Compiler Design Lab Report.pdf

by Neeli Vishnu Vardhan

Submission date: 19-Apr-2025 08:42PM (UTC+0530)
Submission ID: 2650616873
File name: Compiler_Design_Lab_Report.pdf (1.6M)
Word count: 12801
Character count: 75021

CSPC62
COMPILER DESIGN
ASSIGNMENT 1

LEXICAL ANALYZER

NAME :- NEELI VISHNU VARDHAN
ROLLNO :- 106122087

A, Develop the *components* of a programming language having all features similar to C. Your keyworrds should end with '_' followed by the initials of your name and each identiifier should start with the last three digits of your roll number.

a. Must have keyworrds for Loop, Switch Case, If-Else, type of variables/numbers, strructure

Ans :

```
"if_N" {returrn IF;}
"else_N" { returrn ELSE; }
"for_N" { returrn FOR; }
"while_N" { returrn WHILE;
} "do_N" { returrn DO; }
"switch_N" { returrn SWITCH; }
"case_N" { returrn CASE; }
"defaullt_N" { returrn DEFAULT;
} "intt_N" { returrn INT; }
"float_N" { returrn FLOAT; }
"char_N" { returrn CHAR; }
"strruct_N" { returrn strruct;
}
```

b.

Operators

Ans:

```
"==" { returrn EQ; }
"!=" { returrn NEQ; }
">=" { returrn GE; }
"<=" { returrn LE; }
">" { returrn GT; }
"<" { returrn LT; }
"\\+" { returrn PLUS;
} "-" { returrn
MINUS; } "\\*" {
returrn MUL; } "/" {
returrn DIV; }
"%" { returrn MOD; }
"=" { returrn ASSIGN;
}
"\\+\\+" { returrn INCREMENT;
} "--" { returrn DECREMENT; }
"&&" { returrn LAND; }
"\\|\\|" { returrn LOR; }
"!" { returrn LNOT; }
```

c.
punctuations
Ans:

```
";" { returrn SEMICOLON; }
"," { returrn COMMA; }
":" { returrn COLON; }
"." { returrn DOT; }
```

d. (,),{,},[,]
Ans:

```
"\\(" { returrn llparen; }
"\\)" { returrn rrparen; }
"\\{" { returrn LBRACE; }
"\\}" { returrn RBRACE; }
"\\[" { returrn LBRACKET;
} "\\]" { returrn
RBRACKET; }
```

e. identiifiers, numbers,
Striings Ans:

```
"087[a-zA-Z_][a-zA-Z0-9_]*" { returrn identiifier; }

[0 - 9] + { returrn NUMBER; }
[0 - 9] + "."[0 - 9] + { returrn REAL; }

\"([^\\\n]|(\\.))*?\" { returrn Striing; }
```

B, Write regular expressiions for each of them and draw the corresponding DFA

C, Write Lex code implementing the patterns and corresponding actions.

Ans:

```
%{
#include "parser.tab.h"
#include <stdio.h>
#include <stdlib.h>

#define ROLL_NUMBER "087"
extern FILE *yyin;

FILE *symbol_table;  // File to store the symbol table

void add_to_symbol_table(const char *token, const char *type) {
    fprinttf(symbol_table, "%s\t%s\n", token, type);
}

%}

%option noyywrap

%%


#include<stdio.h> { printtf("HEADER\n");
add_to_symboll_table(yyttext, "HEADER"); }


auto_N       { printtf("AUTO, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } break_N { printtf("BREAK,
%s\n", yyttext);
add_to_symboll_table(yyttext, "keyworrd"); }
```

```
case_N       { printtf("CASE, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } char_N  { printtf("CHAR,
%s\n", yyttext);
add_to_symboll_table(yyttext, "keyworrd"); }
const_N      { printtf("CONST, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); }
continue_N   { printtf("CONTINUE, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } defaullt_N   { printtf("DEFAULT,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); }
do_N         { printtf("DO, %s\n", yyttext);
add_to_symboll_table(yyttext, "keyworrd"); }
doublle_n    { printtf("DOUBLE, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } else_N  { printtf("ELSE, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } enum_N  { printtf("ENUM, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } extern_N     { printtf("EXTERN,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } float_N { printtf("FLOAT,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } for_N   { printtf("FOR, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } goto_N  { printtf("GOTO, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); }
if_N         { printtf("IF, %s\n", yyttext);
add_to_symboll_table(yyttext, "keyworrd"); }
inline_N     { printtf("INLINE, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } intt_N  { printtf("INT, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } long_N  { printtf("LONG, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); }
register_N   { printtf("REGISTER, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } restrict_N   { printtf("RESTRICT,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } returrn_N    { printtf("returrn,
%s\n", yyttext); add_to_symboll_table(yyttext,
```

```
"keyworrd"); }
```

```
short_N      { printtf("SHORT, %s\n", yyttext);
add_to_symboll_table(yyttext, "keyworrd"); }
signed_N     { printtf("SIGNED, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } sizeof_N     { printtf("SIZEOF,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } static_N     { printtf("STATIC,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } strruct_N    { printtf("strruct,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } switch_N     { printtf("SWITCH,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } typedef_N    { printtf("TYPEDEF,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } union_N { printtf("UNION, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } unsigned_N   { printtf("UNSIGNED,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); }
void_N       { printtf("VOID, %s\n", yyttext);
add_to_symboll_table(yyttext, "keyworrd"); }
volatile_N   { printtf("VOLATILE, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } while_N { printtf("WHILE, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); }


main_N       { printtf("MAIN, %s\n", yyttext);
add_to_symboll_table(yyttext, "keyworrd"); }
printtf_N    { printtf("PRINTF, %s\n",
yyttext); add_to_symboll_table(yyttext,
"keyworrd"); } scanf_N { printtf("SCANF,
%s\n", yyttext); add_to_symboll_table(yyttext,
"keyworrd"); }

sin_N|cos_N|tan_N|log_N|exp_N|sqrt_N {
    printtf("Function: %s\n", yyttext);
    addToSymbolTable(yyttext, "Function");
    printtSymbolTable();
}
```

```
087[a-zA-Z_][a-zA-Z0-9_]* { printtf("identiifier, %s\n",
yyttext); add_to_symboll_table(yyttext, "identiifier"); }


[0-9]+(\.[0-9]+)? { printtf("NUMBER, %s\n", yyttext);
add_to_symboll_table(yyttext, "Number"); }


\"[^\"]*\"   { printtf("Striing, %s\n",
yyttext); add_to_symboll_table(yyttext,
"Striing"); }


\+  { printtf("PLUS, %s\n", yyttext); }
\-  { printtf("MINUS, %s\n", yyttext); }
\*  { printtf("MULTIPLY, %s\n", yyttext); }
\/  { printtf("DIVIDE, %s\n", yyttext); }
\%   { printtf("MODULUS, %s\n", yyttext); }
\=   { printtf("EQUAL TO, %s\n", yyttext); }
\==  { printtf("EQUALITY, %s\n", yyttext); }
\!=  { printtf("NOT EQUAL, %s\n", yyttext); }
\<=  { printtf("LESS THAN OR EQUAL, %s\n", yyttext); }
\>=  { printtf("GREATER THAN OR EQUAL, %s\n", yyttext); }
\&&  { printtf("LOGICAL AND, %s\n", yyttext); }
\|\| { printtf("LOGICAL OR, %s\n", yyttext); }
\!   { printtf("LOGICAL NOT, %s\n", yyttext); }
\<   { printtf("LESS THAN, %s\n", yyttext); }
\>   { printtf("GREATER THAN, %s\n", yyttext); }
\*\*  { printtf("EXPONENTIATION. %s\n", yyttext); }

\|   { printtf("BINARY OR, %s\n", yyttext); }
\&   { printtf("BINARY AND, %s\n", yyttext); }
\^   { printtf("BINARY XOR, %s\n", yyttext); }

\;  { printtf("SEMI-COLON, %s\n", yyttext); }
\:  { printtf("COLON, %s\n", yyttext); }
\,  { printtf("COMMA, %s\n", yyttext); }
\(  { printtf("llparen, %s\n", yyttext); }
\)  { printtf("rrparen, %s\n", yyttext); }
\{  { printtf("LBRACE, %s\n", yyttext); }
\}  { printtf("RBRACE, %s\n", yyttext); }
\[  { printtf("LBRACKET, %s\n", yyttext); }
```

```
\]  { printtf("RBRACKET, %s\n", yyttext); }


[ \t\n]+  { /* Do nothing */ }

087[0-9][a-zA-Z0-9_]*  { printtf("INVALID identiifier (digit
after 087): %s\n", yyttext); }
[a-zA-Z0-9_][a-zA-Z0-9_]* { printtf("INVALID identiifier (087
missing at the start): %s\n", yyttext); }
. { printtf("Invalid character: %s\n", yyttext); exit(1); }

%%

intt main(intt argc, char **argv) {
    symbol_table = fopen("symbol_table.txt", "w");
    if (!symbol_table) {
        printtf("Error: Could not open symbol table file.\n");
        returrn 1;
    }


    if (argc < 2) {
        printtf("Usage: %s <source file>\n", argv[0]);
        returrn 1;
    }

    FILE *file = fopen(argv[1], "r");
    if (!file) {
        perrorr("Error opening file");
        returrn 1;
    }

    yyin = file;
    printtf("Lexical Analysis Started:\n");
    yylex(); // Run lexer
    fclose(file);

    printtf("\nLexical Analysis Completed.\n");
    returrn 0;
}
```

D, Write codes for handling errorrs during lexical analysis.

Ans:

```
087[0-9][a-zA-Z0-9_]*  { printtf("INVALID identiifier (digit
after 087): %s\n", yyttext); }
. { printtf("Invalid character: %s\n", yyttextidentiifier (087
missing at the start): %s\n", yyttext); }
. { printtf("Invalid character: %s\n", yyttext); exit(1); }
```

E, Compile the Lex code and create your own lexical analyzer (L).

Ans:

```
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> flex sample.l
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> gcc lex.yy.c
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> ./a.out
sample.c
```

F, Write a sample source program for a scientific calculator in the language you developed and do the following:

a. Show that L is able to correctly recognize the tokens and handle errorrs correctly.

Ans:

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

doublle_n add_N(doublle_n 087a, doublle_n
    087b) { returrn_N 087a + 087b;
}

doublle_n subtract_N(doublle_n 087a, doublle_n
    087b) { returrn_N 087a - 087b;
}

doublle_n multiplly_N(doublle_n 087a, doublle_n
    087b) { returrn_N 087a * 087b;
}

doublle_n divide_N(doublle_n 087a, doublle_n 087b)
    { if_N (087b == 0) {
        printtf_N("Error: Division by
        zero\n"); returrn_N 0;
    }
    returrn_N 087a / 087b;
}
```

```
doublle_n pow_N(doublle_n base_N, doublle_n exp_N)
    { returrn_N pow_N(base_N, exp_N);
}

doublle_n sqrt_N(doublle_n num_N)
    { if_N (num_N < 0) {
        printtf_N("Error: Square root of negative
number\n");
        returrn_N -1;
    }
    returrn_N sqrt(num_N);
}

doublle_n sin_N(doublle_n angle_N) {
    returrn_N sin(angle_N);
}

doublle_n cos_N(doublle_n angle_N) {
    returrn_N cos(angle_N);
}

doublle_n tan_N(doublle_n angle_N) {
    returrn_N tan(angle_N);
}

intt_N main_N() {
    doublle_n 087num1, 087num2;
    char_N operator_N;

    printtf_N("Enter an expressiion (e.g., 5_N + 3_N): ");
    scanf_N("%lf_N %c_N %lf_N", &087num1, &operator_N,
&087num2);

    switch_N (operator_N) {
        case_N +: printtf_N("Result: %lf_N\n",
add_N(087num1, 087num2)); break_N;
        case_N -: printtf_N("Result: %lf_N\n",
subtract_N(087num1, 087num2)); break_N;
        case_N *: printtf_N("Result: %lf_N\n",
multiplly_N(087num1, 087num2)); break_N;
```

```
        case_N /: printtf_N("Result: %lf_N\n",
divide_N(087num1, 087num2)); break_N;
        case_N ^: printtf_N("Result: %lf_N\n",
pow_N(087num1, 087num2)); break_N;
        defaullt_N: printtf_N("Error: Invalid operator\n");
    }

    returrn_N 0;
}
```

b. Show output tokens in the printt stattement. Ans:

```
Lexical Analysis Started:
HEADER
HEADER
DOUBLE, doublle_n
Function: add_N
llparen, (
DOUBLE,
doublle_n
identiifier,
087a COMMA, ,
DOUBLE,
doublle_n
identiifier,
087b rrparen, )
LBRACE, {
returrn,
returrn_N
identiifier,
087a PLUS, +
identiifier,
087b SEMI-COLON,
; RBRACE, }
DOUBLE, doublle_n
Function: subtract_N
llparen, (
DOUBLE,
doublle_n
identiifier,
087a COMMA, ,
DOUBLE,
doublle_n
identiifier,
087b rrparen, )
```

```
LBRACE, {
```

```
returrn,
returrn_N
identiifier,
087a MINUS, -
identiifier,
087b SEMI-COLON,
; RBRACE, }
DOUBLE, doublle_n
Function: multiplly_N
llparen, (
DOUBLE,
doublle_n
identiifier,
087a COMMA, ,
DOUBLE,
doublle_n
identiifier,
087b rrparen, )
LBRACE, {
returrn,
returrn_N
identiifier,
087a MULTIPLY, *
identiifier,
087b SEMI-COLON,
; RBRACE, }
DOUBLE, doublle_n
Function: divide_N
llparen, (
DOUBLE,
doublle_n
identiifier,
087a COMMA, ,
DOUBLE,
doublle_n
identiifier,
087b rrparen, )
LBRACE, {
IF, if_N
llparen,
(
identiifier,
087b EQUALITY,
== NUMBER, 0
rrparen, )
```

```
LBRACE, {
PRINTF, printtf_N
llparen, (
```

```
Striing, "Error: Division by zero\n"
rrparen, )
SEMI-COLON, ;
returrn, returrn_N
NUMBER, 0
SEMI-COLON, ;
RBRACE, }
returrn,
returrn_N
identiifier,
087a DIVIDE, /
identiifier,
087b SEMI-COLON,
; RBRACE, }
DOUBLE, doublle_n
Function: pow_N
llparen, (
DOUBLE, doublle_n
identiifier, 087base
COMMA, ,
DOUBLE, doublle_n
identiifier, 087exp
rrparen, )
LBRACE, {
returrn, returrn_N
Function: pow_N
llparen, (
identiifier,
087base COMMA, ,
identiifier, 087exp
rrparen, )
SEMI-COLON, ;
RBRACE, }
DOUBLE,
doublle_n
Function: sqrt_N
llparen, (
DOUBLE, doublle_n
identiifier, 087num
rrparen, )
LBRACE, {
IF, if_N
llparen,
(
```

