**[HINT: Here I haven't done 2 pass Assembler, Code Optimization and First and Follow ]**

1. All three programs have to be performed (5 marks for each output)
2. Write a program to create your own ‘C’ library using macros that can find the area of geometrical shapes (any 4).

**Area.h**

#define pi 3.14

#define C\_area(r) pi \* r \* r

#define S\_area(side) side \* side

#define R\_area(length, width) length \* width

#define T\_area(base, height)  0.5 \* base \* height

**Area.c**

#include <stdio.h>

#include "Area.h"

int main()

{

        int n;

        float radius, length, width, base, height, side;

        float A\_circle, A\_rectangle, A\_triangle, A\_square;

        printf("Enter the choice: \n1. Circle\n2. Rectangle\n3. Square\n4. Triangle\n");

        scanf("%d",&n);

        switch(n)

        {

                case 1:

                        printf("Enter the radius: ");

                        scanf("%f",&radius);

                        A\_circle = C\_area(radius);

                        printf("Area of the Circle: %f", A\_circle);

                        break;

                case 2:

                        printf("Enter the length and width: ");

                        scanf("%f%f",&length,&width);

                        A\_rectangle = R\_area(length,width);

                        printf("Area of the Rectangle: %f", A\_rectangle);

                        break;

                case 3:

                        printf("Enter the side: ");

                        scanf("%f",&side);

                        A\_square = S\_area(side);

                        printf("Area of the Square: %f", A\_square);

                        break;

                case 4:

                        printf("Enter the base and height: ");

                        scanf("%f%f",&base,&height);

                        A\_triangle = T\_area(base,height);

                        printf("Area of the Triangle: %f", A\_triangle);

                        break;

                default:

                                   printf("Wrong Choice");

        }

        return 0;

}

1. Write a Lex program to show the vowels and consonants of a token

%option noyywrap

%{

        #include<stdio.h>

        int V=0, C=0;

%}

%%

[AEOIUaeiou] {V++;}

[^AEOIUaeiou\n] {C++;}

\n {printf("Vowels = %d \nConsanants = %d",V,C);}

%%

int main()

{

        printf("Enter string: ");

        yylex();

        return 0;

}

1. Write a program to convert the given computation into three address code.: [ x = (a+b) \* (c-d) ]

import java.util.\*;

public class Main{

    public static void qQuadruple(String exp[]){

        System.out.println("op\targ1\targ2\tresult");

        for (int i=0;i < exp.length; i++){

            String[]expr = exp[i].split("");

            System.out.println(expr[3] + "\t" + expr[2] + "\t" + expr[4] + "\t" + expr[0]);

        }

    }

    public static void rTriple(String exp[]){

        System.out.println("op\targ1\targ2");

        for (int i=0;i < exp.length; i++){

            String[]expr = exp[i].split("");

            System.out.println(expr[3] + "\t" + expr[2] + "\t" + expr[4]);

        }

    }

    public static void main(String args[]){

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of expressions: \t");

        int n = sc.nextInt();

        System.out.println("Enter each expression in one line:  \t");

        sc.nextLine();

        String expn[] = new String[n];

        for (int i=0;i < n;i++)

        {

            expn[i]= sc.nextLine();

        }

        System.out.println("\n\n\tQuadruples:   ");

        System.out.println();

        qQuadruple(expn);

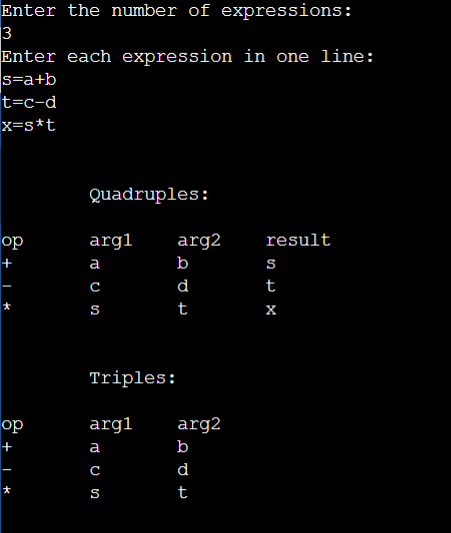
        System.out.println("\n\n\tTriples:   ");

        System.out.println();

        rTriple(expn);

    }

}



1. All three programs have to be performed (5 marks for each output)
2. Write a program to create your own ‘C’ library using macros for conversions.

(metre → feet, litre → cubic feet, °C → °F)

**Cvt.h**

#define CF(n1) ((n1 \* 1.8) + 32)

#define FC(n1) ((n1 - 32) \* 0.56)

#define MF(n1) n1 \* 3.281

#define FM(n1) n1 / 3.281

#define LCF(n1) n1 / 28.317

#define CFL(n1) n1 \* 28.317

**Cvt.c**

#include <stdio.h>

#include "Convert.h"

int main(){

        int n;

        float n1;

        float ctf,ftc,mf,fm,lcf,cfl;

        printf("Enter the choice: \n1. Celsius to Fahrenheit \n2. Fahrenheit to Celsius \n3. Metre to Feet \n4. Feet to Metre \n5. Litre to Cubic Feet \n6. Cubic Feet to Litre\n");

        scanf("%d",&n);

        switch(n)

        {

                case 1:

                        printf("Enter the temperature in Degree Celsius: ");

                        scanf("%f",&n1);

                        ctf = CF(n1);

                        printf("Temperature in Degree Fahrenheit: %f", ctf);

                        break;

                case 2:

                        printf("Enter the temperature in Degree Fahrenheit: ");

                        scanf("%f",&n1);

                        ftc = FC(n1);

                        printf("Temperature in Degree Celsius: %f", ftc);

                        break;

                case 3:

                        printf("Enter the Metre: ");

                        scanf("%f",&n1);

