ΕΠΑΝΑΛΗΨΗ ΣΕ ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΤΗΣ C (1)

Ε1. Δημιουργήστε μία συνάρτηση που να δέχεται σαν παραμέτρους έναν πίνακα ακεραίων και έναν ακέραιο αριθμό και να επιστρέφει πόσες φορές περιέχεται ο αριθμός στον πίνακα. Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει 10 ακεραίους και να τους αποθηκεύει σε έναν πίνακα ακεραίων. Στη συνέχεια, να διαβάζει έναν ακέραιο αριθμό και να εμφανίζει πόσες φορές περιέχεται στον πίνακα με χρήση της συνάρτησης.

```
#include <stdio.h>

#define SIZE 10

int find(int pin[], int num); // Πρωτότυπο συνάρτησης.

int main()
{
	int i, j, k, arr[SIZE];

	for(i = 0; i < SIZE; i++)
	{
		printf("Enter number: ");
		scanf("%d", &arr[i]);
	}
	printf("Enter number to search: ");
	scanf("%d", &j);

	k = find(arr, j); /* Κλήση της συνάρτησης find(). Το k θα γίνει
ίσο με την τιμή επιστροφής της find(). */
	printf("Times:%d\n", k);
	return 0;
}

int find(int pin[], int num) /* Αντιστοίχιση των παραμέτρων μία-προς-
μία. Δηλαδή, pin=arr και num=j */
{
	int i, cnt;

	cnt = 0;
	for(i = 0; i < SIZE; i++)
	{
		if(pin[i] == num)
		{
			cnt++;
		}
	}
	return cnt;
}
```

Θ2. Να γραφεί ένα πρόγραμμα το οποίο να δηλώνει μία ακέραια μεταβλητή με τιμή 100. Στη συνέχεια, να δηλώνει ένα δείκτη σε αυτή και με χρήση του δείκτη να εμφανίζει τη διεύθυνση της μεταβλητής και το περιεχόμενό της.

```
#include <stdio.h>
int main()
{
	int i = 100;
```

```
      int *ptr; /* Η μεταβλητή ptr δηλώνεται ως δείκτης προς κάποια ακέραια μεταβλητή. */

      ptr = &i; /* Τώρα, ο δείκτης ptr δείχνει στη διεύθυνση μνήμης του i. Λέμε, ότι ο ptr δείχνει στο i. */
      printf("Addr = %d\n", ptr); /* Ισοδύναμο με printf("Addr = %d\n", &i); */
      printf("Periex = %d\n", *ptr); /* Αφού ο ptr δείχνει στο i, το *ptr είναι ισοδύναμο με το i. */
      return 0;
}
```

Ε3. Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main()
{
      int *ptr1, *ptr2, *ptr3;
      int i = 10, j = 20, k = 30;

      ptr1 = &i;
      i = 100;

      ptr2 = &j;
      j = *ptr2 + *ptr1;

      ptr3 = &k;
      k = *ptr3 + *ptr2;
      printf("%d %d %d\n", *ptr1, *ptr2, *ptr3);

      *ptr1 = *ptr2 = *ptr3; /* i=j=k. */
      printf("%d %d %d\n", i, j, k);
      return 0;
}
```

Απάντηση: 100 120 150
150 150 150

Θ4. Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>

void test(int a, int *p);

int main()
{
      int i = 10, j = 20;

      test(i, &j);
      printf("%d %d\n", i, j);
      return 0;
}

void test(int a, int *p) // a=i, p=&j
{
      a = 300;
      *p = 100;
}
```

Σχόλια: Για να αλλάξει η τιμή της μεταβλητής πρέπει να διοχετεύεται στη συνάρτηση η διεύθυνση μνήμης της μεταβλητής (π.χ. &j). Το πρόγραμμα εμφανίζει 10 και 100.

Ε5. Δημιουργήστε μία **void** συνάρτηση που να δέχεται σαν παραμέτρους έναν πίνακα ακεραίων και να επιστρέφει την τιμή του μεγαλύτερου στοιχείου του. Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει 10 ακεραίους, να τους αποθηκεύει σε έναν πίνακα, να καλεί την συνάρτηση και να εμφανίζει την μεγαλύτερη τιμή.

```
#include <stdio.h>

#define SIZE 10

void find(int pin[], int *p);

int main()
{
	int i, j, arr[SIZE];

	for(i = 0; i < SIZE; i++)
	{
		printf("Enter number: ");
		scanf("%d", &arr[i]);
	}
	find(arr, &j);
	printf("Max:%d\n", j);
	return 0;
}

void find(int pin[], int *p) // pin=arr, p=&j
{
	int i, max;

	max = pin[0];
	for(i = 1; i < SIZE; i++)
	{
		if(pin[i] > max)
		{
			max = pin[i];
		}
	}
	*p = max; // j = max;
}
```

Σχόλια: Αφού ο τύπος επιστροφής της συνάρτησης είναι **void**, η συνάρτηση δεν μπορεί να επιστρέψει άμεσα κάποια τιμή. Επομένως, προσθέτουμε μία παράμετρο δείκτη στη συνάρτηση, ώστε μέσω αυτής της παραμέτρου να επιστραφεί η τιμή του μεγαλύτερου στοιχείου του πίνακα.

β. Να τροποποιήσετε το πρόγραμμα, ώστε η συνάρτηση να επιστρέφει και την μικρότερη τιμή.

```
#include <stdio.h>

#define SIZE 10
```

```c
void find(int pin[], int *p1, int *p2);

int main()
{
	int i, j, k, arr[SIZE];

	for(i = 0; i < SIZE; i++)
	{
		printf("Enter number: ");
		scanf("%d", &arr[i]);
	}
	find(arr, &j, &k);
	printf("Max:%d Min:%d\n", j, k);
	return 0;
}

void find(int pin[], int *p1, int *p2)
{
	int i, min, max;

	min = max = pin[0];
	for(i = 1; i < SIZE; i++)
	{
		if(pin[i] > max)
		{
			max = pin[i];
		}
		if(pin[i] < min)
		{
			min = pin[i];
		}
	}
	*p1 = max;
	*p2 = min;
}
```

Σχόλια: Όπως πριν, προσθέτουμε δύο παραμέτρους δείκτες στη συνάρτηση, ώστε μέσω αυτών των παραμέτρων να επιστραφούν οι αντίστοιχες τιμές.

Ε6. Ποια είναι η έξοδος του παρακάτω προγράμματος;

```c
#include <stdio.h>
int main()
{
	int *ptr1, *ptr2, i = 20, j = 30;

	ptr1 = &i;
	ptr2 = &j;
	ptr2 = ptr1;
	*ptr1 = *ptr1 + 30;
	*ptr2 = *ptr2 + 50;

	printf("Val = %d\n", *ptr1 + *ptr2);
	return 0;
}
```

Απάντηση: Basket case από Green Day.

