COMPILER DESIGN LAB PROGRAMS

➔ **1. Write a Program in C++ to work as a calculator.**

```
#include <iostream>
using namespace std;

int main() {
    char op;
    double num1, num2;

    cout << "Enter operator (+, -, *, /): ";
    cin >> op;

    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    switch(op) {
        case '+':
            cout << num1 << " + " << num2 << " = " << num1 + num2;
            break;
        case '-':
            cout << num1 << " - " << num2 << " = " << num1 - num2;
            break;
        case '*':
            cout << num1 << " * " << num2 << " = " << num1 * num2;
            break;
        case '/':
            if (num2 != 0)
                cout << num1 << " / " << num2 << " = " << num1 / num2;
            else
                cout << "Error! Division by zero!";
            break;
        default:
            cout << "Error! Invalid operator!";
            break;
    }

    return 0;
}
```

➔ 2. FLEX PROGRAMS

● CAPITAL LETTERS

```
%{
 int count = 0;
 %}
 %%
 [A-Z] {printf("%s is a capital letter",yytext);
 count++;
 }
 . {printf("%s is not a capital letter",yytext);}
 \n {return 0;}
 %%
 int yywrap(){}
 int main()
 {
 yylex();
 printf("%d letters are capital", count);
 return 0;
 }
```

● Count the number of characters and number of lines in the input

```
%{

int no_of_lines = 0;

int no_of_chars = 0;

%}

%%

\n ++no_of_lines;

. ++no_of_chars;

end return 0;

%%

int yywrap(){}
int main(int argc, char **argv)
{
yylex();
printf(&quot;number of lines = %d, number of chars = %d\n&quot;,
no_of_lines, no_of_chars );
return 0;
}
```

- **FIND WORDS AND NUMBERS**

```
%{
#include <stdio.h>
%}
letter [a-zA-Z]
%%
[0-9]+ { printf("NUM:  %s\n", yytext); }
{letter}+ { printf("WORD: %s\n", yytext); }
.|\n     ;
%%
int main() {
    yylex();
}
int yywrap() {
    return 1;
}
```

➔ 3. Implementation of scanner by specifying Regular Expressions.

```
%{
 #include <stdio.h>
 int countCapital = 0;
 int countDigits = 0;
 %}

 %%
 [A-Z] {
     printf("%s is a capital letter\n", yytext);
     countCapital++;
 }

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

 . {
     printf("%s is not a capital letter or digit\n", yytext);
 }

 \n {
     printf("Total capital letters: %d\n", countCapital);
     printf("Total digits: %d\n", countDigits);
     return 0;
 }
 %%

 int yywrap() {}

 int main(int argc, char *argv[]) {

     FILE *inputFile = fopen("input.txt", "r");


     yyin = inputFile; // Set the file pointer for flex to read from

     yylex();

     fclose(inputFile);
     return 0;
 }
```

➔ 4. BISON

- **To implement a scanner for calculator, we can write the file “cal1.l”**

```
%{
int lineNum = 0;
%}
%%
"(" { printf("(\n"); }
")" { printf(")\n"); }
"+" { printf("+\n"); }
"*" { printf("*\n"); }
\n { lineNum++; }
[ \t]+ { }
[0-9]+ { printf("%s\n", yytext); }
%%
int yywrap() {
return 1;
}
int main () {
yylex();
return 0;
}
```

- **To implement a parser for calculator, we can write the file “cal.y”**

```
%{
#include <stdio.h>
#include <ctype.h>
int lineNum = 1;
void yyerror(char *ps, ...) { /* need this to avoid
link problem */
printf("%s\n", ps);
}
%}
%union {
int d;
}
// need to choose token type from union above
%token <d> NUMBER
%token '(' ')'
%left '+'
%left '*'
%type <d> exp factor term
%start cal
%%
```

```
cal
: exp
{ printf("The result is %d\n", $1); }
;
exp
: exp '+' factor
{ $$ = $1 + $3; }
| factor
{ $$ = $1; }
;
factor
#ifdef DEBUG
printf("token '+' at line %d\n", lineNum);
#endif
return '+';
}
"*" {
#ifdef DEBUG
printf("token '*' at line %d\n", lineNum);
#endif
return '*';
}
[0-9]+ {
#ifdef DEBUG
printf("token %s at line %d\n", yytext, lineNum);
#endif
yylval.d = atoi(yytext);
return NUMBER;
}
%%
int yywrap() { /* need this to avoid link problem */
return 1;
}
```