                        mf = MF(n1);

                        printf("Feet in metre: %f", mf);

                        break;

                case 4:

                        printf("Enter the Feet: ");

                        scanf("%f",&n1);

                        fm = FM(n1);

                        printf("Metre in feet: %f", fm);

                        break;

                case 5:

                        printf("Enter the volume in Litre: ");

                        scanf("%f",&n1);

                        lcf = LCF(n1);

                        printf("Cubic Feet in Litre: %f", lcf);

                        break;

                case 6:

                        printf("Enter the volume in Cubic feet: ");

                        scanf("%f",&n1);

                        cfl = CFL(n1);

                        printf("Litre in Cubic Feet: %f", cfl);

                          break;

                default:

                  printf("Wrong Choice");

        }

        return 0;

}

1. Write a Lex program to count the number of words in a source program.

%option noyywrap

%{

        #include<stdio.h>

        int i=0;

%}

%%

([a-zA-Z0-9])\* {i++;}

\n {printf("Total number of words in a source program is %d",i); i=0;}

%%

int main()

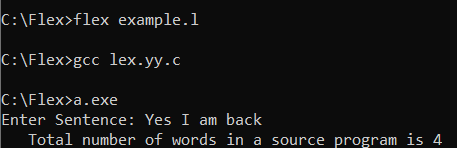
{

        printf("Enter Sentence: ");

        yylex();

        return 0;

}



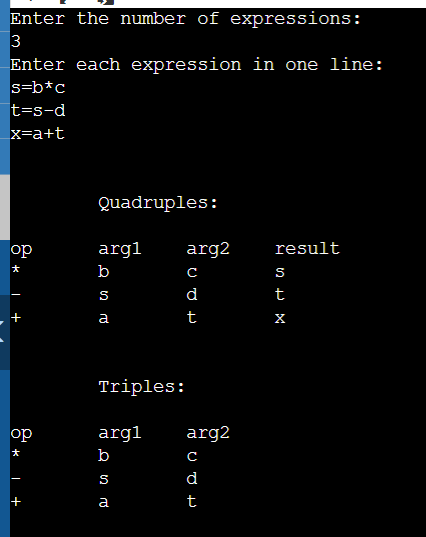
1. Write a program to convert the given computation into three address codes.

x = a+ b\*c -d;

s=b\*c

t=s-d

x=a+t



1. All three programs have to be performed (5 marks for each output)
2. Write a program to create your own ‘C’ library using macros for conversions.

(binary → decimal, binary → hexadecimal)

**bindeche.c**

#include<stdio.h>

#include<conio.h>

#include "bindeche.h"

int main()

{

   long int binary;

   int choice,check=0;

   printf("Enter the binary number: ");

   scanf("%ld", &binary);

   printf("\n0 = Binary to Decimal\n1 = Binary to Hexadecimal\n Enter your choice : ");

   scanf("%d", &choice);

   while(check!=1){

           if(choice ==0 || choice ==1){

                   check=1;

           }

           else{

                   printf("Wrong choice.\n 0 = Binary to Decimal\n1 = Binary to Hexadecimal\nEnter your choice again : ");

                   scanf("%d", &choice);

           }

   }

   if(choice == 0){

           bin\_dec(binary);

   }

   else{

           bin\_hex(binary);

   }

   return 0;

}

**bindeche.h**

int conversion(binary)

{

        long int binaryval, decimalval = 0, i = 1, remainder;

        binaryval = binary;

        while (binaryval != 0){

        remainder = binaryval % 10;

        decimalval = decimalval + remainder \* i;

        i = i \* 2;

        binaryval = binaryval / 10;

        }

        return decimalval;

}

void bin\_dec(binary)

{

        long int n;

        n=conversion(binary);

        printf("Equivalent decimal value: %ld", n);

}

void bin\_hex(binary)

{

        long int n;

        n=conversion(binary);

        printf("Equivalent hexadecimal value: %lX", n);

}

1. Write a Lex program to identify tokens.

%option noyywrap

%{

        #include<stdio.h>

%}

%%

"while"|"if"|"else"|"elseif"|"do"|"int"|"float" {printf("\n%s is Keywords\n",yytext);}

"<="|"="|"=>"|"+"|"-"|"/"|"\*" {printf("\n%s is Operators\n",yytext);}

[a-zA-Z]\* {printf("\n%s is Identifiers\n",yytext);}

[0-9]+ {printf("\n%s is Numbers\n",yytext);}

[{}()|, ;] {printf("\n%s is Separators\n",yytext);}

%%

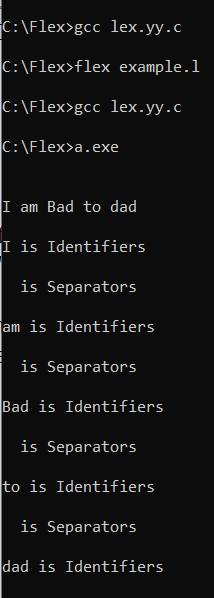
int main()

{

        yylex();

        return 0;

}



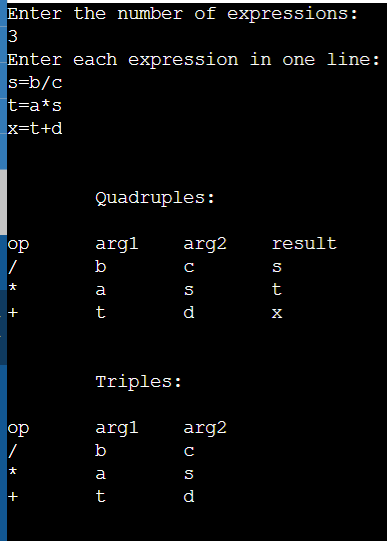
1. Write a program to convert the given computation into three address codes.