ΑΛΓΟΡΙΘΜΟΙ ΔΙΑΧΕΙΡΙΣΗΣ ΠΙΝΑΚΑ (2)

1. Αναζήτηση Στοιχείου.

Η γραμμική αναζήτηση (linear search) είναι ο πιο απλός τρόπος αναζήτησης μίας τιμής μέσα σε έναν μη ταξινομημένο πίνακα. Η αναζήτηση γίνεται σειριακά, ξεκινώντας από το πρώτο στοιχείο του πίνακα μέχρι το τελευταίο. Σε έναν πίνακα `n` στοιχείων, το μέγιστο πλήθος αναζητήσεων μίας τιμής είναι `n` και συμβαίνει όταν η προς αναζήτηση τιμή είναι είτε το τελευταίο στοιχείο είτε δεν υπάρχει στον πίνακα.

Θ1. Δημιουργήστε μία συνάρτηση που να δέχεται σαν παραμέτρους ένα αλφαριθμητικό και έναν χαρακτήρα και να επιστρέφει τον αριθμό των εμφανίσεων του χαρακτήρα στο αλφαριθμητικό. Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει συνεχώς ένα αλφαριθμητικό (μέχρι 100 χαρακτήρες) και έναν χαρακτήρα, να καλεί τη συνάρτηση και να εμφανίζει την τιμή επιστροφής της. Αν ο χρήστης εισάγει το `end`, η εισαγωγή των αλφαριθμητικών να τερματίζει.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int find(char str[], char ch);

int main(void)
{
	char ch, str[100];
	int cnt;

	while(1) /* Ατέρμονος βρόχος. */
	{
		printf("Enter text: ");
		gets(str);  /* Προσοχή, μη ασφαλής συνάρτηση. Για ασφάλεια, χρησιμοποιούμε την fgets(). */
		if(strcmp(str, "end") == 0)
			break; /* Τερματισμός του βρόχου. */

		printf("Enter character to search: ");
		scanf("%c", &ch);

		cnt = find(str, ch);
		printf("Times: %d\n", cnt);
		getchar(); /* Για να απομακρύνουμε τον χαρακτήρα αλλαγής γραμμής '\n' που έχει παραμείνει στο ρεύμα εισόδου. */
	}
	return 0;
}

int find(char str[], char ch)
{
	int i, cnt, len;

	cnt = 0;
	len = strlen(str);
	for(i = 0; i < len; i++)
	{
		if(str[i] == ch)
		{
```

```
                cnt++;
            }
        }
        return cnt;
}
```

E2. Δημιουργήστε μία **void** συνάρτηση που να δέχεται σαν παραμέτρους δύο πίνακες ακεραίων και να ελέγχει αν έχουν κοινά στοιχεία. Αν ναι, η συνάρτηση να επιστρέφει την τιμή 1, αλλιώς την τιμή 0. Να γραφεί ένα πρόγραμμα το οποίο να δηλώνει δύο πίνακες 10 ακεραίων και να διαβάζει τις τιμές των στοιχείων τους. Στη συνέχεια, να καλεί την συνάρτηση και, ανάλογα με την τιμή επιστροφής, να εμφανίζει αντίστοιχο μήνυμα.

```
#include <stdio.h>

#define SIZE 10

void common(int arr1[], int arr2[], int* found);

int main(void)
{
        int i, j, k, arr1[SIZE], arr2[SIZE];

        for(i = 0; i < SIZE; i++)
        {
                printf("Enter number_%d for the 1st array: ", i+1);
                scanf("%d", &arr1[i]);

                printf("Enter number_%d for the 2nd array: ", i+1);
                scanf("%d", &arr2[i]);
        }
        common(arr1, arr2, &k);
        if(k == 0)
                printf("No common elements were found\n");
        else
                printf("The arrays have one common element at least\n");

        return 0;
}

void common(int arr1[], int arr2[], int* found)
{
        int i, j;

        *found = 0;
        for(i = 0; i < SIZE; i++)
        {
                for(j = 0; j < SIZE; j++) /* Αυτός ο βρόχος ελέγχει αν το
στοιχείο του πρώτου πίνακα υπάρχει στον δεύτερο. */
                {
                        if(arr1[i] == arr2[j])
                        {
                                *found = 1;
                                return; /* Αφού βρήκαμε ένα κοινό στοιχείο,
δεν χρειάζεται να συνεχίσουμε τις επαναλήψεις. Η εκτέλεση της
συνάρτησης τερματίζεται άμεσα με την εντολή return. */
                        }
                }
        }
}
```

2. Ταξινόμηση Πίνακα

Για την ταξινόμηση ενός πίνακα υπάρχουν πολλοί αλγόριθμοι. Ας περιγράψουμε τον αλγόριθμο επιλογής *(selection sort)*.

Έστω ότι θέλουμε να ταξινομήσουμε τα στοιχεία ενός πίνακα κατά αύξουσα σειρά. Αρχικά, βρίσκουμε το στοιχείο του πίνακα με τη μικρότερη τιμή και αντιμεταθέτουμε την τιμή του με την τιμή του πρώτου στοιχείου του πίνακα. Άρα, η ελάχιστη τιμή του πίνακα αποθηκεύεται στην πρώτη θέση του.

Στη συνέχεια, βρίσκουμε τη νέα ελάχιστη τιμή μεταξύ των υπολοίπων στοιχείων του πίνακα, δηλαδή χωρίς το πρώτο στοιχείο. Όπως προηγουμένως, αντιμεταθέτουμε αυτή την τιμή με την τιμή του δεύτερου στοιχείου. Άρα, η δεύτερη μικρότερη τιμή του πίνακα αποθηκεύεται στη δεύτερη θέση του. Η διαδικασία αυτή επαναλαμβάνεται για τα υπόλοιπα στοιχεία του πίνακα. Ο αλγόριθμος τερματίζεται όταν γίνει η σύγκριση και των δύο τελευταίων στοιχείων του.

Ο αλγόριθμος λειτουργεί με παρόμοιο τρόπο για την ταξινόμηση ενός πίνακα κατά φθίνουσα σειρά. Η μοναδική διαφορά είναι ότι, αντί να ψάχνουμε να βρούμε κάθε φορά την ελάχιστη τιμή, τώρα βρίσκουμε τη μέγιστη τιμή. Στο επόμενο πρόγραμμα, η συνάρτηση `sel_sort()` υλοποιεί τον αλγόριθμο της επιλογής για την ταξινόμηση των στοιχείων ενός πίνακα κατά αύξουσα σειρά.

Θ3. Δημιουργήστε μία συνάρτηση που να δέχεται σαν παράμετρο έναν πίνακα πραγματικών και να τον ταξινομεί κατά αύξουσα σειρά σύμφωνα με τον αλγόριθμο της ταξινόμησης με επιλογή. Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τους βαθμούς 10 φοιτητών, να τους αποθηκεύει σε έναν πίνακα, να καλεί τη συνάρτηση και να εμφανίζει τον ταξινομημένο πίνακα.