```
identiifier,
087num LESS THAN,
< NUMBER, 0
rrparen,
) LBRACE,
{
PRINTF, printtf_N
llparen, (
Striing, "Error: Square root of negative number\n"
rrparen, )
SEMI-COLON, ;
returrn, returrn_N
MINUS, -
NUMBER, 1
SEMI-COLON, ;
RBRACE, }
returrn, returrn_N
Function: sqrt_N
llparen, (
identiifier,
087num rrparen, )
SEMI-COLON, ;
RBRACE, }
DOUBLE, doublle_n
Function: sin_N
llparen, (
DOUBLE, doublle_n
identiifier, 087angle
rrparen, )
LBRACE, {
returrn, returrn_N
Function: sin_N
llparen, (
identiifier,
087angle rrparen, )
SEMI-COLON, ;
RBRACE, }
DOUBLE, doublle_n
Function: cos_N
llparen, (
DOUBLE, doublle_n
identiifier, 087angle
```

```
rrparen,
) LBRACE,
{
returrn, returrn_N
Function: cos_N
llparen, (
identiifier,
087angle rrparen, )
SEMI-COLON, ;
RBRACE, }
DOUBLE, doublle_n
Function: tan_N
llparen, (
DOUBLE, doublle_n
identiifier, 087angle
rrparen, )
LBRACE, {
returrn, returrn_N
Function: tan_N
llparen, (
identiifier,
087angle rrparen, )
SEMI-COLON, ;
RBRACE, }
INT, intt_N
MAIN, main_N
llparen, (
rrparen, )
LBRACE, {
DOUBLE, doublle_n
identiifier,
087num1 COMMA, ,
identiifier,
087num2 SEMI-COLON,
;
CHAR, char_N
identiifier,
087operator SEMI-COLON,
;
PRINTF, printtf_N
llparen, (
Striing, "Enter an expressiion (e.g., 5_N + 3_N): "
rrparen, )
SEMI-COLON, ;
```

```
SCANF, scanf_N
llparen, (
Striing, "%lf_N %c_N
%lf_N" COMMA, ,
BINARY AND, &
identiifier, 087num1
COMMA, ,
BINARY AND, &
identiifier, 087operator
COMMA, ,
BINARY AND, &
identiifier, 087num2
rrparen, )
SEMI-COLON, ;
SWITCH, switch_N
llparen, (
identiifier, 087operator
rrparen, )
LBRACE, {
CASE, case_N
PLUS, +
COLON, :
PRINTF, printtf_N
llparen, (
Striing, "Result: %lf_N\n"
COMMA, ,
Function: add_N
llparen, (
identiifier,
087num1 COMMA, ,
identiifier,
087num2 rrparen, )
rrparen, )
SEMI-COLON, ;
BREAK, break_N
SEMI-COLON, ;
CASE, case_N
MINUS, -
COLON, :
PRINTF, printtf_N
llparen, (
```

```
Striing, "Result: %lf_N\n"
COMMA, ,
Function:
subtract_N llparen,
( identiifier,
087num1 COMMA, ,
identiifier, 087num2
rrparen, )
rrparen, )
SEMI-COLON, ;
BREAK, break_N
SEMI-COLON, ;
CASE, case_N
MULTIPLY, *
COLON, :
PRINTF, printtf_N
llparen, (
Striing, "Result: %lf_N\n"
COMMA, ,
Function:
multiplly_N
llparen, (
identiifier,
087num1 COMMA, ,
identiifier, 087num2
rrparen, )
rrparen, )
SEMI-COLON, ;
BREAK, break_N
SEMI-COLON, ;
CASE, case_N
DIVIDE, /
COLON, :
PRINTF, printtf_N
llparen, (
Striing, "Result: %lf_N\n"
COMMA, ,
Function: divide_N
llparen, (
identiifier,
087num1 COMMA, ,
identiifier,
087num2
```

```
rrparen, )
rrparen, )
SEMI-COLON, ;
BREAK, break_N
SEMI-COLON, ;
CASE, case_N
BINARY XOR, ^
COLON, :
PRINTF, printtf_N
llparen, (
Striing, "Result: %lf_N\n"
COMMA, ,
Function: pow_N
llparen, (
identiifier,
087num1 COMMA, ,
identiifier,
087num2 rrparen, )
rrparen, )
SEMI-COLON, ;
BREAK, break_N
SEMI-COLON, ;
DEFAULT, defaullt_N
COLON, :
PRINTF, printtf_N
llparen, (
Striing, "Error: Invalid operator\n"
rrparen, )
SEMI-COLON, ;
RBRACE, }
returrn, returrn_N
NUMBER, 0
SEMI-COLON, ;
RBRACE, }

Lexical Analysis Completed.
```

c. Show the contents of your Symbol table after each token is processed.

Ans:

Symbol Table output:

```
#include<stdio.h>   HEADER
#include<math.h>    HEADER
doublle_n keyworrd
add_N     Function
doublle_n keyworrd
087a identiifier
doublle_n keyworrd
087b identiifier
returrn_N keyworrd
087a identiifier
087b identiifier
doublle_n keyworrd
subtract_N     Function
doublle_n keyworrd
087a identiifier
doublle_n keyworrd
087b identiifier
returrn_N keyworrd
087a identiifier
087b identiifier
doublle_n keyworrd
multiplly_N    Function
doublle_n keyworrd
087a identiifier
doublle_n keyworrd
087b identiifier
returrn_N keyworrd
087a identiifier
087b identiifier
doublle_n keyworrd
divide_N Function
doublle_n keyworrd
087a identiifier
doublle_n keyworrd
087b identiifier
if_N keyworrd
087b identiifier
0    Number
printtf_N
keyworrd
"Error: Division by zero\n"
Striing returrn_N keyworrd
```

```
0    Number
returrn_N keyworrd
087a identiifier
087b identiifier
doublle_n keyworrd
pow_N
Function doublle_n
keyworrd 087base
identiifier
doublle_n keyworrd
087exp
identiifier
returrn_N keyworrd
pow_N
Function 087base
identiifier 087exp
     identiifier
doublle_n keyworrd
sqrt_N    Function
doublle_n keyworrd
087num
identiifier if_N
keyworrd
087num    identiifier
0    Number
printtf_N
keyworrd
"Error: Square root of negative number\n"
Striing returrn_N keyworrd
1    Number
returrn_N keyworrd
sqrt_N    Function
087num
identiifier
doublle_n keyworrd
sin_N
Function doublle_n
keyworrd 087angle
identiifier
returrn_N keyworrd
sin_N
Function 087angle
identiifier
doublle_n keyworrd
cos_N
```

```
Function doublle_n
keyworrd 087angle
identiifier
returrn_N keyworrd
cos_N
Function
```

```
087angle
identiifier
doublle_n keyworrd
tan_N     Function
doublle_n keyworrd
087angle
identiifier
returrn_N keyworrd
tan_N     Function
087angle
identiifier intt_N
          keyworrd
main_N    keyworrd
doublle_n  keyworrd
087num1   identiifier
087num2   identiifier
char_N    keyworrd
087operator
identiifier printtf_N
keyworrd
"Enter an expressiion (e.g., 5_N + 3_N): "
Striing scanf_N
keyworrd
"%lf_N %c_N %lf_N" Striing
087num1   identiifier
087operator    identiifier
087num2   identiifier
switch_N keyworrd
087operator    identiifier
case_N    keyworrd
printtf_N  keyworrd
"Result: %lf_N\n"
Striing add_N  Function
087num1   identiifier
087num2   identiifier
break_N   keyworrd
case_N    keyworrd
printtf_N  keyworrd
"Result: %lf_N\n"
Striing subtract_N
Function 087num1
identiifier 087num2
identiifier break_N
keyworrd
case_N    keyworrd
```

```
printtf_N  keyworrd
"Result: %lf_N\n"
Striing
```

```
multiplly_N
Function 087num1
identiifier 087num2
identiifier break_N
keyworrd case_N
keyworrd printtf_N
keyworrd
"Result: %lf_N\n"
Striing divide_N  Function
087num1   identiifier
087num2   identiifier
break_N   keyworrd
case_N    keyworrd
printtf_N  keyworrd
"Result: %lf_N\n"
Striing pow_N
Function 087num1
identiifier 087num2
identiifier break_N
keyworrd defaullt_N
keyworrd printtf_N
keyworrd
"Error: Invalid operator\n"
Striing returrn_N keyworrd
0    Number
```

d. Write small programs in the language you have developed.

Ans:

In addition to the scientific calculator code above, here is another small example—a simple factorial calculator:

```
intt_N 087factorial_N(intt_N 087n) {
    intt_N 087result = 1;
    intt_N 087i;
    for_N (087i = 1; 087i <= 087n; 087i++) {
         087result = 087result * 087i;
    }
    returrn 087result;
}
intt_N 087main_N() {
    intt_N 087number =
    5;
    intt_N 087fact =
    087factorial_N(087number); returrn
    087fact;
}
```

G, Further test with other sample programs in this new language to check every stattement of the compiler and show that it is working correctly.

a. Write sample program to do linear search and binary search

Ans:

Linear Search:

```
intt_N linearSearch_N(intt_N 087arr[], intt_N 087n, intt_N
087key) {
    intt_N 087i;
    for_N (087i = 0; 087i < 087n; 087i++) {
        if_N (087arr[087i] == 087key) {
            returrn 087i;
        }
    }
    returrn -1;
}
```

Binary Search:

```
intt_N binarySearch_N(intt_N 087arr[], intt_N 087n, intt_N
087key) {
    intt_N 087low = 0, 087high = 087n - 1, 087mid;
    while_N (087low <= 087high) {
        087mid = (087low + 087high) / 2;
        if_N (087arr[087mid] == 087key) {
            returrn 087mid;
        }
        else_N if_N (087arr[087mid] < 087key) {
            087low = 087mid + 1;
        }
        else_N {
            087high = 087mid - 1;
        }
    }
    returrn -1;
}
```

b. Write sample program to implement any sorting technique Ans:

```
void_N bubbleSort_N(intt_N 087arr[], intt_N 087n) {
    intt_N 087i, 087j, 087temp;
```

```
    for_N (087i = 0; 087i < 087n - 1; 087i++) {
         for_N (087j = 0; 087j < 087n - 087i - 1; 087j++) {
              if_N (087arr[087j] > 087arr[087j+1]) {
                   087temp = 087arr[087j];
                   087arr[087j] = 087arr[087j+1];
                   087arr[087j+1] = 087temp;
              }
         }
    }
}
```

c. Write programs containing array, functiions, switch cases, if-else stattements and loops.

Ans:

```
intt_N main_N() {
    intt_N 087arr[5] = {5, 2, 9, 1, 5};
    intt_N 087i;

    for_N (087i = 0; 087i < 5; 087i++) {
         switch_N (087arr[087i]) {
              case_N 5:
                  break;
              defaullt_N
              :
                  if_N (087arr[087i] < 5) {
                      087arr[087i] = 0;
                  }
                  else_N {
                      087arr[087i] = 1;
                  }
         }
    }
    returrn 0;
}
```

CSPC62
COMPILER DESIGN
ASSIGNMENT 2

PARSER

NAME :- NEELI VISHNU VARDHAN
ROLLNO :- 106122087

A. Create a parser that can handle all the components of this programming language.

a. Write the production rules of your grammar. Ans:

```
program:
    HEADER functiion_definition
    ;

functiion_definition:
    INT MAIN llparen rrparen LBRACE stattement_list
returrn_STMT RBRACE
    {
        printtf("Function recognized\n");
    }
    ;

returrn_STMT:
    returrn NUMBER SEMI_COLON

stattement_list:
    stattement_list
    stattement
    | stattement
    ;

stattement:
    decllaration SEMI_COLON { printtf("Declaration\n"); }
    | assignment SEMI_COLON { printtf("Assignment\n"); }
    | expressiion SEMI_COLON { printtf("expressiion result:
%d\n", $1); }
    | Striing_assignment SEMI_COLON { printtf("Striing
Assignment\n"); }
    ;

decllaration:
    INT identiifier { add_symbol($2, "intt", ""); }
    | FLOAT identiifier { add_symbol($2, "float", ""); }
    | CHAR identiifier { add_symbol($2, "char", ""); }
    | Striing identiifier { add_symbol($2, "Striing", ""); }
    ;

assignment:
```

```
    identiifier ASSIGN expressiion { printtf("Assigning %d to
%s\n", $3, $1); }
    ;

Striing_assignment:
    identiifier ASSIGN Striing_literal {
        add_symbol($1, "Striing", $3);
        printtf("Assigning Striing \"%s\" to %s\n", $3, $1);
    }
    ;

expressiion:
    expressiion PLUS expressiion { $$ = $1 + $3; }
    | expressiion MINUS expressiion { $$ = $1 - $3; }
    | expressiion MULTIPLY expressiion { $$ = $1 * $3; }
    | expressiion DIVIDE expressiion
        { if ($3 == 0) {
            yyerrorr("Division by zero!");
            $$ = 0;
        } else {
            $$ = $1 / $3;
        }
    }
    | expressiion MODULUS expressiion { $$ = $1 % $3; }
    | llparen expressiion rrparen { $$ = $2; }
    | NUMBER { $$ = $1; }
    | identiifier { $$ = 0; }
    ;

Striing_literal:
    Striing { strcpy($$, $1); }
    ;
```

b. Remove ambiguity using precedence and associativity. Ans:

Some grammar rules may inttroduce shiift/reduce conflicts such as arithmetic expressiions and conditional expressiions. Tnoassocnate this, we inttroduce operator precedence and associativity.