[ x = a\*b/c+d ]

s=b/c

t=a\*s

x=t+d



1. All three programs have to be performed (5 marks for each output)
2. Write a program to create your own ‘C’ library using macros to generate series. (Fibonacci, prime numbers, leap years)

**fib\_prime\_leap.h**

int fib(n)

{

        int i;

        int a=0;

        int b=1;

        int nt=a+b;

        printf("Fibonacci Series:%d\t%d",a,b);

        for(i=3;i<=n;++i)

        {

                printf("\t%d",nt);

                a=b;

                b=nt;

                nt=a+b;

        }

}

int prime(n)

{

        int a,i,flag=0;

        if(n == 0 || n == 1){

                flag = 1;

        }

        for(i=2;i<=n/2;++i)

        {

                if(n%i==0)

                {

                        flag = 1;

                        break;

                }

        }

        if(flag == 0)

        {

                printf("It is a prime number!!");

        }

        else

        {

                printf("It is not a prime number!!");

        }

}

int leap(n)

{

        if(n % 400 == 0)

        {

                printf("%d is a leap year.", n);

        }

    else if (n % 100 == 0)

        {

                printf("%d is not a leap year.", n);

        }

           else if (n % 4 == 0)

        {

                printf("%d is a leap year.", n);

        }

        else

        {

                printf("%d is not a leap year.", n);

        }

}

**fib\_prime\_leap.c**

#include <stdio.h>

#include "fib\_prime\_leap.h"

int main()

{

        int n;

        int n1;

        printf("Enter the choice: \n1. Fibonacci series \n2. Prime numbers\n3. Leap Years\n");

        scanf("%d",&n);

        switch(n)

        {

                case 1:

                        printf("\nEnter the total number of terms to get Fibonacci series: ");

                        scanf("%d",&n1);

                        fib(n1);

                        break;

                case 2:

                        printf("\nCheck whether the number is prime or not: ");

                        scanf("%d",&n1);

                        prime(n1);

                        break;

                case 3:

                        printf("\nEnter the year to check whether it is leap or not: ");

                        scanf("%d",&n1);

                        leap(n1);

                        break;

                default:

                        printf("Wrong Choice");

        }

        return 0;

}

1. Write a Lex program to count the number of tokens with uppercase characters.

%option noyywrap

%{

        #include<stdio.h>

        int UC=0;

%}

%%

[A-Z] {UC++;}

\n {printf("Total uppercase characters are %d", UC);}

%%

int main()

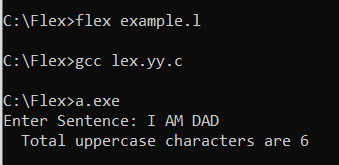
{

        printf("Enter Sentence: ");

        yylex();

        return 0;

}



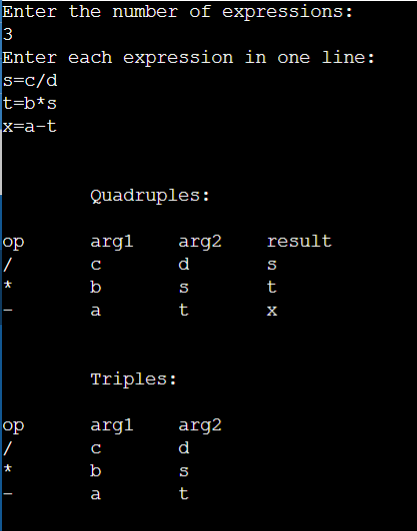
1. Write a program to convert the given computation into three address codes.

[ x = a-b\*c/d ]

s=c/d

t=b\*s

x=a-t



1. All three programs have to be performed (5 marks for each output)
2. Write a program to create your own ‘C’ library using macros to find the properties of a given number n – factorial of n, sum of natural numbers till n, and factors of n.

**properties.h**

int fact(x)

{

        int p=1, i=1;

        for(i=1; i<=x; i++)

        {

                p =p \* i;

        }

          return p;

}

#define sum\_n(n) n \* (n + 1) / 2

int print\_factors(n)

{

        int i;

    for(i=1; i<=n; i++){

        if(n%i == 0){

            printf("%d ", i);

        }

    }

}

**properties.c**

#include<stdio.h>

#include "properties.h"

int main()

{

        int n;

        int n1;

        printf("Enter the choice: \n1. Factorial  \n2. Sum of N natural numbers\n3. Factors of N\n");

        scanf("%d",&n);

        switch(n)

        {

                case 1:

                        printf("\nEnter the number to find the Factorial: ");

                        scanf("%d",&n1);

                        printf("Factorial of %d is %d",n1,fact(n1));

                        break;

                case 2:

                        printf("\nEnter the number to find the sum of N natural numbers: ");

                        scanf("%d",&n1);

                        printf("Sum of %d natural number is %d",n1,sum\_n(n1));

                        break;

                case 3:

                        printf("\nEnter the number to find the factors: ");

                        scanf("%d",&n1);

                        printf("Factors of %d are ",n1);

                        print\_factors(n1);

                        break;

                default:

                        printf("Wrong Choice");

        }

        return 0;

}

1. Write a Lex program to show the count of the number of lines in the source program.

%{

        #include<stdio.h>

        int nLines=1;

%}

%%

\n {

        nLines++;

        }

%%

int yywrap(void)

{

        return 1;

}

int main()

{

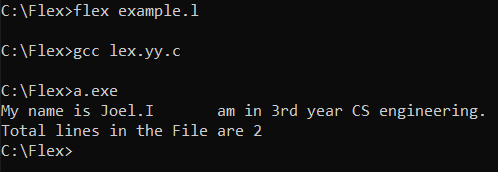
        yyin = fopen("abc.txt","r");

        yylex();

        printf("\nTotal lines in the File are %d",nLines);

        return 0;