➔ 5. A) TOP DOWN PARSER

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

#define SIZE 20

int main()
{
   char pro[SIZE], alpha[SIZE], beta[SIZE];
   int nont_terminal,i,j, index=3;

   printf("Enter the Production as E->E|A: ");
   scanf("%s", pro);

   nont_terminal=pro[0];
   if(nont_terminal==pro[index]) //Checking if the Grammar is LEFT RECURSIVE
   {
      //Getting Alpha
      for(i=++index,j=0;pro[i]!='|';i++,j++){
         alpha[j]=pro[i];
         //Checking if there is NO Vertical Bar (|)
         if(pro[i+1]==0){
            printf("This Grammar CAN'T BE REDUCED.\n");
            exit(0); //Exit the Program
         }
      }
      alpha[j]='\0'; //String Ending NULL Character

      if(pro[++i]!=0) //Checking if there is Character after Vertical Bar (|)
      {
         //Getting Beta
         for(j=i,i=0;pro[j]!='\0';i++,j++){
            beta[i]=pro[j];
         }
         beta[i]='\0'; //String Ending NULL character

         //Showing Output without LEFT RECURSION
         printf("\nGrammar Without Left Recursion: \n\n");
         printf(" %c->%s%c'\n", nont_terminal,beta,nont_terminal);
         printf(" %c'->%s%c'|#\n", nont_terminal,alpha,nont_terminal);
      }
      else
         printf("This Grammar CAN'T be REDUCED.\n");
   }
   else
      printf("\n This Grammar is not LEFT RECURSIVE.\n"); }
```

5. B) FIRST & FOLLOW

```
#include<iostream>
    #include<string.h>
    #define max 20

    using namespace std;

    char prod[max][10];
    char ter[10],nt[10];
    char first[10][10],follow[10][10];
    int eps[10];
    int count_var=0;

    int findpos(char ch) {
        int n;
        for(n=0;nt[n]!='\0';n++)
            if(nt[n]==ch) break;
            if(nt[n]=='\0') return 1;
            return n;
    }

    int IsCap(char c) {
        if(c >= 'A' && c<= 'Z')
            return 1;
        return 0;
    }

    void add(char *arr,char c) {
        int i,flag=0;
        for(i=0;arr[i]!='\0';i++) {
            if(arr[i] == c) {
                flag=1;
                break;
            }
        }
        if(flag!=1) arr[strlen(arr)] = c;
    }

    void addarr(char *s1,char *s2) {
        int i,j,flag=99;
        for(i=0;s2[i]!='\0';i++) {
            flag=0;
            for(j=0;;j++) {
                if(s2[i]==s1[j]) {
                    flag=1;
                    break;
                }
                if(j==strlen(s1) && flag!=1) {
                    s1[strlen(s1)] = s2[i];
                    break;
                }
            }
        }
    }

    void addprod(char *s) {
        int i;
        prod[count_var][0] = s[0];
        for(i=3;s[i]!='\0';i++) {
            if(!IsCap(s[i])) add(ter,s[i]);
            prod[count_var][i-2] = s[i];
        }
        prod[count_var][i-2] = '\0';
        add(nt,s[0]);
        count_var++;
    }

    void findfirst() {
        int i,j,n,k,e,n1;
        for(i=0;i<count_var;i++) {
            for(j=0;j<count_var;j++) {
                n = findpos(prod[j][0]);
                if(prod[j][1] == (char)238) eps[n] = 1;
                else {
                    for(k=1,e=1;prod[j][k]!='\0' && e==1;k++) {
                        if(!IsCap(prod[j][k])) {
                            e=0;
                            add(first[n],prod[j][k]);
                        }
                        else {
                            n1 = findpos(prod[j][k]);
                            addarr(first[n],first[n1]);
                            if(eps[n1]==0)
                                e=0;
                        }
                    }
                    if(e==1) eps[n]=1;
                }
            }
        }
    }

