11. Διεπαφές

Σύνοψη
Στο κεφάλαιο αυτό ο αναγνώστης εισάγεται στην έννοια της διασύνδεσης ή διεπαφής. Αρχικά, δίνονται η φιλοσοφία και τα γενικά χαρακτηριστικά των διεπαφών. Ακολούθως, παρουσιάζονται τα βήματα δημιουργίας μίας διεπαφής και τα κριτήρια για τον σχεδιασμό των διεπαφών. Τα θέματα της δημιουργίας διεπαφών και της διάσπασης του κώδικα σε πολλά αρχεία μελετώνται με μία σειρά τεσσάρων παραδειγμάτων, όπου γίνεται χρήση των εννοιών, των γλωσσικών κατασκευών και των εργαλείων που μελετήθηκαν στο παρόν και τα προηγούμενα κεφάλαια.

Λέξεις κλειδιά
διασύνδεση/διεπαφή, αρχείο κεφαλίδας, αρχείο βιβλιοθήκης, ifndef, απόκρυψη πληροφοριών.

Προαπαιτούμενη γνώση
Λεξιλόγιο της γλώσσας C – μεταβλητές – εκφράσεις – τελεστές – έλεγχος ροής προγράμματος – συναρτήσεις – πίνακες – δείκτες – δυναμική διαχείριση μνήμης – δομές δεδομένων – αρχεία

11.1 Η έννοια της διεπαφής

Στα προγράμματα που παρουσιάστηκαν στα προηγούμενα κεφάλαια, ο κώδικας φιλοξενείτο σε ένα μόνο αρχείο, π.χ. **`myProgram.c`**. Οι δηλώσεις συναρτήσεων και οι ορισμοί συναρτήσεων αποτελούσαν τμήματα του προγράμματος μέσα σε αυτόν τον ενιαίο χώρο. Από την άλλη πλευρά, ο κώδικας των πρότυπων συναρτήσεων βιβλιοθήκης (συναρτήσεις εισόδου–εξόδου, μαθηματικές συναρτήσεις κ.ο.κ.) δεν ήταν μεν ορατός από το πρόγραμμα, γινόταν όμως χρήση του μέσω των κλήσεων σε αυτές τις συναρτήσεις. Συνεπώς, έως τώρα στο ίδιο πρόγραμμα εμφανίζονταν δύο διαφορετικές προσεγγίσεις για την καταγραφή του κώδικα των συναρτήσεων: η άμεση προσέγγιση, όπου το σώμα της συνάρτησης βρισκόταν μέσα στο πρόγραμμα και η έμμεση – που αφορά στις συναρτήσεις βιβλιοθήκης – όπου δεν ενδιέφερε ο τρόπος υλοποίησης της συνάρτησης αλλά μόνο ο τρόπος χρήσης της και η κλήση της. Προφανώς, η δεύτερη προσέγγιση αφορούσε σε κώδικα που είχε γραφτεί σε διαφορετικό χρόνο και από διάφορους συγγραφείς, η χρησιμότητα του οποίου απαιτούσε την συμπερίληψή του χωρίς να υπάρχει ανάγκη ανάγνωσης και κατανόησής του.

Οι συναρτήσεις βιβλιοθήκης επικοινωνούν με ένα πρόγραμμα μέσω της οδηγίας προεπεξεργαστή **`#include`**, η οποία συνδέει στο πρόγραμμα αρχεία κεφαλίδας, όπως το αρχείο πρότυπης βιβλιοθήκης εισόδου–εξόδου **`stdio.h`**. Αυτά τα αρχεία αποτελούν ένα σύνορο μεταξύ δύο διακεκριμένων οντοτήτων: του αρχείου (βιβλιοθήκης), στο οποίο είναι γραμμένος ο κώδικας των συναρτήσεων, και του προγράμματος που χρησιμοποιεί τις συναρτήσεις. Επομένως, διασυνδέουν διαφορετικούς κώδικες, γι' αυτό και ονομάζονται **διασυνδέσεις** ή **διεπαφές** ή **διαπροσωπείες** (interfaces). Κάθε φορά που καλούνται συναρτήσεις αυτής της βιβλοθήκης, οι πληροφορίες περνούν διαμέσου αυτού του συνόρου. Η διεπαφή μεσολαβεί στην ανταλλαγή πληροφοριών μεταξύ της βιβλιοθήκης και του προγράμματος–χρήστη, το οποίο χρησιμοποιεί τις συναρτήσεις της.

Η ανάπτυξη και χρήση διεπαφών διευκολύνει τη συγγραφή κώδικα για τους ακόλουθους λόγους:

1. Επιτυγχάνεται επαναχρησιμοποίηση κώδικα, καθώς χρησιμοποιείται κώδικας που αναπτύχθηκε από τον ίδιο συγγραφέα ή από άλλους συγγραφείς, έχει αποσφαλματωθεί και είναι έτοιμος προς χρήση.

2. Ο προγραμματιστής δεν είναι υποχρεωμένος να έχει πρόσβαση στον κώδικα των συναρτήσεων που ορίζει μία διεπαφή. Απλώς χρειάζεται τις δηλώσεις των συναρτήσεων και επαρκή περιγραφή (υπό τη μορφή

σχολίων) του τις υλοποιούν, για να μπορεί να τις χρησιμοποιεί. Με αυτόν τον τρόπο αντιμετωπίζει τη συνάρτηση ως «μαύρο κουτί», όπου ο τρόπος υλοποίησης είναι αδιάφορος, και μπορεί να επικεντρωθεί στον πυρήνα του προγράμματός του. Η τακτική της απόκρυψης των εσωτερικών διεργασιών μίας βιβλιοθήκης ονομάζεται *απόκρυψη πληροφοριών* (information hiding).

3. Επιτυγχάνεται η οργάνωση και η ομαδοποίηση λειτουργιών σε συγκεκριμένα αρχεία, καθώς οι διεπαφές περιλαμβάνουν συναρτήσεις από συναφείς λειτουργίες. Για παράδειγμα, το αρχείο κεφαλίδας **`math.h`** αναφέρεται σε μαθηματικές συναρτήσεις και δεν περιλαμβάνει π.χ. συναρτήσεις διαχείρισης αλφαριθμητικών. Στις τελευταίες αναφέρεται το αρχείο κεφαλίδας **`string.h`**.

