Programming report
assignment 1 for A&DinC

Thomas Janssen (s1565001)
Robin Sommer (s2997592)

February 9, 2016

1 Problem description

The problem was as follows. We were given two sets of sentences and we needed to compute for each sentence in the second set which sentences of the first set are an anagram of it. An anagram is a sentence which exists of the same characters.

2 Problem analysis

An anagram is a sentence which contains the same amount of characters and the amount of each different character is also the same. By checking if the sentences have the same amount of characters is the first step to see whether they could be anagrams of each other. If the amount of characters isn't the same it cannot be an anagram of the sentence. If the amount is the same we go to the next step. To determine whether they really are anagrams you should check if the characters in the sentences are exactly the same.

3 Program design

First it is important to store the given sentences correctly. We chose to use 2D-arrays for this. First we scan the integer which stands for the amount of sentences that will be given. This enables us to create an array with that given size. Then we call a function which stores the given input (sentences) in the 2D array. We do this in such a way that the sentences are stored as histograms. So the amount of each character in the sentence is stored. The simplest way to do this is to read the character and subtract the character 'a' of it. In this way we generate an array where the a is on the first index and the z is on the last. We use the last index to store the amount of characters in that particular sentence. The second array for the test sentences will be generated in the same way.
The next step is to compare the test sentence with every sentence in the first set. We use a double for-loop for this, where the outer loop represents the test sentences and the inner loop the sentences from the first set. Within the inner loop we check whether the amount of characters in the sentences are the same. If it is, we check whether the characters are also the same. We do this by comparing the two histograms. If they are equal we print the number of the sentence of the first set.
And last, but not least, we free the two array's that we have used.

4 Evaluation of the program

We used the original test set which was provided with the assignment. We slightly altered it for testing the program extensions.

- Input:

```
7
astronomers.
leaving your idol.
moon starers.
the fine game of nil.
avoiding our yell.
no.
lin grad.
4
no more stars.
none.
the meaning of life.
darling i love you.
```

Output: (for anagram.c)

```
1 3

4
2 5
```

Output: (for anagram_ext1.c)

```
no more stars. - astronomers.
no more stars. - moon starers.
the meaning of life. - the fine game of nil.
darling i love you. - leaving your idol.
darling i love you. - avoiding our yell.
```

Output: (for anagram2_extension2.c)

```
Test sentence 1, word 1 - sentence 6
Test sentence 1, words 1 till 3 - sentence 1
Test sentence 1, words 1 till 3 - sentence 3
Test sentence 3, words 1 till 4 - sentence 4
Test sentence 4, word 1 - sentence 7
Test sentence 4, words 1 till 4 - sentence 2
Test sentence 4, words 1 till 4 - sentence 5
```

We also checked the programs with valgrind and there were no memory leaks.

5 Extension of the program (optional)

We did both of the suggested extension for the program. For the first extension we needed to use an additional array to store the sentences. This way we could print them to the screen in the output. The comparison of the sentences remained the same. Only this time we print the test sentence and sentence with which it is an anagram.

For the second extension we had to modify the array for the histograms of the test sentences. We now used a 3d-array. So we now could store histograms for each word in the test sentence. We use slightly adjusted functions for this. We also had to use an additional function to read the input for the test sentences. In that function we now first store the sentence in an array and then use that array to create the histograms for every word in the sentence. Next, before we compare the histograms we now add the histograms. So that each histogram of the words contains all the histograms of previous words in the test sentence. Now we can compare an initial segment of the test sentence with all the sentences from the first set. The result of this is printed to the screen.
Finally we had to add an extra function to free the memory of the 3d-array.

6 Process description

This programming assignment went well. Except for one segmentation fault we haven't really encountered any problems. The most difficult part was probably not creating any segmentation fault by allocating and freeing the memory correctly.

7 Conclusions

Our program solves the problem correctly. We had a minor problem with a segmentation fault, but we fixed that. The first and the second extensions also work correctly.
The program itself is very efficient, because it directly generates a histogram of the sentences. There is no time and space wasted to make an extra array to first store the sentence and after that make an histogram. In extension 1 we do make use of a second array to store the sentence. But it is still efficient because while scanning the sentence we store the sentence and make a histogram at the same time.
In extension 2 we had to store the sentences first and then make the histograms. But only for the test sentences.
We think this is the optimal solution to the problem.