    void findfollow() {
        int i,j,k,n,e,n1;
        n = findpos(prod[0][0]);
        add(follow[n],'#');
        for(i=0;i<count_var;i++) {
            for(j=0;j<count_var;j++) {
                k = strlen(prod[j])-1;
                for(;k>0;k--) {
                    if(IsCap(prod[j][k])) {
                        n=findpos(prod[j][k]);
                        if(prod[j][k+1] == '\0')
                        {
                            n1 = findpos(prod[j][0]);
                            addarr(follow[n],follow[n1]);
                        }
                        if(IsCap(prod[j][k+1]))
                        {
                            n1 = findpos(prod[j][k+1]);
                            addarr(follow[n],first[n1]);
                            if(eps[n1]==1)
                            {
                                n1=findpos(prod[j][0]);
                                addarr(follow[n],follow[n1]);
                            }
                        }
                        else if(prod[j][k+1] != '\0')
                            add(follow[n],prod[j][k+1]);
                    }
                }
            }
        }
    }

    int main() {
        char s[max],i;
        cout<<"Enter the productions\n";
        cin>>s;
        while(strcmp("end",s)) {
            addprod(s);
            cin>>s;
        }
        findfirst();
        findfollow();
        for(i=0;i<strlen(nt);i++) {
            cout<<nt[i]<<"\t";
            cout<<first[i];
            if(eps[i]==1) cout<<((char)238)<<"\t";
            else cout<<"\t";
            cout<<follow[i]<<"\n";
        }
        return 0;;
    }
```

- **5. C) Predictive Parsing Table**

```
#include<iostream>
#include<vector>
#include<map>
#include<set>
#include<string>
using namespace std;

map<char,vector<string>>grammar;
map<char,set<char>>first,follow;
map<pair<char,char>,string>parsing_table;

set<char>computeFirst(char nt){
    set<char>res;
    if(!isupper(nt)){
        res.insert(nt);
        return res;
    }
    for(string p:grammar[nt]){
        char fc=p[0];
        if(fc==nt)continue;
        set<char>temp=computeFirst(fc);
        res.insert(temp.begin(),temp.end());
    }
    return res;
}

set<char>computeFollow(char nt){
    set<char>res;
    if(nt=='S')res.insert('$');
    for(auto e:grammar){
        char k=e.first;
        for(string p:e.second){
            size_t pos=p.find(nt);
            while(pos!=string::npos){
                if(pos+1<p.size()){
                    set<char>temp=computeFirst(p[pos+1]);
                    res.insert(temp.begin(),temp.end());
                    if(temp.find('ε')==temp.end())break;
                }
                if(pos+1==p.size()){
                    set<char>temp=computeFollow(k);
                    res.insert(temp.begin(),temp.end());
                    break;
                }
                pos=p.find(nt,pos+1);
            }
        }
    }
    return res;
}

void constructParsingTable(){
    for(auto e:grammar){
        char nt=e.first;
        for(string p:e.second){
            set<char>fs=computeFirst(p[0]);
            for(char t:fs){
                if(t!='ε')parsing_table[{nt,t}]=p;
            }
            if(fs.find('ε')!=fs.end()){
                set<char>fol=computeFollow(nt);
                for(char t:fol)parsing_table[{nt,t}]=p;
            }
        }
    }
}

int main(){
    int n;
    cout<<"Enter the number of productions: ";
    cin>>n;
    cout<<"Enter the productions in the form A->α (use 'ε' for epsilon):"<<endl;
    for(int i=0;i<n;++i){
        string prod;
        cin>>prod;
        char nt=prod[0];
        string rhs=prod.substr(3);
        grammar[nt].push_back(rhs);
    }
    for(auto&e:grammar){
        char k=e.first;
        first[k]=computeFirst(k);
        follow[k]=computeFollow(k);
    }
    constructParsingTable();
    cout<<"\nParsing Table:"<<endl;
    for(auto e:parsing_table){
        cout<<"M["<<e.first.first<<", "<<e.first.second<<"] = "<<e.second<<endl;
    }
    return 0;
}
```

- **5. D) Left Recursion**

```
#include<stdio.h>

#include<string.h>

void main()  {

    char input[100],l[50],r[50],temp[10],tempprod[20],productions[25][50];

    int i=0,j=0,flag=0,consumed=0;

    printf("Enter the productions: ");

    scanf("%1s->%s",l,r);

    printf("%s",r);

    while(sscanf(r+consumed,"%[^|]s",temp) == 1 && consumed <= strlen(r))  {

        if(temp[0] == l[0])  {

            flag = 1;

            sprintf(productions[i++], "%s->%s%s'", l, temp+1, l);