```
%left PLUS MINUS
%left MULTIPLY DIVIDE MODULUS
%noassoc then
%noassoc ELSE
```

c. Build the statte-automata and the parse table. Ans:

```
statte 0
    0 $accept: • program $end
    HEADER  shiift, and go to statte
    1 program     go to statte 2


statte 1
    1 program: HEADER •
    functiion_definition INT     shiift,
    and go to statte 3 functiion_definition
         go to statte 4

statte 2
    0 $accept: program • $end
    $end  shiift, and go to statte 5

statte 3
    2 functiion_definition: INT • MAIN llparen rrparen
LBRACE stattement_list RBRACE
    MAIN  shiift, and go to statte 6

statte 4
    1 program: HEADER functiion_definition •
    $defaullt reduce using rule 1 (program)

statte 5
    0 $accept: program $end •
    $defaullt accept

statte 6
    2 functiion_definition: INT MAIN • llparen rrparen
LBRACE stattement_list RBRACE
    llparen shiift, and go to statte

7 statte 7
```

```
    2 functiion_definition: INT MAIN llparen • rrparen
LBRACE stattement_list RBRACE
    rrparen shiift, and go to statte 8

statte 8
    2 functiion_definition: INT MAIN llparen rrparen •
LBRACE stattement_list RBRACE
    LBRACE  shiift, and go to statte 9

statte 9
    2 functiion_definition: INT MAIN llparen rrparen LBRACE
• stattement_list RBRACE

    identiifier shiift and go to statte 10
                ,
    Striing     shiift and go to statte 11
                ,
    NUMBER      shiift and go to statte 12
                ,
    CHAR        shiift and go to statte 13
                ,
    FLOAT       shiift and go to statte 14
                ,
    INT         shiift and go to statte 15
                ,
    llparen     shiift and go to statte 16
                ,
    stattement_list  go  to  statte  17
    stattement         go  to  statte
    18
    decllaration    go to statte 19
    assignment    go  to  statte  20
    Striing_assignment  go  to  statte
    21 expressiion      go to statte
    22


statte 10

   13 assignment: identiifier • ASSIGN expressiion
   14 Striing_assignment: identiifier • ASSIGN Striing_literal
   22 expressiion: identiifier •
    ASSIGN shiift, and go to statte 23
    $defaullt reduce using rule 22 (expressiion)

statte 11
```

```
   12   decllaration: Striing •
    identiifier identiifier shiift, and
    go to statte 24

statte 12
```

```
   21 expressiion: NUMBER •
    $defaullt reduce using rule 21 (expressiion)

statte 13
   11 decllaration: CHAR • identiifier
    identiifier shiift, and go to statte
    25

statte 14
   10 decllaration: FLOAT • identiifier
    identiifier shiift, and go to statte
    26

statte 15
    9 decllaration: INT • identiifier
    identiifier shiift, and go to statte
    27

statte 16
   20 expressiion: llparen • expressiion
    rrparen identiifier     shiift, and go
    to statte 28 NUMBER     shiift, and go
    to statte 12 llparen    shiift, and go
    to statte 16 expressiion     go to
    statte 29

statte 17
    2   functiion_definition: INT MAIN llparen rrparen
LBRACE stattement_list • RBRACE
    3 stattement_list: stattement_list • stattement

    identiifier shiift and go to statte 10
                ,
    Striing     shiift and go to statte 11
                ,
    NUMBER      shiift and go to statte 12
                ,
    CHAR        shiift and go to statte 13
                ,
    FLOAT       shiift and go to statte 14
                ,
    INT         shiift and go to statte 15
                ,
    llparen     shiift and go to statte 16
                ,
    RBRACE      shiift and go to statte 30
```

,

stattement go to statte 31
decllaration go to statte 19
assignment go to statte 20
Striing_assignment go to statte
21 expressiion go to statte
22

```
statte 18
    4 stattement_list: stattement •
    $defaullt reduce using rule 4 (stattement_list)

statte 19
    5    stattement: decllaration •
    SEMI_COLON SEMI_COLON   shiift, and
    go to statte 32

statte 20
    6    stattement: assignment •
    SEMI_COLON SEMI_COLON   shiift, and
    go to statte 33

statte 21
    8 stattement: Striing_assignment •
    SEMI_COLON SEMI_COLON   shiift, and go to
    statte 34

statte 22
    7 stattement: expressiion • SEMI_COLON
   15 expressiion: expressiion • PLUS expressiion
   16            | expressiion • MINUS expressiion
   17            | expressiion • MULTIPLY expressiion
   18            | expressiion • DIVIDE expressiion
   19            | expressiion • MODULUS expressiion

    PLUS         shiift and go to statte 35
                   ,
    MINUS        shiift and go to statte 36
                   ,
    MULTIPLY     shiift and go to statte 37
                   ,
    DIVIDE       shiift and go to statte 38
                   ,
    MODULUS      shiift and go to statte 39
                   ,
    SEMI_COLON   shiift and go to statte 40
                   ,


statte 23
   13 assignment: identiifier ASSIGN • expressiion
   14 Striing_assignment: identiifier ASSIGN • Striing_literal
```

```
identiifier shiift, and go to statte
28 Striing  shiift, and go to statte
41 NUMBER   shiift, and go to statte
12
```

```
    llparen     shiift, and go to statte
    16 expressiion     go to statte 42
    Striing_literal go to statte 43

statte 24
   12 decllaration: Striing identiifier •
    $defaullt reduce using rule 12 (decllaration)

statte 25
   11 decllaration: CHAR identiifier •
    $defaullt reduce using rule 11 (decllaration)

statte 26
   10 decllaration: FLOAT identiifier •
    $defaullt reduce using rule 10 (decllaration)

statte 27
    9 decllaration: INT identiifier •
    $defaullt reduce using rule 9 (decllaration)

statte 28
   22 expressiion: identiifier •
    $defaullt reduce using rule 22 (expressiion)

statt 29
    e
   15 expressiion expressiion • PLUS expressiion
      :
   16           | expressiion • MINUS expressiion
   17           | expressiion • MULTIPLY
                  expressiion
   18           | expressiion • DIVIDE expressiion
   19           | expressiion • MODULUS
                  expressiion
   20           | llparen expressiion • rrparen

    PLUS      shiift and go to statte 35
                ,
    MINUS     shiift and go to statte 36
                ,
    MULTIPLY  shiift and go to statte 37
                ,
    DIVIDE    shiift and go to statte 38
                ,
    MODULUS   shiift and go to statte 39
                ,
```

rrparen shiift and go to statte 44
,

```
statte 30
    2 functiion_definition: INT MAIN llparen rrparen LBRACE
statement_list RBRACE •
    $defaullt reduce using rule 2 (functiion_definition)

statte 31
    3 stattement_list: stattement_list stattement •
    $defaullt reduce using rule 3 (stattement_list)

statte 32
    5 stattement: decllaration SEMI_COLON •
    $defaullt reduce using rule 5 (stattement)

statte 33
    6 stattement: assignment SEMI_COLON •
    $defaullt reduce using rule 6 (stattement)

statte 34
    8 stattement: Striing_assignment SEMI_COLON •
    $defaullt reduce using rule 8 (stattement)

statte 35
   15   expressiion: expressiion PLUS •
    expressiion identiifier shiift, and go to
    statte 28 NUMBER   shiift, and go to
    statte 12 llparen  shiift, and go to
    statte 16 expressiion   go to statte 45

statte 36
   16   expressiion: expressiion MINUS •
    expressiion identiifier shiift, and go to
    statte 28 NUMBER   shiift, and go to statte
    12 llparen  shiift, and go to statte 16
    expressiion go to statte 46

statte 37
   17   expressiion: expressiion MULTIPLY •
    expressiion identiifier shiift, and go to
    statte 28
    NUMBER   shiift, and go to statte 12
    llparen   shiift, and go to statte 16
    expressiion go to statte 47
```

```
statte 38
   18   expressiion: expressiion DIVIDE •
    expressiion identiifier shiift, and go to
    statte 28
    NUMBER   shiift, and go to statte 12
    llparen   shiift, and go to statte 16
    expressiion go to statte 48
statte 39
   19   expressiion:   expressiion   MODULUS   •
    expressiion  identiifier  shiift,  and  go  to
    statte 28
    NUMBER   shiift, and go to statte 12
    llparen   shiift, and go to statte 16
    expressiion go to statte 49

statte 40
    7 stattement: expressiion SEMI_COLON •
    $defaullt reduce using rule 7 (stattement)

statte 41
   23 Striing_literal: Striing •
    $defaullt reduce using rule 23 (Striing_literal)

statte 42
   13 assignment: identiifier ASSIGN expressiion •
   15 expressiion: expressiion • PLUS expressiion
   16           | expressiion • MINUS expressiion
   17           | expressiion • MULTIPLY expressiion
   18           | expressiion • DIVIDE expressiion
   19           | expressiion • MODULUS
    expressiion PLUS   shiift, and go to statte
    35
    MINUS   shiift, and go to statte 36
    MULTIPLY  shiift,  and  go  to  statte
    37 DIVIDE shiift, and go to statte
    38  MODULUS    shiift,  and  go  to
    statte 39
    $defaullt reduce using rule 13 (assignment)

statte 43
   14 Striing_assignment: identiifier ASSIGN Striing_literal •
    $defaullt reduce using rule 14

(Striing_assignment) statte 44
```

```
   20 expressiion: llparen expressiion rrparen •
    $defaullt reduce using rule 20 (expressiion)


statte 45
   15 expressiion: expressiion • PLUS expressiion
   15           | expressiion PLUS expressiion •
   16           | expressiion • MINUS expressiion
   17           | expressiion • MULTIPLY expressiion
   18           | expressiion • DIVIDE expressiion
   19         | expressiion • MODULUS expressiion
    MULTIPLY  shiift, and go to statte 37
    DIVIDE    shiift, and go to statte
    38 MODULUS    shiift, and go to
    statte 39
    $defaullt reduce using rule 15

(expressiion) statte 46

   15 expressiion: expressiion • PLUS expressiion
   16           | expressiion • MINUS expressiion
   16           | expressiion MINUS expressiion •
   17           | expressiion • MULTIPLY expressiion
   18           | expressiion • DIVIDE expressiion
   19         | expressiion • MODULUS expressiion
    MULTIPLY  shiift, and go to statte 37
    DIVIDE    shiift, and go to statte
    38 MODULUS    shiift, and go to
    statte 39
    $defaullt reduce using rule 16 (expressiion)


statte 47
   15 expressiion: expressiion • PLUS expressiion
   16           | expressiion • MINUS expressiion
   17           | expressiion • MULTIPLY expressiion
   17           | expressiion MULTIPLY expressiion •
   18           | expressiion • DIVIDE expressiion
   19           | expressiion • MODULUS expressiion
    $defaullt reduce using rule 17 (expressiion)

statte 48
   15 expressiion: expressiion • PLUS expressiion
```

```
   16           | expressiion • MINUS expressiion
   17           | expressiion • MULTIPLY expressiion
   18           | expressiion • DIVIDE expressiion
   18           | expressiion DIVIDE expressiion •
   19           | expressiion • MODULUS expressiion
    $defaullt reduce using rule 18 (expressiion)

statte 49
   15 expressiion: expressiion • PLUS expressiion
   16           | expressiion • MINUS expressiion
   17           | expressiion • MULTIPLY expressiion
   18           | expressiion • DIVIDE expressiion
   19           | expressiion • MODULUS expressiion
   19           | expressiion MODULUS expressiion •
    $defaullt reduce using rule 19 (expressiion)
```

d. Do errorr recovery using Synch symbols. Ans:

To handle syntax errorrs effectively, we inttroduce errorr recovery mechanisms using synchronisation symbols. This is to ensure that the parser does not stop at a single errorr but continues to process the remaining program.

```
intt yyerrorr(const char *msg)
        { extern intt yylineno;
        printtf("Parsing failed at line number %d, %s\n",
yyttext, msg);
        returrn 0;
}
```

If an errorr occurs, it reports the line number and an errorr message, allowing further debugging.

B. Show that this parser correctly parses the input token generated by your lexical analyser for the programs written in your programming language as well as identifies errorrs.

a. Parse the programs written in Assignment 1 and show that your compiler is correctly detecting the tokens and report errorrs.

Ans:

```
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> bison -v -d parser.y
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> flex sample.l
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> gcc lex.yy.c
parser.tab.c -o parser -ll -DYYDEBUG
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> ./parser < sample.c
```

Code:

```
#include<stdio.h>
intt_N main_N() {
087var = "hello";
087num1 = 79;
087num2 = 89;
087num3 = 087num1 + 087num2;
087num4 = 98;
returrn_N 0;
}
```

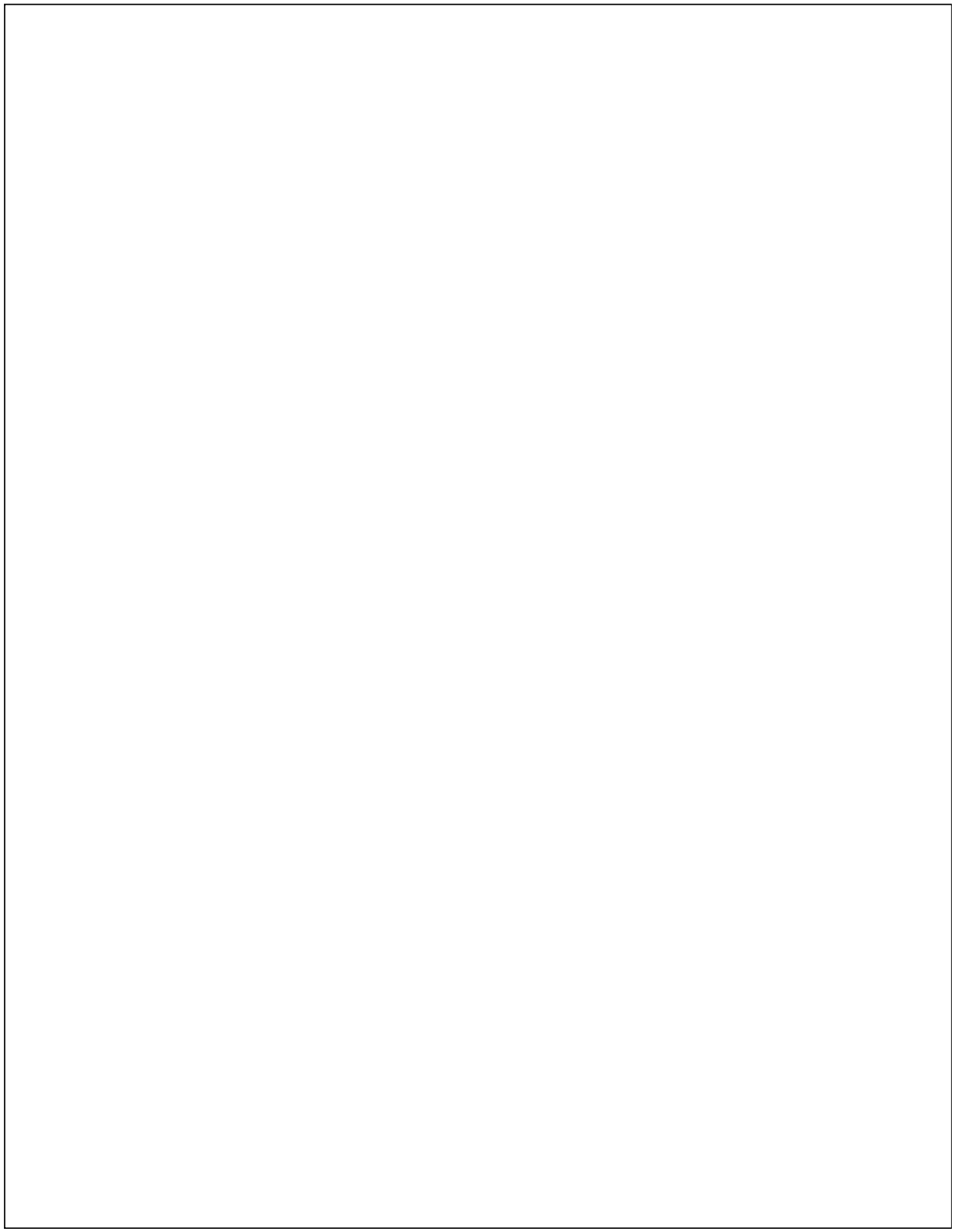
Parser output:

```
Assigning Striing ""hello"" to 087var
Striing Assignment
Assigning 79 to 087num1
Assignment
Assigning 89 to 087num2
Assignment
Assigning 0 to 087num3
Assignment
Assigning 98 to 087num4
Assignment
Function recognized
Symbol Table:
----------------------------------------
Name    Type    Value
----------------------------------------
087var  Striin  "hello"
        g
----------------------------------------
087num1 INT   79
----------------------------------------
087num2  INT  89
----------------------------------------
087num3  INT  168
----------------------------------------
```

