Α' Εξάμηνο
ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Εργαστήριο – 5η σειρά ασκήσεων.
Κοζάνη, 7 Νοεμβρίου 2007.

Άσκηση p5-1 (μεταγλώττιση και εκτέλεση)
Δημιουργήστε ένα project, ονομάστε το p5 και δημιουργήστε τα παρακάτω αρχεία που θα ανήκουν σε αυτό:
α) Αρχείο με όνομα p5main.c που να περιέχει τον παρακάτω κώδικα (με γαλάζιο οι γραμμές που δίνονται ήδη από το περιβάλλον ανάπτυξης και με κόκκινο αυτές που θα προσθέσουμε):

```
#include <stdio.h>
#include "p5lib.h"

int main(int argc, char *argv[])
{
  do {
    int n;
    printf("\nΔώσε έναν ακέραιο: ");
    scanf("%d", &n);
    printf("Το παραγοντικό του %d είναι: %d\n\n", n, fact(n));
  } while(getchar() != ' ');
  system("PAUSE");
  return 0;
}
```

Σημείωση: *η συνθήκη* while(getchar( ) != ' ') *συνεπάγεται ότι η εκτέλεση συνεχίζεται όσο πατάτε Enter και μέχρι να πατήσετε κενό (space).*

β) Αρχείο με όνομα p5lib.c που περιέχει τον παρακάτω κώδικα.

```
#include "p5lib.h"

int fact(int n) {
    int p = 1;
    while( n > 0 ) p *= n--;
    return p;
}
```

Ο κώδικας αυτός υλοποιεί τον υπολογισμό του παραγοντικού ενός ακέραιου n, $n!=1\cdot 2\cdot 3\cdot ...(n-1)\cdot n$. Παρατηρείστε ότι χρησιμοποιούμε ως μεταβλητή αρίθμησης το ίδιο το n και ας είναι παράμετρος επειδή περνάει στη συνάρτησης μέσω τιμής και έτσι δεν κινδυνεύει να έχει αλλάξει με το τέλος της κλήσης της.

γ) Αρχείο “επικεφαλίδας” με όνομα p5lib.h και τον παρακάτω κώδικα.

```
#ifndef P5LIB_H
#define P5LIB_H

int fact(int );

#endif
```

Σε αυτό το αρχείο ορίζεται το πρωτότυπο της συνάρτησης που υπολογίζει το παραγοντικό. Προσέξτε τις οδηγίες προς τον προεπεξεργαστή για την “προστασία” του αρχείου από διπλή συμπερίληψη. Αν το αρχείο έχει ήδη συμπεριληφθεί μέσω κάποιας άλλης συνάρτησης τότε θα έχει ορίσει και τη συμβολική σταθερά P5LIB_H (μέσω της δεύτερης οδηγίας) οπότε τώρα η πρώτη οδηγία (#ifndef) δεν ισχύει και οι υπόλοιπες γραμμές μέχρι το #endif αγνοούνται. Με αυτή την τεχνική προστατεύουμε τα αρχεία επικεφαλίδας από το ενδεχόμενο πολλαπλής συμπερίληψης. Φυσικά, το σύστημα αυτό δουλεύει επειδή τηρούμε τη σύμβαση του να ορίζουμε συμβολικές σταθερές για το σκοπό αυτό με όνομα το όνομα του αντίστοιχου αρχείου επικεφαλίδας, αλλά με κεφαλαία γράμματα και _ αντί για τελεία.

Εισάγετε τα αρχεία στο project (αν δεν υπάρχουν ήδη σε αυτό), μεταγλωττίστε και εκτελέστε. Παρατηρείστε ότι το πρόγραμμα δουλεύει σωστά για μικρές τιμές του n αλλά από ένα σημείο και μετά βγάζει λάθος αποτελέσματα. Μπορείτε να καταλάβετε από ποιο σημείο και μετά συμβαίνει αυτό; Μπορείτε να κάνετε κάτι για να το διορθώσετε;