        }

        else

            sprintf(productions[i++], "%s'->%s%s'", l, temp, l);

        consumed += strlen(temp)+1;

             }

    if(flag == 1)  {

        sprintf(productions[i++], "%s->ε", l);

        printf("The productions after eliminating Left Recursion are:\n");

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

            printf("%s\n",productions[j]);

             }

    else

        printf("The Given Grammar has no Left Recursion"); }
```

- **5. E) Left Factoring**

```
#include<stdio.h>

 #include<string.h>

 int main()

 {

       char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];

      int i,j=0,k=0,l=0,pos;

    printf("Enter Production : A->");

      gets(gram);

      for(i=0;gram[i]!='|';i++,j++)

           part1[j]=gram[i];

      part1[j]='\0';

       for(j=++i,i=0;gram[j]!='\0';j++,i++)

          part2[i]=gram[j];

       part2[i]='\0';

     for(i=0;i<strlen(part1)||i<strlen(part2);i++)

      {

           if(part1[i]==part2[i])

           {

                         modifiedGram[k]=part1[i];

            k++;

            pos=i+1;

          }

    }

       for(i=pos,j=0;part1[i]!='\0';i++,j++){

           newGram[j]=part1[i];
```

```
        }
         newGram[j++]='|';
        for(i=pos;part2[i]!='\0';i++,j++){
            newGram[j]=part2[i];
        }
            modifiedGram[k]='X';
            modifiedGram[++k]='\0';
            newGram[j]='\0';
         printf("\n A->%s",modifiedGram);
         printf("\n X->%s\n",newGram);
}
```

➔ 6. BOTTOM UP PARSER

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

using namespace std;

char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR";
int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
int noitem=0;

struct Grammar
{
char lhs;
char rhs[8];
}g[20],item[20],clos[20][10];

int isvariable(char variable)
{
for(int i=0;i<novar;i++)
if(g[i].lhs==variable)
return i+1;
return 0;
}
void findclosure(int z, char a)
{
int n=0,i=0,j=0,k=0,l=0;
for(i=0;i<arr[z];i++)
{
for(j=0;j<strlen(clos[z][i].rhs);j++)
{
if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a)
{
clos[noitem][n].lhs=clos[z][i].lhs;
strcpy(clos[noitem][n].rhs,clos[z][i].rhs);
char temp=clos[noitem][n].rhs[j];
clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];
clos[noitem][n].rhs[j+1]=temp;
n=n+1;
}
}
}
for(i=0;i<n;i++)
{
```

```
for(j=0;j<strlen(clos[noitem][i].rhs);j++)
{
if(clos[noitem][i].rhs[j]=='.' && isvariable(clos[noitem][i].rhs[j+1])>0)
{
for(k=0;k<novar;k++)
{
if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)
{
for(l=0;l<n;l++)
if(clos[noitem][l].lhs==clos[0][k].lhs && strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)
break;
if(l==n)
{
clos[noitem][n].lhs=clos[0][k].lhs;
strcpy(clos[noitem][n].rhs,clos[0][k].rhs);
n=n+1;
}
}
}
}
}
}
arr[noitem]=n;
int flag=0;
for(i=0;i<noitem;i++)
{
if(arr[i]==n)
{
for(j=0;j<arr[i];j++)
{
int c=0;
for(k=0;k<arr[i];k++)
if(clos[noitem][k].lhs==clos[i][k].lhs && strcmp(clos[noitem][k].rhs,clos[i][k].rhs)==0)
c=c+1;
if(c==arr[i])
{
flag=1;
goto exit;
}
}
}
}
exit:;
if(flag==0)
```

```
arr[noitem++]=n;
}

int main()
{
cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";
do
{
cin>>prod[i++];
}while(strcmp(prod[i-1],"0")!=0);
for(n=0;n<i-1;n++)
{
m=0;
j=novar;
g[novar++].lhs=prod[n][0];
for(k=3;k<strlen(prod[n]);k++)
{
if(prod[n][k] != '|')
g[j].rhs[m++]=prod[n][k];
if(prod[n][k]=='|')
{
g[j].rhs[m]='\0';
m=0;
j=novar;
g[novar++].lhs=prod[n][0];
}
}
}
for(i=0;i<26;i++)
if(!isvariable(listofvar[i]))
break;
g[0].lhs=listofvar[i];
char temp[2]={g[1].lhs,'\0'};
strcat(g[0].rhs,temp);
cout<<"\n\n augumented grammar \n";
for(i=0;i<novar;i++)
cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";

for(i=0;i<novar;i++)
{
clos[noitem][i].lhs=g[i].lhs;
strcpy(clos[noitem][i].rhs,g[i].rhs);
if(strcmp(clos[noitem][i].rhs,"ε")==0)
strcpy(clos[noitem][i].rhs,".");
else
{
for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)
clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];
clos[noitem][i].rhs[0]='.';
}
}
arr[noitem++]=novar;
for(int z=0;z<noitem;z++)
{
char list[10];
int l=0;
for(j=0;j<arr[z];j++)
{
for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
{
if(clos[z][j].rhs[k]=='.')
{
for(m=0;m<l;m++)
if(list[m]==clos[z][j].rhs[k+1])
break;
if(m==l)
list[l++]=clos[z][j].rhs[k+1];
}
}
}
for(int x=0;x<l;x++)
findclosure(z,list[x]);
}
cout<<"\n THE SET OF ITEMS ARE \n\n";
for(int z=0; z<noitem; z++)
{
cout<<"\n I"<<z<<"\n\n";
for(j=0;j<arr[z];j++)
cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";

}

}
```

