SAKHI PATEL
wz7995

CODE GENERATION

```
package Compiler;
/*
 * This Program performs the code generaion
 *
 * Skeleton for this program
#Prolog:
.text
.globl  main
main:
move  $fp  $sp                  #frame pointer will be start of active stack
la  $a0  ProgBegin
li  $v0 4
syscall
#End of Prolog
          #all code will go below here...
#Postlog:
la  $a0  ProgEnd
li  $v0  4
syscall
li  $v0  10
syscall
.data
ProgBegin:  .asciiz      "Program Begin\n"
ProgEnd:   .asciiz      "\nProgram  End\n"
 */
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Stack;

import Compiler.Scanner;
import Compiler.Token;

public class Parse
        {

                static Scanner lexer; //object of the Scanner Class used to access Scanner methods
                static String inFile = "C:\\Users\\sakhi\\workspace\\Compiler\\src\\test.txt";//Input File

                protected static Token[] gettoken; //array that stores all the tokens from the scanner
                protected static Token token;
                protected static int i=0;
                public static ExpRec expRec; //Object for accessing Expression_Record(here ExpRec class)
                                                                        //which stores type and location

                static File file;
                static FileWriter fw;
                public static int currOff=0;
                public static int labelNo=0;
                public static int loopNo=1;
                static Symbol_Table st;
                static Hashtable<String,IdenInfo> newBlock = null;
                static Stack<Integer> scope_noStack = new Stack<Integer>();
                static int count=-1;
```

```
public Parse() throws IOException
{
        file = new File("output.s");
        fw = new FileWriter(file);
}
```

```
private static void Prolog()
{
        codeGen("#Prolog:");
        codeGen(".text");
        codeGen(".globl main");
        codeGen("main:");
        codeGen("la $a0 ProgStart");
        codeGen("li $v0 4");
        codeGen("syscall");
        codeGen("#End of Prolog\n");
}
```

```
//generates the actual code between prolog and postlog

private static void codeGen(String code){
        try
        {
                fw.write(code+"\n");
        }
        catch (IOException e)
        {
                e.printStackTrace();
        }
}
```

```
//generate postlog

private static void Postlog() throws IOException
{
        codeGen("\n#PostLog:");
        codeGen("la $a0 ProgEnd");
        codeGen("li $v0 4");
        codeGen("syscall");
        codeGen("li $v0 10");
        codeGen("syscall");
        codeGen(".data");
        codeGen("ProgBegin: .asciiz \"Program Begin\"");
        codeGen("ProgEnd: .asciiz \"Program End\"");
        fw.close();
}
```

```
//returns the next lable
public static String jumpLable(String str)
{
        String lable=null;
        if(str.equals("IF"))
        {
                lable = "Lable"+str+labelNo;
                labelNo++;
        }
        else if(str.equals("WHILE"))
        {
                lable = "Lable"+str+labelNo;
                labelNo++;
        }
        return lable;
}
```

```
        public static void blockst(Object tok) throws IOException, ParseException
        {
 //RULE [1] program start
        Prolog();
        System.out.println("1");
        S(tok);
        match(token,18);
        Postlog();
}
```

```
public static void S(Token tok) throws IOException, ParseException
{

                match(tok,7);
                // For begin
                newBlock = new Hashtable<String, IdenInfo>();
                st.mainHTable.add(newBlock) ;
                count= count+1;
                scope_noStack.add(count);

                Stats(token);
                // For end
                scope_noStack.pop();
                st.old_mainHTable.add(newBlock);
                count= count-1;

                System.out.println(token.tok());
                match(token,8);                         //(Token no : 8 for END)
```

```
}



        // RULE [3] || [4] -> stats :   statmt ';'  stats   | <empty>
/*token no.   1 -> identifier, token no. 3 -> Keyword, token no. 7 -> BEGIN, token no. 9 -> IF,
 token no. 11 -> WHILE, token no. 13 -> READ, token no. 16 -> ';' */

        private static void Stats(Token tok) throws IOException, ParseException
        {
if(tok.tok()==1 || tok.tok()==3 || tok.tok()==7 || tok.tok()==9 || tok.tok()==11 || tok.tok()==13 ||
tok.tok()==16)
                {
                //System.out.println("3");
                Statmt(tok);
                match(token,16);
                   Stats(token);

                }

else
                {
                        //System.out.println("4");
                        //System.out.println("Success");
                        //System.exit(0);
        }

        }




//check if token is a statement
// RULE[6,7,8,9,10,11,12] statmt: decl | ifstat | assstat |  blockst | loopst | iostat | <empty>
public static void Statmt(Token token) throws IOException //check if token is a statement, ParseException
, ParseException
        {
                if(token.tok()==1 )                     //token 1 -> identifier
                {
                 //     System.out.println("8");
                        asst(token);

                }
                else if(token.tok()==3)                         //token 3 -> keywords
                {
                decl(token);
                }
                else if(token.tok()==7)                         //token 7 -> BEGIN
                {
                //System.out.println("9");
                S(token);
