Α' Εξάμηνο
ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Εργαστήριο – 10η εβδομάδα.
Κοζάνη, 14 Ιανουαρίου 2009.

Τα προγράμματα 1 έως 4 αφορούν την ταξινόμηση και αναζήτηση δεδομένων. Το πρόγραμμα 5 υλοποιεί μια στοίβα ως γραμμική συνδεδεμένη λίστα.

Άσκηση 10-1 ***Μέθοδος Ταχείας Ταξινόμησης (quick sort)***
Δημιουργήστε τα παρακάτω αρχεία που θα ανήκουν στην εφαρμογή μας:
α) Ένα αρχείο με το όνομα main.c και τα ακόλουθα περιεχόμενα (με γαλάζιο οι γραμμές που δίνονται ήδη από το περιβάλλον ανάπτυξης και με κόκκινο αυτές που θα προσθέσουμε):

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

#define NDAT 10

int main(char argc, char *argv[])
{
    int i, unsorted[NDAT], sorted[NDAT];

    for (i = 0; i < NDAT; i++)
        unsorted[i] = sorted[i] =
        (int) (NDAT * (float) rand() / (float) RAND_MAX);

    sortMethod();
    sort(sorted, NDAT);
    printf("\n\tNo\tUnsorted\tSorted\n");
    for(i = 0; i < NDAT; i++)
       printf("\t%d\t   %d\t\t  %d\n", i, unsorted[i], sorted[i]);
    printf("Finished!\n");
    system("PAUSE");
    return 0;
}
```

Στο πρόγραμμα αυτό δημιουργούμε δύο ακέραιους πίνακες διάστασης NDAT = 10 (το NDAT ορίστηκε πιο πάνω ως συμβολική σταθερά με την οδηγία #define). Σε αυτούς δίνουμε σε κάθε στοιχείο την ίδια (τυχαία) τιμή που θα είναι ένας τυχαίος ακέραιος αριθμός από 0 έως NDAT-1. Αρχικά λοιπόν τα περιεχόμενα των δύο πινάκων ταυτίζονται. Μετά, ο ένας από τους δύο, με το όνομα sorted, θα υποστεί την επεξεργασία μιας συνάρτησης που υλοποιεί κάποιον αλγόριθμο ταξινόμησης μετά από την οποία, τα περιεχόμενά του θα έχουν τοποθετηθεί κατ' αύξουσα σειρά. Τα στοιχεία των πινάκων τυπώνονται στη συνέχεια, στην ίδια σειρά για κάθε αριθμοδείκτη.

β) Ένα αρχείο με το όνομα mylib.c και τα ακόλουθα περιεχόμενα:

```
#include <stdio.h>
#include "mylib.h"
```

```
void sortMethod( void ) {

    int method;

    printf("\nPick a method: \n");
    printf("1 = quick sort\n");
    scanf("%d", &method);

    switch(method) {
        case 1:
            sort = quick_sort;
            break;
        default:
            printf("\nwrong option!");
            sort = do_nothing; /* to avoid segmentation fault */
    }
}

void do_nothing( int *a, int n ) {}

void swap( int *x, int *y ) {
    int t;
    t = *x;
    *x = *y;
    *y = t;
}

void quick_sort(int *a, int n) {
    int i, j, middle;

    i = 0;
    j = n - 1;
    middle = a[j/2];
    do {
        while(a[i] < middle)
            i++;
        while(a[j] > middle)
            j--;
        if(i < j)
            swap(a+i, a+j);
        else if(i > j)
            break;
        i++;
        j--;
    } while(i <= j);
    if(j > 0) quick_sort(a, j+1);
    if(i < n-1) quick_sort(a+i, n-i);
```

```
}
```

Σε αυτό το αρχείο ορίζεται πρώτα η συνάρτηση sortMethod η οποία καλείται και στο κύριο πρόγραμμα. Αυτή ορίζει ότι το sort που είναι δείκτης σε συνάρτηση θα δείχνει σε μία συνάρτηση που διαλέγουμε μεταξύ αυτών που μας προτείνει. Προς το παρόν προτείνει μόνο μία. Αν δώσουμε αριθμό που δεν αντιστοιχεί σε προτεινόμενη επιλογή, τότε θα δείχνει στη συνάρτηση do_nothing που, όπως μαρτυρεί και το όνομά της, δεν κάνει τίποτε, ενώ τυπώνεται και ένα μήνυμα λάθους. Σε αυτό το αρχείο υπάρχει επίσης και η συνάρτηση swap για την ανταλλαγή των τιμών δύο μεταβλητών (η μία παίρνει την τιμή της άλλης) που παίζει βοηθητικό ρόλο στην ταξινόμηση όπως αυτή υλοποιείται αμέσως παρακάτω. Τέλος, υπάρχει η συνάρτηση που υλοποιεί την πρώτη μέθοδο που εξετάζουμε, τη μέθοδο ταχείας ταξινόμησης. Η συγκεκριμένη υλοποίηση είναι αναδρομική. Οι αναδρομικές κλήσεις της συνάρτησης υποδεικνύονται με έντονα γράμματα.

γ) Ένα αρχείο "επικεφαλίδας" με το όνομα mylib.h και τα ακόλουθα περιεχόμενα:

