Α' Εξάμηνο
ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Εργαστήριο – 8η σειρά ασκήσεων.
Κοζάνη, 7 Δεκεμβρίου 2007.

Κάθε πρόγραμμα εξαρτάται από αλγόριθμους και δομές δεδομένων, αλλά μόνο λίγα απαιτούν κάποια καινοτομία ως προς αυτά τα δύο. Ακόμη και σε ένα πολύπλοκο πρόγραμμα, οι περισσότερες δομές δεδομένων θα είναι πίνακες, λίστες, δέντρα ή πίνακες κατακερματισμού (hash tables). Όποτε χρειάζεται κάτι πολύπλοκο, πιθανότατα θα βασίζεται σε απλούστερα γνωστά στοιχεία. Όπως υπάρχουν λίγες θεμελιώδεις δομές δεδομένων, έτσι υπάρχουν λίγα θεμελιώδη προβλήματα σχετικά με αυτές και λίγες μέθοδοι για τη λύση αυτών των προβλημάτων. Τα κυριότερα, θεμελιωδέστερα και συχνότερα προβλήματα είναι η **ταξινόμηση** και η **αναζήτηση** σε ένα σύνολο δεδομένων. Τα επόμενα προγράμματα παρουσιάζουν τις κυριότερες μεθόδους ταξινόμησης[1]. Η ταξινόμηση είναι απαραίτητη για την ευχερέστερη και ταχύτερη αναζήτηση μέσα από ένα σύνολο δεδομένων. Με τις μεθόδους αναζήτησης θα ασχοληθούμε την επόμενη φορά.

Πρόγραμμα p9-1 ***Ταξινόμηση Φυσαλίδας (bubble sort)***
Δημιουργήστε ένα project, ονομάστε το p9 και δημιουργήστε τα παρακάτω αρχεία που θα ανήκουν σε αυτό:
α) Ένα αρχείο με το όνομα p9main.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 "p9lib.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++)
```

1 Τα υποπρογράμματα bubble_sort, straight_insertion, shell_sort, heap_sort, adjust και quick_sort που υλοποιούν τις διάφορες μεθόδους βασίστηκαν σε αντίστοιχα παραδείγματα από την εξής πηγή: Β. Σεφερίδης, "C για αρχάριους", εκδόσεις Κλειδάριθμος, Αθήνα 1995.

```
        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 p9lib.c και τα ακόλουθα περιεχόμενα:

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

void sortMethod( void ) {

    int method;

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

    switch(method) {
        case 1:
            sort = bubble_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 bubble_sort( int *a, int n ) {
    int i, flag = 0;
    while(flag == 0) {
        flag = 1;
        for(i = 0; i < n-1; i++)
            if(a[i] > a[i+1]) {
                swap(a+i, a+i+1);
                flag = 0;
            }
    }
}
```

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. Τέλος, υπάρχουν δύο συναρτήσεις που υλοποιούν την πρώτη μέθοδο που εξετάζουμε, τη μέθοδο της φυσαλίδας. Η συνάρτηση swap υλοποιεί την αντιμετάθεση δύο αριθμών (δηλαδή ο ένας παίρνει την τιμή του άλλου) ενώ η συνάρτηση bubble_sort είναι η υλοποίηση του αλγόριθμου της φυσαλίδας.

γ) Ένα αρχείο "επικεφαλίδας" με το όνομα p9lib.h και τα ακόλουθα περιεχόμενα:

```
#ifndef P9LIB_H
#define P9LIB_H

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

#endif
```

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

Άσκηση p9-2 ***Ταξινόμηση παρεμβολής (straight insertion)***

Στο αρχείο επικεφαλίδας p9lib.h που δημιουργήσατε, εισάγετε άλλη μία δήλωση συνάρτησης, όπως φαίνεται παρακάτω (με κόκκινο η νέα γραμμή που θα προστεθεί):