```c
#include <stdio.h>

#define STUD 10

void sel_sort(double arr[]);

int main(void)
{
	int i;
	double grd[STUD];

	for(i = 0; i < STUD; i++)
	{
		printf("Enter grade of stud_%d: ", i+1);
		scanf("%lf", &grd[i]);
	}
	sel_sort(grd);

	printf("\n***** Sorted array *****\n");
	for(i = 0; i < STUD; i++)
		printf("%.2f\n", grd[i]);
	return 0;
}

void sel_sort(double arr[])
{
	int i, j;
```

```
        double tmp;

        for(i = 0; i < STUD-1; i++)
        {
                for(j = i+1; j < STUD; j++)
                {
                        if(arr[i] > arr[j])
                        {
                                /* Αντιμετάθεση τιμών. */
                                tmp = arr[i];
                                arr[i] = arr[j];
                                arr[j] = tmp;
                        }
                }
        }
}
```

Σχόλια: Σε κάθε επανάληψη του εσωτερικού βρόχου, το `arr[i]` συγκρίνεται με τα στοιχεία του πίνακα από τη θέση `i+1` μέχρι και τη θέση `STUD-1`. Αν κάποιο στοιχείο έχει μικρότερη τιμή, γίνεται αντιμετάθεση των τιμών τους. Για παράδειγμα, στην πρώτη επανάληψη του εξωτερικού βρόχου `(i = 0)`, η ελάχιστη τιμή των στοιχείων από τη θέση `1` μέχρι τη θέση `STUD-1` αποθηκεύεται στο στοιχείο `arr[0]`. Για να ταξινομήσουμε τον πίνακα κατά φθίνουσα σειρά, απλά αλλάζουμε την **`if`** συνθήκη σε: **`if`**`(arr[i] < arr[j])`

β. Σαν συνέχεια της προηγούμενης άσκησης, το πρόγραμμα να διαβάζει τα ονόματα των 10 φοιτητών (μέχρι 100 χαρακτήρες το καθένα) και να εμφανίζει την ονομαστική κατάσταση βαθμολογίας σε αύξουσα σειρά ως προς τον βαθμό.

```
#include <stdio.h> /* Ασκηση Κ.12.6. */
#include <stdlib.h>
#include <string.h>

#define STUD 10
#define SIZE 100

void sel_sort(char str[][SIZE], double arr[]);

int main(void)
{
        char name[STUD][SIZE]; /* Πίνακας STUD γραμμών, όπου σε κάθε γραμμή αποθηκεύεται ένα όνομα μέχρι SIZE χαρακτήρες. Κάθε name[i] μπορεί να χρησιμοποιηθεί σαν δείκτης προς την αντίστοιχη γραμμή. */
        int i;
        double grd[STUD];

        for(i = 0; i < STUD; i++)
        {
                printf("Enter name: ");
                fgets(name[i], sizeof(name[i]), stdin); /* Ασφαλές διάβασμα. */
                printf("Enter grade: ");
                scanf("%lf", &grd[i]);
                getchar();
        }
        sel_sort(name, grd);

        printf("\n***** Grades in increase order *****\n");
        for(i = 0; i < STUD; i++)
```

```
            printf("%s\t%.2f\n", name[i], grd[i]);
      return 0;
}

void sel_sort(char str[][SIZE], double arr[])
{
      char tmp[100];
      int i, j;
      double k;

      for(i = 0; i < STUD-1; i++)
      {
            for(j = i+1; j < STUD; j++)
            {
                  if(arr[i] > arr[j])
                  {
                        /* Παράλληλη αντιμετάθεση βαθμών και των
αντίστοιχων ονομάτων. */
                        k = arr[i];
                        arr[i] = arr[j];
                        arr[j] = k;

                        strcpy(tmp, str[j]);
                        strcpy(str[j], str[i]);
                        strcpy(str[i], tmp);
                  }
            }
      }
}
```

Ε4. Να συμπληρώσετε το παρακάτω πρόγραμμα χρησιμοποιώντας τους δείκτες `p1` και `p2` για να διαβάσει δύο πραγματικούς αριθμούς και να εμφανίσει την απόλυτη τιμή του αθροίσματός τους. Οι μεταβλητές `i` και `j` να χρησιμοποιηθούν μόνο μία φορά.

```
#include <stdio.h>
int main(void)
{
      float *p1, *p2, i, j; /* Δεν επιτρέπεται να δηλώσετε άλλες
μεταβλητές. */
      ...
}
```

Απάντηση:

```
#include <stdio.h>
int main(void)
{
      float *p1, *p2, i, j;

      p1 = &i;
      p2 = &j;

      printf("Enter numbers: ");
      scanf("%f%f", p1, p2);

      if(*p1+*p2 < 0)
            printf("%f\n", -(*p1+*p2));
      else
```

```
            printf("%f\n", *p1+*p2);
        return 0;
}
```

ΔΟΜΕΣ ΚΑΙ ΔΥΝΑΜΙΚΗ ΔΙΑΧΕΙΡΙΣΗ ΜΝΗΜΗΣ (3)

Ε1. Δημιουργήστε μία συνάρτηση που να δέχεται σαν παραμέτρους δύο δείκτες σε ακέραιες μεταβλητές και να επιστρέφει τον δείκτη στη μεταβλητή με τη μεγαλύτερη τιμή. Αν οι τιμές είναι ίδιες, η συνάρτηση να επιστρέφει την τιμή NULL. Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει δύο ακεραίους, να καλεί την συνάρτηση και να χρησιμοποιεί την τιμή επιστροφής για να εμφανίσει τον μεγαλύτερο από αυτούς ή μήνυμα ότι είναι ίσοι.