```
#ifndef MYLIB_H
#define MYLIB_H

void (*sort)();
void sortMethod( void );
void do_nothing( int * , int );
void swap( int * , int * );
void quick_sort(int * , int );

#endif
```

Εδώ απλά δηλώνεται ο δείκτης σε συνάρτηση sort, καθώς και οι συναρτήσεις που ορίσαμε πιο πάνω.

Μεταγλωττίστε και εκτελέστε.

Άσκηση 10-2 ***Συνάρτηση ταχείας ταξινόμησης πρότυπης βιβλιοθήκης (standard library qsort)***

Η επίδοση ενός αλγόριθμου μπορεί να επηρεαστεί από τη συγκεκριμένη μορφή των δεδομένων εισόδου. Η παραπάνω υλοποίηση της ταχείας ταξινόμησης γενικά δουλεύει καλά, είναι όμως δυνατό να γίνει πολύ αργή αν π.χ. το μεσαίο στοιχείο κάθε τμήματος χωρίζει αυτό σε μία πολύ μικρή και μία πολύ μεγάλη περιοχή. Τότε θα γίνουν πολλές αναδρομικές κλήσεις με κίνδυνο, μάλιστα, υπερχείλισης της στοίβας του υπολογιστή. Ευτυχώς υπάρχουν και άλλοι τρόποι υλοποίησης που αποφεύγουν τα προβλήματα αυτών των ειδικών περιπτώσεων.

Η μέθοδος ταχείας ταξινόμησης υπάρχει και σαν συνάρτηση της πρότυπης βιβλιοθήκης με το όνομα qsort, δηλωμένη στο αρχείο επικεφαλίδας <stdlib.h> και θα την προσθέσουμε σαν άλλη μία επιλογή για το πρόγραμμά μας. Η συνάρτηση qsort παίρνει ως παραμέτρους εισόδου τον πίνακα (ως δείκτη τύπου void για να είναι γενικής χρήσης) και τον αριθμό των στοιχείων του, το πλάτος σε bytes του τύπου των στοιχείων (double, int, char, ...), και ένα δείκτη σε μία ακέραια συνάρτηση η οποία έχει σκοπό να συγκρίνει δύο αριθμούς. Αυτή η συνάρτηση, έστω compare το όνομά της, δε δίνεται έτοιμη αλλά πρέπει να οριστεί από τον προγραμματιστή έτσι ώστε να παίρνει για είσοδο δύο δείκτες τύπου const void και να επιστρέφει 1, 0 ή -1 ανάλογα με το αν η πρώτη παράμετρος είναι μεγαλύτερη, ίση ή μικρότερη από τη δεύτερη.

Σημείωση: Οι δείκτες τύπου void είναι δείκτες γενικής χρήσης και μπορούμε να τους χρησιμοποιούμε π.χ. για να περνάμε ορίσματα διαφόρων τύπων σε μία συνάρτηση.

Βάσει των παραπάνω, αλλά και για να μπορέσουμε να διατηρήσουμε τη δομή του προγράμματός μας, θα ορίσουμε μία συνάρτηση την οποία θα δείχνει η sort, με ίδιο πρωτότυπο (τύπο και λίστα παραμέτρων εισόδου) με τις προηγούμενες η οποία θα καλεί την qsort. Επίσης, θα ορίσουμε μία συνάρτηση για τη σύγκριση δύο ακέραιων, αλλά με το πρωτότυπο που απαιτεί η qsort.

Στο αρχείο επικεφαλίδας κάνουμε τις προσθήκες που φαίνονται με κόκκινο χρώμα, ανάμεσα από την τελευταία προηγούμενη δήλωση και το #endif:

```
#ifndef MYLIB_H
#define MYLIB_H

void (*sort)();
void sortMethod( void );
void do_nothing( int * , int );
void quick_sort(int * , int );
void standard_qsort(int * , int );
int compare(const void * , const void * );

#endif
```

Η συνάρτηση standard_qsort θα καλεί τη συνάρτηση βιβλιοθήκης qsort ενώ η compare θα συγκρίνει δύο ακέραιους αριθμούς με τον τρόπο που θα δούμε αμέσως.

Στο αρχείο mylib.c πρέπει πρώτα να προσθέσουμε το αρχείο <stdlib.h> για να μπορέσουμε να κάνουμε χρήση της qsort:

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

Μετά, προσθέτουμε τον κώδικα για τη συμπλήρωση του μενού επιλογών:

