Manuale
Del compilatore

Riferimento	COMP_V3.0
Versione	3.0
Data	23/01/2019
Destinatario	Prof. G. Costagliola
Presentato da	Antonio De Piano – Matr.: 0522500397
Approvato da	

Coordinatori del Progetto
Prof. Gennaro Costagliola

Partecipanti
Antonio De Piano

Revision History

Data	Versione	Descrizione	Autori
14/01/2019	1.0	Prima stesura	Antonio De Piano
15/01/2019	1.1	Ho aggiunto il for con scope	Antonio De Piano
16/01/2019	2.0	Ho aggiunto gli operatori (incremento, decremento e modulo)	Antonio De Piano
18/01/2019	2.1	Ho aggiunto le funzioni a funzioni	Antonio De Piano
20/01/2019	2.2	Ho aggiunto lo switch	Antonio De Piano
21/01/2019	2.3	Ho aggiunto l'operatore logico AND alla prima esercitazione	Antonio De Piano
22/01/2019	3.0	Release	Antonio De Piano

Sommario

FOR CON SCOPE

LEX

Aggiungo alla classe LEX la parola chiave for.

```
"for"                          {return symbol (YASPL3Sym.FOR, yytext());}
```

CUP

Aggiungo il terminale FOR

```
terminal                FOR;
```

STAT (*Aggiungo a STAT quest'insieme di Regole & Azioni*)

```
|
FOR LPAR ID:id1 ASSIGN Expr:expr1 SEMI Expr:expr2 SEMI ID:id2 ASSIGN Expr:expr3 RPAR
Body:bd

{: RESULT=new ForOp("ForOp",new AssignOp("AssignOp",new
Leaf("ID",id1,""),expr1),expr2,new AssignOp("AssignOp",new Leaf("ID",id2,""),expr3),bd); :}
```

Visitor

Ho aggiunto questo metodo all'interfaccia Visitor.

```
Object visit(ForOp node) throws Exception;
```

SyntaxTree

Aggiungo la classe ForOp che estende la classe ScopeNode (ho gestito lo scope anche nel for)

```
package SyntaxTree;

import Visitor.Visitor;

public class ForOp extends ScopeNode{


        public ForOp(String op, AssignOp assign, Node expr, AssignOp increments, Node corpo) {
                super(op,assign,expr, increments, corpo);

        }
        public Object accept(Visitor v) throws Exception{
                return v.visit(this);
        }
}
```

SemanticVisitor

Override del metodo con parametro d'input ForOp

```
@Override
	public Object visit(ForOp node) throws Exception {

		stack.addIdentifier("FOR", new EntryInfo("For"));
		SymbolTable st = new SymbolTable("FOR");
		stack.push(st);

		//setto la symboltable della classe ScopeNode per mantenere il riferimento del
for(usato per type checking)

		node.setSymbolTable(st);

		node.nodeList().get(0).accept(this);
		node.nodeList().get(1).accept(this);
		node.nodeList().get(2).accept(this);
		node.nodeList().get(3).accept(this);

		//System.out.println("assignop type:"+node.nodeList().get(0).getType()+ "
expr1:"+node.nodeList().get(1).getType()+ " expr2:"+node.nodeList().get(2).getType());

		if(!(node.nodeList().get(0).getType().equals("no-type") &&
node.nodeList().get(1).getType().equals("boolean") && node.nodeList().get(2).getType().equals("no-
type")))
			throw new Exception("Type Mismatch ---");


		stack.pop();//rilascio quando ho visitato tutti i figli
		System.out.println("SymbolTable: " + node.getSymbolTable());

		return null;

	}
```

XMLVisitor

Override del metodo con parametro d'input ForOp

```
@Override
	public Object visit(ForOp node) throws Exception {
		return this.visit((OpNode) node);
	}
```

YASPL3ToCVisitor

Override del metodo con parametro d'input ForOp

```
@Override
	public Object visit(ForOp node) throws Exception {
```

```
        ArrayList<Node> nodes= node.nodeList();
        forYes=true;
        fileOutput.write("for(");
        nodes.get(0).accept(this);
        fileOutput.write(";");
        nodes.get(1).accept(this);
        fileOutput.write(";");
        nodes.get(2).accept(this);
        fileOutput.write("){\n");
        nodes.get(3).accept(this);
        forYes=false;
        fileOutput.write("}\n");

        return null;
}
```

OPERATORI

INCREMENTO

CUP

STAT (*Aggiungo a STAT quest'insieme di Regole & Azioni*)

```
                |
                ID:id PLUS PLUS SEMI
                {: RESULT=new IncrementsOp("IncrementOp",new Leaf("ID",id,"")); :};
```