Άσκηση p5-2 (μεταγλώττιση και εκτέλεση)
Στο αρχείο p5lib.h εισάγετε άλλο ένα πρωτότυπο συνάρτησης (κάτω από την fact), της παρακάτω μορφής:

```
double dpow(double , int );
```

Στη συνέχεια στο τέλος του αρχείου p5lib.c προσθέστε τις παρακάτω γραμμές:

```
double dpow(double x, int n)
{
    double y = 1.0;
    if( x == 0)
        y = 0.0;
    else if( n == 0 )
        y = 1.0;
    else if( n > 0)
        for ( ; n > 0; n--) y *= x;
    else
        for ( ; n < 0; n++) y /= x;
    return y;
}
```

Αυτή η συνάρτηση υπολογίζει τη δύναμη του πραγματικού διπλής ακρίβειας x υψωμένου στη n, όπου n είναι ακέραιος, θετικός είτε αρνητικός (συμβατικά, η αόριστη παράσταση 0^0 θεωρείται ότι έχει την τιμή 0). Παρατηρείστε ότι στις δομές for, η συνθήκη αρχικοποίησης δεν υπάρχει. Πράγματι, χρησιμοποιούμε ως μεταβλητή αρίθμησης την παράμετρο n η οποία έχει ήδη μια αρχική τιμή και τη μειώνουμε σταδιακά κατά 1. Αφού η μεταβίβαση γίνεται μέσω τιμής και όχι μέσω διεύθυνσης, μετά την επιστροφή από τη συνάρτηση το όρισμα δε θα έχει μεταβληθεί.

Τροποποιήστε το αρχείο με τη main, δηλαδή το με το όνομα p5main.c ώστε να περιέχει τα παρακάτω (με γαλάζιο οι γραμμές που διατηρούνται από την τελευταία φορά, με κόκκινο οι νέες αλλαγές):

```
#include <stdio.h>
#include "p5lib.h"

int main(int argc, char *argv[])
{
  do {
    int n;
    double x;
    printf("\nΔώσε έναν πραγματικό: ");
    scanf("%lf", &x);
    printf("Δώσε έναν ακέραιο: ");
    scanf("%d", &n);
    printf("Η %d-η δύναμη του %lf είναι: %lf\n\n", n, x, dpow(x,n));
  } while(getchar() != ' ');
  system("PAUSE");
  return 0;
}
```

Μεταγλωττίστε και εκτελέστε.

Άσκηση p5-3 (μεταγλώττιση και εκτέλεση)
Στο αρχείο p5lib.h εισάγετε άλλο ένα πρωτότυπο συνάρτησης (κάτω από την dpow), της παρακάτω μορφής:

```
double dexp(double , int );
```

Στη συνέχεια στο τέλος του αρχείου p5lib.c προσθέστε τις παρακάτω γραμμές:

```
double dexp(double x, int n) {
    int i;
    double t, y = 0.0;
    for ( i = 0; i < n; i++) {
        t = dpow(x, i) / fact(i);
        y += t;
    }
    return y;
}
```

Η παραπάνω συνάρτηση υπολογίζει την εκθετική συνάρτηση χρησιμοποιώντας το ανάπτυγμα Taylor $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$. Η ακρίβεια του υπολογισμού μπορεί να ρυθμιστεί με τη βοήθεια του άλλου ορίσματος (n) με το οποίο ρυθμίζουμε πόσους όρους θέλουμε να περιλάβουμε στο ανάπτυγμα (n). Υποτίθεται ότι όσο περισσότερους όρους περιλάβουμε τόσο μεγαλύτερη είναι η ακρίβεια με την οποία προσεγγίζουμε το τελικό αποτέλεσμα.

