# תכנות מתקדם 1

אבני דרך 1,2 בפרויקט

ד"ר אליהו חלסצ'י
Eliahu.khalastchi.biu@gmali.com

## רקע

פרויקט זה בקורס תכנות מתקדם הוא חלון הראווה שלכם כשתרצו להציג את הניסיון התכנותי שצברתם. פרויקט זה מכיל את האלמנטים הבאים:

- שימוש בתבניות עיצוב וארכיטקטורה
- תקשורת וארכיטקטורת שרת-לקוח
- שימוש במבני נתונים ובמסד נתונים
- הזרמת נתונים (קבצים ותקשורת)
- השוואה, בחירה והטמעה של אלגוריתמים בתוך המערכת שניצור
- תכנות מקבילי באמצעות ת'רדים
- תכנות מוכוון אירועים, אפליקציית desktop עם GUI
- תכנות מוכוון אירועים, אפליקציית Web בסגנון REST
- אפליקציית מובייל (אנדרואיד)

בסמסטר זה עליכם להגיש 2 אבני דרך:

1. מפרש קוד המאפשר שליטה מרחוק בסימולטור טיסה
2. מימוש של מספר אלגוריתמי חיפוש, השוואה ביניהם מי הכי מוצלח, והטמעת המנצח כפותר הבעיות בצד השרת. כך נטמיע את צורת העבודה המלאה של בוגר מדעי המחשב.

בהצלחה!

## אבן דרך 1 – מפרש קוד (interpreter) השולט מרחוק במל"ט

**היכרות עם סימולטור הטיסה**

ברצוננו לכתוב מפרש לקוד שליטה במל"ט (מטוס ללא טייס). המטוס שלנו יטוס במרחב הווירטואלי של סימולטור הטיסה FlightGear. את סימולטור הטיסה תוכלו להוריד מ http://home.flightgear.org/ .

בין היתר, סימולטור זה מהווה גם שרת שאפשר להתחבר אליו כלקוח (ולהיפך). כך נוכל בקלות לשלוף מידע אודות הפרמטרים השונים של הטיסה בזמן אמת ואף להזריק לו פקודות שינהגו את המטוס.

