Compiler Construction

It is commonly suggested that Algol-60 call-by-name can be modelled by passing a function as a call-by-value parameter. Show how a program containing a definition

\[
\text{int } f(\text{int } x:\text{name}) \{ \ldots x \ldots x \ldots \}
\]

of \(f\) (where \(x\) occurs only in Rvalue context) and a call \(f(e)\) to \(f\) can be replaced by an equivalent definition and call using only call-by-value. [6 marks]

Most such explanations assume that the uses of \(x\) within \(f\) occur only in Rvalue context. However, Algol-60 also permits the equivalent of

\[
\text{int } g(\text{int } x:\text{name}) \{ \text{if (p) \{ \ldots x := x+1; x := -x; \ldots \}} \}
\text{return } x;
\]

and calls like \(g(a[k()])\) which, when \(p\) is \textbf{true}, would have the effect of calling \(k()\) five times and consequent access to five (possibly different) subscripts of array \(a[\].\) Develop your explanation for the first part of this question to cover also the case of a call-by-name parameter being used in both Lvalue and Rvalue contexts. [Hint: note that when \(p\) is \textbf{false} then the actual parameter to \(g\) need not be an Lvalue, so you may need two parameterless procedure arguments ("thunks").] [8 marks]

Using the previous part or otherwise, give a translation of a definition and call \(h(e)\) using call-by-value-result (Ada \textbf{in out} mode) with no uses of the address-of (\&) operator other than those involved in call-by-name. Your explanation is allowed to deviate from call-by-value-result by allowing side-effects in \(e\) to take place twice. [6 marks]