

Oblig 2: AeroScript-språket

Frist: 6. oktober 2026 kl. 23:59

1 Mål

I den andre obligatoriske oppgaven skal du utvide det abstrakte syntakstreet (AST) som utgjør strukturen til AeroScript. Du skal implementere både en lekser og en parser for å analysere og kjøre AeroScript-kode. Deretter skal du utvikle en komplett interpret ved å bygge videre på uttrykks-evaluatoren du laget i Oblig 1.

2 Syntaks

For enkel referanse finner du den konkrete syntaksen til AeroScript i Figur 1.

$n \in \mathbb{Q}^p$	
$id \in \text{NAME}$	
<i>program</i>	$:= (\text{execution})^+$
<i>execution</i>	$:= \text{'} \rightarrow \text{'? } id \text{' } \{ \text{' } (statement)^* \text{' } \} \text{' } (\text{' } \rightarrow \text{' } id \text{' })^?$
<i>statement</i>	$:= action$
<i>action</i>	$:= (acDock \mid acMove \mid acTurn \mid acAscend \mid acDescend) \text{' } (\text{'for' } expression \text{'seconds' } \mid \text{'at speed' } expression)^?$
<i>acDock</i>	$:= \text{'return to base'}$
<i>acMove</i>	$:= \text{'move' } (\text{'to' } point \mid \text{'by' } expression)$
<i>acTurn</i>	$:= \text{'turn' } (\text{'right' } \mid \text{'left' })^? \text{'by' } expression$
<i>acAscend</i>	$:= \text{'ascend by' } expression$
<i>acDescend</i>	$:= \text{'descend by' } expression \mid \text{'descend to ground'}$
<i>expression</i>	$:= \text{'neg' } expression \mid expression \text{ op } expression \mid \text{'random' } range^? \mid \text{'point' } point \mid n \mid \text{'(}' expression \text{'')}$
<i>range</i>	$:= \text{'[}' expression \text{'', } expression \text{'']'}$
<i>point</i>	$:= \text{'(}' expression \text{'', } expression \text{'')}$
<i>op</i>	$:= \text{'+' } \mid \text{'-'} \mid \text{'*'}$

Figur 1. Syntaksen til AeroScript. Nye konstruksjoner for denne innleveringen er markert med farget bakgrunn. $\mathbb{Q}^p$ betyr positive rasjonalle tall. Stjernen ($\odot^*$) angir en sekvens med 0 eller flere forekomster, spørsmålsteget ($\odot^?$) angir et valg (null eller én forekomst), og plussteget ($\odot^+$) angir en sekvens med én eller flere forekomster. NAME er mengden av alle mulige identifikatorer.

Intuisjon: Et AeroScript-program består av én eller flere *eksekveringer* (*executions*, *oppdrag*, *kommandoer*), som har enkelte likhetstrekk med metoder i Java. Hver eksekvering (*execution*) er definert av en identifikator og handlinger omsluttet av klammeparenteser. En eksekvering kan ha en valgfri pil foran seg og kan avsluttes med en valgfri pil og navnet på en eksekvering. Hvis en eksekvering har en pil foran seg, skal denne eksekveringen automatisk startes som en del av programmets kjøreblokk. Hvis en eksekvering avsluttes med en pil til en annen eksekvering, skal programflyten gå over til denne og utføre de tilhørende instruksjonene før den fortsetter med resten av hovedblokken. Følgende er noen eksempler på eksekveringer:

```

-> MissionControl {} -> TakeOffLand

TakeOffLand {
    ascend by 20
    move to point (50, 50)
} -> ScanArea

ScanArea {
    move to point (100, 100) at speed 10
} -> EndMission

EndMission { return to base }

-> RandomTour {} -> TakeOff

Explore {
    move to point (random[0, 100], random[0, 100]) at speed 10
} -> ReturnToBase

TakeOff {
    ascend by 20
    move to point (random[0, 100], random[0, 100])
} -> Explore

ReturnToBase { return to base }

```

Når et AeroScript-program kjøres, skal programmet ditt lese innholdet i filen `program.aero` (som finnes under `src/test/resources/`) og parse det til et AST. Programmet skal deretter kjøre AeroScript-programmet ved å traversere AST-et og utføre setningene.

Merk: de definerte eksikveringene (f.eks. `TakeOffLand`, `TakeOff` osv.) **skal ikke** kjøres når de ikke er kalt.

3 Semantikken til AeroScript

Syntaksen og semantikken for uttrykk i AeroScript ble gjennomgått (og forhåpentligvis implementert) i forrige obligatoriske oppgave. Som en del av utdelingen til denne oppgaven fikk du kode som implementerer evaluering av uttrykk. Denne kan avvike noe fra det du selv implementerte tidligere. Det er helt greit. Du kan enten fortsette å arbeide videre med din egen kode fra forrige oppgave eller bruke den utdelte koden. Uansett hva du velger, bør koden for å parse og evaluere uttrykk nå være lett å forstå.

I denne oppgaven skal vi fokusere på hvordan eksekveringer evalueres. Hver eksekvering består av en sekvens av handlinger. Når en eksekvering startes, utføres handlingene i rekkefølge. Programmet går dermed videre til den neste setningen i sekvensen. Programmet må til enhver tid holde oversikt over gjeldende

- posisjon,
- høyde,
- tilbakelagt avstand, og
- batterinivå.

Hvis batterinivået synker til null, skal programmet utløse et unntak.

Hver handling skal skrives ut på kommandolinjen som en logg, og vil påvirke variablene i programmet (enten posisjonen som en koordinat i X og Y, eller høyden ved opp- eller nedstigning) og redusere batterinivået.