}



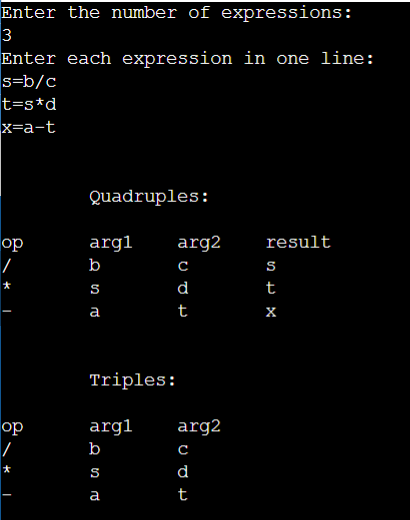
1. Write a program to convert the given computation into three address codes.

x = a-b/c\*d;

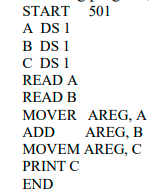
s=b/c

t=s\*d

x=a-t



1. Both programs have to be performed (10marks for part a, 5 marks for part b)
2. Consider the following program, **Display the Pass-1** of the Program  [10 M]



NOT DONE

1. Write a Lex program to count the number of words in the source program. (05 marks)

%{

        #include<stdio.h>

        int nwords;

%}

%%

[^ \n\t]+ {nwords++;}

%%

int yywrap(void)

{

        return 1;

}

int main()

{

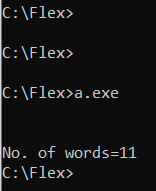
        yyin= fopen("abc.txt","r");

        yylex();

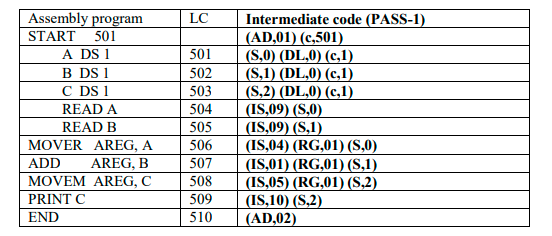
        printf("\nNo. of words=%d", nwords);

        return 0;

}



1. Both programs have to be performed (10marks for part a, 5 marks for part b)
2. For the given program, **Display the Pass-2**by taking intermediate code as an input Assembly program LC Intermediate code (PASS-1)                                       [10 M]



NOT DONE

1. Write a Lex program to count the number of tokens with uppercase characters.

%option noyywrap

%{

        #include<stdio.h>

        int UC=0;

%}

%%

[A-Z] {UC++;}

\n {printf("Total uppercase characters are %d", UC);}

%%

int main()

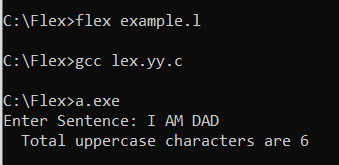
{

        printf("Enter Sentence: ");

        yylex();

        return 0;

}



1. All three programs have to be performed (5 marks for each output)
2. Write a program to create your own ‘C’ library using macros to find the properties of a given number n – factorial of n, sum of natural numbers till n, and factors of n.

**properties.h**

int fact(x)

{

        int p=1, i=1;

        for(i=1; i<=x; i++)

        {

                p =p \* i;

        }

          return p;

}

#define sum\_n(n) n \* (n + 1) / 2

int print\_factors(n)

{

        int i;

    for(i=1; i<=n; i++){

        if(n%i == 0){

            printf("%d ", i);

        }

    }

}

**properties.c**

#include<stdio.h>

#include "properties.h"

int main()

{

        int n;

        int n1;

        printf("Enter the choice: \n1. Factorial  \n2. Sum of N natural numbers\n3. Factors of N\n");

        scanf("%d",&n);

        switch(n)

        {

                case 1:

                        printf("\nEnter the number to find the Factorial: ");

                        scanf("%d",&n1);

                        printf("Factorial of %d is %d",n1,fact(n1));

                        break;

                case 2:

                        printf("\nEnter the number to find the sum of N natural numbers: ");

                        scanf("%d",&n1);

                        printf("Sum of %d natural number is %d",n1,sum\_n(n1));

                        break;

                case 3:

                        printf("\nEnter the number to find the factors: ");

                        scanf("%d",&n1);

                        printf("Factors of %d are ",n1);

                        print\_factors(n1);

                        break;

                default:

                        printf("Wrong Choice");

        }

        return 0;

}

1. Write a Lex program to identify if the expression is valid or not. (Eg. 2+3 is valid expression and 4+ is invalid expression)

%option noyywrap

%{

        #include<stdio.h>

        #include<string.h>

        int operands\_count=0,operators\_count=0,valid=1,top=-1,l=0,j=0;

        char operators[10][10],operands[10][10],stack[100];

%}

%%

"(" {

        top++;

        stack[top]= '(';

}

"[" {

        top++;

        stack[top]= '[';

}

"{" {

        top++;

        stack[top]= '{';

}

")" {

        if(stack[top]!='(')

        {

                valid=0;

        }

        else if(operands\_count>0 && (operands\_count-operators\_count)!=1)

        {

                valid=0;

        }

        else

        {

                top--;

                operands\_count=1;

                operators\_count=0;

        }

}

"]" {

        if(stack[top]!='[')

        {

                valid=0;

        }

        else if(operands\_count>0 && (operands\_count-operators\_count)!=1)

        {

                valid=0;

        }

        else

        {

                top--;

                operands\_count=1;

                operators\_count=0;

        }

}

"}" {

        if(stack[top]!='{')

        {

                valid=0;

        }

        else if(operands\_count>0 && (operands\_count-operators\_count)!=1)

        {

                valid=0;

        }

        else

        {

                top--;

                operands\_count=1;

                operators\_count=0;

        }

}

"+"|"-"|"/"|"\*" {

        operators\_count++;

        strcpy(operators[l], yytext);

        l++;

