COMP3002
Alternative Computing Paradigms

20 CREDIT MODULE

ASSESSMENT: 100% Coursework	**W1: 30% Set Exercises**
	W2: 70% Report

MODULE LEADER: Thomas Wennekers

MODULE AIMS

- To expose students to ways of thinking about computational problems different from mainstream imperative styles.
- To train students in alternative computing paradigms like declarative or functional programming
- To widen students' perspective on computing by introducing them to state-of-the-art novel technology like quantum computing or neural computing

ASSESSED LEARNING OUTCOMES (ALO):

1. Critically evaluate the appropriateness of computing paradigms for a particular application.
2. Apply the use of an alternative paradigm (e.g. functional programming) to produce a solution to a problem.
3. Analyse a technology at the forefront of computational science.

Overview

This document contains all the necessary information pertaining to the assessment of *COMP3002 Alternative Computing Paradigms*. The module is assessed via **100% coursework**, across two elements: *30% Set Exercises* and *70% Report*.

The sections that follow will detail the assessment tasks that are to be undertaken. The submission and expected feedback dates are presented in Table 1. All assessments are to be submitted electronically via the respective DLE module pages before the stated deadlines.

	Submission Deadline	Feedback
Set Exercises (30%)	**22/03/21 4pm**	22/04/21
Report (70%)	**24/05/21 4pm**	16/06/21

Table 1: Assessment Deadlines

All assessments will be introduced in class to provide further clarity over what is expected and how you can access support and formative feedback prior to submission. Whilst the assessment information is provided at the start of the module, it is not necessarily expected you will start this immediately – as you will often not have sufficient understanding of the topic. The module leader will provide guidance in this respect.

Assessment 1: Set Exercises

Tasks:

Q1: Programming Paradigms in BF

This task builds on Lab 02 about the programming language BF.

a) Procedural Programming

   The BF interpreter implementation in the Lab does not allow for function calls. It is therefore not possible to write procedural code (only spaghetti code if using jumps).

   Starting from the file bf.py for the code interpreter, extend the BF language and interpreter such that one can use function calls with an arbitrary number of arguments larger or equal to zero. Assume that arguments are bytes to avoid complications due to argument sizes.

   Functions would be located in the code area as BF code, for example at the end of the code string. They are called by providing the start address of the function code in the string; they would not have names. You would need a new symbol to encode function calls in BF. If the code pointer encounters this symbol during execution the function call mechanism is initiated. You would also need a stack to pass arguments and results. You may want to implement some auxiliary jump mechanisms in which case you would need a few more new BF commands.

   Examples can be found in Lab02.

   The interpreter code, bf.py, can be found at the end of this specification file, and on the DLE in the directory holding the Lab material.

[20 marks]

b) Object Orientated Programming in BF

   How could you extend the BF language and interpreter code to allow for ‘Objects’?

   This part does not ask for code, do not submit any. Instead explain your solution informally in less than 250 words.

[20 marks]

Q2: Recursive and Functional Programming

a) Write Python code that computes the Ackermann function. The code must use decorators and memoization.

   Hint: Lab 03 about Recursion and Dynamic Programming has an example how this tasks could be done.

[20 marks]

b) Implement Depth First **and** Breadth First Traversal (BFT) on trees implemented as dictionaries. The trees do not have to be binary, but nodes may have an arbitrary non-negative number of children. Each entry (ie node) in the dictionary would hold an integer value and a list of children. Implement Depth First Traversal using recursive function calls. Try to also implement Breadth First Traversal using recursive function calls.

   Hint: The Depth First Traversal may be a bit tricky. If you don’t get this working submit a non-recursive version of BFT.

   [20 marks]

Assessment Criteria:

Marks for sub-tasks are indicated in the questions above.

An additional 10 marks are attracted by proper testing of the implementations. This would make use of small examples in the main routine for typical cases.

The remaining 10 marks are for good programming practice, layout, indentifier names, commenting, etc.

Submission Format:

For each of the questions Q1a, Q2a and Q2b submit a single Python code file (3 files in total) that contains all code and some tests demonstrating correctness of the code. Do not submit full Visual Studio solutions, only the code. Code that does not run will be capped at half the marks available for the respective Question.

For Question Q1b submit a word or text file.

Zip everything up into a single file for upload on the DLE.

Assessment 2: Report

Task:

This course work part consists of research into a topic related to alternative computing paradigms written up in form of an individual scientific report of 2,500 words. See below for topics to choose from.

