Halt-no-Input

A reduction proof is a style of proof common in computer science. It is often used to show the equivalence (or non-equivalence) of two classes of algorithms. Do not confuse reduction-style proofs with the "reduction" steps in a lambda calculus proof; these are different ideas.

A form of a reduction proof is an algorithm: we construct a function called a reducer. So keep mind that when you are asked produce a reduction-style proof, you're being asked to write code. You write the code for the reducer function.

When employed to prove undecidability (i.e., that a problem has no solution), the "direction" of the reduction is counterintuitive. Let `B` be the problem you intend to prove something about. Let `A` be a problem you already know something about (e.g., the Halting Problem). We then attempt to construct a reducer that converts problems of type `A` into problems of type `B` and assume the existence of an algorithm that solves instances of type `B`.

If we are successful in constructing a reducer (using ordinary code), the existence of the reducer demonstrates a contradiction: we already know that solving problems of type `A` is impossible. We have therefore proved, by reductio ad absurdum, that we could solve problems of type `B`; that we could solve `B` was the only fact we assumed to be true.

Prove that the Halt-no-Input problem is undecidable.

Halt-no-Input problem: given a program `P` that requires no input, does `P` halt?

Form of Expected Answer. Your answer should be in the form of a reducer (an algorithm). Write your answer (code) in the language of your choice.

Hint. Be careful to pay attention to the direction of the reduction. `Halt` should make an appearance in your proof. On which side of the reduction does it appear?