8 Appendix: program text

Listing 1: `anagram.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4
5  #define TRUE 1
6  #define FALSE 0
7
8  char **safeCallocChar(int size) {
9     char **ptr = calloc(size, sizeof(char*));
10    if (ptr == NULL) {
11       printf("\nError: memory allocation failed....abort\n");
12       exit(-1);
13    }
14    for(int i=0; i<size; i++) {
15       ptr[i] = calloc(100, sizeof(char));
16          if (ptr[i] == NULL) {
17             printf("\nError: memory allocation failed....abort\n");
18             exit(-1);
19          }
20    }
```

```
21    return ptr;
22 }
23
24 int **safeCallocInt(int size) {
25    int **ptr = calloc(size, sizeof(int*));
26    if (ptr == NULL) {
27       printf("\nError: memory allocation failed....abort\n");
28       exit(-1);
29    }
30    for(int i=0; i<size; i++) {
31       ptr[i] = calloc(27, sizeof(int));
32          if (ptr[i] == NULL) {
33             printf("\nError: memory allocation failed....abort\n");
34             exit(-1);
35          }
36    }
37    return ptr;
38 }
39
40 void readInput(int size, char **array) {
41    for(int i = 0; i < size; i++) {
42       scanf("%[^.]%*c", array[i]);
43    }
44 }
45
46 void makeHisto(char **string, int size, int **histo) {
47    int place;
48
49    for(int i = 0; i < size; i++) {
50       for(int j = 0; string[i][j] != 0; j++) {
51          if(isalpha(string[i][j])) {
52             place = string[i][j] - 'a';
53             histo[i][place]++;
54             histo[i][26]++;
55          } else {
56             continue;
57          }
58       }
59    }
60 }
61
62 int isEqual(int *array1, int *array2, int size){
63    for(int i = 0; i < size; i++) {
64       if (array1[i] == array2[i]) {
65          continue;
66       } else {
67          return FALSE;
68       }
69    }
70    return TRUE;
71 }
72
73 void free2DArrayChar(char **array, int size) {
74    for(int i = 0; i < size; i++) {
75       free(array[i]);
76    }
77    free(array);
78 }
```

```
79
80  void free2DArrayInt(int **array, int size) {
81     for(int i = 0; i < size; i++) {
82        free(array[i]);
83     }
84     free(array);
85  }
86
87  int main(int argc, char *argv[]) {
88     int sizeOne = 0;
89     int sizeTwo = 0;
90
91     scanf("%d", &sizeOne);
92     char** str1 = safeCallocChar(sizeOne);
93     int** histo1 = safeCallocInt(sizeOne);
94     readInput(sizeOne, str1);
95
96     scanf("%d", &sizeTwo);
97     char** str2 = safeCallocChar(sizeTwo);
98     int** histo2 = safeCallocInt(sizeTwo);
99     readInput(sizeTwo, str2);
100
101    makeHisto(str1, sizeOne, histo1);
102    makeHisto(str2, sizeTwo, histo2);
103
104    for(int j = 0; j < sizeTwo; j++) {
105       for(int i = 0; i < sizeOne; i++) {
106          if (histo1[i][26] == histo2[j][26]) {
107             if (isEqual(histo1[i], histo2[j], 26)) {
108                printf("%d␣", i+1);
109             }
110          } else {
111             continue;
112          }
113       }
114       printf("\n");
115    }
116
117    free2DArrayChar(str1, sizeOne);
118    free2DArrayChar(str2, sizeTwo);
119    free2DArrayInt(histo1, sizeOne);
120    free2DArrayInt(histo2, sizeTwo);
121
122    return 0;
123 }
```

9 Appendix: extended program text (optional)

Listing 2: anagram_ext1.c

```c
/* file: anagram_ext1.c */
/* author: Thomas Janssen (t.m.janssen.2@student.rug.nl) */
/* author: Robin Sommer (r.m.sommer@student.rug.nl) */
/* date: mon feb 8 2016 */
/* version: 1 */