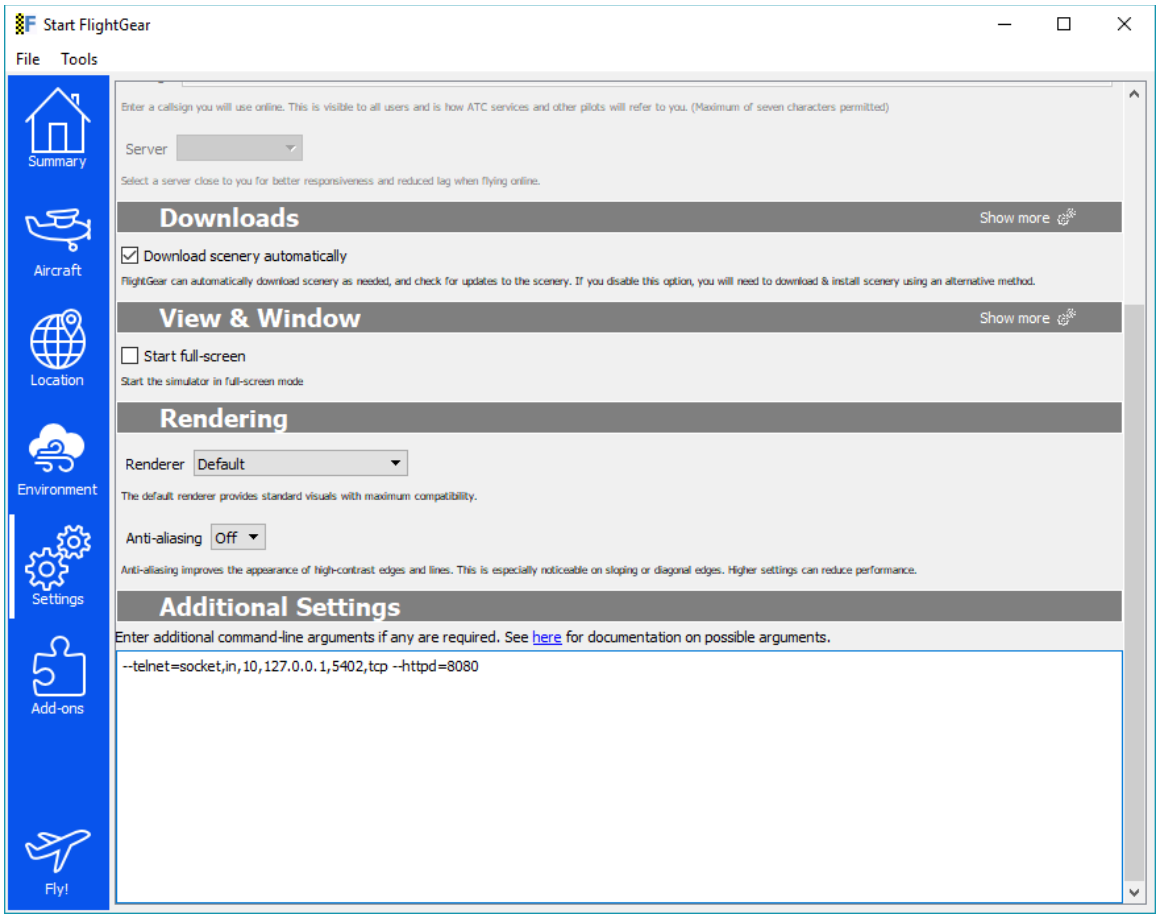
במסך הפתיחה ניתן לגשת ל settings ולהוסיף הגדרות שבד"כ נכתבות ב command line בסעיף של Additional Settings (ראו צילום מסך בהמשך).

למשל ההגדרה --telnet=socket,in,10,127.0.0.1,5402,tcp

אומרת לסימולטור לפתוח ברקע שרת שניתן להתחבר אליו באמצעות כל telnet client. השרת מבוסס על socket, והוא נועד לקרוא מידע שיגיע מהלקוח (in) בקצב של 10 פעמים בשנייה, ב local host כלומר באותו המחשב (ip 127.0.0.1) על port 5402 מעל פרוטוקול tcp/ip.

למשל ההגדרה –httpd=8080 תפתח web server על פורט 8080.

הריצו את הסימולטור עם ההגדרות לעיל (Fly!).




*הערה:* לבעלי כרטיס גרפי חלש – תוכלו בתפריט האפשרויות שמופיע בהתחלה להוריד רזולוציה ולבטל מאפיינים נוספים כגון עננים \ תנועה על הכביש וכדומה.

*הערה:* השעה בסימולטור אמתית, כך שסביר להניח שאם אתם פותחים את המשחק באור יום ומסלול הטיסה נמצא איפשהו בהוואי למשל, אז יהיה שם חושך. תוכלו לשונות את ההגדרות בתפריט environment את ה time of day ל noon.

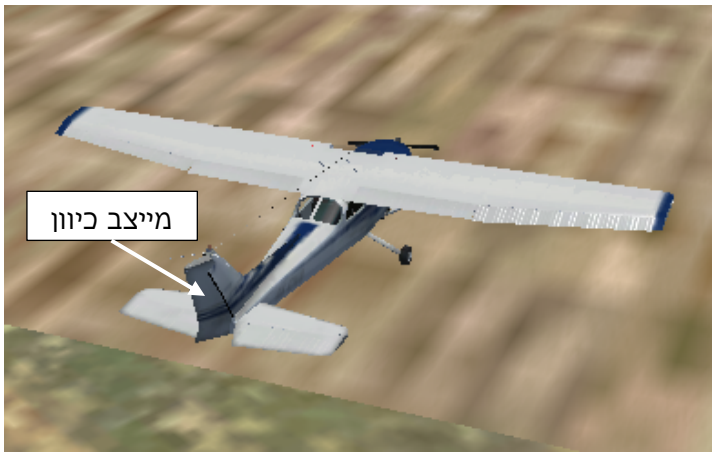
לאחר שהסימולטור פעל, פיתחו את הדפדפן בכתובת http://localhost:8080 ותוכלו לראות את האפליקציה ה web-ית שבאה עם הסימולטור.

