Memory

22-DEC-2016

Report to accompany my second computing assignment

Niall Hunt

CS1021
Intro to Computing 1

Assignment #2

Contents

Solution

Part 1: Sets - Closure

Part 1

In this part we had to create a program that takes a set stored in memory and returns true (1) if the set is closed under negation and false (0) if it is not. For a set to be closed, every integer has to have a corresponding integer with the opposite sign. (Example. For a +1 a -1 is needed). Zero is a valid element but it does not require a pair to be considered closed.

For a set to be closed under negation all the elements added together must equal zero. This is the first thing I check in my program. I use a simple loop to add all the elements into a running total. If this does not equal zero I immediately return false.

We know form set theory that a valid set doesn't have duplicated elements. I check to see if there are any duplicated elements next in my program. If there are duplicated elements I can immediately say that the set is not closed under negation as it is not a valid set.

After this I check each element in the set to see if it has a corresponding integer. I do this by looping through each element. I add the current element to each other element one by one. If this addition equals zero we know that it is the corresponding integer. I then reset isClosed (my Boolean that is returned) to true. I cross out each element by replacing it with zero. I can do this because I know in the beginning that there will only be a max of one zero, as zero doesn't require a pair I can skip it if the current element equals zero. If an element doesn't have a pair I return false as it is not closed under negation. The pseudo-code for this is shown below.

Solution

Part 1:
Sets - Closure

Pseudo-Code

```
while(isClosed && count1 < ASize)
{
        isClosed = false;
        currElem = memory.Word([currElemAdr]);
        count2 = count1;
        testElemAdr = (currElemAdr + 4);
        while(!isClosed && count2 < ASize)
        {
                testElem = memory.Word([testElemAdr]);
                sum = currElem + testElem;
                if(sum == 0)
                {
                        isClosed = true;
                        currElem = 0;                   //Crossing it out
                        testElem = 0;                   //Crossing it out
                }
                else
                {
                        count2++;
                        testElemAdr += 4;
                }
        }
        count1++;
        currElemAdr += 4;
}
```

Part 2: Sets – Symmetric Difference

Solution

Part 2

In this part we had to create a program that would compare two sets, A and B, stored in memory and store the symmetric difference of these two sets in a new set, C. The symmetrical difference of two sets is all of the elements except the intersection of the two sets.

To do this I had to compare each element of set A with each element of set B. If the current element (from set A) is equal to the test element (from set B) then it is not stored in set C. If the current element was tested against every element of B and there is no match, it is stored in the start address of set C which is already set up. Each time an element is added to set C, countC is increased by one, and the address of CElems is increased by 4 to get the address for the next element.

When all elements in A are tested against the elements in B I do the same except in the opposite direction. I compare all of the elements of B against all the elements of A. This is done so that I am able to create set C without destroying sets A and B. When this is completed I store countC as CSize. The pseudo-code for checking one set against the other is shown below.

To test this I have a simple loop that loops through every element in set C. If you step through it you can see each element and check if they are right.

Solution

Part 2: Sets – Symmetric Difference

Pseudo-Code

```
while(countA < ASize)
{
        isStored = true;
        currElemA = memory.Word([AElemsAdr]);
        AElemsAdr += 4;
        while((countB <= BSize) && isStored)
        {
                currElemB = memory.Word([BElemsAdr]);
                BElemsAdr += 4;
                if(countB == BSize)
                {
                        memory.Store(currElemA, [CElemsAdr]);
                        CElemsAdr += 4;
                        countC++;
                }
                else if(currElemB == currElemA)
                {
                        isStored = false;
                }
                else
                {
                        countB++;
                }
        }
        countA++;
        countB = 0;
        BElemsAdr = BElemsStartAdr;
}
```

Part 3: Anagrams

Solution

Part 3

In the final part of this assignment we had to create a program that takes in two strings stored in memory and checks them to see if they are anagrams. I have to compare each character in the first string and check to see if it is present in the second string. If so I cross them both out and keep the Boolean isAnagram true.

I first check to see if the two strings are equal in length. If not they can't be anagrams. I decided to allow the user to enter strings that have non-alphabetical characters in them to allow anagrams with words that have apostrophes (Example. "Tom's" and "Most") I did not count characters unless they were alphabetic. I am assuming the user will be slightly well behaved and not enter strings with absurd amounts of special characters. While this shouldn't break the program, it makes it a little bit less polished if they are able to do it if they want.

My program then loops through all the characters in the first string and checks them against the characters in the second. It will however skip special characters. If no matching character is found the Boolean isAnagram is set to false. If a character is in upper case I add 0x20 to it to make it lower case. This way the program is case insensitive.

If all the characters from the first string have a matching characters from the other string then isAnagram is left as true. My pseudo-code for checking the characters is shown below.

Solution

Part 3:
Anagrams

Pseudo-Code

```
stringAAdr = startStringA;
while(memory.Byte[stringAAdr] != NUL)
{
        stringBAdr = startStringB;
        currAChar = memory.Byte[stringAAdr];
        stringAAdr++;
        if((currAChar >= 'A') && (currAChar <= 'Z'))
        {
                currAChar += 0x20;
        }
        if(currAChar >= 'a' && currAChar <= 'z')
        {

                while(memory.Byte[stringBAdr] != NUL)
                {
                        currBChar = memory.Byte[stringBAdr];
                        stringBAdr++;
                        if((currAChar >= 'A') && (currAChar <= 'Z'))
                        {
                                currAChar += 0x20;
                        }
                        if(currAChar >= 'a' && currAChar <= 'z')
                        {


                                if(currAChar == currBChar)
                                {
                                        isAnagram = true;
                                        currAChar = #0x1;
                                        currBChar = #0x1;
                                }
                                else if(currBChar == NUL)
                                {
                                        isAnagram = false;
                                }
                        }
                }
        }
}
```

Testing

Part 1

Input	Reason	Result	Correct?	Fixed?
0 Elements	To see how the program no inputs	Is not negated	Correct	N/A
1 Element	To see what happens if there is an odd number of inputs	Is not negated	Correct	N/A
1,2,3,-1,-2,-3	To see if a set that should be negated is negated	Is negated	Correct	N/A
2,4,6,8,-5,-3,-2,-10	To see what happens when a set that totals to zero but is not negated	Is not negated	Correct	N/A
0	To see what happens when zero is inputted	Is negated	Correct	N/A
1,2,-1,-2,0	To see what happens when zero is added to a closed string	Is negated	Correct	N/A
1,1,-1,-1	To see what happens with duplicate elements	Is not negated	Correct	N/A

Part 2

Input	Reason	Result	Correct?	Fixed?
A = {1,2} **B = {3,4}**	To see what happens when all elements should be stored	CSize = 4 C = {1,2,3,4}	Correct	N/A
A = {1,2,3,4} **B = {1,2}**	To see what happens when no elements from one set should be saved	CSize = 2 C = {3,4}	Correct	N/A
A = {1,2,3,4} **B = {3,4,5,6}**	To see what happens for two sets that have some of the same elements	CSize = 4 C = {1,2,5,6}	Correct	N/A

Part 3

Input	Reason	Result	Correct?	Fixed?
“coats” **“tacos**	To see if it recognized to actual anagrams	isAnagram = true	Correct	N/A
“COATS” **“tacos”**	To see if the case insensitivity workes	isAnagram = true	Correct	N/A
“bests” **“beets”**	To see if two similar strings that are not anagrams would work	isAnagram = false	Correct	N/A
“Tom’s” **“Most”**	To test the allowance of non-alphabetical characters	isAnagram = true	Correct	N/A