```
#ifndef P9LIB_H
#define P9LIB_H

void (*sort)();
void sortMethod( void );
void swap( int * , int * );
void do_nothing( int * , int );
void bubble_sort( int * , int );
void straight_insertion( int * , int );
```

```
#endif
```
Εδώ δηλώνεται η συνάρτηση που υλοποιεί την επόμενη μέθοδο ταξινόμησης που εξετάζουμε, τη μέθοδο της παρεμβολής.

Στο αρχείο p9lib.c κάντε τις παρακάτω προσθήκες (με γαλάζιο ο,τι έμεινε από πριν και με κόκκινο οι νέες γραμμές):

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

void sortMethod( void ) {

    int method;

    printf("\nPick a method: \n");
    printf("1 = bubble sort\n");
    printf("2 = straight insertion\n");
    scanf("%d", &method);

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

 ...
(υπόλοιπες προηγούμενες γραμμές)
 ...

void straight_insertion ( int *a, int n ) {
    int i, j, element;

    for(i = 1; i < n; i++) {
        element = a[i];
        j = i - 1;
        while((j >= 0) && (a[j] > element)) {
```

```
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = element;
    }
}
```

Εδώ απλά προσθέσαμε μία ακόμη επιλογή στη συνάρτηση sortMethod, για τη μέθοδο παρεμβολής, καθώς και τη συνάρτηση που υλοποιεί αυτή τη μέθοδο. Παρατηρείστε ότι δε χρειάζεται να κάνουμε καμία αλλαγή στο κύριο πρόγραμμα όταν προσθέτουμε τη νέα μέθοδο και αυτό θα ισχύει και παρακάτω με τις υπόλοιπες μεθόδους που θα προσθέσουμε. Σε αυτό μας βοηθά ιδιαίτερα η χρήση του δείκτη-σε-συνάρτηση sort.

Μεταγλωττίστε και εκτελέστε.

Άσκηση p9-3 ***Ταξινόμηση φλοιού (shell sort)***

Συνεχίζουμε με εντελώς ανάλογο τρόπο, προσθέτοντας τη συνάρτηση για τη μέθοδο ταξινόμησης φλοιού (shell sort).

Πρώτα τροποποιούμε το αρχείο επικεφαλίδας. Εδώ θα κάνουμε μόνο μία προσθήκη προς το τέλος, γι' αυτό και δεν το γράφουμε ολόκληρο (και πάλι, με κόκκινο τα νέα στοιχεία ή αλλαγές):

```
#ifndef P9LIB_H
#define P9LIB_H
......
void straight_insertion( int * , int );
void shell_sort(int * , int );


#endif
```

Απλά προσθέσαμε τη δήλωση της νέας συνάρτησης.

Μετά, τροποποιούμε και το αρχείο p9lib.c όπου προσθέτουμε την αντίστοιχη επιλογή στη sortMethod ενώ ορίζουμε και τη νέα μας συνάρτηση στο τέλος του αρχείου. κάτω από τα προηγούμενα περιεχόμενα. Και πάλι αφού πρόκειται απλώς για μια προσθήκη στο τέλος του αρχείου παραλείπουμε τις προηγούμενες γραμμές.

```
......
    printf("2 = straight insertion\n");
    printf("3 = shell sort\n");
......
        case 2:
            sort = straight_insertion;
            break;
        case 3:
            sort = shell_sort;
            break;
       default:
            printf("\nwrong option!");
            sort = do_nothing; /* to avoid segmentation fault */
    }
```

```
}
......
void shell_sort( int *a, int n ) {
    int i, j, distance, element;

    distance = n;
    do {
        distance /= 3;
        for(i = distance; i < n; i++) {
            element = a[i];
            j = i;
            while(a[j-distance] > element) {
                a[j] = a[j-distance];
                j -= distance;
                if (j < distance) break;
            }
            a[j] = element;
        }
    } while(distance > 0);
}
```

Μεταγλωττίστε και εκτελέστε.

