# COMPUTER SCIENCE TRIPOS Part IB

Monday 4 June 2018 1.30 to 4.30

COMPUTER SCIENCE Paper 4

*Answer* ***five*** *questions:* ***up to four*** *questions from Section A, and* ***at least one*** *question from Section B.*

*Submit the answers in five* ***separate*** *bundles, each with its own cover sheet. On each cover sheet, write the numbers of* ***all*** *attempted questions, and circle the number of the question attached.*

**You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator**

STATIONERY REQUIREMENTS
*Script paper*
*Blue cover sheets*
*Tags*

SPECIAL REQUIREMENTS
*Approved calculator permitted*

## SECTION A

### 1 Programming in C

(*a*) The following function is specified to return the quotient of two integers, returning zero when the answer is undefined.

```
#include <stdint.h>
#include <limits.h>

int64_t divide(int64_t x, int64_t y) {
  return x / y;
}
```

(*i*) Identify two bugs in this program.

(*ii*) Write a correct version of this program.

[6 marks]

(*b*) The `strlen` function takes a valid C string as an argument, and returns the length of the string up to and not including the first null character. An (erroneous) implementation is given below:

```
#include <stddef.h>

size_t strlen(const char *s) {
  size_t i;
  while (s[i] >= 0)
    i++;
  return i;
}
```

(*i*) Find two errors in this program.

(*ii*) Give a correct implementation of this function.

[6 marks]

(*c*) Write a function with the prototype

```
void rotate(int len, int *array, int k)
```

which rotates its input `k` elements to the right. E.g., if the input `array` is the array `[0, 1, 2, 3, 4, 5]`, then the call `rotate(6, array, 2)` should result in `array` being modified to `[4, 5, 0, 1, 2, 3]`. Assume the array length is passed in the `len` argument and $0 \leq$ `k` $<$ `len`. [8 marks]

## 2 Programming in C

Consider the following structure declaration for a linked list in C:

```
struct node {
  int data;
  struct node* tail;
};
typedef struct node Node;
```

We represent linked lists as pointers to `Node` structs. Empty lists are represented with the null pointer. Non-empty lists are represented as a valid pointer to a `Node`, with the head data in the `data` field of the struct and the tail in the `tail` field of the struct. All lists will be assumed to be non-cyclic.

(*a*) Write a function to add an element to the head of the list with the following prototype.

```
Node *cons(int n, Node *tail);
```

[5 marks]

(*b*) Write a function to free the memory associated with a linked list, with the following prototype:

```
void free_list(Node *list);
```

[5 marks]

(*c*) Write a function to do an in-place list reversal. It should take a list as an argument, and return a list with the elements reversed. This function may not do any heap allocation, but may modify its input. It should have the following prototype:

```
Node *reverse(Node *list);
```

[10 marks]

## 3 Compiler Construction

This question covers a wide variety of topics in the interdisciplinary subject of Compiler Construction.

(*a*) What is the difference between lexical analysis and parsing? [2 marks]

(*b*) Give an example of an ambiguous context-free grammar together with a demonstration that it is indeed ambiguous. [3 marks]

(*c*) In the context of functional programming, describe the concept of a *closure* and how it is used. [3 marks]

(*d*) Can any recursive function be transformed into a tail-recursive function? Briefly explain you answer. [3 marks]

(*e*) Describe one advantage and one disadvantage for using *reference counting* as a technique for memory management. [2 marks]

(*f*) For what kinds of programming languages might you employ *static links* on the stack? Explain your answer. [3 marks]

(*g*) Consider an optimisation that replaces any expression of the form

$$0 \times e$$

with 0, where $e$ is an arbitrary expression. Why might this rule be useful in a compiler's optimisation phase? Is it always correct? [4 marks]

## 4 Compiler Construction

Suppose that we are to implement a compiler for the following simple, strongly-typed language with types $t$, expressions $e$, and programs $p$.

