For stage zero, I can successful parse action and loop. At first, I create PRONode to implements Robot Program which contain one or more statement node. Thus, I put List<STMTNode> statements here. The execute method that interface method in robot program node, which control the behavior of robot, And using tostring method to print parsing text . Because of the action and loop parse statement node and action need execute method. Thus, interface action node and loop node implements statements node respectively. Block node which inside of loop implements statements and block contain one or more statement. Move class and turnL, turnR, takeFuel, wait implements action node. That is all rules in slide.

```
// program
class PRONode implements RobotProgramNode{
    List<STMTNode> statements;
    public PRONode(){
        statements = new ArrayList<STMTNode>();
    }
    @Override
    public void execute(Robot robot) {
        for(STMTNode s : statements)
            s.execute(robot);
    }
    public List<STMTNode> getStatements() {
        return statements;
    }

class BLOCKNode implements STMTNode {
    List<STMTNode> statements;
    public BLOCKNode(){
        this.statements = new ArrayList<STMTNode>();
    }
    @Override
    public void execute(Robot robot) {
            for(STMTNode s : statements) {
                s.execute(robot);
            }
    }
```

When finish all method above, finally connect them together.

```
static RobotProgramNode parseProgram(Scanner s) {
    // THE PARSER GOES HERE
    PRONode main = new PRONode();
    while(s.hasNext()) {
        main.getStatements().add(parseStatementNode(s));
    }
    return main;
}

// STMT ::= ACT ";" | LOOP | IF | WHILE | ASSGN ";"
static STMTNode parseStatementNode(Scanner s){
    if(s.hasNext(ACTION)) {
        return parseAction(s);
    }else if(checkFor("loop", s)) {
        return parseLoop(s);
    }else if(checkFor("if", s)) {
        return parseIf(s);
    }else if(checkFor("while", s)) {
        return parseWhile(s);
    }else if(s.hasNext(VARIABLES))  {
        return parseASSGN(s);
    }
    else{
        fail("Expecting ACT|LOOP|WHILE|IF|ASSGN ;", s);
    }
    return null;
}
```

```
//LOOP ::= loop BLOCK
static STMTNode parseLoop(Scanner s) {
    BLOCKNode blockNode = parseBlock(s);
    return new LOOPNode(blockNode);
}
```

Other statements like action or block was using the same way to get new type node in different class node, but different logic. There are conditions In block and action, that represent not empty in block and action must be the class type that is Move class turnL, turnR, takeFuel and wait. And the sign was expected in some case. For example, “{“ “,” ”}”.

For Stage one, add condition, sensor and number. Condition include great then class , less then class, equal then class. I create constructor in these class which contain sensor node and number node. And according logic to compare the value of sensor node and number node. Both node classes which contain execute and gettype. Gettype is return the node type which help us to check error. The difference is that sensor and number return int. But condition return boolean like great than. This is parse condition method. Both left and right can not be sensor. If both of them are number,number can not be same value in logic equal. Because it will cause infinite loop.

```
}else if(checkFor("gt", s)){
    require(OPENPAREN, "need ( ", s);
    EXPNode l = parseEXP(s);
    require(COMMON, "need , ", s);
    EXPNode r = parseEXP(s);
    if((l.getType()=="sensor"&&r.getType()=="sensor")) {
        fail("sensor left, num right",s);
    }
    require(CLOSEPAREN, "need ) ", s);
    return new gtNode(l, r);
}else if(checkFor("lt", s)){
    require(OPENPAREN, "need ( ", s);
    EXPNode l = parseEXP(s);
    require(COMMON, "need , ", s);
    EXPNode r = parseEXP(s);
    if((l.getType()=="sensor"&&r.getType()=="sensor")) {
        fail("wrong",s);
    }
    require(CLOSEPAREN, "need ) ", s);
    return new ltNode(l, r);
}else if(checkFor("eq", s)){
    require(OPENPAREN, "need ( ", s);
    EXPNode l = parseEXP(s);
    require(COMMON, "need , ", s);
    EXPNode r = parseEXP(s);
    if((l.getType()=="sensor"&&r.getType()=="sensor")) {
        fail("sensor left, num right",s);
    }

}
if((l.getType()=="num"&&r.getType()=="num")&&l.getNum()==r.getNum()) {
    fail("infinite loop",s);
```