11.2 Δημιουργία διεπαφής

Η δημιουργία διεπαφής έγκειται στην υλοποίηση του αρχείου κεφαλίδας, π.χ. **`myLibray.h`**, και του αρχείου, στο οποίο θα βρίσκονται τα σώματα των συναρτήσεων, π.χ. **`myLibrary.c`**. Τα περιεχόμενα του αρχείου κεφαλίδας καθορίζονται από μία σειρά κανόνων:

1. Αρχικά παρατίθενται εκτενή σχόλια για τον σκοπό και τα περιεχόμενα της διεπαφής.

2. Μετά τα σχόλια περιέχονται οι γραμμές

```
#ifndef  _<όνομα διεπαφής>_h
#define  _<όνομα διεπαφής>_h
```

3. Η τελευταία γραμμή του αρχείου κεφαλίδας είναι

```
#endif
```

4. Ο κώδικας της διεπαφής θα περικλείεται ανάμεσα σε αυτές τις οδηγίες προεπεξεργαστή. Η υποθετική πρόταση που υλοποιούν ελέγχει κατά πόσον υπάρχει αρχείο κεφαλίδας με το ίδιο όνομα. Εάν δεν υπάρχει, τότε με την **`#define`** ορίζεται το όνομα και εκτελούνται οι επόμενες γραμμές του αρχείου κεφαλίδας. Πλέον, το συγκεκριμένο όνομα της διεπαφής έχει οριστεί και, εάν το ξανασυναντήσει ο μεταγλωττιστής, δεν θα χρειαστεί να ξαναδιαβάσει τη διεπαφή. Έτσι αποφεύγεται πολλαπλή ανάγνωση της ίδιας διεπαφής από τον μεταγλωττιστή. Η ανωτέρω τριάδα προτάσεων ονομάζεται **στερεότυπο** (boilerplate) και η παρουσία τους είναι υποχρεωτική σε κάθε διεπαφή.

5. Ο κώδικας της διασύνδεσης περιλαμβάνει τα **στοιχεία διεπαφής** (inteface entries). Αυτά είναι

(α) πρωτότυπα των συναρτήσεων (συνήθως συνοδευόμενα από σχόλια)

(β) (πιθανόν) σταθερές, π.χ. προσέγγιση του π ή η σταθερά του Planck,

(γ) (πιθανόν) τύποι δεδομένων οριζόμενοι από τον χρήστη, οι οποίοι χρησιμοποιούνται σε συναρτήσεις της βιβλιοθήκης.

Εάν οι συναρτήσεις αυτές χρειάζονται στοιχεία που βρίσκονται σε άλλες βιβλιοθήκες, π.χ. μία συνάρτηση χρειάζεται την **`fabs()`** που περιλαμβάνεται στο **`math.h`**, το σχετικό αρχείο κεφαλίδας πρέπει να συμπεριληφθεί μέσω της **`#include`**. Άπαξ και συμπεριληφθεί ένα αρχείο κεφαλίδας σε μία διεπαφή, δεν χρειάζεται να συμπεριληφθεί ξανά σε άλλο αρχείο που συναπαρτίζει το συνολικό πρόγραμμα.

Το αρχείο της βιβλιοθήκης των συναρτήσεων είναι συνώνυμο της διεπαφής και τη συμπεριλαμβάνει με την οδηγία προεπεξεργαστή **`#include`**, π.χ.

```
#include "myLibrary.h"
```

Σε αντιδιαστολή με τα πρότυπα αρχεία κεφαλίδας, όπου χρησιμοποιείται το ζεύγος **`< >`**, στις διασυνδέσεις που σχεδιάζουμε χρησιμοποιείται το ζεύγος **`" "`**. Η ίδια οδηγία προεπεξεργαστή εφαρμόζεται και στο κύριο πρόγραμμα, προκειμένου να συνδεθεί η βιβλιοθήκη με αυτό.

Η τελική μορφή μίας διεπαφής και του αρχείου βιβλιοθήκης που τη συνοδεύει είναι η ακόλουθη:

(i) **Αρχείο myLibrary.h**

```
/*  Σχόλια για τον σκοπό και τα περιεχόμενα της διεπαφής.  */
#ifndef  _myLibrary_h
#define  _ myLibrary _h
```

```
#include "mySecondLibrary.h"/* Άλλη βιβλιοθήκη, στοιχείο της οποίας
χρησιμοποιείται στην παρούσα βιβλιοθήκη  */
#include <πρότυπη βιλιοθήκη.h>   /* π.χ. string.h, στοιχείο της
οποίας χρησιμοποιείται στην παρούσα βιβλιοθήκη*/

/* Εάν απαιτείται: Δήλωση τύπων δεδομένων (π.χ. δομή)  */
#ifndef _dataTypeT_
#define _dataTypeT_
  typedef struct dataTypeT
  {
     δήλωση μεταβλητών-μελών
  };
#endif

/* Δήλωση συναρτήσεων */
<τύπος επιστρεφόμενης τιμής> firstFuncName(λίστα ορισμάτων);
<τύπος επιστρεφόμενης τιμής> secondFuncName (λίστα ορισμάτων);
......
<τύπος επιστρεφόμενης τιμής> lastFuncName(λίστα ορισμάτων);

#endif
```

(ii) **Αρχείο myLibrary.c**

```
#include "myLibrary.h"

/*   Σχόλια για τη λειτουργία της ακόλουθης συνάρτησης    */
<τύπος επιστρεφόμενης τιμής> firstFuncName(λίστα ορισμάτων);

/*   Σχόλια για τη λειτουργία της ακόλουθης συνάρτησης    */
<τύπος επιστρεφόμενης τιμής> secondFuncName (λίστα ορισμάτων);
......

/*   Σχόλια για τη λειτουργία της ακόλουθης συνάρτησης    */
<τύπος επιστρεφόμενης τιμής> lastFuncName(λίστα ορισμάτων);
```

Τα αρχεία βιβλιοθήκης που δημιουργούνται με τις διασυνδέσεις, πρέπει να μεταγλωττιστούν, όπως ακριβώς μεταγλωττίζεται το κύριο πρόγραμμα. Στα ολοκληρωμένα περιβάλλοντα ανάπτυξης προγράμματος προστίθενται στα *σχέδια προγράμματος* (Projects) και μεταγλωττίζονται όλα μαζί. Τα αρχεία κεφαλίδας δεν χρειάζονται μεταγλώττιση. Έτσι, παράγεται το αντικείμενο αρχείο (με κατάληξη **.obj** για συστήματα Windows και **.o** για συστήματα Unix) για κάθε πηγαίο αρχείο **.c**. Τα αντικείμενα αρχεία συνδέονται στον συνδέτη μαζί με τα πρότυπες βιβλιοθήκες και παράγεται το τελικό εκτελέσιμο αρχείο.