Τροποποιήστε το αρχείο με τη main, δηλαδή το με το όνομα p5main.c ώστε να περιέχει τα παρακάτω (με γαλάζιο οι γραμμές που διατηρούνται από την τελευταία φορά, με κόκκινο οι νέες αλλαγές):

```
#include <stdio.h>
#include "p5lib.h"

int main()
{
  do {
    int n;
    double x, p;
    printf("\nΔώσε έναν πραγματικό: ");
    scanf("%lf", &x);
    printf("Δώσε αριθμό όρων: ");
    scanf("%d", &n);
    printf("Η τιμή του e στην %lf είναι: %lf\n\n", x, dexp(x,n));
  } while(getchar() != ' ');
  system("PAUSE");
}
```

Μεταγλωττίστε και εκτελέστε.
Κάντε δοκιμές για x = 1 και διάφορα n. Λογικά, για x = 1 και μεγάλα n (πάνω από 10) προσεγγίζεται η βάση των φυσικών λογάριθμων e = 2,7182818285... Κάντε δοκιμές για ακόμα μεγαλύτερα n και για διάφορα x (2, 3 κλπ). Υπάρχει κάποιο πρόβλημα; Έχει σχέση με το πρόβλημα της Άσκησης 1;

Άσκηση p5-4 (συνέχεια του προηγούμενου)
Η σχέση για το ανάπτυγμα Taylor της εκθετικής συνάρτησης μπορεί να γραφεί και στην εξής μορφή: $e^x = 1 + t_1 + t_2 + t_3 + t_4 + \ldots$ όπου $t_{n+1} = t_n \cdot \frac{x}{n}$. Με βάση αυτή την παρατήρηση μπορούμε να ξαναγράψουμε την υλοποίηση της εκθετικής συνάρτησης χωρίς να χρησιμοποιήσουμε δυνάμεις και παραγοντικά. Αρκεί μέσα στο for να πολλαπλασιάζουμε κάθε φορά το t με x διά i πριν το προσθέσουμε στο y.
Επιστρέψτε στη συνάρτηση dexp και ξαναγράψτε τη στην παρακάτω μορφή (με κόκκινο οι αλλαγές, όπως και πριν):

```
double dexp(double x, int n ) {
    int i;
    double t = 1.0, y = 1.0;
    for ( i =1; i < n; i++) {
        t *= (x / i);
        y += t;
    }
    return y;
}
```

Κάντε και πάλι δοκιμές για διάφορα x και διάφορα n. Παρατηρείτε κάποια βελτίωση;

Άσκηση p5-5

Δοκιμάστε να παραλείψετε περικλείοντας σε σχόλια /* */, τη γραμμή `#include "p5lib.h"` από το αρχείο p5main.c, από το αρχείο p5lib.c, καθώς και από τα δύο αρχεία συγχρόνως. Παρατηρείστε τι συνέπειες έχει αυτό στη μεταγλώττιση και εκτέλεση.

Άσκηση p5-6 (ανάπτυξη με τη βοήθεια ψευδοκώδικα)

Μετατρέψτε τις συναρτήσεις που γράψατε μέχρι τώρα στο αρχείο p5lib.c σε αναδρομικές. Υποδεικνύεται λύση σε μορφή ψευδοκώδικα.

Παραγοντικό

Εδώ καλούμε την fact με όλο και μικρότερο όρισμα n μέχρι να μηδενιστεί οπότε θα έχουμε μία συνθήκη που θα λέει ότι για n = 0 να επιστραφεί η μονάδα

```
Έστω p = 1
Αν n > 0 τότε θέσε p ίσο με n επί fact(n-1)
Αλλιώς δώσε πίσω (return) 1
```

Ύψωση σε δύναμη

Εδώ η λογική είναι παρόμοια, μόνο που αν η δύναμη είναι αρνητική πρέπει να αυξάνουμε αντί να μειώνουμε το n ώσπου να φτάσει στο μηδέν και ακόμη, να διαιρούμε διά του x αντί να πολλαπλασιάζουμε με αυτό.

```
Αν x ίσο με 0 τότε δώσε πίσω 0
Αλλιώς αν n ίσο με 0 δώσε πίσω 1
Αλλιώς αν n θετικό πολλαπλασίασε dpow(x,n-1)επί x
Αλλιώς διαίρεσε dpow(x,n+1) διά του x.
```

