A10098W1

FIRST PUBLIC EXAMINATION

Preliminary Examination in Computer Science
Preliminary Examination in Mathematics and Computer Science
Preliminary Examination in Computer Science and Philosophy

Imperative Programming

LONG VACATION 2017
Monday 11th September, 9:30 am – 12:30 pm

*Candidates should answer no more than **five** questions.*
*with no more than **three** questions from either half.*
Please start the answer to each question on a new page.

Answers to Questions 1–4 and Questions 5–8 should be submitted in separate bundles.

Do **not** turn over until told that you may do so.

Imperative Programming 1

Question 1

This question is about implementing exact arithmetic for irrational numbers of the form $a+b\sqrt{5}$ where a and b are rational numbers.

In most of this question the numbers will be represented by elements of

type RATIONAL = **pointer to record** *num*, *den*: INTEGER **end**;
IRRATIONAL = **pointer to record** *rat*, *irr*: RATIONAL **end**;

The components of a RATIONAL are the numerator and denominator (normally in lowest terms), and those of an IRRATIONAL are the rational part and the coefficient of $\sqrt{5}$.

(Recall that if z is a variable of a pointer type, NEW(z) allocates storage for a record of the appropriate kind and sets z to point at that record.)

(a) Write a procedure

procedure *mkRat*(n, d: INTEGER): RATIONAL;

which returns a RATIONAL representing n/d. (2 marks)

(b) Write a procedure

procedure *reduce*(x: RATIONAL);

which reduces x (which might not be in lowest terms) to its lowest terms. You may assume the existence of a procedure

procedure *gcd*(x, y: INTEGER): INTEGER;

which returns the greatest common divisor of x and y. (2 marks)

(c) Write procedures

procedure *addRat*(x, y: RATIONAL): RATIONAL;
procedure *mulRat*(x, y: RATIONAL): RATIONAL;
procedure *negRat*(x: RATIONAL): RATIONAL;

which return the sum of x and y, the product of x and y, and the value of $-x$ respectively. (4 marks)

(d) Write procedures

procedure *mkIrr*(r, i: RATIONAL): IRRATIONAL;
procedure *addIrr*(x, y: IRRATIONAL): IRRATIONAL;
procedure *mulIrr*(x, y: IRRATIONAL): IRRATIONAL;

which make an IRRATIONAL number, the sum of x and y, and the product of x and y respectively. (4 marks)

(e) Write a procedure

procedure *OutIrr*(x: IRRATIONAL);

which prints a representation of an IRRATIONAL in as simple a form as possible, for example `1/2+1/2*sqrt(5)` for $\frac{1+\sqrt{5}}{2}$, but 2 for $2+0\sqrt{5}$. (4 marks)

(f) The ith Fibonacci number can be shown to be

$$\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^i - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^i$$

Use the procedures above to write a program that prints the first n Fibonacci numbers by evaluating this expression.

For clarity, you may want to initalise named variables of the appropriate types to useful values such as the rationals 0 and $\frac{1}{2}$ and $\frac{1}{5}$, and the values $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$. (4 marks)

Question 2

The *spectrum* of a real number x is the infinite set of integers $\{\lfloor kx \rfloor \mid k \in \mathbb{N} \wedge 1 \leqslant k\}$. For example the spectrum of $\sqrt{2}$ consists of $\{1, 2, 4, 5, 7, 8, 9, \ldots\}$ and larger integers, omitting in particular 3 because $\lfloor 2\sqrt{2} \rfloor < 3 < \lfloor 3\sqrt{2} \rfloor$.

This question is about exact calculations of the spectrum of $\sqrt{r}$ for positive integers r.

(a) An integer i is in the spectrum of $\sqrt{r}$ if and only if there is some natural number k for which $i = \lfloor k\sqrt{r} \rfloor$. Express this condition in a way that can be computed using only integer arithmetic. (2 marks)

(b) Write a (non-recursive) procedure

procedure *spectrum*(r, n: INTEGER);

which prints all those elements of the spectrum of $\sqrt{r}$ less than n by calling *Out.Int* on each i in the spectrum for $1 \leqslant i < n$.

Your program should take a time that grows no faster than $O(n)$. (5 marks)

(c) Give enough of an invariant for any loop to explain why the program works. (3 marks)

(d) Write a (non-recursive) proceedure

procedure *test*(i, r: INTEGER): BOOLEAN;

which for any $i > 0$ checks whether i is in the spectrum of $\sqrt{r}$, and does so in a time that grows no faster than $O(\log(i))$. (7 marks)

(e) Give an explanation of why the program works, and does so in logarithmic time, based on invariants for each loop. (3 marks)

Question 3

In this question you are asked to write fragments of non-recursive programs, that is to write loops, and you should give an invariant for each loop sufficient to justify that it meets the specification. In each case your program should run in a time proportional to n.