$$
\begin{array}{rcll}
t & ::= & \textbf{int} & \\
  & \mid & t * t & \text{(product type)} \\
  & & & \\
e & ::= & n & \text{(integer)} \\
  & \mid & \textbf{?} & \text{(read integer input by user)} \\
  & \mid & e + e & \text{(addition)} \\
  & \mid & e - e & \text{(subtraction)} \\
  & \mid & (e, e) & \text{(pair)} \\
  & \mid & \textbf{fst}\ e & \text{(first projection)} \\
  & \mid & \textbf{snd}\ e & \text{(second projection)} \\
  & \mid & f(e) & \text{function application} \\
  & \mid & \textbf{let}\ x : t = e\ \textbf{in}\ e\ \textbf{end} & \text{(let binding)} \\
  & & & \\
p & ::= & e & \\
  & \mid & \textbf{fun}\ f(x : t) : t = e\ \textbf{;}\ p & \text{(function definition, recursion allowed)}
\end{array}
$$

In the above $x$ and $f$ range over identifiers. For example, here is a simple program:

```
fun swap (p : int * int) : int * int = (snd p, fst p) ;

fun swizzle (p : int * (int * int)) : (int * int) * int =
     (swap (snd p), fst p) ;

swizzle (?, (?, ?))
```

You are asked to implement this language on a stack machine **that has no heap**. All stack entries are simple words (integers or pointers). Hint: consider using type information.

(*a*) Describe how your compiler will use the stack to implement function calls and returns. Describe any auxiliary pointers that you might need. Is there anything about the language above that makes this especially easy? [5 marks]

(*b*) Describe how you allocate space on the stack for a value of type $t$. [5 marks]

(*c*) Describe how your compiler will implement expressions of the form $(e_1,\ e_2)$. Explain how the order of evaluation (left-to-right, or right-to-left) impacts your choices. [5 marks]

(*d*) Describe how your compiler will implement expressions of the form **fst** $e$ and **snd** $e$. [5 marks]

## 5 Further Java

A social network can be represented as an undirected graph where vertexes represent people and edges represent bidirectional friendship relationships between two people. A developer constructs a simple representation of a social network using instances of the following Java class:

```
public class Person implements Serializable {
  private final long accountId;
  private String fullname;
  private Set<Person> friends;
  public Person(String fullname) { ... }
  public void addFriends(Set<Person> friends) { ... }
}
```

(*a*) The developer wishes to ensure that all public methods for `Person` are thread-safe. Describe what thread-safety means in this context. [1 mark]

(*b*) Write a thread-safe implementation of the constructor for `Person`. You must ensure no two instances of `Person` share the same `accountId`. You may create additional private fields, methods or inner classes in `Person`. [4 marks]

(*c*) Write a thread-safe implementation of `addFriends` which uses fine-grained locking to atomically establish new bidirectional friendship links between people in the social network. [5 marks]

(*d*) The developer wishes to serialize a copy of the graph of `Person` objects to disk. Without modifying the definition of `Person`, write an implementation of a non-thread-safe static method `void SocialNet.save(Set<Person> everyone, ObjectOutputStream oos)` which will write a *single* copy of all people in `everyone` to `oos`. Hint: `oos.writeObject(obj)` saves a copy of all objects reachable via references from `obj` and safely handles any cycles of references. [6 marks]

(*e*) Give two reasons why it is useful to store a single copy of each `Person` object in Part (*d*). [2 marks]

(*f*) Describe in words the modifications you would need to make if there are multiple threads attempting to add friendship links to the social network concurrently with the execution of `SocialNet.save`. [2 marks]

## 6 Security

(*a*) A Linux cloud server used by your team has the following discretionary access-control setup:

```
$ getent group admin users
admin:*:9001:alice
users:*:9002:alice,bobby,carla
$ ls -ld . * */*
drwxr-xr-x  3 carla users     4096 Apr  2  2017 .
-rwsr--r-x  1 bobby admin   241859 Jan  1  2013 proedit
-r--rw--w-  1 bobby admin     6355 Jul 24  2016 readme.txt
-rw----r--  1 carla admin     1459 Jun 12  2016 runtime.cfg
dr--r-xr-x  2 bobby users     4096 Jul 23  2016 src
-rw-r--r--  1 bobby users    26339 Apr 28  2018 src/code.c
-r--rw----  1 alice admin     6701 Jan 23  2017 src/code.h
```

The file `proedit` is a normal text editor, which allows its users to open, edit, save and execute files.