Θα πρέπει να προσεχθεί, ώστε να μην δημιουργείται κυκλική συμπερίληψη σε διασυνδέσεις. Δηλαδή, εάν μία διεπαφή **myLibrary.h** συμπεριλαμβάνει μία δεύτερη διεπαφή με την πρόταση **#include "mySecondLibrary.h"**, δεν είναι δυνατόν η **mySecondLibrary.h** να συμπεριλαμβάνει τη διεπαφή **myLibrary.h**.

11.3 Ιδιότητες διεπαφής

Όπως αναφέρθηκε προηγουμένως, οι διασυνδέσεις ελαττώνουν την πολυπλοκότητα του προγραμματισμού, κατ' αντιστοιχία με τις συναρτήσεις αλλά σε ανώτερο επίπεδο λεπτομέρειας: μία συνάρτηση παρέχει σε αυτόν που την καλεί την πρόσβαση σε ένα σύνολο βημάτων, τα οποία υλοποιούν μία συγκεκριμένη λειτουργία. Μία διεπαφή παρέχει στο κύριο πρόγραμμα (ή σε άλλα αρχεία βιλιοθήκης που τη χρησιμοποιούν)

πρόσβαση σε ένα σύνολο συναρτήσεων, οι οποίες υλοποιούν μία σειρά συναφών λειτουργιών. Ωστόσο, ο βαθμός απλοποίησης του προγραμματισμού σχετίζεται με το πόσο καλά σχεδιασμένη είναι η διεπαφή.

Για να σχεδιαστεί μία αποτελεσματική διεπαφή, πρέπει να ληφθούν υπόψη κάποια κριτήρια, τα οποία ορισμένες φορές έχουν μεταξύ τους ανταγωνιστική φύση. Οι διασυνδέσεις πρέπει εν γένει:

***(α)* Να έχουν ένα ενοποιητικό θέμα.** Οι συναρτήσεις που θα επιλεγούν να συμμετάσχουν σε μία διεπαφή, θα πρέπει να έχουν ένα συνεκτικό θέμα. Η βιβλιοθήκη **math** περιέχει αποκλειστικά μαθηματικές συναρτήσεις και κάθε συνάρτηση που εξάγεται από αυτή τη διεπαφή ταιριάζει με τον σκοπό της διεπαφής.

Επιπλέον, οι συναρτήσεις μίας διεπαφής θα πρέπει να συμπεριφέρονται με κατά το δυνατόν συνεπέστερο τρόπο. Για παράδειγμα, οι τριγωνομετρικές συναρτήσεις θα πρέπει να έχουν το ίδιο όρισμα σε ό,τι αφορά τη γωνία. Δεν πρέπει οι μισές να δέχονται μοίρες και οι μισές ακτίνια, έτσι ώστε ο προγραμματιστής που τις χρησιμοποιεί να γνωρίζει ότι η είσοδος είναι κοινή σε συναρτήσεις της ίδιας οικογένειας.

***(β)* Να είναι απλές**. Η απλότητα αποτελεί απαραίτητη προϋπόθεση για την ελάττωση της πολυπλοκότητας του προγραμματισμού, την οποία οι διασυνδέσεις στοχεύουν να προσφέρουν. Προς αυτή την κατεύθυνση κινείται η αρχή της απόκρυψης πληροφοριών, που αναφέρθηκε προηγουμένως. Ο προγραμματιστής–χρήστης πρέπει να δέχεται την κατάλληλη ποσότητα πληροφορίας. Ορισμένες φορές η αξία μίας διεπαφής δεν αναδεικνύεται από την πληροφορία που παρέχει, αλλά από αυτήν που αποκρύβει.

Η αρχή της απόκρυψης πληροφοριών έχει συνέπειες στη συγγραφή των διασυνδέσεων, καθώς δεν θα πρέπει να αποκαλύπτονται λεπτομέρειες της διεπαφής ούτε ακόμη και στα ερμηνευτικά σχόλια. Η διεπαφή έχει σκοπό να διευκολύνει τον προγραμματιστή που θα τη χρησιμοποιήσει και θα πρέπει να περιέχει μόνο όσα αυτός χρειάζεται να γνωρίζει.

Παρόμοια, οι συναρτήσεις μίας βιβλιοθήκης θα πρέπει να είναι σχεδιασμένες με απλό τρόπο. Εάν είναι δυνατόν να μειωθεί η λίστα των ορισμάτων ή να να απαλειφθούν περιπτώσεις που προκαλούν σύγχυση, θα είναι ευκολότερο για τον προγραμματιστή–χρήστη να κατανοήσει πώς θα χρησιμοποιήσει τις συναρτήσεις. Επιπρόσθετα, αποτελεί συνήθως καλή πρακτική να περιορίζεται ο συνολικός αριθμός των συναρτήσεων που περιλαμβάνονται στη διεπαφή, έτσι ώστε η τελευταία να μην καταντά χαώδης και να μην χάνεται ο προγραμματιστής–χρήστης μέσα σε πλήθος συναρτήσεων, μην μπορώντας να αποκτήσει αίσθηση του συνόλου.

(γ) **Να είναι επαρκείς**. Η διεπαφή θα πρέπει να παρέχει επαρκή λειτουγικότητα, ώστε να ικανοποιεί τις ανάγκες εκείνων που θα τη χρησιμοποιήσουν. Εάν μία λειτουργία απουσιάζει από μία διεπαφή, τότε – αργά ή γρήγορα – η τελευταία θα εγκαταληφθεί.

Είναι προφανές ότι η επάρκεια μπορεί να προσκρούει στην απλότητα, που αναφέρθηκε παραπάνω. Επομένως θα πρέπει να γίνει ένας συμβιβασμός ανάμεσα σε αυτά τα δύο χαρακτηριστικά, συνήθως ανάλογα με τα ιδιαίτερα χαρακτηριστικά της οικογένειας των συναρτήσεων που φιλοξενούνται στην εκάστοτε διεπαφή. Για παράδειγμα, εάν μία διεπαφή περιλαμβάνει συναρτήσεις για ένα σύστημα ελέγχου εναέριας κυκλοφορίας, η απλότητα έρχεται σε δεύτερη μοίρα σε σχέση με την ταχύτητα απόκρισης της συνάρτησης και η επάρκεια σε αυτήν την περίπτωση ταυτίζεται με την ταχύτητα στην επιστροφή σωστών απαντήσεων.

