9. Write a program to implement intermediate code generation algorithm

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

void generateAssembly(char *tacLine)
{
   char result[10], op1[10], op2[10], op;

   if (sscanf(tacLine, "%s = %s %c %s", result,
op1, &op, op2) == 4)
   {
      printf("MOV %s, R1\n", op1);
      if (op == '+')
         printf("ADD %s, R1\n", op2);
      else if (op == '-')
         printf("SUB %s, R1\n", op2);
      else if (op == '*')
         printf("MUL %s, R1\n", op2);
      else if (op == '/')
         printf("DIV %s, R1\n", op2);
      printf("MOV R1, %s\n", result);
   }
   else if (sscanf(tacLine, "%s = %s", result,
op1) == 2)
   {
      printf("MOV %s, R1\n", op1);
      printf("MOV R1, %s\n", result);
   }
}

int main()
{
   char tac[10][20];
   int n;

   printf("Enter number of TAC instructions: ");
   scanf("%d", &n);
   getchar(); // Consume newline

   printf("Enter TAC instructions:\n");
   for (int i = 0; i < n; i++)
   {
      fgets(tac[i], 20, stdin);
      tac[i][strcspn(tac[i], "\n")] = 0; // Remove
newline
   }

   printf("\nAssembly Code:\n");
   fflush(stdout); // Ensure output is displayed
   for (int i = 0; i < n; i++)
   {
      generateAssembly(tac[i]);
   }

   getchar(); // Pause to view output
   return 0;
}
```

10. Write a program to demonstrate code generation algorithm

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

int tempVarCount = 1;

char *newTemp()
{
   static char temp[10][5];
   snprintf(temp[tempVarCount - 1], 5, "t%d",
tempVarCount);
   return temp[tempVarCount++ - 1];
}

void generateTAC(char expr[])
{
   char lhs[10], rhs[50];
   if (sscanf(expr, "%s = %[^\n]", lhs, rhs) < 2)
   {
      printf("; Invalid expression\n");
      return;
   }

   char tokens[5][10];
   int tokIndex = 0;

   for (int i = 0; rhs[i] && tokIndex < 5;)
   {
      if (isspace(rhs[i]))
         i++;
      else if (isalnum(rhs[i]))
      {
         int j = 0;
         while (isalnum(rhs[i]) && j < 9)
            tokens[tokIndex][j++] = rhs[i++];
         tokens[tokIndex][j] = '\0';
         tokIndex++;
      }
      else
      {
         tokens[tokIndex][0] = rhs[i++];
         tokens[tokIndex][1] = '\0';
         tokIndex++;
      }
   }

   if (tokIndex == 0)
   {
      printf("; Empty RHS\n");
      return;
   }

   for (int i = 0; i < tokIndex; i++)
   {
      if (strcmp(tokens[i], "*") == 0 ||
strcmp(tokens[i], "/") == 0)
      {
         if (i < 1 || i + 1 >= tokIndex)
            continue;
         char *temp = newTemp();
```

```
            printf("%s = %s %s %s\n", temp,
tokens[i - 1], tokens[i], tokens[i + 1]);
            strcpy(tokens[i - 1], temp);
            for (int j = i; j < tokIndex - 2; j++)
                strcpy(tokens[j], tokens[j + 2]);
            tokIndex -= 2;
            i = -1;
        }
    }

    for (int i = 0; i < tokIndex; i++)
    {
        if (strcmp(tokens[i], "+") == 0 ||
strcmp(tokens[i], "-") == 0)
        {
            if (i < 1 || i + 1 >= tokIndex)
                continue;
            char *temp = newTemp();
            printf("%s = %s %s %s\n", temp,
tokens[i - 1], tokens[i], tokens[i + 1]);
            strcpy(tokens[i - 1], temp);
            for (int j = i; j < tokIndex - 2; j++)
                strcpy(tokens[j], tokens[j + 2]);
            tokIndex -= 2;
            i = -1;
        }
    }

    if (tokIndex > 0)
        printf("%s = %s\n", lhs, tokens[0]);
}

int main()
{
    char expr[50];
    printf("Enter expression: ");
    if (!fgets(expr, sizeof(expr), stdin))
        return 1;
    expr[strcspn(expr, "\n")] = 0;

    printf("\nTAC:\n");
    generateTAC(expr);
    getchar();
    return 0;
}
```

