Engineering, Built Environment and IT
Department of Computer Science
Programming Languages
COS 333

Examination
14 November 2023

Examiner: Mr W. S. van Heerden
External Examiner: Prof AB van der Merwe (Stellenbosch)
Instructions:
Read the questions carefully and answer all questions.
This section comprises of **29** questions, and a total of **40** marks.
You have **3** hours to complete this test.
This test is an *online* assessment:
The test will automatically submit when the time (3 hours) expires. You can also submit the test yourself if you are done ahead of time.
This test is **closed book** and is subject to the University of Pretoria integrity statement provided below.
You are not allowed to consult any sources other than the MIT/GNU Scheme Reference Manual (provided as documentation for Practical 2) and the SWI-Prolog Reference Manual (provided as documentation for Practical 3).
Your Scheme submission for Question 6 must be working Scheme code that will successfully interpret using version 8.8 of the DrRacket interpreter. You may (and should) use this interpreter to test your submission. Once you have finished writing your program, make sure it is saved in a text file with the appropriate name (described in the question), and submit it to the correct upload slot.
Your Prolog submission for Question 7 must be working Prolog code that will successfully interpret using version 7.6.4 of the SWI-Prolog interpreter. You may (and should) use this interpreter to test your submission. Once you have finished writing your program, make sure it is saved in a text file with the appropriate name (described in the question), and submit it to the correct upload slot.
You may use a non-programmable calculator.
You may not use any other programmable devices.
Integrity Statement:
The University of Pretoria commits itself to producing academic work of integrity. I affirm that I am aware of and have read the Rules and Policies of the University, more specifically the Disciplinary Procedure and the Tests and Examinations Rules, which prohibit any unethical, dishonest or improper conduct during tests, assignments, examinations and/or any other forms of assessment. I am aware that no student or any other person may assist or attempt to assist another student, or obtain help, or attempt to obtain help from another student or any other person during tests, assessments, assignments, examinations and/or any other forms of assessment.

QUESTION 1

1. Various factors can influence programming language design. Consider the following programming language design influences, and **select** the one that had its earliest impact on the design of SIMULA 67.

○ Computer architecture.

○ Programmer efficiency becoming more important than machine efficiency.

○ The move from process oriented programming to data oriented programming.

○ The move to object-oriented programming.

1 points

QUESTION 2

1. Consider the following programming language evaluation criteria, and **select** only those that benefit from the use of a hybrid implementation system. Note that incorrectly selected options will be penalised.

□ Readability.

□ Writability.

□ Cost.

□ Portability.

2 points

QUESTION 3

1. Consider only the syntax used by Plankalkül for an assignment using arrays and subscripts. Consider the following statements, and **select** the one that is the most correct.

○ The syntax had poor readability.

○ The syntax had poor writability.

○ The syntax had poor reliability.

○ The syntax had poor execution cost.

1 points

QUESTION 4

1. The language specification of ALGOL 60 lacked support for I/O operations. Consider the following statements, and **select** the one that most correctly describes a consequence this lack of support had on a main design goal of ALGOL 60.

○ The reliability of ALGOL 60 suffered.

○ The execution cost of ALGOL 60 suffered.

○ The execution cost of ALGOL 60 improved.

○ The machine independence of ALGOL 60 suffered.

1 points

QUESTION 5

1. Consider the following statements about Lua, and **select** the one that is the most correct.

○ Lua uses full compilation for the sake of reduced execution cost.

○ Lua requires heap allocated variables to be explicitly deallocated.

○ Lua supports functional programming.

○ Lua is a fully featured object-oriented programming language.

1 points

QUESTION 6

1. **Write** a Scheme function named `doublePositives`, which receives one parameter. The parameter is a simple list containing only numeric atoms. The function should yield (not print out) a list containing the same values as the parameter list, but with positive values doubled. Negative and zero values remain unchanged.

For example, the function application

```
        (doublePositives '())
```

should yield an empty list because there are no elements in the parameter list. As another example, the function application

```
        (doublePositives '(2 -4 7 -3))
```

Should yield the list `(4 -4 14 -3)` because 2 doubled is 4, and 7 doubled is 14, and the remaining values remain unchanged because they are negative.
Submit your program code in a file named `s99999999.scm` (where `99999999` is your student number) to the assignment submission slot labelled "Examination: Scheme Function", and select the "Yes" answer below once you have done so. If you do not make a submission, select the "No" answer below.