                }
                else if(token.tok()==9)                           //token 9 -> IF
                {
                //System.out.println("7");
                        If(token);
                }
                else if(token.tok()==11)                        //token 11 -> WHILE
                {
                        //System.out.println("10");
                        loop(token);

                }
                else if(token.tok()==13)                        //token 13 -> READ WRITE WRITELN
                        {
                        //System.out.println("11");
                        IO(token);
```

```
                }
                  else
                {
                        //System.out.println("12");

                }

        }
```

```
//RULE [5] decl:  BASICTYPETOK IDTOK
        public static void decl(Token tok) throws IOException, ParseException
        {
                match(tok,3);                           //token 3 -> keywords
                IdenInfo iden=st.find_in_current(token.name(), scope_noStack.peek());
                if(iden!=null){
                        System.out.println("Error");
                }
                else{
                        st.Insert(newBlock, token.name(), new
IdenInfo(token.name(),scope_noStack.peek(),tok.tok(),currOff));
                        currOff-=4;
                }
                match(token,1);                         //token 1 -> identifier

        }
```

```
//RULE [13] assstat ->  idref   ASTOK  expression
        public static void asst(Token tok) throws IOException, ParseException
        {
                //System.out.println("13");
                int lhs = 0,rhs=0,lhsLoc = 0;
                IdenInfo iden=st.find_in_All(tok.name());
                if(iden==null)
                        System.out.println("Variable "+iden.name+" not declared");
                else{
                        lhs=iden.type;
                        lhsLoc=iden.loc;
                }
                idref(tok);                             //call  idref
                match(token,19);                        // := (ASTOK) token
                Exp(token);
                if(iden!=null && lhs==expRec.getType()){
                        codeGen("lw $t0 "+expRec.getLoc()+"($sp)");
                        codeGen("sw $t0 "+lhsLoc+"($sp)");
                }
                else{
                        System.out.println("Type mismatch");
                }
        }
```

```
//RULE [14] IFTOK  expression THENTOK condition
 public static void If(Token tok) throws IOException //If Condition , ParseException
, ParseException
        {
                String jump=jumpLable("IF");


         //     System.out.println("14");
                match(tok,9);                                   //token for IF

                Exp(token);
                codeGen("lw $t0 "+(currOff+4)+"($sp)");
                codeGen("beq $t0 $0 "+jump);

                match(token,10);                                //token for THEN
                Stats(token);
                codeGen(jump+":");

         }









//RULE [15] WHILETOK expression DOTOK statmt
        private static void loop(Token tok) throws IOException //While Loop , ParseException
, ParseException
{
        //System.out.println("15");
        String jump=jumpLable("WHILE");

        match(tok,11);                          //token num for WHILE
        codeGen("loop"+ loopNo++ +":");

        Exp(token);
        codeGen("lw $t0 "+(expRec.getLoc())+"($sp)");
        codeGen("beq $t0 $0 "+jump);

        match(token,12);                         //token num for DO
        Stats(token);
        codeGen("j loop"+(loopNo-1));
        codeGen(jump+":");


}
```

```
//RULE[16] || [17] READTOK ( idref ) | WRITETOK (Exp)
public static void IO(Token tok) throws IOException //IO
, ParseException
        {
                        if (tok.tok()==13)
                          {
                 //             System.out.print("16 ");
                               match(tok, 13);                  //token for Read
                               match(token, 14); //token for '('
                                    idref(token);
                                    match(token, 15);                   //token num for ')'
                                }

                        else if (token.tok()==20)
                                {
                               //             System.out.print("17 ");
                                     match(token, 20);           //token num for WRITE
                                            match(token, 14);           //token num for (
                                            Exp(token);
                                            match(token, 15);           //token num for )
                                        }
                                   else    System.exit(0);

        }









 // RULE [18] E -> T E_
        public static void Exp(Token tok) throws IOException  //Expression
, ParseException
        {
                //System.out.println("18");
                T(tok);
                E_(token);

        }









//RULE [19]||[20] E_ -> ADD T E_ | eps
public static void E_(Token tok) throws IOException //E Prime
, ParseException
        {

                if(tok.tok()==4)
                {
                String operator="";
                int op1Loc=expRec.getLoc();
                if(tok.name().equals("+"))
                        operator="add";
                else if(tok.name().equals("-"))
                        operator="sub";