כעת, תפתחו telnet client בשורת הפקודה, על local host ופורט 5402 (בהתאם להגדרות שראינו בסימולטור)

הערה: בלינוקס ניתן ישר לפתוח מהטרמינל, בחלונות יש לוודא שהתקנתם telnet client, מי שלא התקין וצריך עזרה יכול לחפש בגוגל "install telnet client windows 10 command line " ולמצוא מדריכים.

כאמור, ב CMD של חלונות הקלידו telnet 127.0.0.1 5402 ובעצם התחברתם כלקוח לשרת שפתח הסימולטור על המחשב שלכם. הממשק הזה נוח מאד מכיוון שהוא בנוי בצורה של file system (קבצים ותיקיות). כתבו ls כדי לראות את "התיקיות והקבצים" במיקום הנוכחי שלכם. תוכלו להיכנס לתיקיה באמצעות הפקודה cd (שזה change directory) למשל cd controls. תטיילו קצת בין אינספור ההגדרות השונות כדי לקבל תחושה מה יש שם.

כעת הביטו בו זמנית בסימולטור וב telnet. בסימולטור אתם יכולים לשנות זוויות צפייה ע"י לחיצה על V ואף משחק עם העכבר. הביטו על המטוס ממבט אחורי. כעת בואו נזין פקודה להזזת מייצב הכיוון של המטוס (rudder):




לא משנה היכן אתם ב telnet, כתבו:

set controls/flight/rudder 1

ראו בסימולטור כיצד מייצב הכיוון זז עד הסוף ימינה.

באופן דומה כתבו

set controls/flight/rudder -1

וראו כיצד מייצב הכיוון של המטוס זז עד הסוף שמאלה. כל ערך בין 1- ל 1 יסובב את מייצב הכיוון בהתאמה.

חוץ מלתת פקודות להגאים של המטוס, ניתן גם לדגום את הערכים השונים שנמדדו ע"י מכשירי הטיסה, כמו כיוון, מהירות, גובה וכו'. כתבו ב telnet את השורה הבאה:

```
get /instrumentation/altimeter/indicated-altitude-ft
```

ניתן לראות בתגובת השרת את הערך של הגובה הנוכחי כפי שנמדד במכשיר טיסה שנקרא `altimeter`.

אבל, את הערכים של הטיסה אנו נדגום בצורה מחוכמת יותר. יחד עם ההפעלה של הסימולטור נוסיף את ההגדרה:

```
--generic=socket,out,10,127.0.0.1,5400,tcp,generic_small
```

משמעותה היא שהפעם הסימולטור יתחבר כלקוח לשרת (שאנו נבנה) באמצעות socket, לצורך פלט (out) בתדירות של 10 פעמים בשנייה, על ה local host בפורט 5400 מעל tcp/ip. הערכים שנדגום מוגדרים בקובץ שנקרא generic_small.xml המצורף כנספח לפרויקט. את הקובץ הזה עליכם לשתול במיקום בו התקנתם את FlightGear בתיקייה data/protocol. פתחו את קובץ ה XML כדי להתרשם אלו נתונים נדגום.

עם הגדרה זו נצטרך לפתוח את השרת שלנו **לפני** שנפתח את הסימולטור כדי שהוא יוכל להתחבר אלינו כלקוח. הסימולטור ישלח 10 פעמים בשנייה את הערכים שדגם מופרדים בפסיק (בדיוק כמו ב CSV) ולפי הסדר שהוגדרו ב XML. בהמשך אבן הדרך תכתבו שרת קטנטן שיאזין לערכים האלו.

