Disclaimer

This PDF contains Compiler Design Lab Experiment source codes with sample outputs. Every C program here has been compiled and thoroughly tested using the GCC compiler in a Linux environment. Please note: these codes are not tested on Windows and are not designed for Turbo C++.

If you're reading this disclaimer, awesome! You now know these codes are meant to run perfectly in a Linux terminal. Trying them on Windows? You might need to tweak a few lines before hitting that compile button.

For those who skip this disclaimer and later say, "Bro, your code isn't working,"—let's just agree the code isn't the problem.

PROGRAM:

SymbolTable.C

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<ctype.h>

void main() {
    int i = 0, j = 0, x = 0, n, flag = 0;
    void *p, *add[15];
    char ch, srch, b[15], d[15], c;

    clrscr();

    printf("expression terminated by $: ");
    while ((c = getchar()) != '$') {
        b[i] = c;
        i++;
    }
    n = i - 1;

    printf("\ngiven expression: ");
    i = 0;
    while (i <= n) {
        printf("%c", b[i]);
        i++;
    }

    printf("\nsymbol table\n");
    printf("\nsymbol\taddr\ttype\n");

    while (j <= n) {
        c = b[j];
        if (isalpha(toascii(c))) {
            if (j == n) {
                p = malloc(c);
                add[x] = p;
                d[x] = c;
                printf("%c\t%d\tidentifier\n", c, p);
            } else {
                ch = b[j + 1];
                if (ch == '/' || ch == '+' || ch == '-' || ch == '*' || ch == '=') {
                    p = malloc(c);
                    add[x] = p;
                    d[x] = c;
                    printf("%c\t%d\tidentifier\n", c, p);
                    x++;
                }
            }
        }
        j++;
    }

    printf("Enter the identifier to be searched\n");
    srch = getch();
    for (i = 0; i <= x; i++) {
        if (srch == d[i]) {
            printf("symbol found\n");
            printf("%c @Adrress %d \n", srch, add[i]);
            flag = 1;
        }
    }

    if (flag == 0)
        printf("symbol not found\n");

    getch();
}
```

OUTPUT:

```
Expression Terminated by $: a+b+c=d$

Given Expression: a+bc=d
Symbol table

Symbol      addr    type
a     1904    identifier
b     2006    identifier
c     2108    identifier
d     2212    identifier
Enter the identifier to be searched
Symbol found
c @Address 2108
```

PROGRAM:

PatternRecognize.C

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

void keyword(char str[10]) {
   if (strcmp("for", str) == 0 || strcmp("while", str) == 0 || strcmp("do", str) == 0 ||
      strcmp("int", str) == 0 || strcmp("float", str) == 0 || strcmp("char", str) == 0 ||
      strcmp("double", str) == 0 || strcmp("static", str) == 0 || strcmp("switch", str) == 0 ||
      strcmp("case", str) == 0 || strcmp("void", str) == 0 || strcmp("printf", str) == 0) {
      printf("\n%s is a keyword", str);
   } else {
      printf("\n%s is an identifier", str);
   }
}

void main() {
   FILE *f1, *f2, *f3;
   char c, str[10], st1[10];
   int num[100], lineno = 0, tokenvalue = 0, i = 0, j = 0, k = 0;

   clrscr();
   printf("\nEnter the C program: ");

   f1 = fopen("input", "w");
   while ((c = getchar()) != EOF)
      putc(c, f1);
   fclose(f1);

   f1 = fopen("input", "r");
   f2 = fopen("identifier", "w");
   f3 = fopen("specialchar", "w");

   while ((c = getc(f1)) != EOF) {
      if (isdigit(c)) {
         tokenvalue = c - '0';
         c = getc(f1);
         while (isdigit(c)) {
            tokenvalue = tokenvalue * 10 + c - '0';
            c = getc(f1);
         }
         num[i++] = tokenvalue;
         ungetc(c, f1);
      } else if (isalpha(c)) {
         putc(c, f2);
         c = getc(f1);
         while (isdigit(c) || isalpha(c) || c == '_' || c == '$') {
            putc(c, f2);
            c = getc(f1);
         }
```

```
            putc(' ', f2);
            ungetc(c, f1);
        } else if (c == ' ' || c == '\t') {
            printf(" ");
        } else if (c == '\n') {
            lineno++;
        } else {
            putc(c, f3);
        }
    }

    fclose(f2);
    fclose(f3);
    fclose(f1);

    printf("\nThe numbers in the program are: ");
    for (j = 0; j < i; j++) {
        printf("%d ", num[j]);
    }
    printf("\n");

    f2 = fopen("identifier", "r");
    k = 0;
    printf("The keywords and identifiers are: ");
    while ((c = getc(f2)) != EOF) {
        if (c != ' ') {
            str[k++] = c;
        } else {
            str[k] = '\0';
            keyword(str);
            k = 0;
        }
    }
    fclose(f2);

    f3 = fopen("specialchar", "r");
    printf("\nSpecial characters are: ");
    while ((c = getc(f3)) != EOF) {
        printf("%c\t", c);
    }
    printf("\n");
    fclose(f3);

    printf("Total number of lines are: %d", lineno);
    getch();
}
```