Εκθετική συνάρτηση

Εδώ, αν δε θέλουμε να χρησιμοποιήσουμε πάλι δυνάμεις και παραγοντικά που είναι σχετικά πιο εύκολο αλλά έχει προβλήματα υπερχείλισης, θα σκεφτούμε πιο πονηρά: ο όρος $t_{n+1}=t_n \cdot x/n$ που προστίθεται στο άπειρο άθροισμα μπορεί να γραφεί σαν αυτοτελής συνάρτηση των x και n, με όνομα έστω term, ενώ το άθροισμα των προηγούμενων όρων $1+t_1+t_2+t_3+t_4+...+t_{n-1}$ δεν είναι παρά το dexp(x, n-1), άρα προκύπτει αμέσως μια αναδρομική σχέση της μορφής

```
dexp(x, n) = dexp(x, n-1) + term(x, n);
```

Τη συνάρτηση term μπορούμε να την υλοποιήσουμε είτε αναδρομικά είτε όχι. Εδώ την αντιμετωπίζουμε και αυτή ως αναδρομική συνάρτηση.

```
dexp
Αν n ίσο με 0 τότε δώσε πίσω 1
Αλλιώς δώσε dexp(x, n-1) + term(x, n)
term
Αν n ίσο με 0 τότε δώσε πίσω 1
Αλλιώς δώσε το γινόμενο του term(x, n-1) επί x/n
```

Προσοχή! Δεν ξεχνάμε να βάλουμε το πρωτότυπο της term στο αρχείο επικεφαλίδας p5lib.h

Άσκηση p5-7

Τις παραπάνω αναδρομικές συναρτήσεις μπορείτε να τις γράψετε και σε πιο "πυκνή" μορφή αν αντί για δομή if χρησιμοποιήσετε τον τριαδικό τελεστή επιλογής (συνθήκη ? έκφραση για αληθή : έκφραση για ψευδή). Δοκιμάστε να χωρέσετε το σώμα της κάθε συνάρτησης σε μία μόνο εντολή.

Προτεινόμενες απαντήσεις – λύσεις.

Άσκηση p5-1

Παρατηρούμε ότι μέχρι $n = 13$ οι τιμές αυξάνονται φυσιολογικά ενώ μετά παίρνουμε "περίεργα" αποτελέσματα, ακόμη και αρνητικούς αριθμούς ή 0. Αυτό οφείλεται στο φαινόμενο της υπερχείλισης (overflow). Το παραγοντικό αυξάνεται τόσο γρήγορα ώστε σύντομα ο τύπος int δεν επαρκεί για την αναπαράστασή του. Μια μικρή βελτίωση μπορεί να περιμένει κανείς αν χρησιμοποιήσει τον τύπο long int, αν και το παραγοντικό αυξάνεται τόσο γρήγορα ώστε για μεγάλα n και πάλι δε μπορούμε να περιμένουμε σωστά αποτελέσματατα. Αν υπάρχουν μαθηματικές παραστάσεις όπου υπεισέρχονται παραγοντικά πρέπει να λάβουμε υπ' όψιν αυτό τον περιορισμό και να ανασχεδιάσουμε τους υπολογισμούς μας ώστε να αποφύγουμε την εμφάνιση παραγοντικών, όπως θα δούμε στη συνέχεια.

Άσκηση p5-3

Το ανάπτυγμα Taylor υποτίθεται ότι δίνει τη σωστή τιμή της συνάρτησης που παριστάνει, για αριθμό όρων αθροίσματος n που τείνει στο άπειρο. Στην πράξη, για αριθμό όρων n της τάξης των δεκάδων περιμένουμε να πάρουμε την τιμή της αναπαριστώμενης συνάρτησης με πολύ μεγάλη ακρίβεια. Έτσι, για $x = 1$, περιμένουμε να πάρουμε τη βάση των φυσικών λογάριθμων αφού $e^1 = e$. Παρατηρούμε ότι καθώς αυξάνουμε το n οι λαμβανόμενες τιμές προσεγγίζουν τη σωστή απάντηση αλλά από ένα σημείο και μετά αποκλίνουν. Για μεγαλύτερα x το πρόβλημα επιδεινώνεται. Αυτό γίνεται κατανοητό αν παρατηρήσουμε ότι στην υλοποίηση που υιοθετούμε εδώ υπάρχουν παραγοντικά (συνάρτηση fact) που "πάσχουν" από προβλήματα υπερχείλισης όπως εξηγήσαμε. Είναι φανερό ότι η απευθείας μεταφορά μαθηματικών τύπων από την αλγεβρική μορφή τους σε πρόγραμμα δεν είναι πάντα καλή λύση, προγραμματιστικά! Στην επόμενη άσκηση έχουμε την ευκαιρία να δούμε ένα παράδειγμα πιο αποτελεσματικού προγραμματισμού.

