Instructions
Answer question #1 and any three others. (If you answer more, only the first four will count.)
Point values are as indicated. Please try to make your answers neat and coherent. Remember, if
we can’t read it, it’s wrong. Partial credit will be given, so try to put something down for each
question (a blank answer always gets 0 points!).

1. (1 point)
The C++ programming language is based upon what programming language?
(a) C
(b) B
(c) BCPL
(d) CPL

2. (a) (18 points)
Recall that the Scheme map function, called as \( \text{map} \ f \ L \) applies function \( f \) to every value in
list \( L \), returning a list of result values. In Scheme it is common to return \#f\ if a function is
undefined for a particular input value. Hence if we use map with a function that may return \#f\ as an error indication, the list of results may be “polluted” by useless \#f\ values.

Create a variant of map, filtered-map, that is called as \( \text{filtered-map} \ f \ L \). filtered-
map discards any result values that are \#f\, returning a list of non-\#f\ results.

(b) (15 points)
If we use filtered-map of part (a) with the output of the subsets function we may run into
the problem that many sublists may be produced that will be discarded when function \( f \) is
applied because \#f\ is produced. Assume that any function that is mapped to the output of
subsets is montone (just as was the case for filtered-subsets). This means that if a
mapped function produces \#f\ for a set \( S \), it will also produce \#f\ for all larger sets that contain \( S \).

Create a variant of filtered-map, map-filtered-subsets, called as
\( \text{map-filtered-subsets} \ f \ L \), that creates subsets of list \( L \) and filters then using \( f \) as they
are generated. The result produced by map-filtered-subsets is the list of all non-\#f\ results produced by mapping function \( f \) to \( L \)’s subsets.
3. (a) (18 points)
Let \( L \) be a list containing \( 2^n \) integers, for \( n \geq 0 \). That is, \( L \) may be of length 1, or 2, or 4, or 8, etc.
Write a Scheme function \( \text{balanced-tree} L \) transforms list \( L \) into a balanced binary tree containing the values of \( L \). At all levels, the left subtree will contain the first half of the tree’s values, while the right subtree will contain the second half of the tree’s values.
For example, \( \text{balanced-tree} '((1 2 3 4)) \) evaluates to \(((1 . 2) . (3 . 4))\).
If list \( L \) contains a single value, \( \text{balanced-tree} \) returns that value. Thus \( \text{balanced-tree} '((1)) \) evaluates to 1.

(b) (5 points)
What does your implementation of \( \text{balanced-tree} \) do if list \( L \) is not equal to a power of 2 in length?

(c) (10 points)
Assume that we wish to sum the integers in a list \( L \) and that we have a compiler for MultiLisp available. Why would the balanced tree format produced by \( \text{balanced-tree} \) be more suitable than a list format if we wished to speed the summation computation for a long list of integers?

4. (a) (13 points)
Explain what \( \text{call/cc} \) (also known as \( \text{call-with-current-continuation} \)) does.

(b) (20 points)
What does the following Scheme expression compute? Why?

\[
\begin{align*}
\text{let ( (C '()))}
&\quad \left(\text{display (= 10 (call/cc (lambda (k) (set! C k) 10))))}\right) \\
&\quad \left(\text{C 20}\right)
\end{align*}
\]

5. (a) (18 points)
Most programming languages, including Scheme, execute a function call by evaluating the function’s parameters sequentially, then executing the function body with the evaluated actual parameters. A number of alternative parameter evaluation mechanisms are possible.

We know that if call by name is used, parameter evaluation is delayed until the parameter is actually used in the function body. In MultiLisp, the pcall mechanism evaluates all the actual parameters in parallel. Function execution begins when all the parameters are fully evaluated. If we transform actual parameters into future calls, we may begin executing the function body immediately, even before the actual parameters are fully computed.

For each of the three alternative parameter evaluation mechanism discussed (by name, pcall, and futures) suggest a plausible programming situation in which the alternative mechanism is preferable to ordinary sequential parameter evaluation. Make clear why the alternative mechanism is to be preferred.

(b) (15 points)
For each of the three alternative parameter evaluation mechanism discussed in part (a), are we guaranteed to always get the same result as if sequential evaluation had been used? If so, explain why. If not, outline the circumstances under which a different result might be obtained.