                //System.out.println("19");
```

```
                match(tok,4);                           //token for '+'
                T(token);

                codeGen("lw $t0 "+op1Loc+"($sp)");
                codeGen("lw $t1 "+expRec.getLoc()+"($sp)");
                codeGen(operator+"$t0 $t0 $t1");
                codeGen("sw $t0 "+currOff+"($sp)");
                expRec.loc=currOff;
                currOff-=4;

                E_(token);

                }
                else
                {
                 //     System.out.println("20");

        }

}
```

```
//RULE [21] T -> RF T_
public static void T(Token tok) throws IOException //Term
, ParseException
        {
                //System.out.println("21");
                RF(tok);
                T_(token);

        }
```

```
//RULE [22] || [23] T-> MUL RF T_ | eps
private static void T_(Token tok) throws IOException //T Prime
, ParseException
        {
                if(tok.tok()==5)
                {
                        int op1Loc=expRec.getLoc();
                        String operator=null;
                        if(tok.name().equals("*"))
                                operator="mul";

                        else if(tok.name().equals("/"))
                                operator="div";

                        //System.out.println("22");
                        match(tok,5);   //token for *
                        RF(token);
                        codeGen("lw $t0 "+op1Loc+"($sp)");
                        codeGen("lw $t1 "+expRec.getLoc()+"($sp)");
                        codeGen(operator+"$t0 $t0 $t1");
                        codeGen("sw $t0 "+currOff+"($sp)");
                        expRec.loc=currOff;
                        currOff-=4;
```

```
                        T_(token);

                }
                else
                {
                        //System.out.println("23");

}

}


//RULE [24] RF -> F F_

private static void RF(Token tok) throws IOException, ParseException
  {
                //System.out.println("24");
                F(tok);
                F_(token);

}




//RULE [25] || [26] F_ -> REL F   | eps
private static void F_(Token tok) throws IOException //F Prime
, ParseException
        {
                if(tok.tok()==6)
                        {
                        //System.out.println("25");
                        match(tok,6);           //token num 6 for relational operators
                        F(token);
                                        }
                else
                {
                        //System.out.println("26");

        }

        }



//RULE [27||28||29||30] F ->NOT ID | LIT | ID | (E)
private static void F(Token tok) throws IOException             //Factor
, ParseException
        {
                if(tok.tok()==17) //NOT ID
                {
                        //System.out.println("27");
                        match(tok,17);
                        idref(token);

                }
                else if(tok.tok()==2)//LITERAL
                {
                        //System.out.println("29");
                        match(tok,2);

                }
                else if(tok.tok()==1)//ID
                {
                        //System.out.println("28");
                        idref(tok);

                }
```

```
                else if(tok.tok()==14)//(F)
                {
                        //System.out.println("30");
                        match(tok,14);
                        Exp(token);
                        match(token,15);

                }

        }



//RULE [31] idref -> IDTOK
        public static void idref(Token tok) throws IOException, ParseException
        {
                if(tok.tok()==1)
                {
                        //System.out.println("31");
                        match(token,1);

                }

}




//MATCH
public static void match(Token tok, int exp)
{
                //System.out.print(tok.name());
        //System.out.println(tok.tok()+" "+exp);
                if(tok.tok()==exp)
        {               i++;
        if(i<gettoken.length)
                        token = gettoken[i];

        }                               //returns next character.
                else
                {
                System.out.println("Error");
                System.exit(0);
                }

        }




//MAIN


        public static void main(String[] args) throws IOException, ParseException
        {

                file = new File("output.s");
                fw = new FileWriter(file);
                        // Buffer of lines from input file
                ArrayList<Token> list = new ArrayList<Token>();
                lexer = new Scanner(inFile);//call to line 1
                Token t;
```

```
                while ((t = lexer.nextToken()) != null) {

                        Token k = t;
                        //System.out.println(k);
                        list.add(k);
                }
                list.add(new Token(0,""));
                 gettoken = list.toArray(new Token[list.size()]);


                        token = gettoken[i];
                        //System.out.println(token.name()+" "+token.tok());
                        blockst(token);
                        if(token.tok()==0)
                                System.out.println("Success");
                        else
                                System.out.println("Failure");

                        // token no. -1 -> end of file
                                System.exit(0);
        }

        }
```

