

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ: PROJECT 2019

Γλαράκης Γιώργος ΑΜ: 1059561

Παπαδιάς Επαμεινώνδας ΑΜ: 1062665

Προκοπίου Ιωάννης ΑΜ: 1059554



Documentary

Part 1

- Για την υλοποίηση των συναρτήσεων ταξινόμησης συμβουλευτήκαμε το βιβλίο του κύριου Τσακαλίδη, τις ηλεκτρονικές διαφάνειες του κύριου Σιούτα και, όπου το θεωρήσαμε απαραίτητο, χρησιμοποιήσαμε και το διαδίκτυο.
- Για την υλοποίηση των συναρτήσεων αναζήτησης συμβουλευτήκαμε το βιβλίο του κύριου Τσακαλίδη, δημιουργώντας τις δικές μας συναρτήσεις βασιζόμενοι στη θεωρία.
- Υλοποιήσαμε όλες τις συναρτήσεις σε ξεχωριστούς Headers και τις καλούμε με τη βοήθεια της συνάρτησης Switch() όποτε το επιθυμήσει ο χρήστης, επιστρέφοντας τον χρόνο εκτέλεσής της και τον αριθμό των συγκρίσεων που έγιναν κατά την εκτέλεσή της.

Part 2

- Για την υλοποίηση των απαραίτητων συναρτήσεων αυτού του Part συμβουλευτήκαμε τη θεωρία από το βιβλίο του κύριου Τσακαλίδη, κάνοντας τις αναγκαίες αλλαγές ώστε να προσαρμοστούν στο ζητούμενο της άσκησης.
- Στην υλοποίηση του Δυαδικού δένδρου με βάση τους
 βαθμούς των φοιτητών χρησιμοποιήσαμε Linked Lists μέσα σε κάθε κόμβο (κάθε ξεχωριστό βαθμό).
- Ολοκληρώσαμε και τα τρία υπο-ερωτήματα με επιτυχία και τα ενώσαμε όπως υπαγορεύει η εκφώνηση της άσκησης.
 Δηλαδή: ο χρήστης ερωτάται αρχικά ποια υλοποίηση επιθυμεί να χρησιμοποιήσει (Δυαδικό δένδρο διατεταγμένο ως προς τον Αριθμό Μητρώου των φοιτητών, Δυαδικό δένδρο διατεταγμένο ως προς το βαθμό των φοιτητών ή Πίνακας Κατακερματισμού με αλυσίδες). Στη συνέχεια καλούνται οι ανάλογες συναρτήσεις από τους αντίστοιχους Headers και υλοποιείται η ζητούμενη ενέργεια.

Αναφορά - PART 1

Στη συγκεκριμένη αναφορά παραθέτουμε τα αποτελέσματα από τις πειραματικές δοκιμές των αλγορίθμων ταξινόμησης και αναζήτησης. Χρησιμοποιήσαμε το dataset με τους 100000 ακεραίους που υπάρχει στο eclass. Για την πειραματική δοκιμή των αλγορίθμων αναζήτησης χρησιμοποιήσαμε το πρώτο, το μεσαίο και το τελευταίο στοιχείο της λίστας, για να προσδιορίσουμε τη μέση, χειρότερη και βέλτιστη περίπτωση.

 Από την πειραματική σύγκριση των τριών αυτών αλγορίθμων παρατηρούμε ότι οι χρόνοι είναι παρόμοιοι .Συγκεκριμένα στην καλύτερη περίπτωση είναι γραμμικοί O(n) ενώ τόσο στην μέση όσο και στην χειρότερη έχουμε O(n^2).

	Bubble Sort	Insertion Sort	Selection Sort
Counter	704982704	*	704982704
Time	37.064	9.294	12.480

^{*}O counter ξεπέρασε το όριο που μπορεί να μετρήσει ως unsinged long int

2. Από την πειραματική σύγκριση των δύο αυτών αλγορίθμων παρατηρούμε ότι οι χρόνοι είναι βελτιωμένοι συγκριτικά με τους προηγουμένους και επίσης παρόμοιοι μεταξύ τους όσο αναφορά την χειρότερη και την μέση περίπτωση δηλαδή της τάξης O(nlogn) αλλά διαφέρουν στην καλύτερη με τον Merge Sort να είναι ταχύτερος και να συνεχίζει με τάξης O(nlogn) ενώ ο Quick Sort μένει σε τάξη O(n^2).

	Merge Sort	Quick Sort
Counter	1536345	2047306
Time	2.4830	0.0180

3. Από την πειραματική σύγκριση των έξι αυτών αλγορίθμων παρατηρούμε ότι η θεωρία επαληθεύεται δηλαδή ο Heap Sort είναι της τάξης O(nlogn) σε κάθε περίπτωση οντάς όμως πιο αργός συγκριτικά με τον Merge Sort που σημειώνει λίγο καλυτέρους χρόνους παρότι είναι της ιδίας τάξης.

	Heap Sort
Counter	1624601
Time	0.0930

4. Από την πειραματική σύγκριση των τριών αυτών αλγορίθμων παρατηρούμε ότι ο Γραμμικής Αναζήτησης είναι της τάξης O(n) σε κάθε περίπτωση, ο Δυαδικής Αναζήτησης είναι της τάξης O(logn) σε κάθε περίπτωση ενώ ο Αναζήτησης με Παρεμβολή είναι της τάξης O(loglogn) στην χειρότερη και την μέση περίπτωση και σημειώνει τάξη O(n) στην καλύτερη περίπτωση.

	Linear Search		Binary Search			Interpolation Search			
	1st	mid	last	1st	mid	last	1st	mid	last
Counter	0	50000	99999	16	1	16	0	5	0

5. Από την πειραματική σύγκριση των δύο αυτών αλγορίθμων παρατηρούμε ότι στην βελτιωμένη έκδοση το άλμα (δηλαδή το Ι στο while) αυξάνεται εκθετικά και έτσι πραγματοποιούνται όλο ένα και μεγαλύτερα άλματα και έτσι ο χρόνος χειρότερης περίπτωσης πέφτει από O(n^1/2) σε O(logn) χωρίς να επηρεάζεται ο χρόνος μέσης περίπτωσης που παραμένει κοινός σε τάξη O(loglogn).

	Binary Interpolation Search			Improved Binary Interpolation Search			
	1st	mid	last	1st	mid	last	
Counter	1	166	330	3	177	390	

