SEMESTER TWO **EXAMINATION – MAY 2022**

**MAIN**

|  |  |
| --- | --- |
| **DEPARTMENT:** | Computing |
| **MODULE TITLE:** | **Introduction to Compilers** |
| **MODULE LEADER:** | **Chris Bates** |
| **EXAM DATE:** | 4th May 2022 at 09:30 UK time |
| **DURATION:** | 2 hours 45 minutes (Including 15 minutes reading time) |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**EXAMINATION CONDUCT:**

1. The [University Academic Conduct Regulation](https://students.shu.ac.uk/regulations/conduct_discipline/Academic%20Conduct%20Regulations%202018-19.pdf) outlines the behavioural expectations of candidates completing any examination.
2. Students are responsible for ensuring that they know how to submit their exam script, when the deadline is and that they submit the script in enough time before the deadline expires. It is anticipated that Blackboard will be slower around submission times.
3. It is a fundamental principle that students are assessed fairly and equitably. The [University Academic Conduct Regulation](https://students.shu.ac.uk/regulations/conduct_discipline/Academic%20Conduct%20Regulations%202018-19.pdf) defines unfair behaviour relating to an examination to be 'cheating'. The University will investigate and may sanction any acts or behaviours which breach the Code of Academic Conduct.
4. Students are reminded that this is an individual task and that students who contact or collude with other students to complete their exam may be subject to sanction later.

**INSTRUCTIONS TO CANDIDATES:**

1. This is a time limited examination; you are responsible for managing your time appropriately. The duration is shown at the top of this page.
2. **Answer SECTION A and TWO questions from SECTION B.**
3. APA referencing is required for all questions if you use sources outside of the teaching materials provided in the module.
4. Academic support will be available for the 15 minutes of reading time from 09:30 via Zoom. The address of the Zoom call is available on the Blackboard site in an announcement.
5. It is possible that you may encounter technical issues during the exam; if you have any difficulty with IT you should consult the below student guidance document on My Hallam which contains useful information on hints and tips, contact numbers and links to support: <https://www.shu.ac.uk/~/media/home/myhallam/Guides/student-exam-guidance.docx>
6. Any changes or clarification to the exam paper will be communicated via the module Blackboard site announcements. It is recommended that students monitor Blackboard announcements prior to submission of their final script but particularly in the first hour after release of the exam paper.
7. Submit your work via the Exam Submission Point under the Assessment section of the module Blackboard site (the one to which this document is attached).
8. Your submission must be a Word document. Check your submission previews successfully in Blackboard once you have uploaded it, and that it is the correct piece of work.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SECTION A**

Give an example of a language that is customarily identified with each of the following paradigms. Use code fragments to demonstrate and describe how your chosen languages implement the key features of the paradigm.