The uploaded report should be a pdf file. Microsoft Word .doc or .docx files, or .odt files will be accepted too, but often display odd formatting for various technical reasons. This could affect marking.

The report must be zipped up with annotated pdfs of the three most important resources used (usually scientific publications) and the zip-file be uploaded via the submission link on the module's DLE pages.

Aim

You are expected to carry out research into an area of Alternative Computing Paradigms and prepare a scientific report about your findings. The aim of the report is to allow you to demonstrate your ability to investigate in depth and present in writing an aspect of the area of Alternative Computing Paradigms.

Content

The work you will carry out must be in one of the following areas:

1) An Application of Functional Programming: This should be a medium size or larger application (i.e., not too small like a Towers of Hanoi or tiny Eliza application). You have 2500 words for the text but can have an appendix for code. (You could also include code in the zipped up submission. The code is not marked, only the report. You can refer to the code in the report to explain aspects of your work.)

2) Non-Boolean Logic: for example, an in-depth exploration of any of first-order logic, modal logic, fuzzy logic, probabilistic logic, multi-valued logic or quantum logic. These alternative logic theories extend the concept of Boolean logic (the one you know) in various ways. The list is not exhaustice, there are more alternative forms of logics yoou could choose from.

3) Quantum Supremacy: You may have seen that Google has recently claimed that it has reached "Quantum Supremacy". Find out what this means, how quantum computing works in principle, what the problems with it are and what precisely Google has achieved.

4) SpiNNaker and Spiking Neurons: SpiNNaker is a neuro-inspired hardware technology currently developed by Steve Furber's group at the University of Manchester. They aim at simulating (mouse) brains in real-time within the next 10 years. Find out what might be possible with this new technology and how it works.

Some of these topics are quite broad and may therefore be difficult to cover in the short space given. You are allowed to constrain them to a more specific sub-topic that can be covered within about 50-70 hours work and within the available space. For this, you would read into the general topics to gain some overview into what they are about and then choose a more constraint topic of your liking to dive into it in depth. If in doubt about your topic of choice ask the module team.

There are few additional constraints other than that your work must have a clear link to Computing/Computer Science. Pure Physics, Maths, Engineering, Philosophy etc. would be inappropriate. However, depending on your topic you may need some interdisciplinary content from these areas of Science. For example, a study on topic 3 would most likely have some Physics content or one on topic 4 some Neuroscience.

You should read some scientific literature about your topic of choice, not just Blog-, News-, Company- or Amateur-webpages about the issues. Scientific journals provide resources of higher quality. You are expected to read and use at least three scientific publications. The manuscripts of three publications must be included in your submission as annotated PDFs (using for example Adobe Reader). The annotations should reflect the level of depth to which you have read the papers. It may be difficult to find scientific papers for some more applied topics; in this case use the best material you can find.

After your research, you will write a scientific report similar in form to scientific reviews or journal papers (compare how the papers you read are composed). The report should have figures to explain concepts and results, and references cited in the text in Harvard style.

One of the lectures will be used to explain the coursework assessment in detail.

Marking scheme

The report should have the form of a scientific review or research paper with the following components (percentages are marks out of 100%).

1. 10% Introduction – what is the investigation about and why is it of interest
2. 20% Background and methods: Description of the fundamental concepts and ideas necessary to understand section 3.
3. 50% Presentation of some specific main issues on the topic.
4. 10% Summary, discussion and conclusions – what are the main points that can be learnt from the investigation and what are problems, that still need to be understood or worked out?
5. 10% Background literature – what are the sources of information for your investigation (journal papers, books, websites)?

Marks will be awarded based for:

1. The organisation and clarity of your written presentation
2. Your use of figures and diagrams where helpful in illustrating key points
3. The breadth and depth of your investigation

4. The extent to which you are able to constructively bring together various aspects of the topic you are investigating to provide greater insight/understanding for the reader
5. Your ability to summarise the key ideas at the end of your report
6. The quality of your referencing and background literature

Pass mark is 40 percent: to pass requires some solid research using reasonably scientific texts or publications documented in a completed report. Distinction level is 70 percent and requires an excellent overview over the chosen field, an in depth understanding of the issues discussed, and the explicit use of high quality academic resources and figures (where appropriate).