ΠΕΡΙΕΧΟΜΕΝΑ ΚΩΔΙΚΑ – Part 1

- Part1.c (main())
- Bubble_Sort.h
- Insertion_Sort.h
- Selection_Sort.h
- Merge_Sort.h
- Quick_Sort.h
- Heap_Sort.h
- Linear_Searching.h
- Binary_Searching.h
- Interpolation_Search.h
- Binary_Interpolation_Search.h
- Improved_Binary_Interpolation_Search.h
- TXT-to-ARRAY.h

```
1 /*-----
2 Computer Engineering and Informatics Department
3 University of Patras
5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
11 ----*/
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <time.h>
16 #include <math.h>
17 #include "Bubble Sort.h"
18 #include "Insertion_Sort.h"
19 #include "Selection_Sort.h"
20 #include "Merge_Sort.h"
21 #include "Quick_Sort.h"
22 #include "Heap_Sort.h"
23 #include "Linear_Searching.h"
24 #include "Binary_Searching.h"
25 #include "Interpolation_Search.h"
26 #include "Binary_Interpolation_Search.h"
27 #include "Improved_Binary_Interpolation_Search.h"
28 #include "TXT-to-ARRAY.h"
29
30 unsigned long int counter = 0;
31 int n = 100000;
32
33 int main()
34 {
35
       int Search_Result = -1; //item found flag
      int sorted = 1; //sorted flag
36
37
      int ch; //switch choice variable
38
39
      int *list;
40
      list = txt_to_array("integers.txt");
41
42
      clock t start, end;
43
      double cpu_time_used;
44
      printf("----\n\n");
45
46
      printf("Computer Engineering and Informatics Department\n");
      printf("
47
                     University of Patras
48
      printf("
                     Data Structures \n");
      printf("
49
                    Project 2019 \n\n");
```

```
...arakis\source\repos\DataStructures\DataStructures\Part1.c
```

```
2
```

```
50
       printf("
                   Glarakis George AM: 1059561 \n");
51
       printf("
                   Papadias Epameinondas AM: 1062665 \n");
52
       printf("
                   Prokopiou Giannis AM : 1059554 \n\n");
53
       printf("-----\n\n");
54
55
       do {
           printf("1. Bubble Sort\n");
56
57
           printf("2. Insertion Sort\n");
58
           printf("3. Selection Sort\n");
           printf("4. Merge Sort\n");
59
           printf("5. Quick Sort\n");
60
           printf("6. Heap Sort\n");
61
62
           printf("\nChoose a sorting function: ");
63
64
           scanf("%d", &ch);
           while ((getchar()) != '\n'); //clear the input buffer
65
66
           switch (ch)
67
           case 1:
68
69
               start = clock();
70
               list = Bubble_Sort(list, n);
71
               end = clock();
72
               break;
73
74
           case 2:
75
               start = clock();
76
               list = Insertion_Sort(list, n);
77
               end = clock();
78
               break;
79
80
           case 3:
               start = clock();
81
82
               list = Selection_Sort(list, n);
83
               end = clock();
84
               break;
85
           case 4:
86
87
               start = clock();
88
               list = Merge_Sort(list, n);
89
               end = clock();
90
               break;
91
92
           case 5:
93
               start = clock();
               list = Quick_Sort(list, n);
94
95
               end = clock();
96
               break;
97
98
           case 6:
```

```
...arakis\source\repos\DataStructures\DataStructures\Part1.c
```

```
3
```

```
99
                 start = clock();
100
                 list = Heap_Sort(list, n);
101
                 end = clock();
102
                 break;
103
104
             default:
105
                 printf("\nPlease choose one of the sorting functions!\n\n");
106
                 sorted = 0;
107
             }
108
109
         } while (sorted == 0);
110
111
         cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
         printf("\nList sorted!\n");
112
113
        printf("Counter: %d\n", counter);
         printf("Time: %lf\n\n", cpu_time_used);
114
115
         printf("%d \n %d \n %d \n", list[0], list[50000], list[n-1]);
116
117
        if (sorted == 1)
118
119
             int num; //the number user is searching
             int wrong_ch = 0; //wrong choice
120
121
122
             do {
123
                 wrong_ch = 0;
                 printf("\nType the number you are searching for: ");
124
125
                 scanf("%d", &num);
126
                 printf("\n1. Linear Searching\n");
127
128
                 printf("2. Binary Searching\n");
129
                 printf("3. Interpolation Searching\n");
                 printf("4. Binary Interpolation Searching\n");
130
131
                 printf("5. Improved Binary Interpolation Searching\n");
                 printf("\nChooe one of the searching functions: ");
132
                 scanf("%d", &ch);
133
                 switch (ch)
134
135
136
                 case 1:
137
                     start = clock();
138
                     Search_Result = Linear_Searching(list, num, n);
139
                     end = clock();
                     break;
140
                 case 2:
141
                     start = clock();
142
143
                     Search_Result = Binary_Searching(list, num, n);
144
                     end = clock();
145
                     break;
                 case 3:
146
                     start = clock();
147
```

```
...arakis\source\repos\DataStructures\DataStructures\Part1.c
```

```
4
```

```
148
                     Search_Result = Interpolation_Search(list, num, n);
149
                     end = clock();
150
                     break;
                 case 4:
151
152
                     start = clock();
153
                     Search_Result = BIS(list, num, n);
                     end = clock();
154
155
                     break;
156
                 case 5:
                     start = clock();
157
158
                     Search_Result = Imp_BIS(list, num, n);
                     end = clock();
159
160
                     break;
161
                 default:
162
163
                     printf("\nPlease choose one of the searching functions!\n\n");
164
                     wrong_ch = 1;
165
                     break;
166
                 }
                 if (Search_Result == -1 && wrong_ch == 0)
167
                     printf("\nThe number you are searching is not in the list!
168
                       \n");
             } while (Search_Result == -1);
169
170
         }
171
         printf("The number you are searching is located in position: %d \n",
172
           Search_Result);
         cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
173
174
        printf("\nCounter: %d\n", counter);
175
         printf("Time: %lf\n", cpu_time_used);
176
177
178
         system("pause");
179
         return 0;
180
181 }
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
4
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 int * Bubble_Sort(int *array, int n)
16 {
17
       int i, d, swap;
18
       counter = 0;
19
       for (i = 0; i < (n - 1); i++)
20
21
            for (d = 0; d < n - i - 1; d++)
22
23
                if (array[d] > array[d + 1]) /* For decreasing order use < */</pre>
24
25
26
                    swap = array[d];
                    array[d] = array[d + 1];
27
28
                    array[d + 1] = swap;
29
                }
30
                counter++;
31
            }
32
       }
33
34
       return array;
35 }
36
```

```
2 Computer Engineering and Informatics Department
3 University of Patras
4
5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 int * Insertion_Sort(int *array, int n)
16 {
17
       int i, j, temp;
18
       counter = 0;
19
20
       for (i = 1; i < n; i++)</pre>
21
22
           j = i;
23
24
           while (j > 0 \&\& array[j] < array[j - 1])
25
26
                temp = array[j];
27
                array[j] = array[j - 1];
28
                array[j - 1] = temp;
29
                j--;
30
31
                counter++;
32
            }
33
34
       }
35
       return array;
36 }
37
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
 4
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 int * Selection_Sort(int *array, int n)
16 {
17
       counter = 0;
       int i, j, swap, position;
18
19
       for (i = 0; i < (n - 1); i++)
20
21
22
           position = i;
23
24
           for (j = i + 1; j < n; j++)
25
26
                if (array[position] > array[j])
27
                    position = j;
28
29
                counter++;
30
            }
31
           if (position != i)
32
33
                swap = array[i];
34
                array[i] = array[position];
35
                array[position] = swap;
36
            }
37
        }
38
39
       return array;
40 }
41
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
 4
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 void merge(int *array, int min, int mid, int max, int n);
16 void part(int *array, int min, int max, int n);
17
18 int * Merge_Sort(int *array, int n)
19 {
20
       counter = 0;
       part(array, 0, n-1, n);
21
22
       return array;
23 }
24
25 void part(int *array, int min, int max, int n)
26 {
27
       int mid;
28
       if (min < max)</pre>
29
       {
30
            mid = (min + max) / 2;
31
            part(array, min, mid, n);
32
            part(array, mid + 1, max, n);
33
            merge(array, min, mid, max, n);
34
       }
35 }
36
37 void merge(int *arr, int min, int mid, int max, int n)
38 {
39
40
       int *tmp;
       tmp = (int*)malloc(n * sizeof(int));
42
43
       int i, j, k, m;
44
       j = min;
45
       m = mid + 1;
       for (i = min; j <= mid && m <= max; i++)</pre>
46
47
            if (arr[j] <= arr[m])</pre>
48
49
```

```
...s\source\repos\DataStructures\DataStructures\Merge_Sort.h
```

```
2
```

```
50
                 tmp[i] = arr[j];
51
                 j++;
52
            }
53
            else
54
            {
55
                 tmp[i] = arr[m];
56
                m++;
57
58
            counter++;
59
        }
        if (j>mid)
60
61
62
            for (k = m; k <= max; k++)</pre>
63
            {
64
                 tmp[i] = arr[k];
65
                 i++;
66
            }
67
        }
        else
68
69
        {
70
            for (k = j; k <= mid; k++)</pre>
71
            {
72
                 tmp[i] = arr[k];
73
                 i++;
74
            }
75
        }
        for (k = min; k <= max; k++)</pre>
76
77
78
            arr[k] = tmp[k];
79
        }
80
81
        free(tmp);
82 }
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
 5 Data Structures
 6 Project 2019
 7
 8 Glarakis George AM: 1059561
 9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 void quick(int *array, int first, int last);
16
17 int * Quick_Sort(int *array, int n)
18 {
19
        counter = 0;
20
        quick(array, 0, n - 1);
21
        return array;
22 }
23
24 void quick(int *array, int first, int last)
25 {
26
        int pivot, j, temp, i;
27
        if (first<last)</pre>
28
        {
29
            pivot = first;
30
            i = first;
31
            j = last;
32
33
            while (i<j)
34
            {
                while (array[i] <= array[pivot] && i < last)</pre>
35
36
37
                    i++;
38
                    counter++;
39
40
                while (array[j] > array[pivot])
42
                    j--;
43
                    counter++;
44
                if (i<j)</pre>
45
46
47
                    temp = array[i];
48
                    array[i] = array[j];
                    array[j] = temp;
49
```

```
...s\source\repos\DataStructures\DataStructures\Quick_Sort.h
```

```
50
           }
51
52
53
           temp = array[pivot];
54
           array[pivot] = array[j];
55
           array[j] = temp;
56
           quick(array, first, j - 1);
           quick(array, j + 1, last);
57
58
       }
59 }
60
```