```
......
    printf("\nPick a method: \n");
    printf("1 = quick sort\n");
    printf("2 = standard library qsort\n");
    scanf("%d", &method);

    switch(method) {
        case 1:
            sort = quick_sort;
            break;
        case 2:
            sort = standard_qsort;
            break;
        default:
            printf("\nwrong option!");
            sort = do_nothing; /* to avoid segmentation fault */
    }
......
```

Τέλος, προσθέτουμε την υλοποίηση των συναρτήσεων που αναφέραμε:

```
void standard_qsort(int *a, int n) {
    qsort(a, n, sizeof(a[0]), compare);
}
```

```
int compare(const void *xx, const void *yy) {
    int value;
    int *x, *y;

    x = (int *) xx;
    y = (int *) yy;
    if(*x > *y)
        value = 1;
    else if (*x == *y)
        value = 0;
    else
        value = -1;

    return(value);
}
```

Η συνάρτηση standard_qsort είναι πολύ απλή γιατί το μόνο που κάνει είναι να καλεί την qsort με τα κατάλληλα ορίσματα. Παρατηρείστε ότι το όνομα της συνάρτησης compare χρησιμεύει ως δείκτης σε αυτή. Η συνάρτηση compare παίρνει δύο δείκτες τύπου void και αποδίδει τις τιμές όπου δείχνουν σε δύο δείκτες τύπου int με τη βοήθεια καταλληλης μετατροπής τύπου που υποδηλώνεται από το (int *).

Άσκηση 10-3 ***Επιδόσεις αλγόριθμων ταξινόμησης***

Για να διαπιστώσουμε την αποδοτικότητα ενός αλγόριθμου στην πράξη είναι χρήσιμο να χρονομετρήσουμε την ολοκλήρωσή του για διάφορα δεδομένα εισόδου.
Στο κύριο πρόγραμμα κάνουμε τις εξής αλλαγές:
Εισάγουμε ένα ακόμη αρχείο επικεφαλίδας για να χρησιμοποιήσουμε συναρτήσεις μέτρησης του χρόνου:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "p9lib.h"
```

Εισάγουμε την εξής δήλωση μεταβλητής (με κόκκινο):

```
    int i, unsorted[NDAT], sorted[NDAT];
    clock_t t;
```

Εισάγουμε δύο εντολές για τον προσδιορισμό της χρονικής στιγμής έναρξης της ταξινόμησης και της διάρκειας αυτής, πριν και μετά από την κλήση της sort, ως εξής:

```
    sortMethod();
    t = clock();
    sort(sorted, NDAT);
    t = clock() - t;
```

Τροποποιούμε το υπόλοιπο τμήμα που αφορά την εκτύπωση αποτελεσμάτων προσθέτοντας την εμφάνιση της διάρκειας ταξινόμησης και περικλείοντας την εμφάνιση των ταξινομημένων δεδομένων σε μια δομή if για να εκτυπώνονται προαιρετικά, ως εξής:

```
     printf("\nData sorted within %lf sec", (double) t / (double)
```

```
CLOCKS_PER_SEC);
    printf("\nDisplay results? 1 for yes ");
    scanf("%d", &i);
    if(i == 1) {
       printf("\n\tNo\tUnsorted\tSorted\n");
       for(i = 0; i < NDAT; i++)
          printf("\t%d\t %d\t\t  %d\n", i, unsorted[i], sorted[i]);
    }
    printf("Finished!\n");
```

Αλλάζοντας τις τιμές του NDAT (π.χ. 1000000), εκτελέστε το πρόγραμμα για τις δύο επιλογές και συγκρίνετε τις υλοποιήσεις της μεθόδου ταχείας ταξινόμησης ως προς την ταχύτητά τους.

Άσκηση 10-4 ***Η μέθοδος δυαδικής αναζήτησης*****: *συνάρτηση βιβλιοθήκης bsearch***

Εφόσον έχουμε ταξινομήσει τα δεδομένα μας είναι εύκολο να αναζητήσουμε σε αυτά το στοιχείο που μας ενδιαφέρει. Διακρίνουμε δύο κύριες μεθόδους: τη γραμμμική αναζήτηση (δηλαδή τον έλεγχο κάθε στοιχείου στη σειρά μέχρι να βρούμε αυτό που θέλουμε) και τη δυαδική αναζήτηση. Η πρώτη είναι γενική με την έννοια ότι εφαρμόζεται ακόμη και όταν τα δεδομένα είναι αταξινόμητα αλλά είναι αργή, με πολυπλοκότητα O(N). Η δεύτερη απαιτεί ταξινομημένα στοιχεία αλλά είναι πολύ γρηγορότερη. Βασίζεται στη σύγκριση του κλειδιού αναζήτησης με το μεσαίο στοιχείο του διαστήματος τιμών και αντικατάσταση αυτού του διαστήματος από το μισό όπου βρίσκεται το ζητούμενο στοιχείο.