(δ) **Να είναι γενικές**. Μία καλοσχεδιασμένη διεπαφή θα πρέπει να είναι αρκετά ευέλικτη, ώστε να ικανοποιεί τις ανάγκες διαφορετικών κατηγοριών προγραμματιστών–χρηστών. Για να επιτευχθεί αυτό, η διεπαφή θα πρέπει να είναι αρκετά γενική, ώστε να λύνει ένα ευρύ φάσμα προβλημάτων.

(ε) **Να είναι σταθερές**. Οι συναρτήσεις που ορίζονται σε μία διεπαφή, θα πρέπει να εξακολουθούν να έχουν την ίδια ακριβώς μακροσκοπική εμφάνιση (είσοδοι – έξοδοι), ακόμη κι αν μεταβληθεί η υποκείμενη υλοποίηση. Εάν η μεταβολή της υλοποίησης οδηγεί σε αλλαγές στη συμπεριφορά μίας διεπαφής, οι προγραμματιστές–χρήστες θα πρέπει να αλλάξουν τα προγράμματά τους και αυτό θα θέσει σε κίνδυνο την αξία της διεπαφής. Η συνάρτηση **`sqrt(x)`**, που βρίσκεται στη διεπαφή **`math.h`**, υπολογίζει την τετραγωνική ρίζα του ορίσματός της, **`x`**. Η μαθηματική μέθοδος που υλοποιείται μέσα στη βιβλιοθήκη, είναι ένα ανάπτυγμα (π.χ. σειρά Taylor), το οποίο δεν είναι ορατό και – σε τελική ανάλυση – δεν ενδιαφέρει, εφόσον παρέχει το αποτέλεσμα με αποδεκτή ακρίβεια. Εάν μεταβληθεί ο τρόπος υπολογισμού, αλλά η συνάρτηση δατηρήσει τη μορφή της (ένα μόνο όρισμα και ίδιος τύπος επιστρεφόμενης τιμής), η διεπαφή παραμένει σταθερή. Εάν, όμως, η μεταβολή στον τρόπο υπολογισμού οδηγήσει σε μετονομασία της συνάρτησης ή μεταβολή του αριθμού ή/και του είδους των ορισμάτων της, τότε προκαλείται αναστάτωση, καθώς τα προγράμματα που τη χρησιμοποιούν πρέπει να τροποποιηθούν ανάλογα. Συμπερασματικά, οι

αλλαγές στις διασυνδέσεις θα πρέπει να επιχειρούνται σπάνια και μόνο με την ενεργό συμμετοχή του κοινού στο οποίο απευθύνονται.

Θα πρέπει να σημειωθεί ότι η προσθήκη μίας νέας συνάρτησης σε μία διεπαφή δεν διασαλεύει τη σταθερότητά της, καθώς πρόκειται για κάτι καινούριο, που δεν επηρεάζει τα υπάρχοντα προγράμματα που χρησιμοποιούν τη διεπαφή. Αυτή η διαδικασία ονομάζεται **επέκταση** (extending) της διεπαφής. Επομένως, εάν διαπιστωθεί ότι πρέπει να γίνουν εξελικτικές αλλαγές κατά τη διάρκεια της ζωής μίας διεπαφής, αυτές θα πρέπει να έχουν τη μορφή της επέκτασης. Υπό αυτή την έννοια, η δημιουργία μίας νέας συνάρτησης υπολογισμού της τετραγωνικής ρίζας, π.χ. `squareRoot(ορίσματα)`, με ταυτόχρονη διατήρηση της κλασσικής `sqrt(x)` είναι προτιμητέα από την αντικατάσταση της `sqrt(x)`.

11.4 Διάσπαση κώδικα σε πολλά αρχεία

Στην ενότητα αυτή θα μελετηθούν τα ζητήματα της δημιουργίας διασυνδέσεων και της διάσπασης του κώδικα σε πολλά αρχεία με τη βοήθεια παραδειγμάτων. Παραδείγματα που αναπτύχθηκαν στα προηγούμενα κεφάλαια, θα αποτελέσουν τη βάση για την παραγωγή δομημένου και επαναχρησιμοποιήσιμου κώδικα.

11.4.1 Παράδειγμα διαχείρισης αρχείων

Στην ενότητα 9.9 παρουσιάστηκε ένα πρόγραμμα διαχείρισης αρχείων, στο οποίο το έργο είχε κατανεμηθεί σε εννέα συναρτήσεις. Οι συναρτήσεις αυτές, οι οποίες είναι όλες συναφείς, μπορούν να ενταχθούν σε μία βιβλιοθήκη διαχείρισης αρχείων **fileHandling**. Η διεπαφή θα περιέχει τον ορισμό της δομής `RecordT`, καθώς αυτός είναι απαραίτητος στη λειτουργία των συναρτήσεων, τη σταθερά μέγιστο μήκος ονόματος `MAXLENGTH`, καθώς και μερικά εκ των αρχείων κεφαλίδας. Τα υπόλοιπα αρχεία κεφαλίδας θα περιλαμβάνονται στο κύριο πρόγραμμα. Ο συνολικός κώδικας περιλαμβάνει το κύριο πρόγραμμα **mainProgram.c**, το αρχείο κεφαλίδας **fileHandling.h** και το αρχείο βιβλιοθήκης **fileHandling.c**. Η διάρθρωσή του είναι η ακόλουθη:

(i) **mainProgram.c**

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

#include "fileHandling.h"

const char *dirpath = "C:\\temp\\"; /* διαδρομή του καταλόγου στον
                                       οποίο βρίσκεται το αρχείο */
const char *file = "data.dat";      /* όνομα αρχείου */
/*-----------------------------------*/
int main(void)
{
  char filename[strlen(dirpath)+strlen(file)+1];
  strcpy(filename,dirpath);
  strcat(filename,file);
  /* Επιλογή λειτουργίας */
  char answer='q';
  while(true)
  {
    printf("\nChoose one of the following options:"
    "\nTo list the file contents enter L"
    "\nTo create a new file enter C"
    "\nTo add new records enter A"
    "\nTo update existing records enter U"
```