OUTPUT:

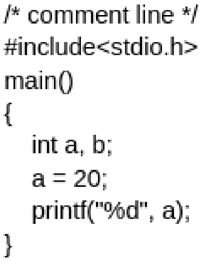
```
Enter the C program..
void main()
{
   int n1,n2;
   float cal;
   cal=n1+n2/17;
   return cal;
}→

The numbers in the program are: 17
The keywords and identifiers are:
void is a keyword
main is an identifier
int is a keyword
n1 is an identifier
n2 is an identifier
float is a keyword
cal is an identifier
cal is an identifier
n1 is an identifier
n2 is an identifier
return is an keyword
cal is an identifier
Special characters are: (          )          {          ,          ;          ;          =          +
/      ;          ;          }
Total number of lines are: 6
```

PROGRAM:

lexprogram.l

```
%{
 /* program to recognize a C program */
 int COMMENT = 0;
%}
identifier [a-zA-Z_][a-zA-Z0-9_]*
%%
#.*                          { printf("\n%s is a PREPROCESSOR DIRECTIVE", yytext); }
int | float | char | double | while | for | do | if | break | continue | void | switch | case | long | struct | const | typedef
| return | else | goto
                              { printf("\n\t%s is a KEYWORD", yytext); }
"/*"                          { COMMENT = 1; }
"*/"                          { COMMENT = 0; }
{identifier}\(                { if (!COMMENT) printf("\n\nFUNCTION\n\t%s", yytext); }
\{                            { if (!COMMENT) printf("\nBLOCK BEGINS"); }
\}                            { if (!COMMENT) printf("\nBLOCK ENDS"); }
{identifier}(\[[0-9]*\])?     { if (!COMMENT) printf("\n%s is an IDENTIFIER", yytext); }
\".*\"                        { if (!COMMENT) printf("\n\t%s is a STRING", yytext); }
[0-9]+                        { if (!COMMENT) printf("\n\t%s is a NUMBER", yytext); }
\)(\;)?                       { if (!COMMENT) { printf("\n\t"); ECHO; printf("\n"); } }
\(                            { ECHO; }
=                             { if (!COMMENT) printf("\n\t%s is an ASSIGNMENT OPERATOR", yytext); }
\<= | \>= | \< | == | \>      { if (!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR", yytext); }
%%
int main(int argc, char **argv)
{
   if (argc > 1)
   {
      FILE *file;
      file = fopen(argv[1], "r");
      if (!file)
      {
         printf("Could not open %s \n", argv[1]);
         exit(0);
      }
      yyin = file;
   }
   yylex();
   printf("\n\n");
   return 0;
}
int yywrap()
{
   return 0;
}
```

Input.txt



```
/* comment line */
#include<stdio.h>
main()
{
   int a, b;
   a = 20;
   printf("%d", a);
}
```

OUTPUT:

```
>>flex program.l
>>gcc lex.yy.c
>>./a.out input.txt

#include<stdio.h> is a PREPROCESSOR DIRECTIVE
FUNCTION
        main(
        )
BLOCK BEGINS
int is a KEYWORD
a is an IDENTIFIER,
b is an IDENTIFIER;
a is an IDENTIFIER
= is an ASSIGNMENT OPERATOR
20 is a NUMBER;
FUNCTION
        printf(
        "%d" is a STRING,
        a is an IDENTIFIER
        );
BLOCK ENDS
```

PROGRAM:

arithexp.y

```
%{
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
%}
%token num let
%left '+' '-'
%left '*' '/'
%%
Stmt : Stmt '\n'
    {
       printf("\n.. Valid Expression ..\n");
       exit(0);
    }
   | expr
   | error '\n'
    {
       printf("\n.. Invalid Expression ..\n");
       exit(0);
    };
expr : num
   | let
   | expr '+' expr
   | expr '-' expr
   | expr '*' expr
   | expr '/' expr
   | '(' expr ')'
   ;
%%
int main() {
   printf("Enter an expression to validate: ");
   yyparse();
   return 0;
}
int yylex() {
   int ch;
   while ((ch = getchar()) == ' ');  // Skip spaces
   if (isdigit(ch)) return num;  // Return token num
   if (isalpha(ch)) return let;  // Return token let
   return ch;
}
void yyerror(char *s) {
   printf("%s\n", s);
}
```

