

ERIC BAILEY

# ADVENT OF CODE



# *Contents*

*Day 1: The Tyranny of the Rocket Equation*      5

*GAP Solution*      5

*Day 2: 1202 Program Alarm*      7

*Haskell Solution*      7

*Day 4: Secure Container*      9

*Haskell Solution*      9



# Day 1: The Tyranny of the Rocket Equation

Copy description

<https://adventofcode.com/2019/day/1>

## GAP Solution

```
<Day01.g 5a>≡
FuelRequiredModule := function( mass )
    return Int( Float( mass / 3 ) ) - 2;
end;;
```

$\text{fuel} := \text{mass} \setminus 3 - 2$

This definition is continued in chunks 5 and 6.  
Root chunk (not used in this document).

```
<Day01.g 5a>+≡
PartOne := function( )
    local input, line, mass, sum;;
    sum := 0;
    input := InputTextFile ( "./input/day01.txt" );
    line := ReadLine( input );
    repeat
        mass := Int( Chomp( line ) );
        sum := sum + FuelRequiredModule( mass );
        line := ReadLine( input );
    until line = fail or IsEndOfStream( input );
    return sum;
end;;
```

```
<Day01.g 5a>+≡
TotalFuelRequiredModule := function( mass )
    local fuel;;
    fuel := FuelRequiredModule( mass );
    if IsPosInt( fuel ) then
        return fuel + TotalFuelRequiredModule( fuel );
    else
        return 0;
    fi;
end;;
```

```
<Day01.g 5a>+≡
PartTwo := function( )
  local input, line, mass, sum;;
  sum := 0;
  input := InputTextFile ( "../input/day01.txt" );
  line := ReadLine( input );
  repeat
    mass := Int( Chomp( line ) );
    sum := sum + TotalFuelRequiredModule( mass );
    line := ReadLine( input );
  until line = fail or IsEndOfStream( input );
  return sum;
end;;
```

## Day 2: 1202 Program Alarm

Copy description

<https://adventofcode.com/2019/day/2>

### Haskell Solution

```
<Day02.hs 7a>≡
module Data.AOC19.Day02 where

import      Control.Arrow      (first, (»>))
import      Data.List          (find)
import      Data.Vector         (Vector, fromList, modify, toList, (!))
import qualified Data.Vector     as V
import      Data.Vector.Mutable (write)
import qualified Data.Vector.Mutable as MV
import      Text.Trifecta        (Parser, Result (..), comma, natural,
                                parseFromFile, parseString, sepBy)
```

This definition is continued in chunks 7 and 8.  
Root chunk (not used in this document).

```
<Day02.hs 7a>+≡
program :: Parser (Vector Int)
program = fromList . map fromInteger <$> (natural 'sepBy' comma)
```

```
<Day02.hs 7a>+≡
partOne :: IO Int
partOne =
  do res ← parseFromFile program "../input/day02.txt"
  case res of
    Nothing   → error "No parse"
    Just state → pure (V.head (runProgram (restoreGravityAssist state)))
```

```

⟨Day02.hs 7a⟩+≡
partTwo :: IO Int
partTwo =
  do res ← parseFromFile program "../../input/day02.txt"
  case res of
    Nothing    → error "No parse"
    Just state →
      do let n = V.length state - 1
         pure . maybe (error "Fail") (first (*100) »> uncurry (+)) $
           find (go state) (concatMap (zip [0..n] . repeat) [0..n])
  where
    go state (noun, verb) =
      19690720 = V.head (runProgram (restoreGravityAssist' noun verb state))

```

```

⟨Day02.hs 7a⟩+≡
restoreGravityAssist :: Vector Int → Vector Int
restoreGravityAssist = restoreGravityAssist' 12 2

```

```

⟨Day02.hs 7a⟩+≡
restoreGravityAssist' :: Int → Int → Vector Int → Vector Int
restoreGravityAssist' noun verb =
  modify (\v → write v 1 noun *> write v 2 verb)

```

```

⟨Day02.hs 7a⟩+≡
runProgram :: Vector Int → Vector Int
runProgram = go 0
  where
    go n state
      | state ! n == 99 = state
      | otherwise       = go (n + 4) $ step (toList (V.slice n 4 state))
    where
      step [1, x, y, dst] = modify (runOp (+) x y dst) state
      step [2, x, y, dst] = modify (runOp (*) x y dst) state
      step _               = state

    runOp f x y dst v = write v dst =< f <$> MV.read v x <*> MV.read v y

```

```

⟨Day02.hs 7a⟩+≡
example1 :: Vector Int
example1 =
  case parseString program mempty "1,9,10,3,2,3,11,0,99,30,40,50" of
    Success prog  → prog
    Failure reason → error (show reason)

```



## Day 4: Secure Container

Copy description

<https://adventofcode.com/2019/day/4>

### Haskell Solution

#### Input

My puzzle input was the range 236491-713787, which I converted into a list of lists of `digits`.

```
<Input 9a>≡  
input :: [[Int]]  
input = digits 10 <$> [236491 .. 713787]
```

This code is used in chunk 10b.

#### Part One

For part one, there must be two adjacent digits that are the same, i.e. there exists at least one `group` of `length`  $\geq 2$ .

```
<has a double 9b>≡  
any (( $\geq$  2) . length) . group
```

This code is used in chunk 9c.

It must also be the case that the `digits` never decrease, i.e. the password `isSorted`.

```
<Part One 9c>≡  
partOne :: Int  
partOne = length $ filter isPossiblePassword input  
where  
    isPossiblePassword :: [Int] → Bool  
    isPossiblePassword = liftM2 (&&) isSorted hasDouble  
  
    hasDouble :: Eq a ⇒ [a] → Bool  
    hasDouble = <has a double 9b>
```

This code is used in chunk 10b.

#### Part Two

For part two, the password still `isSorted`, but must also have a strict double, i.e. at least one `group` of `length`  $= 2$ .

```
<has a strict double 9d>≡  
any ((= 2) . length) . group
```

This code is used in chunk 10a.

```

⟨Part Two 10a⟩≡
partTwo :: Int
partTwo = length $ filter isPossiblePassword input
  where
    isPossiblePassword :: [Int] → Bool
    isPossiblePassword = liftM2 (&&) isSorted hasDouble

    hasDouble :: Eq a ⇒ [a] → Bool
    hasDouble = ⟨has a strict double 9d⟩

```

This code is used in chunk 10b.

### Full Solution

```

⟨Day04.hs 10b⟩≡
module Data.AOC19.Day04 where

import           Control.Monad      (liftM2)
import           Data.Digits        (digits)
import           Data.List          (group)
import           Data.List.Ordered  (isSorted)

```

⟨Input 9a⟩

⟨Part One 9c⟩

⟨Part Two 10a⟩

Root chunk (not used in this document).

# Chunks

$\langle \textit{Day01.g} \textcolor{red}{5a} \rangle$

$\langle \textit{Day02.hs} \textcolor{red}{7a} \rangle$

$\langle \textit{Day04.hs} \textcolor{red}{10b} \rangle$

$\langle \textit{has a double} \textcolor{red}{9b} \rangle$

$\langle \textit{has a strict double} \textcolor{red}{9d} \rangle$


$\langle \textit{Input} \textcolor{red}{9a} \rangle$

$\langle \textit{Part One} \textcolor{red}{9c} \rangle$

$\langle \textit{Part Two} \textcolor{red}{10a} \rangle$



*To-Do*

 Copy description . . . . .	5
 Copy description . . . . .	7
 Copy description . . . . .	9