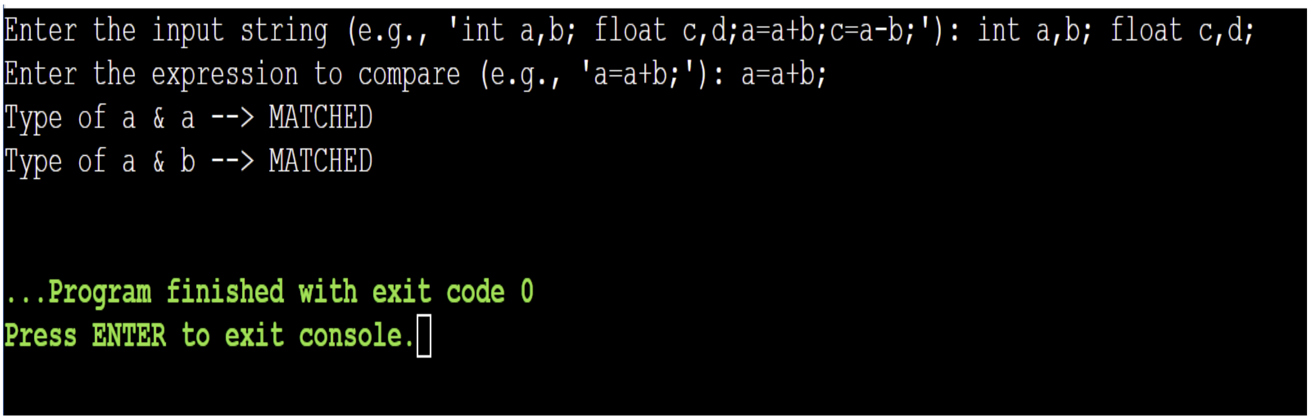
```
Enter the number of non-terminals:
2
Enter the non-terminals (single characters):
S B
Enter the number of terminals:
3
Enter the terminals (single characters, use # for epsilon):
a b #
Enter the number of productions:
2
Enter the productions in the grammar (use # for epsilon):
S->aBa
B->bB|#
FIRST sets:
FIRST[S] = a
FIRST[B] = b, #
FOLLOW sets:
FOLLOW[S] = $
FOLLOW[B] = a

M[S, a] = S->aBa
M[B, b] = B->bB
M[B, a] = B->#
Handling error cases:
Error: No combination found for M[S, $]
Error: No combination found for M[B, a]
```

Q1) CODE

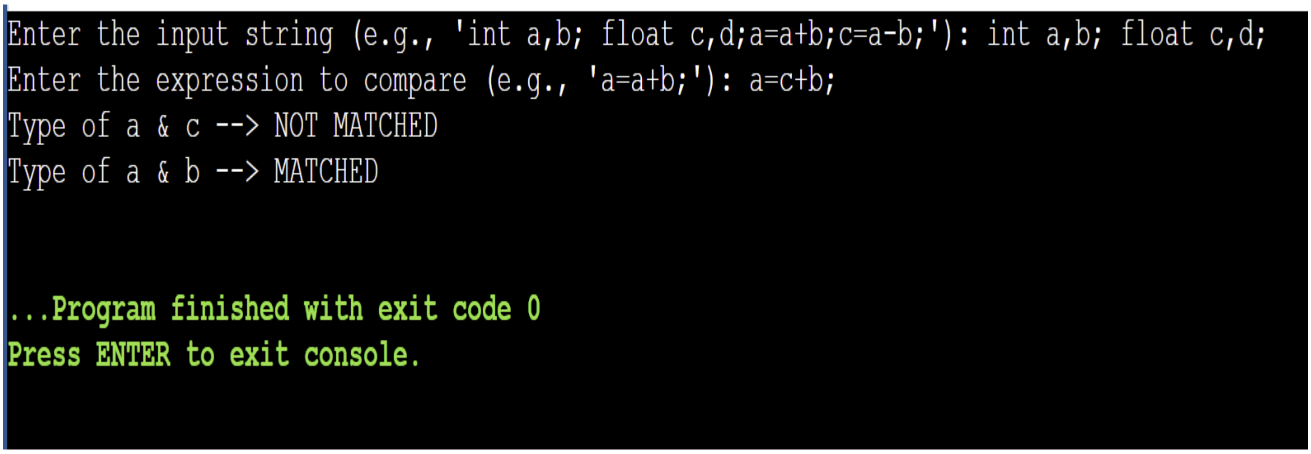
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

