UNIVERSITY OF PATRAS
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

# ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ: PROJECT 2019

**Γλαράκης Γιώργος ΑΜ: 1059561**

**Παπαδιάς Επαμεινώνδας ΑΜ: 1062665**

**Προκοπίου Ιωάννης ΑΜ: 1059554**

CEID
COMPUTER ENGINEERING & INFORMATICS DEPARTMENT

# 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.

1. 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 I 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

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "Bubble_Sort.h"
#include "Insertion_Sort.h"
#include "Selection_Sort.h"
#include "Merge_Sort.h"
#include "Quick_Sort.h"
#include "Heap_Sort.h"
#include "Linear_Searching.h"
#include "Binary_Searching.h"
#include "Interpolation_Search.h"
#include "Binary_Interpolation_Search.h"
#include "Improved_Binary_Interpolation_Search.h"
#include "TXT-to-ARRAY.h"

unsigned long int counter = 0;
int n = 100000;

int main()
{
    int Search_Result = -1; //item found flag
    int sorted = 1; //sorted flag
    int ch; //switch choice variable

    int *list;
    list = txt_to_array("integers.txt");

    clock_t start, end;
    double cpu_time_used;

    printf("--------------------------------------------------\n\n");
    printf("Computer Engineering and Informatics Department\n");
    printf("        University of Patras    \n\n");
    printf("        Data Structures \n");
    printf("        Project 2019    \n\n");
    printf("    Glarakis George AM: 1059561 \n");
    printf("    Papadias Epameinondas AM: 1062665 \n" );
    printf("    Prokopiou Giannis AM : 1059554 \n\n" );
    printf("--------------------------------------------------\n\n" );

    do {
        printf("1. Bubble Sort\n");
        printf("2. Insertion Sort\n");
        printf("3. Selection Sort\n");
        printf("4. Merge Sort\n");
        printf("5. Quick Sort\n");
        printf("6. Heap Sort\n");
        printf("\nChoose a sorting function: " );

        scanf("%d", &ch);
        while ((getchar()) != '\n'); //clear the input buffer
        switch (ch)
        {
        case 1:
            start = clock();
            list = Bubble_Sort(list, n);
            end = clock();
            break;

        case 2:
            start = clock();
            list = Insertion_Sort(list, n);
            end = clock();
            break;

        case 3:
            start = clock();
            list = Selection_Sort(list, n);
            end = clock();
            break;

        case 4:
            start = clock();
            list = Merge_Sort(list, n);
            end = clock();
            break;

        case 5:
            start = clock();
            list = Quick_Sort(list, n);
            end = clock();
            break;

        case 6:
            start = clock();
            list = Heap_Sort(list, n);
            end = clock();
            break;

        default:
            printf("\nPlease choose one of the sorting functions!\n\n");
            sorted = 0;
        }

    } while (sorted == 0);

    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("\nList sorted!\n");
    printf("Counter: %d\n", counter);
    printf("Time: %lf\n\n", cpu_time_used);
    printf("%d \n %d \n %d \n", list[0], list[50000], list[n-1]);

    if (sorted == 1)
    {
        int num; //the number user is searching
        int wrong_ch = 0; //wrong choice

        do {
            wrong_ch = 0;
            printf("\nType the number you are searching for: ");
            scanf("%d", &num);

            printf("\n1. Linear Searching\n");
            printf("2. Binary Searching\n");
            printf("3. Interpolation Searching\n");
            printf("4. Binary Interpolation Searching\n");
            printf("5. Improved Binary Interpolation Searching\n");
            printf("\nChooe one of the searching functions: ");
            scanf("%d", &ch);
            switch (ch)
            {
            case 1:
                start = clock();
                Search_Result = Linear_Searching(list, num, n);
                end = clock();
                break;
            case 2:
                start = clock();
                Search_Result = Binary_Searching(list, num, n);
                end = clock();
                break;
            case 3:
                start = clock();
```

```c
                Search_Result = Interpolation_Search(list, num, n);
                end = clock();
                break;
            case 4:
                start = clock();
                Search_Result = BIS(list, num, n);
                end = clock();
                break;
            case 5:
                start = clock();
                Search_Result = Imp_BIS(list, num, n);
                end = clock();
                break;

            default:
                printf("\nPlease choose one of the searching functions!\n\n");
                wrong_ch = 1;
                break;
            }
            if (Search_Result == -1 && wrong_ch == 0)
                printf("\nThe number you are searching is not in the list!\n");
        } while (Search_Result == -1);
    }

    printf("The number you are searching is located in position: %d \n", Search_Result);
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("\nCounter: %d\n", counter);
    printf("Time: %lf\n", cpu_time_used);

    system("pause");

    return 0;
}
```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern unsigned long int counter;

int * Bubble_Sort(int *array, int n)
{
    int i, d, swap;
    counter = 0;

    for (i = 0; i < (n - 1); i++)
    {
        for (d = 0; d < n - i - 1; d++)
        {
            if (array[d] > array[d + 1]) /* For decreasing order use < */
            {
                swap = array[d];
                array[d] = array[d + 1];
                array[d + 1] = swap;
            }
            counter++;
        }
    }

    return array;
}