Άσκηση p5-4

Ο ανασχεδιασμός που προτείνεται εδώ δεν κάνει καθόλου χρήση παραγοντικών και δυνάμεων οπότε και τα προβλήματα που προαναφέρθηκαν εξαφανίζονται! Ακόμη και για $n = 100$ και για διάφορες τιμές του x τα αποτελέσματα είναι απολύτως ικανοποιητικά. Στην πράξη αρκούν πολύ μικρότερες τιμές του n για να επιτύχουμε εξαιρετική ακρίβεια. Μια δεύτερη ευνοϊκή συνέπεια είναι ότι ελαττώθηκαν οι απαιτούμενες πράξεις για τη λήψη του αποτελέσματος. Πράγματι, τόσο η συνάρτηση dpow όσο και η fact απαιτούν n πολλαπλασιασμούς ή διαιρέσεις, άρα $2n$ τέτοιες πράξεις απαιτούνται για τον n-στό όρο του αθροίσματος Taylor. Για όλη τη σειρά θα έχουμε $n(n-1)/2$ πολλαπλασιασμούς και άλλες τόσες διαιρέσεις. Υποθέτοντας ότι απαιτούν τον ίδιο χρόνο, έχουμε χρόνο υπολογισμού ανάλογο του n^2-n . Επειδή το n^2 είναι πολύ μεγαλύτερο από το n λέμε ότι ο χρόνος υπολογισμού αυξάνεται σαν n^2 και γράφουμε $O(n^2)$ για να δείξουμε την τάξη μεγέθους αυτής της εξάρτησης.

Με τη νέα υλοποίηση αρκούν μόλις μία διαίρεση (x / i) και ένας πολλαπλασιασμός (t *= (x/i)) για κάθε προστιθέμενο όρο. Για όλη τη σειρά έχουμε χρόνο ανάλογο του $2n$ δηλαδή μόλις γραμμική εξάρτηση και γράφουμε $O(n)$. Από αυτό φαίνεται ότι το όφελος με όρους χρόνου εκτέλεσης μεταφράζεται σε επιτάχυνση κατά μία τάξη μεγέθους. Αν και σε ένα μικρό πρόγραμμα επίδειξης όπως το παρόν, δεν είναι φανερό, αν σκοπεύαμε να χρησιμοποιήσουμε τη συνάρτηση dexp σε μία "σοβαρή" εφαρμογή που πρέπει να κάνει χιλιάδες ή εκατομμύρια μαθηματικές πράξεις αυτού του είδους σε κάθε υπολογισμό, η διαφορά στην ταχύτητα εκτέλεσης θα ήταν παραπάνω από αισθητή!

Συμπέρασμα: ο αποτελεσματικός προγραμματισμός δεν είναι μόνο υπόθεση της γνώσης κάποιας γλώσσας προγραμματισμού αλλά περιλαμβάνει και σχεδιασμό τέτοιο που μεταξύ άλλων να εξοικονομεί μνήμη και υπολογιστικό χρόνο ενώ συγχρόνως θα οδηγεί σε αποτελέσματα με την ορθότητα και ακρίβεια που επιβάλλει το εκάστοτε πρόβλημα.

Άσκηση p5-5

Κατ' αρχήν το πρωτότυπο της συνάρτησης χρησιμεύει στην προστασία από λάθη κατά τον προγραμματισμό, όπως λάθος λίστα παραμέτρων κατά την κλήση της συνάρτησης. Πριν υιοθετηθούν τα πρωτότυπα συναρτήσεων από την ANSI C, τέτοια λάθη οφειλόμενα στη μεγάλη ελευθερία που παρείχε η C στον προγραμματιστή, ήταν συχνά και περνούσαν απαρατήρητα οδηγώντας σε εσφαλμένα αποτελέσματα. Τώρα, τα λάθη αυτά εντοπίζονται αμέσως κατά τη μεταγλώττιση εφόσον με την οδηγία #include έχουν περιληφθεί τα κατάλληλα αρχεία επικεφαλίδων.