מי שרוצה לקבל עוד קצת רקע לגבי שליטה במטוס יוכל לקרוא (בוויקפדיה למשל) אודות

- ההגאים: aileron, elevators, rudder (מייצב כיוון, מייצב גובה, מאזנות בהתאמה)
- וכיצד הם משפיעים בעת טיסה על ה roll, pitch, yaw בהתאמה.
  - ובעברית סבסוב, עלרוד, גלגול בהתאמה.

דרך התפריט של המטוס (Cessna בד"כ הוא המטוס הדיפולטיבי) תוכלו להפעיל את המטוס ע"י autostart או לבצע הסדרה של פעולות בצורה ידנית (אם אתם ממש רוצים).

## מפרש קוד לשליטה בטיסה

כאמור, ברצוננו לכתוב מפרש לשפת תכנות חדשה שמטרתה להטיס את המטוס שבסימולטור. נתחיל מלהגדיר קוד לדוגמה שמטרתו לגרום למטוס להמריא בצורה ישרה.

קוד לדוגמה:

```
openDataServer(5400)                        // blocking call
connectControlClient("127.0.0.1",5402)      // blocking call
var breaks -> sim("/controls/flight/speedbreak")
var throttle -> sim("/controls/engines/current-engine/throttle")
var heading <- sim("/instrumentation/heading-indicator/offset-deg")
var airspeed<-sim("/instrumentation/airspeed-indicator/indicated-speed-kt")
var roll <- sim("/instrumentation/attitude-indicator/indicated-roll-deg")
var pitch <- sim("/instrumentation/attitude-indicator/internal-pitch-deg")
var rudder <- sim("/controls/flight/rudder")
var aileron <- sim("/controls/flight/aileron")
var elevator <- sim("/controls/flight/elevator")
var alt <- sim("/instrumentation/altimeter/indicated-altitude-ft")
breaks = 0
throttle = 1
var h0 = heading
while alt < 1000 {
   rudder = (h0 – heading)/20
   aileron = - roll / 70
   elevator = pitch / 50
   Print(alt)
   Sleep(250)
}
Print("done")
```

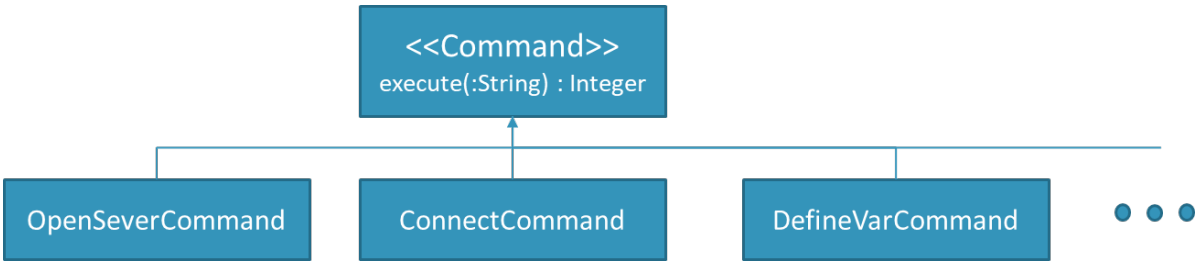
כיצד נפרש קוד כזה?

(1)תחילה נרצה לכתוב **פונקציה שתשמש אותנו כ lexer**. היא תקרא קובץ ובו קוד, ותחזיר הקוד מפורק למערך של מחרוזות.

openDataServer, 5400, connectControlClient, “127.0.0.1”, 5402, var, breaks, ->, sim, ...