|  |  |  |
| --- | --- | --- |
| a) | Object orientation | (20 marks) |
|  | As a paradigm, Object-Oriented Programming (OOP), is heavily focused on the ease of programming, rather than the optimisation. Due to this, it has only more recently become a dominant paradigm, and this is thanks to the excess of computing power enjoyed by modern systems. It is very focused on breaking down problems into their smallest possible parts, making complex tasks easier to solve and large programs easier to develop, test, and maintain.  Java is perhaps the most well-known OOP language and focuses even more than others on the core aspects of the paradigm.  When programming in Java, it is often encouraged to encapsulate whatever data and algorithms that make up the program within classes, as this helps ensure that the program is modular enough in its design to be not only easy to ‘get your head around’, but straightforward to develop, debug, test, and maintain.  class Vehicle {  protected String brand = "Ford";  public void honk() {  System.out.println("Tuut, tuut!");  }  }  class Car extends Vehicle {  private String modelName = "Mustang";  public static void main(String[] args) { Car myCar = new Car();  myCar.honk();  }  }[1]  [1]. <https://www.w3schools.com/java/java_inheritance.asp> |  |
| b) | Declarative programming | (20 marks) |
|  | While it is a somewhat loosely defined paradigm (many modern languages blur the lines and implement features from numerous different programming paradigms), declarative programming is essential in many computer systems and is a great tool. The basis behind declarative programming is that instead of describing the steps that must be taken, the programmer instead describes the result they want. The method used to reach this result is left up to the system, and implementations can be updated and optimised over time, making pure declarative languages quite future-proof[1].  SQL is a well-known declarative language and is the most popular language used for managing and querying databases. It is domain-specific, meaning it is optimised for a predefined type of problem[2] (database management), and is typically not used for anything else. The idea of domain-specificity is closely interlinked with the declarative paradigm, as a limitation on the type of problems presented to the system allows it to produce the results requested by the user with greater ease and efficiency.  SELECT CustomerName,City FROM Customers;  This is an example of a simple SQL ‘select’ statement. The syntax is designed to be almost like natural language, which helps achieve an intuitive understanding, even with little programming experience, making SQL accessible and easy to work with.  While this is not critically linked to the declarative paradigm, it certainly supports it, as the general idea is to lessen the necessity of problem solving when writing code, and instead shift the focus onto simply reaching a solution that has likely already been defined by business processes.  For example, if a boss wants to know how many customers are from a certain city, a simple query can be issued to discover this:  SELECT CustomerName FROM Customers WHERE City=”Sheffield”;  When writing an SQL statement, it is almost like asking a friend or colleague to do something. This reflects the core idea behind declarative programming – the true process is not of any bother to the programmer; all they care about is the result.  [1]. <https://www.ionos.co.uk/digitalguide/websites/web-development/declarative-programming/>  [2]. <https://www.jetbrains.com/mps/concepts/domain-specific-languages/>  SQL code samples taken from <https://www.w3schools.com/sql/sql_examples.asp> |  |
| c) | Procedural programming | (20 marks) |
|  | Procedural programming is one of the older programming paradigms, and in modern languages is often overshadowed by OOP, though procedures are still very much present. Procedural programming is, at its core, a set of instructions that tell the computer what to do. At a more advanced level, the ‘procedural’ part refers to the separation of parts of the program into procedures that usually perform a specific task and can be called many times in the rest of the program.  C is one of the last languages that remains purely procedural, and even then, most new developers will opt for C++ with its OOP capabilities instead. However, plain C still finds use in embedded systems and solutions that cannot offer great computational power, or large storage or memory capacity, as it can be written in an extremely minimal and efficient way whilst still being a very powerful language thanks to how low level it is. |  |

**SECTION B**

**QUESTION ONE**

Seymour Papert has written that each programming language “favours certain metaphors, images, and ways of thinking” in its design. Show how an object-oriented language of your choice is designed to support the implementation of large and complex software.

(20 marks)

Java is one of the most popular programming languages in use today and is well known for its heavy design emphasis on Object-Oriented Programming (OOP), to the point that it makes some efficiency sacrifices in favour of ease of design, iterability, and deployment.

When programming in Java, it is often encouraged to contain whatever data and algorithms that make up the program within classes, where other languages (for example python or C) would not do this, as it costs memory and execution time.

However, this benefits ease of design especially with large and complex software because if all data and procedures are contained within classes with standardised interfaces, it is easy to debug issues, as each part of the program can be tested individually, and it is easy to maintain code or update implementations of algorithms so long as the standardised inputs and outputs remain unchanged. This general concept is split into four different ideas, which are considered the pillars of OOP[1]:

* **Abstraction** is the idea of ‘hiding’ functionality behind simple function calls and return values, with descriptive documentation. This helps the overarching logic of a complex or large program be much easier to grasp, and also helps test each small part of the code individually.
* **Encapsulation** is the practice of protecting data within classes, and providing public methods, such as getters, setters, or custom procedures, that can access and work with this data. This ensures validity, and in some cases, security of data, which is important when developing a large or complex program, as already developed and finalised parts of the program can be trusted to not fail.
* **Inheritance** lets new classes literally ‘inherit’ data members and functionality from other classes, meaning new custom classes don’t have to needlessly repeat core code. This is helpful when creating a large project, as often there are simple or core logical concepts that repeat many times throughout the codebase.
* **Polymorphism** is an intelligent feature of OOP that allows different classes to be treated the same way but react differently based on their design. An example of this in Java is method overloading, which can be done when inheriting. This allows for an elegant way of interacting with many different objects dynamically in real-time.