(a) A *plateau* in an array $a[0\,..\,n)$, where $n \geqslant 0$, is a segment $a[p\,..\,q)$ such that if $p \leqslant i < q$ then $a[i] = a[p]$. Write a program that in a time proportional to n computes the length m of the longest plateau in the array a. (4 marks)

(b) Show how to modify your program to identify the position of a plateau with the length identified in part (a), without changing its asymptotic complexity. (2 marks)

(c) For the purposes of this question an *increasing* segment $a[p\,..\,q)$ of a is a non empty segment, with $p < q$, which satisfies $a[i] \leqslant a[j]$ for all $p \leqslant i \leqslant j < q$. So in particular every segment of length one is increasing.

Using your answer to earlier parts as a model, or otherwise, write a program which in a time proportional to n finds the length of the longest ascending segment of $a[0\,..\,n)$ and identifies the position of an ascending segment of that length. (6 marks)

(d) A *bitonic* segment is one which consists of an increasing segment followed by a decreasing segment, or vice versa. For definiteness, since a single element is enough for an increasing or a decreasing segment, this means that every increasing segment is bitonic, and so is every decreasing segment. The longest bitonic segments in $[1, 2, 3, 3, 2, 4, 5]$ are $[1, 2, 3, 3, 2]$ and $[3, 3, 2, 4, 5]$. Notice that maximal bitonic segments overlap, and that in particular a plateau can appear either at the beginning or at the end of an increasing (or a decreasing) sequence.

Formulate a suitable invariant and use it to write a program which finds the length of the longest bitonic segment of $a[0\,..\,n)$ and identifies the position of a bitonic segment of that length. (8 marks)

Question 4

This question is about manipulating sparse polynomials of one variable with coefficients from some number type T. A polynomial is sparse if it has many zero coefficients compared with its degree, and will be represented by a list of coefficient-power pairs, omitting the zeroes. So for example $x^{100} + 13x + 1$ would be represented by $\langle(1, 100), (13, 1), (1, 0)\rangle$.

(a) Choose suitable types for representing a polynomial by a linked list of records of coefficient-power pairs, and give the Oberon type definitions. (2 marks)

(b) Write a procedure

procedure *normalise*(**var** r: POLY);

which removes from a polymomial any terms that have zero coefficients. (2 marks)

(c) Write a procedure

procedure *addPoly*(x, y: POLY): POLY;

which returns a polynomial representing the sum of x and y, without in any way changing its arguments.

You may want to design and use subroutines which can be useful both in this part and in the next part of the question. (4 marks)

(d) Write a procedure

procedure *mulPoly*(x, y: POLY): POLY;

which returns a polynomial representing the product of x and y, without in any way changing its arguments. (4 marks)

(e) Write a procedure

procedure *OutPoly*(p: POLY);

which writes a representation of a polynomial, for example something like `-1+x^2` for $\langle(-1, 0), (1, 2)\rangle$.

You should assume that there is a procedure

procedure *OutT*(a: T);

which you can use to print a coefficient. (4 marks)

(f) A user might prefer *OutPoly* to produce its output in a standard order: perhaps listing terms in decreasing order of degree. This might be done by sorting a representation when it is printed, or by keeping all representations in a sorted order.

Without writing the code to do this, outline the costs and benefits of keeping all polynomial representations in sorted order. (4 marks)

Imperative Programming 2

Question 5

This question is about double dispatch and trait linearization.

(a) Consider the following Scala program:

```
class Drink
class Milk extends Drink
class Boy {
def concept(m: Milk) { println("Milk") }
def concept(d: Drink) { println("Drink") }
}
class Baby extends Boy {
override def concept(m: Milk) { println("White drink") }
override def concept(d: Drink) { println("Food") }
}
val d: Drink = new Milk
val b: Boy = new Baby
b.concept(d)
```

The output of the above program is `Food`.

