Experiment No. 4

Lex Program (lex.l):

```
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+  { yylval = atoi(yytext); return NUM; }
"="     { return '='; }
"+"     { return '+'; }
\n      { return 0; }
.       { return yytext[0]; }
%%
int yywrap() {
   return 1;
}
```

Yacc Program (x1.y):

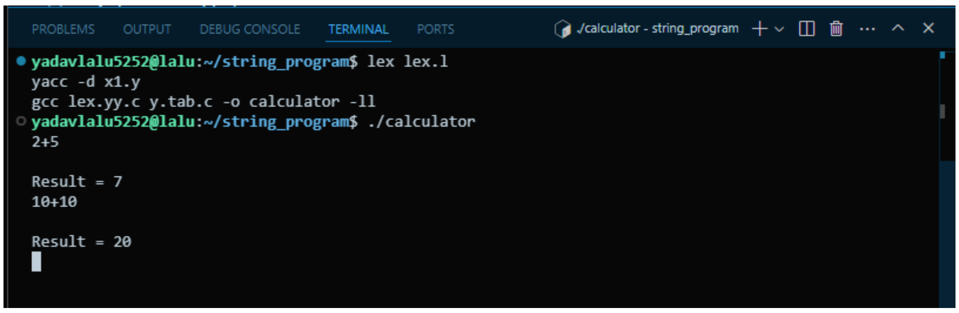
```
%{
#include <stdio.h>
extern int yylex(void);  // Explicit declaration for yylex
void yyerror(const char *s);  // Proper prototype for yyerror
extern FILE *yyin;
%}
%token NUM
%%
start:
   expr '=' expr { printf("\nResult = %d\n", $3); }
   | expr        { printf("\nResult = %d\n", $1); } ;
```

```
expr:
    expr '+' NUM { $$ = $1 + $3; }
    | NUM       { $$ = $1; }
    ;
%%
int main() {
    yyin = stdin;
    do {
        yyparse();
    } while (!feof(yyin));
    return 0;
}
void yyerror(const char *s) {  // Match prototype with const
    fprintf(stderr, "Error: %s\n", s);
}
```

Run command:

```
lex lex.l
yacc -d x1.y
gcc lex.yy.c y.tab.c -o calculator -ll
```

Output:



```
yadavlalu5252@lalu:~/string_program$ lex lex.l
yacc -d x1.y
gcc lex.yy.c y.tab.c -o calculator -ll
yadavlalu5252@lalu:~/string_program$ ./calculator
2+5

Result = 7
10+10

Result = 20
```

Experiment No.3

1. Calculator

Lex Program(x1.l):

```
%{
#include "y.tab.h"  // Include the Yacc header file
#include <stdlib.h>   // For atoi
extern int yylval;   // Declare external variable for Yacc
%}

%%
[0-9]+     {
   yylval = atoi(yytext); // Convert string to integer
   return NUM;           // Return NUM token
}
[A-Za-z]    {
   return A;             // Return A for variables (like A)
}
"="        {
   return '=';           // Return '=' token
}
"+"        {
   return '+';           // Return '+' token
}
[ \t]      { /* Ignore spaces and tabs */ }
\n         {
   return 0;             // End of input line
}
%%
```

```
int yywrap() {
    return 1; // Indicates no more input
}
```

Yacc Program(x1.y):

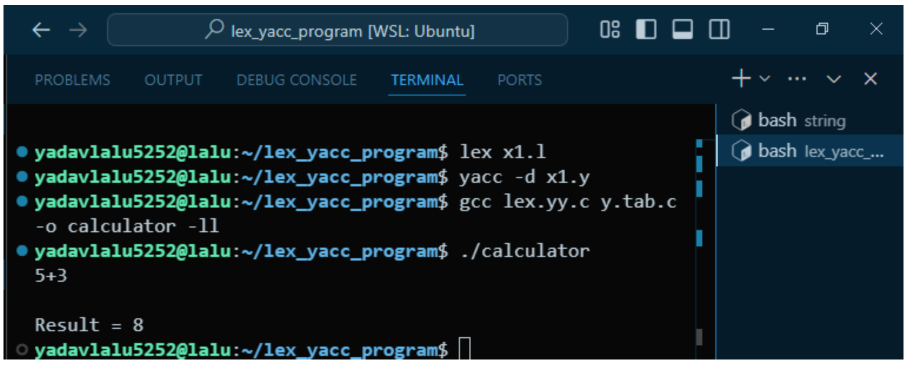
```
%{
#include <stdio.h>
#include <stdlib.h>
extern int yylex();               // Declare yylex
void yyerror(const char *s);      // Declare yyerror
%}

%token NUM A

// Define operator precedence (to resolve conflicts)
%left '+'

%%
state:
    A '=' E { printf("\nResult = %d\n", $3); }  // Assignment
  | E     { printf("\nResult = %d\n", $1); }   // Expression
  ;

E:
    E '+' E { $$ = $1 + $3; }  // Addition
  | NUM     { $$ = $1; }      // Number
  | A      { $$ = 0; }       // Variable (unused in your current logic)
  ;
%%
```

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

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}
```

Run command:

```
lex x1.l
yacc -d x1.y
gcc lex.yy.c y.tab.c -o calculator -ll
```

Output:



```
yadavlalu5252@lalu:~/lex_yacc_program$ lex x1.l
yadavlalu5252@lalu:~/lex_yacc_program$ yacc -d x1.y
yadavlalu5252@lalu:~/lex_yacc_program$ gcc lex.yy.c y.tab.c
-o calculator -ll
yadavlalu5252@lalu:~/lex_yacc_program$ ./calculator
5+3

Result = 8
yadavlalu5252@lalu:~/lex_yacc_program$
```

2. String

Lex Program (anbn.l)

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

%%
a      { return A; }
b      { return B; }
\n     { return '\n'; }
.      { /* Ignore other characters */ }
%%

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

Yacc Program (anbn.y):

```
%{
#include <stdio.h>
extern int yylex(void);
void yyerror(const char *);
%}

%token A B

%%
start:
   statements
```

```
    ;

statements:
    statement
    | statements statement
    ;

statement:
    anbn '\n'   { printf("Valid: a^n b^n\n"); }
    ;

anbn:
    A B        /* Base case: ab */
    | A anbn B  /* Recursive case: a(...)b */
    ;

