UNIVERSITY OF CAMBRIDGE

COMPUTER SCIENCE TRIPOS Part IB

Monday 30 May 2016 1.30 to 4.30

COMPUTER SCIENCE Paper 3

*Answer **five** questions.*

*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

1 Programming in C and C++

(*a*) Explain the difference between `'x'` and `"x"` when used as constants in C. Describe the memory representation of both values. [4 marks]

(*b*) Consider the following C program:

```
void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}

int main(int argc, char **argv) {
  int x = 0;
  int y = 1;
  swap(x, y);
  assert(x == 1);
  return 0;
}
```

Briefly explain the role of the `assert` statement and why this program will trigger an `assert` failure when executed. Supply *two* modified versions of the program that alter the `swap` function definition and, if necessary, its calls, to avoid this `assert` failure. One version should be in C, and the other should use C++ language features. [4 marks]

(*c*) Describe the address-space layout (highlighting four areas of memory) of a typical compiled x86 C program, and how each of these areas are used by C constructs. [8 marks]

(*d*) Briefly explain what *undefined behaviour* is in the C standard. Under what circumstance(s) would calling the following C function result in undefined behaviour?

```
int32_t divide(int32_t a, int32_t b)
{
  return a / b;
}
```

[4 marks]

2 Programming in C and C++

(*a*) Consider *unspecified behaviour* in C.

(*i*) Define what unspecified behaviour means in the C standard and give two examples of such behaviour. [3 marks]

(*ii*) Briefly explain why it is important to have unspecified behaviour in the definition of the C language. [1 mark]

(*b*) Compare and contrast the `struct` and `union` keywords in C, supplying an example of a situation where it would be more appropriate to use a `union` rather than a `struct`. [4 marks]

(*c*) Explain the following C or C++ language concepts. You may find it helpful to use short code fragments or diagrams to illustrate your answer.

(*i*) The `virtual` keyword used to qualify a C++ member function and its impact on generated code. [4 marks]

(*ii*) The role of the C preprocessor in the source-code compilation cycle, and why it is a useful tool for debugging. [4 marks]

(*iii*) Templated functions in C++, giving one benefit and one drawback of using them compared with using a `void*` function in C. [4 marks]

3 Compiler Construction

Programming answers should be written in some notation approximating SML or OCaml.

(*a*) Describe what is meant by *tail recursion*. [4 marks]

(*b*) Eliminate tail recursion from `foldl` given below. Explain your answer.

```
(*
    foldl : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
*)
let rec foldl f accu l =
  match l with
      [] -> accu
  | a::l -> foldl f (f accu a) l
```

[8 marks]

(*c*) Eliminate tail recursion from the following mutually tail-recursive functions. Explain your answer.

```
let rec is_even n =
    if n = 0
    then true
    else is_odd (n - 1)

and is_odd n =
    if n = 0
    then false
    else is_even(n - 1)
```

[8 marks]

4 Compiler Construction

Consider writing a compiler for a simple language of expressions given by the following grammar,

$$
\begin{array}{rcll}
e & ::= & n & \text{(integer)} \\
 & | & ? & \text{(read integer input from user)} \\
 & | & e + e & \text{(addition)} \\
 & | & e - e & \text{(subtraction)} \\
 & | & e * e & \text{(multiplication)} \\
 & | & (e, e) & \text{(pair)} \\
 & | & \texttt{fst}\ e & \text{(first projection)} \\
 & | & \texttt{snd}\ e & \text{(second projection)}
\end{array}
$$

(*a*) Describe the tasks that should be carried in implementing a front end for this language and any difficulties that might be encountered. [5 marks]

(*b*) Suppose that the target virtual machine is stack-oriented and that the stack elements are integer values, and addresses can be stored as integers. Explain which other features are required in such a virtual machine. Invent a simple language of instructions for such a machine and show how it would be used to implement each of the expressions. [10 marks]

(*c*) Suppose that the following rules are proposed as possible optimizations to be implemented in your compiler.

expression	simplifies to	expression
$(\texttt{fst}\ e,\ \texttt{snd}\ e)$	$\rightarrow$	e
$\texttt{fst}\ (e_1,\ e_2)$	$\rightarrow$	e_1
$\texttt{snd}\ (e_1,\ e_2)$	$\rightarrow$	e_2

Describe how you could implement these rules so that the simplifications are made *only* when the program's semantics is correctly preserved. [5 marks]