```

```
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern unsigned long int counter;

int * Insertion_Sort(int *array, int n)
{
    int i, j, temp;
    counter = 0;

    for (i = 1; i < n; i++)
    {
        j = i;

        while (j > 0 && array[j] < array[j - 1])
        {
            temp = array[j];
            array[j] = array[j - 1];
            array[j - 1] = temp;
            j--;

            counter++;
        }

    }
    return array;
}

```

```c
/*-----------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern unsigned long int counter;

int * Selection_Sort(int *array, int n)
{
    counter = 0;
    int i, j, swap, position;

    for (i = 0; i < (n - 1); i++)
    {
        position = i;

        for (j = i + 1; j < n; j++)
        {
            if (array[position] > array[j])
                position = j;

            counter++;
        }
        if (position != i)
        {
            swap = array[i];
            array[i] = array[position];
            array[position] = swap;
        }
    }

    return array;
}

```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern unsigned long int counter;

void merge(int *array, int min, int mid, int max, int n);
void part(int *array, int min, int max, int n);

int * Merge_Sort(int *array, int n)
{
    counter = 0;
    part(array, 0, n-1, n);
    return array;
}

void part(int *array, int min, int max, int n)
{
    int mid;
    if (min < max)
    {
        mid = (min + max) / 2;
        part(array, min, mid, n);
        part(array, mid + 1, max, n);
        merge(array, min, mid, max, n);
    }
}

void merge(int *arr, int min, int mid, int max, int n)
{

    int *tmp;
    tmp = (int*)malloc(n * sizeof(int));

    int i, j, k, m;
    j = min;
    m = mid + 1;
    for (i = min; j <= mid && m <= max; i++)
    {
        if (arr[j] <= arr[m])
        {
            tmp[i] = arr[j];
            j++;
        }
        else
        {
            tmp[i] = arr[m];
            m++;
        }
        counter++;
    }
    if (j>mid)
    {
        for (k = m; k <= max; k++)
        {
            tmp[i] = arr[k];
            i++;
        }
    }
    else
    {
        for (k = j; k <= mid; k++)
        {
            tmp[i] = arr[k];
            i++;
        }
    }
    for (k = min; k <= max; k++)
    {
        arr[k] = tmp[k];
    }

    free(tmp);
}
```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern unsigned long int counter;

void quick(int *array, int first, int last);

int * Quick_Sort(int *array, int n)
{
    counter = 0;
    quick(array, 0, n - 1);
    return array;
}

void quick(int *array, int first, int last)
{
    int pivot, j, temp, i;
    if (first<last)
    {
        pivot = first;
        i = first;
        j = last;

        while (i<j)
        {
            while (array[i] <= array[pivot] && i < last)
            {
                i++;
                counter++;
            }
            while (array[j] > array[pivot])
            {
                j--;
                counter++;
            }
            if (i<j)
            {
                temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }

        temp = array[pivot];
        array[pivot] = array[j];
        array[j] = temp;
        quick(array, first, j - 1);
        quick(array, j + 1, last);
    }
}

```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern unsigned long int counter;

// A heap has current size and array of elements
struct MaxHeap
{
    int size;
    int* array;
};

// A utility function to swap to integers
void swap(int* a, int* b)
{
    int t = *a; *a = *b;  *b = t;
}

// The main function to heapify a Max Heap. The function
// assumes that everything under given root (element at
// index idx) is already heapified
void maxHeapify(struct MaxHeap* maxHeap, int idx)
{
    int largest = idx;  // Initialize largest as root
    int left = (idx << 1) + 1;  // left = 2*idx + 1
    int right = (idx + 1) << 1; // right = 2*idx + 2

                                // See if left child of root exists and is greater than
                                // root
    if (left < maxHeap->size && maxHeap->array[left] > maxHeap->array[largest])
        largest = left;

    // See if right child of root exists and is greater than
    // the largest so far
    if (right < maxHeap->size && maxHeap->array[right] > maxHeap->array[largest])
        largest = right;

    // Change root, if needed
    if (largest != idx)
    {
        swap(&maxHeap->array[largest], &maxHeap->array[idx]);
        maxHeapify(maxHeap, largest);
    }

    counter++;
}

// A utility function to create a max heap of given capacity
struct MaxHeap* createAndBuildHeap(int *array, int size)
{
    int i;
    struct MaxHeap* maxHeap = (struct MaxHeap*) malloc(sizeof(struct MaxHeap));
    maxHeap->size = size;   // initialize size of heap
    maxHeap->array = array; // Assign address of first element of array

                            // Start from bottommost and rightmost internal mode and heapify all
                            // internal modes in bottom up way
    for (i = (maxHeap->size - 2) / 2; i >= 0; --i)
        maxHeapify(maxHeap, i);
    return maxHeap;
}

// The main function to sort an array of given size
int * Heap_Sort(int *array, int size)
{
    counter = 0;

    // Build a heap from the input data.
    struct MaxHeap* maxHeap = createAndBuildHeap(array, size);

    // Repeat following steps while heap size is greater than 1.
    // The last element in max heap will be the minimum element
    while (maxHeap->size > 1)
    {
        // The largest item in Heap is stored at the root. Replace
        // it with the last item of the heap followed by reducing the
        // size of heap by 1.
        swap(&maxHeap->array[0], &maxHeap->array[maxHeap->size - 1]);
        --maxHeap->size;  // Reduce heap size