14. Write a LEX-YACC specification program for Finding whether a given number is even or odd

Lex CODE:

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

%%

[0-9]+     { yylval = atoi(yytext); return NUMBER; }
[\n]       { return EOL; }
.          { return yytext[0]; }

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

YACC CODE:

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

%token NUMBER EOL

%%
input:
   NUMBER EOL {
      if ($1 % 2 == 0)
         printf("%d is even.\n", $1);
      else
         printf("%d is odd.\n", $1);
   }
   ;

%%
int main() {
   printf("Enter a number: ");
   yyparse();
   return 0;
}
void yyerror(const char *s) {
   printf("Error: %s\n", s);
}
```

13. Write a LEX-YACC specification program for Finding whether a given number is prime or not

Lex CODE:

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

%%

[0-9]+     { yylval = atoi(yytext); return NUMBER; }
[\n]       { return EOL; }
.          { return yytext[0]; }

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

YACC CODE

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


int isPrime(int n) {
    if (n < 2) return 0;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) return 0;
    return 1;
}
%}

%token NUMBER EOL

%%
input:
    NUMBER EOL {
        if (isPrime($1))
            printf("%d is prime.\n", $1);
        else
            printf("%d is not prime.\n", $1);
    }
    ;

%%
int main() {
    printf("Enter a number: ");
    yyparse();
    return 0;
}
void yyerror(const char *s) {
    printf("Error: %s\n", s);
}
```

12. Write a LEX-YACC specification program for Finding factorial of a number

Lex CODE:

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

%%

[0-9]+     { yylval = atoi(yytext); return NUMBER; }
[\n]       { return EOL; }
.          { return yytext[0]; }

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

YACC CODE

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

int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
%}

%token NUMBER EOL

%%
input:
    NUMBER EOL {
        printf("Factorial of %d is %d\n", $1, factorial($1));
    }
    ;

%%
int main() {
    printf("Enter a number: ");
    yyparse();
    return 0;
}
void yyerror(const char *s) {
    printf("Error: %s\n", s);
}
```

6. Write a LEX-YACC specification program for scientific calculator

Lex CODE:

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

%%

[0-9]+     { yylval = atoi(yytext); return NUMBER; }
[\n]       { return EOL; }
.          { return yytext[0]; }

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

YACC CODE

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

%token NUMBER EOL

%%
input:
    expr EOL {
        printf("Result = %d\n", $1);
    }
    ;

expr:
     expr '+' expr  { $$ = $1 + $3; }
    | expr '-' expr  { $$ = $1 - $3; }
    | expr '*' expr  { $$ = $1 * $3; }
    | expr '/' expr  {
        if ($3 == 0) {
            yyerror("Division by zero");
            $$ = 0;
        } else {
            $$ = $1 / $3;
        }
    }
    | NUMBER         { $$ = $1; }
    ;

%%
int main() {
    printf("Enter an expression: ");
    yyparse();
    return 0;
}
void yyerror(const char *s) {
    printf("Error: %s\n", s);
}
```

4. Define a macro with two arguments, expansion of macro calls & generating expanded source code.

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

// Macro definitions
#define ADD(a, b) ((a) + (b))
#define SUB(a, b) ((a) - (b))
#define MUL(a, b) ((a) * (b))
#define DIV(a, b) ((a) / (b))

int main() {
   char macroName[10];
   int arg1, arg2;

   printf("Available Macros: ADD(a, b), SUB(a, b), MUL(a, b), DIV(a, b)\n");
   printf("Enter Macro Name: ");
   scanf("%s", macroName);

   printf("Enter First Argument: ");
   scanf("%d", &arg1);

   printf("Enter Second Argument: ");
   scanf("%d", &arg2);

   printf("\nExpanded Source Code:\n");