%%

int main() {
    printf("Enter strings (e.g., 'ab', 'aabb'):\n");
    yyparse();
    return 0;
}

void yyerror(const char *s) {  // Added 'const' for modern C
    printf("Invalid: Not a^n b^n\n");
}
```

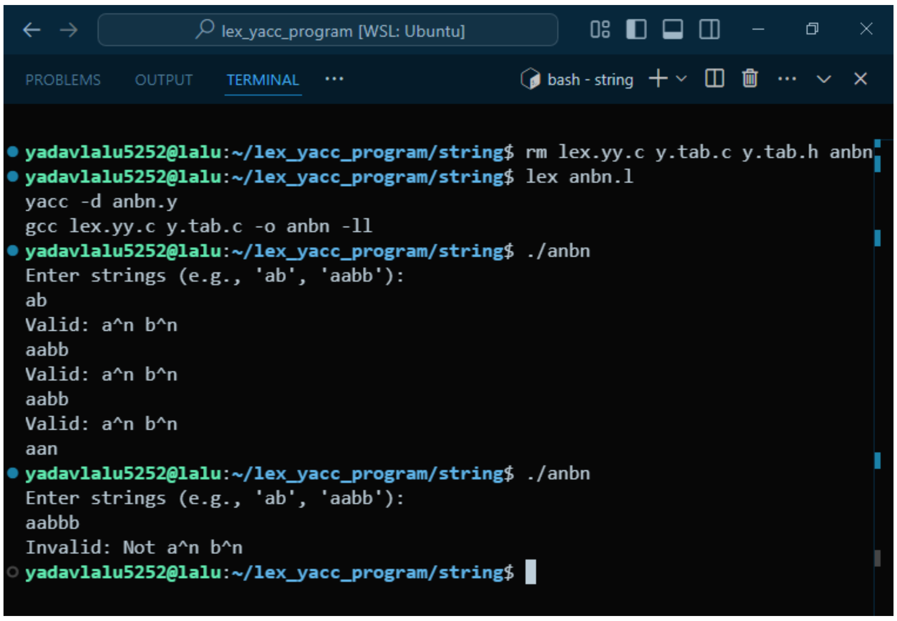
Run Command:

lex anbn.l

yacc -d anbn.y

gcc lex.yy.c y.tab.c -o anbn -ll

Output:



```
yadavlalu5252@lalu:~/lex_yacc_program/string$ rm lex.yy.c y.tab.c y.tab.h anbn
yadavlalu5252@lalu:~/lex_yacc_program/string$ lex anbn.l
yacc -d anbn.y
gcc lex.yy.c y.tab.c -o anbn -ll
yadavlalu5252@lalu:~/lex_yacc_program/string$ ./anbn
Enter strings (e.g., 'ab', 'aabb'):
ab
Valid: a^n b^n
aabb
Valid: a^n b^n
aabb
Valid: a^n b^n
aan
yadavlalu5252@lalu:~/lex_yacc_program/string$ ./anbn
Enter strings (e.g., 'ab', 'aabb'):
aabbb
Invalid: Not a^n b^n
yadavlalu5252@lalu:~/lex_yacc_program/string$
```

Experiment No. 1

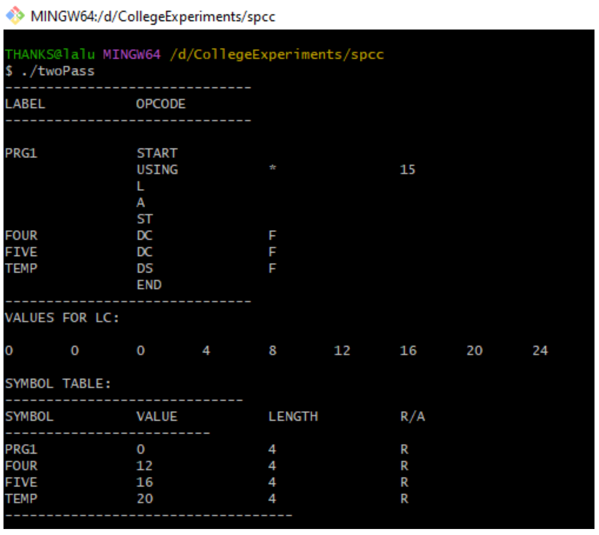
Two pass assembler:

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

int main() {
    char *code[9][4] = {
        {"PRG1", "START", "", ""},
        {"", "USING", "*", "15"},
        {"", "L", "", ""},
        {"", "A", "", ""},
        {"", "ST", "", ""},
        {"FOUR", "DC", "F", ""},
        {"FIVE", "DC", "F", ""},
        {"TEMP", "DS", "F", ""},
        {"", "END", "", ""}
    };
    char av[3] = {'\0'};
    char avail[15] = {'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N'};
    int i, j, k, count[3] = {0}, lc[9] = {0}, loc = 0;
    printf("-----------------------------\n");
    printf("LABEL\t\tOPCODE\n");
    printf("-----------------------------\n\n");

    for (i = 0; i < 9; i++) {
        for (j = 0; j < 4; j++) {
            printf("%s\t\t", code[i][j]);
        }
        printf("\n");
    }

    printf("-----------------------------\n");
    printf("VALUES FOR LC:\n\n");

    for (j = 0; j < 9; j++) {
        if ((strcmp(code[j][1], "START") != 0) && (strcmp(code[j][1], "USING") != 0) &&
(strcmp(code[j][1], "L") != 0)) {
            lc[j] = lc[j - 1] + 4;
        }
        printf("%d\t", lc[j]);
    }

    printf("\n\nSYMBOL TABLE:\n-----------------------------\n");
    printf("SYMBOL\t\tVALUE\t\tLENGTH\t\tR/A");
    printf("\n-------------------------\n");

    loc = 0;
    for (i = 0; i < 9; i++) {
        if (strcmp(code[i][1], "START") == 0) {
            printf("%s\t\t%d\t\t%d\t\t%c\n", code[i][0], loc, 4, 'R');
        } else if (strcmp(code[i][0], "") != 0) {
            printf("%s\t\t%d\t\t%d\t\t%c\n", code[i][0], loc, 4, 'R');
            loc += 4;
        } else if (strcmp(code[i][1], "USING") == 0) {
            continue;
        } else {
            loc += 4;
        }
    }
```

```c
printf("-----------------------------------\n");
printf("\nBASE TABLE:\n-----------------------------\n");
printf("REG NO\t\tAVAILABILITY\n");
printf("------------------------------------\n");

for (j = 0; j < 9; j++) {
   if (strcmp(code[j][1], "USING") == 0) {
      strcpy(av, code[j][3]);
   }
}

count[0] = av[0] - '0'; // Convert character to integer
count[1] = av[1] - '0'; // Convert character to integer (if applicable)
count[2] = count[0] * 10 + count[1];
avail[count[2] - 1] = 'Y';

for (k = 0; k < 15; k++) {
   printf("%d\t\t%c\n", k + 1, avail[k]);
}

printf("----------------------------------\n");
printf("PASS2 TABLE:\n\n");
printf("LABEL\t\tOP1\t\tLC\t\t");
printf("\n---------------------\n");

for (i = 0; i < 9; i++) {
   for (j = 0; j < 4; j++) {
      printf("%s\t\t", code[i][j]);
```