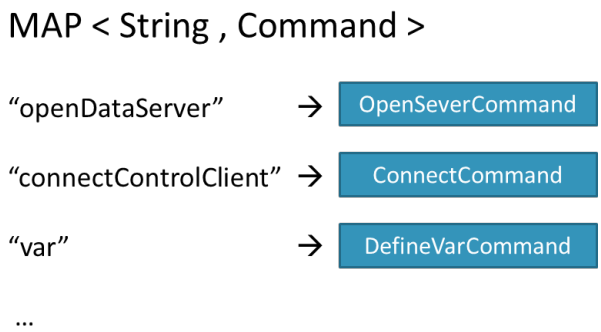
כעת נשתמש ב Command Pattern ובמפה כדי לקשר בין מחרוזת במערך לבין אובייקט Command שיבצע את מה שצריך.

ה Command Pattern:



התבנית אומרת לנו להגדיר ממשק בשם Command עם מתודה ק execute()`. כל פקודה במערכת שלנו (אצלנו זה פקודות שיש לפרש) תהיה מחלקה מהסוג של Command. כך Command פולימורפי יכול להיות פקודה ספציפית כלשהי, ונפעיל את כולן באותו האופן. לצרכים שלנו execute יכולה לקבל כפרמטר מערך של מחרוזות שיש לפרש.

הטריק התכנותי שנבצע הוא שנכניס את כל הפקודות למפה מבוססת hash כך שהמפתח הוא מחרוזת, והערך הוא אובייקט ספציפי מסוג Command. כך בהינתן המחרוזת נוכל לשלוף מידית את ה Command שצריך לפעול.




פונקציה בשם parser תעבור (כמעט) על כל מחרוזת במערך ותשלוף באמצעות המחזורת אובייקט Command ותקרא ל execute() שלו שיריץ בתורו את מה שצריך להריץ.

עבור הצרכים שלנו אפשרי ש execute תחזיר int שמהווה את כמות הדילוגים שיש לבצע במערך המחרוזות כדי לבצע את הפקודה הבאה. האם execute צריכה לקבל פרמטרים? לשיקולכם.

פסאודו-קוד לדוגמה להפעלת ה parser:

```
While index < length(array) //lexer returns array of commands

      Command c =  map.get(array[index]);

      if(c!=Null)

            index += c.execute()
```

כך למשל המחרוזת openDataServer תוביל להחזרה של האובייקט OpenServerCommand שבתורו ישלוף את האיבר הבא ממערך המחרוזות ("5400"), יבצע את הפקודה הנדרשת, ויחזיר "2" כי היה לו רק פרמטר אחד לאסוף. כך הפעולה הבאה של ה parser תשלוף את ConnectCommand וכך הלאה עד לסוף התוכנית.

זה שלב טוב להסביר את משמעות הפקודות שבקוד לדוגמה לעיל.

שורה 1: **openDataServer** היא פונקציה ש:

- פותחת ת'רד ברקע שמריץ **שרת** שמאזין על פורט 5400 (הפרמטר) וקורא שורה שורה
  - הקצב נשלט ע"י הסימולטור שיתחבר אלינו כלקוח ויכתוב לנו שורה שורה
  - (הרי כל קריאה בצד השרת היא blocking call)
- בכל שורה נקבל מחרוזת של ערכים מופרדים ע"י פסיק. אלו הם הערכים של מכשירי הטיסה כגון גובה, מהירות, כיון וכדומה. אלו הערכים המוגדרים בקובץ generic_small.xml
- את הערכים הנדגמים יש לאכסן במבנה נתונים שבאמצעותו נוכל לשלוף ב O(1) את הערך העדכני של משתנה כלשהו שבחרנו.

- **כמו כן, השורה הזו תגרום למפרש להמתין עד אשר הגיע הפידבק הראשון מהסימולטור. כך נדע שהוא כבר פתוח ואפשר להתחבר אליו כלקוח בשורה 2.**

שורה 2: **ConnectControlClient** היא פונקציה שבאמצעותה נתחבר כ**לקוח** לסימולטור שנמצא ב 127.0.0.1 ומאזין על פורט 5402. נעשה זאת גם בת'רד נפרד. בשורות הבאות נשתמש בלקוח כדי לשלוח הוראות טיסה לסימולטור.