Take careful note of the following requirements:

- Your submission must be working Scheme code that will be successfully interpreted by version 8.8 of the DrRacket interpreter (this means, amongst other things, that lowercase letters should be used for all built-in function names). You may (and should) use the DrRacket interpreter to test your submission.
- Please start your program with the line `#lang racket`, not `#lang sicp` as specified in Practical 2.
- You may define additional helper functions.
- Your Scheme functions may only use the following built-in functions in the way they are used in the slides and textbook **(failure to observe this rule will result in all marks for this question being forfeited)**:
  - Function construction: `lambda`, `define`
  - Binding: `let`
  - Arithmetic: `+`, `-`, `*`, `/`, `abs`, `sqrt`, `remainder`, `modulo`, `min`, `max`
  - Boolean values: `#t`, `#f`
  - Equality predicates: `=`, `>`, `<`, `>=`, `<=`, `even?`, `odd?`, `zero?`, `negative?`, `eqv?`, `eq?`
  - Logical predicates: `and`, `or`, `not`
  - List predicates: `list?`, `null?`
  - Conditionals: `if`, `cond`, `else`
  - Quoting: `quote`, `'`
  - List manipulation: `list`, `car`, `cdr`, `cons`
  - Input and output: `display`, `printf`, `newline`, `read`

○ Yes

○ No

5 points

QUESTION 7

1. **Write** the Prolog proposition called `doubleNonMatching(L1, X, L2)`, which succeeds if the list `L2` contains the same values as the list `L1`, without their order being modified, but with all values that do not match `X` doubled. All values that do match `X` should remain unchanged. You may assume that `L1` and `L2` are both simple lists containing only integer values, and that `X` is an integer value. For example, the query:

```
?- doubleNonMatching([], 2, X).
```

should be true with the instantiation `X = []`. This is because no values are contained in an empty list. Similarly, the query

```
?- doubleNonMatching([3, 2, 4, 4, 2], 2, X).
```

should be true with the instantiation `X = [6, 2, 8, 8, 2]`. This is because 3 and 4 do not match 2, and 3 doubled is 6, while 4 doubled is 8, and the rest of the values remain unchanged because they match 2.
Hint 1: When testing your proposition, use variables in your queries, in a similar fashion to the example queries provided above.
Hint 2: In Prolog arithmetic, * represents a multiplication operation.
Submit your program code in a file named `s99999999.pl` (where `99999999` is your student number) to the assignment submission slot labelled "Examination: Prolog Proposition", and select the "Yes" answer below once you have done so. If you do not make a submission, select the "No" answer below.

Take careful note of the following requirements:

- Your submission must be working Prolog code that will be successfully interpreted by the SWI-Prolog interpreter installed on the Windows computers in the Informatorium. You may (and should) use this SWI-Prolog interpreter to test your submission. For notes on using the interpreter, and re-querying, see the specification for Practical 3. Once you have finished writing your program, save it in a text file with the appropriate name, and submit it to the correct upload slot.
- You may define additional helper propositions.
- **Note that you may only use the simple constants, variables, list manipulation methods, arithmetic and relational expressions, and built-in predicates discussed in the textbook and slides.** In particular, do not use if-then, if-then-else, and similar constructs. You may NOT use any more complex predicates provided by the Prolog system itself. In other words, you must write all your own propositions. **Failure to observe this rule will result in all marks for this question being forfeited.**

○ Yes

○ No

5 points

QUESTION 8

1. Consider the following program code in a hypothetical programming language:

```
void main() {
   var myVariable = "Test"
   print(myVariable)

   var anotherVariable = 5
   myVariable = anotherVariable * 2
   print(myVariable)
}
```

Consider the following options, and **select** the one that most correctly describes the category of `myVariable`, in terms of lifetime.

○ Static variable.

○ Stack-dynamic variable.

○ Explicit heap-dynamic variable.

○ Implicit heap-dynamic variable.

1 points

QUESTION 9

1. Consider the following two subprograms in a hypothetical programming language:

```
void f1() {
   constant LENGTH = 3
   var firstArray[LENGTH] = {6, 8, 7}
}
void f2() {
   constant INPUT_LENGTH
   readInteger(INPUT_LENGTH)
   var secondArray[INPUT_LENGTH]
}
```

In this code, the special word `constant` is the only way used to indicate the declaration of a named constant. Subprogram `f1` is legal, while subprogram `f2` is illegal and will not compile. **Name** the type of named constants supported in this programming language.

1 points

QUESTION 10

1. Consider the following statements about list comprehensions in Python, and **select** the one that is the most correct.