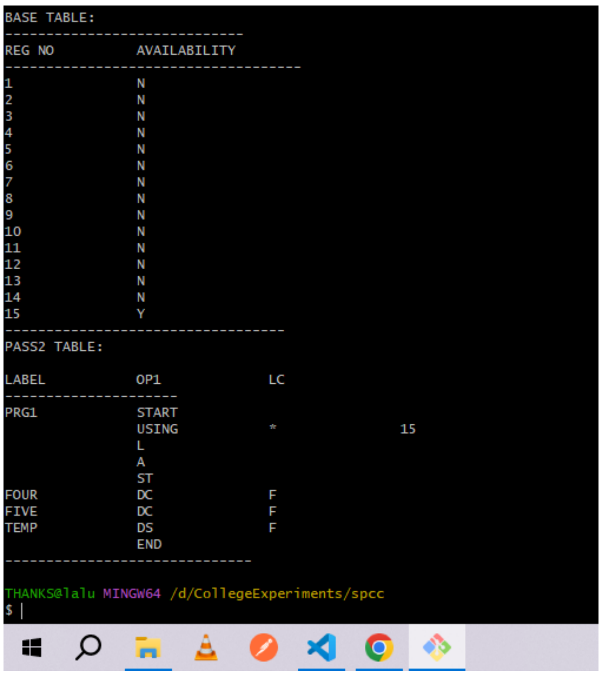
```
        }
        printf("\n");
    }

    printf("-----------------------------\n");
    return 0;
}
```

Output:



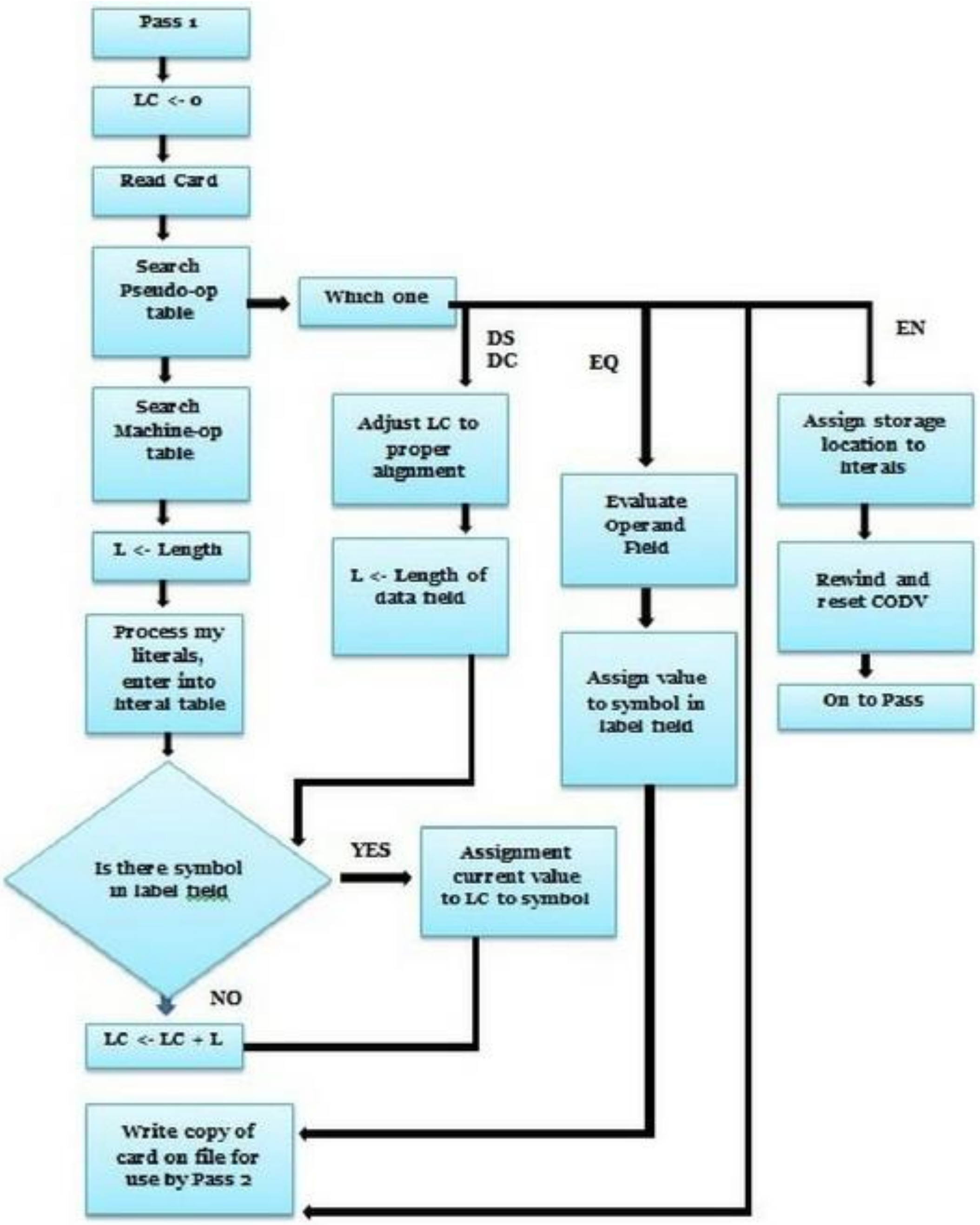
MINGW64:/d/CollegeExperiments/spcc

```
THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$ ./twoPass
------------------------------
LABEL           OPCODE
------------------------------

PRG1            START
                USING           *               15
                L
                A
                ST
FOUR            DC              F
FIVE            DC              F
TEMP            DS              F
                END
------------------------------
VALUES FOR LC:

0       0       0       4       8       12      16      20      24

SYMBOL TABLE:
-----------------------------
SYMBOL          VALUE           LENGTH          R/A
-------------------------
PRG1            0               4               R
FOUR            12              4               R
FIVE            16              4               R
TEMP            20              4               R
-----------------------------------
```