Visitor

Ho aggiunto questo metodo all'interfaccia Visitor.

```
Object visit(IncrementsOp node) throws Exception;
```

SyntaxTree

Aggiungo la classe IncrementsOp che estende la classe OpNode

```
package SyntaxTree;

import Visitor.Visitor;

public class IncrementsOp extends OpNode {

        public IncrementsOp(String op, Leaf id) {
                super(op, id);
                // TODO Auto-generated constructor stub
        }

        public Object accept(Visitor v) throws Exception{
                return v.visit(this);
        }


}
```

SemanticVisitor

Override del metodo con parametro d'input IncrementsOp

```
@Override
        public Object visit(IncrementsOp node) throws Exception {

                node.nodeList().get(0).accept(this);
```

```
                node.setType("no-type");

                if(!node.nodeList().get(0).getType().equals("integer"))
                        throw new Exception("Type Mismatch");
                System.out.println("Operation: "+node.getOp()+" type: "+node.getType());

                return null;
        }
```

XMLVisitor

Override del metodo con parametro d'input IncrementsOp

```
@Override
        public Object visit(IncrementsOp node) throws Exception {
                return this.visit((OpNode) node);
        }
```

YASPL3ToCVisitor

Override del metodo con parametro d'input IncrementsOp

```
@Override
        public Object visit(IncrementsOp node) throws Exception {
                node.nodeList().get(0).accept(this);
                fileOutput.write("++;\n");
                return null;
        }
```

DECREMENTO

CUP

```
                |

                ID:id MINUS MINUS SEMI
                {: RESULT=new DecreaseOp("DecreaseOp",new Leaf("ID",id,"")):};
```

Visitor

Ho aggiunto questo metodo all'interfaccia Visitor.

```
Object visit(DecreaseOp node) throws Exception;
```

SyntaxTree

Aggiungo la classe DecreaseOp che estende la classe OpNode

```
package SyntaxTree;

import Visitor.Visitor;

public class DecreaseOp extends OpNode{

	public DecreaseOp(String op, Leaf id) {
		super(op, id);
	}

	public Object accept(Visitor v) throws Exception{
		return v.visit(this);
	}


}
```

SemanticVisitor

Override del metodo con parametro d'input DecreaseOp

```
@Override
	public Object visit(DecreaseOp node) throws Exception {

		node.nodeList().get(0).accept(this);
		if(!node.nodeList().get(0).getType().equals("integer"))
			throw new Exception("Type Mismatch - Decrease operator only for integers" );
		node.setType("no-type");

		//System.out.println("Operation: "+node.getOp()+" type: "+node.getType());

		return null;
	}
```

XMLVisitor

Override del metodo con parametro d'input DecreaseOp

```
@Override
	public Object visit(DecreaseOp node) throws Exception {
		return this.visit((OpNode)node);
	}
```

YASPL3ToCVisitor

Override del metodo con parametro d'input DecreaseOp

```
@Override
	public Object visit(DecreaseOp node) throws Exception {
		node.nodeList().get(0).accept(this);
		fileOutput.write("--;\n");
		return null;
	}
```

MODULO

LEX

```
"%"                                 {return symbol (YASPL3Sym.MOD, yytext());}
```

CUP

```
terminal                MOD;

precedence nonassoc GT, GE, LT, LE, EQ , MOD;
```

Expr (*Aggiungo a Expr quest'insieme di Regole & Azioni*)

```
                |

                Expr:expr1 MOD Expr:expr2
                {: RESULT = new ModOp("ModOp",expr1,expr2); :}
```

Visitor

Ho aggiunto questo metodo all'interfaccia Visitor.

```
Object visit(ModOp node) throws Exception;
```

SyntaxTree

Aggiungo la classe ModOp che estende la classe OpNode

```
package SyntaxTree;

import Visitor.Visitor;

public class ModOp extends OpNode {

        public ModOp(String op, Node expr1, Node expr2) {
                super(op,expr1 ,expr2);

        }

        public Object accept(Visitor v) throws Exception{
                return v.visit(this);
        }

        public Node getFirstExpr(){
                return this.nodeList().get(0);
        }
```

```
	public Node getSecondExpr(){
		return this.nodeList().get(1);
	}

}
```