Ακόμη και αν οι συναρτήσεις χρησιμοποιούνται σωστά, η παράλειψη των πρωτοτύπων δεν είναι καλή ιδέα. Το τι ακριβώς θα συμβεί εξαρτάται και από το μεταγλωττιστή που χρησιμοποιούμε. Εδώ αναφέρουμε αποτελέσματα από τη γραμμή εντολών του SUSE Linux Desktop Enterprise 10.0 με χρήση του μεταγλωττιστή της C που διαθέτει η συγκεκριμένη διανομή (εντολή cc). Δοκιμάζοντας να παραλείψουμε την οδηγία #include “p5lib.h” παίρνουμε τα εξής αποτελέσματα.

- Για παράλειψη και από τα δύο αρχεία πηγαίου κώδικα (p5main.c και p5lib.c) ο μεταγλωττιστής εντοπίζει πρόβλημα και διακόπτει τη μεταγλώττιση δίνοντας το παρακάτω μήνυμα:

```
p5lib.c:34: error: conflicting types for ‘term’
p5lib.c:31: error: previous implicit declaration of ‘term’ was here
```

- Το ίδιο συμβαίνει και αν παραλείψουμε την εν λόγω οδηγία μόνο από το αρχείο p5lib.c.
- Αν αφαιρέσουμε αυτή την οδηγία από το κύριο πρόγραμμα, αρχείο p5main.c, έχουμε το χειρότερο ενδεχόμενο, δηλαδή το πρόγραμμα μεταγλωττίζεται κανονικά σα να μη συμβαίνει τίποτα αλλά αν δώσουμε συγκεκριμένες τιμές θα πάρουμε κάτι σαν αυτό:

```
Δώσε έναν πραγματικό: 5
Δώσε αριθμό όρων: 5
Η τιμή του e στην 5.000000 είναι: 0.000000

Δώσε έναν πραγματικό: 1
Δώσε αριθμό όρων: 100
Η τιμή του e στην 1.000000 είναι: 0.000000

Δώσε έναν πραγματικό: 2.34
Δώσε αριθμό όρων: 22
Η τιμή του e στην 2.340000 είναι:
  -867922379887512348474590444521339851718469349529074666013828745876385635798009328443860528337036591525091612650377128135222999873606426050022084241614921579503157208363070750529525319223256375651417911336108032.000000
```

Σε συμφωνία με τους κανόνες της γλώσσας C ο υπολογιστής θεωρεί ότι η παράσταση dexp ακολουθούμενη από παρενθέσεις είναι συνάρτηση αλλά, απουσία δηλωμένου πρωτοτύπου, το αποτέλεσμα είναι τελείως όσο και απρόβλεπτα λάθος!

Άσκηση p5-6

Η προτεινόμενη υλοποίηση σε μορφή κώδικα C είναι η εξής (με κόκκινο χρώμα οι εντολές όπου εμφανίζεται η αναδρομή και με έντονα γράμματα η αναδρομικά καλούμενη συνάρτηση):

```
#include "p5lib.h"

int fact(int n) {
    int p = 1;
    if(n > 0) p = n * fact(n-1);
```

