ΕΡΓΑΣΙΑ 4
ΥΠΟΠΡΟΓΡΑΜΜΑΤΑ

Χρήστος Μαργιώλης – Εργαστηριακό Τμήμα 9

Περιεχόμενα

1. Δομή των προγραμμάτων και τρόποι εκτέλεσης....3
1.1. Εκτέλεση από Linux....3
1.2. Εκτέλεση από Windows....5
2. sine-cos-taylor – Άσκηση 1....7
2.1. main.c....7
2.2. sine-cos-taylor-libs.c....7
2.3. sine-cos-taylor.h....10
2.4. Περιγραφή υλοποίησης άσκησης 1....10
2.4.1. Μεταβλητές....10
2.4.2. Λειτουργία του προγράμματος....11
2.4.3. Σχεδιάγραμμα συναρτήσεων....12
3. menu – Άσκηση 2....12
3.1. menu.c....12
3.2. menu-libs.c....13
3.3. menu.h....15
3.4. Περιγραφή υλοποίησης άσκησης 2....15
3.4.1. Μεταβλητές....15
3.4.2. Λειτουργία του προγράμματος....16
3.4.3. Σχεδιάγραμμα συναρτήσεων....16
4. Άσκηση 3....17
5. hanoi-tower – Άσκηση 4....17
5.1. hanoi-tower.c....17
5.2. Περιγραφή υλοποίησης άσκησης 4....18
5.2.1. Κανόνες παιχνιδιού....18
5.2.2. Αλγόριθμος....19
5.2.3. Μεταβλητές....19
5.2.4. Λειτουργία του προγράμματος....19
6. Διευκρινήσεις....20
7. Εργαλεία....20

1. Δομή των προγραμμάτων και τρόποι εκτέλεσης

1. Όπου *program_name* = το όνομα του εκάστοτε[2] προγράμματος (*menu* ή *sine-cos-taylor*).
2. Όπου *full_path/* = ολόκληρο το path.
3. Όπου $ = εντολή terminal. (**Δεν** αντίγράφεται μαζί με την υπόλοιπη εντολή)

Τα προγράμματα είναι δομημένα ώς εξής:

Υπάρχουν 3 φάκελοι – στον φάκελο src βρίσκονται τα αρχεία main.c και *program_name*-libs.c, οι πηγαίοι κώδικες, καθώς και το header file *program_name*.h, το οποίο περιέχει τα prototypes των συναρτήσεων που χρησιμοποιούνται. Το main.c περιέχει μόνο την συνάρτηση main του προγράμματος, στην οποία καλούνται συναρτήσεις που περιέχονται – *μαζί με επιπλέον συναρτήσεις που είναι απαραίτητες για την υλοποίηση* – στο αρχείο *program_name*-libs.c. Το αρχείο αυτό έχει ως σκοπό την συσσώρευση και την οργάνωση όλων των συναρτήσεων που έφτιαξα σε ένα ξεχωριστό αρχείο.

Στον φάκελο src υπάρχει *επίσης* και το αρχείο *program_name*-full.c, το οποίο είναι το κάθε πρόγραμμα με τον *ίδιο* κώδικα, μαζεμένο όλο σε ένα αρχείο, και υπάρχει για διευκόλυνση σε περίπτωση που εμφανίστει οποιοδήποτε πρόβλημα στην ένωση πολλαπλών αρχείων (αν η εκτέλεση γίνεται στο Dev++[1] είναι επίσης πιο πρακτικό *μόνο* με αυτό το αρχείο).

Στον φάκελο obj αποθηκεύονται τα .o object αρχεία που παράγονται κατα την μεταγλώττιση και στον φάκελο exec αποθηκεύεται το *τελικό* εκτέλεσιμο αρχείο.

1.1. Εκτέλεση από Linux

Για συστήματα ***Linux***, προκειμένου να εκτελεστεί το πρόγραμμα[2] τα βήματα είναι τα εξής:

1. Άνοιγμα οποιουδήποτε terminal emulator, ή του integrated terminal του εκάστοτε editor.
2. Μετάβαση στο directory[3] που βρίσκονται οι πηγαίοι κώδικες: `$ cd full_path/programs/program_name/src`
3. Compilation του main.c και των βιβλιοθηκών: `$ gcc -c main.c program_name-libs.c && mv main.o program_name-libs.o ../obj && cd ../obj`
4. Linking των βιβλιοθηκών με το main: `$ gcc main.o program_name-libs.o -lm && mv a.out ../exec && cd ../exec`
5. Εκτέλεση του .out αρχείου: `$ ./a.out`

Εναλλακτικός τρόπος εκτέλεσης από Linux:

1. Άνοιγμα οποιουδήποτε terminal emulator, ή του integrated terminal του editor.
2. Μετάβαση στο directory που βρίσκονται τα προγράμματα: `$ cd full_path/programs`
3. *Σε περίπτωση που δεν δουλέψει κατευθείαν το βήμα 4:* `$ chmod +x autocompile.sh`
4. Εκτέλεση του script `autocompile.sh`: `$ ./autocompile.sh program_name`