[1]. <https://stackify.com/oops-concepts-in-java/>

**~~QUESTION TWO~~**

~~Using supporting examples taken from the module, compare and contrast a stream parser with a parser that creates a static tree structure. Consider the following aspects~~

|  |  |  |
| --- | --- | --- |
| ~~a)~~ | ~~The use of memory and other system resources~~ | ~~(5 marks)~~ |
| ~~b)~~ | ~~Flexibility~~ | ~~(10 marks)~~ |
| ~~c)~~ | ~~Complexity~~ | ~~(5 marks)~~ |

**QUESTION THREE**

|  |  |  |
| --- | --- | --- |
| a) | What is a templating engine used for? | (5 marks) |
|  | When processing large amounts of data and wishing to format it in a specific way, code solutions utilising features like print or other output statements and formatted strings quickly become unreasonably complex and thus difficult to test, maintain, or develop further.  A good example of a real-world use of a templating engine is marketing emails, which are sent en masse to sometimes millions of people. Obviously, automation is required to personalise these emails with individual details like usernames, as such a task is insurmountable with any manual solution, and this is what templating engines are designed for:  A template such as “Hello $user, have you heard about our May deals!?” could be placed into a templating-enabled email distribution system, linked to a company’s user database. The system would identify the “$user” tag in the email as a key to be replaced and would fetch the appropriate username (value) from the database, substituting it into the email in place of the “$user” before sending a ‘rendered’ final text version as an email to the user; “Hello Chris, have you heard about our May deals!?” |  |
| b) | Show how a templating engine can be used to create the text of a JavaScript program from a simple parse tree.  The fundamental idea behind parse trees is to represent a set of instructions and information in a structured manner, as defined by a ruleset, such as a grammar. When this is achieved, a templating engine can process the data presented by a parse tree using the aforementioned ruleset, and format it using its own specified template, to reliably and efficiently produce a newly formatted output.  This is the basis behind compilation of code, and it can be applied to produce a JavaScript program from a parse tree like so:  Text  Description automatically generatedThe image above shows a small parse tree representing a while loop in PDL, a simple procedural programming language.  The tree provides a structured form of a set of instructions that would, in their raw form, be difficult to write code to parse. Each instruction or piece of data which is comprised of more than one atomic ‘element’, as defined by the PDL grammar, has child nodes. All nodes in the tree are labelled with a key and contain a value.  By walking, or in some way parsing the tree, a templating engine can find keys and their corresponding values, like “number = 0” instead of “user = Chris”, and using the structure of the tree, which maintains the logical order of instructions, can use its template, which might contain blueprints like the one shown below, can ‘render’ text that constitutes a valid JavaScript program.  enterProgram(v) ::= << \<script> function pdlProgram() { document.getElementById("output").innerHTML = ""<\n> >>  This code sample is from my assignment and defines in a string template file, which is used by my templating engine, that when the “enterProgram” key is found in a PDL parse tree, the standard start of a HTML-embedded JavaScript script should be returned to be added to the final rendered text.  Another example below shows another part of the template file that accepts a value, and dynamically varies the returned text based on that value, which is the basis of how actual user-defined code can be produced.  globals(ident) ::= << var <ident>;<\n> >> | (15 marks) |

**~~QUESTION FOUR~~**

|  |  |  |
| --- | --- | --- |
| ~~a)~~ | ~~What is a Antlr?~~ | ~~(6 marks)~~ |
| ~~b)~~ | ~~Discuss the use of the listener pattern in an Antlr-based code generator.~~ | ~~(14 marks)~~ |

*[ END OF PAPER ]*