בשורות 3-12 (**var**) אנו מגדירים את המשתנים שאיתם נעבוד במהלך התוכנית.

- למשל var breaks מצהיר על קיומו של משתנה בשם breaks.
- לעומת זאת sim היא פונקציה שמחזירה "מצביע" למשתנה שקיים בסימולטור.
- בניגוד לאופרטור ההשמה ("="), החצים מגדירים את כיוון ה**כריכה** שבין משתנה בתוכנית לבין משתנה בסימולטור. דהיינו, כאשר האחד משתנה אז גם השני ישתנה.
- כך למשל בשורה 4 הגדרנו שהמשתנה throttle בתוכנית ישפיע על ערכו של המשתנה `/controls/engines/current-engine/throttle` בסימולטור. ובשורה 14 כשנפרש את `throttle = 1`, מלבד ההשמה של 1 למשתנה `throttle` בתוכנית נבצע ע"י הלקוח שלנו שליחה של המחרוזת "`set /controls/engines/current-engine/throttle 1`" וגרמנו למצערת להיפתח עד הסוף (כוח מלא למנוע כדי שהמטוס יתחיל לנוע)
- דוגמה לכריכה הפוכה ניתן למצוא למשל בשורה 12. המשתנה `alt` בתוכנית כל הזמן מתעדכן מערכו של "`/instrumentation/altimeter/indicated-altitude-ft`" בסימולטור. העדכון מתרחש ברקע ע"י ה `DataServer` שיצרנו בשורה 1.
- כך הלולאה בשורה 16 יכולה להסתיים למרות שלא עדכנו את המשתנה `alt` בתוכנית. אלא, יצרנו סדרה של פעולות שבתורן גרמו למטוס להמריא וכתוצאה מכך לשנות את הגובה.

בתוך הלולאה אנו מעדכנים את הערכים

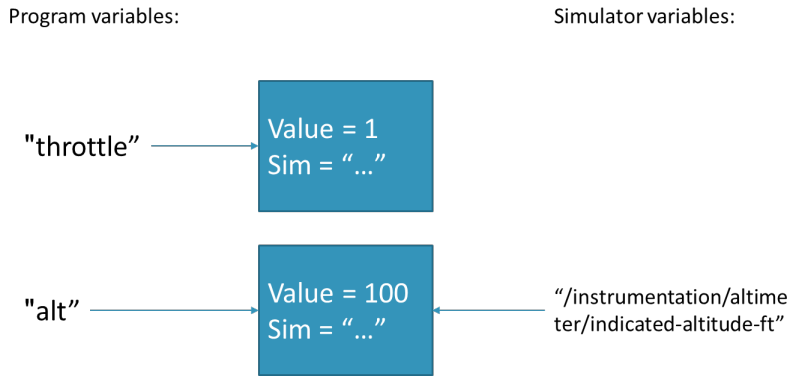
- של ה `rudder` כפונקציה של הכיוון
- של ה `aileron` כפונקציה של הגלגול
- ושל ה `elevator` כפונקציה של ה `pitch`

*הערה:* ערכים אלו ניתנו עבור המטוס הדיפולטיבי בסימולטור – `Cessna C172p` ובהחלט יכולים להיות שינויים בין גרסאות שונות כדי שהמטוס באמת יתייצב בהמראה. תשחקו עם הערכים הללו עד שהמטוס ימריא ישר.

כמו כן, בכל איטרציה אנו מדפיסים את הערך של `alt` וממתינים 250 מילישניות לפני המעבר לאיטרציה הבאה. בסוף הרוטינה אנו כותבים done.