2

```
...is\source\repos\DataStructures\DataStructures\Heap_Sort.h
```

```
1
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 // A heap has current size and array of elements
16 struct MaxHeap
17 {
18
       int size;
19
       int* array;
20 };
21
22 // A utility function to swap to integers
23 void swap(int* a, int* b)
24 {
       int t = *a; *a = *b; *b = t;
25
26 }
27
28 // The main function to heapify a Max Heap. The function
29 // assumes that everything under given root (element at
30 // index idx) is already heapified
31 void maxHeapify(struct MaxHeap* maxHeap, int idx)
32 {
33
       int largest = idx; // Initialize largest as root
       int left = (idx << 1) + 1; // left = 2*idx + 1</pre>
34
       int right = (idx + 1) << 1; // right = 2*idx + 2
35
36
37
                                   // See if left child of root exists and is
                       greater than
38
                                   // root
       if (left < maxHeap->size && maxHeap->array[left] > maxHeap->array
         [largest])
40
           largest = left;
41
       // See if right child of root exists and is greater than
42
43
       // the largest so far
       if (right < maxHeap->size && maxHeap->array[right] > maxHeap->array
         [largest])
           largest = right;
45
46
```

```
...is\source\repos\DataStructures\DataStructures\Heap_Sort.h
```

```
// Change root, if needed
       if (largest != idx)
48
49
            swap(&maxHeap->array[largest], &maxHeap->array[idx]);
50
51
           maxHeapify(maxHeap, largest);
52
        }
53
54
       counter++;
55 }
56
57 // A utility function to create a max heap of given capacity
58 struct MaxHeap* createAndBuildHeap(int *array, int size)
59 {
60
        int i;
61
        struct MaxHeap* maxHeap = (struct MaxHeap*) malloc(sizeof(struct
         MaxHeap));
62
        maxHeap->size = size;
                               // initialize size of heap
        maxHeap->array = array; // Assign address of first element of array
63
64
65
                                // Start from bottommost and rightmost internal
                        mode and heapify all
                                // internal modes in bottom up way
66
67
        for (i = (maxHeap->size - 2) / 2; i >= 0; --i)
68
            maxHeapify(maxHeap, i);
69
        return maxHeap;
70 }
71
72 // The main function to sort an array of given size
73 int * Heap_Sort(int *array, int size)
74 {
75
        counter = 0;
76
77
        // Build a heap from the input data.
        struct MaxHeap* maxHeap = createAndBuildHeap(array, size);
78
79
        // Repeat following steps while heap size is greater than 1.
80
81
        // The last element in max heap will be the minimum element
82
       while (maxHeap->size > 1)
83
84
            // The largest item in Heap is stored at the root. Replace
85
            // it with the last item of the heap followed by reducing the
            // size of heap by 1.
86
            swap(&maxHeap->array[0], &maxHeap->array[maxHeap->size - 1]);
87
            --maxHeap->size; // Reduce heap size
88
89
                              // Finally, heapify the root of tree.
90
91
            maxHeapify(maxHeap, 0);
92
       }
93
```

```
...is\source\repos\DataStructures\Heap_Sort.h
94     return array;
```

```
95 }
```

```
1 /*-----
2 Computer Engineering and Informatics Department
3 University of Patras
5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 int Linear_Searching(int *array, int key, int n)
16 {
       counter = 0;
17
18
       int i = 0;
       while (i < n)
19
20
          if (array[i] == key)
21
22
              return i;
23
          else
24
              i++;
25
26
          counter++;
27
       }
28
       return -1;
29 }
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
 4
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 int Binary_Searching(int *array, int key, int n)
16 {
17
       counter = 0;
       return Binary_Search(array, 0, n, key);
18
19 }
20
21 int Binary_Search(int *array, int 1, int r, int key)
22 {
23
       if (r >= 1)
24
       {
25
           counter++;
26
           int mid = 1 + (r - 1) / 2;
27
28
29
           // If the element is present at the middle
30
           // itself
31
           if (array[mid] == key)
32
               return mid;
33
34
           // If element is smaller than mid, then
35
           // it can only be present in left subarray
           if (array[mid] > key)
36
37
               return Binary_Search(array, 1, mid - 1, key);
38
           // Else the element can only be present
39
40
           // in right subarray
41
           return Binary_Search(array, mid + 1, r, key);
42
       }
43
       // We reach here when element is not
44
45
       // present in array
       return -1;
46
47 }
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
 4
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
int Interpolation_Search(int array[], int key, int n)
16 {
17
       counter = 0;
18
       int left = 0;
19
       int right = (n - 1);
20
21
       // Since array is sorted, an element present
       // in array must be in range defined by corner
22
23
       while (left <= right && key >= array[left] && key <= array[right])</pre>
24
       {
25
           if (left == right) {
26
               if (array[left] == key) return left;
27
               return -1;
28
           // Probing the position with keeping
29
30
           // uniform distribution in mind.
31
           int next = left + (((double)(right - left) /
32
               (array[right] - array[left]))*(key - array[left]));
33
34
           // Condition of target found
35
           if (array[next] == key)
36
               return next;
37
38
           // If x is larger, x is in upper part
39
           if (array[next] < key)</pre>
40
               left = next + 1;
41
42
           // If x is smaller, x is in the lower part
43
           else
44
               right = next - 1;
45
           counter++;
46
       }
47
       return -1;
48 }
49
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 int BIS(int *array, int key, int n)
16 {
17
       int left = 1; //pointing the first element of sublist
       int right = n; //pointing the last element of sublist
18
       int size = right - left + 1; //size of sublist
19
       int next = (size * ((key - array[left]) / (array[right] - array[left]))) + >
20
          1; //prediction of position
       int i = 0;
21
22
       int temp = 0;
23
       counter = 0;
24
25
       // Handling Exception (key greater than max)
26
       if (key > array[n - 1])
27
           return -1;
28
29
       while (key != array[next])
30
31
           i = 0;
32
           size = right - left + 1;
33
34
           // if the sublist is too small it performs linear search
           if (size <= 3)
35
36
           {
37
               while (i < right)</pre>
38
39
                   if (array[left + i] == key)
40
                       return left + i;
                   else
41
42
                       i++;
43
                   counter++;
44
               }
45
           }
46
47
           // narrow the sublist
           if (key >= array[next])
48
```

```
...taStructures\DataStructures\Binary_Interpolation_Search.h
                                                                                        2
49
50
                temp = next + (i * sqrt(size)) - 1;
51
                if (temp >= n)
52
                    temp = n - 1;
53
                while (key > array[temp])
54
55
                    i++;
56
                    temp = next + (i * sqrt(size)) - 1;
                    if (temp >= n)
57
58
                         temp = n - 1;
                    counter++;
59
60
                }
61
                right = next + (i * sqrt(size));
62
                left = next + ((i - 1) * sqrt(size));
63
            }
64
65
            else if (key < array[next])</pre>
66
                temp = next - (i * sqrt(size)) + 1;
67
68
                if (temp < 0)
69
                    temp = 0;
70
                while (key < array[temp])</pre>
71
72
                    i++;
73
                    temp = next - (i * sqrt(size)) + 1;
74
                    if (temp < 0)
75
                         temp = 0;
76
                    counter++;
77
                }
78
                right = next - ((i - 1) * sqrt(size));
79
                left = next - (i * sqrt(size));
80
            }
81
82
            // Handling Exception (exceeding array bounds)
83
            if (left < 0)
                left = 0;
84
85
            if (right >= n)
86
                right = n;
87
            if (array[left] == key)
88
                return left;
89
            if (array[right] == key)
90
                return right;
91
92
            next = left + ((right - left + 1) * ((key - array[left]) / (array
              [right] - array[left]))) - 1;
93
        }
94
95
        if (key == array[next])
96
            return next;
```