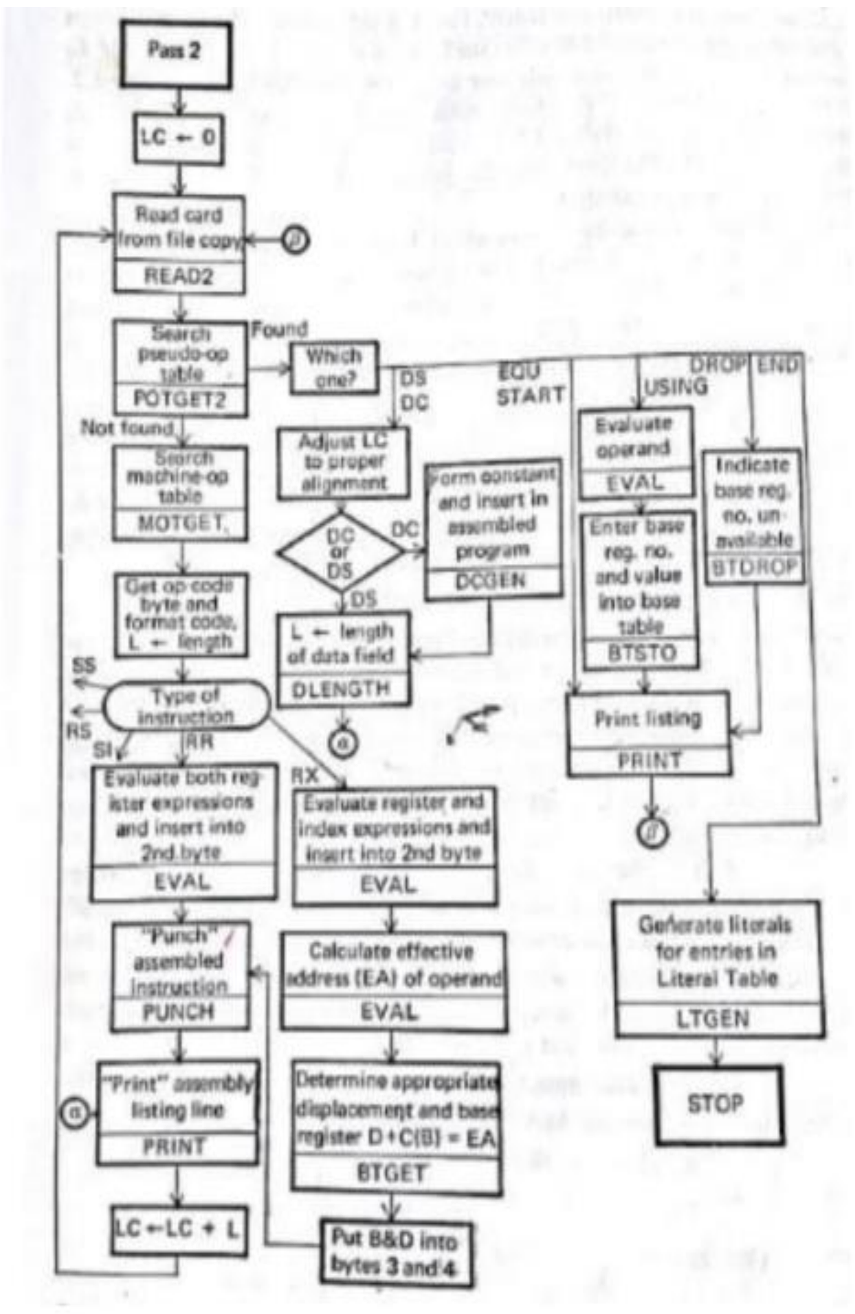


```
BASE TABLE:
-----------------------------
REG NO          AVAILABILITY
------------------------------------
1               N
2               N
3               N
4               N
5               N
6               N
7               N
8               N
9               N
10              N
11              N
12              N
13              N
14              N
15              Y
----------------------------------
PASS2 TABLE:

LABEL           OP1             LC
---------------------
PRG1            START
                USING           *               15
                L
                A
                ST
FOUR            DC              F
FIVE            DC              F
TEMP            DS              F
                END
------------------------------

THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$ |
```

Pass 1:


Pass 1
LC <- 0
Read Card
Search Pseudo-op table
Which one
DS DC
EQ
EN
Search Machine-op table
Adjust LC to proper alignment
Evaluate Operand Field
Assign storage location to literals
L <- Length
L <- Length of data field
Rewind and reset CODV
Process my literals, enter into literal table
Assign value to symbol in label field
On to Pass
Is there symbol in label field
YES
Assignment current value to LC to symbol
NO
LC <- LC + L
Write copy of card on file for use by Pass 2

Pass 2:


Pass 2
LC ← 0
Read card from file copy
READ2
Search pseudo-op table
POTGET2
Found
Not found
Which one?
DS
DC
EQU
START
USING
DROP
END
Search machine-op table
MOTGET
Adjust LC to proper alignment
DC or DS
Form constant and insert in assembled program
DCGEN
Evaluate operand
EVAL
Enter base reg. no. and value into base table
BTSTO
Indicate base reg. no. unavailable
BTDROP
Get op code byte and format code, L ← length
L ← length of data field
DLENGTH
Type of instruction
SS
RS
SI
RR
RX
Print listing
PRINT
Evaluate both register expressions and insert into 2nd byte
EVAL
Evaluate register and index expressions and insert into 2nd byte
EVAL
Generate literals for entries in Literal Table
LTGEN
"Punch" assembled instruction
PUNCH
Calculate effective address (EA) of operand
EVAL
STOP
"Print" assembly listing line
PRINT
Determine appropriate displacement and base register D+C(B) = EA
BTGET
LC ← LC + L
Put B&D into bytes 3 and 4

Experiment No. 2

Two pass macro preprocessor:

```
class macroprocessor {
   macroprocessor() {
   }


   public static void main(String[] var0) {
 String[][] var1 = new String[][]{{"ADD", "A", "", "", ""}, {"MACRO", "ADD1", "&ARG", "", ""},
{"LOAD", "ARG", "", "", ""}, {"MEND", "", "", "", ""}, {"MACRO", "PQR", "&A", "&B", "&C"},
{"ADD", "B", "", "", ""}, {"READ", "C", "", "", ""}, {"READ", "A", "", "", ""}, {"MEND", "", "", "",
""}, {"MACRO", "LMN", "", "", ""}, {"LOAD", "C", "", "", ""}, {"MEND", "", "", "", ""}, {"LOAD",
"B", "", "", ""}, {"PQR", "5", "3", "2", ""}, {"ADD1", "1", "", "", ""}, {"LMN", "", "", ""}, {"SUB",
"C", "", "", ""}, {"ENDP", "", "", "", ""}};
      String[] var2 = new String[3];
      String[] var3 = new String[4];
      String[] var4 = new String[4];
      String[] var5 = new String[4];
      int[] var6 = new int[3];
      int var7 = 0;
      int var8 = 0;
      int var9 = 0;
      boolean var10 = false;
      int var10001;
      for(int var11 = 0; var11 < 18; ++var11) {
         if (var1[var11][0].equals("MACRO")) {
            var2[var7] = var1[var11][1];


            for(int var12 = 2; var12 < 5; ++var12) {
```

```
        if (!var1[var11][var12].equals("")) {
          var3[var9] = var1[var11][1];
          var4[var9] = var1[var11][var12];
          var10001 = var9++;
          ++var8;
          var5[var10001] = "#" + var8;
        }
      }
      var6[var7++] = var8;
      var8 = 0;
    }
  }
  String[] var19 = new String[4];
  String[] var20 = new String[4];
  String[] var13 = new String[4];
  var7 = 1;
  var8 = 0;
  int var14;
  int var15;
  for(var14 = 0; var14 < 18; ++var14) {
    for(var15 = 0; var15 < var2.length; ++var15) {
      if (var1[var14][0].equals(var2[var15]) && !var1[var14][1].equals("")) {
        while(!var1[var14][var7].equals("")) {
          var19[var8] = var1[var14][0];
          var20[var8] = var1[var14][var7];
          var13[var8] = "#" + var7;
          ++var7;
          ++var8;
        }
        var7 = 1;
      }
   }
}

System.out.println("Macro Name Table");
System.out.println("___________________________");
System.out.println("Macro Name\tNo. of Parameters");
System.out.println("___________________________");

for(var14 = 0; var14 < var2.length; ++var14) {
   System.out.println(var2[var14] + "\t\t" + var6[var14]);
}

System.out.println("-----------------------\n");
System.out.println("Macro Definition Table");
System.out.println("-----------------------");
System.out.println("Index\tInstruction");
System.out.println("---------------------");
var14 = 1;
var15 = 0;

while(true) {
   while(var15 < 18) {
      if (var1[var15][0].equals("MACRO")) {
         ++var15;

         for(; !var1[var15][0].equals("MEND"); ++var15) {
            for(int var16 = 0; var16 < var4.length; ++var16) {
```