5 Concepts in Programming Languages

(*a*) Explain what is meant by a *monad* in a programming language, giving the two fundamental operations of a monad along with their types. [3 marks]

(*b*) Consider the use of a monad for input-output. For the purposes of this question, take the `IO` monad as including two operations `readint` and `writeint` which respectively read integers from *stdin* and write integers to *stdout*. Give the types of these operators. [2 marks]

(*c*) Assume `MLreadint` and `MLwriteint` are primitives with side effects for input-output and consider the ML expression *add1* of type `int`:

```
let val x = MLreadint() in MLwriteint(x+1); x end
```

(*i*) Give an equivalent expression which uses the `IO` monad instead of side-effects, and state its type. [3 marks]

(*ii*) Give a function `run2diff` which can be applied to your answer to part (*c*)(*i*). When so applied it should give a value in the `IO` monad which corresponds to ML code that runs *add1* twice and returns the difference between the values read. [4 marks]

(*d*) State what happens when attempting to compile and execute the following Java fragment (explaining the origin of any error messages or exceptions which might arise).

```
Object n = new Integer(42), o = new String("Whoops");
Object [] v;
Integer [] w = new Integer[10];
v = w;
v[4] = n;
v[5] = o;
```

[4 marks]

(*e*) Consider the Java code:

```
Object n = new Integer(42);
ArrayList<? extends Object> v1;
ArrayList<Object> v2;
ArrayList<Integer> w = new ArrayList<>(10);
```

Explain any differences in behaviour between assignments `v1 = w` and `v2 = w` and also between method calls `v1.set(4,n)` and `v2.set(4,n)`. [4 marks]

6 Further Java

(*a*) Describe the operation of `wait()` and `notifyAll()`. Ensure that your answer explains when locks are acquired and released. [5 marks]

(*b*) A *future* is a mechanism to store the eventual result of a computation done in another thread. The idea is that the computation is run asynchronously and the calling thread only blocks if it tries to use a result that hasn't been computed yet. An example program using a future is shown below.

```
Future<String> f = new Future<String>() {
    @Override
    public String execute() {
       // ...long running computation...
       return data;
    };

// ...

String result = f.get(); // blocks if execute() unfinished
```

Use `wait()` and `notifyAll()` to provide an implementation of the `Future` class that would work with the example program above. [10 marks]

(*c*) Give one potential advantage and one potential disadvantage of using `notify()` instead of `notifyAll()`. [2 marks]

(*d*) Would it have been beneficial to use `notify()` instead of `notifyAll()` in your implementation? Justify your answer. [3 marks]

7 Prolog

In this question you should ensure that your predicates behave appropriately with backtracking and avoid over-use of cut. You should provide an implementation of any library predicates used. You **may not** make use of extra-logical built-in predicates such as `findAll`. Minor syntactic errors will not be penalised.

(*a*) Explain the operation of cut (`!`) in a Prolog program. [2 marks]

(*b*) Rewrite `choose` without using cut. [2 marks]

```
choose(0,_,[]) :- !.
choose(N,[H|T],[H|R]) :- M is N-1, choose(M,T,R).
choose(N,[_|T],R) :- choose(N,T,R).
```

(*c*) Explain the operation of `not` (also written as `\+`) in a Prolog program. [1 mark]

(*d*) Rewrite `chooseAll` without using `not` and cut (`!`). [10 marks]

```
chooseAll(N,L,Res) :- chooseAll(N,L,[],Res).
chooseAll(N,L,Seen,Res) :- choose(N,L,R),
                           not(member(R,Seen)), !,
                           chooseAll(N,L,[R|Seen],Res).
chooseAll(_,_,Res,Res).
```

(*e*) What is *Last Call Optimisation* and why is it beneficial? [3 marks]

(*f*) Rewrite `pos` to enable Last Call Optimisation. [2 marks]

```
pos([],[]).
pos([H|T],[H|R]) :- H >= 0, pos(T,R).
pos([H|T],R) :- H < 0, pos(T,R).
```

8 Software Engineering

Discuss the contribution and the relative value of the following aspects of the modern development environment. Illustrate with examples from your group project, or from experience you gained working for a commercial software developer, or both. In each case, would you discard this feature if your employer let you, or insist on retaining it (even covertly) should your employer not value it? Explain your reasons.

(a) Dividing a project into short development episodes or sprints.

(b) Project progress visualisation tools such as PERT and GANTT charts.

(c) Automated regression testing tools.

(d) Source code management tools.

(e) Scrumming.

[4 marks each]

END OF PAPER