/* Description: */
/* Given is an amount of sentences(n) and test sentences(m). This program checks for each of the test sentences if it is
 * an anagram of one of the sentences. Each line of the output corresponds with a test sentence and it will print the number
 * of each sentence with wich it is an anagram.
 * This extension prints the sentences.
 */

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>



void createCharArray(char *array[]){
   *array = (char *)calloc(50, sizeof(char));
   assert(*array != NULL);
}

void createHistogram(int *array[]){
   *array = (int *)calloc(27, sizeof(int));
   assert(*array != NULL);
}

void readInput(char **array, int **array2, int size){
char c;
int index;

   for(int i=0; i < size; i++){
      createCharArray(&array[i]);
      createHistogram(&array2[i]);

      index = 0;
      c = '\0';
      while(c != '.'){
         /* Read a char but exclude newline */
         scanf( "%c%*[\n]", &c);

         array[i][index] = c;
         index++;

         if(c != '␣' && c != '.' ){
            array2[i][(int)c-'a'] +=1;
            array2[i][26] += 1;
         }
      }
   }
```

```
53 }
54
55 void freeMem(int **array, int size){
56    for(int i=0; i<size; i++){
57       free(array[i]);
58    }
59
60 }
61 void freeMemChar(char **array, int size){
62    for(int i=0; i<size; i++){
63       free(array[i]);
64    }
65
66 }
67
68 int main(int argc, char *argv[]){
69    int n, m;
70
71    scanf("%d%*[ \n]", &n);
72    char *sentencesArray[n];
73    int *sentencesHistoArray[n];
74    readInput(sentencesArray, sentencesHistoArray, n);
75
76    scanf("%d%*[ \n]", &m);
77    char *testSentencesArray[m];
78    int *testSentencesHistoArray[m];
79    readInput(testSentencesArray, testSentencesHistoArray, m);
80
81    /* Compare and print results */
82    for(int i=0; i<m; i++){
83       for(int j=0; j<n; j++){
84          if(testSentencesHistoArray[i][26] == sentencesHistoArray[j][26]){
85             int k;
86             for(k=0; k<26; k++){
87                if(testSentencesHistoArray[i][k] != sentencesHistoArray[j][k]){
88                   break;
89                }
90             }
91             if(k == 26){
92                printf("%s - ", testSentencesArray[i]);
93                printf("%s \n", sentencesArray[j]);
94             }
95          }
96       }
97    }
98
99    freeMem(sentencesHistoArray, n);
100   freeMem(testSentencesHistoArray, m);
101   freeMemChar(sentencesArray, n);
102   freeMemChar(testSentencesArray, m);
103
104   return 0;
105 }
```

Listing 3: anagram2extension2.c

```
1  /* file: anagram2extension2.c */
2  /* author: Thomas Janssen (t.m.janssen.2@student.rug.nl) */
3  /* author: Robin Sommer (r.m.sommer@student.rug.nl) */
4  /* date: mon feb 8 2016 */
5  /* version: 1 */
6
7  /* Description: */
8  /* Given is an amount of sentences(n) and test sentences(m). This program
       checks for each of the test sentences if it is
9   * an anagram of one of the sentences. And it also checks for part of the test
        sentences. On each line of the output
10  * it will print a test sentence or part of it and it will print the sentence
      with wich it is an anagram.
11  */
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16 #include <assert.h>
17
18
19
20 void createCharArray(char *array[]){
21    *array = (char *)calloc(50, sizeof(char));
22    assert(*array != NULL);
23 }
24
25 void createHistogram(int *array[]){
26    *array = (int *)calloc(27, sizeof(int));
27    assert(*array != NULL);
28 }
29
30 void createArrayWordCount(int **array[], int size){
31    *array = calloc(size, sizeof(int*));
32    assert(*array != NULL);
33 }
34
35 void readInput(char **array, int **array2, int size){
36 char c;
37 int index;
38
39    for(int i=0; i < size; i++){
40
41       createCharArray(&array[i]);
42       createHistogram(&array2[i]);
43
44       index = 0;
45       c = '\0';
46       while(c != '.'){
47          /* Read a char but exclude newline */
48          scanf( "%c%*[\n]", &c);
49
50          array[i][index] = c;
51          index++;
52
53          if(c != '␣' && c != '.' ){
54             array2[i][(int)c-'a'] +=1;
```