```
#include <stdio.h>

int* max(int *p1, int *p2);

int main()
{
	int *ptr;
	int i, j;

	printf("Enter numbers: ");
	scanf("%d%d", &i, &j);

	ptr = max(&i, &j);
	if(ptr == NULL)
		printf("Equal numbers\n");
	else
		printf("Max = %d\n", *ptr);

	return 0;
}

int* max(int *p1, int *p2) /* Όταν γίνεται η κλήση της max() έχουμε p1 = &i και p2 = &j. */
{
	if(*p1 > *p2) /* Αφού το p1 δείχνει στο i το *p1 είναι ίσο με i. Παρόμοια, αφού το p2 δείχνει στο j το *p2 είναι ίσο με j. Ουσιαστικά ελέγχουμε αν i > j. */
		return p1;
	else if(*p1 < *p2)
		return p2;
	else
		return NULL;
}
```

Θ2: Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τον αριθμό των φοιτητών ενός τμήματος, να δεσμεύει δυναμικά μνήμη για να αποθηκεύσει τους βαθμούς τους, να τους διαβάζει και να τους αποθηκεύει σε αυτή την μνήμη. Στη συνέχεια, να διαβάζει έναν βαθμό και να εμφανίζει πόσοι φοιτητές πήραν μεγαλύτερο βαθμό από αυτόν.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	float *arr, grd;
	int i, cnt, num;

	cnt = 0;
	printf("Enter number of students: ");
	scanf("%d", &num);
```

```
	arr = (float*) malloc(num * sizeof(float)); /* Δυναμική δέσμευση μνήμης για num πραγματικούς αριθμούς. */
	if(arr == NULL)
	{
		printf("Error: No memory");
		return 0;
	}
	for(i = 0; i < num; i++)
	{
		printf("Enter grade: ");
		scanf("%f", &arr[i]);
	}
	printf("Enter base: ");
	scanf("%f", &grd);

	for(i = 0; i < num; i++)
		if(arr[i] > grd)
			cnt++;

	printf("%d students got more than %.2f\n", cnt, grd);
	free(arr); /* Απελευθέρωση μνήμης. */
	return 0;
}
```

Σχόλια: Όπως βλέπετε, μπορούμε να χειριστούμε τον δείκτη που επιστρέφει η συνάρτηση `malloc()` σαν πίνακα.

Θ3. Να ορίσετε μία δομή `Student` με πεδία: όνομα φοιτητή, αριθμός μητρώου και βαθμός. Να γραφεί ένα πρόγραμμα που να δηλώνει μία τέτοια δομή, να διαβάζει τα στοιχεία ενός φοιτητή, να τα αποθηκεύει σε αυτήν και να τα εμφανίζει στην οθόνη.

```
#include <stdio.h>

struct Student
{
	char name[50];
	int code;
	float grd;
};

int main()
{
	struct Student s; /* Ο τύπος της μεταβλητής s είναι δομή τύπου Student. Σε αυτό το παράδειγμα, για την πρόσβαση στα πεδία της δομής s χρησιμοποιούμε την μεταβλητή και τον τελεστή τελεία (.) */

	printf("Name: ");
	gets(s.name); /* Μη ασφαλής συνάρτηση. */

	printf("Code: ");
	scanf("%d", &s.code);

	printf("Grade: ");
	scanf("%f", &s.grd);

	printf("N:%s C:%d G:%f\n", s.name, s.code, s.grd);
	return 0;
}
```

β. Να τροποποιήσετε το πρόγραμμα, ώστε το διάβασμα των στοιχείων του φοιτητή να γίνεται με χρήση δείκτη.

```
#include <stdio.h>

struct Student
{
	char name[50];
	int code;
	float grd;
};

int main()
{
	struct Student s;
	struct Student *ptr; /* Σε αυτό το παράδειγμα, για την πρόσβαση στα πεδία της δομής s χρησιμοποιούμε δείκτη στη δομή s και τον τελεστή -> */

	ptr = &s; /* Υπενθυμίζεται ότι πριν χρησιμοποιήσουμε ένα δείκτη πρέπει να τον έχουμε αρχικοποιήσει. */

	printf("Name: ");
	gets(ptr->name);

	printf("Code: ");
	scanf("%d", &ptr->code);

	printf("Grade: ");
	scanf("%f", &ptr->grd);

	printf("N:%s C:%d G:%f\n", ptr->name, ptr->code, ptr->grd);
	return 0;
}
```

Θ4. Να ορίσετε μία δομή `Student` με πεδία: ονοματεπώνυμο φοιτητή, αριθμός μητρώου και βαθμός. Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τα στοιχεία 100 φοιτητών, τα οποία να αποθηκεύονται σε έναν πίνακα 100 τέτοιων δομών. Στη συνέχεια, το πρόγραμμα να διαβάζει έναν αριθμό και να εμφανίζει τα στοιχεία των φοιτητών που έχουν μεγαλύτερο βαθμό από αυτόν.

```
#include <stdio.h>

#define SIZE 100

struct Student
{
	char name[50];
	int code;
	float grd;
};

int main()
{
	int i;
	float grd;
	struct Student stud[SIZE];
```

```
	for(i = 0; i < SIZE; i++)
	{
		printf("\nName: ");
		gets(stud[i].name);

		printf("Code: ");
		scanf("%d", &stud[i].code);

		printf("Grade: ");
		scanf("%f", &stud[i].grd);

		getchar();
	}
	printf("\nBase: ");
	scanf("%f", &grd);
	for(i = 0; i < SIZE; i++)
	{
		if(stud[i].grd > grd)
			printf("N:%s	C:%d	G:%f\n",	stud[i].name,
stud[i].code, stud[i].grd);
	}
	return 0;
}
```

Σχόλια: Ένας πίνακας δομών είναι ένας πίνακας όπου το κάθε στοιχείο του είναι δομή.

Ε5. Να τροποποιήσετε το προηγούμενο πρόγραμμα, ώστε ο χρήστης να εισάγει το πλήθος των φοιτητών (π.χ. `num`) και το πρόγραμμα να δεσμεύει μνήμη για να αποθηκεύσει `num` δομές τύπου `Student`.

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

struct Student
{
	char name[50];
	int code;
	float grd;
};

int main()
{
	int i, num;
	float grd;
	struct Student *stud; /* Δήλωση του δείκτη. */

	printf("Enter number: ");
	scanf("%d", &num);

	stud = (struct Student*)malloc(num * sizeof(struct Student));
/* Δέσμευση μνήμης για  num δομές τύπου Student. */
	if(stud == NULL)
	{
		printf("Error: No memory");
		return 0;
	}
	getchar();
```

```
	for(i = 0; i < num; i++)
	{
		printf("\nName: ");
		gets(stud[i].name);

		printf("Code: ");
		scanf("%d", &stud[i].code);

		printf("Grade: ");
		scanf("%f", &stud[i].grd);

		getchar();
	}
	printf("\nBase: ");
	scanf("%f", &grd);
	for(i = 0; i < num; i++)
	{
		if(stud[i].grd > grd)
			printf("N:%s	C:%d	G:%f\n",	stud[i].name,
stud[i].code, stud[i].grd);
	}
	free(stud);
	return 0;
}
```

Σχόλια: Βλέπετε πάλι ότι τον δείκτη που επιστρέφει η `malloc()` τον χειριζόμαστε σαν πίνακα.

β. Να εμφανίσετε τα στοιχεία των φοιτητών με χρήση του δείκτη `ptr`.