Σε περίπτωση που το πρόγραμμα προς εκτέλεση είναι το **`hanoi-tower`**, χρειάζεται μόνο το εξής: `$ cd full_path/programs/hanoi-tower && gcc -c hanoi-tower.c && gcc hanoi-tower.o && ./a.out`

Στα βήματα 3 και 4 της *πρώτης μεθόδου* υπάρχουν και *δύο επιπλέον* εντολές. Στο βήμα 3 η εντολή `$ mv …` θα μεταφέρει τα `.o` αρχεία στον φάκελο `obj`, και η εντολή `$ cd …` θα μεταφέρει τον χρήστη στον φάκελο `obj`. Στο βήμα 4 η εντολή `$ mv …` θα μεταφέρει το εκτελέσιμο αρχείο στον φάκελο `exec` και με την εντολή `$ cd …` ο χρήστης θα μεταφερθεί στον φάκελο `exec`, ώστε να μπορέσει να εκτελέσει αρχείο.

Και οι 2 αυτές εντολές έχουν χρησιμοποιηθεί για περισσότερη οργάνωση, καθώς και για να αποφευχθούν τυχόν λάθη κατά την μεταφορά αρχείων και αλλαγές directory.

Το `autocompile.sh` είναι ένα *bash shell script* που έφτιαξα *πειραματικά*[4], για περισσότερη ευκολία, το οποίο περιέχει τις εντολές της πρώτης μεθόδου, συν τις επιπλέον λειτουργίες ότι δημιουργεί τους φακέλους `obj` και `exec` σε περίπτωση που δεν υπάρχουν, και εκτελεί το πρόγραμμα. Ο παρακάτω κώδικας είναι ο κώδικας του `autocompile.sh`:

```
#!/bin/bash

# Cd to program directory
cd $1

# Create (if missing) obj and exec directories
mkdir -p obj exec

# Cd to source code directory
cd src

# Compile and move new files to the right directory
gcc -c main.c $1-libs.c && mv main.o $1-libs.o ../obj &&
cd ../obj;
gcc main.o $1-libs.o -lm && mv a.out ../exec && cd ../exec;

# Execute program
./a.out
```

1.2. Εκτέλεση από Windows

Για συστήματα ***Windows***, η διαδικασία είναι η εξής:

Αν *υπάρχει* ο gcc compiler από την MinGW, και η εκτέλεση του πρόγραμματος *δεν* γίνεται στο Dev++, τότε

1. Άνοιγμα της γραμμής εντολών Command Prompt, ή του Integrated Terminal του εκάστοτε editor.
2. Μετάβαση στο directory που βρίσκονται οι πηγαίοι κώδικες του προγράμματος: `$ cd full_path/programs/program_name/src`
3. Compilation του `main.c` και των βιβλιοθηκών: `$ gcc -c main.c program_name-libs.c && move main.o ../obj && move program_name-libs.o ../obj && cd ../obj`
4. Linking των βιβλιοθηκών με το `main`: `$ gcc main.o program_name-libs.o -lm && move a.exe ../exec && cd ../exec`
5. Εκτέλεση του `.exe` αρχείου: `$ a.exe`

Εναλλακτικός τρόπος εκτέλεσης από Windows:

1. Άνοιγμα της γραμμής εντολών Command Prompt, ή του Integrated Terminal του εκάστοτε editor.
2. Μετάβαση στο directory των προγραμμάτων: `$ cd full_path/programs`
3. Εκτέλεση του `autocompile.bat`: `$ autocompile.bat`
4. Εισαγωγή ονόματος προγράμματος *ακριβώς* όπως αναγράφεται στον φάκελο του. Το όνομα θα εισαχθεί όταν εμφανιστεί το μήνυμα "`Program name: `"

[1]Αν η εκτέλεση του προγράμματος γίνεται στο Dev++, τότε

1. Άνοιγμα του `program-full.c` στο Dev++.
2. Εκτέλεση (Build and Run).

Σε περίπτωση που το πρόγραμμα προς εκτέλεση είναι το **`hanoi-tower`**, χρειάζεται μόνο το εξής: `$ cd full_path/programs/hanoi-tower && gcc -c hanoi-tower.c && gcc hanoi-tower.o && a.exe`

Όπως και στη διαδικασία εκτέλεσης από Linux, έτσι και για τα Windows μετέτρεψα σε batch script το `autocompile.sh`, ώστε να εκτελεί *ακριβώς* τις ίδιες λειτουργίες. Ο παρακάτω κώδικας είναι ο κώδικας του `autocompile.bat`

```
@ECHO OFF

set /p "program=Program name: "

:: Cd to program directory
cd %program%

:: Create (if missing) obj and exec directories
if not exist obj md obj
if not exist exec md exec

:: Cd to source code directory
cd src

:: Compile and move object files to the right directory
gcc -c main.c %program%-libs.c
move main.o ../obj && move %program%-libs.o ../obj
cd ../obj

:: Link and move executable to the right directory
gcc main.o %program%-libs.o -lm
move a.exe ../exec
cd ../exec

:: Execute program
a.exe
```