Copy and complete the access-control matrix illustrated below, such that it shows for each of the above five files, whether `alice`, `bobby`, or `carla` are able to obtain, directly or indirectly, read (R) or replace (W) access to its contents. Underline any access that can only be obtained through elevated rights.

| | proedit | readme.txt | runtime.cfg | src/code.c | src/code.h |
|---|---|---|---|---|---|
| alice | | | | | |
| bobby | | | | | |
| carla | | | | | |

[12 marks]

(*b*) Several Linux file systems extend the POSIX file permission bits with an access-control list mechanism defined in POSIX.1e Draft 17. Explain four significant differences between these Linux ACLs and those of Windows NTFS.

[8 marks]

## 7 Security

(*a*) An application process receives information via a UDP packet over a wired Ethernet LAN connection. If the packet carries a source port number below 1024, under which conditions can the information be trusted? [6 marks]

(*b*) What is a *UDP-based amplification attack* and why are similar attacks far less practical via TCP? [6 marks]

(*c*) Name and briefly explain *four* techniques that the designers of C compilers and operating system kernels can implement to reduce the risk of stack-based buffer-overflow attacks. [4 marks]

(*d*) How can an implementation of the C function `strcmp()` cause a vulnerability to a side-channel attack, and how can this be mitigated? [4 marks]

## SECTION B

### 8 Semantics of Programming Languages

Consider the following syntax up to alpha equivalence, where $n$ ranges over natural numbers, $x$ over a set of variables, and (as usual) $x$ is binding in $e$ in $\mathbf{fn}\, x \Rightarrow e$.

$$\begin{array}{ll} \textit{expressions},\ e ::= & n \mid x \mid \mathbf{fn}\, x \Rightarrow e \mid e\, e' \\ \textit{values},\ v ::= & n \mid x \mid \mathbf{fn}\, x \Rightarrow e \end{array}$$

(*a*) Define free variables $\mathrm{fv}(e)$ and capture-avoiding substitution $\{e/z\}e'$. [3 marks]

(*b*) Define a left-to-right call-by-value reduction relation $e \longrightarrow e'$. [3 marks]

Implementing a language using substitution is inefficient, as each substitution has to traverse a potentially large subterm. Consider the following proposal for an abstract machine for this language using environments $E$, lists of variable/value pairs.

$\boxed{\langle E, e\rangle \longrightarrow \langle E', e'\rangle}$

$$\frac{(x, v) \in E}{\langle E, x\rangle \longrightarrow \langle E, v\rangle}\ \text{LOOKUP}$$

$$\frac{x \notin \mathrm{dom}(E) \cup \mathrm{fv}(\mathrm{range}(E)) \cup \mathrm{fv}(v)}{\langle E, (\mathbf{fn}\, x \Rightarrow e)\, v\rangle \longrightarrow \langle (x, v) :: E, e\rangle}\ \text{FN}$$

$$\frac{\langle E, e_1\rangle \longrightarrow \langle E', e_1'\rangle}{\langle E, e_1\, e_2\rangle \longrightarrow \langle E', e_1'\, e_2\rangle}\ \text{APP\_LEFT}$$

$$\frac{\langle E, e_2\rangle \longrightarrow \langle E', e_2'\rangle}{\langle E, v_1\, e_2\rangle \longrightarrow \langle E', v_1\, e_2'\rangle}\ \text{APP\_RIGHT}$$

(*c*) Give the sequence of abstract-machine reduction steps, including the configurations and the names of the rules used, for the initial configuration below. You need not give full derivation trees.

$$\langle [], ((\mathbf{fn}\, x \Rightarrow (\mathbf{fn}\, y \Rightarrow x\, y))\, (\mathbf{fn}\, z \Rightarrow z))\, 3\rangle$$