```
int main()
{
	struct Student *ptr;

	... /* Ίδιος κώδικας. */

	for(i = 0; i < num; i++)
	{
		ptr = &stud[i];
		if(ptr->grd > grd)
			printf("N:%s  C:%d  G:%f\n",  ptr->name,  ptr->code,
ptr->grd);
	}
	return 0;
}
```

γ. Να αντικαταστήσετε το τελευταίο `for` με μία συνάρτηση, η οποία να εμφανίζει τα στοιχεία των φοιτητών που έχουν μεγαλύτερο βαθμό.

```
void show(struct Student st[], int num, float grd)
{
	int i;

	for(i = 0; i < num; i++)
	{
		if(st[i].grd > grd)
			printf("N:%s  C:%d  G:%f\n",  st[i].name,  st[i].code,
st[i].grd);
	}
```

```
}
```

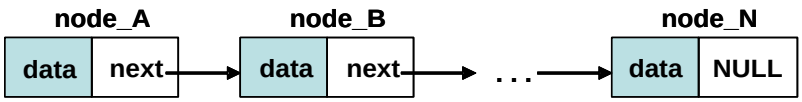
Για να την καλέσουμε γράφουμε: `show(stud, num, grd);`

ΣΤΟΙΒΑ (4)

Ε1. Να ορίσετε μία δομή `Point` με πεδία τις συντεταγμένες ενός σημείου. Δημιουργήστε μία συνάρτηση που να δέχεται σαν παραμέτρους δύο δείκτες σε δομές τύπου `Point` και να αντιμεταθέτει τα πεδία τους με χρήση του τελεστή `->`. Να γραφεί ένα πρόγραμμα που να δηλώνει δύο τέτοιες δομές, να θέτει τις τιμές 1 και 2 στα πεδία της πρώτης δομής και τις τιμές 3 και 4 στα πεδία της δεύτερης δομής, να καλεί την συνάρτηση και να εμφανίζει τις τιμές των πεδίων τους.

```
#include <stdio.h>

struct Point
{
	float x;
	float y;
};

void swap(struct Point *ptr1, struct Point *ptr2);

int main()
{
	struct Point p1, p2;

	p1.x = 1;
	p1.y = 2;
	p2.x = 3;
	p2.y = 4;

	swap(&p1, &p2);

	printf("X:%f Y:%f\n", p1.x, p1.y);
	printf("X:%f Y:%f\n", p2.x, p2.y);
	return 0;
}

void swap(struct Point *ptr1, struct Point *ptr2)
{
	float tmp;

	tmp = ptr1->x;
	ptr1->x = ptr2->x;
	ptr2->x = tmp;

	tmp = ptr1->y;
	ptr1->y = ptr2->y;
	ptr2->y = tmp;
}
```

Θ2. Μία δυναμική δομή δεδομένων, δηλαδή μία δομή της οποίας το μέγεθος μπορεί να αυξομειώνεται κατά την εκτέλεση του προγράμματος, αποτελείται από ένα ή περισσότερα συνδεδεμένα στοιχεία, που συνήθως ονομάζονται κόμβοι. Κάθε κόμβος αντιπροσωπεύεται από μία δομή, η οποία περιέχει τα δεδομένα του κόμβου και ένα πεδίο που είναι δείκτης (π.χ. το πεδίο `next`) στον επόμενο κόμβο. Το επόμενο σχήμα παρουσιάζει την πιο συνηθισμένη δυναμική δομή δεδομένων, που είναι μία απλά συνδεδεμένη λίστα.



Ο πρώτος κόμβος ονομάζεται κεφαλή (head) της λίστας και ο τελευταίος ουρά (tail). Η τιμή του πεδίου δείκτη του τελευταίου κόμβου πρέπει να είναι ίση με `NULL`, ώστε να προσδιορίζεται το τέλος της λίστας.

Η LIFO *(Last In First Out)* στοίβα αποτελεί μία ειδική περίπτωση λίστας με τους ακόλουθους περιορισμούς:

α) Η εισαγωγή ενός νέου κόμβου γίνεται μόνο στην αρχή της στοίβας. Δηλαδή, κάθε κόμβος που εισάγεται γίνεται η νέα κεφαλή της στοίβας.

β) Ο μόνος κόμβος που επιτρέπεται να διαγράψουμε είναι η κεφαλή της στοίβας.

Μία τέτοια στοίβα λειτουργεί όπως μία στοίβα από άπλυτα πιάτα, όπου το κάθε νέο πιάτο το τοποθετούμε στην κορυφή της στοίβας και το πιάτο που πλένουμε το αφαιρούμε από την κορυφή της.

Να ορίσετε μία δομή `Node` με πεδία: αριθμός μητρώου και βαθμός. Να γραφεί ένα πρόγραμμα το οποίο να δημιουργεί μία LIFO στοίβα της οποίας κάθε κόμβος να είναι μία τέτοια δομή. Συγκεκριμένα, να υλοποίησετε την συνάρτηση `add_stack()`, η οποία να αποθηκεύει τα στοιχεία ενός φοιτητή στην στοίβα (κάθε φοιτητής αντιστοιχεί σε έναν κόμβο). Κάθε αποθήκευση πρέπει να γίνεται σε έναν νέο κόμβο, ο οποίος θα αποτελεί την νέα κεφαλή της στοίβας. Να αποθηκεύσετε στη στοίβα τα στοιχεία 5 φοιτητών.

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

typedef struct Node
{
	int code;
	float grd;
	struct Node *next; /* Δείκτης στον επόμενο κόμβο της στοίβας.
*/
} Node;

Node *head; /* Καθολική μεταβλητή που δείχνει πάντα στην κεφαλή της
στοίβας. */

void add_stack(Node *p);

int main()
{
	int i;
	Node n;

	head = NULL; /* Αρχική τιμή στον δείκτη head, η οποία δηλώνει
ότι η στοίβα είναι άδεια. */
	for(i = 0; i < 5; i++)
	{
		printf("\nEnter code: ");
		scanf("%d", &n.code);
```

```
            printf("Enter grade: ");
            scanf("%f", &n.grd);

            add_stack(&n);
      }
      return 0;
}

void add_stack(Node *p)
{
      Node *new_node;

      /* Δέσμευση μνήμης για τη δημιουργία του νέου κόμβου. */
      new_node = (Node*) malloc(sizeof(Node));
      if(new_node == NULL)
      {
            printf("Error: No memory");
            exit(1);
      }
      /* Αντιγραφή των στοιχείων του φοιτητή στον νέο κόμβο. Αντί να
αντιγράφουμε ένα προς ένα τα στοιχεία, μπορούμε να γράψουμε
*new_node = *p; */
      new_node->code = p->code;
      new_node->grd = p->grd;

      new_node->next = head; /* Ο νέος κόμβος εισάγεται στην αρχή της
στοίβας μέσω του δείκτη next. Για παράδειγμα, την πρώτη φορά που
εισάγεται ένας φοιτητής η τιμή του new_node->next γίνεται ίση με την
αρχική τιμή του head που είναι NULL. */
      head = new_node; /* Ο νέος κόμβος γίνεται η νέα κεφαλή της
στοίβας. */
}
```