}

[0-9]+|[a-zA-Z][a-zA-Z0-9\_]\* {

        operands\_count++;

        strcpy(operands[j], yytext);

        j++;

}

%%

int main()

{

        printf("Enter the arithmetic expression: ");

        yylex();

        if(valid==1 && top==-1)

        {

                printf("\nValid Expression\n");

        }

        else

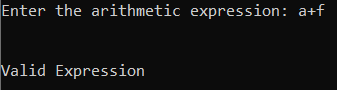
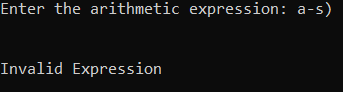
        {

                printf("\nInvalid Expression\n");

        }

        return 0;

}

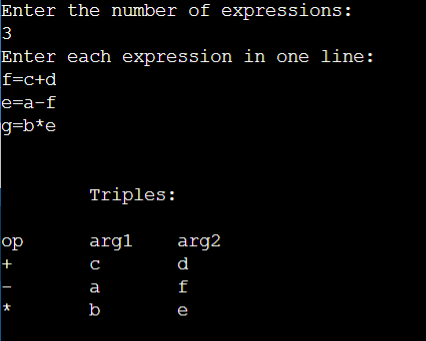


1. Consider the following **Three address code** *and***display Triples**

f=c+d

e=a-f

g=b\*e



1. Both programs have to be performed (10marks for part a, 5 marks for part b)
2. Write a program to remove left recursion from a given context free grammar. Nonterminal ={S,L} terminal={( , x, , ,) } S(L)/x LL,S/S

NOT DONE

1. Write a lex program to identify all the keywords (if, else, while etc.)

%option noyywrap

%{

        #include<stdio.h>

        int n=0;

%}

%%

["while"|"if"|"else"|"elseif"]\*  {printf("Valid Keyword");}

^[^("while")|^("if")|^("else")|^("elseif")]\*  {printf("InValid Keyword");break;}

%%

int main()

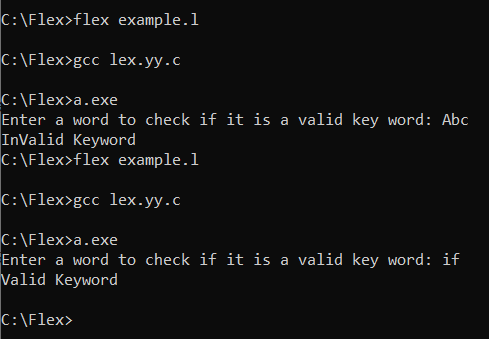
{

        printf("Enter a word to check if it is a valid keyword: ");

        yylex();

        return 0;

}



1. Both programs have to be performed (10marks for part a, 5 marks for part b)
2. Write a program to optimize the given three address codes.                         [10 M]

T1= 5\*3+10         // Constant folding

T3=T1                 // variable propagation

T2=T1+T3

T5=4\*T2         // common subexpression elimination

T6=4\*T2+100

NOT DONE

1. Write a program to count the number of characters, words, sentences present in input using LEX.

%{

        #include<stdio.h>

        int nlines=1,nwords,nchars;

%}

%%

\n {

        nchars++;nlines++;

}

[^ \n\t]+ {nwords++, nchars=nchars+yyleng;}

. {nchars++;}

%%

int yywrap(void)

{

        return 1;

}

int main()

{

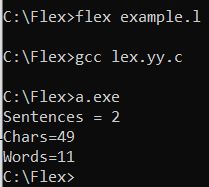
        yyin=fopen("abc.txt","r");

        yylex();

        printf("Sentences = %d\nChars=%d\nWords=%d",nlines,nchars,nwords);

        return 0;

}

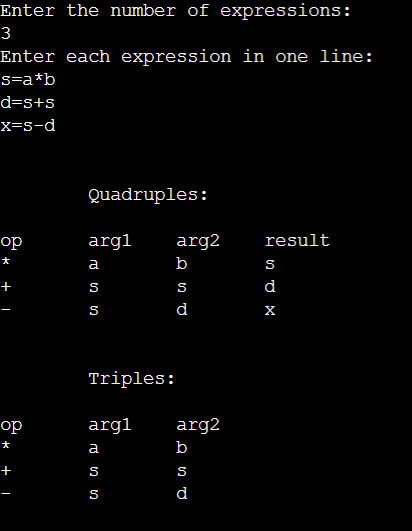


1. Both programs have to be performed (10marks for part a, 5 marks for part b)
2. Write a program to generate the three address code of x = a\*b – a\*b + a\*b;      [10 M]

s=a\*b

d=s+s

x=s-d



1. Write a program to count the number of lines, numbers and blank spaces present in input using LEX.

        %{

        #include<stdio.h>

        int nlines=1,nnos,sc=0;

%}

%%

\n {

        nlines++;

}

([ ])+ sc++;

[0-9] {nnos++;}

%%

int yywrap(void)

{

        return 1;

}

int main()

{

        yyin= fopen("abc.txt","r");

        yylex();

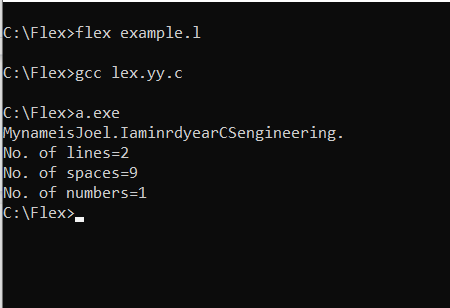
        printf("\nNo. of lines=%d", nlines);

        printf("\nNo. of spaces=%d", sc);

        printf("\nNo. of numbers=%d", nnos);

        return 0;

}



1. Both programs have to be performed (10marks for part a, 5 marks for part b)
2. Find the First() and Follow() sets of each non-terminal.         [10 M]