3

```
...taStructures\DataStructures\Binary_Interpolation_Search.h
97 else
```

return -1;

98

99 }

```
1 /*-----
2 Computer Engineering and Informatics Department
3 University of Patras
5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern unsigned long int counter;
14
15 int Imp_BIS(int *array, int key, int n)
16 {
17
       int left = 1; //pointing the first element of sublist
       int right = n; //pointing the last element of sublist
18
       int size = right - left + 1; //size of sublist
19
       int next = (size * ((key - array[left]) / (array[right] - array[left]))) +>
20
          1; //prediction of position
21
       int i,j,k = 0;
22
       int temp = 0;
23
       int mid, l, r = 0;
24
       counter = 0;
25
26
       // Handling Exception (key greater than max)
27
       if (key > array[n - 1])
28
           return -1;
29
30
       while (key != array[next])
31
       {
32
           i = 0;
33
           size = right - left + 1;
34
           // if the sublist is too small it performs linear search
35
           if (size <= 3)
36
37
           {
38
               while (i < right)</pre>
39
               {
40
                   if (array[left + i] == key)
41
                       return left + i;
42
                   else
43
                       i++;
44
                   counter++;
45
               }
46
           }
47
           // narrow the sublist
48
```

```
\dots \verb|res\DataStructures\Improved_Binary_Interpolation_Search.h|
```

```
2
```

```
if (key >= array[next])
50
                temp = next + ((2 ^i) * sqrt(size)) - 1;
51
52
                if (temp >= n)
53
                     temp = n - 1;
54
                while (key > array[temp])
55
                {
56
                     i ++;
57
                     temp = next + ((2 ^i) * sqrt(size)) - 1;
58
                     if (temp >= n)
59
                         temp = n - 1;
60
                     counter++;
61
                }
62
                right = next + ((2 ^ i) * sqrt(size));
63
                left = next + ((2 ^(i-1)) * sqrt(size));
            }
64
65
            else if (key < array[next])</pre>
66
67
                temp = next - ((2 ^i) * sqrt(size)) + 1;
68
69
                if (temp < 0)
70
                     temp = 0;
71
                while (key < array[temp])</pre>
72
73
                     i ++;
74
                     temp = next - ((2 ^ i)* sqrt(size)) + 1;
75
                     if (temp < 0)
76
                         temp = 0;
77
                     counter++;
78
                }
                right = next - ((2 ^ (i - 1)) * sqrt(size));
79
                left = next - ((2 ^ i) * sqrt(size));
80
81
            }
82
83
            // Binary Search
84
            j = 0;
            1 = 2 ^ (i - 1);
85
86
            r = 2 ^i;
            for (j = 1; j <= r; j++)</pre>
87
88
89
                mid = (1 + r) / 2;
90
                k = next + (mid * sqrt(size));
91
                if (key == array[k])
92
                     return k;
93
                if (key < array[k])</pre>
94
                     r = mid - 1;
95
                else if (key > array[k])
96
                     l = mid + 1;
97
                counter++;
```

```
\underline{\dots} res \backslash \texttt{DataStructures} \backslash \texttt{Imp} roved\_\texttt{Binary\_Interpolation\_Search.h}
                                                                                                 3
 98
 99
100
              // Handling Exception (exceeding array bounds)
101
              if (left < 0)
                   left = 0;
102
              if (right >= n)
103
104
                   right = n;
              if (array[left] == key)
105
106
                   return left;
107
              if (array[right] == key)
108
                   return right;
109
              next = left + ((right - left + 1) * ((key - array[left]) / (array
110
                 [right] - array[left]))) - 1;
         }
111
112
          if (key == array[next])
113
114
              return next;
115
         else
116
              return -1;
117 }
```

```
1 /*-----
2 Computer Engineering and Informatics Department
3 University of Patras
5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 extern int n;
14
15 int * txt_to_array(char file_source[])
16 {
17
       int *array;
18
       array = (int*) malloc(n * sizeof(int));
19
       FILE *file;
20
       file = fopen(file_source, "r");
21
22
       int i = 0;
23
       while (!feof(file))
24
25
26
           fscanf(file, "%d", &array[i]);
27
           i++;
28
       }
29
30
       fclose(file);
31
32
       return array;
33 }
34
```

ΠΕΡΙΕΧΟΜΕΝΑ ΚΩΔΙΚΑ – Part 2

- ALL (main())
- Student.h
 - 1° Ερώτημα
- Binary_Search_Tree_AM.h
- Read_Students_Info_A.h
- File_Close_A.h
- Menu A.h
 - 2° Ερώτημα
- Binary_Search_Tree_Grade.h
- Read_Students_Info_B.h
- File_Close_B.h
- Menu_B.h
 - 3° Ερώτημα
- Hash_Table.h
- Read_Students_Info_Hash.h
- File_Close_ Hash.h
- Menu Hash.h

