Block 1: Compiler Construction

**CC1.1**Back end: “Achterkant” van een compiler, waar de gelezen code omgezet wordt naar code die voor de computer uitvoerbaar is.  
Front end: “Voorkant” van een compiler, waar de code gelezen en geanalyseerd wordt.  
Grammar: Beschrijving van een taal en hoe woorden binnen die taal opgebouwd zijn  
Instruction scheduling: Het herordenen van instructies om code snel uit te kunnen voeren   
Instruction selection: De instructies die nodig zijn om een programma uit te voeren  
Optimizer: Kan IR-code analyseren en evt. herschrijven om het sneller uitvoerbaar te maken  
Parsing: Het bepalen of een woord onderdeel is van de taal  
Register allocation: Het toewijzen van virtuele registers aan fysieke registers  
Scanning: Leest een input-stroom en zet deze om in een stroom van woorden  
Type checking: Het controleren van de blocks op contextuele fouten

**CC1.2**1. “Most students is good programmers.”

1. Sentence
2. Subject *verb* Object *endmark*
3. Modifier *noun verb* Modifier *noun endmark*
4. *Adjective noun verb adjective noun endmark*

2. Het werkwoord is in enkelvoudige vorm, terwijl het onderwerp van de zin een meervoud is. Dit is te vergelijken met het toewijzen van de waarde “test” aan een integer variabele. De compiler ziet dit bij het type checking.

3. Scanning/parsing: 1, Type checking: 2

**CC1.3**   
Subject -> particle Subject’  
Subject -> Subject’  
Subject’ -> noun  
Subject’ -> Modifier Subject’  
Subject’ -> Subject’ noun  
Modifier -> Modifier adjective  
Modifier -> adjective  
Object -> particle Object’  
Object’ -> Modifier noun  
Object -> Object’

**CC1.4**1. d ← d + 2 × (a + b)

loadAI Rarp, @d => Rd  
loadI 2 => R2  
loadAI Rarp, @a => Ra  
loadAI Rarp, @b => Rb  
add Ra, Rb => Ra  
mult R2, Ra => Ra  
add Rd, Ra => Rd  
storeAI Rd => Rarp, @d

2. loadAI Rarp, @a => Ra  
loadAI Rarp, @b => Rb  
add Ra, Rb => Ra  
add Ra, Ra => Ra  
loadAI Rarp @d => Rb  
add Ra, Rb => Rb  
storeAI Rb => Rarp, @d

3. loadAI Rarp, @a => Ra 1 - 3  
loadAI Rarp, @b => Rb 2 - 4  
loadAI Rarp, @d => Rd 3 - 5  
add Ra, Rb => Ra 5 - 5  
add Ra, Ra => Ra 6 - 6  
add Ra, Rd => Rd 7 - 7   
storeAI Rb => Rarp, @d 8 - 10