SCHOOL OF COMPUTING (SOC)

Diploma in Applied AI and Analytics

ST1507 DATA STRUCTURES AND ALGORITHMS (AI)

2025/26 SEMESTER 2
ASSIGNMENT TWO (CA2)

~ DASK Expressions Evaluator ~
(using parse trees)

Objective of Assignment

For this assignment you will have an opportunity to apply all that you have learnt with regards to data structures, algorithms, and object-oriented programming to develop an application that can solve special mathematical expressions by making use of *parse trees*.

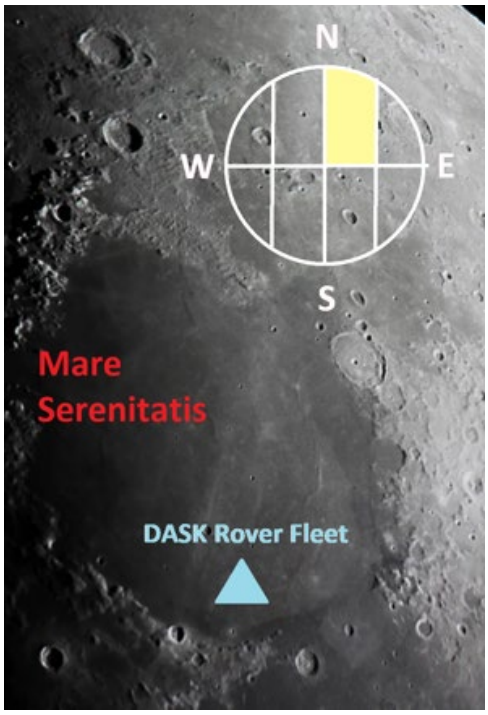
Instructions and Guidelines:

1. This is a group assignment (you will work in pairs, only one group of 3 would be allowed if there is an odd number of total students).
2. This assignment accounts for **40%** of your final grade.
3. The assignment comprises a group component (70%) and an individual component (30%).
4. The submission date is **Friday 6 February 1:00 pm**.
5. The development will be carried out in Python using Anaconda.
6. The demonstrations/interviews will be conducted during the DSAA lessons in week 17/18. You are expected to explain your code and program logic. Take note that the interview is compulsory.
7. **50% of marks** will be deducted for submission of assignment within **ONE** calendar day after the deadline. **No marks shall** be awarded for assignments submitted **more than one day** after the deadline.

Warning: Plagiarism means passing off as one's own the ideas, works, writings, etc., which belong to another person. In accordance with this definition, you are committing plagiarism if you copy the work of another person and turning it in as your own, even if you would have the permission of that person.
Plagiarism is a serious offence, and if you are found to have committed, aided, and/or abetted the offence of plagiarism, disciplinary action will be taken against you. If you are guilty of plagiarism, you may fail all modules in the semester or even be liable for expulsion.

1. Background

ALOYD LUNAR DASK (acronym ALD) is a startup mining company with headquarters located in the business district of Heatherthorn County. ALD was set up by serial entrepreneur, and space explorer, Aloyd Dask. ALD is currently operating a fleet of robotic vehicles, called Dask Rovers, to survey the surface of the moon. Their mission is to look for rare metals. About a dozen of Dask Rovers are currently deployed at Mare Serenitas (Sea of Serenity, collegiality known as the right eye of 'the man in the moon'). The Dask Rovers, equipped with special sensors, are working together making systematic measurements of the moon's surface. Their findings are then broadcast back to ALD ground station in the form of special mathematical expressions called 'DASK Expressions'. Back on earth, ALD personnel will then proceed with the tedious task of evaluating and sorting the DASK Expressions to extract geological insights.



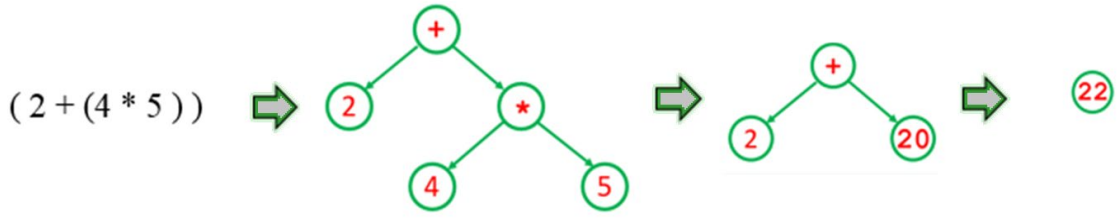
Location of DASK Rover Fleet on Lunar Surface.