```
            "\nTo delete the file enter D"
            "\nTo end the program enter Q\n : ");
            scanf("\n%c", &answer);
            switch(tolower(answer))
            {
               case 'l':
                  listFile(filename);
               break;
               case 'c':
                  writeFile(filename,"wb+");
                  printf("\nFile creation complete.");
               break;
               case 'a':
                  writeFile(filename, "ab+");
                  printf("\nFile append complete.");
               break;
               case 'u':
                  updateFile(filename);
               break;
               case 'd':
                  printf("Are you sure you want to delete %s (y or n)? ",
filename);
                  scanf("\n%c", &answer);
                  if(tolower(answer)=='y')
                     remove(filename);
                     /* η συνάρτηση remove διαγράφει ένα αρχείο */
               break;
               case 'q': /* έξοδος από το πρόγραμμα */
                  printf("\nEnding the program.", filename);
                  return 0;
               default:
                  printf("Invalid selection. Try again.");
               break;
            }
         }

         return 0;
      }
```

(ii) **fileHandling.h**

```
      #ifndef _fileHandling_
      #define _fileHandling_

         #include <stdio.h>
         #include <assert.h>
         #include <stdbool.h>

         #define MAXLENGTH 80

         #ifndef _RecordT_
         #define _RecordT_
            struct RecordT
            {
               char name[MAXLENGTH];
```

```
        int age;
    };
#endif
/*-------------------------------------*/
void listFile(char *filename); /* Εμφανίζει τα περιεχόμενα του
αρχείου στην οθόνη  */

void updateFile(char *filename); /* Τροποποιεί τα περιεχόμενα του
αρχείου */

struct RecordT *getRecord(struct RecordT *precord);/* Λαμβάνει
δεδομένα (εγγραφές) από το πληκτρολόγιο  */

void getName(char *pname);  /* Διαβάζει  ένα  όνομα  από  το
πληκτρολόγιο */

void writeFile(char *filename, char *mode);        /* Έχει διπλή
λειτουργία: την εγγραφή σε ένα νέο αρχείο και την προσάρτηση σε
ένα υφιστάμενο αρχείο  */

void writeRecord(struct RecordT *precord, FILE *pFile); /* Γράφει
δεδομένα (εγγραφές) στο αρχείο */

struct RecordT *readRecord(struct RecordT *precord, FILE *pFile);
/* Λαμβάνει δεδομένα (εγγραφές) από το αρχείο  */

int findRecord(struct RecordT *precord, FILE *pFile); /* Αναζητά
μία εγγραφή στο αρχείο, βάσει μίας μεταβλητής-μέλος της εγγραφής*/

void duplicateFile(struct RecordT *pnewrecord, int index, char
*filename, FILE *pFile);  /* Αναπαράγει το αρχείο αντικαθιστώντας
μία  εγγραφή,  όταν  αυτή  έχει  διαφορετικό  μήκος  από  την
αντικαθιστώμενη */

#endif
```

***(iii)*fileHandling.c**

```
#include "fileHandling.h"

void listFile(char *filename)
{
......
}

void updateFile(char *filename)
{
......
}

struct RecordT *getRecord(struct RecordT *precord);
{
......
}

void getName(char *pname)
```

```
{
......
}

void writeFile(char *filename, char *mode)
{
......
}

void writeRecord(struct RecordT *precord, FILE *pFile)
{
......
}

struct RecordT *readRecord(struct RecordT *precord, FILE *pFile)
{
......
}

int findRecord(struct RecordT *precord, FILE *pFile)
{
......
}

void duplicateFile(struct RecordT *pnewrecord, int index, char
*filename, FILE *pFile)
{
......
}
```

Τα σώματα των συναρτήσεων, με σχετικό σχολιασμό παρατίθενται στην ενότητα 9.9.

Αποτίμηση της διεπαφής

Η διεπαφή είναι ενοποιημένη, καθώς όλες οι συναρτήσεις εντάσσονται στο ενιαίο θέμα της διαχείρισης στοιχείων του αρχείου ή ολόκληρου του αρχείου. Επιπρόσθετα, η διεπαφή μπορεί να θεωρηθεί ότι παρέχει κάποιο μέτρο απλούστευσης, καθώς αποκρύβει από τον προγραμματιστή–χρήστη ένα μεγάλο μέρος της πολυπλοκότητας των συναρτήσεων.

Σε ό,τι αφορά την επάρκεια και τη γενικότητά της, εφόσον οι συναρτήσεις σχετίζονται με τη δομή της εγγραφής ,η γενικότητα αναπόφευκτα περιορίζεται. Το πλήθος των λειτουργιών μπορεί να θεωρηθεί ικανοποιητικό για τη διαχείριση, επομένως η διεπαφή διαθέτει επάρκεια, χωρίς αυτό να σημαίνει ότι ίσως να απαιτείται κάποια επιπέον σχεδίαση, προκειμένου να ικανοποιηθούν πιο εξειδικευμένες λειτουργίες.

Τέλος, το ζήτημα της σταθερότητας δεν είναι ένα ερώτημα που αφορά τη φάση της σχεδίασης, αλλά μάλλον τον κύκλο της μακροπρόθεσμης συντήρησής της. Το σημαντικό ερώτημα είναι κατά πόσον η σχεδίαση προάγει με κάποιον τρόπο τη μακροπρόθεσμη σταθερότητα. Γενικά, μία διεπαφή που ικανοποιεί τα υπόλοιπα κριτήρια μπορεί, κατά πάσα πιθανότητα, να παραμείνει σταθερή, χρησιμοποιώντας στο μέλλον και την τεχνική της επέκτασης.

11.4.2 Παράδειγμα δυναμικής διαχείριση μνήμης

Στην ενότητα **7.7** και στα παραδείγματα **7.7.1** και **7.7.2** παρουσιάστηκαν συναρτήσεις για τη δυναμική δέσμευση και αποδέσμευση μνήμης. Αυτές οι συναρτήσεις, οι οποίες επιστρέφουν δείκτη στους προκαθορισμένους τύπους δεδομένων, μπορούν να ενσωματωθούν σε ένα αρχείο βιβλιοθήκης **allocateMemory**. Για λόγους οικονομίας του χώρου η δέσμευση θα αφορά σε δισδιάστατους δυναμικούς πίνακες, ωστόσο

μπορεί να επεκταθεί σε περισσότερες διαστάσεις. Ακολούθως, παρατίθενται το αρχείο κεφαλίδας **allocateMemory.h** και το αρχείο βιβλιοθήκης **allocateMemory.c**.

(i) **allocateMemory.h**