β. Να προσθέσετε μία συνάρτηση, η οποία να εμφανίζει τα στοιχεία των φοιτητών. Να τροποποιήσετε το πρόγραμμα, ώστε να ελέγξετε την λειτουργία της συνάρτησης.

```
void show_stack()
{
      Node *p;

      p = head;
      printf("\n***** Student Data *****\n\n");
      while(p != NULL)
      {
            printf("C:%d G:%.2f\n\n", p->code, p->grd);
            p = p->next; /* Σε κάθε επανάληψη, ο p δείχνει στον
επόμενο κόμβο. Όταν η τιμή του γίνει ίση με NULL σημαίνει ότι δεν
υπάρχει άλλος κόμβος και ο βρόχος τερματίζεται. */
      }
}

int main()
{
      ...
      show_stack();
      return 0;
}
```

γ. Να προσθέσετε μία συνάρτηση, η οποία να διαγράφει την κεφαλή της στοίβας. Να τροποποιήσετε το πρόγραμμα, ώστε να ελέγξετε την λειτουργία της συνάρτησης.

```
void pop()
{
	Node *p;

	p = head->next; /* Ο p δείχνει στον επόμενο κόμβο της στοίβας.
Αυτός ο κόμβος θα γίνει η νέα κεφαλή. */
	printf("\nStudent with code = %d is deleted\n", head->code);
	free(head); /* Αποδέσμευση μνήμης. Η πληροφορία για το ποιος
είναι ο επόμενος κόμβος δεν χάθηκε, γιατί αποθηκεύτηκε προηγουμένως
στον δείκτη p. */
	head = p; /* Τώρα, ο head δείχνει στη νέα κεφαλή της στοίβας.
*/
/* Για να είναι σαφές, όπως επεξηγείται στα παραπάνω σχόλια, είναι
λάθος να γράψουμε την pop() σαν:
void pop()
{
	free(head);
	head = head->next;
}
γιατί αφού αποδεσμεύεται η μνήμη για τον δείκτη head, δεν μπορούμε να
τον χρησιμοποιήσουμε πάλι. Αυτός είναι ο λόγος που χρησιμοποιούμε τη
βοηθητική μεταβλητή p. */
}

int main()
{
	...
	pop();
	show_stack();
	return 0;
}
```

δ. Να προσθέσετε μία συνάρτηση, η οποία να αποδεσμεύει τη μνήμη της στοίβας. Να τροποποιήσετε το πρόγραμμα, ώστε να ελέγξετε την λειτουργία της συνάρτησης.

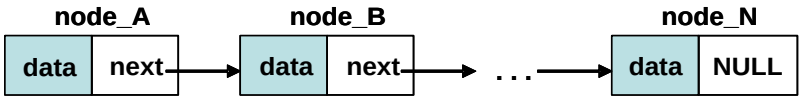
```
void free_stack()
{
	Node *p, *next_node;

	p = head;
	while(p != NULL)
	{
		next_node = p->next; /* Ο next_node δείχνει στον κόμβο
που είναι μετά από αυτόν που θα διαγραφεί. */
		printf("Student with code = %d is deleted\n", p->code);
		free(p); /* Αποδέσμευση μνήμης. Η πληροφορία για το ποιος
είναι ο επόμενος κόμβος δεν χάθηκε, γιατί αποθηκεύτηκε προηγουμένως
στον δείκτη next_node. */
		p = next_node; /* Τώρα, ο p δείχνει στον επόμενο κόμβο.
*/
	}
}
int main()
{
	...
	free_stack();
	return 0;
}
```

ΑΠΛΑ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ (5)

Ε1. Θεωρήστε ότι έχει υλοποιηθεί μία στοίβα με κόμβους δομές τύπου `Node` (δείτε προηγούμενο εργαστήριο). Να προσθέσετε μία συνάρτηση που να δέχεται σαν παράμετρο τον κωδικό ενός φοιτητή και αν υπάρχει στη στοίβα να επιστρέφει ένα δείκτη στην αντίστοιχη δομή, αλλιώς `NULL`. Γράψτε ένα τμήμα κώδικα στη `main()`, ώστε να ελέγξετε την λειτουργία της συνάρτησης.

```
Node* find(int code)
{
	Node *ptr;

	ptr = head;
	while(ptr != NULL)
	{
		if(ptr->code == code)
			return ptr;
		else
			ptr = ptr->next;
	}
	return NULL;
}

int main()
{
	Node *ptr;
	int code;

	...

	printf("\nEnter code to search: ");
	scanf("%d", &code);

	ptr = find(code);
	if(ptr == NULL)
		printf("Code:%d doesn't exist\n", code);
	else
		printf("Code:%d Grade:%.2f\n", ptr->code, ptr->grd);

	...
	return 0;
}
```

Θ2. Μία δυναμική δομή δεδομένων, δηλαδή μία δομή της οποίας το μέγεθος μπορεί να αυξομειώνεται κατά την εκτέλεση του προγράμματος, αποτελείται από ένα ή περισσότερα συνδεδεμένα στοιχεία, που συνήθως ονομάζονται κόμβοι. Κάθε κόμβος αντιπροσωπεύεται από μία δομή, η οποία περιέχει τα δεδομένα του κόμβου και ένα πεδίο που είναι δείκτης (π.χ. το πεδίο `next`) στον επόμενο κόμβο. Το επόμενο σχήμα παρουσιάζει την πιο συνηθισμένη δυναμική δομή δεδομένων, που είναι μία απλά συνδεδεμένη λίστα.



Ο πρώτος κόμβος ονομάζεται κεφαλή (head) της λίστας και ο τελευταίος ουρά (tail). Η τιμή του πεδίου δείκτη του τελευταίου κόμβου πρέπει να είναι ίση με `NULL`, ώστε να προσδιορίζεται το τέλος της λίστας.

α. Εισαγωγή Κόμβου

Ένας νέος κόμβος μπορεί να εισαχθεί σε οποιαδήποτε θέση της λίστας. Για να τον εισάγουμε εξετάζουμε τις ακόλουθες περιπτώσεις:

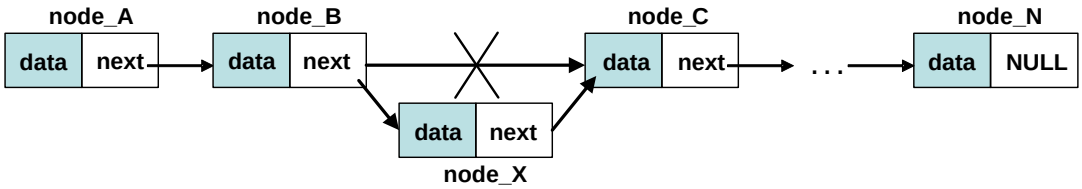
1) Αν η λίστα είναι κενή, ο κόμβος που εισάγεται γίνεται ταυτόχρονα και κεφαλή και ουρά της λίστας. Η τιμή του δείκτη του γίνεται `NULL`, αφού δεν υπάρχει επόμενος κόμβος.