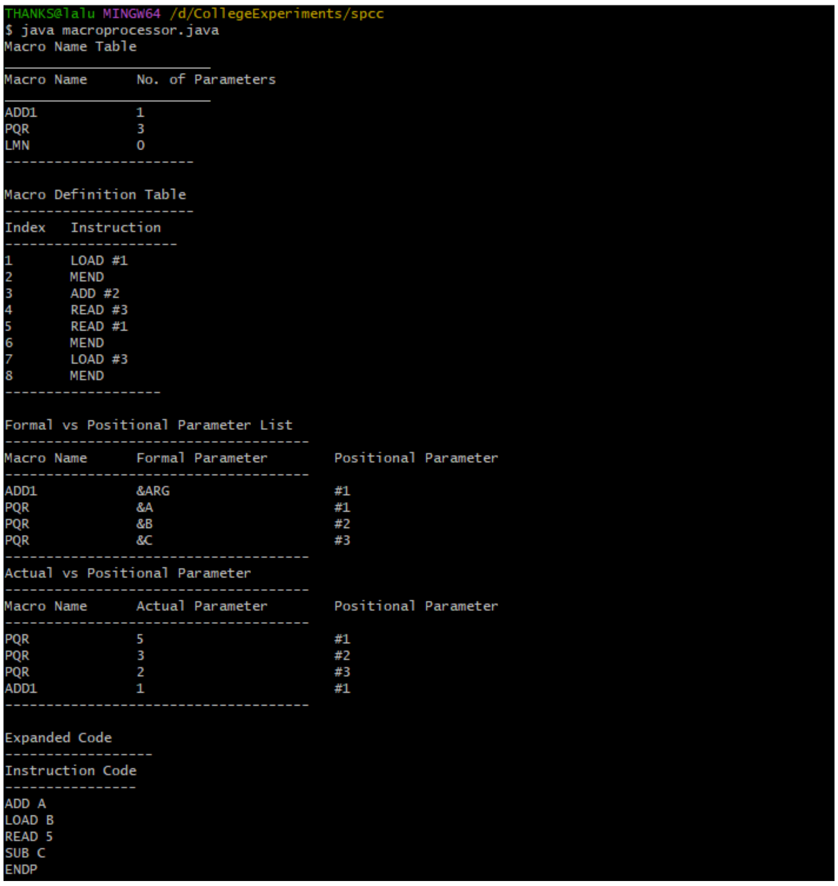
```
        if (("&" + var1[var15][1]).equals(var4[var16])) {
          var10001 = var14++;
          System.out.println("" + var10001 + "\t" + var1[var15][0] + " " + var5[var16]);
          break;
        }
      }
    }
    var10001 = var14++;
    System.out.println("" + var10001 + "\tMEND");
  } else {
    ++var15;
  }
}

System.out.println("-------------------\n");
System.out.println("Formal vs Positional Parameter List");
System.out.println("-------------------------------------");
System.out.println("Macro Name\tFormal Parameter\tPositional Parameter");
System.out.println("-------------------------------------");

for(var15 = 0; var15 < var3.length; ++var15) {
  System.out.println(var3[var15] + "\t\t" + var4[var15] + "\t\t\t" + var5[var15]);
}

System.out.println("-------------------------------------");
System.out.println("Actual vs Positional Parameter");
System.out.println("-------------------------------------");
System.out.println("Macro Name\tActual Parameter\tPositional Parameter");
System.out.println("-------------------------------------");
```

```
for(var15 = 0; var15 < var19.length; ++var15) {
  System.out.println(var19[var15] + "\t\t" + var20[var15] + "\t\t\t" + var13[var15]);
}

System.out.println("-----------------------------------\n");
System.out.println("Expanded Code");
System.out.println("-----------------");
System.out.println("Instruction Code");
System.out.println("---------------");
String[][] var21 = new String[4][2];

int var17;
for(var15 = 0; var15 < 4; ++var15) {
  for(var17 = 0; var17 < 4; ++var17) {
    if (var3[var15].equals(var19[var17]) && var5[var15].equals(var13[var17])) {
      var21[var15][0] = var4[var15];
      var21[var15][1] = var20[var17];
      break;
    }
  }
}

var15 = 0;

while(true) {
  while(true) {
    while(var15 < 18) {
      if (!var1[var15][0].equals("ADD") && !var1[var15][0].equals("SUB") &&
!var1[var15][0].equals("ENDP") && !var1[var15][0].equals("LOAD")) {
```

```
if (var1[var15][0].equals("MACRO")) {
  ++var15;

  while(!var1[var15][0].equals("MEND")) {
    ++var15;
  }

  ++var15;
} else {
  label115:
  for(var17 = 0; var17 < 18; ++var17) {
    if (var1[var17][1].equals(var1[var15][0])) {
      ++var17;

      while(true) {
        if (var1[var17][0].equals("MEND")) {
          break label115;
        }

        for(int var18 = 0; var18 < 4; ++var18) {
          if (("&" + var1[var17][var18]).equals(var21[var18][0])) {
            System.out.println(var1[var17][0] + " " + var21[var18][1]);
          }
        }

        ++var17;
      }
    }
  }
```