Στη συνέχεια προσθέτουμε στο πρόγραμμα μία λειτουργία δυαδικής αναζήτησης βασισμένη στη συνάρτηση βιβλιοθήκης bsearch, που καλείται με τρόπο παρόμοιο με αυτόν που είδαμε σχετικά με την qsearch. Η συνάρτηση bsearch, δηλωμένη στο αρχείο stdlib.h, παίρνει ως παραμέτρους εισόδου ένα δείκτη στο κλειδί αναζήτησης, τον πίνακα δεδομένων και τον αριθμό των στοιχείων του, το πλάτος σε bytes του τύπου των στοιχείων (double, int, char, ...), και ένα δείκτη σε μία ακέραια συνάρτηση η οποία έχει σκοπό να συγκρίνει δύο αριθμούς. Αυτή η συνάρτηση δε θα είναι άλλη από την compare που ορίσαμε πιο πριν και χρησιμοποιήσαμε στην qsort.

Θέλουμε να βρούμε αν ένας αριθμός που δίνουμε από το πληκτρολόγιο υπάρχει στον ταξινομημένο πίνακα. Η bsearch μας επιστρέφει ένα δείκτη (τύπου void, δηλαδή δείκτη “μπαλαντέρ”) στην περιοχή μνήμης όπου βρίσκεται το στοιχείο του οποίου το κλειδί αναζητούμε. Αν αυτός ο δείκτης είναι NULL τότε το στοιχείο δε βρέθηκε στον πίνακα.

Προσθέτουμε μία νέα δήλωση στις δηλώσεις μεταβλητών στην αρχή του προγράμματος:

```
    int *found;
```

Μετά, στο τέλος του προγράμματος, πριν από την `printf("Finished!\n");` προσθέτουμε τον ακόλουθο κώδικα:

```
    printf("\nNumber between 0 and NDAT to find: ");
    scanf("%d", &i);
       found = (int *) bsearch(&i, sorted, NDAT, sizeof(int),
compare);
    if(found == NULL)
        printf("\n%d not found!\n", i);
    else
        printf("\n%d has been found!\n", i);
```

Μεταγλωττίστε και εκτελέστε

Άσκηση 10-5 ***Υλοποίηση στοίβας***

Παρακάτω δίνεται ένα πρόγραμμα το οποίο υλοποιεί μία στοίβα όπου αποθηκεύονται κάποιοι αριθμοί. Δομές όπως στοίβες και ουρές μπορούν να υλοποιηθούν τόσο με πίνακες όσο και με συνδεδεμένες λίστες. Εδώ χρησιμοποιήσαμε μία γραμμική συνδεδεμένη λίστα. Η λίστα δουλεύει ως στοίβα επειδή οι συναρτήσεις push και pop που επιτρέπουν την προσθήκη και αφαίρεση νέων στοιχείων ακολουθούν την αρχή FIFO. Το πρόγραμμα δείχνει τον τρόπο λειτουργίας αυτών των συναρτήσεων.

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

typedef struct list {
    int value;
    struct list *next;
} LIST;

LIST *first, *last; /* global variables */

void push(void);
void pop(LIST *);
void display(void);
void empty(void);

int main (int argc, char *argv[]) {

    int i;

    last = malloc(sizeof(LIST));
    last->value = -1;
    last->next = NULL;
    first = malloc(sizeof(LIST));
    first->value = -1;
    first->next = last;
    for(i=0;i<20;i++)
        push();
    printf("\n\nPrint list\n");
    display();
    empty();
    getchar();
    printf("\nPrint empty list\n");
    display();
    return 0;
}

void push(void)
{
```

```
    static int n=0;
    LIST *p = malloc(sizeof(LIST));
    p->next = first->next;
    first->next = p;
/*
    printf("Give me a number: ");
    scanf("%d", &p->value);*/
    p->value = ++n;
    return;
}

void pop(LIST *p)
{
    if(p->next != NULL)
        p->next = p->next->next;
    return;
}

void display(void)
{
    int i=1;
    LIST *p;
    for(p=first->next;p!=NULL,p->next!=NULL;p=p->next)
        printf("number %d is %d\n", i++, p->value);
    return;
}

void empty(void)
{
    LIST *p;
    p = first;
    while(p->next->next!=NULL)
        pop(p);
    return;
}
```

Μεταγλωττίστε και εκτελέστε