```
55              array2[i][26] += 1;
56           }
57        }
58     }
59  }
60
61  void readTestInput(char **array, int ***array2, int size, int *wordCountArray)
        {
62  char c;
63  int index, index2;
64  int lengthArray;
65
66     for(int i=0; i < size; i++){
67
68        createCharArray(&array[i]);
69
70        index = 0;
71        c = '\0';
72        while(c != '.'){
73           /* Read a char but exclude newline */
74           scanf( "%c%*[\n]", &c);
75
76           array[i][index] = c;
77           index++;
78
79           if(c == '␣'){
80              wordCountArray[i] += 1;
81           }
82        }
83
84        createArrayWordCount(&array2[i], wordCountArray[i]);
85
86        index2 = 0;
87        createHistogram(&array2[i][index2]);
88
89        lengthArray = strlen(array[i]);
90        for(int j = 0; j<lengthArray; j++){
91           c = array[i][j];
92
93           if(c == '␣'){
94              index2++;
95              createHistogramTest(&array2[i][index2]);
96           }
97
98           if(c != '␣' && c != '.' ){
99
100             array2[i][index2][(int)c-'a'] +=1;
101             array2[i][index2][26] +=1;
102          }
103       }
104    }
105 }
106
107 void freeMem(int **array, int size){
108    for(int i=0; i<size; i++){
109       free(array[i]);
110    }
111
```

```
112 }
113 void freeMemChar(char **array, int size){
114    for(int i=0; i<size; i++){
115       free(array[i]);
116    }
117
118 }
119 void freeMemTest(int ***array, int size, int *wordCount){
120    for(int i=0; i<size; i++){
121       for(int j=0; j< wordCount[i]; j++){
122          free(array[i][j]);
123       }
124       free(array[i]);
125    }
126
127 }
128
129 int main(int argc, char *argv[]){
130    int n, m;
131
132    scanf("%d%*[␣\n]", &n);
133    char *sentencesArray[n];
134    int *sentencesHistoArray[n];
135    readInput(sentencesArray, sentencesHistoArray, n);
136
137    scanf("%d%*[␣\n]", &m);
138    char *testSentencesArray[m];
139    int **testSentencesHistoArray[m];
140    int wordCount[m];
141    /* initialize wordCount array with ones */
142    for (int i=0; i<m; i++){
143       wordCount[i] = 1;
144    }
145    readTestInput(testSentencesArray, testSentencesHistoArray, m, wordCount);
146
147    /* Add all the histograms of the words of each test sentence so that the
        next histogram of a word
148       has all the values of the previous histograms of words */
149    for(int i=0; i<m; i++){
150       for(int j=0; j<27; j++){
151          for(int k=0; k<wordCount[i]; k++){
152
153             if(k+1 < wordCount[i]){
154                testSentencesHistoArray[i][k+1][j] += testSentencesHistoArray[i
                      ][k][j];
155             }
156          }
157       }
158    }
159
160    /* Compare and print results. Use of 3 for loops because
        testSentencesHistoArray is a 3d-array */
161    for(int i=0; i<m; i++){
162       for(int j=0; j<wordCount[i]; j++){
163          for(int k=0; k<n; k++){
164
165             if(testSentencesHistoArray[i][j][26] == sentencesHistoArray[k
                   ][26]){
```

```
166                 int l;
167                 for(l=0; l<26; l++){
168                    if(testSentencesHistoAray[i][j][l] != sentencesHistoArray[k
                           ][l]){
169                       break;
170                    }
171                 }
172                 if(l == 26){
173                    (j == 0) ? printf("Test␣sentence␣%d,␣word␣%d", i+1, j+1) :
                           printf("Test␣sentence␣%d,␣words␣1␣till␣%d", i+1, j+1);
174                    printf("␣-␣sentence␣%d\n", k+1);
175                 }
176              }
177           }
178        }
179     }
180
181     freeMem(sentencesHistoArray, n);
182     freeMemTest(testSentencesHistoArray, m, wordCount);
183     freeMemChar(sentencesArray, n);
184     freeMemChar(testSentencesArray, m);
185
186     return 0;
187 }
```