For stage two, I add expression class, else in if node. In this stage, I changed sensor and number node class to implement expression node class. According to slide parsing logic, sensor and number is type of expression node. A new operation node class also implements expression node. Operation node include sub, add, multiple and divide. Here is a multiple example, and the others was written in same way, but different logic. When two expression node was put in multiple class that calculate the value of it. When finish all class in expression, add new block node which represent else node in if node. If condition is true, then run ifblock else run the elseblock. Finally, add expression to move and wait in sensor node.

```
class mulNode implements OPNode{
    EXPNode left;
    EXPNode right;
    public mulNode(EXPNode left,EXPNode right){
        this.left = left;
        this.right = right;
    }
    @Override
    public int execute(Robot robot) {
        return left.execute(robot) * right.execute(robot);
    }
    public String toString(){
        return "mul( "+left+", "+ right + ")";
    }
    @Override
    public String getType() {
        return "op";
    }
    @Override
    public int getNum() {
        // TODO Auto-generated method stub
        return 0;
    }
```

For stage three, I was done variable and elseif and access to barrelLR and FB. Creating var class which implements assignment node class. In assignment class it contains var node and expression node in constructor. I give var node a integer in it’s constructor to represent value of expression. If we do not have this var node then add it, otherwise, put expression node value to number of var node. Else if map was add to if node class, because else if contain tow significant component which is condition and block. Thus, I use map to store them in to else if node. Logic code is too long in else if. Here is part of them. It have five possible situations. When we do not have elseif node. It can be ifblock or elseblock. If we have elseif, it can be ifblock, elseifblock, or else block. For access to barrelLR and FB, add new constructor which have parameter expression node. Thus, that is optional. If exp is not empty, control robot for steps.

```
class ASSGNNode implements STMTNode{
    VARNode var;
    EXPNode exp;
    public ASSGNNode(VARNode var,EXPNode exp) {
        this.var = var;
        this.exp = exp;
    }
    @Override
    public void execute(Robot robot) {
        boolean haveVariable = false;
        VARNode variable = null;
        for(VARNode e : Parser.variables) {
            if(e.value.equals(var.value)) {
                variable = e;
                haveVariable = true;
                break;
            }
        }
        if(haveVariable){
            int val = exp.execute(robot);
            variable.num = val;
        }else if(!haveVariable) {
            Parser.variables.add(var);
        }
    }
    public String toString(){
        return var + " = " + exp;
    }
}
```

```java
class VARNode implements EXPNode{
    String value = null;
    int num;
    public VARNode(String v) {
        this.value = v;
    }
    @Override
    public int execute(Robot robot) {
        for(VARNode e : Parser.variables){
            if(this.value.equals(e.value)){
                return e.num;
            }
        }
        return 0;
    }
    public String toString(){
        return value;
    }
    @Override
    public String getType() {
        return " ";
    }
    @Override
    public int getNum() {
        // TODO Auto-generated method stub
        return 0;
    }
}
```

```java
      ,
      if(elifBlocks.size() > 0) {
          if(elseBlock == null){
              if(condition.evaluate(robot)) {
                  ifBlock.execute(robot);
              }else {
                  for(Map.Entry<CONDNode, BLOCKNode> e : elifBlocks.entrySet()){
                      if(e.getKey().evaluate(robot)){
                          e.getValue().execute(robot);
                          break;
                      }
                  }
              }
          }else{
              if(condition.evaluate(robot)){
                  ifBlock.execute(robot);
              }else{
                  int num = 0;
                  for(Map.Entry<CONDNode, BLOCKNode> e : elifBlocks.entrySet()){
                      if(e.getKey().evaluate(robot)){
                          num++;
                          e.getValue().execute(robot);
                          break;
                      }
                  }
                  if(num == 0){
                      elseBlock.execute(robot);
                  }
              }
```