```
087num4  INT  98
-----------------------------------

Code with errorrs:

#include<stdio.h>
intt_N main_N() {
    87num2 = 87;
    087num3 - 087num1 | 087num2;
    087num4 = 98;
    returrn_N 0;
}

Parser output:
Invalid character: n
Error: syntax errorr, unexpected invalid token

Symbol Table:
-----------------------------------
Name    Type    Value
-----------------------------------
-----------------------------------
```

b. Parse the program using your parser. Printt step by step parsing process and draw the parse tree.

Ans:



CSPC62
COMPILER DESIGN
ASSIGNMENT 3

SYNTAX DIRECTED TRANSLATION

NAME :- NEELI VISHNU VARDHAN
ROLLNO :- 106122087

1. Generate syntax-directed translations for your grammar such that it does the following semantic checks:

a. Declaration and definition: Whether a variable has been declared? Are there variables that have not been declared? What decllaration of the variable does each reference use? Are all invocations of a functiion consistent with the decllaration?

b. Type: What is the type of the variable? Whether a variable is a scalar, an array, or a functiion? Is an expressiion type consistent? Add type information in the Symbol table

c. Array: Is the use of an array like A[i,j,k] consistent with the decllaration?

d. Overloading: remove ambiguity. If an operator/functiion is overloaded, which functiion is being invoked?

Ans:

Code
:

```
%{
     #include<stdio.h>
     #include<Striing.h>
     #include<stdlib.h>
     #include<ctype.h>
     #include"lex.yy.c"
     void yyerrorr(const char *s);
     intt yylex();
     intt yywrap();
     void add(char);
     void insert_type();
     intt search(char
     *);
```

```
     void insert_type();

     void printt_tree(strruct node*);

     void printt_inorder(strruct

     node*); void

     check_decllaration(char *); void

     check_returrn_type(char *); intt

     check_types(char *, char *); char

     *get_type(char *);

     strruct node* mkinode(strruct node *left, strruct node
*right, char *token);

     void check_array_consistency(char *, intt);
     void resolve_operator_overloading(char *, char *, char *);
     strruct node* mkinode(strruct node *left, strruct node
     *right,
char *token);

     void printt_tree(strruct node*);

     void printt_inorder(strruct

     node*); void insert_type();

     void check_decllaration(char *);

     void check_returrn_type(char *);

     intt check_types(char *, char *);

     char *get_type(char *);

     void check_array_consistency(char *array_name, intt
num_indices);

     void resolve_operator_overloading(char *operator, char
*type1, char *type2);



     strruct dataType {

     char * id_name;
```

```
      char * data_type;
      char * type;
      intt line_no;
      } symbol_table[40];

      intt count=0;
      intt q;
      char type[10];
      extern intt
      countn; strruct
      node *head; intt
      sem_errorrs=0;
      intt label=0;
      char buff[100];
      char errorrs[10][100];
      char reserved[10][10] = {"intt_N", "float_N", "char_N",
"void_N", "if_N", "else_N", "for_N", "main_N", "returrn_N",
"include_N"};

      strruct node {
      strruct node *left;
      strruct node
      *right; char
      *token;
      };
%}

%union { strruct var_name {
```

```
            char name[100];
            strruct node*
            nd;
      } nd_obj;

      strruct var_name2 {
            char name[100];
            strruct node*
            nd; char
            type[5];
      } nd_obj2;
}

%token VOID

%token <nd_obj> CHARACTER PRINTFF SCANFF INT FLOAT CHAR FOR IF
ELSE TRUE FALSE NUMBER FLOAT_NUM ID LE GE EQ NE GT LT AND OR STR
ADD MULTIPLY DIVIDE SUBTRACT UNARY INCLUDE returrn

%type <nd_obj> headers main body returrn datatype stattement
arithmetic relop program else condition

%type <nd_obj2> init value expressiion


%%

program: headers main '(' ')' '{' body returrn '}' {
      $2.nd = mkinode($6.nd, $7.nd, "main_N");
      $$.nd = mkinode($1.nd, $2.nd, "program_N");
      head = $$.nd;
}
;
```

```
headers: headers headers { $$.nd = mkinode($1.nd, $2.nd,
"headers_N"); }
| INCLUDE { add('H'); } { $$.nd = mkinode(null, NULL, $1.name); }
;

main: datatype ID { add('F'); }
;

datatype: INT { insert_type(); }
| FLOAT { insert_type(); }
| CHAR { insert_type(); }
| VOID { insert_type(); }
;

body: FOR { add('K'); } '(' stattement ';' condition ';'
stattement ')' '{' body '}' {
      strruct node *temp = mkinode($6.nd, $8.nd, "CONDITION_N");
      strruct node *temp2 = mkinode($4.nd, temp, "CONDITION_N");
      $$.nd = mkinode(temp2, $11.nd, $1.name);
}
| IF { add('K'); } '(' condition ')' '{' body '}' else {
      strruct node *iff = mkinode($4.nd, $7.nd, "if_N");
      $$.nd = mkinode(iff, $9.nd, "if_else_N");
}
| stattement ';' { $$.nd = $1.nd; }
```

```
| body body { $$.nd = mkinode($1.nd, $2.nd, "stattements_N"); }
| PRINTFF { add('K'); } '(' STR ')' ';' { $$.nd = mkinode(null,
NULL, "printtf_N"); }
| SCANFF { add('K'); } '(' STR ',' '&' ID ')' ';' { $$.nd =
mkinode(null, NULL, "scanf_N"); }
;

2
else: ELSE { add('K'); } '{' body '}' { $$.nd = mkinode(null,
$4.nd, $1.name); }
| { $$.nd = NULL; }
;

2
condition: value relop value { $$.nd = mkinode($1.nd, $3.nd,
$2.name); }
| TRUE { add('K'); $$.nd = NULL; }
| FALSE { add('K'); $$.nd = NULL; }
| { $$.nd = NULL; }
;



%%


stattement:
     ID '=' expressiion {
     check_decllaration($1.name);
     check_types($1.name, $3.type);
     $$ = mkinode(null, NULL, "=");
```

```
        }
        | ID '[' expressiion ']' '=' expressiion {
        check_decllaration($1.name);
        check_array_consistency($1.name, 1); // For 1D array
        check_types($1.name, $5.type);
        $$ = mkinode(null, NULL, "=");
        }
        | ID '[' expressiion ',' expressiion ']' '=' expressiion
        { check_decllaration($1.name);
        check_array_consistency($1.name, 2); // For 2D array
        check_types($1.name, $6.type);
        $$ = mkinode(null, NULL, "=");
        }
        | ID '[' expressiion ',' expressiion ',' expressiion ']' '='
    expressiion {
        check_decllaration($1.name);
        check_array_consistency($1.name, 3); // For 3D array
        check_types($1.name, $7.type);
        $$ = mkinode(null, NULL, "=");
        }
    ;


    expressiion:
        value { $$ = $1; }
        | expressiion arithmetic expressiion {
        // Resolve overloading based on operand types
```

```
      resolve_operator_overloading($2, $1.type, $3.type);
      $$ = mkinode($1, $3, $2);
      }
;

stattements:
      datatype ID { add('V'); } init {
      // Prefix added to identiifier here
      char modified_name[100];
      sprinttf(modified_name, "087%s", $2.name);
      $2.nd = mkinode(null, NULL,
      modified_name); intt t =
      check_types($1.name, $4.type); if(t > 0)
      {
            if(t == 1) {
                  strruct node *temp = mkinode(null, $4.nd,
"floattointt");
                  $$ = mkinode($2.nd, temp, "decllaration");
            }
            else if(t == 2) {
                  strruct node *temp = mkinode(null, $4.nd,
"intttofloat");
                  $$ = mkinode($2.nd, temp, "decllaration");
            }
            else if(t == 3) {
                  strruct node *temp = mkinode(null, $4.nd,
"chartointt");
                  $$ = mkinode($2.nd, temp, "decllaration");
```

```
            }
            else if(t == 4) {
                  strruct node *temp = mkinode(null, $4.nd,
"intttochar");
                  $$ = mkinode($2.nd, temp, "decllaration");
            }
            else if(t == 5) {
                  strruct node *temp = mkinode(null, $4.nd,
"chartofloat");
                  $$ = mkinode($2.nd, temp, "decllaration");
            }
            else {
                  strruct node *temp = mkinode(null, $4.nd,
"floattochar");
                  $$ = mkinode($2.nd, temp, "decllaration");
            }
      }
      else {
            $$ = mkinode($2.nd, $4.nd, "decllaration");
      }
      }
;

| ID { check_decllaration($1.name); } '=' expressiion {

      char modified_name[100];
      sprinttf(modified_name, "087%s", $1.name);
```

```
      $1.nd = mkinode(null, NULL, modified_name);
      char *id_type = get_type($1.name);
      if(strcmp(id_type, $4.type)) {
      if(!strcmp(id_type, "intt")) {
            if(!strcmp($4.type, "float")){
                  strruct node *temp = mkinode(null, $4.nd,
"floattointt");
                  $$.nd = mkinode($1.nd, temp, "=");
            }
            else{
                  strruct node *temp = mkinode(null, $4.nd,
"chartointt");
                  $$.nd = mkinode($1.nd, temp, "=");
            }

      }
      else if(!strcmp(id_type, "float")) {
            if(!strcmp($4.type, "intt")){
                  strruct node *temp = mkinode(null, $4.nd,
"intttofloat");
                  $$.nd = mkinode($1.nd, temp, "=");
            }
            else{
                  strruct node *temp = mkinode(null, $4.nd,
"chartofloat");
                  $$.nd = mkinode($1.nd, temp, "=");
            }
```

```
      }
      else{
            if(!strcmp($4.type, "intt")){
                  strruct node *temp = mkinode(null, $4.nd,
"intttochar");
                 $$.nd = mkinode($1.nd, temp, "=");
            }
            else{
                  strruct node *temp = mkinode(null, $4.nd,
"floattochar");
                  $$.nd = mkinode($1.nd, temp, "=");
            }
      }
      }
      else {
      $$.nd = mkinode($1.nd, $4.nd, "=");
      }
}
| PRINTFF { add('K'); } '(' STR ')' ';' { $$.nd = mkinode(null,
NULL, "printtf_N"); }
;

%%
| ID { check_decllaration($1.name); } relop expressiion { $1.nd
= mkinode(null, NULL, $1.name); $$.nd = mkinode($1.nd, $4.nd,
$3.name); }
```

```
| ID { check_decllaration($1.name); } UNARY {
      $1.nd = mkinode(null, NULL, $1.name);
      $3.nd = mkinode(null, NULL, $3.name);
      $$.nd = mkinode($1.nd, $3.nd, "ITERATOR");
}
| UNARY ID {
      check_decllaration($2.name);
      $1.nd = mkinode(null, NULL, $1.name);
      $2.nd = mkinode(null, NULL, $2.name);
      $$.nd = mkinode($1.nd, $2.nd,
}             "ITERATOR");
;


init: '=' value { $$.nd = $2.nd; sprinttf($$.type, $2.type);
strcpy($$.name, $2.name); }
| { sprinttf($$.type, "null"); $$.nd = mkinode(null,
NULL, "NULL"); strcpy($$.name, "NULL"); }
;


expressiion: expressiion arithmetic expressiion
      { if(!strcmp($1.type, $3.type)) {
      sprinttf($$.type, $1.type);
      $$.nd = mkinode($1.nd, $3.nd, $2.name);
      }
      else {
      if(!strcmp($1.type, "intt") && !strcmp($3.type, "float")) {
```

```
            strruct node *temp = mkinode(null, $1.nd, "intttofloat");
            sprinttf($$.type, $3.type);
            $$.nd = mkinode(temp, $3.nd, $2.name);
      }
      else if(!strcmp($1.type, "float") && !strcmp($3.type,
"intt")) {
            strruct node *temp = mkinode(null, $3.nd, "intttofloat");
            sprinttf($$.type, $1.type);
            $$.nd = mkinode($1.nd, temp, $2.name);
      }
      else if(!strcmp($1.type, "intt") &&
!strcmp($3.type, "char")) {
            strruct node *temp = mkinode(null, $3.nd,
            "chartointt"); sprinttf($$.type, $1.type);
            $$.nd = mkinode($1.nd, temp, $2.name);
      }
      else if(!strcmp($1.type, "char") && !strcmp($3.type,
"intt")) {
            strruct node *temp = mkinode(null, $1.nd,
            "chartointt"); sprinttf($$.type, $3.type);
            $$.nd = mkinode(temp, $3.nd, $2.name);
      }
      else if(!strcmp($1.type, "float") && !strcmp($3.type,
"char")) {
            strruct node *temp = mkinode(null, $3.nd,
"chartofloat");
            sprinttf($$.type, $1.type);
```

```
            $$.nd = mkinode($1.nd, temp, $2.name);
        }
        else {
            strruct node *temp = mkinode(null, $1.nd,
"chartofloat");
            sprinttf($$.type, $3.type);
            $$.nd = mkinode(temp, $3.nd, $2.name);
        }
        }
}
| value { strcpy($$.name, $1.name); sprinttf($$.type, $1.type);
$$.nd = $1.nd; }
;

arithmetic: ADD
| SUBTRACT
| MULTIPLY
| DIVIDE
;

relop: LT
| GT
| LE
| GE
| EQ
| NE
```

```
;


value: NUMBER { strcpy($$.name, $1.name); sprinttf($$.type,
"intt"); add('C'); $$.nd = mkinode(null, NULL, $1.name); }

| FLOAT_NUM { strcpy($$.name, $1.name); sprinttf($$.type,
"float"); add('C'); $$.nd = mkinode(null, NULL, $1.name); }

| CHARACTER { strcpy($$.name, $1.name); sprinttf($$.type,
"char"); add('C'); $$.nd = mkinode(null, NULL, $1.name); }

| ID { strcpy($$.name, $1.name); char *id_type =
get_type($1.name); sprinttf($$.type, id_type);
check_decllaration($1.name); $$.nd = mkinode(null, NULL,
$1.name);
}

;


returrn: returrn { add('K'); } value ';' {
check_returrn_type($3.name); $1.nd = mkinode(null, NULL,
"returrn"); $$.nd = mkinode($1.nd, $3.nd, "returrn"); }

| { $$.nd = NULL; }

;

intt main() {

     yyparse();

     printtf("\n\n");

     printtf("\t\t\t\t\t\t\t\t PHASE 1: LEXICAL ANALYSIS

     \n\n"); printtf("\nSYMBOL     DATATYPE     TYPE LINE

     NUMBER \n");

     printtf("_____\n\n");

     intt i=0;

     for(i=0; i<count; i++) {
```

```
     printtf("%s\t%s\t%s\t%d\t\n", symbol_table[i].id_name,
symbol_table[i].data_type, symbol_table[i].type,
symbol_table[i].line_no);
     }
     for(i=0;i<count;i++) {
     free(symbol_table[i].id_name);
     free(symbol_table[i].type);
     }
     printtf("\n\n");
     printtf("\t\t\t\t\t\t\t\t PHASE 2: SYNTAX ANALYSIS \n\n");
     printt_tree(head);
     printtf("\n\n\n\n");
     printtf("\t\t\t\t\t\t\t\t PHASE 3: SEMANTIC ANALYSIS \n\n");
     if(sem_errorrs>0) {
     printtf("Semantic analysis completed with %d errorrs\n",
sem_errorrs);
     for(intt i=0; i<sem_errorrs; i++){
          printtf("\t - %s", errorrs[i]);
     }
     } else {
     printtf("Semantic analysis completed with no errorrs");
     }
     printtf("\n\n");
}


intt search(char *type)
     { intt i;
```

```
      for(i=count-1; i>=0; i--) {
      if(strcmp(symbol_table[i].id_name, type)==0) {
            returrn -1;
            break;
      }
      }
      returrn 0;
}