                          // Finally, heapify the root of tree.
        maxHeapify(maxHeap, 0);
    }

    return array;
}
```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern unsigned long int counter;

int Linear_Searching(int *array, int key, int n)
{
    counter = 0;
    int i = 0;
    while (i < n)
    {
        if (array[i] == key)
            return i;
        else
            i++;

        counter++;
    }
    return -1;
}
```

```
/*--------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
-------------------------------------------*/

extern unsigned long int counter;

int Binary_Searching(int *array, int key, int n)
{
    counter = 0;
    return Binary_Search(array, 0, n, key);
}

int Binary_Search(int *array, int l, int r, int key)
{
    if (r >= l)
    {
        counter++;

        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        // itself
        if (array[mid] == key)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (array[mid] > key)
            return Binary_Search(array, l, mid - 1, key);

        // Else the element can only be present
        // in right subarray
        return Binary_Search(array, mid + 1, r, key);
    }

    // We reach here when element is not
    // present in array
    return -1;
}
```

```c
/*--------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
--------------------------------------------*/

extern unsigned long int counter;

int Interpolation_Search(int array[], int key, int n)
{
    counter = 0;
    int left = 0;
    int right = (n - 1);

    // Since array is sorted, an element present
    // in array must be in range defined by corner
    while (left <= right && key >= array[left] && key <= array[right])
    {
        if (left == right) {
            if (array[left] == key) return left;
            return -1;
        }
        // Probing the position with keeping
        // uniform distribution in mind.
        int next = left + (((double)(right - left) /
            (array[right] - array[left]))*(key - array[left]));

        // Condition of target found
        if (array[next] == key)
            return next;

        // If x is larger, x is in upper part
        if (array[next] < key)
            left = next + 1;

        // If x is smaller, x is in the lower part
        else
            right = next - 1;
        counter++;
    }
    return -1;
}

```

```c
/*--------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
--------------------------------------------*/

extern unsigned long int counter;

int BIS(int *array, int key, int n)
{
    int left = 1; //pointing the first element of sublist
    int right = n; //pointing the last element of sublist
    int size = right - left + 1; //size of sublist
    int next = (size * ((key - array[left]) / (array[right] - array[left]))) + 1; //prediction of position
    int i = 0;
    int temp = 0;
    counter = 0;

    // Handling Exception (key greater than max)
    if (key > array[n - 1])
        return -1;

    while (key != array[next])
    {
        i = 0;
        size = right - left + 1;

        // if the sublist is too small it performs linear search
        if (size <= 3)
        {
            while (i < right)
            {
                if (array[left + i] == key)
                    return left + i;
                else
                    i++;
                counter++;
            }
        }

        // narrow the sublist
        if (key >= array[next])
        {
            temp = next + (i * sqrt(size)) - 1;
            if (temp >= n)
                temp = n - 1;
            while (key > array[temp])
            {
                i++;
                temp = next + (i * sqrt(size)) - 1;
                if (temp >= n)
                    temp = n - 1;
                counter++;
            }
            right = next + (i * sqrt(size));
            left = next + ((i - 1) * sqrt(size));
        }

        else if (key < array[next])
        {
            temp = next - (i * sqrt(size)) + 1;
            if (temp < 0)
                temp = 0;
            while (key < array[temp])
            {
                i++;
                temp = next - (i * sqrt(size)) + 1;
                if (temp < 0)
                    temp = 0;
                counter++;
            }
            right = next - ((i - 1) * sqrt(size));
            left = next - (i * sqrt(size));
        }

        // Handling Exception (exceeding array bounds)
        if (left < 0)
            left = 0;
        if (right >= n)
            right = n;
        if (array[left] == key)
            return left;
        if (array[right] == key)
            return right;

        next = left + ((right - left + 1) * ((key - array[left]) / (array[right] - array[left]))) - 1;
    }

    if (key == array[next])
        return next;
    else
        return -1;
}
```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern unsigned long int counter;

int Imp_BIS(int *array, int key, int n)
{
    int left = 1; //pointing the first element of sublist
    int right = n; //pointing the last element of sublist
    int size = right - left + 1; //size of sublist
    int next = (size * ((key - array[left]) / (array[right] - array[left]))) + 1; //prediction of position
    int i,j,k = 0;
    int temp = 0;
    int mid, l, r = 0;
    counter = 0;

    // Handling Exception (key greater than max)
    if (key > array[n - 1])
        return -1;

    while (key != array[next])
    {
        i = 0;
        size = right - left + 1;

        // if the sublist is too small it performs linear search
        if (size <= 3)
        {
            while (i < right)
            {
                if (array[left + i] == key)
                    return left + i;
                else
                    i++;
                counter++;
            }
        }

        // narrow the sublist
        if (key >= array[next])
        {
            temp = next + ((2 ^ i) * sqrt(size)) - 1;
            if (temp >= n)
                temp = n - 1;
            while (key > array[temp])
            {
                i ++;
                temp = next + ((2 ^ i) * sqrt(size)) - 1;
                if (temp >= n)
                    temp = n - 1;
                counter++;
            }
            right = next + ((2 ^ i) * sqrt(size));
            left = next + ((2 ^ (i-1)) * sqrt(size));
        }

        else if (key < array[next])
        {
            temp = next - ((2 ^ i) * sqrt(size)) + 1;
            if (temp < 0)
                temp = 0;
            while (key < array[temp])
            {
                i ++;
                temp = next - ((2 ^ i)* sqrt(size)) + 1;
                if (temp < 0)
                    temp = 0;
                counter++;
            }
            right = next - ((2 ^ (i - 1)) * sqrt(size));
            left = next - ((2 ^ i) * sqrt(size));
        }