➔ 7. JAVA PROGRAMS

- **Write a Java Program to prin the message.**

```
import java.io.*;
public class Jpgm1
{
public static void main(String[] args)
{
System.out.println(“Hellow World”);
}
}
```

- **Write a Java Program to get the value from keyboard and print.**

```
import java.io.*;
import java.util.*;
public class Jpgm2
{
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
String s1 = sc.nextLine();
System.out.println(s1);
}
}
```

- **Write a Java Program to add two integer numbers.**

```
import java.io.*;
import java.util.*;
public class Jpgm3
{
public static void main(String[] args)
{
//Declare the necessary variables
int a,b,c;
//Create Scanner Object to get input from keyboard
Scanner sc = new Scanner(System.in);
//Get the 1st Number
System.out.println(“Enter first Number”);
a = sc.nextInt();
//Get the 2nd Number
System.out.println(“Enter second Number”);
b = sc.nextInt();
//Find the Addition
c = a +b ;
//Print the result
System.out.println(“The Addition is: “ + c);
}
}
```

- **Write a Java Program to implement a Calculator Program.**

```
import java.io.*;
import java.util.*;
public class Jpgm4
class Main
{
public static void main(String[] args)
{
//declare the necessary variables
char operator;
Double number1, number2, result;
// create an object of Scanner class
Scanner input = new Scanner(System.in);
// ask users to enter operator
System.out.println("Choose an operator: +, -, *, or /");
operator = input.next().charAt(0);
// ask users to enter numbers
System.out.println("Enter first number");
number1 = input.nextDouble();
System.out.println("Enter second number");
number2 = input.nextDouble();
switch (operator)
{
// performs addition between numbers
case '+':
result = number1 + number2;

System.out.println(number1 + " + " + number2 + " = " + result);
break;
// performs subtraction between numbers
case '-':
result = number1 - number2;
System.out.println(number1 + " - " + number2 + " = " + result);
break;
// performs multiplication between numbers
case '*':
result = number1 * number2;
System.out.println(number1 + " * " + number2 + " = " + result);
break;
// performs division between numbers
case '/':
result = number1 / number2;
System.out.println(number1 + " / " + number2 + " = " + result);
break;
default:
System.out.println("Invalid operator!");
break;
}
input.close();
}
}
```

➔ 8. TRAVERSE SYNTAX TREE

```
import java.util.Scanner;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

public class ArithmeticVisitor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an arithmetic expression: ");
        String expression = scanner.nextLine();
        try {
            double result = evaluateExpression(expression);
            System.out.println("Result: " + result);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static double evaluateExpression(String expression) throws ScriptException {
        ScriptEngineManager manager = new ScriptEngineManager();
        ScriptEngine engine = manager.getEngineByName("js");
        Object result = engine.eval(expression);
        if (result instanceof Integer) {
            return ((Integer) result).doubleValue();
        } else if (result instanceof Double) {
            return (Double) result;
        } else {
            throw new IllegalArgumentException("Unexpected result type: " + result.getClass());
        }
    }
}
```