OUTPUT:

```
>>bison -d arithexp.y
>>gcc arithexp.tab.c
>>a.exe
Enter an expression to validate: (a+b)/c**d
Error: syntax error

.. Invalid Expression ..

>>a.exe
Enter an expression to validate: a+b*c/d

.. Valid Expression ..
```

PROGRAM:

variablevalid.y

```
%{
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
%}
%token let dig
%%
TERM : XTERM '\n'
    {
        printf("\nAccepted\n");
        exit(0);
    }
    | error
    {
        printf("\nRejected\n");
        exit(0);
    };
XTERM : XTERM let
      | XTERM dig
      | let;
%%
int yylex()
{
    char ch;
    while ((ch = getchar()) == ' ' || ch == '\t');
    if (isalpha(ch))
        return let;
    if (isdigit(ch))
        return dig;
    return ch;
}
int main()
{
    printf("Enter a variable: ");
    yyparse();
    return 0;
}
void yyerror(const char *s)
{
    printf("Error: %s\n", s);
}
```

OUTPUT:

```
>>bison -d variablevalid.y
>>gcc variablevalid.tab.c
>>a.exe
Enter a variable: boLt124

Accepted

>>a.exe
Enter a variable: 17-boLt-03
Error: syntax error

Rejected
```

PROGRAM:

calc.y

```
%{
#include <stdio.h>
#include <stdlib.h>
int regs[26];
int base;
%}
%start list
%token DIGIT LETTER
%left '|'
%left '&'
%left '+' '-'
%left '*' '/' '%'
%left UMINUS /* Supplies precedence for unary minus */
%%
list: /* empty */
   | list stat '\n'
   | list error '\n'
   {
      yyerrok;
   };
stat: expr
   {
      printf("%d\n", $1);
   }
   | LETTER '=' expr
   {
      regs[$1] = $3;
   };
expr: '(' expr ')'
   {
      $$ = $2;
   }
   | expr '*' expr
   {
      $$ = $1 * $3;
   }
   | expr '/' expr
   {
      $$ = $1 / $3;
   }
   | expr '%' expr
   {
      $$ = $1 % $3;
   }
   | expr '+' expr
   {
      $$ = $1 + $3;
   }
   | expr '-' expr
```

```
    {
        $$ = $1 - $3;
    }
    | expr '&' expr
    {
        $$ = $1 & $3;
    }
    | expr '|' expr
    {
        $$ = $1 | $3;
    }
    | '-' expr %prec UMINUS
    {
        $$ = -$2;
    }
    | LETTER
    {
        $$ = regs[$1];
    }
    | number;
number: DIGIT
    {
        $$ = $1;
        base = ($1 == 0) ? 8 : 10;
    }
    | number DIGIT
    {
        $$ = base * $1 + $2;
    };
%%
int main(){
    return yyparse();
}
void yyerror(const char *s){
    fprintf(stderr, "Error: %s\n", s);
}
int yywrap(){
    return 1;
}
```

OUTPUT:

```
>>bison -d calc.y
>>flex calc.l
>>gcc lex.yy.c calc.tab.c
>>a.exe
8 / 2
4
(3 + 4) * 2
14
3 + 4 * 2 - (6 / 2) | 5
7
3 +
Error: syntax error
```

PROGRAM:

lexbnf.l

```
%{
#include "yaccbnf.tab.h"
#include <stdio.h>
#include <string.h>
int LineNo = 1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\)          return MAIN;
if                return IF;
else              return ELSE;
while             return WHILE;
int |
char |
float             return TYPE;
{identifier}      { strcpy(yylval.var, yytext); return VAR; }
{number}          { strcpy(yylval.var, yytext); return NUM; }
\< |
\> |
\>= |
\<= |
==                { strcpy(yylval.var, yytext); return RELOP; }
[ \t]             ;
\n                LineNo++;
.                 return yytext[0];
%%
int yywrap() {
   return 1;
}
```

yaccbnf.y

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct quad {
   char op[5];
   char arg1[10];
   char arg2[10];
   char result[10];
} QUAD[30];
struct stack {
   int items[100];
   int top;
} stk;
int Index = 0, tIndex = 0, StNo, Ind, tInd;
extern int LineNo;
void push(int data);
int pop();
```