        // Binary Search
        j = 0;
        l = 2 ^ (i - 1);
        r = 2 ^ i;
        for (j = l; j <= r; j++)
        {
            mid = (l + r) / 2;
            k = next + (mid * sqrt(size));
            if (key == array[k])
                return k;
            if (key < array[k])
                r = mid - 1;
            else if (key > array[k])
                l = mid + 1;
            counter++;
        }

        // Handling Exception (exceeding array bounds)
        if (left < 0)
            left = 0;
        if (right >= n)
            right = n;
        if (array[left] == key)
            return left;
        if (array[right] == key)
            return right;

        next = left + ((right - left + 1) * ((key - array[left]) / (array[right] - array[left]))) - 1;
    }

    if (key == array[next])
        return next;
    else
        return -1;
}
```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern int n;

int * txt_to_array(char file_source[])
{
    int *array;
    array = (int*) malloc(n * sizeof(int));
    FILE *file;

    file = fopen(file_source, "r");

    int i = 0;
    while (!feof(file))
    {
        fscanf(file, "%d", &array[i]);
        i++;
    }

    fclose(file);

    return array;
}

```

# ΠΕΡΙΕΧΟΜΕΝΑ ΚΩΔΙΚΑ – Part 2

- ALL (main())
- Student.h

- 1ο Ερώτημα

- Binary_Search_Tree_AM.h
- Read_Students_Info_A.h
- File_Close_A.h
- Menu_A.h

- 2ο Ερώτημα

- Binary_Search_Tree_Grade.h
- Read_Students_Info_B.h
- File_Close_B.h
- Menu_B.h

- 3ο Ερώτημα

- Hash_Table.h
- Read_Students_Info_Hash.h
- File_Close_ Hash.h
- Menu_ Hash.h

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "Student.h"
#include "Binary_Search_Tree_AM.h"
#include "Binary_Search_Tree_Grade.h"
#include "Hash_Table.h"
#include "Read_Students_Info_A.h"
#include "Read_Students_Info_B.h"
#include "Read_Students_Info_Hash.h"
#include "File_Close_A.h"
#include "File_Close_B.h"
#include "File_Close_Hash.h"
#include "Menu_A.h"
#include "Menu_B.h"
#include "Menu_Hash.h"

const int w = 5;

void main()
{
    node_A * root_A = NULL;
    node_B * root_B = NULL;
    hash * hash_list = NULL;

    int ch = -1;
    int ans = 0;
    int wrong_selection = 0;

    printf("--------------------------------------------------\n\n");
    printf("Computer Engineering and Informatics Department\n");
    printf("        University of Patras    \n\n");
    printf("        Data Structures \n");
    printf("        Project 2019    \n\n");
    printf("    Glarakis George AM: 1059561 \n");
    printf("    Papadias Epameinondas AM: 1062665 \n");
    printf("    Prokopiou Giannis AM : 1059554 \n\n");
    printf("--------------------------------------------------\n\n");

    do {
        ans = 0;
        wrong_selection = 0;
        printf("\n1. Binary Search Tree (by AM)\n");
        printf("2. Binary Search Tree (by Grade)\n");
        printf("3. Hashing chaining\n");
        printf("\nChoose a function: ");

        scanf("%d", &ch);
        while ((getchar()) != '\n'); //clear the input buffer
        switch (ch)
        {
        case 1:
            printf("\nBinary Search Tree by AM\n");
            root_A = read_file_A(root_A, "Foitites-Vathmologio-DS.txt");
            root_A = menu_A(root_A);
            break;

        case 2:
            printf("\nBinary Search Tree by Grade\n");
            root_B = read_file_B(root_B, "Foitites-Vathmologio-DS.txt");
            root_B = menu_B(root_B);
            break;

        case 3:
            printf("\nHash Table with chains by AM\n");
            hash_list = hash_setup(w);
            hash_list = read_file_C(hash_list, "Foitites-Vathmologio-DS.txt");
            hash_list = menu_C(hash_list);
            break;

        default:
            wrong_selection = 1;
        }

        if (wrong_selection == 1)
        {
            printf("\nPlease choose one of the Menu Numbers!\n");
            ans = 1;
        }
    } while (ans);
}
```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

// Struct Student
typedef struct Student {

    char AM[9];
    char Name[100];
    char Surname[100];
    double Grade;
} st;

// Print the student's info
void print_st(st st0)
{
    printf("%s\t%s\t%s\t\t%.2lf\n", st0.AM, st0.Name, st0.Surname, st0.Grade);
}

// Functions to make a change at student's info
st change_am(st st, char *new_am)
{
    strcpy(st.AM, new_am);
    return st;
}
st  change_name(st st, char *new_name)
{
    strcpy(st.Name, new_name);
    return st;
}
st change_surname(st st, char *new_surname)
{
    strcpy(st.Surname, new_surname);
    return st;
}
st change_grade(st st, double new_grade)
{
    st.Grade = new_grade;
    return st;
}