2) Αν η λίστα δεν είναι κενή εξετάζουμε τις ακόλουθες υπο-περιπτώσεις:

α) Αν ο νέος κόμβος τοποθετηθεί στην αρχή της λίστας, τότε αποτελεί τη νέα κεφαλή της και ο δείκτης του δείχνει στην παλιά κεφαλή, που τώρα γίνεται ο δεύτερος κόμβος της λίστας.

β) Αν ο νέος κόμβος τοποθετηθεί στο τέλος της λίστας, τότε αποτελεί τη νέα ουρά της και η τιμή του δείκτη του γίνεται `NULL`. Ο κόμβος που ήταν η ουρά της λίστας γίνεται ο προτελευταίος κόμβος με την τιμή του δείκτη του να αλλάζει από `NULL` και να δείχνει στον νέο κόμβο.

γ) Αν ο νέος κόμβος τοποθετηθεί μετά από έναν ενδιάμεσο κόμβο της λίστας, κάνουμε τον δείκτη αυτού του κόμβου να δείχνει στον νέο κόμβο και τον δείκτη του νέου κόμβου να δείχνει στον κόμβο που έδειχνε ο ενδιάμεσος κόμβος. Η διαδικασία αυτή απεικονίζεται στο παρακάτω σχήμα με τον κόμβο X να εισάγεται μεταξύ των κόμβων B και C.



Σχήμα_1. Εισαγωγή Κόμβου

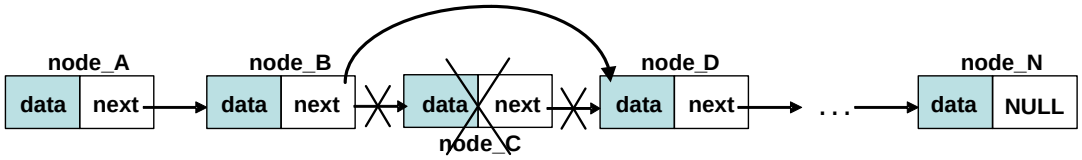
β. Διαγραφή Κόμβου

Για να διαγράψουμε έναν κόμβο της λίστας και να αποδεσμεύσουμε τη μνήμη που καταλαμβάνει εξετάζουμε τις ακόλουθες περιπτώσεις:

1) Αν ο κόμβος είναι η κεφαλή της λίστας εξετάζουμε τις ακόλουθες υπο-περιπτώσεις:

α) αν υπάρχει επόμενος κόμβος, τότε αυτός ο κόμβος γίνεται η νέα κεφαλή της λίστας.
β) αν δεν υπάρχει επόμενος κόμβος, η λίστα γίνεται κενή.

2) Αν ο κόμβος είναι η ουρά της λίστας, ο προηγούμενος κόμβος γίνεται η νέα ουρά της λίστας και η τιμή του δείκτη του γίνεται `NULL`.
3) Αν ο κόμβος βρίσκεται ανάμεσα σε δύο κόμβους, κάνουμε τον δείκτη του προηγούμενου κόμβου να δείχνει στον επόμενο κόμβο από αυτόν που θέλουμε να διαγράψουμε. Η διαδικασία αυτή απεικονίζεται στο παρακάτω σχήμα με τη διαγραφή του κόμβου C.



Σχήμα_2. Διαγραφή Κόμβου

Να ορίσετε μία δομή `Node` με πεδία: αριθμός μητρώου και βαθμός. Να γραφεί ένα πρόγραμμα το οποίο να δημιουργεί μία απλά συνδεδεμένη λίστα της οποίας κάθε κόμβος να είναι μία τέτοια δομή. Συγκεκριμένα, να υλοποίησετε την συνάρτηση `add_list_end()`, η οποία να αποθηκεύει τα στοιχεία ενός φοιτητή στη λίστα. Κάθε αποθήκευση πρέπει να γίνεται σε έναν νέο κόμβο, ο οποίος θα αποτελεί την νέα ουρά της λίστας. Να αποθηκεύσετε στη λίστα τα στοιχεία 5 φοιτητών.

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

typedef struct Node
{
	int code;
	float grd;
	struct Node *next; /* Δείκτης στον επόμενο κόμβο της λίστας. */
} Node;

Node *head; /* Καθολική μεταβλητή που δείχνει πάντα στον πρώτο κόμβο της λίστας. */
Node *tail; /* Καθολική μεταβλητή που δείχνει πάντα στον τελευταίο κόμβο της λίστας. */

void add_list_end(Node *p);
void free_list();

int main()
{
	int i;
	Node n;

	head = NULL; /* Αρχική τιμή στον δείκτη head, η οποία δηλώνει ότι η λίστα είναι άδεια. */
	for(i = 0; i < 5; i++)
	{
		printf("\nEnter code: ");
		scanf("%d", &n.code);

		printf("Enter grade: ");
		scanf("%f", &n.grd);

		add_list_end(&n);
	}
```

```
	free_list();
	return 0;
}

void add_list_end(Node *p)
{
	Node *new_node;

	/* Δέσμευση μνήμης για τη δημιουργία του νέου κόμβου. */
	new_node = (Node*) malloc(sizeof(Node));
	if(new_node == NULL)
	{
		printf("Error: No memory");
		exit(1);
	}
	/* Αντιγραφή των στοιχείων του φοιτητή στον νέο κόμβο. Αντί να
αντιγράφουμε ένα προς ένα τα στοιχεία, μπορούμε να γράψουμε
*new_node = *p; */
	new_node->code = p->code;
	new_node->grd = p->grd;
	new_node->next = NULL;

	if(head == NULL)
		head = tail = new_node; /* Αν η λίστα είναι άδεια οι
δείκτες head και tail θα δείχνουν στον νέο κόμβο. */
	else
	{
		tail->next = new_node; /* Ο νέος κόμβος εισάγεται στο
τέλος της λίστας μέσω του δείκτη next. */
		tail = new_node; /* Τώρα, ο δείκτης tail δείχνει στον νέο
τελευταίο κόμβο. */
	}
}

void free_list()
{
	Node *p, *next_node;

	p = head;
	while(p != NULL)
	{
		next_node = p->next; /* Ο next_node δείχνει στον κόμβο
που είναι μετά από αυτόν που θα διαγραφεί. */
		printf("Student with code = %d is deleted\n", p->code);
		free(p); /* Αποδέσμευση μνήμης. Η πληροφορία για το ποιος
είναι ο επόμενος κόμβος δεν χάθηκε, γιατί αποθηκεύτηκε προηγουμένως
στον δείκτη next_node. */
		p = next_node; /* Τώρα, ο p δείχνει στον επόμενο κόμβο.
*/
	}
}
```

β. Να προσθέσετε μία συνάρτηση, η οποία να εμφανίζει τα στοιχεία των φοιτητών.