2. sine-cos-taylor – Άσκηση 1

2.1. main.c

```c
#include <stdio.h>
#include <math.h>
#include "sine-cos-taylor.h"


int main(int argc, char **argv)
{
    double xDegrees, xRads;

    xDegrees = input();
    xRads = rads_conv(xDegrees);

    if (compare(sine_calc(xRads), sin(xRads)))
        printf("sin(%f) = sin(%f) = %f rads\n", xDegrees,
xRads, sine_calc(xRads));
    else
        printf("Error.\n");

    if (compare(cosine_calc(xRads), cos(xRads)))
        printf("cos(%f) = cos(%f) = %f rads\n", xDegrees,
xRads, cosine_calc(xRads));
    else
        printf("Error.\n");

    return 0;
}
```

2.2. sine-cos-taylor-libs.c

```c
#include <stdio.h>
#include <math.h>
#include "sine-cos-taylor.h"

#define PI 3.141592654
#define ACCURACY 0.000001
#define COMPARISON 0.000009


double input()
{
    double xDegrees;
```

```
    printf("x (in degrees): ");
    scanf("%lf", &xDegrees);

    return xDegrees;
}

double rads_conv(double xDegrees)
{
    return xDegrees * (PI/180.0);
}

double sine_calc(double xRads)
{
    double currentFrac, previousFrac, sineValue = 0;
    int exponent = 1, sign = 1;

    currentFrac = power(xRads, exponent) factorial(exponent);

    do
    {
        sineValue += sign * currentFrac;

        exponent += 2;

        previousFrac = currentFrac;
        currentFrac = power(xRads, exponent) /
factorial(exponent);

        sign *= -1;
    } while (fabs(previousFrac - currentFrac) > ACCURACY);

    return sineValue;
}

double cosine_calc(double xRads)
{
    double currentFrac = 1, previousFrac, cosineValue = 0;
    int exponent = 0, sign = 1;

    do
    {
        cosineValue += sign * currentFrac;

        exponent += 2;

        previousFrac = currentFrac;
        currentFrac = power(xRads, exponent) /
factorial(exponent);
```

```
        sign *= -1;
    } while (fabs(previousFrac - currentFrac) > ACCURACY);

    return cosineValue;
}

double power(double xRads, int exponent)
{
    int i;
    double value;

    for (i = 0, value = 1; i < exponent; i++)
        value *= xRads;

    return value;
}

double factorial(int exponent)
{
    int i;
    double fac;

    for (i = 1, fac = 1; i <= exponent; i++)
        fac *= i;

    return fac;
}

int compare(double calcValue, double funcValue)
{
    if (fabs(calcValue - funcValue) <= COMPARISON)
        return 1;
    else
        return 0;
}
```

2.3. sine-cos-taylor.h

```
#ifndef HEADER_FILE
#define HEADER_FILE

double input();
double rads_conv(double);
double sine_calc(double);
double cosine_calc(double);
double power(double, int);
double factorial(int);
int compare(double, double);

#endif
```

2.4. Περιγραφή υλοποίησης άσκησης 1

2.4.1. Μεταβλητές

`xDegrees`: x σε μοίρες
`xRads`: x σε ακτίνια (rads)
`currentFrac`: τρέχον κλάσμα
`previousFrac`: προηγούμενο κλάσμα
`sineValue`: τιμή ημιτόνου
`cosineValue`: τιμή συνημιτόνου
`exponent`: εκθέτης
`sign`: πρόσιμο
`value`: επιστρεφόμενη τιμή συνάρτησης δυνάμεων
`fac`: επιστρεφόμενη τιμή συνάρτησης παραγοντικών
`calcValue`: τιμή ημιτόνου/συνημιτόνου των συναρτήσεων `sine_calc()` και `cosine_calc()` που δέχεται η συνάρτηση σύγκρισης
`funcValue`: τιμή ημιτόνου/συνημιτόνου των έτοιμων συναρτήσεων `sin()` και `cos()` της `math.h` που δέχεται η συνάρτηση σύγκρισης