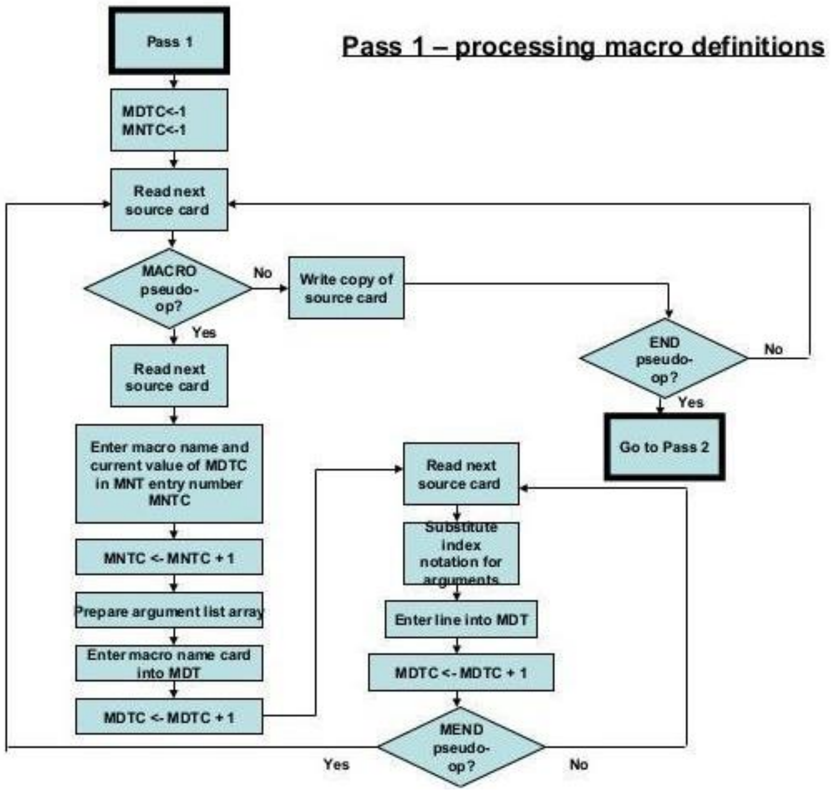
```
                    ++var15;
                  }
                } else {
                  System.out.println(var1[var15][0] + " " + var1[var15][1]);
                  ++var15;
                }
              }

              return;
            }
          }
        }
      }
    }
```

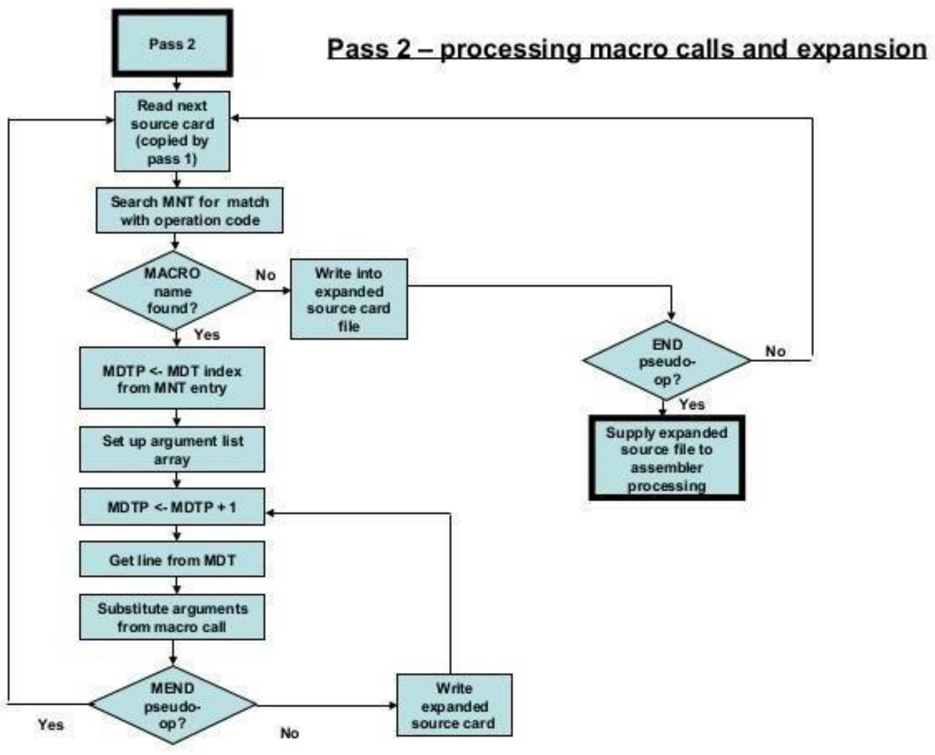
Output:



```
THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$ java macroprocessor.java
Macro Name Table
_________________________
Macro Name      No. of Parameters
_________________________
ADD1            1
PQR             3
LMN             0
-----------------------

Macro Definition Table
-----------------------
Index   Instruction
---------------------
1       LOAD #1
2       MEND
3       ADD #2
4       READ #3
5       READ #1
6       MEND
7       LOAD #3
8       MEND
-------------------

Formal vs Positional Parameter List
-------------------------------------
Macro Name      Formal Parameter        Positional Parameter
-------------------------------------
ADD1            &ARG                    #1
PQR             &A                      #1
PQR             &B                      #2
PQR             &C                      #3
-------------------------------------
Actual vs Positional Parameter
-------------------------------------
Macro Name      Actual Parameter        Positional Parameter
-------------------------------------
PQR             5                       #1
PQR             3                       #2
PQR             2                       #3
ADD1            1                       #1
-------------------------------------

Expanded Code
------------------
Instruction Code
----------------
ADD A
LOAD B
READ 5
SUB C
ENDP
```

Flowchart of pass 1 and pass2 macro preprocessor:


Pass 1
Pass 1 – processing macro definitions
MDTC<-1
MNTC<-1
Read next source card
MACRO pseudo-op?
No
Write copy of source card
Yes
END pseudo-op?
No
Yes
Go to Pass 2
Read next source card
Enter macro name and current value of MDTC in MNT entry number MNTC
MNTC <- MNTC + 1
Prepare argument list array
Enter macro name card into MDT
MDTC <- MDTC + 1
Read next source card
Substitute index notation for arguments
Enter line into MDT
MDTC <- MDTC + 1
MEND pseudo-op?
Yes
No


Pass 2 – processing macro calls and expansion
Pass 2
Read next source card (copied by pass 1)
Search MNT for match with operation code
MACRO name found?
No
Write into expanded source card file
Yes
MDTP <- MDT index from MNT entry
Set up argument list array
MDTP <- MDTP + 1
Get line from MDT
Substitute arguments from macro call
MEND pseudo-op?
Yes
No
Write expanded source card
END pseudo-op?
No
Yes
Supply expanded source file to assembler processing