2. Your job as team

As a team of AI programmers, you have been roped in to implement an application that can help ALOYD LUNAR DASK to automate the process of evaluating and sorting the DASK expressions. You will be required to make use of *parse trees* to interpret, visualize and solve DASK expressions.

3. What are parse trees?

Parse trees are special binary trees that can be used to represent and solve mathematical expressions. Below is an example of a parse tree that has been extracted from an expression, *(2+(4*5))*. The tree can be solved by starting at the leaves and then solve the sub-tree expressions first and progressively work yourself upwards until you reach the root of the tree (we are using the method of expression parsing as explained in topic 5 slides).



4. What are DASK Expressions?

DASK expressions are fully parenthesized expressions comprising variables and binary operators. The DASK binary operators are listed below.

DASK binary operators:

Operator	Description	Examples
+	*A regular addition.*	`A=(1+2) => 3`
-	*A regular subtraction.*	`A=(3-1) => 2`

*	*A regular multiplication.*	`A=(2*3) => 6`
		`B=(2*3.14) => 6.28`
		`C=(4*3.14) => 12.56`

/	*A regular division.*	`A=(1/4) => 0.25`
		`B=(3/4) => 0.75`
		`C=(3.14/2) => 1.57`

**	*A regular exponent, like* a^b	`A=(2**3) => 8` `B=(3**2) => 9`
++	*Addition of the summative of two operands.* $\sum a + \sum b$	`A=(3++4) => 16` **Take note:** $\sum 3 = 3+2+1=6$ $\sum 4 = 4+3+2+1=10$
//	*Division of the summative of two operands.* $\sum a \ / \ \sum b$	`A=(3//4) => 0.6`

Your application should be able to solve DASK expressions using parse trees. Here are some examples of DASK expressions:

```
a= (2+3)
bc= (a*3)
abc= (a+ (bc/2))
Mu=((Alpha+(Delta+(Pi*(Epsilon*(Kappa/Sigma)))))/2)
Zeta=(3//(4++5))
```

*Take note that variables names in DASK expression may only contain letters, so no space characters, no digits and no special characters. The variable names are case sensitive.

5. Overview of the basic features of the system

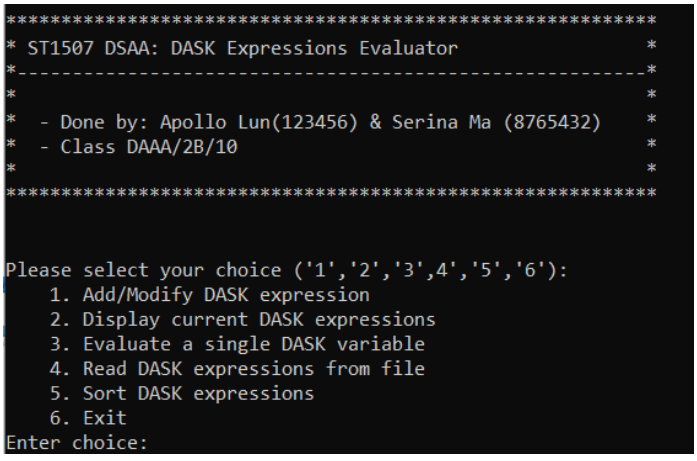
Your group is tasked to implement an application that can evaluate DASK expressions (using parse trees) and then sort them.

Your application should support the following features:

- The user should be able to add individual DASK expressions by typing them on the screen or reading a whole series of them directly from an input file.
- The user should be able to evaluate and print the parse tree for a DASK variable at any one time.
- The user should be able to evaluate and print all the currently loaded DASK expressions at any one time.
- The user should be able to modify individual DASK expressions at any one time.
- The user should be able to sort the currently loaded DASK expressions by their value (in descending order) followed by alphabetically order by variable name. The sorted results will then be written back to an output file.

6. Selection menu

The user starts the application in the Anaconda Prompt by typing **python main.py**. After that the user is presented with a menu as is shown below, allowing the user to choose from 6 options (1','2','3','4','5','6').