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
 4
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
11 ----*/
12
13 #include <stdio.h>
14 #include <string.h>
15 #include <stdlib.h>
16 #include "Student.h"
17 #include "Binary_Search_Tree_AM.h"
18 #include "Binary_Search_Tree_Grade.h"
19 #include "Hash_Table.h"
20 #include "Read_Students_Info_A.h"
21 #include "Read_Students_Info_B.h"
22 #include "Read_Students_Info_Hash.h"
23 #include "File_Close_A.h"
24 #include "File Close B.h"
25 #include "File_Close_Hash.h"
26 #include "Menu_A.h"
27 #include "Menu_B.h"
28 #include "Menu_Hash.h"
29
30 const int w = 5;
31
32 void main()
33 {
34
       node_A * root_A = NULL;
35
       node_B * root_B = NULL;
      hash * hash_list = NULL;
36
37
38
      int ch = -1;
39
      int ans = 0;
40
      int wrong_selection = 0;
41
       printf("-----\n\n");
42
43
       printf("Computer Engineering and Informatics Department\n");
      printf("
44
                    University of Patras
                                           \n\n");
45
       printf("
                     Data Structures \n");
       printf("
                     Project 2019
46
                                   \n\n");
47
       printf("
                 Glarakis George AM: 1059561 \n");
      printf("
                 Papadias Epameinondas AM: 1062665 \n");
48
      printf("
                 Prokopiou Giannis AM : 1059554 \n\n");
49
```

```
...ource\repos\Data_Structures_ALL\Data_Structures_ALL\All.c
```

```
2
```

```
printf("----\n\n");
50
51
52
       do {
53
           ans = 0;
54
           wrong_selection = 0;
55
           printf("\n1. Binary Search Tree (by AM)\n");
           printf("2. Binary Search Tree (by Grade)\n");
56
57
           printf("3. Hashing chaining\n");
58
           printf("\nChoose a function: ");
59
           scanf("%d", &ch);
           while ((getchar()) != '\n'); //clear the input buffer
62
           switch (ch)
63
           {
64
           case 1:
               printf("\nBinary Search Tree by AM\n");
65
               root_A = read_file_A(root_A, "Foitites-Vathmologio-DS.txt");
66
               root_A = menu_A(root_A);
67
68
               break;
69
70
           case 2:
               printf("\nBinary Search Tree by Grade\n");
71
               root_B = read_file_B(root_B, "Foitites-Vathmologio-DS.txt");
72
73
               root B = menu B(root B);
74
               break;
75
76
           case 3:
77
               printf("\nHash Table with chains by AM\n");
78
               hash_list = hash_setup(w);
               hash_list = read_file_C(hash_list, "Foitites-Vathmologio-DS.txt");
79
80
               hash_list = menu_C(hash_list);
81
               break;
82
83
           default:
84
               wrong_selection = 1;
85
           }
86
87
           if (wrong_selection == 1)
88
89
               printf("\nPlease choose one of the Menu Numbers!\n");
90
               ans = 1;
91
92
       } while (ans);
93 }
```

```
1 /*-----
2 Computer Engineering and Informatics Department
3 University of Patras
5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
11 ----*/
12
13 // Struct Student
14 typedef struct Student {
15
      char AM[9];
16
      char Name[100];
      char Surname[100];
19
      double Grade;
20 } st;
21
22 // Print the student's info
23 void print_st(st st0)
24 {
      printf("%s\t%s\t\t%.21f\n", st0.AM, st0.Name, st0.Surname, st0.Grade);
25
26 }
27
28 // Functions to make a change at student's info
29 st change_am(st st, char *new_am)
30 {
31
      strcpy(st.AM, new_am);
32
      return st;
33 }
34 st change_name(st st, char *new_name)
35 {
36
      strcpy(st.Name, new_name);
37
      return st;
38 }
39 st change_surname(st st, char *new_surname)
40 {
      strcpy(st.Surname, new_surname);
41
42
      return st;
43 }
44 st change_grade(st st, double new_grade)
45 {
46
      st.Grade = new_grade;
47
      return st;
48 }
49
```

```
\dotse\repos\Data_Structures_ALL\Data_Structures_ALL\Student.h
```

```
50 char * read_change()
51 {
52
       char ch[100];
53
        while ((getchar()) != '\n'); // clear the input buffer
54
       printf("Give the change: ");
       scanf("%s", ch);
55
56
        return ch;
57 }
58
59 double read_grade()
60 {
        double newgrade = -1;
61
62
        printf("Give the change: ");
        scanf("%lf", &newgrade);
63
64
       return newgrade;
65 }
66
67 st choose_change(st st_change)
68 {
69
       int n;
70
       char ch[100];
       printf("1. Change Name\n");
71
72
        printf("2. Change Surame\n");
73
        printf("3. Change Grade\n");
74
       printf("\nChoose a change function: ");
75
       while (1)
76
       {
77
            scanf("%d", &n);
78
            switch (n)
79
80
            case 1:
81
                strcpy(ch, read_change());
82
                st_change = change_name(st_change, ch);
83
                break;
84
            case 2:
85
                strcpy(ch, read_change());
                st_change = change_surname(st_change, ch);
86
87
                break;
88
            case 3:
89
                st_change = change_grade(st_change, read_grade());
90
                break;
            default:
91
92
                printf("Please choose one of the above options: ");
93
            }
            break;
94
95
        }
96
        return st_change;
97 }
```

```
1 /*----
2 Computer Engineering and Informatics Department
3 University of Patras
4
5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
11 ----*/
12
13 // Struct node for the BTS
14 typedef struct node_A
15 {
16
       st st0;
       struct node_A *left, *right;
17
18 } node_A;
19
20 // Create a new Node
21 node_A * newNode_A(st st1)
22 {
23
       node_A *temp = (node_A *)malloc(sizeof(node_A));
24
       temp->st0 = st1;
25
26
       temp->left = temp->right = NULL;
27
       return temp;
28 }
29
30 // Insert a new student to the Binary Search Tree
31 node_A* insert_A(node_A* node_A, st st2)
32 {
33
       // If the tree is empty, return a new node
34
       if (node_A == NULL) return newNode_A(st2);
35
       int res = strcmp(st2.AM, node_A->st0.AM);
36
37
      // Otherwise, recur down the tree
38
39
       if (res < 0)
40
          node_A->left = insert_A(node_A->left, st2);
       else if (res > 0)
41
42
          node_A->right = insert_A(node_A->right, st2);
43
44
       return node A;
45 }
46
47 // Print the BST sorted
48 void inorder_A(node_A *root)
49 {
50
       if (root != NULL)
51
          inorder_A(root->left);
52
53
          print_st(root->st0);
```

```
54
             inorder A(root->right);
 55
        }
 56 }
 57
 58 node_A* search_A(node_A* root, char am[9])
 59 {
 60
        int res = strcmp(am, root->st0.AM);
 61
 62
        // Base Cases: root is null or key is present at root
 63
        if (res == 0)
 64
         {
            return root;
                             }
 65
 66
        // Key is greater than root's key
        if (res > 0)
 67
 68
         {
             if (root->right == NULL)
 69
 70
             {
 71
                 return -1;
 72
             }
 73
             return search_A(root->right, am);
 74
        }
 75
 76
         // Key is smaller than root's key
        else
 77
 78
         {
             if (root->left == NULL)
 79
 80
             {
 81
                 return -1;
 82
             }
 83
             return search_A(root->left, am);
 84
         }
 85 }
 86
 87 // Find the minimum value of the BTS
 88 node_A * minValueNode_A(node_A* node_A)
 89 {
        while (node_A->left != NULL)
 90
 91
         {
 92
             node_A = node_A->left;
 93
         }
 94
        return node_A;
 95 }
 96
 97 // Delete a node from the BTS
 98 node_A* deleteNode_A(node_A* root, st st)
99 {
100
         if (root == NULL) return root;
101
102
        int res = strcmp(st.AM, root->st0.AM);
103
        // If the key to be deleted is smaller than the root's key,
104
        // then it lies in left subtree
105
        if (res < 0)
106
```