COMP3002 – Assessment 2 – Feedback

Introduction	/10
Background and Methods	/20
Main Issues	/50
Summary, Discussion and Conclusions	/10
Literature and References	/10
Overall	/100

Table: Feedback Template for the Assessment

General Guidance

Extenuating Circumstances

There may be a time during this module where you experience a serious situation which has a significant impact on your ability to complete the assessments. The definition of these can be found in the University Policy on Extenuating Circumstances here:
https://www.plymouth.ac.uk/uploads/production/document/path/15/15317/Extenuating_Circumstances_Policy_and_Procedures.pdf

Plagiarism

All of your work must be of your own words. You must use references for your sources, however you acquire them. Where you wish to use quotations, these must be a very minor part of your overall work.

To copy another person's work is viewed as plagiarism and is not allowed. Any issues of plagiarism and any form of academic dishonesty are treated very seriously. All your work must be your own and other sources must be identified as being theirs, not yours. The copying of another persons' work could result in a penalty being invoked.

Further information on plagiarism policy can be found here:

Plagiarism: https://www.plymouth.ac.uk/student-life/your-studies/essential-information/regulations/plagiarism

Examination Offences: https://www.plymouth.ac.uk/student-life/your-studies/essential-information/exams/exam-rules-and-regulations/examination-offences

Turnitin (http://www.turnitinuk.com/) is an Internet-based 'originality checking tool' which allows documents to be compared with content on the Internet, in journals and in an archive of previously submitted works. It can help to detect unintentional or deliberate plagiarism.

It is a formative tool that makes it easy for students to review their citations and referencing as an aid to learning good academic practice. Turnitin produces an 'originality report' to help guide you. To learn more about Turnitin go to:
https://guides.turnitin.com/01_Manuals_and_Guides/Student/Student_User_Manual

Referencing
The University of Plymouth Library has produced an online support referencing guide which is available here: http://plymouth.libguides.com/referencing.

Another recommended referencing resource is Cite Them Right Online; this is an online resource which provides you with specific guidance about how to reference lots of different types of materials.

The Learn Higher Network has also provided a number of documents to support students with referencing:
References and Bibliographies Booklet:
http://www.learnhigher.ac.uk/writing-for-university/referencing/references-and-bibliographies-booklet/
Checking your assignments' references:
http://www.learnhigher.ac.uk/writing-for-university/academic-writing/checking-your-assigments-references/

bf.py

```python
def evaluate(code):

    cells = [0]          # the memory space; 1 element; expands automaticall as required
    cellptr = 0          # pointer into the memory
    codeptr = 0          # pointer into the code provided as a string later

    stack = []

    while codeptr < len(code):

        command = code[codeptr]

        if command == ">":
            cellptr += 1
            if cellptr == len(cells): cells.append(0)       # if memory too small, append a 0

        if command == "<":
            cellptr = 0 if cellptr <= 0 else cellptr - 1

        if command == "+":
            cells[cellptr] = cells[cellptr] + 1 if cells[cellptr] < 255 else 0

        if command == "-":
            cells[cellptr] = cells[cellptr] - 1 if cells[cellptr] > 0 else 255


        # messing with Unicode to get the ' and . operators working

 #    if command == ".": sys.stdout.write(chr(cells[cellptr]))
 #      if command == ".": print( chr(cells[cellptr]))
 #      if command == ",": cells[cellptr] = ord(getch.getch())

        if command == ".": print(chr(cells[cellptr]), end='' )  # end='' overwrites '\n'
        if command == ",": cells[cellptr] = ord(input())


        if command == "[" and cells[cellptr] == 0:  # skip or end the loop
            loop = 1
            while loop > 0:                         # there could be nested loops
                codeptr += 1                        #    which need to be skipped
                c = code[codeptr]
                if c == '[' :
                    loop += 1
                elif c == ']' :
                    loop -= 1                       # if loop gets 0 we found the closing ]


        if command == "]": # and cells[cellptr] != 0:  # end of loop but need to do more iterations
                                                       # definitions of bf ] vary a bit
            loop = 1
            while loop > 0:                         # there could be nested loops
                codeptr -= 1                        # which need to be skipped
                c = code[codeptr]
                if c == '[' :
                    loop -= 1
                elif c == ']' :
                    loop += 1                       # if loop gets 0 we found the closing [
            codeptr -= 1                            #  one left becaus the main increments (as below)

        codeptr+=1

    return cells


# print  "Hello World!"
s = """
    ++++++++++[>+++++++>++++++++++>+++>+<<<<-]
    >++.>+.+++++++..+++.>++.<<+++++++++++++++.
    >.+++.------.--------.>+.>.
"""

# s = """+++>++[<+>-]"""  # add 3 to 2
# s = """,."""            # input, output

print(evaluate(s))
```