Άσκηση p9-4 ***Ταξινόμηση σωρού (heap sort)***
Τώρα εισάγουμε την ταξινόμηση με τη μέθοδο του σωρού (heap sort). Αυτή χρειάζεται μια επιπλέον βοηθητική συνάρτηση η οποία φτιάχνει το "σωρό" των δεδομένων. Πρώτα δηλώνουμε αυτές τις δύο στο αρχείο επικεφαλίδας:

```
#ifndef P9LIB_H
#define P9LIB_H
......
void shell_sort(int * , int );
void heap_sort(int * , int );
void adjust(int * , int , int );

#endif
```

Στη συνέχεια τροποποιούμε το αρχείο p9lib.c προσθέτοντας τη νέα επιλογή μεθόδου και τον ορισμό των νέων συναρτήσεων στο τέλος, μετά από τις προηγούμενες:

```
......
    printf("3 = shell sort\n");
    printf("4 = heap sort\n");
......
        case 3:
            sort = shell_sort;
```

```
            break;
        case 4:
            sort = heap_sort;
            break;
        default:
.....
void heap_sort( int *a, int n ) {
    int i;

    for(i = n/2; i > 0; i--)
        adjust(a, i, n);
    for(i = n-1; i > 0; i--) {
        swap(a, a+i);
        adjust(a, 1, i);
    }
}

void adjust(int *a, int m, int n) {
    int i, j;

    i = m;
    j = 2 * m;
    while (j <= n ) {
        if( (j < n) && (a[j-1] < a[j]) )
            j++;
        if( a[i-1] < a[j-1] )
            swap(a+i-1, a+j-1);
        i = j;
        j *= 2;
    }
}
```

Μεταγλωττίστε και εκτελέστε.

Άσκηση p9-5 ***Ταχεία ταξινόμηση (quick sort)***

Συνεχίζουμε εισάγοντας και τη μέθοδο ταχείας ταξινόμησης (quick sort). Στο αρχείο επικεφαλίδας εισάγουμε την αντίστοιχη δήλωση ανάμεσα από την τελευταία προηγούμενη και το #endif:

```
    void quick_sort(int * , int );
```

Στο αρχείο p9lib.c κάνουμε τις αντίστοιχες προσθήκες για τη νέα επιλογή:

```
.....
    printf("5 = quick sort\n");
.....
        case 5:
            sort = quick_sort;
```

```
            break;
```
και στο τέλος του αρχείου προσθέτουμε τον ορισμό της συνάρτησης που υλοποιεί τη μέθοδο. Η συγκεκριμένη υλοποίηση είναι αναδρομική. Οι αναδρομικές κλήσεις της συνάρτησης υποδεικνύονται με έντονα γράμματα.

```
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);
}
```
Μεταγλωττίστε και εκτελέστε.

Άσκηση p9-6 ***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 τελευταία 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: const είναι μία δήλωση που λέει ότι κάποια μεταβλητή... δεν πρέπει να μεταβληθεί στη διάρκεια του προγράμματος! Αυτό μας προφυλάσσει από το να αλλάξουμε την τιμή της κατά λάθος σε κάποιο σημείο του προγράμματος ενώ δεν πρέπει. 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 εξής προσθήκες ανάμεσα από την τελευταία προηγούμενη δήλωση και το #endif:

```
void standard_qsort(int * , int );
int compare(const void * , const void * );
```

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

Στο αρχείο p9lib.c πρέπει πρώτα να προσθέσουμε το αρχείο <stdlib.h> για να μπορέσουμε να κάνουμε χρήση της qsort:

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

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

```
......
    printf("6 = standard library qsort\n");
......
        case 6:
            sort = standard_qsort;
            break;
......
```

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 p9-6 ***Επιδόσεις αλγόριθμων ταξινόμησης***

Αλλάζοντας τις τιμές του NDAT (π.χ. 1000000) και κάνοντας comment την εκτύπωση των στοιχείων των πινάκων, εκτελέστε το πρόγραμμα για διάφορες επιλογές και συγκρίνετε τις διάφορες μεθόδους ως προς την ταχύτητά τους.