CSU11022 – End of Year Examination

**Part 1**

To find the word "hello" in a word search, a common strategy is to first find the letter "h" and see if the letter "e" is adjacent to it. If it is, you would then check if the letter "l" is next to that, and repeat. I adopted a similar strategy.

When the word is going left to right, the row the word is in is stagnant, while the column each letter is in changes. A human would know when to stop searching for letters, but a computer wouldn't. To fix this, I first created a subroutine of my own called "stringLength" that would return the length of a string given its address. Then, I created a counter that would update every iteration. When this counter was equal to the string length, the program would know to stop.

Since the grid is just several lines of bytes, the search can sometimes overflow onto the next line, which isn't allowed. To combat this, I kept track of the current column the program was searching in. The program used two nested for loops, so whenever the inner temporary loop counter reached the edge of the grid, the loop would reset.

Pseudocode:

|  |
| --- |
| nAddress = address  strLength = stringLength(tStrAddress)  for (i < 12; i++) {  for (j < 12, j++) {  k = 0  temp = LENGTH  pos = temp \* i  pos = pos + j  curChar = nAddress[i][j]  strChar = string[k]  tJ = j // temporary j. used to make sure check doesn't go out of bounds  if (curChar = strChar) {  counter = 1 // offset counter by 1 to sync with second character  k = k + 1 // offset k by 1 to sync characters and ignore first character  for (k in string) {  if (tJ < LENGTH) {  pos = pos + 1  curChar = nAddress[pos]address  strChar = string[k]tStrAddress  if (curChar == strChar) {  counter += 1  }  k += 1  tJ += 1  }  }  }  if (counter == strLength)  return 1  }  j = j + 1  }  i = i + 1  }  return 0 |