Faculty of Engineering and Information Technology
School of Computer Science

42889 & 41889

Autumn 2021

# Assessment Task 1

Individual programming project: Command-line calculator
Due 19 Mar 2021 at 11:59 pm
This project is worth 25% of the overall mark for this subject.

## Objectives

The purpose of this project is to demonstrate competence in the following skills:

- [ ] Program design
- [ ] Array and string manipulation
- [ ] Command line arguments
- [ ] Creating classes and methods in Swift
- [ ] Exception handling
- [ ] Automated testing

These tasks reflect all the subject objectives.

The reference implementation takes about 200 lines of code. As part of your subject workload assessment, it is estimated this project will take 22.5 hours to complete.

## Instructions

1. Fork the project template from Canvas into a private repository.
2. Implement the specified functionality in the `calc` target.
3. Use "Product > Test" to test your functionality.
4. Upload your submission to Canvas. Compress your Xcode project into a zip file and upload the file to "Assessment 1 submission." You may submit maximum 3 times until the final due date. The final submission you make in the one that will be marked.

## Specification

You are to prepare a macOS command-line tool that will act as a simple calculator. The calculator will be run from the command line and will only work with integer numbers and the following arithmetic operators: `+` `-` `x` `/` `%`. The `%` operator is the modulus operator, not percentage.

For example, if the program is compiled to `calc`, the following demonstrates how it will work

```
./calc 3 + 5 - 7
1
```

In the command line, the arguments are a repeated sequence in the form

$$[number\ operator\ \ldots]$$

and ending in a

$$number$$

Hitting the enter key will cause the program to evaluate the arguments and print the result. In this case `1`.

The program must follow the usual rules of arithmetic which say:

1. The `x` `/` and `%` operators must all be evaluated before the `+` and `-` operators.
2. Operators must be evaluated from left to right.

For example, using Rule 1

$$2 + 4 \times 3 - 6$$

becomes

$$2 + 12 - 6$$

which results in

$$8$$

If we did not use Rule 1 then `2 + 4 x 3 - 6` would become `6 x 3 - 6` and then `18 - 6` and finally `12`. This is an incorrect result.

If we do not use Rule 2 then the following illustrates how it can go wrong

$$4 \times 5 \,\%\, 2$$

Going from left to right we evaluate the `x` first, which reduces the expression to `20 % 2` which becomes `0`. If we evaluated the `%` first then the expression would reduce to `4 x 1` which becomes `4`. This is an incorrect result.

Remember, we are using integer mathematics when doing our calculations, so we get integer results when doing division. For example

```
./calc 20 / 3
6
```

Also note that we can use the unary `+` and `-` operators. For example

```
./calc -5 / +2
-2

./calc +2 - -2
4
```

Your program must also check to make sure the command line arguments are valid. If not your program must generate an appropriate error message and then terminate with nonzero exit status.

You should also check for division by zero errors and numeric out-of-bounds errors.

As part of your program design, it is expected you will create classes to model the problem domain.

## Program Hints

1. Getting your program to solve expressions that only use the `+` and `-` operators is fairly easy. I would suggest you get your program working at this level before attempting to get it to work with the other operators.
2. While this problem can be solved using iteration, it is easier to solve using recursion.

## Reference Implementation

The template contains a `reference/calc` executable that you can compare your implementation against. Your implementation should work in **exactly** the same manner.

Please note that it prints results to standard output and additional messages to standard error. Only standard output is used for functionality testing.

# Assessment

**Max score**: 25 marks

### Functionality: 16 marks

The Xcode project must unzip successfully and compile without errors.

- Deduct 3 marks if there are **any** compiler warnings.
- Deduct 1 mark for **each** failing test in the `CalcTest` suite.

### Style: 3 marks

- Deduct up to 1 mark for bad or inconsistent indentation, whitespace, or braces.
- Deduct up to 1 mark for bad or misleading comments.
- Deduct up to 1 mark for unclear symbol naming.

### Design: 6 marks

Good design consists of domain specific classes with good functional separation.