```
107
             root->left = deleteNode A(root->left, st);
108
        // If the key to be deleted is greater than the root's key,
109
        // then it lies in right subtree
110
111
        else if (res > 0)
            root->right = deleteNode_A(root->right, st);
112
113
        // if key is same as root's key, then this is the node
114
115
        // to be deleted
116
        else
117
        {
             // node with only one child or no child
118
119
            if (root->left == NULL)
120
121
                 node_A *temp = root->right;
122
                 free(root);
123
                 return temp;
124
            }
            else if (root->right == NULL)
125
126
                 node_A *temp = root->left;
127
                 free(root);
128
129
                 return temp;
130
            }
131
132
            // node with two children: Get the inorder successor (smallest
133
            // in the right subtree)
134
            node_A* temp = minValueNode_A(root->right);
135
            // Copy the inorder successor's content to this node
136
137
            root->st0 = temp->st0;
138
139
            // Delete the inorder successor
            root->right = deleteNode_A(root->right, temp->st0);
140
141
        }
        return root;
142
143 }
144
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 // Read students' info from a file and make a BTS by the AMs
14 node_A * read_file_A(node_A * root, char students_data[])
15 {
16
       st student;
17
       // open the given file
18
19
       FILE const *file;
       file = fopen(students_data, "r");
20
21
       if (file == NULL)
22
23
24
           printf("Error in reading Students' Data...");
25
        }
26
27
       int i = 0;
28
       while (!feof(file))
29
30
            // Scan for student's info
31
           fscanf(file, "%s %s %s %lf\n", student.AM, student.Name,
              student.Surname, &student.Grade);
32
33
           // Insert the new student to the BTS
            if (i == 0)
35
36
                root = insert_A(root, student);
37
            }
38
           else
39
                insert_A(root, student);
40
41
42
           i++;
43
        }
       fclose(file);
44
45
        return root;
46 }
47
```

```
1 /*-----
2 Computer Engineering and Informatics Department
3 University of Patras
4
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 // Print the sudents sorted by AM in the file
14 void write_inorder_A(FILE *file, node_A *root)
15 {
16
       if (root != NULL)
17
       {
           write_inorder_A(file, root->left);
           fprintf(file, "%s %s %s %.2lf\n", root->st0.AM, root->st0.Name, root-
19
             >st0.Surname, root->st0.Grade);
20
          write_inorder_A(file, root->right);
21
       }
22 }
23
24 // Create a new file to save the BTS
25 void File_Close_A(node_A *root, char *dir)
26 {
27
       FILE *file;
28
       file = fopen(dir, "w+");
29
       write_inorder_A(file, root);
30
31
       fclose(file);
32 }
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
4
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
11 ----*/
12
13 char * AM_read_A()
14 {
15
       char am[100];
       printf("Give the student's AM: ");
16
17
       gets(am);
       return am;
18
19 }
20
21 int answer_A()
22 {
23
       char ans = NULL;
24
25
       do
26
       {
27
           ans = getchar();
28
           while ((getchar()) != '\n'); //clear the input buffer
29
           if (ans == 'N' || ans == 'n')
30
31
32
               return 0;
33
           else if (ans == 'Y' || ans == 'y')
34
35
           {
36
               return 1;
37
38
           else if (ans != 'N' && ans != 'n' && ans != 'Y' && ans != 'y')
39
40
               printf("Press (Y/N): ");
41
           }
42
43
       } while (1);
44 }
45
46 // Check if the AM exists
47 node_A * search_AM_A(node_A * root)
48 {
49
       char am[100];
```

```
50
        node_A *node_A;
51
        do
52
        {
53
            strcpy(am, AM_read_A());
54
            node_A = search_A(root, am);
55
            if (node_A == -1)
                printf("\nThis AM does not exist! Try again!\n");
56
57
        } while (node_A == -1);
58
59
        return node_A;
60 }
61
62 node_A* menu_A(node_A *root)
63 {
64
        int n;
        node_A *node_A;
65
66
        int wrong_selection = 0;
        int ans = -1;
67
68
69
       do
70
        {
71
            wrong_selection = 0;
72
            printf("\n1. Sort List\n");
73
            printf("2. Search\n");
            printf("3. Make a Change\n");
74
75
            printf("4. Delete\n");
            printf("5. Exit\n");
76
77
            printf("\nChoose a function: ");
78
79
            scanf("%d", &n);
80
            while ((getchar()) != '\n'); //clear the input buffer
81
            switch (n)
82
            {
83
            // Print the soerted BTS
84
            case 1:
85
                printf("AM\t\tFirst Name\tLast Name\tGrade\n\n");
                inorder_A(root);
86
87
                break;
88
89
            // Search for a student by the AM
90
                node A = search AM A(root);
91
92
                print_st(node_A->st0);
93
                break;
94
95
            // Make a change to a student's info
            case 3:
96
97
                do
                {
98
```

```
\dots \texttt{ce} \texttt{\partial} \texttt{Structures\_ALL} \texttt{\partial} \texttt{
```

```
3
```

```
99
                     node_A = search_AM_A(root);
                     node_A->st0 = choose_change(node_A->st0);
100
101
                     print_st(node_A->st0);
                     printf("Would you like to change something else?(Y/N) ");
102
103
                     while ((getchar()) != '\n'); //clear the input buffer
104
                 } while (answer_A());
105
                 break;
106
107
             // Delete a student from the BTS
             case 4:
108
109
                 node_A = search_AM_A(root);
                 root = deleteNode_A(root, node_A->st0);
110
111
                 printf("\nDeletion was successful!\n");
112
                 break;
113
             // Exit the program
114
115
             case 5:
                 break;
116
117
             default:
118
119
                 wrong_selection = 1;
120
             }
121
             if (n == 5)
122
                 break;
123
             else if (wrong_selection == 1)
124
125
             {
126
                 printf("\nPlease choose one of the Menu Numbers!\n");
127
                 ans = 1;
128
             }
129
             else
             {
130
131
                 printf("\nWould you like to call an other function?(Y/N): ");
132
                 ans = answer_A();
133
             }
134
         } while (ans);
135
136
         // Write in a new file the new changed BTS of students
         File_Close_A(root, "New_Foithtes_A.txt");
137
138
139
         return root;
140 }
141
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
4
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
11 ----*/
12
13 // Struct st_node for the linked list of BTS nodes
14 typedef struct st_node
15 {
16
       st st;
17
       struct st_node * next;
19
20 // Struct node for the BTS
21 typedef struct node_B
22 {
23
       struct node_B *left, *right;
24
      st node * head;
      int counter;
25
26 } node_B;
27
28 // Create a new Node
29 node_B *newNode_B(st st1)
30 {
31
       node_B *temp = (node_B *)malloc(sizeof(node_B));
32
33
      temp->counter = 0;
34
       temp->head = (st_node*)malloc(sizeof(st_node));
35
       temp->head->st = st1;
36
37
       temp->head->next = NULL;
38
      temp->left = temp->right = NULL;
39
40
       return temp;
41 }
42
43 // Create a new st_node (if a student with the same grade already exists)
44 st_node * new_st_node(st st)
45 {
46
       st_node* temp = (st_node*)malloc(sizeof(st_node));
47
       temp->st = st;
       temp->next = NULL;
48
49 }
```