```
void AddQuadruple(char op[5], char arg1[10], char arg2[10], char result[10]);
%}
%union {
   char var[10];
}
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK ;
BLOCK : '{' CODE '}' ;
CODE : BLOCK
    | STATEMENT CODE
    | STATEMENT ;
STATEMENT : DESCT ';'
         | ASSIGNMENT ';'
         | CONDST
         | WHILEST ;
DESCT : TYPE VARLIST ;
VARLIST : VAR ',' VARLIST
       | VAR ;
ASSIGNMENT : VAR '=' EXPR {
   strcpy(QUAD[Index].op, "=");
   strcpy(QUAD[Index].arg1, $3);
   strcpy(QUAD[Index].arg2, "");
   strcpy(QUAD[Index].result, $1);
   strcpy($$, QUAD[Index++].result);
};
EXPR : EXPR '+' EXPR { AddQuadruple("+", $1, $3, $$); }
    | EXPR '-' EXPR { AddQuadruple("-", $1, $3, $$); }
    | EXPR '*' EXPR { AddQuadruple("*", $1, $3, $$); }
    | EXPR '/' EXPR { AddQuadruple("/", $1, $3, $$); }
    | '-' EXPR      { AddQuadruple("UMIN", $2, "", $$); }
    | '(' EXPR ')'  { strcpy($$, $2); }
    | VAR
    | NUM ;
CONDST : IFST {
   Ind = pop();
   sprintf(QUAD[Ind].result, "%d", Index);
   Ind = pop();
   sprintf(QUAD[Ind].result, "%d", Index);
}
| IFST ELSEST ;
IFST : IF '(' CONDITION ')' {
   strcpy(QUAD[Index].op, "==");
   strcpy(QUAD[Index].arg1, $3);
   strcpy(QUAD[Index].arg2, "FALSE");
   strcpy(QUAD[Index].result, "-1");
   push(Index++);
} BLOCK {
   strcpy(QUAD[Index].op, "GOTO");
   strcpy(QUAD[Index].arg1, "");
   strcpy(QUAD[Index].arg2, "");
   strcpy(QUAD[Index].result, "-1");
   push(Index++);
};
ELSEST : ELSE {
   tInd = pop();
   Ind = pop();
   push(tInd);
   sprintf(QUAD[Ind].result, "%d", Index);
} BLOCK {
   Ind = pop();
   sprintf(QUAD[Ind].result, "%d", Index);
};
CONDITION : VAR RELOP VAR {
   AddQuadruple($2, $1, $3, $$);
   StNo = Index - 1;
}
| VAR
| NUM ;
WHILEST : WHILELOOP {
   Ind = pop();
   sprintf(QUAD[Ind].result, "%d", StNo);
   Ind = pop();
   sprintf(QUAD[Ind].result, "%d", Index);
};
WHILELOOP : WHILE '(' CONDITION ')' {
   strcpy(QUAD[Index].op, "==");
   strcpy(QUAD[Index].arg1, $3);
   strcpy(QUAD[Index].arg2, "FALSE");
   strcpy(QUAD[Index].result, "-1");
   push(Index++);
} BLOCK {
   strcpy(QUAD[Index].op, "GOTO");
   strcpy(QUAD[Index].arg1, "");
   strcpy(QUAD[Index].arg2, "");
   strcpy(QUAD[Index].result, "-1");
   push(Index++);
};
%%
int main(int argc, char *argv[]) {
   FILE *fp;
   int i;
   stk.top = -1;
   if (argc > 1) {
       fp = fopen(argv[1], "r");
       if (!fp) {
           printf("\nFile not found");
           exit(0);
       }
       yyin = fp;
   }
   yyparse();
   printf("\n\n\t\t --------------------------------------");
   printf("\n\t\t Pos Operator\t Arg1\t Arg2\t Result");
   printf("\n\t\t --------------------------------------");
   for (i = 0; i < Index; i++) {
       printf("\n\t\t %d\t %s\t %s\t %s\t %s", i, QUAD[i].op, QUAD[i].arg1, QUAD[i].arg2, QUAD[i].result);
   }
   printf("\n\t\t --------------------------------------\n\n");
   return 0;
}
void push(int data) {
   if (++stk.top == 100) {
       printf("\nStack overflow\n");
       exit(0);
   }
   stk.items[stk.top] = data;
}
int pop() {
   if (stk.top == -1) {
       printf("\nStack underflow\n");
       exit(0);
   }
   return stk.items[stk.top--];
}
void AddQuadruple(char op[5], char arg1[10], char arg2[10], char result[10]) {
   strcpy(QUAD[Index].op, op);
   strcpy(QUAD[Index].arg1, arg1);
   strcpy(QUAD[Index].arg2, arg2);
   sprintf(QUAD[Index].result, "t%d", tIndex++);
   strcpy(result, QUAD[Index++].result);
}
void yyerror() {
   printf("\nError on line no: %d", LineNo);
}
```

input.c