```

```c
char * read_change()
{
    char ch[100];
    while ((getchar()) != '\n'); // clear the input buffer
    printf("Give the change: ");
    scanf("%s", ch);
    return ch;
}

double read_grade()
{
    double newgrade = -1;
    printf("Give the change: ");
    scanf("%lf", &newgrade);
    return newgrade;
}

st choose_change(st st_change)
{
    int n;
    char ch[100];
    printf("1. Change Name\n");
    printf("2. Change Surame\n");
    printf("3. Change Grade\n");
    printf("\nChoose a change function: ");
    while (1)
    {
        scanf("%d", &n);
        switch (n)
        {
        case 1:
            strcpy(ch, read_change());
            st_change = change_name(st_change, ch);
            break;
        case 2:
            strcpy(ch, read_change());
            st_change = change_surname(st_change, ch);
            break;
        case 3:
            st_change = change_grade(st_change, read_grade());
            break;
        default:
            printf("Please choose one of the above options: ");
        }
        break;
    }
    return st_change;
}
```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

// Struct node for the BTS
typedef struct node_A
{
    st st0;
    struct node_A *left, *right;
} node_A;

// Create a new Node
node_A * newNode_A(st st1)
{
    node_A *temp = (node_A *)malloc(sizeof(node_A));
    temp->st0 = st1;

    temp->left = temp->right = NULL;
    return temp;
}

// Insert a new student to the Binary Search Tree
node_A* insert_A(node_A* node_A, st st2)
{
    // If the tree is empty, return a new node
    if (node_A == NULL) return newNode_A(st2);

    int res = strcmp(st2.AM, node_A->st0.AM);

    // Otherwise, recur down the tree
    if (res < 0)
        node_A->left = insert_A(node_A->left, st2);
    else if (res > 0)
        node_A->right = insert_A(node_A->right, st2);

    return node_A;
}

// Print the BST sorted
void inorder_A(node_A *root)
{
    if (root != NULL)
    {
        inorder_A(root->left);
        print_st(root->st0);
        inorder_A(root->right);
    }
}

node_A* search_A(node_A* root, char am[9])
{
    int res = strcmp(am, root->st0.AM);

    // Base Cases: root is null or key is present at root
    if (res == 0)
    {   return root;    }

    // Key is greater than root's key
    if (res > 0)
    {
        if (root->right == NULL)
        {
            return -1;
        }
        return search_A(root->right, am);
    }

    // Key is smaller than root's key
    else
    {
        if (root->left == NULL)
        {
            return -1;
        }
        return search_A(root->left, am);
    }
}

// Find the minimum value of the BTS
node_A * minValueNode_A(node_A* node_A)
{
    while (node_A->left != NULL)
    {
        node_A = node_A->left;
    }
    return node_A;
}

// Delete a node from the BTS
node_A* deleteNode_A(node_A* root, st st)
{
    if (root == NULL) return root;

    int res = strcmp(st.AM, root->st0.AM);

    // If the key to be deleted is smaller than the root's key,
    // then it lies in left subtree
    if (res < 0)
```

```
        root->left = deleteNode_A(root->left, st);

    // If the key to be deleted is greater than the root's key,
    // then it lies in right subtree
    else if (res > 0)
        root->right = deleteNode_A(root->right, st);

    // if key is same as root's key, then this is the node
    // to be deleted
    else
    {
        // node with only one child or no child
        if (root->left == NULL)
        {
            node_A *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            node_A *temp = root->left;
            free(root);
            return temp;
        }

        // node with two children: Get the inorder successor (smallest
        // in the right subtree)
        node_A* temp = minValueNode_A(root->right);

        // Copy the inorder successor's content to this node
        root->st0 = temp->st0;

        // Delete the inorder successor
        root->right = deleteNode_A(root->right, temp->st0);
    }
    return root;
}

```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

// Read students' info from a file and make a BTS by the AMs
node_A * read_file_A(node_A * root, char students_data[])
{
    st student;

    // open the given file
    FILE const *file;
    file = fopen(students_data, "r");

    if (file == NULL)
    {
        printf("Error in reading Students' Data...");
    }

    int i = 0;
    while (!feof(file))
    {
        // Scan for student's info
        fscanf(file, "%s %s %s %lf\n", student.AM, student.Name,
          student.Surname, &student.Grade);

        // Insert the new student to the BTS
        if (i == 0)
        {
            root = insert_A(root, student);
        }
        else
        {
            insert_A(root, student);
        }
        i++;
    }
    fclose(file);
    return root;
}

```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

// Print the sudents sorted by AM in the file
void write_inorder_A(FILE *file, node_A *root)
{
    if (root != NULL)
    {
        write_inorder_A(file, root->left);
        fprintf(file, "%s %s %s %.2lf\n", root->st0.AM, root->st0.Name, root->st0.Surname, root->st0.Grade);
        write_inorder_A(file, root->right);
    }
}

// Create a new file to save the BTS
void File_Close_A(node_A *root, char *dir)
{
    FILE *file;
    file = fopen(dir, "w+");
    write_inorder_A(file, root);

    fclose(file);
}
```

```c
/*--------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
--------------------------------------------*/

char * AM_read_A()
{
    char am[100];
    printf("Give the student's AM: ");
    gets(am);
    return am;
}

int answer_A()
{
    char ans = NULL;

    do
    {
        ans = getchar();
        while ((getchar()) != '\n'); //clear the input buffer

        if (ans == 'N' || ans == 'n')
        {
            return 0;
        }
        else if (ans == 'Y' || ans == 'y')
        {
            return 1;
        }
        else if (ans != 'N' && ans != 'n' && ans != 'Y' && ans != 'y')
        {
            printf("Press (Y/N): ");
        }