[5 marks]

(*d*) Explain, with a concrete example and its reduction sequence, what could go wrong if the premise of FN had been omitted. [5 marks]

(*e*) Write $\{E\}e$ for the iterated substitution defined by

$$\begin{array}{lcl} \{[]\}e & = & e \\ \{(x, v) :: E\}e & = & \{E\}(\{v/x\}e) \end{array}$$

Prove that $\{E\}(e_1\, e_2) = (\{E\}e_1\, \{E\}e_2)$. [4 marks]

## 9 Semantics of Programming Languages

Consider the following language with higher-order functions and mutable global references. Here $l$ ranges over mutable location names, that can hold arbitrary values $v$, $n$ ranges over natural numbers, and $x$ ranges over immutable variable names.

$$\begin{aligned} e &::= \; n \mid l \mid x \mid \mathbf{fn}\, x : T \Rightarrow e \mid e\, e' \mid e := e' \mid !e \\ v &::= \; n \mid l \mid \mathbf{fn}\, x : T \Rightarrow e \end{aligned}$$

Suppose it has a standard left-to-right call-by-value operational semantics. You need not state this semantics.

The definition of $\Gamma \vdash_{\mathit{eff}} e : T$ below, where an effect *eff* is a subset of $\{\mathbf{R}, \mathbf{W}\}$, is a flawed attempt to statically compute a sound approximation of the possible dynamic side-effects of expressions. Such an analysis is *sound* if *eff* contains **R** and/or **W** whenever there is any execution of $\langle e, s \rangle$, for any store $s$ that is well-typed with respect to $\Gamma$, that involves (respectively) reading and/or writing the store.

Function types are annotated with the latent effects that may occur when the function is applied: $T ::= \; \mathbf{int} \mid T \rightarrow_{\mathit{eff}} T' \mid T\, \mathbf{ref}$

$\boxed{\Gamma \vdash_{\mathit{eff}} e : T}$

$$\frac{}{\Gamma \vdash_{\{\}} n : \mathbf{int}}\ \text{NUM} \qquad \frac{l : T\,\mathbf{ref} \in \Gamma}{\Gamma \vdash_{\{\mathbf{R},\mathbf{W}\}} l : T\,\mathbf{ref}}\ \text{LOC} \qquad \frac{x : T \in \Gamma}{\Gamma \vdash_{\{\}} x : T}\ \text{VAR}$$

$$\frac{\Gamma, x : T \vdash_{\mathit{eff}} e : T'}{\Gamma \vdash_{\{\}} \mathbf{fn}\, x : T \Rightarrow e : T \rightarrow_{\mathit{eff}} T'}\ \text{FN} \qquad \frac{\begin{array}{c}\Gamma \vdash_{\mathit{eff}} e : T_1 \rightarrow_{\mathit{eff}''} T_2 \\ \Gamma \vdash_{\mathit{eff}'} e' : T_1\end{array}}{\Gamma \vdash_{\mathit{eff} \cup \mathit{eff}'} e\, e' : T_2}\ \text{APP}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathit{eff}} e : T\,\mathbf{ref} \\ \Gamma \vdash_{\mathit{eff}'} e' : T\end{array}}{\Gamma \vdash_{\mathit{eff} \cup \mathit{eff}' \cup \{\mathbf{W}\}} e := e' : T}\ \text{ASSIGN} \qquad \frac{\Gamma \vdash_{\mathit{eff}} e : T\,\mathbf{ref}}{\Gamma \vdash_{\{\mathbf{R}\}} !e : T}\ \text{DEREF}$$

(*a*) There are three flaws in the above rules, which make them either not sound or an unnecessarily coarse approximation. Explain each flaw, giving a corrected rule for each and an example that shows the problem (assuming the other flaws are fixed). [15 marks]

(*b*) In the system above, functions have to be applied to arguments of exactly the expected type. Define a subtype relation $T <: T'$ and subsumption rule that would let function arguments be used even if they have fewer (latent) effects than those anticipated by the function. [5 marks]

**END OF PAPER**