```
void show_list()
{
	Node *p;

	p = head;
	printf("\n***** Student Data *****\n\n");
	while(p != NULL)
```

```
	{
		printf("C:%d G:%.2f\n\n", p->code, p->grd);
		p = p->next; /* Σε κάθε επανάληψη, ο p δείχνει στον επόμενο κόμβο. Όταν η τιμή του γίνει ίση με NULL σημαίνει ότι δεν υπάρχει άλλος κόμβος και ο βρόχος τερματίζεται. */
	}
}

int main()
{
	...
	show_list();
	free_list();
	return 0;
}
```

γ. Να προσθέσετε μία συνάρτηση, η οποία να εισάγει τα στοιχεία ενός νέου φοιτητή μετά από ένα συγκεκριμένο κόμβο της λίστας. Το πρόγραμμα να διαβάζει τον αριθμό μητρώου, να εντοπίζει τον αντίστοιχο κόμβο στη λίστα (αν υπάρχει) και να εισάγει τα στοιχεία ενός νέου φοιτητή σε έναν νέο κόμβο αμέσως μετά από αυτόν. Γράψτε ένα τμήμα κώδικα στη `main()`, ώστε να ελέγξετε την λειτουργία της συνάρτησης.

```
int add_list(Node *p, int code)
{
	Node *new_node, *ptr;

	ptr = head;
	/* Διατρέχουμε τη λίστα μέχρι να βρούμε τον κόμβο με τον συγκεκριμένο αριθμό μητρώου. Αν βρεθεί, ο νέος κόμβος προστίθεται μετά από αυτόν και η συνάρτηση τερματίζει. */
	while(ptr != NULL)
	{
		if(ptr->code == code)
		{
			new_node = (Node*)malloc(sizeof(Node));
			if(new_node == NULL)
			{
				printf("Error: Not available memory\n");
				exit(1);
			}
			/* Αντιγραφή των στοιχείων του φοιτητή. */
			new_node->code = p->code;
			new_node->grd = p->grd;

			/* Οι παρακάτω δύο εντολές υλοποιούν το σχήμα 1. */
			new_node->next = ptr->next; /* Τώρα, ο νέος κόμβος δείχνει στον κόμβο που δείχνει ο τρέχων κόμβος. */
			ptr->next = new_node; /* Τώρα, ο τρέχων κόμβος δείχνει στον νέο κόμβο, ο οποίος έτσι προστίθεται στη λίστα. */

			if(ptr == tail) /* Ελέγχουμε αν ο νέος κόμβος προστέθηκε στο τέλος της λίστας. Αν ναι, αποτελεί τη νέα ουρά της λίστας. */
				tail = new_node;
			return 0;
		}
		else
			ptr = ptr->next; /* Ελέγχουμε τον επόμενο κόμβο. */
	}
```

```
	return -1; /* Αν ο κώδικας φτάσει σε αυτό το σημείο σημαίνει ότι δεν υπάρχει κόμβος στη λίστα που να έχει αριθμό μητρώου ίσο με αυτό που εισήγαγε ο χρήστης. */
}

int main()
{
	int i, code;
	Node n;

	... /* Μετά τον for βρόχο. */
	printf("\nEnter student code after which the new student will be added: ");
	scanf("%d", &code);

	printf("\nEnter code: ");
	scanf("%d", &n.code);

	printf("Enter grade: ");
	scanf("%f", &n.grd);

	if(add_list(&n, code) == -1)
		printf("\nStudent with code = %d does not exist\n", code);
	else
		printf("\nStudent with code = %d is added\n", code);

	free_list();
	return 0;
}
```

δ. Να προσθέσετε μία συνάρτηση, η οποία να δέχεται σαν παράμετρο τον κωδικό ενός φοιτητή και αν υπάρχει στην λίστα να τον διαγράφει. Γράψτε ένα τμήμα κώδικα στη `main()`, ώστε να ελέγξετε την λειτουργία της συνάρτησης.

```
int del_node(int code)
{
	Node *ptr, *prev_node; /* Ο δείκτης prev_node θα δείχνει στον προηγούμενο κόμβο από αυτόν που θα διαγράψουμε. */

	ptr = prev_node = head;
	while(ptr != NULL)
	{
		if(ptr->code == code)
		{
			if(ptr == head)
				head = ptr->next; /* Αν ο κόμβος είναι η κεφαλή της λίστας, τότε ο επόμενος κόμβος γίνεται η νέα κεφαλή. Αν δεν υπάρχει επόμενος κόμβος, δηλαδή η λίστα περιέχει μόνο την κεφαλή, τότε η τιμή του δείκτη head θα γίνει ίση με NULL. */
			else
			{
				/* Τώρα, ο ptr δείχνει στον κόμβο που θα διαγραφεί και ο δείκτης prev_node στον προηγούμενο. Η επόμενη εντολή συνδέει τον προηγούμενο κόμβο με τον επόμενο από αυτόν που θα διαγραφεί, δηλαδή, υλοποιεί το σχήμα 2. */
				prev_node->next = ptr->next;
```

```
                        if(ptr == tail) /* Ελέγχουμε αν ο κόμβος που
θα διαγράψουμε είναι η ουρά της λίστας. Αν είναι, ο προηγούμενός του
γίνεται η νέα ουρά της λίστας. */
                                tail = prev_node;
                    }
                    free(ptr); /* Διαγραφή κόμβου. */
                    return 0;
              }
              else
              {
                    prev_node = ptr; /* Τώρα, ο prev_node δείχνει στον
κόμβο που μόλις ελέγξαμε και διαπιστώσαμε ότι δεν έχει τον επιθυμητό
αριθμό μητρώου. */
                    ptr = ptr->next; /* Τώρα, ο ptr δείχνει στον
επόμενο κόμβο. Σημειώστε ότι ο prev_node θα δείχνει πάντα στον
προηγούμενο κόμβο από αυτόν που ελέγχουμε. */
              }
        }
        return -1;
}

int main()
{
        int i, code;

        ... /* Μετά τον for βρόχο. */
        printf("\nEnter student code to delete: ");
        scanf("%d", &code);
        if(del_node(code) == -1)
              printf("\nStudent with code = %d does not exist\n",
code);
        else
              printf("\nStudent with code = %d is deleted\n", code);

        free_list();
        return 0;
}
```