```
main()
{
    int a, b, c;
    a = 10;
    b = 20;
    if (a < b)
    {
        a = a + b;
    }
    while (a < b)
    {
        a = a + b;
    }
    if (a <= b)
    {
        c = a - b;
    }
    else
    {
        c = a + b;
    }
}
```

OUTPUT:

```
>>bison -d yaccbnf.y
>>flex lexbnf.l
>>gcc lex.yy.c yaccbnf.tab.c
>>a.exe
```

```
        ----------------------------------------------------
        Pos   Operator    Arg1      Arg2      Result
        ----------------------------------------------------
        0        =         10                    a
        1        =         20                    b
        2        <          a         b          t0
        3        ==        t0       FALSE        7
        4        +          a         b          t1
        5        =         t1                    a
        6      GOTO                              7
        7        <          a         b          t2
        8        ==        t2       FALSE        12
        9        +          a         b          t3
        10       =         t3                    a
        11     GOTO                              7
        12       <=         a         b          t4
        13       ==        t4       FALSE        17
        14       -          a         b          t5
        15       =         t5                    c
        16     GOTO                              19
        17       +          a         b          t6
        18       =         t6                    c
        ----------------------------------------------------
```

PROGRAM:

datatype_checker.c

```
#include <stdio.h>
void main() {
   int n, i, k, flag = 0;
   char vari[15], typ[15], b[15], c;
   printf("Enter the number of variables: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
      printf("Enter the variable[%d]: ", i);
      scanf(" %c", &vari[i]);
      printf("Enter the variable-type[%d] (float-f, int-i): ", i);
      scanf(" %c", &typ[i]);
      if (typ[i] == 'f')
         flag = 1;
   } printf("Enter the Expression (end with $): ");
   i = 0;
   getchar(); // to consume the newline character
   while ((c = getchar()) != '$') {
      b[i] = c;
      i++;
   } k = i;
   for (i = 0; i < k; i++) {
      if (b[i] == '/') {
         flag = 1;
         break;
      }
   } for (i = 0; i < n; i++) {
      if (b[0] == vari[i]) {
         if (flag == 1) {
            if (typ[i] == 'f') {
               printf("\nThe datatype is correctly defined..!\n");
               break;
            } else {
               printf("Identifier %c must be a float type..!\n", vari[i]);
               break;
            }
         } else {
            printf("\nThe datatype is correctly defined..!\n");
            break;
         }
      }
   }
}
```

OUTPUT:

```
>>gcc datatype_checker.c
>>a.exe

Enter the number of variables: 4
Enter the variable[0]: x
Enter the variable-type[0](float-f,int-i): i
Enter the variable[1]: y
Enter the variable-type[1](float-f,int-i): i
Enter the variable[2]: z
Enter the variable-type[2](float-f,int-i): f
Enter the variable[3]: w
Enter the variable-type[3](float-f,int-i): f
Enter the Expression(end with $): x*y/z+w$

Identifier x must be a float type..!
```

PROGRAM:

stack.c

```
#include <stdio.h>

int stack[100], choice, n, top, x, i;

void push(void);
void pop(void);
void display(void);

void main() {
   top = -1;

   printf("\n Enter the size of STACK [MAX=100]: ");
   scanf("%d", &n);

   printf("\n\t STACK OPERATIONS USING ARRAY");
   printf("\n\t--------------------------------");
   printf("\n\t 1. PUSH");
   printf("\n\t 2. POP");
   printf("\n\t 3. DISPLAY");
   printf("\n\t 4. EXIT");

   do {
      printf("\n Enter the Choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1: {
            push();
            break;
         }
         case 2: {
            pop();
            break;
         }
         case 3: {
            display();
            break;
         }
         case 4: {
            printf("\n\t EXIT POINT ");
            break;
         }
         default: {
            printf("\n\t Please Enter a Valid Choice (1/2/3/4)");
         }
      }
   } while (choice != 4);
}
```

