ADS 2 Mid-term CW

This coursework is in the form of a project. You are asked to design and implement software and write a report on your activities.

Postfix++

Postfix++ is a stack-based computer language directed at the evaluation of arithmetic expressions. You will implement a Postfix++ interpreter that can evaluate Postfix++ code line-by-line, as entered, for example, on a mobile device.

Postfix arithmetic

Operators, in postfix arithmetical expressions, follow operands. For example, 3 4 + means 3 + 4. The postfix expression 3 4 5 + * is evaluated as follows:

3 4 5 + *
3 9 * (replace 4 5 + with the result of adding 4 to 5)
27 (replace 3 9 * with 27)

Postfix expressions are conveniently evaluated using a stack. An expression consisting of operands and operators (collectively, 'tokens'), is read from left to right. Successive operands are pushed on a stack until an operator arrives. The appropriate number of operands are then popped from the stack, combined with the operator, and the result is pushed back on the stack. The result of a calculation is always to be found at the top of the stack. A stack is notated [a b c…] in the following example; the stack top is the leftmost token.

Tokens	Stack before	Action	Stack after
3 4 5 + *	[]	Read 3	[3]
4 5 + *	[3]	Read 4	[4 3]
5 + *	[4 3]	Read 5	[5 4 3]
+ *	[5 4 3]	Pop twice, evaluate, and push	[9 3]
*		Pop two twice, evaluate, and push	[27]

Postfix with variables

The '++' refers to an enhanced form of postfix: a postfix with variables. Expressions can contain variable names; the value of the variable is used for the calculation. The assignment operator '=' assigns a value to a variable. For example

A 3 =

will set the value of A to 3. It is equivalent to the infix assignment statement A = 3 that is to be found in many computer languages.

Tokens	Stack before	Action	Stack after
A 3 =	[]	Read A	[A]
3 =	[A]	Read 3	[3 A]
=	[3 A]	Pop twice and set the value of A to 3	[]

An interactive session might proceed as follows

```
> A 2 =
[]
> B 3 =
[]
>A B *
[6]
```

Symbol Table

A **symbol table** data structure associates keys with values. For example,

Example	Key	Value
Phone book	Name	Telephone number
DNS	URL	IP address
Education	Student ID	Module grades
Compiler	Variable name	Memory address
Dictionary	Word	Definition

A generic symbol table should support two operations, INSERT and SEARCH. There may also be a DELETE operation. No assumptions are made on the type and format of keys and values.

You will need a symbol table in your Postfix++ interpreter. The table will map variable name (key) to value.

The target hardware

A Postfix++ interpreter might run on a small device with very limited memory. This means that the variable namespace is small, for example, ‘A’-‘Z’.

Work plan

You must consider what data structures and algorithms are required in your Postfix++ interpreter.

Phase 1: Design

Consider first your choice of data structures. Why are they appropriate? How will they be implemented? Make a list of the data structures and alongside each structure write a few words of justification.

Then consider the major algorithms. There be will algorithms to access and modify the symbol table, and an algorithm to perform postfix arithmetic. Your report will contain pseudocode for each algorithm.

Phase 2: Implementation

You can use any language you wish. Javascript or C++ are good choices.

You must write your own data structures and algorithms. **The only language structure you can use is the array.** Marks will be awarded for the extent of relevant original code.

You should implement at least the four arithmetical operators +, -, *, /.

Phase 3: The report

Your report should contain the following sections

Section 1 The essence of your solution to the challenge in a single paragraph.

Section 2 An explanation of the **original** algorithms of your solution in non-technical language. You do not need to describe the workings of any standard algorithm that might comprise part of your solution.

Section 3 Pseudocode for each original algorithm. You are urged to follow the pseudocode conventions of Cormen at al, Chapter 2. Pay great attention to how you lay out your pseudocode. In particular, take care to use structured indentation. Pseudocode that is not correctly indented, or is otherwise unreadable, will not be marked.

Section 4 A list of the **data structures** in your solution. Explain why each chosen data structure is suitable for the task.

Section 5 Commented source code and a link to a short (< 5 minute) video that shows your code in execution. Ensure that source code is an exact implementation of your pseudocode. An efficient way to demonstrate the relationship between pseudo and actual code is to use lines of pseudocode as comments directly above their implementation in source code. The source code must be visible in the video, as well as the command line interface.

Section 6 Point out any defects of your design and/or implementation. Suggest remedies for these shortcomings.

Submit your report as a pdf. Pay attention to clarity of expression, document presentation, and pseudocode and code formatting. Handwritten answers are prohibited.

Mark scheme

Mark	Descriptor	Specific Criteria
80-100%	**Exceptional**	Exceptional work that **far exceeds the brief**; evidence of an understanding of data structures and algorithms and implementational skill that **surpasses this course** Indicative expectation: a complete implementation of a Postfix++ interpreter for the target hardware; a very high standard of documentation; full coverage of mathematical functions and arithmetical operators; exceptionally clean design and implementation; insightful evaluation
70-79%	**Excellent**	Demonstration of a **thorough grasp** of data structures and algorithms and their implementation; an indication of **considerable independent thought** Indicative expectation: the brief is surpassed; a very high standard of documentation; a successful implementation of a Postfix++ interpreter for the target hardware; wide coverage of mathematical functions and arithmetical operators; excellent design and implementation; full consideration of drawbacks and possible remedies
60-69%	**Very good**	Demonstration of a **sound and competent grasp** of data structures and algorithms and **sufficient implementational skill** Indicative expectation: the brief is met; a very high standard of documentation; a successful implementation of a Postfix++ interpreter for the target hardware; coverage of some mathematical functions and all arithmetical operators; very good design and implementation; wide-ranging consideration of drawbacks and possible remedies
50-59%	**Good**	Demonstration of an **adequate grasp** of data structures and algorithms and **sufficient** practical skills Indicative expectation: a good standard of documentation; the account is satisfactory (it makes sense even if poorly expressed); the implementation has minor errors and/or does not quite meet the proposed objective; design has some failings; some consideration of drawbacks
40-49%	**Pass**	Demonstration of a **threshold grasp** of data structures and algorithms and **limited** practical skills Indicative expectation: documentation is partially deficient; the account is muddled in places; the implementation has many errors and/or does not meet the objective; design has many failings; a superficial account of drawbacks
25-39%	Fail	An **overall failure** of understanding of data structures and algorithms and their implementation Indicative expectation: poor documentation; the account is muddled and incoherent; no or very incomplete implementation; design has many errors; no account of drawbacks
10-24%	Bad fail	A **significant failure** of understanding of data structures and understanding and their implementation Indicative expectation: document standard is unacceptable; the account is incomprehensible; no implementation; erroneous design; no account of drawbacks
1-9%	Very bad fail	**No attempt** to address the brief

Mark	Descriptor	Specific Criteria
0%		The work has not been submitted and/or plagiarised and/or handwritten