    } while (1);
}

// Check if the AM exists
node_A * search_AM_A(node_A * root)
{
    char am[100];
    node_A *node_A;
    do
    {
        strcpy(am, AM_read_A());
        node_A = search_A(root, am);
        if (node_A == -1)
            printf("\nThis AM does not exist! Try again!\n");
    } while (node_A == -1);

    return node_A;
}

node_A* menu_A(node_A *root)
{
    int n;
    node_A *node_A;
    int wrong_selection = 0;
    int ans = -1;

    do
    {
        wrong_selection = 0;
        printf("\n1. Sort List\n");
        printf("2. Search\n");
        printf("3. Make a Change\n");
        printf("4. Delete\n");
        printf("5. Exit\n");
        printf("\nChoose a function: ");

        scanf("%d", &n);
        while ((getchar()) != '\n'); //clear the input buffer
        switch (n)
        {
        // Print the soerted BTS
        case 1:
            printf("AM\t\tFirst Name\tLast Name\tGrade\n\n");
            inorder_A(root);
            break;

        // Search for a student by the AM
        case 2:
            node_A = search_AM_A(root);
            print_st(node_A->st0);
            break;

        // Make a change to a student's info
        case 3:
            do
            {
                node_A = search_AM_A(root);
                node_A->st0 = choose_change(node_A->st0);
                print_st(node_A->st0);
                printf("Would you like to change something else?(Y/N) ");
                while ((getchar()) != '\n'); //clear the input buffer
            } while (answer_A());
            break;

        // Delete a student from the BTS
        case 4:
            node_A = search_AM_A(root);
            root = deleteNode_A(root, node_A->st0);
            printf("\nDeletion was successful!\n");
            break;

        // Exit the program
        case 5:
            break;

        default:
            wrong_selection = 1;
        }

        if (n == 5)
            break;
        else if (wrong_selection == 1)
        {
            printf("\nPlease choose one of the Menu Numbers!\n");
            ans = 1;
        }
        else
        {
            printf("\nWould you like to call an other function?(Y/N): ");
            ans = answer_A();
        }
    } while (ans);

    // Write in a new file the new changed BTS of students
    File_Close_A(root, "New_Foithtes_A.txt");

    return root;
}

```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

// Struct st_node for the linked list of BTS nodes
typedef struct st_node
{
    st st;
    struct st_node * next;
} st_node;

// Struct node for the BTS
typedef struct node_B
{
    struct node_B *left, *right;
    st_node * head;
    int counter;
} node_B;

// Create a new Node
node_B *newNode_B(st st1)
{
    node_B *temp = (node_B *)malloc(sizeof(node_B));

    temp->counter = 0;
    temp->head = (st_node*)malloc(sizeof(st_node));

    temp->head->st = st1;
    temp->head->next = NULL;
    temp->left = temp->right = NULL;

    return temp;
}

// Create a new st_node (if a student with the same grade already exists)
st_node * new_st_node(st st)
{
    st_node* temp = (st_node*)malloc(sizeof(st_node));
    temp->st = st;
    temp->next = NULL;
}
```

```

// Create a new st_node in an existing Node
node_B *existingNode(node_B *node_B, st st1)
{
    node_B->counter++;
    st_node * current = node_B->head;

    // Find the last student in the linked list
    while (current->next != NULL)
    {
        current = current->next;
    }
    current->next = new_st_node(st1);

    return node_B;
}

void print_list(node_B * root) {

    st_node * current = root->head;

    while (current != NULL) {
        print_st(current->st);
        current = current->next;
    }
}

int double_to_int(double a)
{
    return (int)a * 100;
}

// Insert a new student to the Binary Search Tree
node_B* insert_B(node_B* node_B, st st2)
{
    // If the tree is empty, create a new node
    if (node_B == NULL) return newNode_B(st2);

    int node_grade = double_to_int(node_B->head->st.Grade);
    int st_grade = double_to_int(st2.Grade);
    int res = node_grade - st_grade;

    // If the grade already exists, create a new st_node
    if (res == 0)   return existingNode(node_B, st2);

    // Otherwise, recur down the tree
    if (res > 0)
        node_B->left = insert_B(node_B->left, st2);
    else if (res < 0)
```

```
        node_B->right = insert_B(node_B->right, st2);

    return node_B;
}

// A utility function to do inorder traversal of BST
void inorder_B(node_B *root)
{
    if (root != NULL)
    {
        inorder_B(root->left);
        print_list(root);
        inorder_B(root->right);
    }
}

// Find the minimum value of the BTS
node_B * minValueNode_B(node_B* node_B)
{
    while(node_B->left != NULL)
    {
        node_B = node_B->left;
    }
    return node_B;
}

// Find the maximum value of the BTS
node_B * maxValueNode_B(node_B* node_B)
{
    while(node_B->right != NULL)
    {
        node_B = node_B->right;
    }
    return node_B;
}