2.4.2. Λειτουργία του προγράμματος

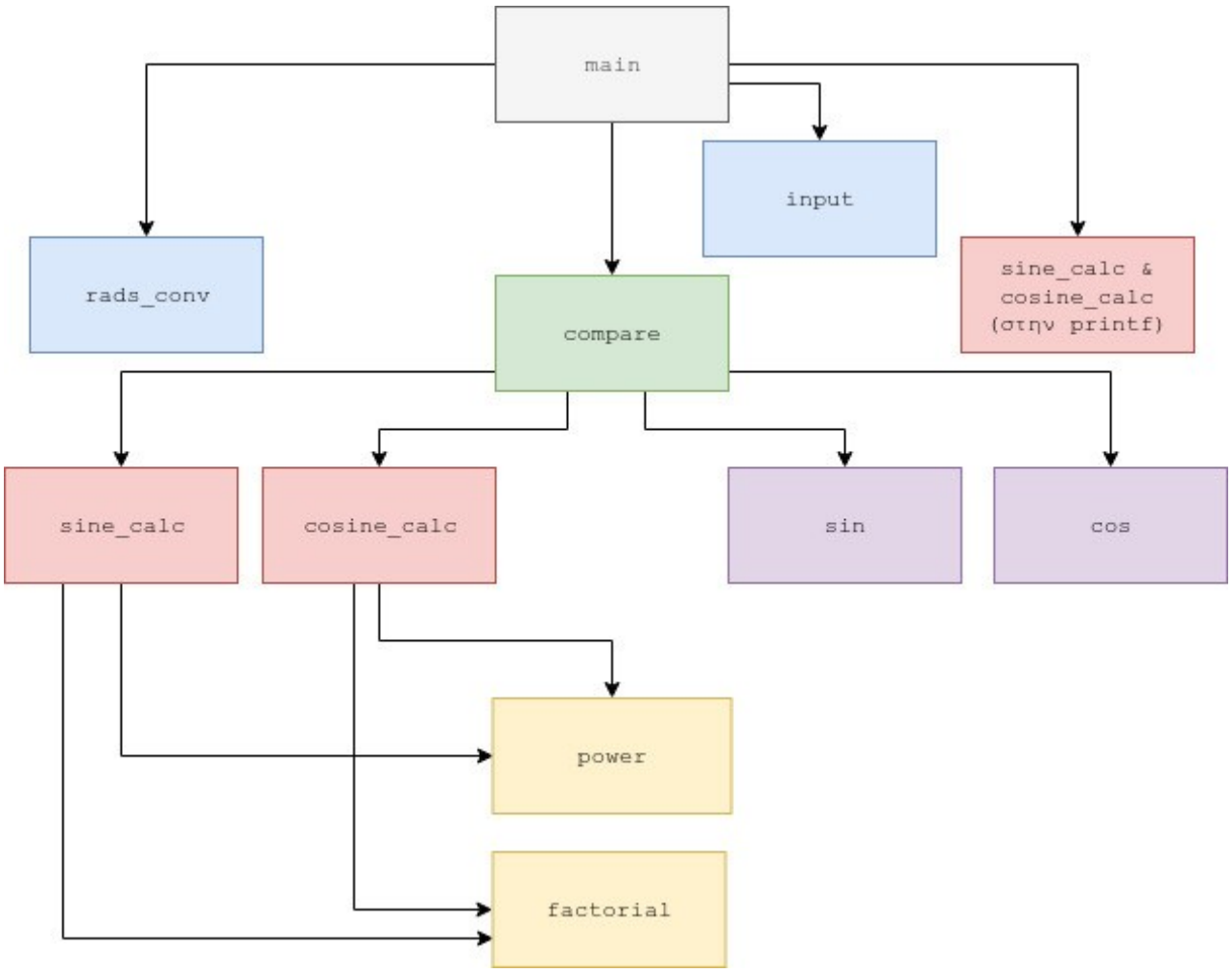
Στο `main.c` καλείται η συνάρτηση `input()` ώστε να διαβάσει το `x` σε μοίρες και έπειτα καλούνται οι συναρτήσεις μετατροπής μοιρών σε rads, υπολογισμού ημιτόνου και συνημιτόνου, καθώς και σύγκρισης των τιμών που δίνουν οι παραπάνω δύο *τελευταίες* συναρτήσεις με τις αντίστοιχες συναρτήσεις που παρέχει η `math.h`. Στο `main.c` επίσης εμφανίζονται και τα αποτέλεσματα, και στο `sine-cos-taylor-libs.c` γίνονται όλοι οι υπολογισμοί. Η διαδικασία είναι η εξής:

Αρχικά ο χρήστης δίνει το `x` σε μοίρες και το πρόγραμμα το μετατρέπει αμέσως σε rads με την συνάρτηση `rads_conv()` και η τιμή του `x` σε rads επιστρέφεται στην `main` στην μεταβλητή `xRads.` Στην συνέχεια καλείται η συνάρτηση `compare()`, η οποία παίρνει ώς παραμέτρους την τιμή που επιστρέφει η συνάρτηση υπολογισμού ημιτόνου `sine_calc()` και η συνάρτηση για υπολογισμό ημιτόνου `sin()` που παρέχει η `math.h`. Η `compare()` θα συγκρίνει τις δύο τιμές και αν $|\tau\iota\mu\acute{\eta}_1 - \tau\iota\mu\acute{\eta}_2| \leq 000009$, δηλαδή αν οι δύο τιμές είναι σχεδόν (ή και ακριβώς) ίσες, θα επιστρέψει 1 (`true`) και θα εμφανιστεί στην οθόνη η τιμή του ημιτόνου του `xRads`, με βάση τους υπολογισμούς της `sine_calc()`. Ειδάλλως, θα επιστρέψει 0 (`false`) και θα εμφανιστεί στην οθόνη το μήνυμα “Error”.