Experiment No.5

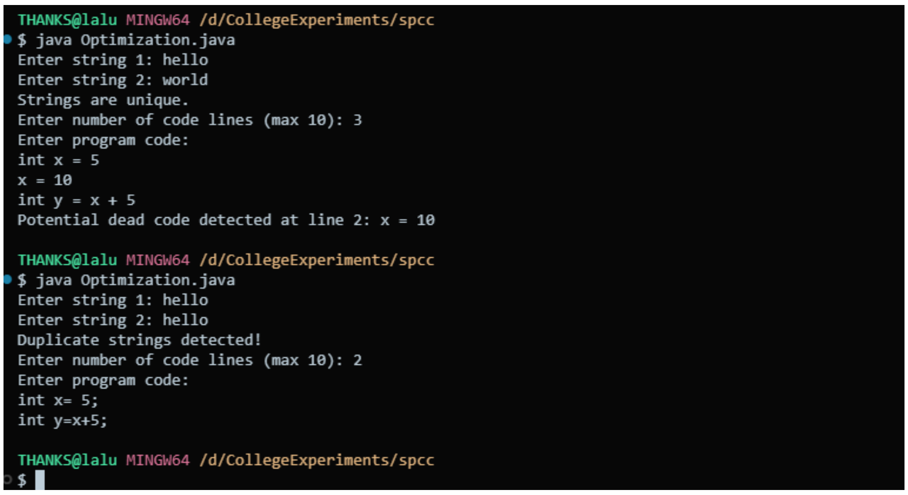
Optimisation Code:

```
import java.io.*;
import java.util.*;

public class Optimization {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        String s1, s2;
        String[] code = new String[10];

        // Input handling for strings
```

```
System.out.print("Enter string 1: ");
s1 = br.readLine();

System.out.print("Enter string 2: ");
s2 = br.readLine();

// String comparison
if(s1.equals(s2)) {
   System.out.println("Duplicate strings detected!");
   s2 = null;
} else {
   System.out.println("Strings are unique.");
}

// Code input handling
System.out.print("Enter number of code lines (max 10): ");
int n = Integer.parseInt(br.readLine());

// Validate input
if(n <= 0 || n > 10) {
   System.out.println("Invalid number of lines!");
   return;
}

System.out.println("Enter program code:");
for(int i = 0; i < n; i++) {
   code[i] = br.readLine();
}
```

```
        // Dead code detection logic
        for(int i = 0; i < n - 1; i++) {
            String currentLine = code[i].trim();
            String nextLine = code[i + 1].trim();

            // Check for variable declaration pattern
            if(currentLine.startsWith("int ")) {
                String[] parts = currentLine.split("=");
                if(parts.length > 0) {
                    String varName = parts[0].replace("int", "").trim().split(" ")[0];

                    // Check if next line uses the same variable
                    if(nextLine.contains(varName + " =")) {
                        System.out.println("Potential dead code detected at line " + (i + 2) + ": " +
nextLine);
                    }
                }
            }
        }
    }
}
```

Output:



```
THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$ java Optimization.java
Enter string 1: hello
Enter string 2: world
Strings are unique.
Enter number of code lines (max 10): 3
Enter program code:
int x = 5
x = 10
int y = x + 5
Potential dead code detected at line 2: x = 10

THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$ java Optimization.java
Enter string 1: hello
Enter string 2: hello
Duplicate strings detected!
Enter number of code lines (max 10): 2
Enter program code:
int x= 5;
int y=x+5;

THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$
```

Experiment No. 6

Target Optimized Code:

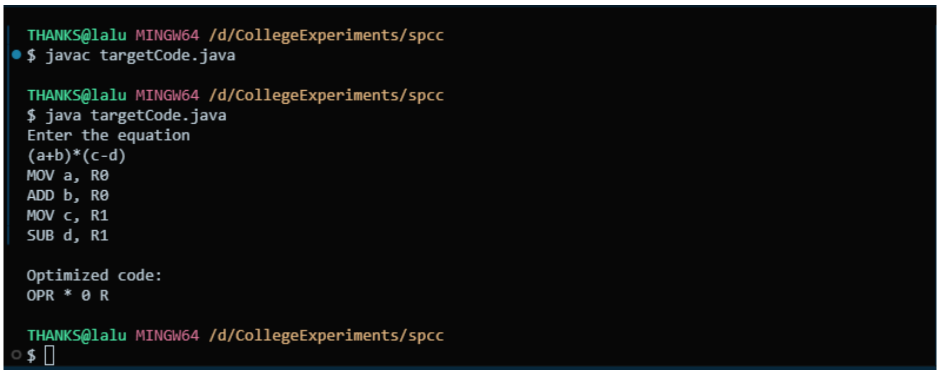
```
import java.io.*;
public class targetCode {
   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

```
System.out.println("Enter the equation");
String stmt = br.readLine();

StringBuffer ans = new StringBuffer("");
int reg = 0;
int parenCount = 0;

// First pass: Process parentheses and generate basic instructions
for(int i = 0; i < stmt.length(); i++) {
    char c = stmt.charAt(i);

    switch(c) {
        case '(':
            parenCount++;
            break;
        case ')':
            parenCount--;
            break;
        case '+':
        case '-':
        case '*':
        case '/':
            if(parenCount > 0 && i > 0 && i < stmt.length()-1) {
                char leftOperand = stmt.charAt(i-1);
                char rightOperand = stmt.charAt(i+1);

                System.out.println("MOV " + leftOperand + ", R" + reg);
                switch(c) {
                    case '+':
```

```
                    System.out.println("ADD " + rightOperand + ", R" + reg);
                    break;
                case '-':
                    System.out.println("SUB " + rightOperand + ", R" + reg);
                    break;
                case '*':
                    System.out.println("MUL " + rightOperand + ", R" + reg);
                    break;
                case '/':
                    System.out.println("DIV " + rightOperand + ", R" + reg);
                    break;
            }
            ans.append("R" + reg);
            reg++;
            i++; // Skip processed right operand
        } else {
            ans.append(c);
        }
        break;
    default:
        if(parenCount == 0) {
            ans.append(c);
        }
        break;
  }
}

// Second pass: Process remaining operations
String ans1 = ans.toString();
```

```java
        System.out.println("\nOptimized code:");
        for(int i = 0; i < ans1.length(); i++) {
            char c = ans1.charAt(i);
            if("+-*/".indexOf(c) != -1 && i > 0 && i < ans1.length()-1) {
                String left = ans1.substring(i-1, i);
                String right = ans1.substring(i+1, i+2);

                System.out.println("OPR " + c + " " + left + " " + right);
            }
        }
    }
}
```

Output:



```
THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$ javac targetCode.java

THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$ java targetCode.java
Enter the equation
(a+b)*(c-d)
MOV a, R0
ADD b, R0
MOV c, R1
SUB d, R1