```
#ifndef _allocateMemory_
#define _allocateMemory_

  #include <assert.h>
  #include <stdlib.h>
  /*-----------------------------------*/
  char **allocChar_2(int lineNumber, int columnNumber); /* Δέσμευση
  μνήμης για δισδιάστατο πίνακα χαρακτήρων */

  int **allocInt_2(int lineNumber, int columnNumber);   /* Δέσμευση
  μνήμης για δισδιάστατο πίνακα ακεραίων */

  double **allocDouble_2(int lineNumber, int columnNumber);
  /*  Δέσμευση μνήμης για δισδιάστατο πίνακα αριθμών κινητής
  υποδιαστολής διπλής ακρίβειας  */

  float **allocFloat_2(int lineNumber, int columnNumber);
  /*  Δέσμευση μνήμης για δισδιάστατο πίνακα αριθμών κινητής
  υποδιαστολής απλής ακρίβειας*/

  void freeChar_2(char **parr, int lineNumber); /* Αποδέσμευση
  μνήμης για δισδιάστατο πίνακα χαρακτήρων */

  void freeInt_2(int **parr, int lineNumber); /* Αποδέσμευση  μνήμης
  για δισδιάστατο πίνακα ακεραίων  */

  void freeDouble_2(double **parr, int lineNumber);  /* Αποδέσμευση
  μνήμης για δισδιάστατο πίνακα αριθμών κινητής υποδιαστολής διπλής
  ακρίβειας */

  void freeFloat_2(float **parr, int lineNumber);  /* Αποδέσμευση
  μνήμης για δισδιάστατο πίνακα αριθμών κινητής υποδιαστολής απλής
  ακρίβειας */

#endif
```

(ii) **allocateMemory.c**

```
#include "allocateMemory.h"
/*-----------------------------------*/
char **allocChar_2(int lineNumber, int columnNumber)
{
  int i;
  char **parr;
  parr=(char **)malloc(lineNumber*sizeof(char *));
  assert(parr!=NULL);
  for (i=0;i<lineNumber;i++)
  {
    parr[i]=(char *)malloc(columnNumber*sizeof(char));
    assert(parr[i]!=NULL);
  }
```

```
  return(parr);
}
/*-----------------------------------*/
int **allocInt_2(int lineNumber, int columnNumber)
{
  int i,**parr;
  parr=(int **)malloc(lineNumber*sizeof(int *));
  assert(parr!=NULL);
  for (i=0;i<lineNumber;i++)
  {
    parr[i]=(int *)malloc(columnNumber*sizeof(int));
    assert(parr[i]!=NULL);
  }
  return(parr);
}
/*-----------------------------------*/
double **allocDouble_2(int lineNumber, int columnNumber)
{
  int i;
  double **parr;
  parr=(double **)malloc(lineNumber*sizeof(double *));
  assert(parr!=NULL);
  for (i=0;i<lineNumber;i++)
  {
    parr[i]=(double *)malloc(columnNumber*sizeof(double));
    assert(parr[i]!=NULL);
  }
  return(parr);
}
/*-----------------------------------*/
float **allocFloat_2(int lineNumber, int columnNumber)
{
  int i;
  float **parr;
  parr=(float **)malloc(lineNumber*sizeof(float *));
  assert(parr!=NULL);
  for (i=0;i<lineNumber;i++)
  {
    parr[i]=(float *)malloc(columnNumber*sizeof(float));
    assert(parr[i]!=NULL);
  }
  return(parr);
}
/*-----------------------------------*/
void freeChar_2(char **parr, int lineNumber)
{
  int i;
  for (i=(lineNumber-1);i>=0;i--)
     free(parr[i]);
  free(parr);
}
/*-----------------------------------*/
void freeInt_2(int **parr, int lineNumber)
{
  int i;
```

```
  for (i=(lineNumber-1);i>=0;i--)
     free(parr[i]);
  free(parr);
}
/*-------------------------------------*/
void freeDouble_2(double **parr, int lineNumber)
{
  int i;
  for (i=(lineNumber-1);i>=0;i--)
     free(parr[i]);
  free(parr);
}
/*-------------------------------------*/
void freeFloat_2(float **parr, int lineNumber)
{
  int i;
  for (i=(lineNumber-1);i>=0;i--)
     free(parr[i]);
  free(parr);
}
```

11.4.3 Παράδειγμα διαχείρισης απλά συνδεδεμένης λίστας

Στην υποενότητα **10.4.1** παρουσιάστηκαν συναρτήσεις για τη διαχείριση απλά συνδεδεμένης λίστας. Αυτές οι συναρτήσεις μπορούν να ενσωματωθούν σε ένα αρχείο βιβλιοθήκης **linkedListProcessing**. Η διεπαφή θα περιέχει τον ορισμό της δομής του κόμβου **node**, καθώς αυτός είναι απαραίτητος στη λειτουργία των συναρτήσεων, καθώς και μερικά εκ των αρχείων κεφαλίδας. Ακολούθως, παρατίθενται το αρχείο κεφαλίδας **linkedListProcessing.h** και το αρχείο βιβλιοθήκης **linkedListProcessing.c**.

(i) **linkedListProcessing.h**

```
#ifndef _linkedListProcessing_
#define _linkedListProcessing_

  #include <assert.h>
  #include <stdlib.h>

  #ifndef _node_
  #define _node_
     struct node
     {
        int data;
        struct node *next;
     };
     typedef struct node *PTR;
  #endif
  /*-------------------------------------*/
  PTR createList(PTR head); /* Δημιουργία λίστας με την προσθήκη
  ενός ή περισσότερων κόμβων  */

       PTR insertToList(PTR head, int x);   /* Εισαγωγή στοιχείου σε
   διατεταγμένη λίστα (οι κόμβοι είναι τοποθετημένοι κατά αύξουσα τιμή των
                              δεδομένων τους)  */

  PTR deleteFromList(PTR head, int x);   /* Διαγραφή στοιχείου   */
```

```
    void printList(PTR head);    /* Εκτύπωση των περιεχομένων της
    λίστας     */