3.1 Detaljert semantikk

Eksekveringer og overganger I AeroScript er **eksekveringer (executions)** ulike tilstander eller scenarioer som en drone kan operere i—for eksempel avgang, flyging eller landing—og de er definert i grammatikken av `execution`. I Oblig 2 ligner en eksekvering på en metode, og en ny metode kan bare kalles etter at metodekroppen til den aktive metoden er utført.

Handling	Beskrivelse	Batterikostnad
<i>acDock</i>	Avsluttende handling som ikke tar noen argumenter. Programmet skal skrive ut tilbakelagt avstand og batterinivå før det avsluttes.	altitude + (time * 0.1) + (speed * 1). En vellykket retur til basen gjenoppretter batterinivået til startverdien.
<i>acMove</i>	Flytt til et punkt eller med n meter. Hvis det er et punkt, må vi beregne den kartesiske forskjellen (i punktets to dimensjoner; ikke ta hensyn til høyden). Ellers flytter vi n meter i enten x - eller y -retningen (velg én).	(distance * 0.7 if point, distance * 0.5 otherwise) + (time * 0.1) + (speed * 1)
<i>acTurn</i>	Sving n grader.	angle * 0.3 + (time * 0.1) + (speed * 1)
<i>acAscend</i>	Stig n meter.	distance * 0.6 + (time * 0.1) + (speed * 1)
<i>acDescend</i>	Gå ned til bakken (altitude blir 0).	altitude * 0.2 + (time * 0.1) + (speed * 1)

Tabell 1. Tabell over kostnader for batteriforbruk. Merk at tid og hastighet kan mangle helt — sett verdien til 0 dersom parameteren ikke er angitt.

Hver eksekvering inneholder et sett med operasjoner som utføres når dronen er i denne tilstanden. En *TakeOff*-eksekvering kan for eksempel inneholde instruksjoner om å starte rotorene og stige til en bestemt høyde. Merk at et program som skal få dronen til å *gjøre* noe, må ha minst én eksekvering med en innledende overgangspil—*start-eksekveringen*.

Handler og kommandoer Handler er byggsteinene for det dronen faktisk kan gjøre. Dette omfatter å bevege seg til en bestemt 2-dimensjonal koordinat, stige eller synke og svinge. Hver handling i *AeroScript* tilsvarer direkte en kommando som sendes til dronens kontrollsistem.

- Kommandoer som `move to point (x, y)` ber dronen fly til et bestemt sted i det todimensjonale planet i gjeldende høyde. Semantikken til denne kommandoen innebærer å beregne flyruten og justere rotorene for å navigere til disse koordinatene.

Uttrykk og beregninger Uttrykk og beregninger ble gjennomgått i forrige oppgave (Oblig 1); det er ikke behov for noen endringer her!

Batteriforbruk Hver handling (bortsett fra utskrift til kommandolinjen) vil redusere batterinivået som angitt i [Tabell 1](#).

Tilbakelagt avstand Hver handling skal oppdatere tilbakelagt avstand som beskrevet i [Tabell 1](#).

3.2 Slik henger alt sammen

Etter at kildekoden er parset til et AST, skal interpreten traversere det og utføre instruksjonene. Mer spesifikt skal den identifisere start-eksekveringen, utføre handlinger, og evaluere uttrykk. Interpreten kontrollerer fortløpende hvilken eksekvering dronen befinner seg i, og hvilke handlinger som skal utføres basert på gjeldende eksekvering.

Merk: Videre skal de definerte eksekveringene (f.eks. `TakeOffLand`, `TakeOff` osv.) **ikke** kjøres når de ikke er kalt (f.eks. kjøres `TakeOffLand` etter at kroppen til `MissionControl` er fullført.) Eksekveringer som starter med ‘`->`’ skal kalles når de nås. Executions som ikke starter med ‘`->`’ skal bare kalles når de kalles fra en annen eksekvering.

4 Din oppgave

Oppgaven inneholder en mal for implementasjonen, tilsvarende del 1. Du skal gjøre følgende:

- Utvide AST-et slik at det dekker hele syntaksen til *AeroScript*.
- Utvide lekseren og parseren slik at de dekker språket som er gitt i [Figur 1](#), ved hjelp av ANTLR.

- Utvid interpreten slik at den dekker språket som er gitt i [Figur 1](#).
- Vi har gitt noen JUnit-tester. **Diskuter testdekningen**, og lag egne tester. Pass særlig på at unntak kastes korrekt.
- Skriv en rapport (2–3 sider) der du diskuterer utfordringene du møtte og valgene du tok under implementasjonen. Sørg for å lese rapportveiledningen på [emnesiden](#) på forhånd. Gjelder ikke ved samretting.

5 Implementasjonstips

- Når du legger elementer til heap-en, skal du bruke disse variablene
 - `altitude` (settes først til 0)
 - `battery level` (startverdien kan oppgis av brukeren på kommandolinjen)
 - `distance travelled` (settes først til 0)
 - `initial execution` (settes først til null)
- Når du implementerer tolkningen av executions:
 - Tenk over hva som skal skje når dronen går tom for batteri...
 - sørg for at koden din kan kjøre rekursive eksekveringer (helt til de går tomme for batteri!)

6 Hvordan og hva skal leveres

Ved samretting før fristen bestemmer retter i samråd med gruppa hva som skal leveres i Devilry for å få godkjent. Se forøvrig [prosjektsida](#).

Du skal levere en zip-fil som inneholder prosjektet. Sørg for å rydde bort unødvendige filer (kjør `./gradlew clean`) før du lager zip-filen. Hvis zip-filen er svært stor, har du sannsynligvis ikke ryddet.

Ved tradisjonell retting med skriftlig tilbakemelding skal rapporten inkluderes i zip-filen.