Madeleine Comtois
09/12/17

Sets, Countdown, and Lotto

I. Sets - Symmetric Difference

Sets creates a new set, C, which is the symmetric difference of given sets A and B. The symmetric difference of two sets is a list of the values in both set that the two sets do not have in common.

Pseudocode:

```
Aaddress = address of A elements
Avalue = 0
Baddress = address of B elements
Bvalue = 0
Caddress = address of C elements
ASize  = 8
BSize = 6
CSize  = 0
Acounter = 0
Bcounter = 0

while (Bcounter != Bsize)
{
        Avalue = next element in Aaddress
        Bvalue = next element in Baddress
        if (Bcounter == Bsize)
        {
                store Avalue in Caddress
                Caddress += 4
                Csize++
                Acounter++
                if (Acounter == Asize)
                {
                        reset Aaddress
                        reset Baddress
                        Acounter = 0
                        Bcounter = 0
                }
                else
                {
                        Aaddress += 4
```

```
                    reset Baddress
                    Bcounter = 0
                }

        }
        else if (Avalue == Bvalue)
        {
                if (Acounter == Asize)
                {
                    reset Aaddress
                    reset Baddress
                    Acounter = 0
                    Bcounter = 0
                }
                else
                {
                    Aaddress += 4
                    Acounter += 1
                    reset Baddress
                    Bcounter = 0
                }
        }
        else
        {
                Baddress += 4
                Bcounter ++
        }
}

while (Acounter != Asize)
{
        Avalue = next element in Aaddress
        Bvalue = next element in Baddress
        if (Acounter == Asize)
        {
                store Bvalue in Celements
                Caddress += 4
                Csize++
                Bcounter++
                if (Bcounter == Bsize)
                {
                    reset Aaddress
```

```
                reset Baddress
                Acounter = 0
                Bcounter = 0
        }
        else
        {
                Baddress += 4
                reset Aaddress
                Acounter = 0
        }
    }
    else if (Avalue == Bvalue)
    {
        if (Bcounter == Bsize)
        {
                reset Aaddress
                reset Baddress
                Acounter = 0
                Bcounter = 0
        }
        else
        {
                Baddress += 4
                Bcounter += 1
                reset Aaddress
                Acounter = 0
        }
    }
    else
    {
        Aaddress += 4
        Acounter ++
    }
}
store Csize
```

Explanation:

This program uses two while loop to first loop through the elements in set A and the second loop to loop through the elements in set B. In the first loop, the first values in A and B are initialized and and Bcounter is checked to see which element in the B list has been initialized. The elements are compared, and if they are different, the next B element is loaded and

compared to A. This loop continues until the current A value has been compared to all the elements in B. If all the elements in B are run without matching the A value, the A value is added into the C set and the size of the C set is increased by 1. After this loop the next value of A is loaded, the B value is reloaded to the first element in the set, and the loop through the B set is repeated to compare the B values to the new A value. This is repeated until every value in A has been compared to every value in B. If A and B match, the A value is not stored in the C set and the loop restarts with the next A value and after resetting the B values. The second while loop in the program is a repeat of the first loop, except the roles of A and B are changed so as to add the unique B values into the C set. After the loops are run, the size of the C set is stored in memory.

Example:

Set A = 4, 6, 2, 13, 19, 7, 1, 3
Set B = 13, 9, 1, 9, 5, 8

4 is first compared to 13, and since they are not equal, the next B value, 9, is loaded, so 4 eventually is compared to 13, 9, 1, 9, 5, and 8. Not equalling any of these values, 4 is added into the C set, then 6 is loaded from the A set and compared to the entire B set again. This is repeated until all the elements in set A are compared. When the A element 13 is compared to the B element 13, neither number is stored in the C set, and the next A element, 19, is loaded, and B is reset again to be 13. Once this is done the second loop loads the B value 13 and compares it to the A value 4. Since it does not equal 4 the next element in A is loaded, 6, and is compared to B. This repeats until every element in B has been compared to every element in A.