**איך נתמודד עם משתנים וכריכה?**

אנו צריכים **`symbol table`** – מפה שתקשר בין שם המשתנה לבין אובייקט שמכיל את כל האינפורמציה שנצטרך אודות המשתנה. למשל:



כך למשל כשנפרש var throttle אז ניצור במפה את הקשר בין המחרוזת "throttle" לאובייקט המשתנה. כשנפרש את החץ ל sim אז ניגש לאובייקט ונמלא את השדה sim (וגם את כיוון הכריכה באיזה אופן שתבחרו). כשנפרש throttle = 1 אז ניגש באמצעות המפה לאובייקט של throttle ונזין ל value את הערך 1. אך מכיוון שכיוון הכריכה מורה לנו לעדכן את הסימולטור אז נבצע באמצעות ה controlClient שליחה של פקודת set למחרוזת הנתונה בשדה sim עם הערך 1:

"set /controls/engines/current-engine/throttle 1”

וכך הסימולטור יזין למצערת את הערך 1.

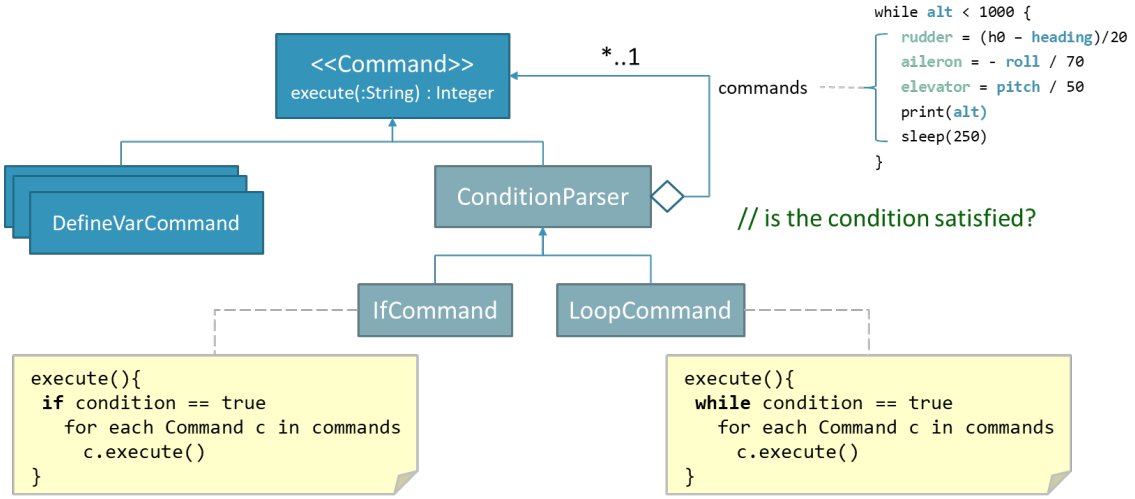
בכיוון השני נצטרך גם מפה משמות המשתנים בסימולטור לאובייקטים הרלוונטים. ה DataServer לשנו לאחר קריאה של כל שורה ישתמש במפה כדי לעדכן את ערכם של משתנים אלו. כך למשל הערך של ה altimeter יזן לשדה value של האובייקט המתאים, ובתוכנית כשנבקש את ערכו של המשתנה alt אז נחזיר אותו באמצעות המפה ( לדוג' symTable.get(“alt”).getValue();)

האם צריך symbol table לכל סקופ בקוד (קוד המוקף בסוגריים מסולסלים) או שנוסיף שדה scope לאובייקט? חישבו על התשובה והחליטו לבד מה יותר נכון.