- Take note you must follow the above format, please ensure you display the names and IDs of all group members, as well as the correct class.

- The user will be able to repeatedly select options from the menu, until he/she selects option 6 after which the application will terminate.

```
Please select your choice ('1','2','3',4','5','6'):
    1. Add/Modify DASK expression
    2. Display current DASK expressions
    3. Evaluate a single DASK variable
    4. Read DASK expressions from file
    5. Sort DASK expressions
    6. Exit
Enter choice: 6

Bye, thanks for using ST1507 DSAA: DASK Expressions Evaluator
```

7. Adding new DASK expressions

- When the user selects option 1, he/she will be prompted to enter a fully parenthesized DASK expression (the user is thereby also shown an example).

```
Please select your choice ('1','2','3',4','5','6'):
    1. Add/Modify DASK expression
    2. Display current DASK expressions
    3. Evaluate a single DASK variable
    4. Read DASK expressions from file
    5. Sort DASK expressions
    6. Exit
Enter choice: 1
Enter the DASK expression you want to add/modify:
For example, a=(1+2)
```

The user may for instance enter:
`a=(2+(4*5))`

```
Enter the DASK expression you want to add/modify:
For example, a=(1+2)
a=(2+(4*5))

Press enter key, to continue....
```

- After the user presses the enter key, the application will store the DASK expression for later usage.

- The user may then proceed by adding more expressions.

For instance:
`b=(a*5)`

```
Enter the DASK expression you want to add/modify:
For example, a=(1+2)
b=(a*5)

Press enter key, to continue....
```

The DASK expressions that we enter may make use of variables that already were introduced beforehand, or you may even use variables that have yet to be introduced.

For instance:
`c=(a+d)`

```
Enter the DASK expression you want to add/modify:
For example, a=(1+2)
c=(a+d)

Press enter key, to continue....
```

- To evaluate the statements that were just entered the user can press Option 2.

```
Please select your choice ('1','2','3',4','5','6'):
    1. Add/Modify DASK expression
    2. Display current DASK expressions
    3. Evaluate a single DASK variable
    4. Read DASK expressions from file
    5. Sort DASK expressions
    6. Exit
Enter choice: 2

CURRENT EXPRESSIONS:
********************
a=(2+(4*5))=> 22
b=(a*5)=> 110
c=(a+d)=> None


Press enter key, to continue....
```

- The expressions will all be evaluated using parse trees, and the resulting values will be displayed at the right-hand side.
- For those expressions that contain variables that have yet to be introduced, their value will be indicated as ‘None’.

*Take note, expressions are listed in alphabetical order, sorted by their variable names.
We may now introduce the variable ‘d’, that was previously used in one of the expressions.

For instance:
`d=(a*b)`

```
Enter the DASK expression you want to add/modify:
For example, a=(1+2)
d=(a*b)

Press enter key, to continue....
```

After selecting Option 2 you will notice that the expressions have been re-evaluated.

```
Enter choice: 2

CURRENT EXPRESSIONS:
********************
a=(2+(4*5))=> 22
b=(a*5)=> 110
c=(a+d)=> 2442
d=(a*b)=> 2420
```

8. Modifying existing DASK expressions

- The user may modify DASK expressions at any one time, by entering the variable name of an existing variable, followed by a new expression.

For instance, we could reassign a new expression to the existing variable ‘b’:
`b=(a*2)`

```
Enter choice: 1
Enter the DASK expression you want to add/modify:
For example, a=(1+2)
b=(a*2)

Press enter key, to continue....
```

After selecting Option 2 you will then notice that all the expressions have been re-evaluated.

```
Enter choice: 2

CURRENT EXPRESSIONS:
********************
a=(2+(4*5))=> 22
b=(a*2)=> 44
c=(a+d)=> 990
d=(a*b)=> 968
```

9. Evaluating a single variable

- The user can use Option 3 to evaluate and print a parse tree (printed in *in-order* format) of an individual variable.