```
void push() {
   if (top >= n - 1) {
      printf("\n\t STACK is Overflow");
   } else {
      printf(" Enter a value to be pushed: ");
      scanf("%d", &x);
      top++;
      stack[top] = x;
   }
}

void pop() {
   if (top <= -1) {
      printf("\n\t STACK is Underflow");
   } else {
      printf("\n\t The popped element is %d", stack[top]);
      top--;
   }
}

void display() {
   if (top >= 0) {
      printf("\n The elements in STACK are:\n");
      for (i = top; i >= 0; i--) {
         printf("\n %d", stack[i]);
      }
      printf("\n Press Next Choice");
   } else {
      printf("\n The STACK is Empty");
   }
}
```

OUTPUT:

```
>>gcc stack.c
>>a.exe

 Enter the size of STACK [MAX=100]: 5

        STACK OPERATIONS USING ARRAY
       -------------------------------
        1. PUSH
        2. POP
        3. DISPLAY
        4. EXIT

 Enter the Choice: 1
 Enter a value to be pushed: 15

 Enter the Choice: 1
 Enter a value to be pushed: 17

 Enter the Choice: 3

 The elements in STACK are:
 15
 17
 Press Next Choice

 Enter the Choice: 2

 The popped element is 15

 Enter the Choice: 4

        EXIT POINT
```

PROGRAM:

dag.c

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

struct da {
   int ptr, left, right;
   char label;
} dag[25];

int main() {
   int ptr, j, n = 0, i = 0, x, k;
   char store, input1[25], input[25], var;

   // Initialize DAG and arrays
   for (i = 0; i < 25; i++) {
      dag[i].ptr = 0;
      dag[i].left = 0;
      dag[i].right = 0;
      dag[i].label = '\0';
      input1[i] = '\0';
      input[i] = '\0';
   }

   printf("Hint: Provide the expression inside parentheses stating the priority.\n");
   printf("For example: a+b*c is given as (a+(b*c))\n");
   printf("ENTER THE EXPRESSION: ");
   scanf("%s", input1);

   while (1) {
      for (i = 0; input1[i] != '\0'; i++) {
         if (input1[i] == ')')
            break;
      }
      if (input1[i] == '\0')  // No closing parenthesis left
         break;

      for (j = i; input1[j] != '('; j--);

      for (x = j + 1; x < i; x++) {
         if (isalpha(input1[x]))
            input[n++] = input1[x];
         else if (input1[x] != '0')
            store = input1[x];
      }

      input[n++] = store;

      for (x = j; x <= i; x++)
         input1[x] = '0';
   }
```

```
    for (i = 0; i < n; i++) {
        dag[i].label = input[i];
        dag[i].ptr = i;

        if (!isalpha(input[i]) && !isdigit(input[i])) {
            dag[i].right = i - 1;
            ptr = i;
            var = input[i - 1];

            if (isalpha(var))
                ptr = ptr - 2;
            else {
                ptr = i - 1;
                while (1) {
                    if (!isalpha(var) && !isdigit(var)) {
                        ptr = dag[ptr].left;
                        var = input[ptr];
                    } else {
                        ptr = ptr - 1;
                        break;
                    }
                }
            }

            dag[i].left = ptr;
        }
    }

    printf("\nDAG FOR GIVEN EXPRESSION");
    printf("\n\nPTR\tLEFT PTR\tRIGHT PTR\tLABEL");
    for (i = 0; i < n; i++) {
        printf("\n%d\t%d\t\t%d\t\t%c", dag[i].ptr, dag[i].left, dag[i].right, dag[i].label);
    }

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

OUTPUT:

```
>>gcc dag.c
>>a.exe

Hint: Provide the expression inside parentheses stating the priority.
For example: a+b*c is given as (a+(b*c))
ENTER THE EXPRESSION: (((a+b)/(c-d))/(a+b))