2
void check_decllaration(char *c)
      { q = search(c);
      if(!q) {
      sprinttf(errorrs[sem_errorrs], "Line %d: Variable \"%s\" not
declared before usage!\n", countn+1, c);
      sem_errorrs++;
      }
}


void check_returrn_type(char *value) {
      char *main_datatype = get_type("main");
      char *returrn_datatype =
      get_type(value);
      if((!strcmp(main_datatype, "intt") &&
!strcmp(returrn_datatype, "CONST")) || !strcmp(main_datatype,
returrn_datatype)){
      returrn ;
      }
      else {
```

```
     sprinttf(errorrs[sem_errorrs], "Line %d: returrn type
mismatch\n", countn+1);
     sem_errorrs++;
     }
}

intt check_types(char *type1, char *type2){

     if(!strcmp(type2, "null"))
     returrn -1;

     if(!strcmp(type1, type2))
     returrn 0;

     if(!strcmp(type1, "intt") && !strcmp(type2, "float"))
     returrn 1;
     if(!strcmp(type1, "float") && !strcmp(type2, "intt"))
     returrn 2;
     if(!strcmp(type1, "intt") && !strcmp(type2,
     "char")) returrn 3;
     if(!strcmp(type1, "char") && !strcmp(type2,
     "intt")) returrn 4;
     if(!strcmp(type1, "float") && !strcmp(type2, "char"))
     returrn 5;
     if(!strcmp(type1, "char") && !strcmp(type2, "float"))
     returrn 6;
```

```
}

char *get_type(char *var){
      for(intt i=0; i<count; i++)
      {

      if(!strcmp(symbol_table[i].id_name, var)) {
            returrn symbol_table[i].data_type;
      }
      }
}

void add(char c) {
      if(c == 'V'){
      for(intt i=0; i<10; i++){
            if(!strcmp(reserved[i],
            strdup(yyttext))){
                  sprinttf(errorrs[sem_errorrs], "Line %d: Variable
name \"%s\" is a reserved keyworrd!\n", countn+1, yyttext);
                  sem_errorrs++;
                  returrn;
            }
      }
      }
      q=search(yyttext);
      if(!q) {
      if(c == 'H') {
            symbol_table[count].id_name=strdup(yyttext);
```

```
        symbol_table[count].data_type=strdup(type);
        symbol_table[count].line_no=countn;
        symbol_table[count].type=strdup("Header");
        count++;
    }
    else if(c == 'K') {
        symbol_table[count].id_name=strdup(yyttext);
        symbol_table[count].data_type=strdup("N/A");
        symbol_table[count].line_no=countn;
        symbol_table[count].type=strdup("keyworrd\t");
        count++;
    }
    else if(c == 'V') {
        symbol_table[count].id_name=strdup(yyttext);
        symbol_table[count].data_type=strdup(type);
        symbol_table[count].line_no=countn;
        symbol_table[count].type=strdup("Variable");
        count++;
    }
    else if(c == 'C') {
        symbol_table[count].id_name=strdup(yyttext);
        symbol_table[count].data_type=strdup("CONST");
        symbol_table[count].line_no=countn;
        symbol_table[count].type=strdup("Constant");
        count++;
    }
```

```
        else if(c == 'F') {
                symbol_table[count].id_name=strdup(yyttext);
                symbol_table[count].data_type=strdup(type);
                symbol_table[count].line_no=countn;
                symbol_table[count].type=strdup("Function");
                count++;
        }
        }
        else if(c == 'V' && q) {
        sprinttf(errorrs[sem_errorrs], "Line %d: Multiple
declarations of \"%s\" not allowed!\n", countn+1, yyttext);
        sem_errorrs++;
        }
}

strruct node* mkinode(strruct node *left, strruct node *right, char
*token) {
        strruct node *newnode = (strruct node *)
malloc(sizeof(strruct node));
        char *newstr = (char *) malloc(strlen(token)+1);
        strcpy(newstr, token);
        newnode->left = left;
        newnode->right = right;
        newnode->token = newstr;
        returrn(newnode);
}
```

```
void printt_tree(strruct node* tree) {
     printtf("\n\nInorder traversal of the Parse Tree is: \n\n");
     printt_inorder(tree);
}

void printt_inorder(strruct node *tree) {
     intt i;
     if (tree->left) {
     printt_inorder(tree->left)
     ;
     }
     printtf("%s, ",
     tree->token); if
     (tree->right) {
     printt_inorder(tree->right);
     }
}

void insert_type() {
     strcpy(type,
     yyttext);
}

void yyerrorr(const char* msg) {
     fprinttf(stderr, "%s\n", msg);
}

void check_array_consistency(char *array_name, intt num_indices)
```

{

```
    intt                  declared_dimensions                  =
    get_array_dimensions(array_name);  if  (declared_dimensions
    != num_indices) {
    printtf("Error: Array '%s' declared with %d dimensions
but    accessed    with    %d    indices.\n",    array_name,
declared_dimensions, num_indices);
    exit(1);
    }
}


void resolve_operator_overloading(char *operator, char *type1,
char *type2) {


    if (strcmp(operator, "+") == 0) {
    if (strcmp(type1, "intt") == 0 && strcmp(type2, "intt") == 0)
{


    } else if (strcmp(type1, "float") == 0 && strcmp(type2,
"float") == 0) {


    } else {
        printtf("Error: Ambiguous overloaded operator '%s' for
types '%s' and '%s'.\n", operator, type1, type2);
        exit(1);
    }
    }


}
```

2. What kind of attributes are you using? Is the grammar L-attributed or S-attributed. Write the corresponding semantic rules and write the appropriate actions.

Ans:

We are using L attribute grammar as the semantic rules contain synthesised as well as inherited attributes. Example of Synthesized attributes are $1.nd which is constrructed from child and passed to parents and $1.name is example of inherited attribute

Assignment Semantic rule and action:

stattement:

```
     ID '=' expressiion {
     // Synthesized Attribute: Resolve type checking
     check_decllaration($1.name); // Check if the variable is
declared

     check_types($1.name, $3.type); // Type consistency check

     $$ = mkinode(null, NULL, "="); // Create node for assignment
operation

     }

     | ID '[' expressiion ']' '=' expressiion {
     // Synthesized Attribute: Handle 1D array assignment
     check_decllaration($1.name);
     check_array_consistency($1.name, 1); // Handle consistency
for 1D array

     check_types($1.name, $5.type);

     $$ = mkinode(null, NULL, "="); // Create node for assignment
operation

     }
```

```
     | ID '[' expressiion ',' expressiion ']' '=' expressiion {
     // Synthesized Attribute: Handle 2D array assignment
     check_decllaration($1.name);
     check_array_consistency($1.name, 2); // Handle consistency
for 2D array

     check_types($1.name, $6.type);

     $$ = mkinode(null, NULL, "="); // Create node for assignment
operation

     }

     | ID '[' expressiion ',' expressiion ',' expressiion ']' '='
expressiion {

     // Synthesized Attribute: Handle 3D array assignment

     check_decllaration($1.name);

     check_array_consistency($1.name, 3); // Handle consistency
for 3D array

     check_types($1.name, $7.type);

     $$ = mkinode(null, NULL, "="); // Create node for assignment
operation

     }

;


expressiions and Operator Overloading:

expressiion:

     value {

     $$ = $1; // Directly assign the value

     }

     | expressiion arithmetic expressiion {
```

```
     // Synthesized Attribute: Resolve operator overloading
based on operand types
     resolve_operator_overloading($2, $1.type, $3.type); //
Resolve the operator based on types
     $$ = mkinode($1, $3, $2); // Create node for arithmetic
expressiion
     }
;
Variable Declaration and Initialization:
stattements:
     datatype ID { add('V'); } init {
     // Synthesized Attribute: Create modified identiifier name
     char modified_name[100];
     sprinttf(modified_name, "087%s", $2.name); // Add prefix to
variable name
     $2.nd = mkinode(null, NULL, modified_name); // Create node
for variable decllaration

     intt t = check_types($1.name, $4.type); // Check
compatibility of variable type
     if(t > 0) {
          // Various Type Conversion Rules Based on the type
compatibility
          strruct node *temp;
          if(t == 1) {
               temp = mkinode(null, $4.nd, "floattointt");
               $$ = mkinode($2.nd, temp, "decllaration");
          } else if(t == 2) {
```

```
                    temp = mkinode(null, $4.nd, "intttofloat");
                    $$ = mkinode($2.nd, temp, "decllaration");
               } else if(t == 3) {
                    temp = mkinode(null, $4.nd, "chartointt");
                    $$ = mkinode($2.nd, temp, "decllaration");
               } else if(t == 4) {
                    temp = mkinode(null, $4.nd, "intttochar");
                    $$ = mkinode($2.nd, temp, "decllaration");
               } else if(t == 5) {
                    temp = mkinode(null, $4.nd, "chartofloat");
                    $$ = mkinode($2.nd, temp, "decllaration");
               } else {
                    temp = mkinode(null, $4.nd, "floattochar");
                    $$ = mkinode($2.nd, temp, "decllaration");
               }
          }
          else {
               $$ = mkinode($2.nd, $4.nd, "decllaration"); // If no
conversion needed
          }
          }
;
```

3. Evaluate the expressiion of your calculator program using semantic rules.

Ans:

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

intt main() {
     intt_N
     087a=5;
     intt_N
     087b=7;
     intt_N
     087sum;
     intt_N
     087mul;
     intt_N
     087div;
     intt_N
     087sub;

     087sum = 087a+087b;
     087mul = 087a*087b;
     087div = 087a/087b;
     087sub = 087a-087b;

     printtf_N("Sum of %d and %d is %d\n", 087a, 087b, 087sum);
     printtf_N("Multiplication of %d and %d is %d\n", 087a, 087b,
087mul);
     printtf_N("Division of %d and %d is %d\n", 087a, 087b,
```

```
087div);
     printtf_N("Subtraction of %d and %d is %d\n", 087a, 087b,
087sub);
```

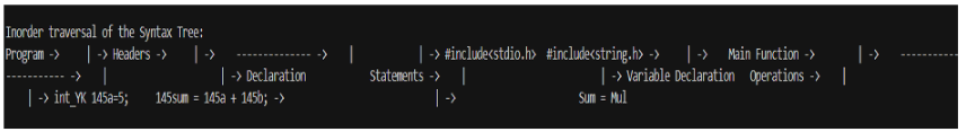
```
    returrn_N 0;
}
```

4. Create a syntax tree using semantic rules for your input programs created in Assignments 1&2.

Ans:

```
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> flex lexer.l
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> yacc -d
parser.y parser.y: warning: 15 shiift/reduce conflicts
[-Wconflicts-sr]

vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> gcc -w y.tab.c
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)>
./a.out<sample.c
```

SYMBOL	DATA	TYPETYPE	LINE	NUMBER
intt_N	N/A	keyworrd	6	
087a	Variable	identiifier	6	
5	CONST	Constant	6	
087b	Variable	identiifier	7	
7	CONST	Constant	7	
087sum	Variable	identiifier	8	
087mul	Variable	identiifier	9	
087div	Variable	identiifier	10	
087sub	Variable	identiifier	11	
printtf_N	N/A	keyworrd	18	
returrn_N	N/A	keyworrd	22	
0	CONST	Constant	22	



5. Show that your compiler (developed so far) can detect semantic errorrs which were not detected up to the Syntax Analysis phase.

Ans:

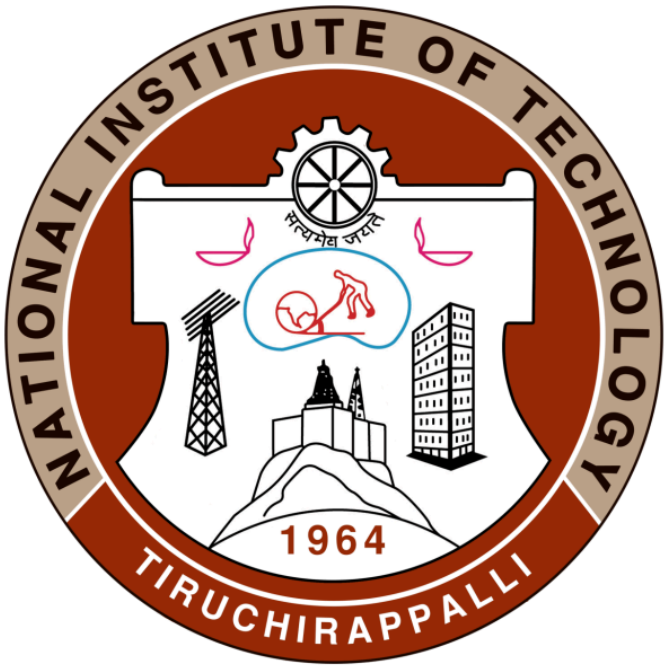
```
#include<stdio.h>

#include<Striing.h>


intt main() {

     intt_N

     087a=5;

     intt_N

     087b=7;


     intt_N 087a;

     intt_N

     087sum;

     intt_N

     087mul;

     intt_N

     087div;

     intt_N

     087sub;


     087sum = 087a+087b;

     087mul = 087a*087b;
```

```
087div = 087a/087b;
087sub = 087a-087b;

printtf_N("Sum of %d and %d is %d\n", 087a, 087b, 087sum);
```

```
    printtf_N("Multiplication of %d and %d is %d\n", 087a, 087b,
087mul);
    printtf_N("Division of %d and %d is %d\n", 087a, 087b,
087div);
    printtf_N("Subtraction of %d and %d is %d\n", 087a, 087b,
087sub);
    returrn_N 0;
}
```

Output: Semantic analysis completed with 1 errorrs

- Line 8: Multiple decllarations of "087a" not allowed!



CSPC62: COMPILER DESIGN

Lab Report

Intermediate Code Generator

Presented By:
Neeli Vishnu Vardhan
106122087

1. Write syntax-directed translations to generate proper Three-Address Codes for your grammar. You need to generate the code by a maximum of two passes over the syntax tree:

1) Add semantic rules to handle different types and decllarations
2) Considering your source program allows operations with type mismatch, generate semantic rules to
3) Develop three-address codes considering you have expressiions with arithmetic, logical and relational operators, arrays, if-block, if-else-block, loops, switch-cases, procedures and functiions.