For instance, let's say the following expressions were entered beforehand:

```
Alpha=(2+(4*5))
Pi=(Alpha*3)
Mango=((Alpha+(Delta+(Pi*(Beta*(Gamma/Sigma)))))/2)
```

Then the associated parse tree for variable 'Alpha' will be printed as follows:
Take note: the tree is printed rotated 90 degrees anticlockwise.

```
Enter choice: 3
Please enter the variable you want to evaluate:
Alpha

Expression Tree:
..5
.*
..4
+
.2
Value for variable "Alpha" is 22

Press enter key, to continue....
```

The associated parse tree for variable 'Pi' will be printed as follows:

```
Enter choice: 3
Please enter the variable you want to evaluate:
Pi

Expression Tree:
.3
*
.Alpha
Value for variable "Pi" is 66

Press enter key, to continue....
```

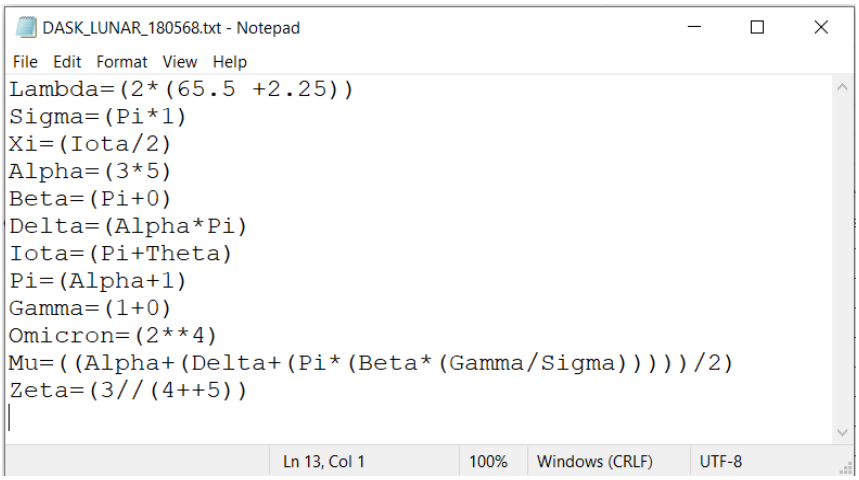
And the associated parse tree for variable ‘Mu’ will be printed as follows:

```
Enter choice: 3
Please enter the variable you want to evaluate:
Mu

Expression Tree:
.2
/
......Sigma
...../
......Gamma
....*
.....Beta
...*
....Pi
..+
...Delta
.+
..Alpha
Value for variable "Mu" is None

Press enter key, to continue....
```

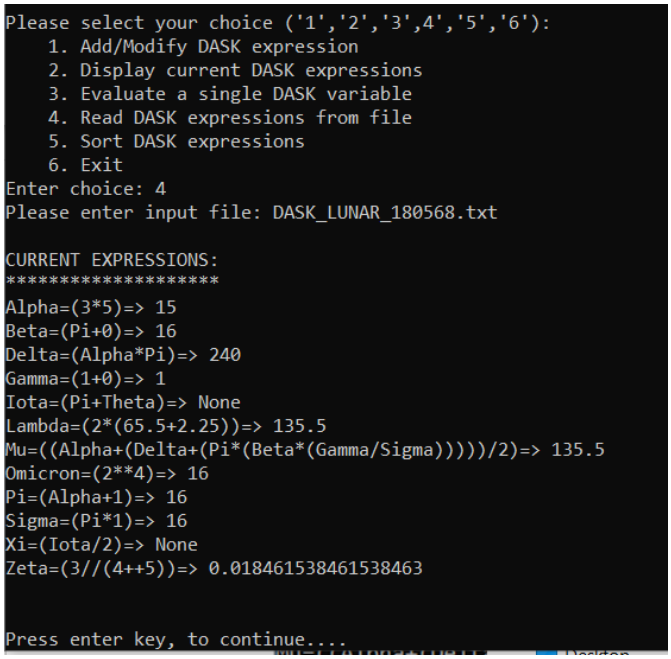
10. Reading Expressions from a file and sorting expressions

- When the user selects option 4, he/she will be prompted to enter an input file containing the DASK expressions to be added (take note, if there are already DASK expressions loaded in the application, then they will just be added to the existing ones, there will be no more than 50 expressions in the file).

Let's say we use this input file to read in:



Then the application will read and evaluate all the expressions from the file and followed by the display of the current expressions (in the same manner as we would do by selecting Option 2).