#endif
```

(ii) **linkedListProcessing.c**

```
#include "linkedListProcessing.h"
/*-----------------------------------*/
PTR createList(PTR head)
{
......
}
PTR insertToList(PTR head, int x)
{
......
}
PTR deleteFromList(PTR head, int x)
{
......
}
void printList(PTR head)
{
......
}
```

Τα σώματα των συναρτήσεων, με σχετικό σχολιασμό παρατίθενται στην υποενότητα 10.4.1.

11.4.4 Παράδειγμα επεξεργασίας πινάκων

Στην ενότητα **7.8** αναπτύχθηκε πρόγραμμα επεξεργασίας τετραγωνικών πινάκων. Εάν οι συναρτήσεις ανάγνωσης (**getData()**) και εκτύπωσης (**`printData()`**) πίνακα θεωρηθεί ότι εμπίπτουν στις λειτουργίες διαχείρισης, τότε όλες οι συναρτήσεις της ενότητας **7.8** – με εξαίρεση τη συνάρτηση **`getSize()`** – ικανοποιούν το κριτήριο του ενοποιητικού θέματος. Εάν, όμως, επιχειρούσαμε να γράψουμε μία διεπαφή, η οποία θα επεξεργαζόταν μαθηματικά τον πίνακα, τότε μόνον οι συναρτήσεις υπολογισμού του ίχνους και αντιμετάθεσης γραμμών/στηλών θα εντάσσονταν σε μία τέτοια διεπαφή. Σε αυτήν θα είχαν θέση και άλλες λειτουργίες, όπως η αναστροφή και η αντιστροφή πίνακα, ο υπολογισμός της ορίζουσας και των ιδιοτιμών του κ.ά.

Η διεπαφή **sqMatrixProcessing** θα ενσωματώσει την πρότυπη βιβλιοθήκη μαθηματικών συναρτήσεων (αρχείο κεφαλίδας **math.h**) για τη χρήση της συνάρτησης **`fabs()`**. Τα υπόλοιπα αρχεία κεφαλίδας θα περιληφθούν στο κύριο πρόγραμμα, καθώς και στη διεπαφή **allocateMemory** (αναπτύχθηκε στην **11.4.2** για τη δυναμική δέσμευση και αποδέσμευση δισδιάστατου δυναμικού πίνακα), ένεκα του γεγονότος ότι το κύριο πρόγραμμα θα ζητά συναρτήσεις από την **allocateMemory**. Ο συνολικός κώδικας περιλαμβάνει το κύριο πρόγραμμα **mainProgram.c**, το αρχείο κεφαλίδας **sqMatrixProcessing.h** και το αρχείο βιβλιοθήκης **sqMatrixProcessing.c**. Η διάρθρωσή του είναι η ακόλουθη και τα αποτελέσματά του απεικονίζονται στην **Εικόνα 7.7**:

(i) **mainProgram.c**

```
#include <stdio.h>

#include "allocateMemory.h"
#include "sqMatrixProcessing.h"
```

```
int getSize(void);
/*-----------------------------------*/
int main()
{
  int i,j,col,size;
  float **pArr;

  size=getSize();
  pArr=allocFloat_2(size,size);

  getData(pArr,size);
  printf("\n\nInitial array A:");
  printData(pArr,size);

  for (i=0;i<size;i++)
     getMax(pArr,size,i);
  printf("\n\nTrace(A)=%f\n",trace(pArr,size));
  permuteColumns(pArr,size,1,2);
  permuteLines(pArr,size,0,2);

  printf("\n\nFinal array:");
  printData(pArr,size);

  freeFloat_2(pArr,size);

  return 0;
}
/*-----------------------------------*/
int getSize(void)
{
  int size;
  do
  {
    printf(" Give the size of the array (>=3):  " );
    scanf("%d",&size);
  } while (size<3);

  return size;
}
```

(ii) **sqMatrixProcessing.h**

```
#ifndef _sqMatrixProcessing_
#define _sqMatrixProcessing_

  #include <math.h>
  /*-----------------------------------*/
  void getData(float **ptr, int size);   /* Ανάγνωση  των  στοιχείων
  του πίνακα*/

  void printData(float **ptr, int size); /* Εκτύπωση των πίνακα  */

  void getMax(float **pArray, int size, int i);
```

```
  /* Εύρεση του στοιχείου της i γραμμής με τη μέγιστη απόλυτη τιμή
  και εμφάνισή της στην οθόνη */

  float trace(float **pArray, int size); /* Υπολογισμός του ίχνους
  του πίνακα*/

  void permuteColumns(float **pArray, int size, int column1, int
  column2); /* Αντιμετάθεση των στηλών column1 και column2 */


  void permuteLines(float **pArray, int size, int line1, int line2);
  /* Αντιμετάθεση των γραμμών line1 και line2 */

#endif
```

***(iii)*sqMatrixProcessing.c**

```
#include "sqMatrixProcessing.h"
/*-------------------------------------*/
void getData(float **ptr, int size)
{
  int i,j;
  for (i=0;i<size;i++)
     for (j=0;j<size;j++)
     {
        printf("\nA[%d][%d]:  ",i+1,j+1);
        scanf("%f",&ptr[i][j]);
     }
}
/*-------------------------------------*/
void printData(float **ptr, int size)
{
  int i,j;
  for (i=0;i<size;i++)
  {
     printf("\n");
     for (j=0;j<size;j++)  printf("\t%10.4f",ptr[i][j]);
  }
}
/*-------------------------------------*/
void getMax(float **pArray, int size, int i)
{
  int j,col;
  float maxim;
  maxim=fabs(pArray[i][0]);
  col=0;
  for (j=1;j<size;j++)
     if (fabs(pArray[i][j])>maxim)
     {
        maxim=fabs(pArray[i][j]);
        col=j;
     }
  printf( "\nLine %d: column %d, size=%f",i+1,col+1,fabs(pArray[i]
[col]) );
}
```