```
50
51 // Create a new st_node in an existing Node
52 node_B *existingNode(node_B *node_B, st st1)
53 {
54
       node_B->counter++;
55
       st_node * current = node_B->head;
56
57
       // Find the last student in the linked list
58
       while (current->next != NULL)
59
60
            current = current->next;
61
       }
62
       current->next = new_st_node(st1);
63
64
       return node_B;
65 }
66
67 void print_list(node_B * root) {
68
69
       st node * current = root->head;
70
71
       while (current != NULL) {
72
            print_st(current->st);
73
            current = current->next;
74
       }
75 }
76
77 int double_to_int(double a)
78 {
79
       return (int)a * 100;
80 }
81
82 // Insert a new student to the Binary Search Tree
83 node_B* insert_B(node_B* node_B, st st2)
84 {
85
       // If the tree is empty, create a new node
       if (node_B == NULL) return newNode_B(st2);
86
87
88
       int node_grade = double_to_int(node_B->head->st.Grade);
89
       int st_grade = double_to_int(st2.Grade);
90
       int res = node_grade - st_grade;
91
92
       // If the grade already exists, create a new st_node
93
                       return existingNode(node_B, st2);
       if (res == 0)
94
95
       // Otherwise, recur down the tree
96
       if (res > 0)
97
            node_B->left = insert_B(node_B->left, st2);
98
       else if (res < 0)</pre>
```

```
node_B->right = insert_B(node_B->right, st2);
99
100
101
        return node_B;
102 }
103
104 // A utility function to do inorder traversal of BST
105 void inorder_B(node_B *root)
106 {
        if (root != NULL)
107
108
        {
109
            inorder_B(root->left);
            print_list(root);
110
            inorder_B(root->right);
111
112
        }
113 }
114
115 // Find the minimum value of the BTS
116  node_B * minValueNode_B(node_B* node_B)
117 {
        while(node_B->left != NULL)
118
119
            node_B = node_B->left;
120
121
        return node_B;
122
123 }
124
125 // Find the maximum value of the BTS
126  node_B * maxValueNode_B(node_B* node_B)
127 {
128
        while(node B->right != NULL)
129
            node_B = node_B->right;
130
131
132
        return node_B;
133 }
134
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 // Read students' info from a file and make a BTS by the Grade
14 node_B * read_file_B(node_B * root, char students_data[])
15 {
16
       st student;
17
       // open the given file
18
       FILE *file;
19
       file = fopen(students_data, "r");
20
21
       if (file == NULL)
22
23
24
           printf("Error in reading Student Data...");
25
       }
26
27
       int i = 0;
28
       while (!feof(file))
29
30
            // Scan for student's info
31
           fscanf(file, "%s %s %s %lf\n", student.AM, student.Name,
             student.Surname, &student.Grade);
32
33
           // Insert the new student to the BTS
            if (i == 0)
            { root = insert_B(root, student); }
35
36
           else
37
               insert_B(root, student);
38
           i++;
39
       }
40
41
       fclose(file);
42
       return root;
43 }
44
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 // Print the sudents sorted by grade in the file
14 void write_inorder_B(FILE *file, node_B *root)
15 {
16
       if (root != NULL)
17
       {
           st node * current = root->head;
18
           write_inorder_B(file, root->left);
19
           while (current != NULL) {
20
21
               fprintf(file, "%s %s %s %.21f\n", current->st.AM, current-
                 >st.Name, current->st.Surname, current->st.Grade);
22
               current = current->next;
23
           }
24
           write_inorder_B(file, root->right);
25
       }
26 }
27
28 // Create a new file to save the BTS
29 void File_Close_B(node_B *root, char *dir)
30 {
31
       FILE *file;
32
       file = fopen(dir, "w+");
33
       write_inorder_B(file, root);
34
35
       fclose(file);
36 }
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
 4
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 int answer_B()
14 {
15
       char ans = NULL;
16
       do
17
       {
18
            ans = getchar();
           while ((getchar()) != '\n'); //clear the input buffer
19
20
            if (ans != 'N' && ans != 'n' && ans != 'Y' && ans != 'y')
21
22
            {
23
                printf("Press (Y/N): ");
24
            }
            else if (ans == 'N' || ans == 'n')
25
26
27
                return 0;
28
            else if (ans == 'Y' || ans == 'y')
29
30
31
                return 1;
32
            }
33
34
       } while (1);
35 }
36
37 node_B* menu_B(node_B *root)
38 {
39
       int n;
40
        char AM[9];
       node_B *node_B;
42
        int ans = -1;
43
       int wrong_selection = 0;
44
       do
45
46
       {
47
            wrong_selection = 0;
48
            printf("\n1. Minimun Grade\n");
            printf("2. Maximun Grade\n");
49
```

```
50
            printf("3. Exit\n");
            printf("\nChoose a function: ");
51
52
53
            scanf("%d", &n);
            while ((getchar()) != '\n'); //clear the input buffer
54
55
            switch (n)
56
            {
57
            // Print a list of the students with the minimum grade
58
            case 1:
59
                print_list(minValueNode_B(root));
60
                break;
61
62
            // Print a list of the students with the maximum grade
63
            case 2:
                print_list(maxValueNode_B(root));
64
65
                break;
66
67
            // Exit the program
68
            case 3:
69
                break;
70
71
            default:
72
                printf("Please choose one of the Menu Numbers!");
73
                wrong_selection = 1;
                ans = 1;
74
75
            }
76
77
            if (n == 3)
78
                break;
            else if (wrong_selection == 0)
79
80
                printf("\nWould you like to call an other function?(Y/N): ");
81
82
                ans = answer_B();
83
            }
        } while (ans);
84
85
        // Write in a new file the new changed BTS of students
86
87
        File_Close_B(root, "New_Foithtes_B.txt");
88
89
       return root;
90 }
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
11 ----*/
12
13 extern const int w;
14
15 // Struct node for the linked lists
16 typedef struct node_C {
17
       st student;
       struct node_C *next_node;
18
20
21 // Struct hash for the Hash Table
22 typedef struct hash {
23
       int value;
24
      node_C *node_C;
25 } hash;
26
27 // Initialize the Hash Table as NULL
28 hash* hash_setup()
29 {
30
      hash* hash_list = (hash*)calloc(w, sizeof(hash));
31
32
      for (int i = 0; i < w; i++)</pre>
33
       {
34
          hash_list[i].value = i;
35
          hash_list[i].node_C = NULL;
36
       }
37
38
       return hash_list;
39 }
40
41 // Create a new node (head of the linked list)
42 node C* newHashNode(st st)
43 {
44
       node_C *temp = (node_C *)malloc(sizeof(node_C));
45
       temp->student = st;
46
       temp->next_node = NULL;
47
48
       return temp;
49 }
```

```
50
51 // Find the hash of student
52 int pos_finder(char *AM)
53 {
54
        int counter = 0;
55
        int sum = 0;
56
       for (int i = 0; i < strlen(AM); i++)</pre>
57
58
            counter = AM[i];
59
            sum += counter;
60
        }
61
        sum = sum % w;
62
        return sum ;
63 }
64
65 // Insert the new student to the linked list
66 node_C * insertNode(node_C* node_C, st st)
67 {
       while (node_C != NULL)
68
69
            node_C->next_node = insertNode(node_C->next_node, st);
70
71
            break;
72
73
        if (node C == NULL)
           node_C = newHashNode(st);
74
75
        return node_C;
76 }
77
78 // Insert the new student to the Hash Table
79 hash* insert_Hash(hash* hash_list, st st)
80 {
81
        int pos = pos_finder(st.AM, w);
82
       hash_list[pos].node_C = insertNode(hash_list[pos].node_C, st);
83
84
        return hash_list;
85 }
86
87 // Search a student by AM in tha linked list
88 node_C * search_next(char AM[9], node_C* node_C)
89 {
90
        int res = strcmp(AM, node_C->student.AM);
91
        if (res == 0)
92
            return node_C;
93
       else
            if (node_C->next_node == NULL)
94
                return -1; }
95
            search_next(AM, node_C->next_node);
96
97 }
98
```

