Menetrend

Kész, hajrá a ZH-n, azért aludjatok is.

A tervezettbõl eddig kész: 100%

1. gyakorlat

A gyakorlat elején gcc-vel fordítottunk egy "http://wiki.math.bme.huhello world"http://wiki.math.bme.hu programot. Hasznos tudni, hogy hogyan lehet így terminálból fordítani egy C programot.

Majd megtanultuk a Codeblocks / Codelite használatát, ezekbõl nem igazán tudnánk mit visszakérdezni, ezeket a kényelem kedvéért mutattuk meg.

A gyakorlat végén pedig egy nagyon egyszerû kiegészítendõ feladatot oldottunk meg, aminek annyi volt a lényege, hogy lássátok hogyan kell deklarálni változókat (pl: int x;) és hogyan kell használni egyszerû elágazást.

2. gyakorlat

Megtanultuk hogyan lehet scanf-el beolvasni:

```
int z;
scanf("%d", &z);
```

Ekkor még nem tudtuk miért kell az & jel a z elé, de mostmár tudjuk, hogy ez azért kellett, hogy a változóba beleírhassa a felhasználó által beírt értéket. Emlékezzük vissza miért nem lehet ezt pointer nélkül megoldani:

```
void swap(int x, int y){
    int temp = x;
    x = y
    y = temp;
}

void swap2(int* x, int* y){
    int temp = *x;
    *x = *y
    *y = temp;
}

int main(void){
    int a = 6;
    int b = 8;
    swap(a, b);
    printf("%d %d", a, b); // 6 8-at ír ki
    swap2(&a, &b);
    printf("%d %d", a, b); // 8 6-ot ír ki
    return 0;
}
```

Itt az elsõ swap függvén igaz úgy tûnik hogy megcseréli a változók értékét, de valójában csak a róluk készült másolatok értéket cserélte meg, és így a main-ben nem történt csere. Viszont a swap2 pointereken keresztül jól megcseréli a valódi **a** és **b** változó értékét.

A gyakorlat része volt még a ciklusokkal való ismerkedés:

```
for(inicializálás; feltétel; inkrementálás){
    utasítások
}

while(feltétel){
    utasítások
}

do{
    utasítások
} while(feltétel);
```

A for ciklust sokszor használtuk, tömbökön való végigjáráshoz, leszámolásokhoz, mindenféléhez. while-t kevésbé, de végülis helyettesíthetõ egy for(;feltétel;) ciklussal. do while-t pedig mégkevésbé használtuk, ennek ugye a sajátossága, hogy amikor belép a ciklusba elõször akkor nem ellenõrzi a feltételt, tehát ha egy mindig hamis feltételünk van a ciklus magja akkor is lefut egyszer.

Ciklusokkal megkerestük folyamatosan érkezõ számoknak a minimumát vagy maximumát, ez azon az egyszerû ötleten alapult, hogy mindig mentettük az eddig talált legkisebb számot (minimum esetén) és ezzel hasonlítottuk össze az éppen érkezõt. Majd aszerint hogy kisebb volt az eddigi legkisebbnél a most vizsgált vagy sem, beállítottuk ezt a legkisebbnek, vagy csak mentünk tovább.

3. Gyakorlat

A gyakorlat elején megtudtuk, hogy a változóink végesek, így nem tárolhatunk bennük akármekkora számot.

Írtunk for ciklusokkal pár feladatot, majd áttértünk a gyakorlat anyagára, amik a típusok voltak.

Tömböket is megtanultunk használni, egy egyszerû példa:

```
#include<stdio.h>

int main(void) {
    int t[20];     // Itt hozzuk letre a tombot, elore megadjuk a meretet
    int i;
    t[5] = 5 * 5;             // Igy tudunk egy elemenek erteket adni
    t[6] = t[5] + 11;         // Vagy felhasznalni ertekkent
    for(i = 0; i < 20; i++) { // Ez a ciklus vegzi el a munkankat
        t[i] = i * i;
    }
    return 0;
}
```

4. Gyakorlat

A gyakorlat két nagyon fontos dolgot vezetett be, a függvényeket és a pointereket.

```
v_ért fv_név(p_típus1 p_név1, p_típus2 p_név2){
    /*  */
    return érték;
}
```

Függvényeknek definíciója a visszatérési értékükkel kezdõdik, majd a függvény neve és zárójelben a bemeneti paraméterek. Mint minden blokkal rendelkezõ szerkezetnél itt is kapcsos zárójellel kell elkezdeni és bezárni a blokkot, azaz a függvény magját. Ha a függvényben valahol return-ölünk egy értéket, a függvény

visszatér, kilép, azaz a return utáni rész már nem fut le. Ezért az alábbi két függvény ekvivalens:

```
int fv(int a){
    if(a < 0){
        return -a;
    }
    else{
        return a;
    }
}

int fv2(int a){
    if(a < 0){
        return -a;
    }
    return a;   // Ez csak akkor fut le ha az if feltétele hamis volt, mert különben a függvény vi
}
```

Függvényeket lehet értékként is használni, így ha van egy átlag függvényünk, nem kell külön változóba elmenteni az általa visszaadott értéket:

```
printf("%lf", atlag(3.2, 6));
```

Pointerekrõl már írtam korábban, amit még megemlítenék, hogy a tömbök pointerek és a pointerek tömbök. Így ha t egy tömb vagy pointer, akkor t[0] és *t ekvivalens, továbbá, ha létezik i. indexû eleme, akkor t[i] és *(t+i) is ekvivalensek.