○ List comprehensions increase the readability of Python.

○ List comprehensions increase the writability of Python.

○ List comprehensions improve the reliability of Python.

○ List comprehensions reduce the execution cost of Python.

1 points

QUESTION 11

1. C++ is not considered a very strongly typed programming language. Consider the following reasons, and **select** the one that most correctly describes a reason contributing to the weak typing of C++.

○ The way in which C++ supports pointers.

○ The fact that C++ supports only discriminated unions.

○ The fact that C++ performs no parameter type checking.

○ The fact that C++ does not perform index range checking in arrays.

1 points

QUESTION 12

1. Both Ruby and Java are object-oriented programming languages. Consider the following statements about arithmetic operators in Ruby and Java, and **select** the one that is the most correct.

○ Arithmetic operators in Ruby are less flexible than arithmetic operators in Java.

○ Arithmetic operators in Ruby are always as reliable as arithmetic operators in Java.

○ Arithmetic operators in Ruby are implemented in a more orthogonal way than arithmetic operators in Java.

○ Arithmetic operators in Ruby have a lower execution cost than arithmetic operators in Java.

1 points

QUESTION 13

1. Consider the following statements about the unary `*` operator in C++, and **select** the one that is the most correct.

○ The unary `*` operator is poorly designed, because it is overloaded to produce very different results depending on the context in which it is used.

○ The unary `*` operator is poorly designed, because C++ could have been designed to avoid its use while leaving the language’s flexibility and conciseness unchanged.

○ The unary `*` operator is well designed, because it is not overloaded.

○ The unary `*` operator is well designed, because it cannot be confused with any non-unary operators.

1 points

QUESTION 14

1. Consider the following statements about conditional targets, and **select** the one that is the most correct.

○ Conditional targets improve the readability of assignments.

○ Conditional targets improve the writability of assignments.

○ Conditional targets improve the reliability of assignments.

○ Conditional targets decrease the cost of execution of assignments.

1 points

QUESTION 15

1. Consider the following statements about mixed-mode assignments in Ada and C, and **select** the one that is the most correct.

○ Mixed-mode assignments in Ada are less readable than mixed-mode assignments in C.

○ Mixed-mode assignments in Ada are less writable than mixed-mode assignments in C.

○ Mixed-mode assignments in Ada are less reliable than mixed-mode assignments in C.

○ Mixed-mode assignments in Ada have a higher execution cost than mixed-mode assignments in C.

1 points

QUESTION 16

1. Consider the following statements about the approaches used by Ruby and Perl to disambiguate nested selectors, and **select** the one that is the most correct.

○ Ruby uses a less reliable approach than Perl.

○ Ruby uses a less readable approach than Perl.

○ Ruby uses a less writable approach than Perl.

○ Ruby uses a more writable approach than Perl.

1 points

QUESTION 17

1. Consider the following statements about selector expressions used in assignments, and **select** the one that is the most correct.

○ Readability is improved if the else clause is left out of a selector expression used in an assignment.

○ Readability is reduced if the else clause is left out of a selector expression used in an assignment.

○ Reliability is improved if the else clause is left out of a selector expression used in an assignment.

○ The programming language evaluation criteria are not affected if the else clause is left out of a selector expression used in an assignment.

1 points

QUESTION 18

1. Consider the following statements about counter-controlled loops in Ada and C, and **select** the one that is the most correct.

○ Counter-controlled loops are more writable in Ada than in C.

○ Counter-controlled loops are less readable in Ada than in C.

○ Counter-controlled loops are more reliable in Ada than in C.

○ Counter-controlled loops have a higher cost of execution in Ada than in C.

1 points

QUESTION 19

1. Consider the following program code in Ruby:

```
def doSomething()
   count = 1
   value = 1
   while count <= 3
      value = value * 2
      count = count + 1
      yield value
   end
end
```

Provide the output of this program code if the `doSomething` subprogram is used with a block that has a single formal parameter, and prints this formal parameter out, followed by a single space. No additional output is printed.

1 points

QUESTION 20

1. Consider the following statements about default values for subprogram parameters, and **select** the one that is the most correct.

○ There is no way for a default parameter value to appear anywhere but the beginning of a subprogram’s list of parameters.

○ There is no way for a default parameter value to appear anywhere but the end of a subprogram’s list of parameters.

○ It is possible for a default parameter to appear anywhere in a subprogram’s list of parameters if positional parameters are used.

○ It is possible for a default parameter to appear anywhere in a subprogram’s list of parameters if keyword parameters are used.