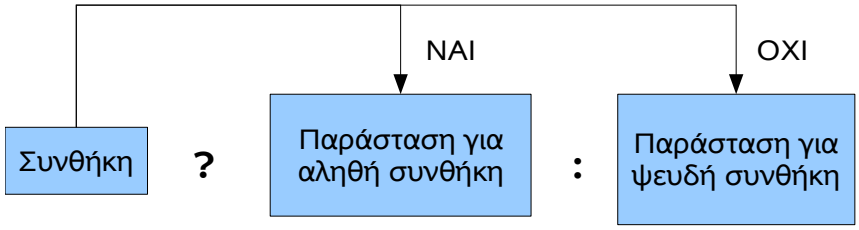
```
    return p;
}

double dpow(double x, int n) {
    if( x == 0)
        y = 0.0;
    else if( n == 0 )
        y = 1.0;
    if (n > 0)
      y = dpow(x, n-1) * x;
    else if (n < 0)
      y = dpow(x, n+1) / x;
    return y;
}

double dexp(double x, int n) {
    double y = 1.0;
    if(n > 0) y = term(x, n) + dexp(x, n-1);
    return y;
}

double term(double x, int n) {
    double y = 1.0;
    if(n > 0) y = term(x, n-1) * x / n;
    return y;
}
```

Σημείωση: η τεχνική της αναδρομής δεν εμπίπτει στην κατηγορία του δομημένου προγραμματισμού καθώς δεν αντιστοιχεί σε κάποια από τις θεμελιώδεις δομές (διαδοχή, επιλογή, επανάληψη) στις οποίες μπορούν να αναχθούν όλοι οι αλγόριθμοι. Αποδεικνύεται ότι όλες οι αναδρομικές συναρτήσεις μπορούν να υλοποιηθούν και χωρίς αναδρομή, με χρήση μόνο των βασικών δομών.

Άσκηση p5-6

Η χρήση του τριαδικού τελεστή γίνεται σύμφωνα με το σχήμα

(συνθήκη ? παράσταση_αν_αληθής : παράσταση_αν_ψευδής)



Για να μεταφράσουμε τις παραπάνω δομές if σε "γλώσσα" του τελεστή επιλογής ? μπορούμε να εργαστούμε ως εξής:

γράφουμε το "σκελετό" της ζητούμενης δομής

```
... ? ... : ... .
```

Στην πρώτη θέση γράφουμε την πρώτη συνθήκη μέσα σε if( ), Σ1, που συναντάμε

```
Σ1 ? ... : ... .
```

Στην τρίτη θέση γράφουμε την παράσταση που επιστρέφεται αν η συνθήκη Σ1 είναι ψευδής:

```
Σ1 ? ... : Π(αν Σ1 ψευδής)
```

Στη δεύτερη θέση αν μεν απομένει άλλη μία παράσταση (για αληθή Σ1) γράφουμε αυτή και τελειώνουμε,

```
Σ1 ? Π(αν Σ1 αληθής) : Π(αν Σ1 ψευδής)
```

ενώ αν υπάρχουν και άλλες επιλογές γράφουμε ξανά σε παρένθεση τον "σκελετό" της τριαδικής δομής

```
Σ1 ? ( ... ? ... : ... ) : Π(αν Σ1 ψευδής)
```

και εντοπίζουμε την επόμενη συνθήκη μέσα σε if( ), Σ2, που συναντάμε, την οποία και βάζουμε στην πρώτη θέση της ένθετης δομής:

```
Σ1 ? ( Σ2 ? ... : ... ) : Π(αν Σ1 ψευδής)
```

και εργαζόμαστε ανάλογα, όπως και με τη Σ1 μέχρι να τελειώσουν όλες οι συνθήκες και οι επιλογές που τους αντιστοιχούν.

Μία προτεινόμενη υλοποίηση των παραπάνω προγραμμάτων με τον τριαδικό τελεστή επιλογής είναι η εξής (με κόκκινο χρώμα η εντολή που περιέχει τον τελεστή ? και με έντονα γράμματα η αναδρομικά καλούμενη συνάρτηση):

```
#include "p5lib.h"

int fact(int n) {
    return ( n > 0 ? n * fact(n-1) : 1 );
}

double dpow(double x, int n) {
    return (x ? (n ? (n > 0 ? dpow(x, n-1) * x : dpow(x, n+1) / x ) : 1) : 0.0);
}

double dexp(double x, int n) {
    return ( n > 0 ? term(x, n) + dexp(x, n-1) : 1.0 );
}

double term(double x, int n) {
    return ( n > 0 ? term(x, n-1) * x / n : 1.0 );
}
```