```
/*-------------------------------------*/
float trace(float **pArray, int size)
{
  int i;
  float trc=0.0;
  for (i=0;i<size;i++)
     trc=trc+pArray[i][i];

  return trc;
}
/*-------------------------------------*/
void permuteColumns(float **pArray, int size, int column1, int col-
umn2)
{
  int j;
  float temp;
  for (j=0;j<size;j++)
  {
    temp=pArray[j][column1];
    pArray[j][column1]=pArray[j][column2];
    pArray[j][column2]=temp;
  }
}
/*-------------------------------------*/
void permuteLines(float **pArray, int size, int line1, int line2)
{
  int j;
  float temp;
  for (j=0;j<size;j++)
  {
    temp=pArray[line1][j];
    pArray[line1][j]=pArray[line2][j];
    pArray[line2][j]=temp;
  }
}
```

Ασκήσεις

Άσκηση 1

Με βάση τις υλοποιήσεις των συναρτήσεων διαχείρισης αλφαριθμητικών που δόθηκαν στις ενότητες 5.9 και 6.6, να γραφεί διεπαφή, που θα περιέχει τις συναρτήσεις αυτές.

Άσκηση 2

Με βάση τον κώδικα δέσμευσης και απελευθέρωσης μνήμης τρισδιάστατων δυναμικών πινάκων, που περιέχεται στο παράδειγμα 7.6.1, και τις συναρτήσεις δέσμευσης–αποδέσμευσης μνήμης του παραδείγματος 11.4.3, να *επεκταθεί* η διεπαφή allocateMemory (αρχεία allocateMemory.h και allocateMemory.c) ώστε να περιλαμβάνει τρισδιάστατους πίνακες.

Άσκηση 3

Λαμβάνοντας υπόψη:
***(α)* την υλοποίηση της ουράς με συνδεδεμένη λίστα στην υποενότητα 10.5.2,**
***(β)* την εκτύπωση των περιεχομένων συνδεδεμένης λίστας στην υποενότητα 10.4.1,**
***(γ)* τη διεπαφή allocateMemory της υποενότητας 11.4.3,**

να γραφεί πρόγραμμα για το πρόβλημα του παραδείγματος 10.3.1, στο οποίο θα δημιουργηθεί διεπαφή queueProcessing (αρχεία queueProcessing.h και queueProcessing.c) για τη φιλοξενία συναρτήσεων διαχείρισης ουράς.

Άσκηση 4
Να γραφεί πρόγραμμα για το πρόβλημα του παραδείγματος 5.3.1, στο οποίο οι πράξεις μεταξύ πινάκων θα υλοποιούνται με συναρτήσεις που θα υπάγονται σε διεπαφή matrixOper.

Άσκηση 5
Λαμβάνοντας υπόψη:
(α) **την υλοποίηση της στοίβας με συνδεδεμένη λίστα στην υποενότητα 10.5.1,**
(β) **την εκτύπωση των περιεχομένων συνδεδεμένης λίστας στην υποενότητα 10.4.1,**
(γ) **τη διεπαφή allocateMemory της υποενότητας 11.4.3,**
να γραφεί πρόγραμμα για το πρόβλημα του παραδείγματος 10.2.2, στο οποίο θα δημιουργηθεί διεπαφή stackProcessing (αρχεία stackProcessing.h και stackProcessing.c) για τη φιλοξενία αποκλειστικά των συναρτήσεων διαχείρισης ουράς.

Άσκηση 6
Να γραφεί πρόγραμμα, το οποίο θα επιτελεί τα ακόλουθα:
(α) **Θα δέχεται από το πληκτρολόγιο έναν ακέραιο αριθμό `n`, οποίος εκφράζει τη διάσταση διανυσμάτων. Ακολούθως, θα δεσμεύεται δυναμικά μνήμη για τρία διανύσματα ακεραίων `a`, `b`, `c`. Θα γίνεται έλεγχος, ώστε η διάσταση που θα δώσει ο χρήστης να είναι μεγαλύτερη ή ίση του 3 και μικρότερη ή ίση του 6.**
(β) **Θα καλείται η συνάρτηση `void readVector(int *vector, int n)`, η οποία θα λαμβάνει από το πληκτρολόγιο τιμές για τα στοιχεία μονοδιάστατου πίνακα `n` θεσεων, έτσι ώστε έμμεσα να αποκτήσουν τιμές τα διανύσματα `a` και `b`.**
(γ) **Θα καλείται η συνάρτηση `void add_vectors(int *vector1, int *vector2, int *vector3, int n)`, η οποία θα αθροίζει τα περιεχόμενα των διανυσμάτων `a` και `b` και θα τα αποθηκεύει στο διάνυσμα `c`.**
(δ) **Θα καλείται η συνάρτηση `int innerProduct(int *vector1, int *vector2, int n)`, η οποία θα υπολογίζει το εσωτερικό γινόμενο των διανυσμάτων `a` και `b` και θα επιστρέφει το αποτέλεσμα στη συνάρτηση *main()*.**
(ε) **Τα περιεχόμενα των διανυσμάτων `a`, `b`, `c` και το εσωτερικό γινόμενο των δύο πρώτων θα εμφανίζονται στην οθόνη.**
(στ) **Οι συναρτήσεις θα βρίσκονται μέσα στη διεπαφή vectorFuncs.**

Άσκηση 7
Να *επεκταθεί* η διεπαφή sqMatrixProcessing (αρχεία sqMatrixProcessing.h και sqMatrixProcessing.c) με την προσθήκη των ακόλουθων λειτουργιών:
(α) **Εξαγωγή του ανάστροφου ενός τετραγωνικού πίνακα (κάθε στοιχείο A_{ij} ενός πίνακα Α γίνεται το στοιχείο A_{ji} του ανάστροφου του πίνακα Α).**
(β) **Έλεγχος κατά πόσον ένας τετραγωνικός πίνακας Α είναι συμμετρικός (κάθε στοιχείο A_{ij} ενός πίνακα Α ισούνται με το στοιχείο A_{ji} του πίνακα).**

Βιβλιογραφία κεφαλαίου

Collopy, D. (2002), *Introduction to C Programming: A Modular Approach*, Prentice Hall.

Deitel, H. & Deitel, P. (2014), *C Προγραμματισμός*, 7η έκδοση, Εκδόσεις Γκιούρδα.

Hanly, J. & Koffman, E. (2013), *Problem Solving and Program Design in C*, 7th ed., Pearson.

Hartel, P. & Muller, H. (1997), *Functional C*, Addison-Wesley.

King, K. (2008), *C Programming: A Modern Approach*, 2nd ed., W.W. Norton & Company.

Kochan, S. (2005), *Programming in C*, 3rd ed., SAMS Publishing.

Roberts, E. (2008), *Η Τέχνη και Επιστήμη της C*, Εκδόσεις Κλειδάριθμος.