EXPRESSION_RECORD CLASS

```
package Compiler;


public class ExpRec
{
        int type;
        int loc;

        public ExpRec(int t,int l)
        {
                this.type=t;
                this.loc=l;
        }


        public int getType()
        {
                return this.type;
        }

        public int getLoc()
        {
                return this.loc;
        }
}
```

IDENTIFIER INFO CLASS

```
package Compiler;

public class IdenInfo
{
        public String name;
        public int scope;
        public int type;
        public int loc;

        public IdenInfo(String n,int s,int t,int l)
        {
                this.name=n;
                this.scope=s;
                this.type=t;
                this.loc=l;
        }
}
```

TOKEN CLASS

```
package Compiler;


import java.text.ParseException;

public class Token
{
        private int token;
        private String name;
        public Token(int token,String n)
        {
                this.token = token;
                this.name=n;
        }

        //@Override
        public int tok()
        {
                return token;
        }

        public String name()
        {
                return this.name;
        }

        public int epsilon() throws ParseException
        {
          throw new ParseException("Unexpected end of input", token);
    }

}
```

SYMBOL TABLE CLASS

```
package Compiler;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Scanner;
import java.util.Set;
import java.util.Stack;

public class Symbol_Table {
	static ArrayList<Hashtable<String, IdenInfo>> mainHTable = new ArrayList<Hashtable<String,
IdenInfo>>();
	static ArrayList<Hashtable<String, IdenInfo>> old_mainHTable = new ArrayList<Hashtable<String,
IdenInfo>>();

	/* Insert Function */
	public static Hashtable<String, IdenInfo> Insert(Hashtable<String, IdenInfo> newBlockk,String key,
IdenInfo i)
	{
		newBlockk.put(key, i);
		return newBlockk;
	}

	/* Find in current Block */
	public static IdenInfo find_in_current(String finding_key, int curr_block)
	{
		Hashtable<String, IdenInfo> scanBlock = mainHTable.get(curr_block);
		if (scanBlock.containsKey(finding_key))
		{
			return scanBlock.get(finding_key);
		} else
			return null;
	}

	/* Find in All Block */
	public static IdenInfo find_in_All(String finding_key) {
		Iterator<Hashtable<String, IdenInfo>> it = mainHTable.iterator();
		while (it.hasNext()) {
			Hashtable<String, IdenInfo> scanBlock = it.next();
			if (scanBlock.containsKey(finding_key)) {
				return scanBlock.get(finding_key);
			}
		}
		return null;
	}

	/* DISPLAY */
	public static void Display() {
		for (Integer i = 0; i < mainHTable.size(); i++) {
			Hashtable<String, IdenInfo> temp = mainHTable.get(i);
			Set<String> value = temp.keySet();// returns keys contained in
                                                                                                //
mainHTable
			System.out.print("scope" + i + " has:");
			for (String token : value) {
				System.out.println(token + "");
			}
```

```
                        System.out.println(" ");
                }
        }
}
```

Output:

```
Exception in thread "main" java.lang.NullPointerException
        at Compiler.Parse.asst(Parse.java:238)
        at Compiler.Parse.Statmt(Parse.java:177)
        at Compiler.Parse.Stats(Parse.java:155)
        at Compiler.Parse.Stats(Parse.java:157)
        at Compiler.Parse.S(Parse.java:132)
        at Compiler.Parse.main(Parse.java:539)
```

I applied all the rules but still There are some bugs in the program