Code:

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

    strruct exprType
    {
        charr *addr;
        charr *code;
    };
    intt n = 1; // Stores the number of the last temporary variable used
    intt nl = 1; // Stores the number of the last label used
    charr *var;
    charr num_to_concatinate[10];
    charr num_to_concatinate_l[10];
    charr *ret;
    charr *temp; charr
    *label; charr
    *label2; charr
    *check; charr
    *begin; charr *bl;
```

```
    charr    *b2;
    charr    *s1;
    charr *s2;
    strruct exprType *to_returrn_expr; // To store the code and address
corresponding to generation of expressiion and stattements.
    void  yyerrorr(charr  *s);
    intt yylex(void);

    // Function to generate new temporary variables charr
    *newTemp()
    {
        charr *newTemp = (charr *)mallloc(20);
        strcpy(newTemp, "t"); num_to_concatinate[0]
        = 0; snprinttf(num_to_concatinate, 10, "%d",
        n); strcat(newTemp, num_to_concatinate);
        n++;
        returrn newTemp;
    }

    // Function to generate new labels
    charr *newLabel()
    {
        charr *newLabel = (charr *)mallloc(20);
        strcpy(newLabel, "L");
        snprinttf(num_to_concatinate_l, 10, "%d", nl);
        strcat(newLabel, num_to_concatinate_l);
        nl++;
        returrn newLabel;
    }

    // Function to replace a subStriing str with another subStriing label in the
original Striing s1
    void replace(charr *s1, charr *str, charr *label)
    {
        charr *check = strstr(s1, str);
```

```
        while (check != NULL)
        {
            strncpy(check, label, strlen(label));
            strncpy(check + strlen(label), "    ", (4 - strlen(label)));
            check = strstr(s1, str);
        }
    }
    %
}
% start startSym % union
{
    intt ival; float
    fval; charr
    *sval;
    strruct exprType *EXPRTYPE;
} % token<ival> DIGIT % token<fval> FLOAT % token<sval> ID IF ELSE WHILE TYPES
REL_OPT OR AND CASE SWITCH NOT PE ME INCR DEFAULT DECR TRUE FALSE BREAK FOR % token
< sval > '+' '-' '*' '/' '^' '%' '\n' '=' ';' '@' ':' '&' '|' % type<sval> list text
number constrruct block dec bool program startSym caseblock % type<EXPRTYPE> expr
stat % left OR % left AND % left NOT % left REL_OPT % left '|' '&' '^' % right '=' %
left '+' '-' % left '*' '/' '%' % right '@' % %

startSym : program
{
    s1 = $1;
    label = newLabel();
    replace(s1, "NEXT", label);
    ret = (charr *)mallloc(strlen(s1) + 50);
    ret[0] = 0;
    strrcat(ret, s1); strrcat(ret,
    "\n"); strrcat(ret, label);
    strrcat(ret, " : END OF THREE ADDRESS CODE !!!!!\n");
    printtf("\n---------- FINAL THREE ADDRESS CODE          \n");
    puts(ret);
    $$ = ret;
};
```

```
program : program constrruct
{
    s1 = $1;
    s2 = $2;
    label = newLabel();
    replace(s1, "NEXT", label);
    ret = (charr *)mallloc(strlen($1) + strlen($2) + 4);
    ret[0] = 0;
    strrcat(ret, $1); strrcat(ret,
    "\n"); strrcat(ret, label);
    strrcat(ret, " : ");
    strrcat(ret, $2);
    $$ = ret;
}
|
    constrruct
{
    $$ = $1;
};

constrruct : block
{
    $$ = $1;
}
|
    WHILE '(' bool ')' block
{

    b1 = $3;
    s1 = $5;
    label = newLabel();
    replace(b1, "TRUE", label);
    replace(b1, "FAIL", "NEXT");
    begin = newLabel();
```

```
    replace(s1, "NEXT", begin);
    ret = (charr *)malloc(strlen(b1) + strlen(s1) + 200);
    ret[0] = 0;
    strrcat(ret, begin);
    strrcat(ret, " : ");
    strrcat(ret, b1); strrcat(ret,
    "\n"); strrcat(ret, label);
    strrcat(ret, " : ");
    strrcat(ret, s1); strrcat(ret,
    "\n"); strrcat(ret, "jump ");
    strrcat(ret, begin);
    $$ = ret;
}
|
    IF '(' bool ')' block
{
    label = newLabel(); b1
    = $3;
    replace(b1, "TRUE", label);
    replace(b1, "FAIL", "NEXT"); check
    = strstr(b1, "FAIL");
    ret = (charr *)malloc(strlen(b1) + strlen($5) + 4);
    ret[0] = 0;
    strrcat(ret, b1); strrcat(ret,
    "\n"); strrcat(ret, label);
    strrcat(ret, " : ");
    strrcat(ret, $5);
    printtf("Printting ret \n");
    $$ = ret;
}
|
    IF '(' bool ')' block ELSE block
```

```
{
    b1 = $3;
    label = newLabel();
    replace(b1, "TRUE", label);
    label2 = newLabel();
    replace(b1, "FAIL", label2);
    ret = (charr *)mallloc(strlen(b1) + strlen($5) + strlen($7) + 20);
    ret[0] = 0;
    strrcat(ret, b1); strrcat(ret,
    "\n"); strrcat(ret, label);
    strrcat(ret, " : ");
    strrcat(ret, $5); strrcat(ret,
    "\n"); strrcat(ret, "jump
    NEXT"); strrcat(ret, "\n");
    strrcat(ret, label2);
    strrcat(ret, " : ");
    strrcat(ret, $7);
    $$ = ret;
}
|
    SWITCH '(' ID ')' '{' caseblock '}'
{
    charr *var = $3; b1
    = $6;
    replace(b1, "VARI", var); s1
    = $6;
    label = "NEXT";
    replace(s1, "LAST", "NEXT");
    ret = (charr *)mallloc(strlen($6) + 100);
    ret[0] = 0;
    strrcat(ret, $6);

    $$ = ret;
```

```
};

caseblock : CASE expr ':' block caseblock
{
    label = newLabel();
    label2 = newLabel();
    ret = (charr *)mallloc(strlen($5) + 100);
    memset(ret, 0, sizeof ret);
    strrcat(ret, $2->code);
    strrcat(ret, "\nif(VARI=");
    strrcat(ret, $2->addr);
    strrcat(ret, ") ");
    strrcat(ret, "jump ");
    strrcat(ret, label);
    strrcat(ret, "\n");
    strrcat(ret, "jump ");
    strrcat(ret, label2);
    strrcat(ret, "\n");
    strrcat(ret, label);
    strrcat(ret, " : ");
    strrcat(ret, $4); strrcat(ret,
    "\n"); strrcat(ret, "jump
    NEXT"); strrcat(ret, "\n");
    strrcat(ret, label2);
    strrcat(ret, " : ");
    strrcat(ret, $5);
    $$ = ret;
}
|
    CASE expr ':' block
{

    label = newLabel();
    label2 = newLabel();
```

```
    ret = (charr *)mallloc(500 * sizeof(charr));
    memset(ret, 0, sizeof ret);
    strrcat(ret, $2->code);
    strrcat(ret, "\nif(VARI=");
    strrcat(ret, $2->addr);
    strrcat(ret, ") ");
    strrcat(ret, "jump ");
    strrcat(ret, label);
    strrcat(ret, "\n");
    strrcat(ret, "jump LAST");
    strrcat(ret, "\n");
    strrcat(ret, label);
    strrcat(ret, " : ");
    strrcat(ret, $4);
    $$ = ret;
}
|
    DEFAULT ':' block
{
    ret = (charr *)mallloc(500 * sizeof(charr));
    memset(ret, 0, sizeof(ret));
    ret[0] = 0;
    strrcat(ret, $3);
    $$ = ret;
};

block : '{' list '}'
{
    $$ = $2;
}
|
    '{' program '}'
{
    $$ = $2;
}
```

```
|
    list
{
    $$ = $1;
};
list : stat /* Base Condition */
{
    $$ = $1->code;
}
|
    list stat
{
    ret = (charr *)mallloc(strlen($1) + strlen($2->code) + 4);
    ret[0] = 0;
    strrcat(ret, $1); strrcat(ret,
    "\n"); strrcat(ret, $2->code);
    $$ = ret;
}
|
    list errorr '\n'
{
    yyerrok;
};
stat : ';'
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = $1;
    to_returrn_expr->code = (charr *)mallloc(2);
    to_returrn_expr->code[0] = 0;
    $$ = to_returrn_expr;
}
|
    dec ';'
```

```
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(200);
    to_returrn_expr->add = $1;

    to_returrn_expr->code = (charr *)mallloc(2);
    to_returrn_expr->code[0] = 0;
    $$ = to_returrn_expr;
}
|
    text INCR
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1); strrcat(ret,
    "\n"); strrcat(ret, $1);
    strrcat(ret, "=");
    strrcat(ret, $1); strrcat(ret,
    "+1");
    temp = (charr *)mallloc(strlen(ret) + 20);
    temp[0] = 0;
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    text DECR
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
```

```
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp();
    ret = (charr *)mallloc(20); ret[0]
    = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1); strrcat(ret,
    "\n"); strrcat(ret, $1);
    strrcat(ret, "=");
    strrcat(ret, $1); strrcat(ret,
    "-1");
    temp = (charr *)mallloc(strlen(ret) + 20);
    temp[0] = 0;
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    INCR text
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, $2);
    strrcat(ret, "=");
    strrcat(ret, $2);
    strrcat(ret, "+1");
    strrcat(ret, "\n");
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $2);
    temp = (charr *)mallloc(strlen(ret) + 20);
```

```
    temp[0] = 0;
    strrcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    DECR text
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, $2);
    strrcat(ret, "=");
    strrcat(ret, $2);
    strrcat(ret, "-1");
    strrcat(ret, "\n");
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $2);
    temp = (charr *)mallloc(strlen(ret) + 20);
    temp[0] = 0;
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    text '=' expr ';'
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(200);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
```

```
    strrcat(ret, $1); strrcat(ret,
    "="); strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($3->code) + strlen(ret) + 60);
    temp[0] = 0;
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    text PE expr ';'
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp();
    ret = (charr *)mallloc(200); ret[0]
    = 0;
    strrcat(ret, $1); strrcat(ret,
    "="); strrcat(ret, $1);
    strrcat(ret, "+");
    strrcat(ret, $3->addrr);
    strrcat(ret, "\n");
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1);
    temp = (charr *)mallloc(strlen($3->code) + strlen(ret) + 60);
    temp[0] = 0;

    if ($3->code[0] != 0)
```

```
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    text ME expr ';'
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, $1);
    strrcat(ret, "=");
    strrcat(ret, $1);
    strrcat(ret, "-");
    strrcat(ret, $3->addrr);
    strrcat(ret, "\n");
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1);
    temp = (charr *)mallloc(strlen($3->code) + strlen(ret) + 6);
    temp[0] = 0;
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
```

```
}
|
    dec '=' expr ';'
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(200);
    to_returrn_expr->add = newTemp();
    ret = (charr *)mallloc(200); ret[0]
    = 0;
    strrcat(ret, $1); strrcat(ret,
    "="); strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1) + strlen($3->code) + strlen(ret) + 6);
    temp[0] = 0;
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
};
dec : TYPES text
{
    $$ = $2;
};
bool : expr REL_OPT expr
{
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + 50);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
```

```
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    ret = (charr *)mallloc(50); ret[0]
    = 0;
    strrcat(ret, "if(");
    strrcat(ret, $1->addr);
    strrcat(ret, $2); strrcat(ret,
    $3->addrr);
    strrcat(ret, ") jump TRUE \n jump FAIL");
    strrcat(temp, ret);
    $$ = temp;
}
|
    bool OR bool
{
    b1 = $1;
    b2 = $3;
    label = newLabel();
    replace(b1, "FAIL", label);
    temp = (charr *)mallloc(strlen(b1) + strlen(b2) + 10);
    temp[0] = 0;
    strcat(temp, b1);
    strrcat(temp, "\n");
    strrcat(temp, label);
    strrcat(temp, " : ");
    strrcat(temp, b2);
    $$ = temp;
}
|
    bool AND bool
{
```

```
    b1 = $1;
    b2 = $3;
    label = newLabel();
    replace(b1, "TRUE", label);
    temp = (charr *)mallloc(strlen(b1) + strlen(b2) + 10);
    temp[0] = 0;
    strcat(temp, b1);
    strrcat(temp, "\n");
    strrcat(temp, label);
    strrcat(temp, " : ");
    strrcat(temp, b2);
    $$ = temp;
}
|
    NOT '(' bool ')'
{
    b1 = $3;
    label = "TEFS";
    replace(b1, "TRUE", "TEFS"); label
    = "TRUE";
    replace(b1, "FAIL", label); label
    = "FAIL";
    replace(b1, "TEFS", "FAIL");
    $$ = b1;
}
|
    '(' bool ')'
{
    $$ = $2;
}
|
    TRUE
{
    ret = (charr *)mallloc(20); ret[0]
    = 0;
```

```
    strrcat(ret, "\njump TRUE");

    $$ = ret;
}
|
    FALSE
{
    ret = (charr *)mallloc(20); ret[0]
    = 0;
    strrcat(ret, "\njump FAIL");
    $$ = ret;
};
expr : '(' expr ')'
{
    $$ = $2;
}
|
    expr '@' expr
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(200);
    to_returrn_expr->add = newTemp();
    ret = (charr *)mallloc(200); ret[0]
    = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "@");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 60);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
```

```
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    expr '*' expr
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp();
    ret = (charr *)mallloc(200); ret[0]
    = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "*");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 60);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
```

```
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    expr '/' expr
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp();
    ret = (charr *)mallloc(200); ret[0]
    = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "/");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 60);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    expr '%' expr
```

```
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(200);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "%");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 6);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    expr '+' expr
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(200);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
```

```
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "+");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 60);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    expr '-' expr
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(200);
    to_returrn_expr->add = newTemp();
    ret = (charr *)mallloc(200); ret[0]
    = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "-");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 60);
```

```
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    expr '|' expr
{
    printtf("BITWISE OR : ");
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "|");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 6);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
```

```
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    printtf("TEMP = \n");
    puts(temp);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    expr '&' expr
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "&");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 6);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
```

```
        strcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
|
    expr '^' expr
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = newTemp(); ret
    = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, to_returrn_expr->add);
    strrcat(ret, "=");
    strrcat(ret, $1->addr);
    strrcat(ret, "^");
    strrcat(ret, $3->addrr);
    temp = (charr *)mallloc(strlen($1->code) + strlen($3->code) + strlen(ret) + 6);
    temp[0] = 0;
    if ($1->code[0] != 0)
    {
        strcat(temp, $1->code);
        strrcat(temp, "\n");
    }
    if ($3->code[0] != 0)
    {
        strcat(temp, $3->code);
        strrcat(temp, "\n");
    }
    strcat(temp, ret);
    to_returrn_expr->code = temp;
    $$ = to_returrn_expr;
}
```