```
99 // Search a student by AM in tha Hash Table
100 node_C * search_st(hash* hash_list, char AM[9])
101 {
        int i = pos_finder(AM);
102
103
        int res = strcmp(AM, hash_list[i].node_C->student.AM);
104
        if (res == 0)
105
             return hash_list[i].node_C;
106
        else
107
             return search_next(AM, hash_list[i].node_C->next_node);
108 }
109
110 // Delete a student from the middle or the end of the linked list
111 node_C * del_mid_node(node_C * prev, node_C * current, node_C * del_node)
112 {
113
        while (current != del node)
114
115
             current = del mid node(current, current->next node, del node);
116
            break;
117
        }
        if (current == del node)
118
119
            prev->next_node = current->next_node;
120
121
        return prev;
122 }
123
124 // Delete a student from the Hash Table
125 hash * delete_node(hash * hash_list, node_C * del_node)
126 {
127
        int pos = pos_finder(del_node->student.AM);
128
        // Delete the student if it's the fist in the linked list
129
        if (del_node == hash_list[pos].node_C)
130
131
        {
             hash_list[pos].node_C = del_node->next_node;
132
133
            free(del_node);
134
        }
135
136
        // Delete if it's in the middle or at the end
137
        else
138
        {
             hash_list[pos].node_C = del_mid_node(hash_list[pos].node_C, hash_list >
139
               [pos].node C->next node, del node);
             free(del_node);
140
        }
141
142
        return hash_list;
143
144 }
145
146 void print_table(hash* hash_list)
```

```
\underline{\dots} epos \\ \texttt{Data\_Structures\_A} \\ \texttt{LL} \\ \texttt{Data\_Structures\_ALL} \\ \texttt{Hash\_Table.h} \\
```

```
4
```

```
<del>147</del> {
148
         node_C* current = NULL;
149
         for (int i = 0; i < w; i++)</pre>
150
         {
151
             current = hash_list[i].node_C;
152
             while (current != NULL)
153
154
                  print_st(current->student);
155
                  current = current->next_node;
156
             }
157
         }
158 }
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
 3 University of Patras
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 // Read students' info from a file and make a Hash Table by the AMs
14 hash * read_file_C(hash * hash_list, char students_data[])
15 {
16
       st student;
17
       // open the given file
18
       FILE *file;
19
       file = fopen(students_data, "r");
20
21
       if (file == NULL)
22
23
24
           printf("Error in reading Student Data...");
25
       }
26
27
       int i = 0;
28
       while (!feof(file))
29
30
           // Scan for student's info
31
           fscanf(file, "%s %s %s %lf\n", student.AM, student.Name,
             student.Surname, &student.Grade);
32
33
           // Insert the new student to the Hash Table
34
           insert_Hash(hash_list, student);
35
       }
36
37
       fclose(file);
38
       return hash_list;
39 }
40
```

```
1 /*-----
 2 Computer Engineering and Informatics Department
3 University of Patras
 5 Data Structures
6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
12
13 // Create a new file to save the Hash Table
14 void File_Close_C(hash * hash_list, char *dir)
15 {
16
       FILE *file;
       file = fopen(dir, "w+");
17
18
       node_C* current = NULL;
19
20
       for (int i = 0; i < w; i++)</pre>
21
           current = hash_list[i].node_C;
22
23
           while (current != NULL)
24
           {
               fprintf(file, "%s %s %s %.21f\n", current->student.AM, current-
25
                 >student.Name, current->student.Surname, current->student.Grade);
26
               current = current->next_node;
27
           }
28
       }
29
       fclose(file);
30 }
```

```
2 Computer Engineering and Informatics Department
 3 University of Patras
4
 5 Data Structures
 6 Project 2019
7
8 Glarakis George AM: 1059561
9 Papadias Epameinondas AM: 1062665
10 Prokopiou Giannis AM: 1059554
11 ----*/
12
13 char * AM_read_C()
14 {
15
       char am[100];
       printf("Give the student's AM: ");
16
17
       gets(am);
       return am;
18
19 }
20
21 int answer_C()
22 {
23
       char ans = NULL;
24
25
       do
26
       {
27
           ans = getchar();
28
           while ((getchar()) != '\n'); //clear the input buffer
29
           if (ans == 'N' || ans == 'n')
30
31
32
               return 0;
33
           else if (ans == 'Y' || ans == 'y')
34
35
           {
36
               return 1;
37
38
           else if (ans != 'N' && ans != 'n' && ans != 'Y' && ans != 'y')
39
40
               printf("Press (Y/N): ");
41
42
       } while (1);
43 }
44
45 // Check if the AM exists
46 node_C * search_AM_C(hash* hash_list)
47 {
48
       char am[100];
       node_C *node_C;
49
```

```
...repos\Data_Structures_ALL\Data_Structures_ALL\Menu_Hash.h
```

```
2
```

```
50
        do
51
        {
52
            strcpy(am, AM_read_C());
53
            node_C = search_st(hash_list, am);
54
            if (node_C == -1)
55
                printf("\nThis AM does not exist! Try again!\n");
56
        } while (node_C == -1);
57
58
        return node_C;
59 }
60
61 hash * menu_C(hash * hash_list)
62 {
63
        int n;
64
        char AM[9];
        node_C* node_C;
65
66
        int wrong_selection = 0;
67
        int ans = -1;
68
69
        do
70
        {
71
            wrong_selection = 0;
72
            printf("1. Search\n");
73
            printf("2. Make a Change\n");
            printf("3. Delete\n");
74
75
            printf("4. Exit\n");
            printf("\nChoose a function: ");
76
77
            scanf("%d", &n);
78
            while ((getchar()) != '\n'); //clear the input buffer
79
80
81
            switch (n)
82
83
            // Search for a student by the AM
84
            case 1:
85
                node C = search AM C(hash list);
                print_st(node_C->student);
86
87
                break;
88
89
            // Make a change to a student's info
            case 2:
90
                do
91
92
                {
93
                    node_C = search_AM_C(hash_list);
94
                    node_C->student = choose_change(node_C->student);
95
                    print_st(node_C->student);
96
                    printf("Would you like to change something else?(Y/N) ");
97
                    while ((getchar()) != '\n'); //clear the input buffer
98
                } while (answer_C());
```

```
99
                 break;
100
101
             // Delete a student from the Hash Table
102
             case 3:
                 node_C = search_AM_C(hash_list);
103
                 hash_list = delete_node(hash_list, node_C);
104
                 printf("\nThe deletion was successful!\n");
105
106
                 break;
107
108
             // Exit the program
109
             case 4:
                 break;
110
111
             default:
112
                 wrong_selection = 1;
113
114
             }
115
116
             if (n == 4)
                 break;
117
             else if (wrong_selection == 1)
118
119
120
                 printf("\nPlease choose one of the Menu Numbers!\n");
121
                 ans = 1;
122
             }
             else
123
124
                 printf("\nWould you like to call an other function?(Y/N): ");
125
126
                 ans = answer_C();
127
             }
         } while (ans);
128
129
         File_Close_C(hash_list, "New_Foithtes_C.txt");
130
131
132
        return hash_list;
133 }
```