1 points

QUESTION 21

1. Consider the following statements about a drawback associated with Lua, and **select** the one that is the most correct.

○ Anonymous functions in Lua cannot be used after the function’s definition.

○ It is easy for a programmer to unintentionally create a stack-dynamic variable in Lua.

○ It is easy for a programmer to unintentionally create a global variable in Lua.

○ Lua does not allow a function to receive a variable number of parameters.

1 points

QUESTION 22

1. Consider the following statements about pass-by-result subprogram parameters, and **select** the one that is the most correct.

○ Pass-by-result parameters can only copy a value to the caller.

○ Pass-by-result parameters can only transfer an access path to the caller.

○ Pass-by-result parameters can either copy a value to the caller, or transfer an access path to the caller.

○ Pass-by-result can neither copy a value to the caller, nor transfer an access path to the caller.

1 points

QUESTION 23

1. Consider the following program code in a hypothetical programming language:

```
var A = 12
fun sub1() {
   print A
}
fun sub2(param) {
   var A = 35
   call param()
}
fun main() {
   var A = 22
   sub2(sub1)
}
```

Assume the programming language allows subprograms to be passed as parameters, uses static scoping rules, and that execution starts with the main subprogram. **Provide the output** of the program code if the following types of binding are used to determine the referencing environment of the passed subprogram.

Ad-hoc binding: □

Shallow binding: □

2 points

QUESTION 24

1. Consider the following program code in a hypothetical programming language:

```
sub main() {
   resume second
}
coroutine first {
   print "1 "
   resume second
   print "2 "
}
coroutine second {
   print "3 "
   resume first
   print "4 "
}
```

Assume that a subprogram definition starts with the special word `sub`, a coroutine definition starts with the special word `coroutine`, and that execution of the program begins with the `main` subprogram. **Provide the output** of the program code. Note that there is a single space after each number that is printed.

1 points

QUESTION 25

1. Consider the following statements about properties in C# classes, and **select** the one that is the most correct.

○ Properties provide instance variables within a C# class.

○ Properties provide class variables within a C# class.

○ Properties provide a less writable alternative to getters and setters within a C# class, from the perspective of a user of the class.

○ Properties provide a more writable alternative to getters and setters within a C# class, from the perspective of a user of the class.

1 points

QUESTION 26

1. Consider the following statements about the approach used by Java to hide the representation of ADT objects from the users of these objects, and **select** the one that is the most correct.

○ The ADT interface is provided in a header file, which the user is allowed to see. The ADT implementation is compiled into a class file, and only the class file is provided to the user.

○ The ADT interface is not provided to the user. The ADT implementation is compiled into a class file, and only the class file is provided to the user.

○ The ADT interface is automatically generated as documentation, which the user is allowed to see. The ADT implementation is compiled into a class file, and only the class file is provided to the user.

○ Java does not hide the representation of ADT objects from the users of these objects.

1 points

QUESTION 27

1. Consider the following program code in Java:

```
public class MyClass<T> {
   private T data;
   public MyClass(T param) {
      data = param;
   }
   public T getData() {
      return data;
   }
}
```

Also assume that the following `main` method is provided:

```
public static void main(String[] args) {
   MyClass<String> a = new MyClass<String> ("test");
   Integer myInt = new Integer(12);
   MyClass<Integer> b = new MyClass<Integer> (myInt);
}
```

Identify the number of versions of `MyClass` that exist after compile time. Type only the digit (e.g. 5).

□

1 points

QUESTION 28

1. Consider a hypothetical programming language that supports object-oriented programming with multiple inheritance. In this programming language four classes are defined, named A, B, C, and D. Classes B and C both inherit only from class A. Class D inherits from both classes B and C. It is possible for a method name collision to occur **implicitly**. Consider the following implementation details, and **select** only those that together result in an **implicit** name collision. Note that incorrectly selected options will be penalised.

□ Defining a method named `myMethod` in class A.

□ Defining a method named `myMethod` in class B.

□ Defining a method named `myMethod` in class C.

□ Defining a method named `myMethod` in class D.

1 points

QUESTION 29

1. C++ has a lower cost of execution than Smalltalk. Consider the following factors, and **select** only those that contribute to the lower cost of execution in C++. Note that incorrectly selected options will be penalised.

□ The approach C++ uses in relation to the exclusivity of classes.

□ The approach C++ uses to perform message binding.

□ The fact that C++ supports heap allocated objects.

□ The fact that subclasses are not necessarily subtypes in C++.

2 points