S -> Xb | XS

X -> a | b

NOT DONE

1. Write a program to generate the three address code of

pi = 3.145;

x = a \* pi \* 180 + b \* pi \* 2;

pi=3.145                         //here pi is refer to as p f=180 l=2

c=p\*f

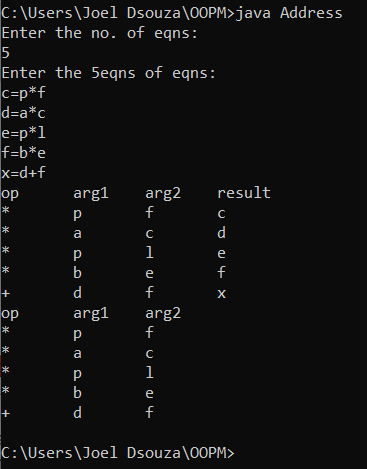
d=a\*c

e=pi\*l

f=b\*e

x=d+f

HERE f=180 and e=2



1. Both programs have to be performed (10marks for part a, 5 marks for part b)
2. Find the First() and Follow() sets of each non-terminal.                       [10 M]

S -> S | a | ε

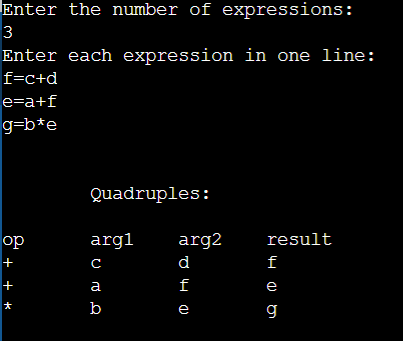
NOT DONE

1. Consider the following Three address code and display Quadruples

f=c+d

e=a-f

g=b\*e



**FIRST AND FOLLOW!!**

// C program to calculate the First and

// Follow sets of a given grammar

#include<stdio.h>

#include<ctype.h>

#include<string.h>

// Functions to calculate Follow

void followfirst(char, int, int);

void follow(char c);

// Function to calculate First

void findfirst(char, int, int);

int count, n = 0;

// Stores the final result

// of the First Sets

char calc\_first[10][100];

// Stores the final result

// of the Follow Sets

char calc\_follow[10][100];

int m = 0;

// Stores the production rules

char production[10][10];

char f[10], first[10];

int k;

char ck;

int e;

int main(int argc, char \*\*argv)

{

        int jm = 0;

        int km = 0;

        int i, choice;

        char c, ch;

        count = 8;

        // The Input grammar

        strcpy(production[0], "E=TR");

        strcpy(production[1], "R=+TR");

        strcpy(production[2], "R=#");

        strcpy(production[3], "T=FY");

        strcpy(production[4], "Y=\*FY");

        strcpy(production[5], "Y=#");

        strcpy(production[6], "F=(E)");

        strcpy(production[7], "F=i");

        int kay;

        char done[count];

        int ptr = -1;

        // Initializing the calc\_first array

        for(k = 0; k < count; k++) {

                for(kay = 0; kay < 100; kay++) {

                        calc\_first[k][kay] = '!';

                }

        }

        int point1 = 0, point2, xxx;

        for(k = 0; k < count; k++)

        {

                c = production[k][0];

                point2 = 0;

                xxx = 0;

                // Checking if First of c has

                // already been calculated

                for(kay = 0; kay <= ptr; kay++)

                        if(c == done[kay])

                                xxx = 1;

                if (xxx == 1)

                        continue;

                // Function call

                findfirst(c, 0, 0);

                ptr += 1;

                // Adding c to the calculated list

                done[ptr] = c;

                printf("\n First(%c) = { ", c);

                calc\_first[point1][point2++] = c;

                // Printing the First Sets of the grammar

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

                        int lark = 0, chk = 0;

                        for(lark = 0; lark < point2; lark++) {

                                if (first[i] == calc\_first[point1][lark])

                                {

                                        chk = 1;

                                        break;

                                }

                        }

                        if(chk == 0)

                        {

                                printf("%c, ", first[i]);

                                calc\_first[point1][point2++] = first[i];

                        }

                }

                printf("}\n");

                jm = n;

                point1++;

        }

        printf("\n");

        printf("-----------------------------------------------\n\n");

        char donee[count];

        ptr = -1;

        // Initializing the calc\_follow array

        for(k = 0; k < count; k++) {

                for(kay = 0; kay < 100; kay++) {

                        calc\_follow[k][kay] = '!';

                }

        }

        point1 = 0;

        int land = 0;

        for(e = 0; e < count; e++)

        {

                ck = production[e][0];

                point2 = 0;

                xxx = 0;

                // Checking if Follow of ck

                // has already been calculated

                for(kay = 0; kay <= ptr; kay++)

                        if(ck == donee[kay])

                                xxx = 1;

                if (xxx == 1)

                        continue;

                land += 1;

                // Function call

                follow(ck);

                ptr += 1;

                // Adding ck to the calculated list

                donee[ptr] = ck;

                printf(" Follow(%c) = { ", ck);

                calc\_follow[point1][point2++] = ck;

                // Printing the Follow Sets of the grammar

                for(i = 0 + km; i < m; i++) {

                        int lark = 0, chk = 0;

                        for(lark = 0; lark < point2; lark++)

                        {

                                if (f[i] == calc\_follow[point1][lark])

                                {

                                        chk = 1;

                                        break;

                                }

                        }

                        if(chk == 0)

                        {

                                printf("%c, ", f[i]);

                                calc\_follow[point1][point2++] = f[i];

                        }

                }

                printf(" }\n\n");

                km = m;

                point1++;

        }

}

void follow(char c)

{

        int i, j;

        // Adding "$" to the follow

        // set of the start symbol

        if(production[0][0] == c) {

                f[m++] = '$';