11. Sorting Expressions

- Once we have a series of DASK expressions loaded in the application, the user may sort them by selecting Option 5.

```
Please select your choice ('1','2','3',4','5','6'):
    1. Add/Modify DASK expression
    2. Display current DASK expressions
    3. Evaluate a single DASK variable
    4. Read DASK expressions from file
    5. Sort DASK expressions
    6. Exit
Enter choice: 5

Please enter output file:DASK_LUNAR_180568_SORTED.txt

>>>Sorting  of DASK expressions completed!

Press enter key, to continue....
```

In this case the assignment statements will be sorted and stored in the output file 'DASK_LUNAR_180568_SORTED.txt' as is shown below:

DASK_LUNAR_180568_SORTED.txt - Notepad

File Edit Format View Help

```
*** Expressions with value=> 240
Delta=(Alpha*Pi)

*** Expressions with value=> 135.5
Lambda=(2*(65.5+2.25))
Mu=((Alpha+(Delta+(Pi*(Beta*(Gamma/Sigma)))))/2)

*** Expressions with value=> 16
Beta=(Pi+0)
Omicron=(2**4)
Pi=(Alpha+1)
Sigma=(Pi*1)

*** Expressions with value=> 15
Alpha=(3*5)

*** Expressions with value=> 1
Gamma=(1+0)

*** Expressions with value=> 0.018461538461538463
Zeta=(3//(4++5))

*** Expressions with value=> None
Iota=(Pi+Theta)
Xi=(Iota/2)
```

Ln 25, Col 12 | 100% | Windows (CRLF) | UTF-8

* You are required to use the same format as is shown in the output file above.

12. Exiting the application

The user may repeatedly select Options 1 till 5. Option 6 is to exit the program.

(*) Take note that once your group adds the additional menu options for extra features (to be done by individual members), they must be added before Option 6. So, Option 6 (to Exit) will then be shifted to the last option (e.g., it will have to remain the last option).

Requirements for Group Component (70%):

- Your application will only need to work with fully parenthesized expressions. Do take note that the placement of the parentheses will dictate the order that the operators are being executed. So, for instance the following 3 DASK expressions will be evaluated to different values as is dictated by the placement of the parentheses.

```
A=((1+2)/(3*5))    =>0.2
B=(1+((2/3)*5))    =>4.33
C=(((1+2)/3)*5)    =>5.0
```

- Your application needs to support the following 7 binary DASK Expression operators:

+ , - , * , / , **, ++, //

- You will need to support both integer as well as float operands. You may assume all DASK operands are positive numbers.

- You are required to design and write the Python application using an Object-Oriented approach (OOP). You should thereby leverage on your knowledge that you have of encapsulation, inheritance, polymorphism etc.

- You may make use of Python's already built data structures, such as list, tuple, dictionary and set. However, you should refrain from using the classes from the collection library. Instead, you are required to write your own classes to support the various data structures that you may need. Of course, you may refer to the lecture slides and lab tasks and expand further on those classes that we had previously developed in the tutorial and lab sessions.

- To run the application there should be no need to install additional libraries, other than those that ship already with Anaconda.

- The user will be able to start the application from Anaconda Prompt as follows:

```
python main.py
```

- Your application should not have to rely on any connection to the Internet.
- The OOP classes that you develop must be placed in separate python files.
- The group will be requested to demonstrate the basic features of the application during the demonstration.
- Take note the group's demonstration should not exceed 15 minutes (including 5 minutes for Q&A).

Requirements for Individual Component (30%):

Each individual team member is required to implement two additional features that need to be added to the application as menu options. These two additional features will need to be presented during the final presentation.

- The additional features added must be integrated in the form of additional menu items in the application.
- The features will be graded on technical sophistication and usability.
- Take note, features within the same group must be different. So please check with your group members first before embarking on implementing the extra features.
- Each group member must submit a short PowerPoint deck of slides. Your PowerPoint slides must briefly describe what extra features you have implemented. You must include screen shots demonstrating your extra features in action. Please explain how the features work, and why they are useful. You are to include your Name, Class, and Group Number in the first slide.
- You must submit the PowerPoint Slides (converted to pdf) together with a compulsory Peer Feedback (template will be provided on Brightspace) as well as a duly filled in and signed Academic Integrity Form.