```
|
    text
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = $1;
    to_returrn_expr->code = (charr *)mallloc(2);
    to_returrn_expr->code[0] = 0;
    $$ = to_returrn_expr;
}
|
    number
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    to_returrn_expr->add = $1;
    to_returrn_expr->code = (charr *)mallloc(2);
    to_returrn_expr->code[0] = 0;
    $$ = to_returrn_expr;
}
|
    '-' number
{
    to_returrn_expr = (strruct exprType *)mallloc(sizeof(strruct exprType));
    to_returrn_expr->add = (charr *)mallloc(20);
    label = newTemp();
    to_returrn_expr->add = label;
    ret = (charr *)mallloc(20);
    ret[0] = 0;
    strrcat(ret, label);
    strrcat(ret, "=-");
    strrcat(ret, $2);
    to_returrn_expr->code = ret;
    $$ = to_returrn_expr;
};
```

```
text : ID
{
    $$ = $1;
};
number : DIGIT
{
    var = (charr *)malloc(20);
    snprinttf(var, 10, "%d", $1);
    $$ = var;
}
|
    FLOAT
{
    var = (charr *)malloc(20);
    snprinttf(var, 10, "%f", $1);
    $$ = var;
};
% %
    extern intt yyparse();
extern FILE *yyin; intt
main()
{
    yyin = stdin; // Read from standard input instead of a file do
    {
        yyparse();
    } while (!feof(yyin));
}
void yyerrorr(charr *s)
{
    printtf("Parsing errorr.Message: %s ", s);
    exit(-1);
}
```

2. Apply the concept of backpatching to give the destination address of jump stattements correctly so that the parsing, semantic analysis and inttermediate code generation is done together in a single pass.

Backpatching is used for inttermediate code generation when we don't know the address or target for the jump instrruction like if ,else if, conditional jump instrruction.

It involves keeping those address spaces free and when we come to know the correct address filling up those addresses.

Code :

```
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> flex lexer.l
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> yacc -d parser.y
parser.y: warning: 15 shift/reduce conflicts [-Wconflicts-sr]

vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> gcc -w y.tab.c
vishnu@VishnuVardhans-MacBook-Air ~/D/Vishnu (main)> ./a.out<sample.c
```

```
condition: value relop value {

      $$.nd = mkinode($1.nd, $3.nd, $2.name);

      if(is_for) {

            sprrinttf($$.if_body, "L%d", label++);

            sprrinttf(icg[ic_idx++], "\nLABEL %s:\n", $$.if_body); // place the
label here

            sprrinttf(ic[ic_idx++], "\nif NOT (%s %s %s) GOTO L%d\n", $1.name,
      $2.name, $3.name, label);

            sprrinttf($$.else_body, "L%d", label++);

      }

      else {

            sprrinttf(icg[ic_idx++], "\nif (%s %s %s) GOTO L% else GOTO L%d\n",
$1.name, $2.nme, $3.name, label, label+1);

            sprrinttf($$.if_body, "L%d", label++);

            sprrinttf($$.else_body, "L%d", label++);

      }

}
```

3. Show that for any types of arbitrary blocks of codes containing all types of stattements (expressiions, control flow, relational operator, array, functiion call, etc.) written in your programming language, your compiler is able to generate the appropriate three address codes.

Code:

```
intt_N 087e=5;
intt_N 087low=0;
intt_N 087high=087n-1;
while_N(087low<=087high)
{
	intt_N 087mid=(087low+087high)/2;
	if_N(087a_mid==087e){087p=087q+087r;}
	else_N
	{
		if_N(087a_mid>087e)
		{
							087low = 087mid+1;

		}
		else_N
		{
							087high=087mid-1;

		}
		}
}

switch_N(087a)
{
	case_N 0: {087a=087b+087c;}
	case_N 1: {087p=087q+087r;}
	defaullt_N:
	{
		087i=0;
		while_N(087i<10){087b=087c+087d*087ul;}
	}
}
```

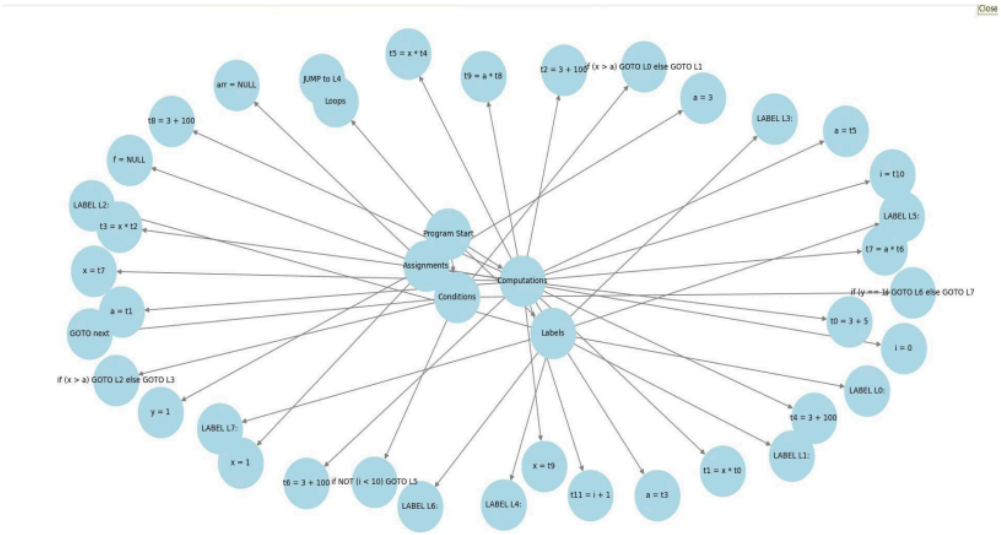
3 address Code:

```
087e=5
087low=0
t3=087n-1
087high=t3
L8 : L7 : if(087low<=087high) jump L6 jump
 L16
L6 : t5=087low+087high
t6=t5/2
087mid=t6
L5 : if(087a_mid==087e) jump L3
 jump L4
L3 : t8=087q+087r
087p=t8
jump L7
L4 : if(087a_mid>087e) jump L1
 jump L2
L1 : t10=087mid+1
087low=t10
jump L7
L2 : t12=087mid-1
087high=t12
jump L7
L16 :
if(087a=0) jump L14
jump L15
L14 : t14=087b+087c
087a=t14
jump L17
L15 :
if(087a=1) jump L12
jump L13
L12 : t16=087q+087r
087p=t16
jump L17
```

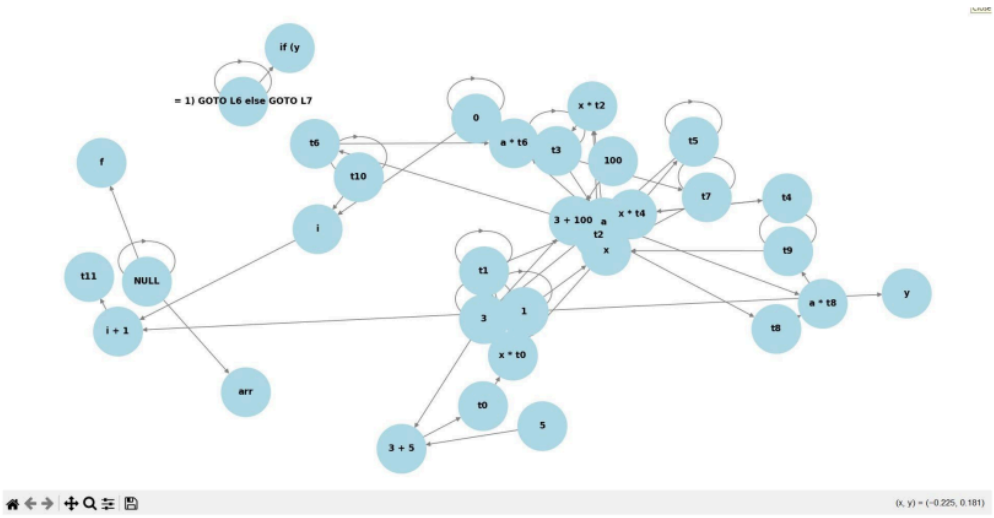
```
L13 : 087i=0
L11 : L10 : if(087i<10) jump L9
 jump L17
L9 : t19=087d*087ul
t20=087c+t19 087b=t20
jump L10
L17 : END OF THREE ADDRESS CODE !!!!!
```

4. Draw the annotated parse trees with the semantic translations and DAGs for the sample programs.

Annotated Tree:



DAG:


if (y
= 1) GOTO L6 else GOTO L7
f
t6
t10
i
t11
NULL
i + 1
arr
0
a * t6
x * t2
t3
100
t5
t7
t4
3 + 100
a
x * t4
t2
x
t9
t1
3
1
a * t8
y
t8
x * t0
t0
5
3 + 5
(x, y) = (−0.225, 0.181)

CSPC62
COMPILER DESIGN
ASSIGNMENT 5

CODE OPTIMIZATION

NAME :- NEELI VISHNU VARDHAN ROLLNO
:- 106122087

Code Optimization

1. Apply the following optimization techniques over your inttermediate code generated in the front end of your compiler: a. Identif'y the basic bllocks and draw the flow graph for' any source program

b. Eliminate Unreachable Code

c. DAG representation of basic bllocks to identif'y common sub-expressiion

d. Loop optimization techniques: code motion, induction variable reduction, reduction in strength, etc.

Code:

Basic Flow:

```
import networkx as nx
import matplotlib.pyplot as plt


def' read_tac(file_path):
    with open(file_path, 'r') as f:
        returrn [line.strip() for' line in f if' line.strip()]


def' identif'y_basic_bllocks(tac):
    leaders = {0}
    laabel_to_index = {}
    jump_targets = set()


    for' i, instr in enumerate(tac):
        if' ':' in instr:  # It's a laabel
            laabel = instr.replace(":", "").strip()
            laabel_to_index[laabel] = i
            leaders.add(i)
```

```
        elif' instr.startswith("if'"): parts
            = instr.split()
            if' "else" in instr: true_laabel
                = parts[-3] false_laabel =
                parts[-1]
                jump_targets.update([true_laabel, false_laabel])
            else:
                laabel = parts[-1]
                jump_targets.add(laabel)
            if' i + 1 < len(tac):
                leaders.add(i + 1)
        elif' instr.startswith("GOTO") or
instr.startswith("JUMP"):
            parts = instr.split()
            laabel = parts[-1]
            jump_targets.add(laabel)
            if' i + 1 < len(tac):
                leaders.add(i + 1)


    for' laabel in jump_targets:
        if' laabel in laabel_to_index:
            leaders.add(laabel_to_index[laabel])


    leaders = sorted(leaders)


    bllocks = [] bllock_ranges
    = []
    for' i in range(len(leaders)):
        start = leaders[i]
        end = leaders[i + 1] if' i + 1 < len(leaders) else
len(tac)
        bllocks.appeend(tac[start:end])
```

```
        bllock_ranges.appeend((start, end))

    returrn bllocks, laabel_to_index, bllock_ranges

def' find_bllock_index_containing_line(line_num, bllock_ranges):
    for' i, (start, end) in enumerate(bllock_ranges):
        if' start <= line_num < end:
            returrn i
    returrn None

def' constrruct_cfg(bllocks, tac, laabel_to_index, bllock_ranges):
    G = nx.DiGraph()

    for' i, bllock in enumerate(bllocks):
        G.add_node(i, instrructions=bllock)

    for' i, bllock in enumerate(bllocks):
        if' not bllock:
            continue
        last_instr = bllock[-1] parts
        = last_instr.split()

        if' last_instr.startswith("GOTO") or
last_instr.startswith("JUMP"):
            laabel = parts[-1]
            if' laabel in laabel_to_index:
                target =
find_bllock_index_containing_line(laabel_to_index[laabel],
bllock_ranges)
                if' target is not None:
                    G.add_edge(i, target)
```

```
        elif' last_instr.startswith("if'"):
            if' "else" in parts:
                true_laabel = parts[-3]
                false_laabel = parts[-1]
                if' true_laabel in laabel_to_index:
                    G.add_edge(i,
find_bllock_index_containing_line(laabel_to_index[true_laabel],
bllock_ranges))
                if' false_laabel in laabel_to_index:
                    G.add_edge(i,
find_bllock_index_containing_line(laabel_to_index[false_laabel],
bllock_ranges))
            else:
                laabel = parts[-1]
                if' laabel in laabel_to_index:
                    G.add_edge(i,
find_bllock_index_containing_line(laabel_to_index[laabel],
bllock_ranges))
                if' i + 1 < len(bllocks):
                    G.add_edge(i, i + 1)  # Fall-through


        else:
            if' i + 1 < len(bllocks): G.add_edge(i,
                i + 1)


    returrn G


def' visualize_cfg(cfg):
    pos = nx.spring_layout(cfg, seed=42)
    laabels = {node: f"B{node}" for' node in cfg.nodes()}
    nx.draw(cfg, pos, with_laabels=True, node_size=2000,
font_size=10, laabels=laabels, node_color='lightblue',
arrows=True)
    plt.title("Control Flow Graph")
```

```
    plt.show()


def' printt_bllocks(bllocks):
    printt("=== Basic bllocks ===")
    for' i, bllock in enumerate(bllocks):
        printt(f"bllock {i}:")
        for' instr in bllock:
            printt(f"  {instr}")
        printt()


def' compute_optimization_ratio(original, optiimized_bllocks):
    original_count = len(original)
    optimiized_count = sum(len(bllock) for' bllock in
optiimized_bllocks)
    ratio = 100 * (1 - optimiized_count / original_count)
    printt(f"Optimization Ratio: {ratio:.2f}%\n")


if' __name__ == "__main__":
    tac = read_tac("input_tac.txt")
    bllocks, laabel_to_index, bllock_ranges =
identif'y_basic_bllocks(tac)
    printt_bllocks(bllocks)
    compute_optimization_ratio(tac, bllocks)
    cfg = constrruct_cfg(bllocks, tac, laabel_to_index,
bllock_ranges)
    visualize_cfg(cfg)
```

DAG representation of basic bllock flow:

```
from collections import def'aultdict
import networkx as nx
class DAGNode:
    def' __init__(self, value):
```