   if (strcmp(macroName, "ADD") == 0) {
      printf("// Macro Call: ADD(%d, %d)\n", arg1, arg2);
      printf("int result = ((%d) + (%d));\n", arg1, arg2);
   } else if (strcmp(macroName, "SUB") == 0) {
      printf("// Macro Call: SUB(%d, %d)\n", arg1, arg2);
      printf("int result = ((%d) - (%d));\n", arg1, arg2);
   } else if (strcmp(macroName, "MUL") == 0) {
      printf("// Macro Call: MUL(%d, %d)\n", arg1, arg2);
      printf("int result = ((%d) * (%d));\n", arg1, arg2);
   } else if (strcmp(macroName, "DIV") == 0) {
      printf("// Macro Call: DIV(%d, %d)\n", arg1, arg2);
      if (arg2 == 0) {
         printf("// Error: Division by zero not allowed\n");
      } else {
         printf("int result = ((%d) / (%d));\n", arg1, arg2);
      }
   } else {
      printf("// Invalid macro name\n");
   }

   return 0;
}
```

3. Define a macro with multiple arguments, expansion of macro calls & generating expanded source code.

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

// Define macros with multiple arguments
#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define MIN(a, b) ((a) < (b) ? (a) : (b))
#define AVG(a, b, c) (((a) + (b) + (c)) / 3)

int main() {
    char macroName[10];
    int a, b, c;

    printf("Available Macros:\n");
    printf("1. MAX(a, b)\n");
    printf("2. MIN(a, b)\n");
    printf("3. AVG(a, b, c)\n\n");

    printf("Enter Macro Name: ");
    scanf("%s", macroName);

    printf("\n");

    // Macro expansion and simulation
    if (strcmp(macroName, "MAX") == 0) {
        printf("Enter two arguments:\n");
        scanf("%d %d", &a, &b);

        printf("\nExpanded Source Code:\n");
        printf("// Macro Call: MAX(%d, %d)\n", a, b);
        printf("int result = ((%d) > (%d) ? (%d) : (%d));\n", a, b, a, b);
        printf("Computed Result: %d\n", MAX(a, b));

    } else if (strcmp(macroName, "MIN") == 0) {
        printf("Enter two arguments:\n");
        scanf("%d %d", &a, &b);

        printf("\nExpanded Source Code:\n");
        printf("// Macro Call: MIN(%d, %d)\n", a, b);
        printf("int result = ((%d) < (%d) ? (%d) : (%d));\n", a, b, a, b);
        printf("Computed Result: %d\n", MIN(a, b));

    } else if (strcmp(macroName, "AVG") == 0) {
        printf("Enter three arguments:\n");
        scanf("%d %d %d", &a, &b, &c);

        printf("\nExpanded Source Code:\n");
        printf("// Macro Call: AVG(%d, %d, %d)\n", a, b, c);
        printf("int result = (((%d) + (%d) + (%d)) / 3);\n", a, b, c);
        printf("Computed Result: %d\n", AVG(a, b, c));

    } else {
        printf("// Error: Invalid macro name.\n");
    }

    return 0;
}
```

5. Implement a 2-level nested macro & its expansion.

Code:

```
#include <stdio.h>

// Define nested macros
#define SQUARE(x) ((x) * (x))
#define SUM_SQUARES(a, b) (SQUARE(a) + SQUARE(b))

int main() {
    int a, b;

    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);

    printf("\nNested Macro Call:\n");
    printf("SUM_SQUARES(%d, %d)\n", a, b);

    printf("\nExpanded Source Code:\n");
    printf("int result = (((%d) * (%d)) + ((%d) * (%d)));\n", a, a, b, b);

    int result = SUM_SQUARES(a, b);
    printf("\nComputed Result: %d\n", result);

    return 0;
}
```

8. Write LEX program to recognize identifiers of C language using symbol table

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

#define MAX 100

char *symtab[MAX];
int symcount = 0;

// Check if identifier is already in the symbol table
int is_present(char *id) {
    for (int i = 0; i < symcount; i++) {
        if (!strcmp(symtab[i], id))
            return 1;
    }
    return 0;
}
%}

ID [a-zA-Z_][a-zA-Z0-9_]*

%%

{ID} {
    if (!is_present(yytext)) {
        symtab[symcount++] = strdup(yytext);
        printf("ID: %s\n", yytext);
    }
}

.|\n    ;   // Ignore all other characters and newlines

%%

int main() {
    printf("Enter code (Ctrl+Z to end):\n");
    yylex();

    printf("Symbols (Total: %d):\n", symcount);
    for (int i = 0; i < symcount; i++) {
        printf("%d: %s\n", i + 1, symtab[i]);
    }

    return 0;
}

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