        }

        for(i = 0; i < 10; i++)

        {

                for(j = 2;j < 10; j++)

                {

                        if(production[i][j] == c)

                        {

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

                                {

                                        // Calculate the first of the next

                                        // Non-Terminal in the production

                                        followfirst(production[i][j+1], i, (j+2));

                                }

                                if(production[i][j+1]=='\0' && c!=production[i][0])

                                {

                                        // Calculate the follow of the Non-Terminal

                                        // in the L.H.S. of the production

                                        follow(production[i][0]);

                                }

                        }

                }

        }

}

void findfirst(char c, int q1, int q2)

{

        int j;

        // The case where we

        // encounter a Terminal

        if(!(isupper(c))) {

                first[n++] = c;

        }

        for(j = 0; j < count; j++)

        {

                if(production[j][0] == c)

                {

                        if(production[j][2] == '#')

                        {

                                if(production[q1][q2] == '\0')

                                        first[n++] = '#';

                                else if(production[q1][q2] != '\0'

                                                && (q1 != 0 || q2 != 0))

                                {

                                        // Recursion to calculate First of New

                                        // Non-Terminal we encounter after epsilon

                                        findfirst(production[q1][q2], q1, (q2+1));

                                }

                                else

                                        first[n++] = '#';

                        }

                        else if(!isupper(production[j][2]))

                        {

                                first[n++] = production[j][2];

                        }

                        else

                        {

                                // Recursion to calculate First of

                                // New Non-Terminal we encounter

                                // at the beginning

                                findfirst(production[j][2], j, 3);

                        }

                }

        }

}

void followfirst(char c, int c1, int c2)

{

        int k;

        if(!(isupper(c)))

                f[m++] = c;

        else

        {

                int i = 0, j = 1;

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

                {

                        if(calc\_first[i][0] == c)

                                break;

                }

                //Including the First set of the

                // Non-Terminal in the Follow of

                // the original query

                while(calc\_first[i][j] != '!')

                {

                        if(calc\_first[i][j] != '#')

                        {

                                f[m++] = calc\_first[i][j];

                        }

                        else

                        {

                                if(production[c1][c2] == '\0')

                                {

                                        // Case where we reach the

                                        // end of a production

                                        follow(production[c1][0]);

                                }

                                else

                                {

                                        // Recursion to the next symbol

                                        // in case we encounter a "#"

                                        followfirst(production[c1][c2], c1, c2+1);

                                }

                        }

                        j++;

                }

        }

}

**CODE OPTIMIZATION**

import java.util.\*;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class CodeOptimize

{

HashMap < String, String > statements = new HashMap <> ();

List < String > result = new ArrayList < String > (Arrays.asList ("a", "b", "c", "d", "e"));

List < String > operators = new ArrayList < String > (Arrays.asList ("+", "\*"));

public static void main (String[]args)

{

CodeOptimize obj = new CodeOptimize();

obj.getStatements ();

System.out.println ("Initaially statements are: ");

obj.putStatements ();

obj.constantFolding ();

System.out.println ("After constant folding: ");

obj.putStatements ();

obj.variablePropagation ();

System.out.println ("After variable propagation: ");

obj.putStatements ();

obj.commonSubexpElim ();

System.out.println ("After Common Sub-expression Elimination: ");

obj.putStatements ();

}

public void getStatements ()

  {

this.statements.put ("a", "5\*3+10");

this.statements.put ("b", "a");

this.statements.put ("c", "a+b");

this.statements.put ("d", "4\*c");

this.statements.put ("e", "4\*c+100");

}

public void putStatements ()

  {

for (Map.Entry mapElement:this.statements.entrySet ())

      {

String key = (String) mapElement.getKey ();

String value = (String) mapElement.getValue ();

System.out.println (key + " : " + value);

}

System.out.println ("----------------------------------------");

}

public int evaluate (String str)

{

    String[]arr = str.split ("\\+");

    for (int i = 0; i < arr.length; i++)

    {

        int result = 1;

        if (arr[i].contains ("\*"))

            {

            String[]num = arr[i].split ("\\\*");

            for (int j = 0; j < num.length; j++)

                {

                result \*= Integer.parseInt (num[j]);

                }

            arr[i] = String.valueOf (result);

        }

    }

    int len = arr.length;

    int sum = 0;

    for(int i = 0; i < len; i++)

    {

        sum += Integer.parseInt (arr[i]);

    }

    return sum;

}

public void constantFolding ()

{

    for (int i = 0; i < this.result.size (); i++)

    {

            String lhs = this.result.get (i);

            String rhs = this.statements.get (lhs);

        Pattern p = Pattern.compile ("[\\d]+([+\*][\\d]+)+");

        Matcher m = p.matcher (rhs);

        while (m.find ())

            {

            String subexpr = m.group ();

                       int result = this.evaluate (subexpr);

                String res = String.valueOf (result);

                rhs = rhs.replace (rhs.substring (m.start (), m.end ()), res);

                m = p.matcher (rhs);

        }

        this.statements.put (lhs, rhs);

    }

}

public void variablePropagation ()

{

    for (int i = 0; i < this.result.size (); i++)

    {

            String lhs1 = this.result.get (i);

        String rhs1 = this.statements.get (lhs1);

            if (rhs1.length () == 1 && this.result.contains (rhs1))

            {

              for (int j = 0; j < this.result.size (); j++)

              {

            String lhs2 = this.result.get (j);

            String rhs2 = this.statements.get (lhs2);

                    if (rhs2.contains (lhs1))

                    {

                rhs2 = rhs2.replace (lhs1, rhs1);

                        this.statements.put (lhs2, rhs2);

            }

              }

              this.result.remove (lhs1);

              this.statements.remove (lhs1, rhs1);

           }

    }

}

public void commonSubexpElim ()