- **Functional separation**
  - Is the problem broken down into functions, classes and different files?
  - Is each class addressing a meaningful problem domain?
  - An example of **bad** functional separation: Everything in one big file with very large functions and many global variables.
- **Loose coupling**
  - Can parts of the code base be modified in isolation? Would changing one portion require significant changes throughout the code base?
  - Is data passed between components in a structured way?
  - An example of **good** loose coupling is when functionality can be re-used in multiple components and potentially different projects.
- **Extensibility**
  - Would it be easy to add more functionality? (more operations, more numerical accuracy, interactivity, variables, etc)
  - Can extra functionality be added to the program with minimal changes. Such as supporting different levels of precedence?
  - **Bad** extensibility would involve many hard-coded strings that are used in multiple places.
- **Control flow**
  - Are all actions of the same type handled at the same level?
  - Can another developer understand the logic flow of your program by reading the main entry point?
  - **Bad** control flow could be caused by exiting the program outside of the main routine.
- **Error handling**
  - Are errors detected at appropriate places? Can they be collected somewhere central?
  - Are errors correctly thrown and caught? Are they appropriately handled in the main routine?
  - Is the user presented with meaningful errors when they do something incorrectly such as providing invalid input?
- **Marker's discretion**

### Late submission

Deduct 1 mark per 24 hours late (rounded up). Submissions will not be accepted after the Final Due Date (7 days after the standard due date).

Please note: Regardless of how many times you have submitted your project, if your final

submission is after the due date it will be considered late and marks will deducted accordingly.

An extension will only be granted if there is a fully documented reason which merits it. The documentation must be presented to the Subject Coordinator before the due date. Extensions after the Final Due Date will never be granted under any circumstance. If an extension is granted that means submission will be accepted up to the extension date without penalty. If an extension is granted, UTS Online will show the extended due date.

Students may apply for special consideration if they consider that illness or misadventure has adversely affected their performance.

## Bug reports

It is quite possible that errors or ambiguities may be found in the task specification, reference implementation, or test suite. If so, updates will be placed on UTSOnline and announcements made regarding the amendment. It is your responsibility to keep up to date on such amendments and ensure you are using the latest version of the Task Specification.

If you discover an error, ambiguity, or bug, you will receive the maximum "Marker's discretion" credit (1 mark). The following rules apply:

1. It must be a report on the currently posted version of the material.
2. It must be reported on the UTS Online discussion board to be accepted.
3. It must be a genuine bug. By genuine I mean it requires me to amend the material.

4. If a number of students post a report on the same bug, the first who posted will receive the mark.
5. No matter how many bug reports you make, you can only get 1 mark.

# Return of Assessed Project

It is expected that marks will be made available one week after the final due date via Canvas. You will also be given a copy of the marking sheet showing a breakdown of the marks and feedback.

# Acceptable Practice vs Academic Malpractice

- Students should be aware that there is no group work within this subject. All work must be individual. However, it is considered acceptable practice to adapt code examples found in the lecture notes, labs and the text book for the assignment. Code adapted from any other source, particularly the Internet and other student assignments, will be considered academic malpractice. The point of the assignment is to demonstrate your understanding of the subject material covered. It's not about being able to find solutions on the Internet.

  You should also note that assignment submissions will be checked using software that detects similarities between students programs.

- Participants are reminded of the principles laid down in the "Statement of Good Practice and Ethics in Informal Assessment" in the Faculty Handbook. Assignments in this subject should be your own original work. Any collaboration with another participant should be limited to those matters described in the "Acceptable Behaviour" section. Any infringement by a participant will be considered a breach of discipline and will be dealt with in accordance with the Rules and By-Laws the University. The Faculty penalty for proven misconduct of this nature is zero marks for the subject. For more information, see UTS Policy on Academic integrity, plagiarism and cheating

# Queries

If you have a question, please contact the instructor as soon as possible. Please do not send email in HTML format or with attachments. They will not be read or opened. Only emails sent in plain text format will be read. Emails without subject lines will be automatically deleted by the junk mail filters I have in place.

If the answer to your questions can be found directly in any of the following:

- subject outline
- task specification
- UTS Canvas discussion board

You will be directed to these locations rather than given a direct answer.