struct Node {
    char var[20];
    char type[10];
    struct Node* next;
};

struct Node* typeMap = NULL;

void insert(char* var, char* type) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->var, var);
    strcpy(newNode->type, type);
    newNode->next = typeMap;
    typeMap = newNode;
}

char* getType(const char* var) {
    struct Node* current = typeMap;
    while (current != NULL) {
        if (strcmp(current->var, var) == 0) {
            return current->type;
        }
        current = current->next;
    }
    return "undefined";
}

bool typeMatched(const char* type1, const char* type2) {
    return strcmp(type1, type2) == 0;
}

int main() {
    char input[200], expr[200];
    char token[20];
    char* delim = " ,;=";

    printf("Enter the input string (e.g., 'int a,b; float c,d;a=a+b;c=a-b;'): ");
    fgets(input, sizeof(input), stdin);



    size_t len = strlen(input);
    if (len > 0 && input[len - 1] == '\n') {
        input[len - 1] = '\0';
    }

    char* ptr = strtok(input, delim);
    while (ptr != NULL) {
        if (strcmp(ptr, "int") == 0 || strcmp(ptr, "float") == 0 || strcmp(ptr, "char") == 0) {
            char type[10];
            strcpy(type, ptr);
            ptr = strtok(NULL, delim);
            while (ptr != NULL && strcmp(ptr, "int") != 0 && strcmp(ptr, "float") != 0 && strcmp(ptr,
"char") != 0) {
```

```
                insert(ptr, type);
                ptr = strtok(NULL, delim);
            }
        } else {
            ptr = strtok(NULL, delim);
        }
    }

    printf("Enter the expression to compare (e.g., 'a=a+b;'): ");
    fgets(expr, sizeof(expr), stdin);

    len = strlen(expr);
    if (len > 0 && expr[len - 1] == '\n') {
        expr[len - 1] = '\0';
    }

    char* lhs = strtok(expr, "=");
    char* rhs = strtok(NULL, ";");

    char* rhs1 = strtok(rhs, "+");
    char* rhs2 = strtok(NULL, "+");

    char* type_lhs = getType(lhs);
    char* type_rhs1 = getType(rhs1);
    char* type_rhs2 = rhs2 ? getType(rhs2) : NULL;

    printf("Type of %s & %s --> ", lhs, rhs1);
    if (typeMatched(type_lhs, type_rhs1))
        printf("MATCHED\n");
    else
        printf("NOT MATCHED\n");
    if (rhs2 != NULL) {
        printf("Type of %s & %s --> ", lhs, rhs2);
        if (typeMatched(type_lhs, type_rhs2))
            printf("MATCHED\n");
        else
            printf("NOT MATCHED\n");
    }

    return 0;
}
```

OUTPUT



```
Enter the input string (e.g., 'int a,b; float c,d;a=a+b;c=a-b;'): int a,b; float c,d;
Enter the expression to compare (e.g., 'a=a+b;'): a=a+b;
Type of a & a --> MATCHED
Type of a & b --> MATCHED


...Program finished with exit code 0
Press ENTER to exit console.
```



```
Enter the input string (e.g., 'int a,b; float c,d;a=a+b;c=a-b;'): int a,b; float c,d;
Enter the expression to compare (e.g., 'a=a+b;'): a=c+b;
Type of a & c --> NOT MATCHED
Type of a & b --> MATCHED


...Program finished with exit code 0
Press ENTER to exit console.
```