SemanticVisitor

Override del metodo con parametro d'input ModOp

```
@Override
	public Object visit(ModOp node) throws Exception {
		// TODO Auto-generated method stub

		Leaf expr1=(Leaf)node.getFirstExpr();
		Leaf expr2=(Leaf)node.getSecondExpr();

		expr1.accept(this);
		expr2.accept(this);


		node.setType("no-type");

		String operation= node.getOp();

		if(operation.equals("ModOp"))
		{
			if((expr1.getType().equals("integer"))&&(expr2.getType().equals("integer")) )
				node.setType("integer");
			else
			{
				throw new Exception("Type Mismatch - Mod operator only for
integers" );
			}
		}

		//System.out.println("Operation: "+node.getOp()+" type: "+node.getType());

		return null;
	}
```

XMLVisitor

Override del metodo con parametro d'input ModOp

```
@Override
	public Object visit(ModOp node) throws Exception {
		return this.visit((OpNode) node);
	}
```

YASPL3ToCVisitor

Override del metodo con parametro d'input ModOp

```
@Override
        public Object visit(ModOp node) throws Exception {
                node.nodeList().get(0).accept(this);
                fileOutput.write("%");
                node.nodeList().get(1).accept(this);
                return null;
        }
```

SWITCH

LEX

```
"case"                  {return symbol (YASPL3Sym.CASE, yytext());}
":"                             {return symbol (YASPL3Sym.TWOPOINTS, yytext());}
"switch"                {return symbol (YASPL3Sym.SWITCH, yytext());}
"break"                 {return symbol (YASPL3Sym.BREAK, yytext());}
"default"               {return symbol (YASPL3Sym.DEFAULT, yytext());}
```

CUP

```
terminal            SWITCH,CASE, TWOPOINTS, BREAK, DEFAULT;
non terminal    Node CaseList;
```

STAT (*Aggiungo a STAT quest'insieme di Regole & Azioni*)

```
CaseList                    ::=CASE Expr:expr1 TWOPOINTS Var_decls:vardecls1
Statements:statements1 BREAK SEMI CaseList:case_list
                                        {:
                                                OpNode toReturn=new
CaseListOp("CaseListOp",new CaseOp("CaseOp",expr1,vardecls1, statements1));
                                                if(case_list!=null){
                                                        for(Node node
:((OpNode)case_list).nodeList()){
                                                                toReturn.addNode(node);

                                                        }

                                                }
                                                RESULT=toReturn;
                                        :}
                                        |
                                        DEFAULT TWOPOINTS Var_decls:vardecls1
Statements:statements1 BREAK SEMI
                                        {: RESULT=new CaseListOp("CaseListOp",new
DefaultOp("DefaultOp",vardecls1, statements1));
                                                :}
                                        |
                                        {:
                                                RESULT=null; :};
```

Visitor

Ho aggiunto questo metodo all'interfaccia Visitor.

```
Object visit(SwitchOp node) throws Exception;
Object visit(CaseOp node)throws Exception;
Object visit(CaseListOp node) throws Exception;
Object visit(DefaultOp node)throws Exception;
```

SyntaxTree

Aggiungo la classe SwitchOp che estende la classe OpNode

```
package SyntaxTree;

import Visitor.Visitor;

public class SwitchOp extends OpNode{

	public SwitchOp(String op, Leaf id, Node caselist) {
		super(op, id, caselist);  }


	public Object accept(Visitor v) throws Exception{
		return v.visit(this);
	}

}
```

Aggiungo la classe CaseOp che estende la classe OpNode

```
package SyntaxTree;

import Visitor.Visitor;

public class CaseOp extends OpNode{

	public CaseOp(String op, Node expr, Node vardecls, Node statements) {
		super(op,expr, vardecls, statements);
	}

	public Object accept(Visitor v) throws Exception{
		return v.visit(this);
	}

}
```

Aggiungo la classe DefaultOp che estende la classe OpNode

```
package SyntaxTree;

import Visitor.Visitor;

public class DefaultOp extends OpNode{

	public DefaultOp(String op, Node vardecls, Node statements) {
		super(op, vardecls, statements);
	}
```

```
	public Object accept(Visitor v) throws Exception{
		return v.visit(this);
	}


}
```

Aggiungo la classe CaseListOp che estende OpNode

```
package SyntaxTree;

import Visitor.Visitor;

public class CaseListOp extends OpNode {

	public CaseListOp(String op, Node casedefaultop) {
		super(op, casedefaultop);
	}

	public Object accept(Visitor v) throws Exception{
		return v.visit(this);
	}
}
```