```

```c
/*--------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

// Read students' info from a file and make a BTS by the Grade
node_B * read_file_B(node_B * root, char students_data[])
{
    st student;

    // open the given file
    FILE *file;
    file = fopen(students_data, "r");

    if (file == NULL)
    {
        printf("Error in reading Student Data...");
    }

    int i = 0;
    while (!feof(file))
    {
        // Scan for student's info
        fscanf(file, "%s %s %s %lf\n", student.AM, student.Name, student.Surname, &student.Grade);

        // Insert the new student to the BTS
        if (i == 0)
        {   root = insert_B(root, student); }
        else
        {   insert_B(root, student);    }
        i++;
    }

    fclose(file);
    return root;
}

```

```c
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

// Print the sudents sorted by grade in the file
void write_inorder_B(FILE *file, node_B *root)
{
    if (root != NULL)
    {
        st_node * current = root->head;
        write_inorder_B(file, root->left);
        while (current != NULL) {
            fprintf(file, "%s %s %s %.2lf\n", current->st.AM, current->st.Name, current->st.Surname, current->st.Grade);
            current = current->next;
        }
        write_inorder_B(file, root->right);
    }
}

// Create a new file to save the BTS
void File_Close_B(node_B *root, char *dir)
{
    FILE *file;
    file = fopen(dir, "w+");
    write_inorder_B(file, root);

    fclose(file);
}
```

```c
/*--------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

int answer_B()
{
    char ans = NULL;
    do
    {
        ans = getchar();
        while ((getchar()) != '\n'); //clear the input buffer

        if (ans != 'N' && ans != 'n' && ans != 'Y' && ans != 'y')
        {
            printf("Press (Y/N): ");
        }
        else if (ans == 'N' || ans == 'n')
        {
            return 0;
        }
        else if (ans == 'Y' || ans == 'y')
        {
            return 1;
        }

    } while (1);
}

node_B* menu_B(node_B *root)
{
    int n;
    char AM[9];
    node_B *node_B;
    int ans = -1;
    int wrong_selection = 0;

    do
    {
        wrong_selection = 0;
        printf("\n1. Minimun Grade\n");
        printf("2. Maximun Grade\n");
        printf("3. Exit\n");
        printf("\nChoose a function: ");

        scanf("%d", &n);
        while ((getchar()) != '\n'); //clear the input buffer
        switch (n)
        {
        // Print a list of the students with the minimum grade
        case 1:
            print_list(minValueNode_B(root));
            break;

        // Print a list of the students with the maximum grade
        case 2:
            print_list(maxValueNode_B(root));
            break;

        // Exit the program
        case 3:
            break;

        default:
            printf("Please choose one of the Menu Numbers!");
            wrong_selection = 1;
            ans = 1;
        }

        if (n == 3)
            break;
        else if (wrong_selection == 0)
        {
            printf("\nWould you like to call an other function?(Y/N): ");
            ans = answer_B();
        }
    } while (ans);

    // Write in a new file the new changed BTS of students
    File_Close_B(root, "New_Foithtes_B.txt");

    return root;
}
```

```
/*---------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

extern const int w;

// Struct node for the linked lists
typedef struct node_C {
    st student;
    struct node_C *next_node;
} node_C;

// Struct hash for the Hash Table
typedef struct hash {
    int value;
    node_C *node_C;
} hash;

// Initialize the Hash Table as NULL
hash* hash_setup()
{
    hash* hash_list = (hash*)calloc(w, sizeof(hash));

    for (int i = 0; i < w; i++)
    {
        hash_list[i].value = i;
        hash_list[i].node_C = NULL;
    }

    return hash_list;
}

// Create a new node (head of the linked list)
node_C* newHashNode(st st)
{
    node_C *temp = (node_C *)malloc(sizeof(node_C));
    temp->student = st;
    temp->next_node = NULL;

    return temp;
}
```

```

// Find the hash of student
int pos_finder(char *AM)
{
    int counter = 0;
    int sum = 0;
    for (int i = 0; i < strlen(AM); i++)
    {
        counter = AM[i];
        sum += counter;
    }
    sum = sum % w;
    return sum ;
}

// Insert the new student to the linked list
node_C * insertNode(node_C* node_C, st st)
{
    while (node_C != NULL)
    {
        node_C->next_node = insertNode(node_C->next_node, st);
        break;
    }
    if (node_C == NULL)
    {   node_C = newHashNode(st);   }
    return node_C;
}

// Insert the new student to the Hash Table
hash* insert_Hash(hash* hash_list, st st)
{
    int pos = pos_finder(st.AM, w);
    hash_list[pos].node_C = insertNode(hash_list[pos].node_C, st);

    return hash_list;
}

// Search a student by AM in tha linked list
node_C * search_next(char AM[9], node_C* node_C)
{
    int res = strcmp(AM, node_C->student.AM);
    if (res == 0)
        return node_C;
    else
        if (node_C->next_node == NULL)
        {   return -1;  }
        search_next(AM, node_C->next_node);
}