Using *double dispatch* modify the class definitions in the above program so that the output is `White drink`. (8 marks)

(b) Consider the following Scala program:

```
trait K { def m = "K"}
trait I extends K { override def m = super.m + "I"}
trait J extends K { override def m = super.m + "J"}
trait G extends I { override def m = super.m + "G"}
trait F extends I with J { override def m = super.m + "F"}
trait H extends J { override def m = super.m + "H"}
trait D extends G with F { override def m = super.m + "D"}
trait E extends K with H { override def m = super.m + "E"}
class A extends D { override def m = super.m + "A"}
class B extends D with E { override def m = super.m + "B"}
class C extends E with H { override def m = super.m + "C"}
val a = new A; val b = new B; val c = new C
println(a.m); println(b.m); println(c.m)
```

Give the output of running the above code as well as the *linearization* of classes and traits. You may ignore traits and classes pre-defined in Scala, such as `AnyRef` and `Any`. (12 marks)

Question 6

This question is about implicits, iterators, and covariant and contravariant types.

(a) Any *rational number* can be represented as a pair of integers, a numerator and a denominator. For instance, $3/7$ can be represented by the pair $(3, 7)$. Integers, such as $47, 35, \ldots$, are also rational numbers, represented as $(47, 1), (35, 1), \ldots$. Using *implicits*, write a Scala program that multiplies rational numbers, even if they are passed as integers to the program. (You do not need to consider handling invalid inputs such as $(2, 0)$.) For instance, the following Scala code should compile successfully:

```
val r1 = new Rational(3, 7); val r2 = new Rational(-11, -5);
println(r1 * r2); println(r1 * 3); println(7 * r1)
```

(6 marks)

(b) A *triangular number* is a number that counts the objects that can form an equilateral triangle. The nth triangular number is the number of dots in a triangle with n dots on a side. More specifically, it is the sum of the n natural numbers from 1 to n, which is given by the following mathematical expression: $n * (n + 1)/2$. The sequence of triangular numbers is, then: $0, 1, 3, 6, 10, 15, 21, 28, \ldots$. Using *iterators*, write a Scala program that generates the sequence of triangular numbers. (6 marks)

(c) Indicate the *variance* of the following two Scala data structures: *Lists* and *Arrays*.

(2 marks)

(d) Complete the missing parts (the parts printed as dots "..." in the Scala program) with the corresponding type annotations (for instance, for invariant, covariant, or contravariant types) so that the Scala program compiles correctly.

```
class Drink {}
class Juice extends Drink {}
class Beer extends Drink {}
class VendingMachine[...](val currentItem:Option[...], items:List[...]) {
  def this(items:List[...]) = this(None, items)
  def dispenseNext():VendingMachine[...] =
    items match {
      case Nil => {
        if (currentItem.isDefined) new VendingMachine(None, Nil)
        else this
      }
      case t :: ts => { new VendingMachine(Some(t), ts) }
    }
  def addAll[...](newItems:List[...]):VendingMachine[...] =
    new VendingMachine(items ++ newItems)
}
val VM1 = new VendingMachine(List(new Juice, new Juice))
val VM2 = VM1.addAll(List(new Beer))
```

(6 marks)

Question 7

This question is about Abstract Data Types (ADTs).

(a) What are *Abstract Data Types* (ADTs)? (1 mark)

(b) What is the difference between the *axiomatic* and the *constructive* approaches to the specification of ADTs? (1 mark)

For the rest of this question we will consider a *queue* ADT. Informally, a queue is a collection of data kept in sequence. Data can be added at one end of the sequence, called its *tail*, and deleted and retrieved at the other end of the sequence, called its *head*. The particular queue ADT specification we will consider has the following five operations:

- *create* returns a new empty queue;
- *head* takes a queue and returns the item at the head when the queue is not empty, or the message "Empty queue" when empty;
- *add* takes and item and a queue and returns a queue with that item added at the tail;
- *delete* takes a queue and returns the queue without the item at the head when the queue is not empty, or the message "Empty queue" when empty; and
- *empty* returns *true* when the queue is empty and *false* when is not empty.

(c) Using the *axiomatic* approach and assuming the existence of the sets below:

- Q, the set of queues;
- I, the set of items;
- B, the set of Boolean values, *true* and *false*;
- M, the set of message values, consisting of the single element "Empty queue".

give the *syntax* for this ADT specification. (6 marks)

(d) Using the *axiomatic* approach, give the *semantics* of the above ADT operations. (12 marks)

Question 8

This question is about design patterns.

(a) What is the *Observer* design pattern? To which kind of patterns does it belong? (2 marks)

(b) Give and describe the parts (classes, traits, etc.) for the Observer design pattern. (4 marks)

(c) Give the generic UML diagram for the Observer design pattern. (4 marks)

(d) For the following five design patterns, say when would you use them and to which kind of patterns they belong: Command, Strategy, Adapter, State, and Visitor. (10 marks)