Όμως στην συνάρτηση `sine_calc()` γίνονται οι εξής λειτουργίες προκειμένου να μας δωθεί μια τιμή: *πρώτα απ’όλα*, η συνάρτηση βρίσκει την σειρά του ημιτόνου κλάσμα προς κλάσμα. Δίνεται το `xRads` ως παρέμετρος και η αρχική τιμή του εκθέτη είναι 1, ώστε να είναι πάντα περιττός. Καλούνται οι συναρτήσεις `power()` και `factorial()` ωστέ να υπολογίσουν την δύναμη και το παραγοντικό και να βρεθεί το τρέχον κλάσμα `currentFrac` της σειράς, και στην συνέχεια το κλάσμα αυτό προστείθεται στην *σειρά* του ημιτόνου και ο εκθέτης αυξάνεται κατά 2. Έτσι, το προηγούμενο κλάσμα (`previousFrac`) τώρα είναι το τρέχον κλάσμα, και το *νέο* τρέχον κλάσμα είναι το κλάσμα που θα προκύψει από την νέα κλήση των συνάρτησεων `power()` και `factorial()`, επειδή ο εκθέτης άλλαξε, άρα θα αλλάξει και η τιμή του νέου κλάσματος. Η μεταβλητή `sign` *αλλάζει πρόσιμο* σε κάθε επανάληψη ώστε να επιτευχθεί η εναλλαγή προσίμων της σειράς $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \ldots$. Αυτή η επανάληψη συνεχίζεται όσο η διαφορά του προηγούμενου και του τρέχοντος κλάσματος κατά απόλυτη τιμή είναι μεγαλύτερη του $10^{-6} = 0.000001$.

Για τον υπολογισμό του συνημιτόνου με την συνάρτηση `cosine_calc()`, η διαδικάσια είναι η ίδια, με την διαφορά οτι η σειρά είναι $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \frac{x^8}{8!} \ldots$ οπότε πρέπει να κάνουμε τις εξής μετατροπές:
Αρχικοποιούμε το τρέχον κλάσμα σε 1 και δεν υπολογίζουμε έξω από την `do-while` το πρώτο κλάσμα, ώστε να ικανποιηθεί το $\cos(x) = 1 - \ldots$. Επίσης αρχικοποιούμε τον εκθέτη στην τιμή 0 αντί για 1 *που ήταν στην συνάρτηση υπολογισμού του ημιτόνου*, ώστε να είναι πάντα ζυγός.

2.4.3. Σχεδιάγραμμα συναρτήσεων



3. **menu –** Άσκηση 2

3.1. menu.c

```c
#include <stdio.h>
#include "menu.h"


int main(int argc, char **argv)
{
    int a, b;

    printf("A: ");
    scanf("%d", &a);
    printf("B: ");
    scanf("%d", &b);
```

```
    menu(a, b);

    return 0;
}
```

3.2. menu-libs.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "menu.h"


void menu(int a, int b)
{
    int menuChoice;

    do
    {
        system("clear||cls");

        printf("[1] %d to the power of %d\n[2] %d! and %d!\
n[3] Number of combinations between %d and %d\n[4] Exit\n",
a, b, a, b, a, b);
        printf("Choice: ");
        scanf(" %d", &menuChoice);

        switch(menuChoice)
        {
            case 1:
                results(a, b, 1);
                break;
            case 2:
                results(a, b, 2);
                break;
            case 3:
                results(a, b, 3);
                break;
        }

        pause();
    } while (menuChoice != 4);

    results(a, b, 4);
}

void pause()
```

```
{
    do
    {
        printf("\nPress [Enter] to continue. . .");
        getchar();
    } while (getchar() != '\n');

}

void results(int a, int b, int menuChoice)
{
    static int numChoices = 0;

    switch(menuChoice)
    {
        case 1:
            printf("%d^%d = %d\n", a, b, power(a, b));
            break;
        case 2:
            printf("%d! = %d\n", a, factorial(a));
            printf("%d! = %d\n", b, factorial(b));
            break;
        case 3:
            printf("Combinations between %d and %d: %d\n", a,
b, combinations(a, b));
            break;
        case 4:
            printf("Total number of choices: %d\n",
numChoices);
    }

    numChoices++;
}

int power(int a, int b)
{
    return pow(a, b);
}

int factorial(int a)
{
    int i, fac;

    for (i = 1, fac = 1; i <= a; i++)
        fac *= i;

    return fac;
}
```

```
int combinations(int a, int b)
{
    return factorial(a) / (factorial (b) * factorial(a-b)));
}
```

3.3. menu.h

```
#ifndef HEADER_FILE
#define HEADER_FILE

void menu(int, int);
void pause();
void results(int, int, int);
int power(int, int);
int factorial(int);
int combinations(int, int);