{

    for (int i = 0; i < this.result.size (); i++)

    {

        String lhs1 = this.result.get (i);

            String rhs1 = this.statements.get (lhs1);

            for (int j = 0; j < this.result.size (); j++)

            {

                String lhs2 = this.result.get (j);

            String rhs2 = this.statements.get (lhs2);

            if (lhs1 == lhs2)

                {

                        continue;

            }

                if (rhs1.contains (rhs2))

                {

                        int start = rhs1.indexOf (rhs2);

                        int len = rhs2.length ();

                        rhs1 = rhs1.replace(rhs1.substring(start,len),lhs2);

                this.statements.put(lhs1,rhs1);

            }

        }

    }

}

}

**PASS1 TO PASS2 -**

#include <bits/stdc++.h>

#define print(n) cout<<n<<endl

#define pb push\_back

#define f first

#define s second

using namespace std;

map<string,int> sym;

int symCounter=0;

int startAddress=-1;

int ind=-1;

map<int,vector<string>> passTwo;

map<string,pair<string,string>> MOT;

bool onlyNum(string s){

    for (auto it: s){

        if (it>='0' && it<='9'){

            continue;

        }

        return false;

    }

    return true;

}

string eval(vector<string> &vs){

    string ans="";

    int n=vs.size();

    bool pass2Flag=false;

    for (int i=0;i<n;i++){

        //print("ITR "<<vs[i]);

        ans+="[ ";

        if (onlyNum(vs[i])){

            ans+="c ";

            ans+=(vs[i]);

        }else if (vs[i]=="START"){

            ans+=MOT[vs[i]].f;

            ans+=" ";

            ans+=MOT[vs[i]].s;

            startAddress=stoi(vs[i+1]);

        }else if (MOT.find(vs[i])!=MOT.end()){

            ans+=MOT[vs[i]].f;

            ans+=" ";

            ans+= MOT[vs[i]].s;

            if (MOT[vs[i]].f=="IS"){

                pass2Flag=true;

                passTwo[ind].pb(MOT[vs[i]].s);

                if (vs.size()==2){

                    passTwo[ind].pb("00");

                }

                ans+=" ] ";

                continue;

            }

            if (pass2Flag){

                passTwo[ind].pb(MOT[vs[i]].s);

            }

        }else{

            ans+="S";

            ans+=" ";

            if (sym.find(vs[i])!=sym.end()){

                ans+=to\_string(sym[vs[i]]);

            }else{

                sym[vs[i]]=symCounter;

                ans+=to\_string(sym[vs[i]]);

                symCounter++;

            }

            if (pass2Flag){

                passTwo[ind].pb(to\_string(startAddress + sym[vs[i]]));

            }

        }

        ans+=" ] ";

        //print(i<<" itr "<<ans);

    }

    //print("RETURN: "<<ans);

    // if (strstr(ans.c\_str(),"IS")){

    // }

    return ans;

}

int main()

{

    MOT["STOP"]={"IS","00"};

    MOT["ADD"]={"IS","01"};

    MOT["SUB"]={"IS","02"};

    MOT["MUL"]={"IS","03"};

    MOT["MOVER"]={"IS","04"};

    MOT["MOVEM"]={"IS","05"};

    MOT["COMP"]={"IS","06"};

    MOT["BC"]={"IS","07"};

    MOT["DIV"]={"IS","08"};

    MOT["READ"]={"IS","09"};

    MOT["PRINT"]={"IS","10"};

    MOT["START"]={"AD","01"};

    MOT["END"]={"AD","02"};

    MOT["ORIGIN"]={"AD","03"};

    MOT["EQU"]={"AD","04"};

    MOT["LTORG"]={"AD","05"};

    MOT["DS"]={"DL","01"};

    MOT["DC"]={"DL","02"};

    MOT["AREG"]={"RG","01"};

    MOT["BREG"]={"RG","02"};

    MOT["CREG"]={"RG","03"};

    MOT["LT"]={"CC","01"};

    MOT["LE"]={"CC","02"};

    MOT["EQ"]={"CC","03"};

    MOT["GT"]={"CC","04"};

    MOT["GE"]={"CC","05"};

    MOT["ANY"]={"CC","06"};

    print("Enter Number of Lines: ");

    int n;

    cin>>n;

    print("Enter "<<n<<" Lines of Code: ");

    vector<vector<string>> v;

    for (int i=0;i<n;i++){

        string s;

        if(i==0) getline(cin,s);

        getline(cin,s);

        //print(s);

        string tmp="";

        vector<string> tv;

        for (auto it: s){

            if (it==' '){

                tv.pb(tmp);

                tmp="";

                continue;

            }

            tmp+=it;

        }

        tv.pb(tmp);

        v.pb(tv);

    }

    print("PASS-1 Table:");

    print("LC \t Intermediate Code");

    // for (auto it: v){

    //     for (auto x: it){

    //         cout<<x<<" ";

    //     }

    //     cout<<endl;

    // }

    for (auto it: v){

        string pass1=eval(it);

        if(ind>=0) cout<<(startAddress+ind);

        cout<<(" \t ");

        print(pass1);

        ind++;

    }

    print("PASS-2 Table:");

    print("LC \t Machine Code");

    for (int j=-1;j<ind;j++){

        if(j>=0) cout<<(startAddress+j);

        cout<<" \t ";

        if (passTwo.find(j)!=passTwo.end()){

            for (auto x: passTwo[j]){

                cout<<x<<" ";

            }

            //cout<<" \t "<<j;

        }else{

            cout<<(" \_\_\_ ");

        }

        cout<<endl;

    }

    print("Symbol Table: ");

    for (auto it: sym){

        if(it.f.size()==1){

            cout<<it.f<<" "<<(501+it.s)<<endl;

        }

    }

    return 0;

}