INSTRUCTIONS TO CANDIDATES

1. Note that ALL QUESTIONS ARE COMPULSORY.

2. DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS. Take note of this in allocating time to questions.

3. This is an OPEN BOOK examination: notes and printed material are allowed, and USB sticks, but no electronic devices.

4. CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

Convener: D. K. Arvind
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY
1. (a) Write a function \( f \) : [Int] -> Bool that, given a non-empty list of numbers, returns True if each successive number (except the first) is at least twice its predecessor in the list. The function should give an error if applied to the empty list. For example:

\[
\begin{align*}
    f \ [1,2,7,18,47,180] &= \text{True} \\
    f \ [17] &= \text{True} \\
    f \ [1,3,5,16,42] &= \text{False} \\
    f \ [1,2,6,6,13] &= \text{False}
\end{align*}
\]

Use basic functions, list comprehension, and library functions, but not recursion. Credit may be given for indicating how you have tested your function. [16 marks]

(b) Write a second function \( g \) : [Int] -> Bool that behaves like \( f \), this time using basic functions and recursion, but not list comprehension or library functions. Credit may be given for indicating how you have tested your function. [16 marks]
2. (a) Write a function \( p : \{\mathbb{Int}\} \rightarrow \mathbb{Int} \) that computes the sum of the cubes of the positive numbers in a list. For example:

\[
\begin{align*}
p \ [-13] &= 0 \\
p \ [\ ] &= 0 \\
p \ [-3,3,1,-3,2,-1] &= 36 \\
p \ [2,6,-3,0,3,-7,2] &= 259 \\
p \ [4,-2,-1,-3] &= 64
\end{align*}
\]

Use basic functions, list comprehension, and library functions, but not recursion. Credit may be given for indicating how you have tested your function. \([12 \text{ marks}]\)

(b) Write a second function \( q : \{\mathbb{Int}\} \rightarrow \mathbb{Int} \) that behaves like \( p \), this time using basic functions and recursion, but not list comprehension or library functions. Credit may be given for indicating how you have tested your function. \([12 \text{ marks}]\)

(c) Write a third function \( r : \{\mathbb{Int}\} \rightarrow \mathbb{Int} \) that also behaves like \( p \), this time using the following higher-order library functions:

\[
\begin{align*}
\text{map} &: \ (a \rightarrow b) \rightarrow \{a\} \rightarrow \{b\} \\
\text{filter} &: \ (a \rightarrow \text{Bool}) \rightarrow \{a\} \rightarrow \{a\} \\
\text{foldr} &: \ (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow \{a\} \rightarrow b
\end{align*}
\]

Do not use recursion or list comprehension. Credit may be given for indicating how you have tested your function. \([12 \text{ marks}]\)
3. The following data type represents arithmetic expressions over a single variable:

```haskell
data Expr = X          -- variable
        | Const Integer -- integer constant
        | Expr :+: Expr -- addition
        | Expr :-: Expr -- subtraction
        | Expr :*: Expr -- multiplication
        | IfLt Expr Expr Expr Expr -- conditional expression
```

`IfLt p q r s` represents the expression that would be written in Haskell as `if p < q then r else s`.

The template file includes a function `showExpr :: Expr -> String` which converts expressions into a readable format, and code that enables QuickCheck to generate arbitrary values of type `Expr`, to aid testing.

(a) Write a function `eval :: Expr -> Integer -> Integer`, which given an expression and the value of the variable `X` returns the value of the expression. For example,

- `eval (X :+: (X :*: Const 2)) 3 = 9`
- `eval (X :-: (X :*: Const 3)) 0 = 0`
- `eval (X :-: (X :*: Const 3)) 7 = -14`
- `eval (X :+: X) 2 = 4`
- `eval (Const 15 :+: (Const 7 :*: (X :-: Const 1))) 0 = 8`
- `eval (X :-: (X :+: X)) 4 = -4`

Credit may be given for indicating how you have tested your function. [16 marks]

(b) Write a function `protect :: Expr -> Expr` that avoids negative results by “guarding” all uses of subtraction `p :-: q` with a check for the value of `p` being less than the value of `q`. In this case the result should be `0`. Do not attempt to simplify the result by omitting tests that appear to be unnecessary. For example,

- `protect (X :-: (X :*: Const 2)) = (X :-: (X :*: Const 2))`
- `protect (X :-: (X :*: Const 3)) = IfLt X (X :*: Const 3) (Const 0) (X :-: (X :*: Const 3))`
- `protect (X :+: X) = X :+: X`
- `protect (Const 15 :+: (Const 7 :*: (X :-: Const 1))) = Const 15 :+: (Const 7 :*: IfLt X (Const 1) (Const 0) (X :-: (X :*: Const 1)))`
- `protect (X :-: (X :+: X)) = IfLt X (X :+: X) (Const 0) (X :-: (X :+: X))`

*QUESTION CONTINUES ON NEXT PAGE*
In order to further guard against negative results, negative constants should not be allowed. This part of `protect` is already supplied in the template file. Furthermore, a negative value of \(X\) should not be used as an argument to `eval`.

When evaluated, the protected versions of these expressions give the following results:

\[
\begin{align*}
\text{eval} \left( \text{protect} \left( X :+\: (X :*: \text{Const} 2) \right) \right) 3 &= 9 \\
\text{eval} \left( \text{protect} \left( X :{-}\: (X :*: \text{Const} 3) \right) \right) 0 &= 0 \\
\text{eval} \left( \text{protect} \left( X :{-}\: (X :*: \text{Const} 3) \right) \right) 7 &= 0 \\
\text{eval} \left( \text{protect} \left( X :+\: X \right) \right) 2 &= 4 \\
\text{eval} \left( \text{protect} \left( \text{Const} 15 :+\: (\text{Const} 7 :*: (X :{-}\: \text{Const} 1)) \right) \right) 0 &= 15 \\
\text{eval} \left( \text{protect} \left( X :{-}\: (X :+\: X) \right) \right) 4 &= 0
\end{align*}
\]

Credit may be given for indicating how you have tested your function. \([16 \text{ marks}]\)