```

```c
// Search a student by AM in tha Hash Table
node_C * search_st(hash* hash_list, char AM[9])
{
    int i = pos_finder(AM);
    int res = strcmp(AM, hash_list[i].node_C->student.AM);
    if (res == 0)
        return hash_list[i].node_C;
    else
        return search_next(AM, hash_list[i].node_C->next_node);
}

// Delete a student from the middle or the end of the linked list
node_C * del_mid_node(node_C * prev, node_C * current, node_C * del_node)
{
    while (current != del_node)
    {
        current = del_mid_node(current, current->next_node, del_node);
        break;
    }
    if (current == del_node)
    {   prev->next_node = current->next_node;   }

    return prev;
}

// Delete a student from the Hash Table
hash * delete_node(hash * hash_list, node_C * del_node)
{
    int pos = pos_finder(del_node->student.AM);

    // Delete the student if it's the fist in the linked list
    if (del_node == hash_list[pos].node_C)
    {
        hash_list[pos].node_C = del_node->next_node;
        free(del_node);
    }

    // Delete if it's in the middle or at the end
    else
    {
        hash_list[pos].node_C = del_mid_node(hash_list[pos].node_C, hash_list[pos].node_C->next_node, del_node);
        free(del_node);
    }

    return hash_list;
}

void print_table(hash* hash_list)
```

```
{
    node_C* current = NULL;
    for (int i = 0; i < w; i++)
    {
        current = hash_list[i].node_C;
        while (current != NULL)
        {
            print_st(current->student);
            current = current->next_node;
        }
    }
}
```

```c
/*--------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
--------------------------------------------*/

// Read students' info from a file and make a Hash Table by the AMs
hash * read_file_C(hash * hash_list, char students_data[])
{
    st student;

    // open the given file
    FILE *file;
    file = fopen(students_data, "r");

    if (file == NULL)
    {
        printf("Error in reading Student Data...");
    }

    int i = 0;
    while (!feof(file))
    {
        // Scan for student's info
        fscanf(file, "%s %s %s %lf\n", student.AM, student.Name, student.Surname, &student.Grade);

        // Insert the new student to the Hash Table
        insert_Hash(hash_list, student);
    }

    fclose(file);
    return hash_list;
}

```

```c
/*--------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

// Create a new file to save the Hash Table
void File_Close_C(hash * hash_list, char *dir)
{
    FILE *file;
    file = fopen(dir, "w+");

    node_C* current = NULL;
    for (int i = 0; i < w; i++)
    {
        current = hash_list[i].node_C;
        while (current != NULL)
        {
            fprintf(file, "%s %s %s %.2lf\n", current->student.AM, current->student.Name, current->student.Surname, current->student.Grade);
            current = current->next_node;
        }
    }
    fclose(file);
}
```

```c
/*----------------------------------------------
Computer Engineering and Informatics Department
University of Patras

Data Structures
Project 2019

Glarakis George AM: 1059561
Papadias Epameinondas AM: 1062665
Prokopiou Giannis AM: 1059554
---------------------------------------------*/

char * AM_read_C()
{
    char am[100];
    printf("Give the student's AM: ");
    gets(am);
    return am;
}

int answer_C()
{
    char ans = NULL;

    do
    {
        ans = getchar();
        while ((getchar()) != '\n'); //clear the input buffer

        if (ans == 'N' || ans == 'n')
        {
            return 0;
        }
        else if (ans == 'Y' || ans == 'y')
        {
            return 1;
        }
        else if (ans != 'N' && ans != 'n' && ans != 'Y' && ans != 'y')
        {
            printf("Press (Y/N): ");
        }
    } while (1);
}

// Check if the AM exists
node_C * search_AM_C(hash* hash_list)
{
    char am[100];
    node_C *node_C;
    do
    {
        strcpy(am, AM_read_C());
        node_C = search_st(hash_list, am);
        if (node_C == -1)
            printf("\nThis AM does not exist! Try again!\n");
    } while (node_C == -1);

    return node_C;
}

hash * menu_C(hash * hash_list)
{
    int n;
    char AM[9];
    node_C* node_C;
    int wrong_selection = 0;
    int ans = -1;

    do
    {
        wrong_selection = 0;
        printf("1. Search\n");
        printf("2. Make a Change\n");
        printf("3. Delete\n");
        printf("4. Exit\n");
        printf("\nChoose a function: ");

        scanf("%d", &n);
        while ((getchar()) != '\n'); //clear the input buffer

        switch (n)
        {
        // Search for a student by the AM
        case 1:
            node_C = search_AM_C(hash_list);
            print_st(node_C->student);
            break;

        // Make a change to a student's info
        case 2:
            do
            {
                node_C = search_AM_C(hash_list);
                node_C->student = choose_change(node_C->student);
                print_st(node_C->student);
                printf("Would you like to change something else?(Y/N) ");
                while ((getchar()) != '\n'); //clear the input buffer
            } while (answer_C());
            break;

        // Delete a student from the Hash Table
        case 3:
            node_C = search_AM_C(hash_list);
            hash_list = delete_node(hash_list, node_C);
            printf("\nThe deletion was successful!\n");
            break;

        // Exit the program
        case 4:
            break;

        default:
            wrong_selection = 1;
        }

        if (n == 4)
            break;
        else if (wrong_selection == 1)
        {
            printf("\nPlease choose one of the Menu Numbers!\n");
            ans = 1;
        }
        else
        {
            printf("\nWould you like to call an other function?(Y/N): ");
            ans = answer_C();
        }
    } while (ans);

    File_Close_C(hash_list, "New_Foithtes_C.txt");

    return hash_list;
}
```