➔ 9. INTERMEDIATE CODE GENERATOR

```
import java.util.Scanner;

public class IntermediateCodeGenerator {
    private static int labelCount = 0;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter an if statement or a while loop:");
        String userInput = scanner.nextLine();

        String intermediateCode = generateIntermediateCode(userInput);
        System.out.println("Intermediate Code:");
        System.out.println(intermediateCode);
    }

    public static String generateIntermediateCode(String userInput) {
        StringBuilder intermediateCode = new StringBuilder();

        // Assuming user input is a simple if statement or while loop
        if (userInput.startsWith("if")) {
            intermediateCode.append(generateIfCode(userInput));
        } else if (userInput.startsWith("while")) {
            intermediateCode.append(generateWhileCode(userInput));
        } else {
            intermediateCode.append("Invalid input. Please enter a valid if statement or while loop.");
        }

        return intermediateCode.toString();
    }

    public static String generateIfCode(String input) {
        // Assuming input format is: if (condition) { ... }
        String condition = input.substring(input.indexOf('(') + 1, input.indexOf(')'));
        String trueLabel = generateLabel();
        String falseLabel = generateLabel();

        StringBuilder code = new StringBuilder();
        code.append("if ").append(condition).append(" goto ").append(trueLabel).append(" else goto ").append(falseLabel);
        code.append("\n").append(trueLabel).append(":");
        // Assuming intermediate code for if body follows
        code.append("\n").append("// Intermediate code for true branch");
        code.append("\n").append(falseLabel).append(":");
        // Assuming intermediate code for false branch follows
        code.append("\n").append("// Intermediate code for false branch");
        return code.toString();
    }

    public static String generateWhileCode(String input) {
        // Assuming input format is: while (condition) { ... }
        String condition = input.substring(input.indexOf('(') + 1, input.indexOf(')'));
        String loopLabel = generateLabel();
        String exitLabel = generateLabel();

        StringBuilder code = new StringBuilder();
        code.append(loopLabel).append(": if ").append(condition).append(" goto
").append(exitLabel).append(" else goto ").append(loopLabel);
        code.append("\n").append("// Intermediate code for loop body");
        code.append("\n").append("goto ").append(loopLabel);
        code.append("\n").append(exitLabel).append(":");

        return code.toString();
    }

    public static String generateLabel() {
        return "L" + labelCount++;
    }
}
```

➔ 11. Generate machine code for a simple statement.

```
#include <stdio.h>
#include <stdint.h>
// Define MIPS instruction opcodes and function codes
#define LUI_OPCODE 0x0F
#define ORI_OPCODE 0x0D
#define NOP_INSTRUCTION 0x00000000
// Function to generate machine code for "li $s0, 42"
void generateMachineCode() {
uint32_t lui_instruction = (LUI_OPCODE << 26) | (16 << 16) | (42 & 0xFFFF); // lui $s0, 0x0000
uint32_t ori_instruction = (ORI_OPCODE << 26) | (16 << 21) | (16 << 16) | (42 & 0xFFFF); // ori $s0, $s0, 42
// Output machine code in hexadecimal format
printf("Machine code for 'li $s0, 42':\n");
printf("lui $s0, 0x0000: 0x%08X\n", lui_instruction);
printf("ori $s0, $s0, 42: 0x%08X\n", ori_instruction);
printf("nop: 0x%08X\n", NOP_INSTRUCTION);
}
int main() {
generateMachineCode();
return 0;
}
```

➔ 12. Generate machine code for an indexed assignment statement.

```
#include <stdio.h>
#include <stdint.h>
// Function to generate machine code for indexed assignment statement
void generateMachineCode() {
// Define array and index variables
uint32_t array[10] = {0}; // Array of 10 integers initialized to 0
uint32_t index = 3; // Index variable with value 3
// Assign a value to an array element at the given index
array[index] = 42;
// Output the machine code for the indexed assignment statement
printf("Machine code for indexed assignment statement:\n");
printf("lw $t0, %u($zero)\n", index * sizeof(uint32_t)); // Load index into $t0
printf("li $t1, 42\n"); // Load value 42 into $t1
printf("la $t2, array\n"); // Load base address of array into $t2
printf("sw $t1, 0($t2, $t0)\n"); // Store value at indexed location in the array
}
int main() {
generateMachineCode(); // Generate machine code for indexed assignment statement
return 0;
}
```