SemanticVisitor

Override dei metodi con parametro d'input SwitchOp, CaseOp, CaseListOp DefaultOp

```
@Override
	public Object visit(SwitchOp node) throws Exception {

		//Visito il leaf dello switch op - sarebbe un Expr
		node.nodeList().get(0).accept(this);
		switch_type=node.nodeList().get(0).getType();


		//Visito la lista dei caseListOp
		node.nodeList().get(1).accept(this);

		return null;
	}

	@Override
	public Object visit(CaseOp node) throws Exception {

		ArrayList<Node> nodeList=node.nodeList();

		if(nodeList != null) {
			for(Node n:nodeList) {
```

```
                        if(n!=null ){

                                n.accept(this);

                        }
                }
        }

        if(!node.nodeList().get(0).getType().equals(switch_type))
                throw new Exception("Type Mismatch in SwitchOp" );

        return null;
}

@Override
public Object visit(CaseListOp node) throws Exception
{
        ArrayList<Node> nodeList=node.nodeList();
        if(nodeList != null) {
                for(Node n:nodeList) {
                        if(n!=null ){

                                n.accept(this);

                        }
                }
        }
        return null;
}


@Override
public Object visit(DefaultOp node) throws Exception {
        node.nodeList().get(0).accept(this);
        node.nodeList().get(1).accept(this);
        return null;
}
```

XMLVisitor

Override del metodo con parametro d'input SwitchOp,CaseOp, CaseListOp e DefaultOp

```
@Override
        public Object visit(SwitchOp node) throws Exception {
                return this.visit((OpNode)node);
        }

        @Override
        public Object visit(CaseOp node) throws Exception {
                return this.visit((OpNode)node);
        }
```

```
@Override
public Object visit(DefaultOp node) throws Exception {
        return this.visit((OpNode)node);
}
```

YASPL3ToCVisitor

Override del metodo con parametro d'input SwitchOp, CaseOp, CaseListOp e DefaultOp

```
@Override
        public Object visit(SwitchOp node) throws Exception {
                fileOutput.write("switch(");
                node.nodeList().get(0).accept(this);
                fileOutput.write("){\n");
                node.nodeList().get(1).accept(this);
                fileOutput.write("}\n");
                return null;
        }

        @Override
        public Object visit(CaseOp node) throws Exception {

                fileOutput.write("case ");
                node.nodeList().get(0).accept(this);
                fileOutput.write(":\n");
                node.nodeList().get(1).accept(this);
                node.nodeList().get(2).accept(this);
                fileOutput.write("break;\n");

                return null;
        }

        @Override
        public Object visit(CaseListOp node) throws Exception {
                for(int i=0;i<node.nodeList().size();i++)
                {
                        node.nodeList().get(i).accept(this);
                }
                return null;
        }

        @Override
        public Object visit(DefaultOp node) throws Exception {
                fileOutput.write("default:\n");
                node.nodeList().get(0).accept(this);
                node.nodeList().get(1).accept(this);
                fileOutput.write("break;\n");
                return null;
        }
```

ARRAY – DICHIARAZIONE E USO

LEX

```
"["                     {return symbol (YASPL3Sym.LQPAR, yytext());}
"]"                     {return symbol (YASPL3Sym.RQPAR, yytext());}
```

CUP

```
terminal            LQPAR,RQPAR;

non terminal    Node Array_init;

Var_decl (Aggiungo a Var_decl quest'insieme di Regole & Azioni)

|

Type:type ID:id LQPAR Expr:expr1 RQPAR SEMI
{: RESULT=new ArrayOp("ArrayOp",type,new ArrayIndexOp("ArrayIndexOp",new
Leaf("ID",id,""),expr1));
:};
```

Visitor

Ho aggiunto questo metodo all'interfaccia Visitor.

```
Object visit(ArrayOp node) throws Exception;
Object visit(ArrayAssignOp node) throws Exception;
Object visit(ArrayIndexOp node) throws Exception;
```

SyntaxTree

Aggiungo la classe ArrayOp,ArrayAssignOp,ArrayIndexOp che estendono la classe OpNode

```
package SyntaxTree;

import Visitor.Visitor;

public class ArrayOp extends OpNode {

        public ArrayOp(String op, Node type, Node arrayindex) {
                super(op, type, arrayindex);
                // TODO Auto-generated constructor stub
        }

        public Object accept(Visitor v) throws Exception{
                return v.visit(this);
        }
}
```

```
package SyntaxTree;

import Visitor.Visitor;

public class ArrayIndexOp extends OpNode{

	public ArrayIndexOp(String op, Leaf id, Node index) {
		super(op,id,index);
		// TODO Auto-generated constructor stub
	}

	public Object accept(Visitor v) throws Exception{
		return v.visit(this);
	}

	public Node getFirstExpr(){
		return this.nodeList().get(0);
	}

	public Node getSecondExpr(){
		return this.nodeList().get(1);
	}

}

package SyntaxTree;

import Visitor.Visitor;

public class ArrayAssignOp extends UseNode {

	public ArrayAssignOp(String op, Node arrayindex, Node expr) {
		super(op,arrayindex,expr);
	}

	public Object accept(Visitor v) throws Exception{
		return v.visit(this);
	}

	public Node getFirstExpr(){
		return this.nodeList().get(0);
	}

	public Node getSecondExpr(){
		return this.nodeList().get(2);
	}
}
```