#endif
```

3.4. Περιγραφή υλοποίησης άσκησης 2

3.4.1. Μεταβλητές

`a`: Ακέραιος Α που δίνεται από το πληκτρολόγιο
`b`: Ακέραιος Β που δίνεται από το πληκτρολόγιο
`menuChoice`: Αριθμός επιλογής από το μενού
`numChoices`: Σύνολο επιλογών από το μενού που έχει κάνει ο χρήστης
`fac`: Τιμή παραγοντικού

3.4.2. Λειτουργία του προγράμματος

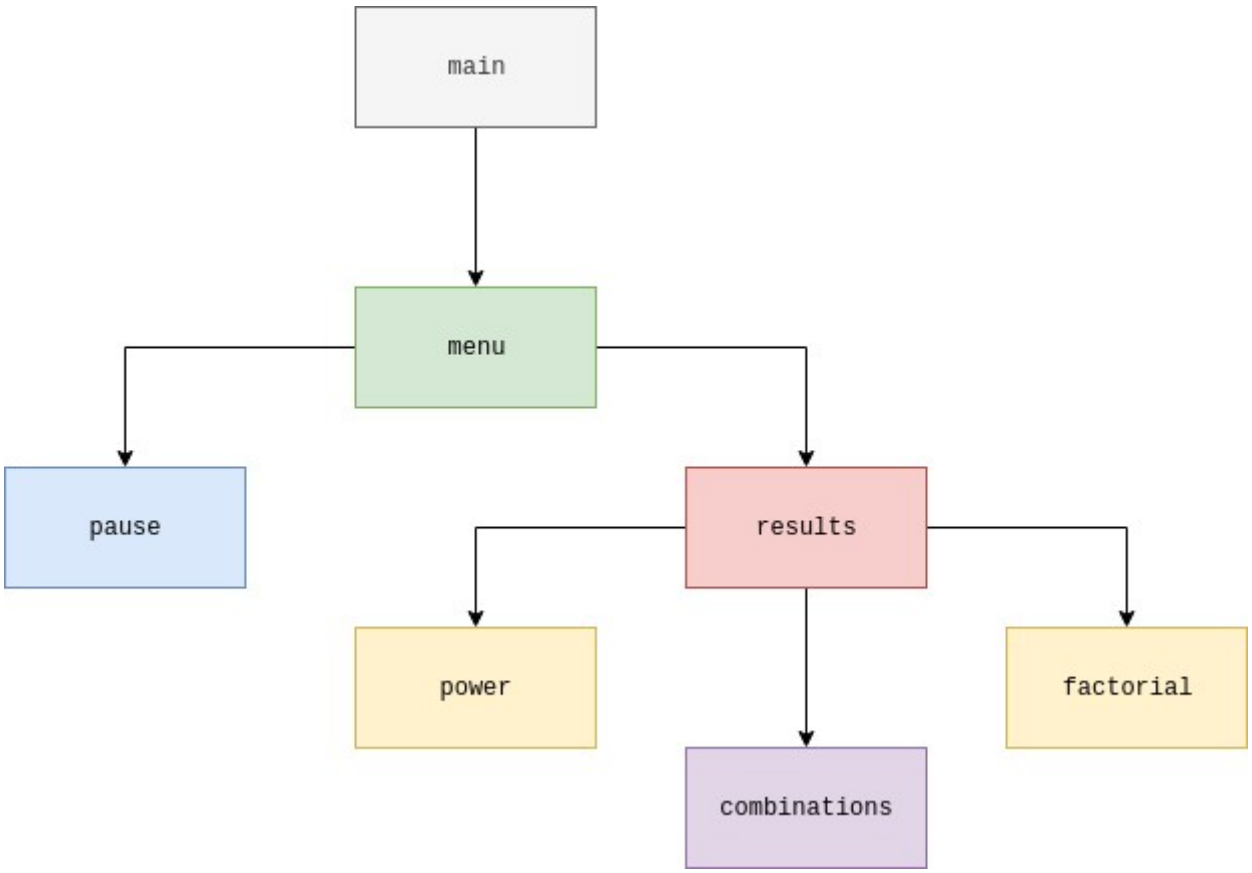
Στο `main.c`, δίνονται από το πληκτρολόγιο οι ακέραιοι Α και Β και παιρνάνε ως παράμετροι στην συνάρτηση `menu()` – από ‘κεί και πέρα τις υπόλοιπες λειτουργίες τις εκτελεί το `menu-libs.c`.

Αρχικά η συνάρτηση `menu()` καθαρίζει την οθόνη και δίνει στον χρήστη να επιλέξει από το μενού μία από τις τέσσερις επιλογές που του δίνονται (δυνάμεις, παραγοντικά, συνδυασμοί, τίποτα). Ανάλογα με την κάθε επιλογή του χρήστη καλείται η συνάρτηση `results()`, η οποία θα εμφανίσει στην οθόνη τα αποτελέσματα τής επιλογής του χρήστη.

Η συνάρτηση `results()` καλεί τρεις επιπλέον συναρτήσεις, την `power()`, `factorial()` και `combinations()`, οι οποίες βρίσκουν την δύναμη A^B , παραγοντικά $A!$ και $B!$ και συνδυασμούς A ανά B αντίστοιχα. Ο τύπος που εφαρμόζει η συνάρτηση `combinations()` είναι ο $K=\frac{A!}{B!\cdot(A-B)!}$ και πρέπει να ισχύει $1<B<A$.

Κατά την εμφάνιση κάποιου αποτελέσματος η συνάρτηση `pause()` “παγώνει” το πρόγραμμα, μέχρι ο χρήστης να πατήσει το πλήκτρο `Enter` ώστε να συνεχίσει η επιλογή από το μενού. Σε περίπτωση που ο χρήστης δώσει ως επιλογή τον αριθμό 4, το πρόγραμμα θα εμφανίσει πόσες επιλογές έκανε συνολικά και θα τερματιστεί αφού πατήσει `Enter`.

3.4.3. Σχεδιάγραμμα συναρτήσεων



4. Άσκηση 3