Testing:
Set A = 1, 2, 3
Set B = 1, 2, 3
Result: Set C contained no elements and its size was 0
Used to test two of the same lists

Set A = 0
Set B = 0
Result: Set C contained no elements and its size was 0
Used to test 0 values

Set A = 1, 2, 3, 4, 5
Set B = 6, 7, 8, 9
Result: Set C contained 1, 2, 3, 4, 5, 6, 7, 8, 9
Used to test lists with no common values

Set A = 1, 2, 3, 4, 5
Set B = 0
Result: Set C contained 1, 2, 3, 4, 5, 0
Used to test a list with a list that only had a 0

II. Countdown Checker

Countdown creates a list of letters and a word and checks to see if the word can be formed out of the given list of letters. If the word can be formed out of the letters (with no repeating values), a 1 is stored in R0. Otherwise, a 0 is stored into R0.

Pseudocode:

```
wordAddress = word to find in list
listAddress = list of letters
wordLetter = first letter in word
listLetter = first letter in list
R0 = 1

while (listLetter != 0)
{
        if (wordLetter == listLetter)
        {
                wordLetter = '$'
                listLetter = '$'
                store wordLetter
                store listLetter
                listAddress++
                next listLetter
        }
        else
        {
                listAddress++
                next listLetter
        }
}

wordAddress++
next wordLetter
reset listAddress
next listLetter
if (wordLetter != 0)
{
```

```
        branch back to while
}
else
{
        reset wordAddress
        next wordLetter
}

while (wordLetter != 0)
{
        if (wordLetter == '$')
        {
                wordAddress++
                next wordLetter
        }
        else
        {
                R0 = 0
        }
}
```

Explanation:

This program uses two loops to compare the letters in a word to a list of letters. Firstly, R0 is initialized as 1, meaning that it is possible to form the word out of the list of letters. The first while loop then compares the first letter in the word to the first letter in the list. If they are equal, both the letter in the word and the letter in the list are replaced by a '$'. If they are not equal, the next letter in the list is compared to the letter in the word, and this loop is repeated until every letter in the word is compared to every letter in the list. The second loop checks to see if every letter in the word has been replaced by a '$'. If not all the letters are '$', a 0 is moved into R0, showing that the word cannot be made out of the letters in the list.

Example:

word = 'beets'
letter list = 'daetebzsb'

The loop starts by comparing 'b' in the word to 'd' in the list. They are not equal, so 'b' is then compared to the next letter in the list, 'a'. This continues until 'b' is compared to the list letter 'b'. Both letters are replaced by '$' and the loop terminates. The next letter in the word, 'e' is loaded and compared to the first letter in the letter list, 'b'. This continues until 'e' is found in the list and both are replaced with a '$'. When all the letters in the word have

been looped through, another while loop is created to loop through the updated word. Since all of the letters in the word were in the list, the word looks like '$$$$$' and the list looks like 'dae$$$z$$'. The 1 remains in R0 because the entire word is made up of '$'. If the word were not, then a 0 would be moved into R0.

Testing:
word = 'happy'
letter list = 'esdhaepewlpye'
Result: letter list = 'esd$$e$ewl$$e' and R0 = 1
Used to test if word was able to be found in the list

word = 'hi'
letter list = 'abcdefg'
Result: letter list = 'abcdefg' and R0 = 0
Used to test if word was not able to be found in the list

word = 'pretty'
letter list = 'rghpenrwezty'
Result = '$gh$$n$wez$$' and R0 = 0
Used to test that double letters could not be counted twice

III. Lottery

Lotto analyzes lottery tickets to determine the number of matches between the ticket numbers and the drawn winning numbers. The number of matches for 4, 5, and 6 matches are stored in memory.

Pseudocode:

lotteryDrawAddress = address of drawNumber
ticketAddress = address of ticketNumber
ticketNumber = 0
drawNumber = 0
matchCounter = 0
match4 = 0
match5 = 0
match6 = 0
countAddress = 3
drawCounter = 0
numOfRounds = 0
ticketCounter = 0

```
while (numOfRounds != numOfTickets)
{
	if (ticketNumber != drawNumber)
	{
		drawCounter++
		if (drawCounter != 6)
		{
			lotteryDrawAddress++
			drawNumber = Memory.byte(lotteryDrawAddress)
		}
		else
		{
			drawCounter = 0
			ticketCounter++
			if (ticketCounter != 6)
			{
				ticketAddress++
				ticketNumber = Memory.byte(ticketAddress)
				reload lotteryDrawAddress
				drawNumber = Memory.byte(lotteryDrawAddress)
			}
			else
			{
				if (matchCounter == 4)
					match4++
				else if (matchCounter == 5)
					match5++
				else if (matchCounter == 6)
					match6++
			}
		}

	}
	else
	{
		matchCounter++
		drawCounter = 0
		ticketAddress++
		ticketCounter++
		if (ticketCounter != 6)
		{
			ticketNumber++
```

```
                reload lotteryDrawAddress
                drawNumber = Memory.byte(lotteryDrawAddress)
        }
        else
        {
                if (matchCounter == 4)
                        match4++
                else if (matchCounter == 5)
                        match5++
                else if (matchCounter == 6)
                        match6++
        }
    }

    numOfRounds++
    ticketCounter = 0
    drawCounter = 0
    matchCounter = 0
    ticketAddress++
    ticketNumber = Memory.byte(ticketAddress)
    reload lotteryDrawAddress
    drawNumber = Memory.byte(drawNumber)
}

R0 = match4
load match4Address
store match4 in match4Address
R0 = match5
load match5Address
store match5 in match5Address
R0 = match6
load match6Address
store match6 in match6Address
```

Explanation:

This program uses a while loop to iterate through each value in the ticketAddress (which holds the values of all the numbers for every ticket) and compares them to the list of numbers in the lotteryDrawAddress. If a match is found, the matchCounter is increased by 1. After the program reaches the end of one ticket, which has a length of 6 values, it evaluates the ticket to see how many matches it has to the drawn numbers. It stores this value in an appropriate register then resets the matchCounter to 0 and increases the ticketAddress so as to

start the comparisons for the next ticket. This process repeats until all tickets have been compared to the drawn numbers. After the while loop has finished, the number of 4, 5, and 6 matches are stored in the memory addresses for R4, R5, and R6 respectively.

Example:

ticketAddress = {3, 8, 11, 21, 22, 31},
{7, 23, 25, 28, 29, 32},
{10, 11, 12, 22, 26, 30}
lotteryDrawAddress = {10, 11, 12, 22, 26, 30}

While the number of tickets does not equal 3, the first ticket number, 3, is compared to the first draw number, 10. There is not a match, so the draw number is increased to 11. There is not a match, so it is increased to 12. This repeats until all the draw numbers have been compared to the ticket value 3. If there were a match, the matchCounter would have been increased. Once 3 has been compared to all of the draw numbers, the ticket number is increased to the next value, 8. The value 8 is compared to the beginning of the draw numbers, 10, again. The same process that was used for 3 is repeated until the first six numbers of the ticket numbers have been compared.

After the numbers 3 - 31 (the first ticket) have been compared, the program analyzes the match counter to see how many matches it found, in this case 2 (values 11 and 22). This value is not 4, 5, or 6, so it is not stored in memory. The next ticket value is loaded and the process repeats comparing values for the second ticket. There are no matches for the second ticket. Finally, the third and last ticket is compared. All the ticket numbers match the draw numbers for this ticket, so a 1 is stored in memory to show that 1 ticket had 6 matches.

Testing:

ticketAddress = {3, 8, 11, 21, 22, 31},
{7, 23, 25, 28, 29, 32},
{1, 1, 1, 2, 6, 0}
lotteryDrawAddress = {10, 11, 12, 22, 26, 30}
Result: no matches are recorded in memory
Used to test for no matches

ticketAddress = {10, 11, 12, 21, 22, 31},
{10, 11, 12, 22, 26, 32},
{10, 11, 12, 22, 26, 30}

lotteryDrawAddress = {10, 11, 12, 22, 26, 30}
Result: 1 match4, 1 match5, 1 match6
Used to test for all matches

ticketAddress = {1},
{10, 11, 12, 22, 26, 32},
{14, 2, 54, 0}
lotteryDrawAddress = {10, 11, 12, 22, 26, 30}
Result: Error, there is not the correct number of values for each ticket
Used to test for incorrect starting values