SemanticVisitor

Override dei metodi con parametro d'input ArrayOp, ArrayIndexOp, ArrayAssignOp

```
@Override //Dichiaro l'array, aggiungendolo alla tabella dei simboli come Array --> int a[3]; sotto HEAD
	public Object visit(ArrayOp node) throws Exception {


		node.nodeList().get(0).accept(this);

		ArrayIndexOp id=(ArrayIndexOp)node.nodeList().get(1);
		Leaf var=(Leaf)id.getFirstExpr();
		stack.addIdentifier(var.getVal(), new EntryInfo("Array", node.nodeList().get(0).getType()));
		var.accept(this);


		return null;
	}

	@Override //Operazioni con gli array, l'assegnamento è valido solo tra tipi uguali.
	public Object visit(ArrayAssignOp node) throws Exception {
		//Node arrayindex, Node expr questo ricevo in input
		node.nodeList().get(0).accept(this);
		node.nodeList().get(1).accept(this);


		//System.out.println(id.getType()+" -- "+node.nodeList().get(2).getType());
		if(!node.nodeList().get(0).getType().equals(node.nodeList().get(1).getType()))
			throw new Exception("Type Mismatch in Array assignop");
		return null;
	}

	@Override
	//Visit per la posizione dell'array es. a[45]; in input: variabile e int_const "indice"
	public Object visit(ArrayIndexOp node) throws Exception {
		node.nodeList().get(0).accept(this);
		node.nodeList().get(1).accept(this);
		node.setType(node.nodeList().get(0).getType());
		return null;
	}
```

XMLVisitor

Override del metodo con parametro d'input ArrayOp, ArrayIndexOp e ArrayAssignOp.

```
	@Override
	public Object visit(ArrayOp node) throws Exception {
		return this.visit((OpNode)node);
	}

	@Override
```

```
public Object visit(ArrayAssignOp node) throws Exception {
        return this.visit((OpNode)node);
}

@Override
public Object visit(ArrayIndexOp node) throws Exception {
        return this.visit((OpNode)node);
}
```

YASPL3ToCVisitor

Override del metodo con parametro d'input ArrayOp, ArrayIndexOp, ArrayAssignOp.

```
@Override
public Object visit(ArrayOp node) throws Exception {
        node.nodeList().get(0).accept(this);
        String type=null;

        if( node.nodeList().get(0).getType().equals("integer") ) {
                type= "int";
        }else if(node.nodeList().get(0).getType().equals("boolean")){
                type= "bool";
        }
        else if(node.nodeList().get(0).getType().equals("double")){
                type= "double";
        }
        else if(node.nodeList().get(0).getType().equals("string")){
                type= "char";
        }
        else if(node.nodeList().get(0).getType().equals("char")){
                type= "char";
        }


        fileOutput.write(type+" ");
        node.nodeList().get(1).accept(this);
        fileOutput.write(";\n");
        return null;
}

@Override
public Object visit(ArrayAssignOp node) throws Exception {
        node.nodeList().get(0).accept(this);
        fileOutput.write("=");
        node.nodeList().get(1).accept(this);
        fileOutput.write(";\n");
        return null;
}

@Override
public Object visit(ArrayIndexOp node) throws Exception {
        node.nodeList().get(0).accept(this);
```

```
        fileOutput.write("[");
        node.nodeList().get(1).accept(this);
        fileOutput.write("]");
        return null;
    }
```

AND – Esercitazione 1

Ho legato l'operatore AND allo stato stato 0 (stato iniziale), da li richiamo lo stato 60 e infine lo stato 61.

Aggiungo allo stato 0 =>

```
……
else if(this.character=='&')
{
        this.state=60;
}
```

Aggiungi lo stato 60 =>

```
case 60:{
        index=file.read();
        if(index!=-1)
        {
                this.character=(char)index;
                if(this.character=='&')
                {
                        this.state=61;
                }
                else
                {
                        this.state=50;
                        this.retract(file);
                        token=new Token("ERROR");
                }
        }
        else
        {
                this.state=50;
                token=new Token("ERROR");
        }
}
break;
```

Aggiungi lo stato 61 =>

```
case 61:
        {
                this.state=50;
                token=new Token("AND","&&");
        }
        break;
```