```
int MSum(int N)
{
    if (N == 1)
        return 1;
    return N + MSum(N - 1);
}
```

Η παραπάνω συνάρτηση υπολογίζει *αναδρομικά* το άθροισμα της ακόλουθης σειράς

$$N+(N-1)+(N-2)+(N-3)\ldots.$$

ώσπου ο αριθμός N να ισούται με 1 – δηλαδή σε κάθε κλήση της συνάρτησης ο N μειώνεται κατά 1 και προστείθεται στο άθροισμα `N + MSum(N – 1)`. Η διαδικασία αυτή θα τελειώσει όταν στην νέα κλήση της `MSum(N – 1)` η τιμή της παραμέτρου είναι 1, οπότε και θα επιστρέψει τον αριθμό 1 η συνάρτηση στην εντολή `return 1`, δηλαδή θα έχει τελειώσει ο υπολογισμός της σειράς.

Για παράδειγμα για $N=5$ η συνάρτηση θα κάνει το εξής:

$$5+(5-1)+(5-2)+(5-3)+(5-4)=5-4-3-2-1=15$$

και η τελική τιμή που θα επιστρέψει θα είναι 15 – το αποτέλεσμα τής αρχικής σειράς.

5. hanoi-tower – Άσκηση 4

5.1. hanoi-tower.c

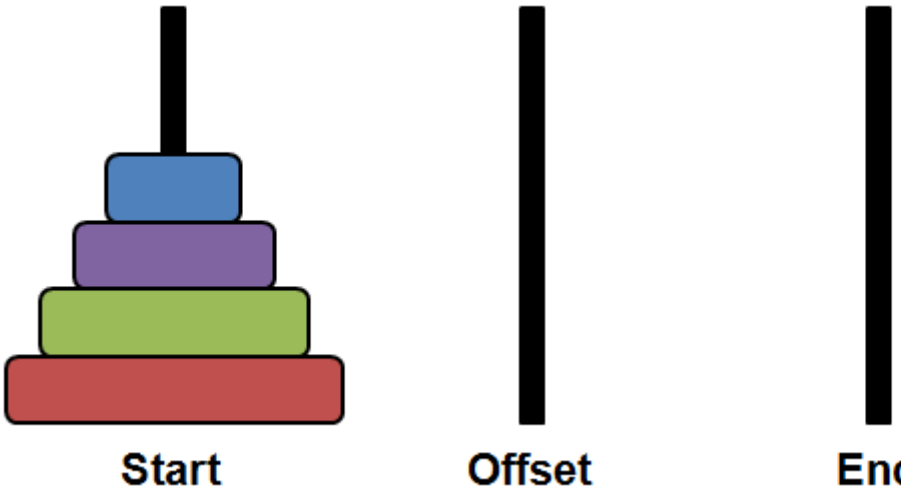
```
#include <stdio.h>

void moves(int, char, char, char);

int main(int argc, char **argv)
{
    int numDisks;

    printf("\t\tTower of Hanoi\n\n");

    do
    {
        printf("Number of disks: ");
        scanf("%d", &numDisks);
    } while (numDisks <= 0);
```

```
    printf("\n");
    moves(numDisks, 'A', 'B', 'C');