Final Deliverables

Your group's final deliverables must include:

(a) Group Report

A report (as pdf file) with a maximum **of 10** pages. This excludes cover page and the appendix with source listing and references. The report should contain:

a) Cover page with group number, names, ids, and class (your instructor will assign group numbers).
b) Description, and user guidelines, on how to operate your application (please include screen shots of your application in action).
c) Describe how you have made use of the Object-Oriented Programming (OOP) approach. You may elaborate on the classes that you have developed, and discuss issues such as encapsulation, function/operator overloading, polymorphism, inheritance etc. Include a class diagram that displays the relation between the various classes you have developed.
d) Discussion on the data structures and algorithms you have developed for your application. You may discuss issues such as the performance of the algorithms in terms of Big (O). Explain why you did develop certain data structures and explain why you deem these data structures suitable for the task(s) at hand. Include a table summarizing all the data structures that you have been using (those that you have developed and those already built in Python).
e) Include a summary of the challenges that the group has faced while developing the application (that should include both technical, as well as group-work challenges). Provide a summary of the key takeaways and learning achievements that you have obtained from this project.
f) Include a clear description of the roles and contributions of each member in the team. Clearly state what each member has been responsible for, and what programming work has been carried out by each member.
g) **All** your python **source code** must be included as an appendix at the end of your report. You must clearly indicate in the source code listings who wrote what code. You may also include in the appendix; those references from literature or internet that you may have consulted.

(b) Source Code

- You must submit all (*) the python files (py files) that make up your application. Ensure that the code is complete, and that it can run directly from the Anaconda Prompt by typing `python main.py`.

  (*) Take note, that includes the code for all the extra features that were coded by each team member as well.

Submission instructions

Group Submission:

- Group Leader must submit all the group deliverables (Source Code and Group Report) in the designated Brightspace Drop Box.

- Important, the source code must include all the extra features that were coded by the team members. (so, when we run the application, we can experience all the extra features that the team members have developed).

- You must submit it as one Zipped folder (RAR will not be accepted, only zip) whereby you label your submission as:

  CA2_GroupNumberClass.zip

  For example: *CA2_GR_10_DAAA_2B08.zip*

- Please ensure that you submit it by the stipulated deadline.

Individual Submission:

- Each individual Group Member is to submit his/her individual deliverables.

- Individual submission to include:

  - PowerPoint slides describing your two extra features (converted to pdf, maximum 8 slides)
  - Peer Feedback form
  - Academic Integrity Form (filled up & signed)

  Please ensure that you submit it in the designated Brightspace Drop Box for individual submissions.

- You must submit it as one Zipped folder (RAR will not be accepted, only zip) whereby you label your submission as:

  CA2_Final_ GroupNumberStudentNameClass.zip

  For example: *CA2_GR_10_JIMMY_TAN_DAAA_2B08.zip*

- Please ensure to submit it by the stipulated deadline.

Assessment Criteria

The group component of the assignment will be assessed based on the following criteria:

Assessment criteria (Group 70 %)	Marks awarded
GROUP COMPONENT (70 %)	
GUI and Expression Management: - GUI with a menu that operates as is prescribed and has appropriate user input validation and error handling. - Can add/modify expressions correctly.	Max 10
Expressions Parsing and Evaluation: - Can display current expressions correctly. - Can evaluate a single variable & prints parse tree correctly. Expressions Reading & Sorting: - Can read expressions from a file correctly. - Can sort expressions correctly and write them to file in correct format.	Max 20
Programming techniques, robustness and readability of code: - Appropriate usage of classes and OOP technology. - Appropriate usage of data structures and algorithms. - Code is properly commented and neatly structured. - Application is free of crashes.	Max 20
Group Report: - Report follows the prescribed format. - Report is well written and comprehensive.	Max 10
Group's demonstration: - Group effectively demonstrates the basic features. - Group's ability to answer questions raised in Q&A.	Max 10
Group Total	**70**

The individual component of the assignment will be assessed based on the following criteria:

Assessment criteria (Individual 30 %)	Marks awarded
INDIVIDUAL COMPONENT (30 %)	
Extra Feature One - Technical sophistication. - Usability.	Max 10
Extra Feature Two - Technical sophistication. - Usability.	Max 10
Presentation & Demonstration: - PowerPoint slides. - Demonstration of features and Q&A.	Max 10
Individual Total	**30**

(*) Take note in case of group members with poor contribution to the group effort a multiplier may be applied (peer feedback may be taken into consideration).

~ End ~