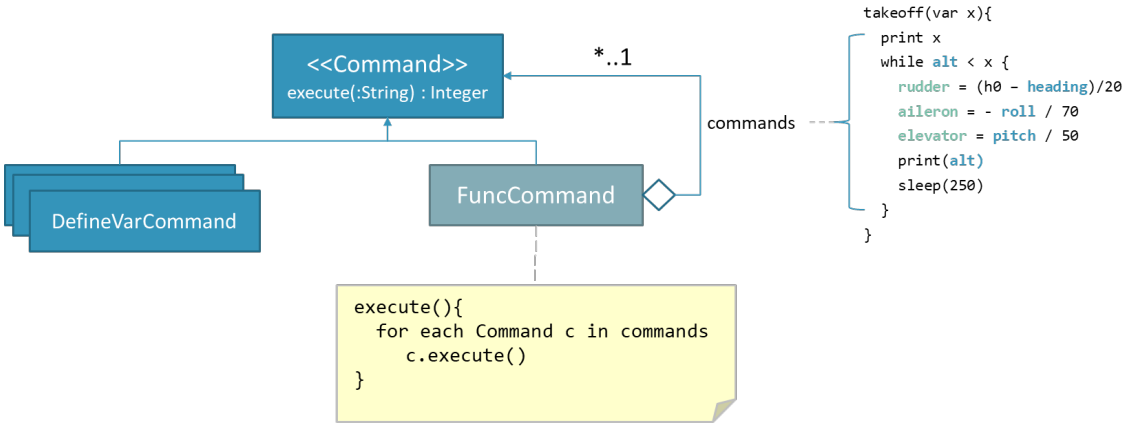
**ומה לגבי סקופים של קוד? כמו למשל הקוד במתרחש בשורות 16-22 בלולאת ה while?**

זה היופי שתבנית ה Command – אין בעיה ליצור Command מורכב (Composite) שמכיל רשימה של אובייקטי Command. קריאה ל execute שלו תוביל לקריאה ל execute של כל מי שברשימה שלו. כמובן שגם הם יכולים להיות מורכבים ובתורם לקרוא לכל מי שברשימה שלהם... ראו תרשימים:

משפט IF או לולאה:




או פונקציה:




**תנאים:**

בשורה 16 המפתח הוא `while` לאחר מכן נצפה לסדרה של תנאים עד להופעת הסימן "}". לשם הקלה עליכם לצפות רק לתנאי אחד. אבל מי שרוצה להשקיע שיכניס גם סוגרים עגולים ופעולות AND ו OR.

התנאי יכול להיות מורכב מהאופרטורים <,>,<=,>=, ==,=! במשמעות הרגיל שאתם כבר מכירים.

**ביטויים:**

ומה לגבי פיענוח ביטויים מתמטיים מורכבים כמו בשורות 17-19?

או למשל אם הינו כותבים `OpenDataServer(5000+4*100)` במקום `5400`?

עם זה אתם כבר יודעים להתמודד כפי שביצעתם בתרגיל הבית הראשון. אם המחרוזת מכילה גרשיים ("`text`") אז מדובר במחרוזת, אחרת מדובר בביטוי...

**`Main`:**

אתם מגישים `main` משלכם. ה`main` מקבלת קלט בתוך ה`argv` שמייצג שם של קובץ.

```
./a.out file_name
```

את הקובץ אתם שולחים ל`lexer` כך שיפרסר את הקובץ שורה שורה לתוך מערך, ואת הפלט שלו לשלוח ל`parser`.

כך שהקוד שהמשתמש כתב מתפרש וגורם לפעולות שונות בסימולטור.

בתום הפעולות צריך לוודא לשחרר את כל הזכרון ולסגור את הקבצים וסוקטים שצריך.

בהצלחה!

## Submission

You will submit all of your code in a zip file.

We will compile with the usual command, except with:

```
g++ -std=c++14 *.cpp -Wall -Wextra -Wshadow -Wnon-virtual-dtor -pedantic -o a.out -pthread
```

1. You must include a **details.txt** file in your zip with the following format:

Daniela

12344

```
Roi

2134235
```

- **Failure to include a details.txt with all the required information will lose you 5 points.**

```
2. README
      a. This file will describe how to compile your code, how to run it,
         where do files need to reside, everything that someone needs to
         know when wanting to run your code. (google how to write a
         README, theres no specific format we want, just for it to be
         descriptive).
      b. Put it in your github.
      c. Add a link to your github (bonus 5 points)
3. Add well written comments that document your code (5 points)
```

We will test your code with similar .txt files that we provided you with. The rest of the code is up to you, meaning you **must also provide** a main.cpp.