    return 0;
}

void moves(int numDisks, char tower1, char tower2, char
tower3)
{
    if (numDisks == 1)
        printf("Move disk 1 from tower %c to tower %c\n",
tower1, tower3);
    else
    {
        moves(numDisks - 1, tower1, tower3, tower2);
        printf("Move disk %d from tower %c to tower %c\n",
numDisks, tower1, tower3);
        moves(numDisks - 1, tower2, tower1, tower3);
    }
}
```

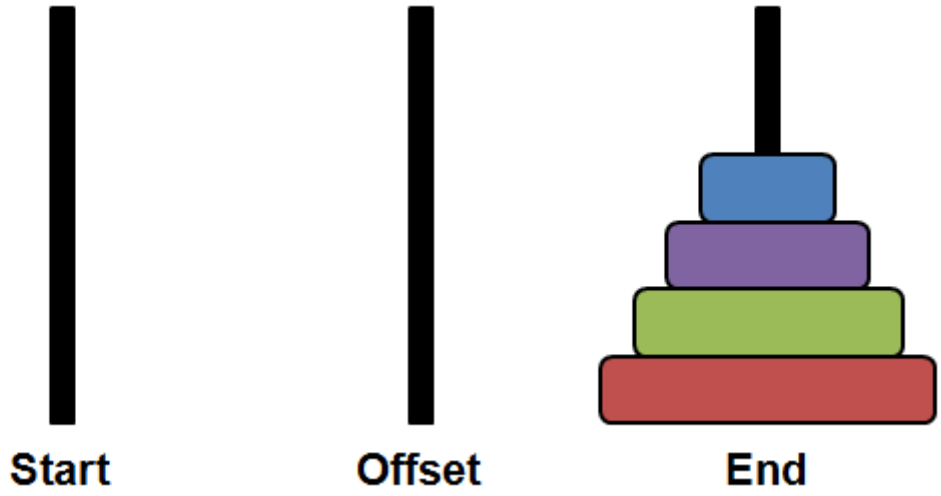
5.2. Περιγραφή υλοποίησης άσκησης 4

5.2.1. Κανόνες παιχνιδιού

1. Μόνο ένας δίσκος μπορεί να μετακινηθεί κάθε φορά από πύργο σε πύργο.
2. Μόνο ο πάνω μπορεί να μετακινηθεί κάθε φορά από έναν πύργο σ'έναν άλλον.
3. Ο κάθε δίσκος πρέπει να είναι μικρότερος από τον από κάτω του.



Σχήμα 1: Αρχική μορφή – Πηγή: ocf.berkeley.edu



Σχήμα 2: Τελική μορφή – Πηγή: ocf.berkeley.edu

5.2.2. Αλγόριθμος

Για n = αριθμός δίσκων:

1. Μετακινούμε τους n-1 δίσκους από τον πρώτο στον δεύτερο πύργο.
2. Μετακινούμε τον n δίσκο από τον πρώτο στον τρίτο πύργο.
3. Μετακινούμε τους n-1 δίσκους από τον δεύτερο στον τρίτο πύργο.

5.2.3. Μεταβλητές

`numDisks`: Αριθμός δισκών που θέλει ο χρήστης να δώσει από το πληκτρολόγιο
`tower1`: Πρώτος/αρχικός πύργος στην τριάδα των πύργων του Hanoi
`tower2`: Δεύτερος/βοηθητικός πύργος στην τριάδα των πύργων του Hanoi
`tower3`: Τρίτος/τελικός πύργος στην τριάδα των πύργων του Hanoi

5.2.4. Λειτουργία του προγράμματος

Στην συνάρτηση `main()` δίνεται από τον χρήστη ο αριθμός των δισκών που θέλει να έχει το παιχνίδι, και στην συνέχεια η συνάρτηση `moves()` βρίσκει ένα-προς-ένα τα βήματα ώστε να μεταφερθούν όλοι οι δίσκοι από τόν πρώτο πύργο (`tower1`) πρός τόν τρίτο πύργο (`tower3`), ακολουθώντας τους παραπάνω κανόνες.

Ο τρόπος που λειτουργεί η συνάρτηση `moves()` είναι ο εξής: Δέχεται ως παραμέτρους τον αριθμό των δίσκων, καθώς και τις ονομασίες των τριών πύργων. Στο σώμα της ελέγχει αν ο αριθμός των δίσκων είναι 1, και έτσι μεταφέρει τον δίσκο από τον πρώτο στον τρίτο πύργο. Σε περίπτωση που ο αριθμός των δίσκων είναι μεγαλύτερος του 1, η συνάρτηση εκτελεί *αναδρομικά* τα βήματα του παραπάνω αλγορίθμου. Παρατηρούμε όμως οτι οι παράμετροι των συναρτήσεων αλλάζουν –

αυτό γίνεται προκειμένου η `printf()` να τυπώνει κάθε φορά τον κατάλληλο πύργο ώς τελικό πύργο, και όχι πάντα τον `tower3`, και έτσι *με βάση τον αλγόριθμο* η πρώτη αναδρομική κλήση της συνάρτησης πρέπει να έχει ώς αρχικό πύργο τον `tower1` και τελικό τον `tower2`, και η δεύτερη ώς αρχικό πύργο τον `tower2`, και τελικό τον `tower3`.

6. Διευκρινήσεις

[2] Το πρόγραμμα ***hanoi-tower*** είναι ένα αρχείο μόνο οπότε για την εκτέλεσή του θα χρειαστεί μόνο *απλή μεταγλώττιση και εκτέλεση* – τα scripts *δεν* θα δουλέψουν.
[3] Τα directories είναι όλα στα *αγγλικά* ώστε να αποφευχθεί τυχόν πρόβλημα κατα την μετάβαση από directory σε directory στο terminal.
[4] Προσπάθησα *ώς ένα βαθμό* μέσω αυτής της εργασίας (αν και προφανώς δεν ήταν το θέμα της εργασίας) να κάνω και μία απόπειρα να πειραματιστώ με τα shell scripts και με το να χωρίζω τα προγράμματα σε ξεχωριστά αρχεία, σίγουρα με κάποιες ελλείψεις ή τυχόν λάθη.

7. Εργαλεία

Editor: Visual Studio Code
Shell: bash
Compiler: gcc
Συγγραφή: Libre Office Writer, draw.io
Γραμματοσειρές: Liberation Sans για τα *κείμενα,* `Liberation Mono` για τους *κώδικες,* και `Courier New` για τα *διαγράμματα*
Λειτουργικό σύστημα: Arch Linux