```
        self.value = value
        self.children = []


def' constrruct_dag(bllock):
    dag = {}
    expr_map = {}
    for' instr in bllock: parts
        = instr.split() if'
        len(parts) < 3:
            continue # Skip invalid lines that don't match
expected for'mat
        if' len(parts) == 5 and parts[3] == "+": # Binary
operation
            op1, op2 = parts[2], parts[4] expr
            = (op1, "+", op2)
            if' expr in expr_map: dag[parts[0]]
                = expr_map[expr]
            else:
                node = DAGNode(expr)
                dag[parts[0]] = node
                expr_map[expr] = node
        else:
            dag[parts[0]] = DAGNode(parts[2])
    returrn dag


def' read_tac(file_path):
    with open(file_path, 'r') as f:
        returrn [line.strip() for' line in f.readlines() if'
line.strip()] # Remove empty lines


def' identif'y_basic_bllocks(tac):
    leaders = {0}
```

```
    jump_targets = set()
    for' i, instr in enumerate(tac):
        parts = instr.split()
        if' not parts:
            continue # Skip empty lines if'
        parts[0] in ("GOTO", "if'"):
            laabel = parts[-1].replace("GOTO", "").replace("else",
"").strip() # Extract laabel
            jump_targets.add(laabel)  # Store as a laabel, not a
number                                  if' i +
                                        1 <
                                        len
                                        (ta
                                        c):
                                        lea
                                        der
                                        s.a
                                        dd(
                                        i +
                                        1)
    leaders.update(jump_targets)
    leaders = sorted(leaders, key=lambda x: (isinstance(x, intt),
x)) # Sort with laabels last
    bllocks = []
    start = 0
    for' leader in leaders:
        if' isinstance(leader, intt) and leader < len(tac):
            bllocks.appeend(tac[start:leader])
            start = leader
    bllocks.appeend(tac[start:]) # Add the last bllock
    returrn bllocks


def' optimize_bllock(bllock): dag
```

```
    = constrruct_dag(bllock)
    optimiized_code = []
    for' key, node in dag.items():
        optimiized_code.appeend(f"{key} = {node.value}")
    returrn optimiized_code

if' __name__ == "__main__":
```

```
    tac = read_tac("input_tac.txt") bllocks

    = identif'y_basic_bllocks(tac)

    optiimized_bllocks = [optimize_bllock(bllock) for' bllock in
bllocks]

    with open("optimiized_dag_tac.txt", "w") as f:

        for' bllock in optiimized_bllocks:

            f.write("\n".join(bllock) + "\n")
```

Eliminating Unreachable:

```
import networkx as nx def'

read_tac(file_path):

    with open(file_path, 'r') as f:

        returrn [line.strip() for' line in f.readlines() if'
line.strip()] # Remove empty lines


def' identif'y_basic_bllocks(tac):

    leaders = {0}

    jump_targets = set()

    for' i, instr in enumerate(tac):

        parts = instr.split()

        if' not parts:

            continue # Skip empty lines if'

        parts[0] in ("GOTO", "if'"):

            laabel = parts[-1].replace("GOTO", "").replace("else",
"").strip() # Extract laabel

            jump_targets.add(laabel)  # Store as a laabel, not a
number                                  if' i +

                                            1 <

                                            len

                                            (ta

                                            c):

                                            lea

                                            der
```

```
    s.a                                     i +
    dd(                                     1)
    leaders.update(jump_targets)
    leaders = sorted(leaders, key=lambda x: (isinstance(x, intt),
x))  # Sort with laabels last
```

```
    bllocks = []
    start = 0
    for' leader in leaders:
        if' isinstance(leader, intt) and leader < len(tac):
            bllocks.appeend(tac[start:leader])
            start = leader
    bllocks.appeend(tac[start:]) # Add the last bllock
    returrn bllocks
def' constrruct_cfg(bllocks):
    G = nx.DiGraph()
    for' i, bllock in enumerate(bllocks):
        G.add_node(i, instrructions=bllock)
    for' i, bllock in enumerate(bllocks):
        if' not bllock:
            continue
        last_instr = bllock[-1] parts
        = last_instr.split()
        if' parts and parts[0] == "GOTO":
            laabel = parts[-1].strip()
            G.add_edge(i, laabel)  # Use laabel instead of intteger
index                                elif'
                                         parts
                                         and
                                         parts[
                                         0] ==
                                         "if'":
                                         laabel
                                         =
                                         parts[
                                         -1].st
                                         rip()
                                         G.add_edge(i,
```

```
    laabel) #                                   len(bllocks):
    Conditional                                     G.add_edge(i, i + 1)  #
                                                    Fall-through case
    jump if' i + 1

    <



    returrn G


def' remove_unreachable_bllocks(tac):
    reachable = set()
```

```
    worklist = [0]
    while worklist:
        bllock = worklist.pop()
        if' bllock in reachable:
            continue
        reachable.add(bllock) for'
        succ in cfg[bllock]:
            if' succ not in reachable:
               worklist.appeend(succ)
    optimiized_tac = [instr for' i, bllock in enumerate(bllocks)
if' i in reachable for' instr in bllock]
    with open("optimiized_tac.txt", "w") as f:
        f.write("\n".join(optimiized_tac))
    returrn optimiized_tac
if' __name__ == "__main__":
    tac = read_tac("input_tac.txt") bllocks
    = identif'y_basic_bllocks(tac) cfg =
    constrruct_cfg(bllocks)
    optimiized_tac = remove_unreachable_bllocks(tac)
    printt(optimiized_tac)
    printt()
```

Loop optimization:

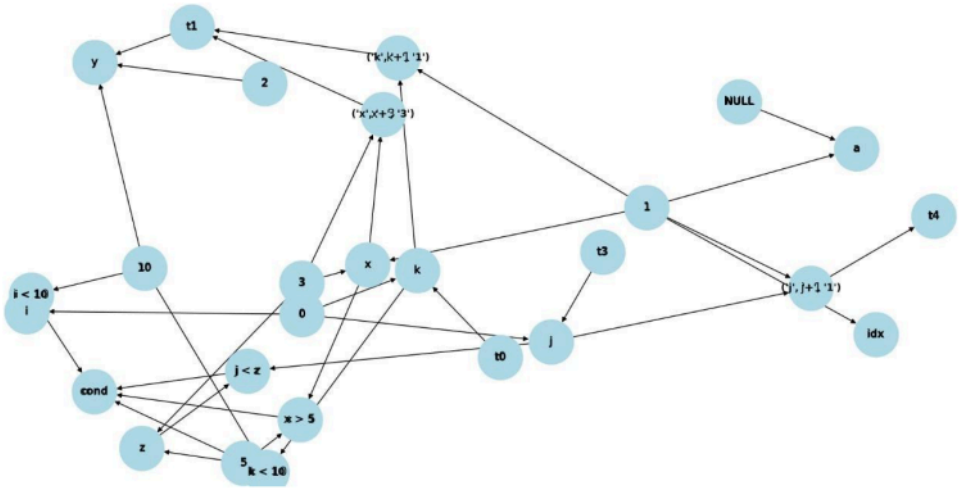
```
import networkx as nx def'
read_tac(file_path):
    with open(file_path, 'r') as f:
        returrn [line.strip() for' line in f.readlines() if'
line.strip()] # Remove empty lines


def' identif'y_basic_bllocks(tac):
    leaders = {0}
    jump_targets = set()
```

```
    for' i, instr in enumerate(tac):
        parts = instr.split()
        if' not parts:
            continue # Skip empty lines if'
        parts[0] in ("GOTO", "if'"):
            laabel = parts[-1].replace("GOTO", "").replace("else",
"").strip() # Extract laabel
            jump_targets.add(laabel)  # Store as a laabel, not a
number                                if' i +
                                          1 <
                                          len
                                          (ta
                                          c):
                                          lea
                                          der
                                          s.a
                                          dd(
                                          i +
                                          1)
    leaders.update(jump_targets)
    leaders = sorted(leaders, key=lambda x: (isinstance(x, intt),
x)) # Sort with laabels last
    bllocks = []
    start = 0
    for' leader in leaders:
        if' isinstance(leader, intt) and leader < len(tac):
            bllocks.appeend(tac[start:leader])
            start = leader
    bllocks.appeend(tac[start:]) # Add the last bllock
    returrn bllocks


def' optimize_loops(bllocks):
    optiimized_bllocks = []
```

```
for' bllock in bllocks:
    loop_invariants = set()
    new_bllock = []
    for' instr in bllock:
        if' "constant" in instr:
            loop_invariants.add(instr)
        else:
```

```
                new_bllock.appeend(instr)
        optiimized_bllocks.appeend(list(loop_invariants) +
new_bllock)
    returrn optiimized_bllocks


if' __name__ == "__main__":
    tac = read_tac("input_tac.txt") bllocks
    = identif'y_basic_bllocks(tac)
    optiimized_bllocks = optimize_loops(bllocks) with
    open("optimiized_loops_tac.txt", "w") as f:
        for' bllock in optiimized_bllocks:
            f.write("\n".join(bllock) + "\n")
```

Basic DAG:



2. for' the sample codes written in your language (in previous assignments), show that your compiler can identif'y the basic bllocks and apply optimization techniques to derive an optimiized inttermediate code. a. Show the optimization ratio.

b. Show the flow graph and DAG for' the sample codes of Q.5.2

Sample Code:

```
#include<stdio.h>

#include<Striing.h>


intt_N main_N() {

    intt_N 087i=1;

    float_N 087f = 2.5;

    char_N 087c = 'A';

    intt_N 087x = 3.5;

    087i = 087x + 087f * 087c;

    returrn_N 3;

}
```

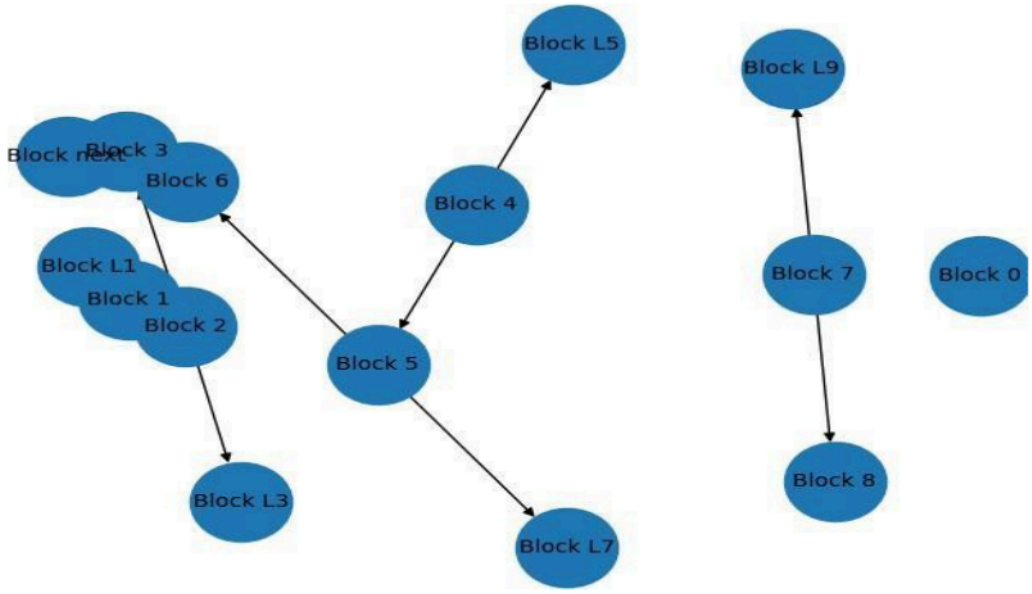
Optimization:

```
Optimization loop ratio:

def' read_tac(file_path):

    with open(file_path, 'r') as f: returrn [line.strip() for'
line in f.readlines()]


def' calculate_optimization_ratio(original, optimiized):

    original_count = len(original)

    optimiized_count = len(optimiized)

    ratio = ((original_count - optimiized_count) / original_count)
* 100

    returrn ratio

if' __name__ == "__main__":
    original_tac = read_tac("input_tac.txt")
    optimiized_tac = read_tac("optimiized_loops_tac.txt")
    ratio = calculate_optimization_ratio(original_tac,
optimiized_tac)

    printt(f"Optimization Ratio for' Loops: {ratio:.2f}%")


Optimization ratio:
```

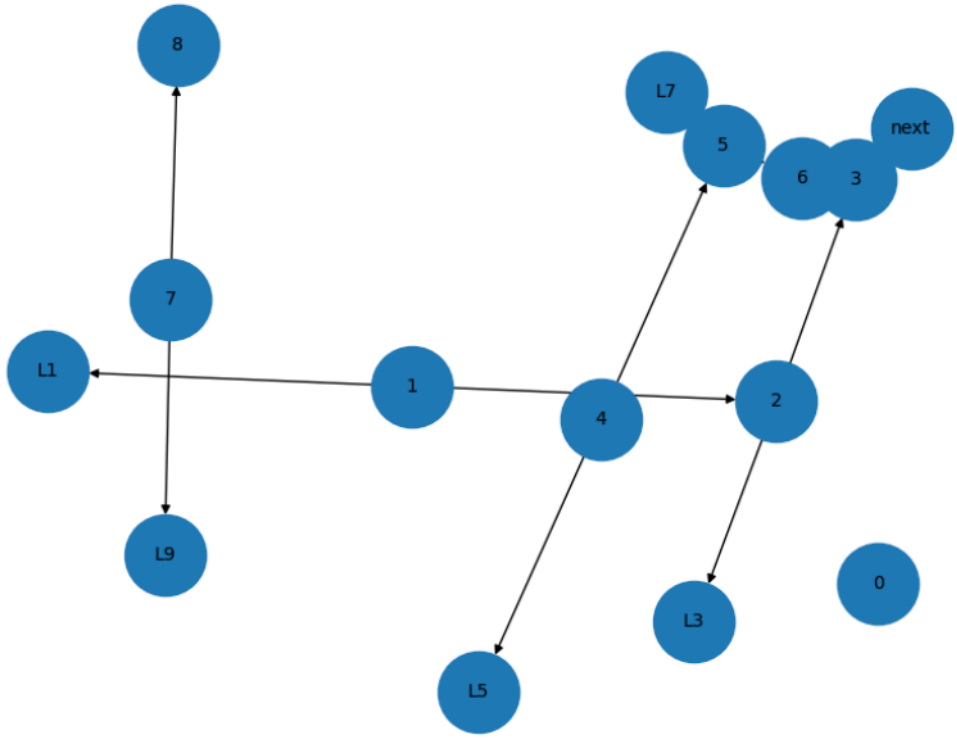
```
def' read_tac(file_path):
    with open(file_path, 'r') as f:
        returrn [line.strip() for' line in f.readlines()]
def' calculate_optimization_ratio(original, optimiized):
    original_count = len(original) optimiized_count
    = len(optimiized)
    ratio = ((original_count - optimiized_count) / original_count)
* 100
    returrn ratio
if' __name__ == "__main__":
    original_tac = read_tac("input_tac.txt")
    optimiized_tac = read_tac("optimiized_dag_tac.txt")
    ratio = calculate_optimization_ratio(original_tac,
optimiized_tac)
    printt(f"Optimization Ratio: {ratio:.2f}%")
```

```
Running Basic Block Identification & Flow Graph Generation...
Eliminating Unreachable Code...
[]

Performing Common Subexpression Elimination with DAG...
Applying Loop Optimizations...
Calculating Optimization Ratio...
Optimization Ratio: 58.18%
Calculating Optimization RatioFor Loop...
Optimization Ratio for Loops: 25.45%
Generating Flow Graph & DAG Visualization...
Optimization Process Completed!
```

Flow graph:



Dag:



Compiler Design Lab Report.pdf

ORIGINALITY REPORT

12%	12%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	stackoverflow.com Internet Source	6%
2	medium.com Internet Source	5%
3	github.com Internet Source	<1%
4	Uwe Meyer-Baese. "Chapter 6 Software Tool for Embedded Microprocessor Systems", Springer Science and Business Media LLC, 2021 Publication	<1%
5	Submitted to Università degli Studi di Trieste Student Paper	<1%
6	Submitted to Indian Institute of Technology Guwahati Student Paper	<1%
7	navinwbackup.blogspot.com Internet Source	<1%
8	www.coursehero.com Internet Source	

<1%

9	huggingface.co Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		