DAG FOR GIVEN EXPRESSION

PTR     LEFT PTR      RIGHT PTR       LABEL
0           0             0             a
1           0             0             b
2           0             1             +
3           0             0             c
4           0             0             d
5           3             4             -
6           2             5             /
7           0             0             a
8           0             0             b
9           7             8             +
10          6             9             /
```

PROGRAM:

optimise.c

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

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

int main() {
    int a, i, k, j, n, z = 0, m, q;
    char *p, *l;
    char temp, t;
    char *tem;
    printf("Enter number of values: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("\tLeft: ");
        scanf(" %c", &op[i].l);  // Space before %c to skip whitespace
        printf("\tRight: ");
        scanf("%s", op[i].r);
    }
    printf("\nIntermediate Code:\n");
    for (i = 0; i < n; i++) {
        printf("%c = %s\n", op[i].l, op[i].r);
    }

    // Dead Code Elimination
    for (i = 0; i < n - 1; i++) {
        temp = op[i].l;
        for (j = 0; j < n; j++) {
            p = strchr(op[j].r, temp);
            if (p) {
                pr[z].l = op[i].l;
                strcpy(pr[z].r, op[i].r);
                z++;
                break;
            }
        }
    }
    pr[z].l = op[n - 1].l;
    strcpy(pr[z].r, op[n - 1].r);
    z++;
    printf("\nAfter Dead Code Elimination:\n");
    for (k = 0; k < z; k++) {
        printf("%c = %s\n", pr[k].l, pr[k].r);
    }
    for (m = 0; m < z; m++) {
        tem = pr[m].r;
        for (j = m + 1; j < z; j++) {
            if (strcmp(tem, pr[j].r) == 0) {
                t = pr[j].l;
                pr[j].l = pr[m].l;
                for (i = 0; i < z; i++) {
                    l = strchr(pr[i].r, t);
                    if (l) {
                        a = l - pr[i].r;
                        pr[i].r[a] = pr[m].l;
                    }
                }
            }
        }
    }
    printf("\nAfter Common Subexpression Elimination:\n");
    for (i = 0; i < z; i++) {
        printf("%c = %s\n", pr[i].l, pr[i].r);
    }
    for (i = 0; i < z; i++) {
        for (j = i + 1; j < z; j++) {
            q = strcmp(pr[i].r, pr[j].r);
            if ((pr[i].l == pr[j].l) && !q) {
                pr[j].l = '\0';
                strcpy(pr[j].r, "");
            }
        }
    }
    printf("\nOptimized Code:\n");
    for (i = 0; i < z; i++) {
        if (pr[i].l != '\0') {
            printf("%c = %s\n", pr[i].l, pr[i].r);
        }
    }
    return 0;
}
```

OUTPUT:

```
>>gcc optimise.c
>>a.exe

Enter number of values: 4
        Left: a
        Right: 5
        Left: b
        Right: c+d
        Left: e
        Right: c+d
        Left: q
        Right: b+e

Intermediate Code:
a = 5
b = c+d
e = c+d
q = b+e

After Dead Code Elimination:
b = c+d
e = c+d
q = b+e

After Common Subexpression Elimination:
b = c+d
b = c+d
q = b+b

Optimized Code:
b = c+d
q = b+b
```

PROGRAM:

basic_blocks.c

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

struct Listnode {
   char data[50];
   int leader, block, u_goto, c_goto;
   struct Listnode *next;
   char label[10], target[10];
} *temp, *cur, *first = NULL, *last = NULL, *cur1;

void createnode(char code[50]) {
   temp = (struct Listnode *)malloc(sizeof(struct Listnode));
   strcpy(temp->data, code);
   strcpy(temp->label, "");
   strcpy(temp->target, "");
   temp->leader = 0;
   temp->block = 0;
   temp->u_goto = 0;
   temp->c_goto = 0;
   temp->next = NULL;

   if (first == NULL) {
      first = temp;
      last = temp;
   } else {
      last->next = temp;
      last = temp;
   }
}

int main() {
   char codeline[50];
   char c, dup[50], target[10];
   char *substring, *token;
   int i = 0, block, block1;
   int j = 0;

   FILE *fpr = fopen("cdp.txt", "r");
   if (fpr == NULL) {
      printf("Error: Unable to open file cdp.txt\n");
      return 1;
   }

   while ((c = getc(fpr)) != EOF) {
      if (c != '\n') {
         codeline[i++] = c;
      } else {
         codeline[i] = '\0';
```

```
        createnode(codeline);
        i = 0;
    }
}
if (i > 0) {
    codeline[i] = '\0';
    createnode(codeline);
}
fclose(fpr);

// Identify leaders and goto statements
cur = first;
cur->leader = 1;
while (cur != NULL) {
    if (strstr(cur->data, "if") != NULL) {
        cur->c_goto = 1;
        if (cur->next != NULL)
            cur->next->leader = 1;
    } else if (strstr(cur->data, "goto") != NULL) {
        cur->u_goto = 1;
        if (cur->next != NULL)
            cur->next->leader = 1;
    } else if (strstr(cur->data, "call") != NULL || strstr(cur->data, "return") != NULL) {
        cur->leader = 1;
        if (cur->next != NULL)
            cur->next->leader = 1;
    }
    if (strchr(cur->data, ':') != NULL) {
        cur->leader = 1;
    }
    cur = cur->next;
}