Optimized code:
OPR * 0 R

THANKS@lalu MINGW64 /d/CollegeExperiments/spcc
$ 
```

Experiment No.7

Program:-

To write a code for LR(0) Parser for following Production:

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

```
// Action Table (axn) and Goto Table (gotot)
int axn[][6][2] = {
    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
    {{-1,-1},{100,6},{-1,-1},{-1,-1},{-1,-1},{102,102}},
    {{-1,-1},{101,2},{100,7},{-1,-1},{101,2},{101,2}},
    {{-1,-1},{101,4},{101,4},{-1,-1},{101,4},{101,4}},
    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
    {{100,5},{101,6},{101,6},{-1,-1},{101,6},{101,6}},
    {{100,5},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1}},
    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
    {{-1,-1},{100,6},{-1,-1},{-1,-1},{100,11},{-1,-1}},
    {{-1,-1},{101,1},{100,7},{-1,-1},{101,1},{101,1}},
    {{-1,-1},{101,3},{101,3},{-1,-1},{101,3},{101,3}},
    {{-1,-1},{101,5},{101,5},{-1,-1},{101,5},{101,5}}
};

int gotot[12][3] = {
    {1,2,3}, {-1,-1,-1}, {-1,-1,-1}, {-1,-1,-1},
    {8,2,3}, {-1,-1,-1}, {-1,9,3}, {-1,-1,10},
    {-1,-1,-1}, {-1,-1,-1}, {-1,-1,-1}, {-1,-1,-1}
};

int a[10];
char b[10];
int top = -1, btop = -1;

void push(int k) {
    if(top < 9) a[++top] = k;
}
```

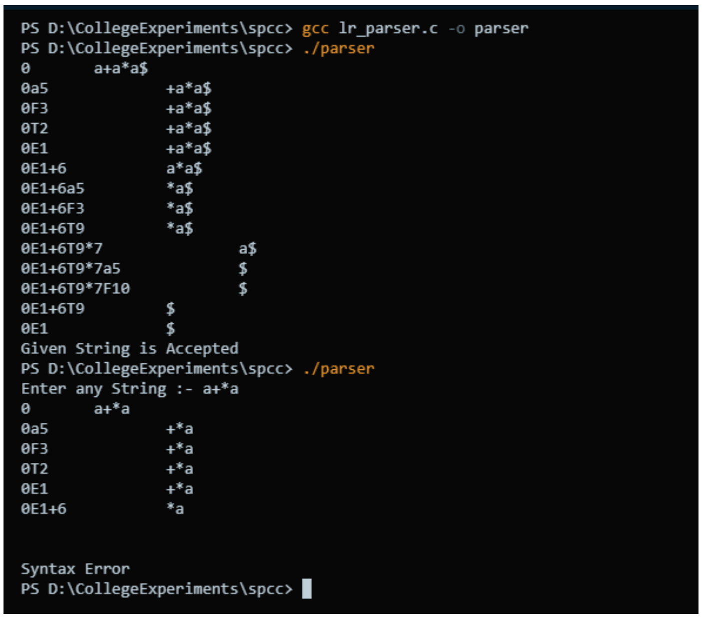
```
void pushb(char k) {
    if(btop < 9) b[++btop] = k;
}

int TOS() {
    return a[top];
}

void pop() {
    if(top >= 0) top--;
}

void popb() {
    if(btop >= 0) b[btop--] = '\0';
}

void display() {
    for(int i = 0; i <= top; i++)
        printf("%d%c", a[i], b[i]);
}

void display1(char p[], int m) {
    printf("\t\t");
    for(int l = m; p[l] != '\0'; l++)
        printf("%c", p[l]);
    printf("\n");
}
```

```
void error() {
    printf("\n\nSyntax Error");
}

void reduce(int p) {
    char *dest = "";
    char src;

    switch(p) {
        case 1: dest = "E+T"; src = 'E'; break;
        case 2: dest = "T";   src = 'E'; break;
        case 3: dest = "T*F"; src = 'T'; break;
        case 4: dest = "F";   src = 'T'; break;
        case 5: dest = "(E)"; src = 'F'; break;
        case 6: dest = "i";   src = 'F'; break;
        default: dest = "";   src = '\0'; break;
    }

    // Pop operations with bounds checking
    int len = strlen(dest);
    while(len-- > 0 && top >= 0 && btop >= 0) {
        pop();
        popb();
    }

    pushb(src);

    // Goto table lookup with bounds checking
    int ad;
    switch(src) {
        case 'E': ad = 0; break;
        case 'T': ad = 1; break;
        case 'F': ad = 2; break;
        default:  ad = -1; break;
    }

    if(ad != -1 && TOS() >= 0 && TOS() < 12 && ad < 3) {
        push(gotot[TOS()][ad]);
    }
}

int main() {
    char ip[20];
    printf("Enter any String :- ");
    fgets(ip, 20, stdin);  // Safer alternative to gets()
    ip[strcspn(ip, "\n")] = '\0'; // Remove newline

    push(0);
    display();
    printf("\t%s\n", ip);

    for(int j = 0; ip[j] != '\0';) {
        int st = TOS();
        char an = ip[j];
        int ic = -1;

        // Fixed logical operators (&& instead of &)
        if(an >= 'a' && an <= 'z') ic = 0;
```

```
else if(an == '+') ic = 1;

else if(an == '*') ic = 2;

else if(an == '(') ic = 3;

else if(an == ')') ic = 4;

else if(an == '$') ic = 5;


if(ic == -1) {

   error();

   break;

}


if(axn[st][ic][0] == 100) {

   pushb(an);

   push(axn[st][ic][1]);

   display();

   j++;

   display1(ip, j);

}

else if(axn[st][ic][0] == 101) {

   reduce(axn[st][ic][1]);

   display();

   display1(ip, j);

}

else if(axn[st][ic][1] == 102) {

   printf("Given String is Accepted");

   break;

}

else {

   error();
```

```
            break;
        }
    }
    return 0;
}
```

Output:



```
PS D:\CollegeExperiments\spcc> gcc lr_parser.c -o parser
PS D:\CollegeExperiments\spcc> ./parser
0       a+a*a$
0a5             +a*a$
0F3             +a*a$
0T2             +a*a$
0E1             +a*a$
0E1+6           a*a$
0E1+6a5         *a$
0E1+6F3         *a$
0E1+6T9         *a$
0E1+6T9*7               a$
0E1+6T9*7a5             $
0E1+6T9*7F10            $
0E1+6T9         $
0E1             $
Given String is Accepted
PS D:\CollegeExperiments\spcc> ./parser
Enter any String :- a+*a
0       a+*a
0a5             +*a
0F3             +*a
0T2             +*a
0E1             +*a
0E1+6           *a


Syntax Error
PS D:\CollegeExperiments\spcc>
```