// Identify labels and targets
cur = first;
while (cur != NULL) {
    if (cur->u_goto || cur->c_goto) {
        substring = strchr(cur->data, ':');
        if (substring != NULL) {
            token = strstr(substring, "L");
            if (token != NULL)
                strcpy(cur->target, token);
        } else {
            substring = strstr(cur->data, "L");
            if (substring != NULL)
                strcpy(cur->target, substring);
        }
    }
    if (strchr(cur->data, ':') != NULL) {
        strcpy(dup, cur->data);
        token = strtok(dup, ":");
        if (token != NULL)
```

```
            strcpy(cur->label, token);
    }
    cur = cur->next;
}

// Identify basic blocks
cur = first;
j = 0;
while (cur != NULL) {
    if (cur->leader)
        j++;
    cur->block = j;
    cur = cur->next;
}

// Print basic blocks
printf("\n\n......Basic Blocks......\n");
cur = first;
int current_block = -1;
while (cur != NULL) {
    if (cur->block != current_block) {
        current_block = cur->block;
        printf("\nBlock %d:\n", current_block);
    }
    printf("%s\n", cur->data);
    cur = cur->next;
}

// Print control flow
printf("\n\t\t.......Control Flow.......\n\n");
cur = first;
while (cur != NULL) {
    if (cur->next == NULL || cur->block != cur->next->block) {
        block = cur->block;
        if (cur->u_goto) {
            strcpy(target, cur->target);
            cur1 = first;
            while (cur1 != NULL) {
                if (strcmp(cur1->label, target) == 0) {
                    block1 = cur1->block;
                    printf("Block %d ----------> Block %d\n", block, block1);
                    break;
                }
                cur1 = cur1->next;
            }
        } else if (cur->c_goto) {
            strcpy(target, cur->target);
            cur1 = first;
            while (cur1 != NULL) {
                if (strcmp(cur1->label, target) == 0) {
                    block1 = cur1->block;
                    printf("Block %d ---TRUE---> Block %d ---FALSE---> Block %d\n", block, block1, block + 1);
```

```
                        break;
                    }
                    cur1 = cur1->next;
                }
            } else if (strstr(cur->data, "return") == NULL) {
                printf("Block %d ----------> Block %d\n", block, block + 1);
            } else {
                printf("Block %d ----------> NULL\n", block);
            }
        }
        cur = cur->next;
    }
    printf("Block %d ----------> NULL\n", last->block);

    return 0;
}
```

cdp.txt

```
m <-0
v <-0
L1:if v<n goto L2
r <-v
s <-0
return
L2:if r>=n goto L1
v <-v+1
```

OUTPUT:

```
>>gcc basic_blocks.c
>>a.exe

......Basic Blocks......

Block 1:
m <-0
v <-0

Block 2:
L1:if v<n goto L2

Block 3:
r <-v
s <-0

Block 4:
return

Block 5:
L2:if r>=n goto L1

Block 6:
v <-v+1


                    .......Control Flow.......

Block 1 --------------> Block 2
Block 2 ---TRUE---> Block 5 ---FALSE---> Block 3
Block 3 --------------> Block 4
Block 4 --------------> NULL
Block 5 ---TRUE---> Block 2 ---FALSE---> Block 6
Block 6 --------------> Block 7
Block 6 --------------> NULL
```

PROGRAM:

data_flow_analysis.c

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

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

int main() {
   int n, i, j, lineno = 1;
   char *match;

   printf("Enter number of values: ");
   scanf("%d", &n);

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

   printf("\nIntermediate Code:\n");
   for (i = 0; i < n; i++) {
      printf("Line No = %d\n", lineno);
      printf("\t%s = %s\n", op[i].l, op[i].r);
      lineno++;
   }

   printf("\n*** Data Flow Analysis for the Above Code ***\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         match = strstr(op[j].r, op[i].l);
         if (match) {
            printf("\n %s is live at %s\n", op[i].l, op[j].r);
         }
      }
   }

   return 0;
}
```

OUTPUT:

```
>>gcc data_flow_analysis.c
>>a.exe

Enter number of values: 4
        Left: a
        Right: a+b
        Left: b
        Right: a+c
        Left: c
        Right: a+b
        Left: d
        Right: b+c+d

Intermediate Code:
Line No = 1
        a = a+b
Line No = 2
        b = a+c
Line No = 3
        c = a+b
Line No = 4
        d = b+c+d

*** Data Flow Analysis for the Above Code ***

 a is live at a+b

 a is live at a+c

 a is live at a+b

 b is live at a+b

 b is live at a+b

 b is live at b+c+d

 c is live at a+c

 c is live at b+c+d

 d is live at b+c+d
```