5. Gyakorlat

A gyakorlat elsõ felében dinamikus memóriakezelésrõl volt szó. Korábban csak fix méretû tömböket tudtunk létrehozni:

```
int t[20];
```

Ha dinamikusan foglalunk le memóriát akkor bármilyen pozitív egész változó méretû tömböt lefoglalhatunk:

```
int m;          // ebbe olvassuk be a tömb méretét
scanf("%d",&m);
int *vec = (int *)malloc(m * sizeof(int)); // itt foglaljuk le a memóriát a tömbnek
```

Itt még megemlíteném, hogy igaz gyakorlatokon csak a malloc-ot használtuk, így a többit nem fogjuk kérdezni gyakorlati feladatokban. De elméleti feladatba bõven belefér pl, hogy mire való a calloc, realloc stb.

A gyakorlat másik felében raktár kezelõ függvényeket írtunk, ezeknek a megoldása megtalálható a megfelelõ házifeladat oldalán.

6. Gyakorlat

A gyakorlat 3 dologról szólt, file kezelés, argumentumok és struktúrák.

A file kezelést nem nehéz elsajátítani, ha a scanf és printf-ekkel már tisztában van az ember:

```
FILE* fp;            // Letrehozzuk a file pointerunket
fp = fopen("test.txt", "w");  // Megnyitjuk a test.txt-t irasra
```

Ezek után a sorok után használhatjuk az fprintf függvényt, hogy írjunk a test.txt file-ba. Ha olvasni szeretnénk egy file-ból akkor a "http://wiki.math.bme.huw"http://wiki.math.bme.hu helyett "http://wiki.math.bme.hur"http://wiki.math.bme.hu-et írjuk, ekkor az fscanf használható. Ha nem szeretnénk felülírni a file-t akkor használhatjuk az "http://wiki.math.bme.hua"http://wiki.math.bme.hu-t, ami a file végére ír.

```
fprintf(fp, "Most irunk a file-ba.\n");
```

Ha "http://wiki.math.bme.hur"http://wiki.math.bme.hu-el van megnyitva:

```
fscanf(fp, "%d", &z);          // Kiolvasunk a file-bol egy int-et
```

Az argumentum kezelésrõl a módosított main függvény fejét mindenképp érdemes megjegyezni:

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

Ha így definiáljuk a main függvényt, akkor argv[0]-ban a program neve lesz, argv[1]-tõl pedig a programnak adott argumentumok találhatók. Ha pl "http://wiki.math.bme.hu./prog_neve teszt 20"http://wiki.math.bme.hu paranccsal futtatjuk a prog_neve nevû programunkat, akkor az argv[1] a teszt string lesz, az argv[2] pedig a 20 string, tehát nem a 20 szám, hogy számként használhassuk az atoi függvényt kell rá használnunk:

```
int m = atoi(argv[2]);
```

Az argc-ben amit igazából nem használtunk az argumentumok száma szerepel, szóval az elõzõ példában pl 2.

A gyakorlat vége struktúrákról és a typedef-rõl szólt.

```
typedef int egesz;
```

Ezután a sor után használhatjuk az egesz kifejezést bárhol int helyett, pl:

```
egesz i;
egesz n;
```

Struktúrák több változót zárnak egységbe, például létrehozhatunk egy dátum változót, amiben tároljuk az évet, hónapot és napot:

```
struct Datum{
    int ev;
    int honap;
    int nap;
};
```

Ezután a következõ képpen hozhatunk létre egy ilyen változót és állíthatjuk be az adattagjait:

```
struct Datum d1;
d1.ev = 2013;
d1.honap = 4;
d1.nap = 4;
```

Amint látható a struct kulcsszót is ki kell írni mikor deklarálunk egy változót, ezt nem igazán szeretjük, így typedef-el szoktuk "http://wiki.math.bme.humegjavítani"http://wiki.math.bme.hu, az elõzõ struktúra definíció helyett:

```
typedef struct{
    int ev;
```

```
    int honap;
    int nap;
} Datum;
```

Ezután létrehozható Datum változó és ugyanúgy mûködik mint korábban:

```
Datum d1;
d1.ev = 2013;
d1.honap = 4;
d1.nap = 4;
```

Ami a typedef-nél történik, hogy azt az egész struktúrát nevezzük el Datum-nak, pont mint amikor az int-et neveztük el egesz-nek.

A gyakorlat végén egy Vektor struktúrával dolgoztunk, írtunk hozzá függvényeket, a házi is errõl szólt, így a kapcsolódó házi oldalán megtalálható pár függvény megoldása.

Konzultáció feladat

A feladatnak nem az volt a célja, hogy egy átlagos ZH feladatot mutasson be, hanem hogy szinte minden szerepeljen benne ami a gyakokon szerepelt, így jól tudjunk magyarázni a segítségével.

A feladat

Írjunk programot, melynek 2 argumentuma van, egy file név és egy szám. Az adott file-ban soronként két egész szám található, pontok a síkon, az adott szám pedig azt határozza meg hogy hány darab ilyen pont található a file-ben.

Hozzunk létre egy struktúrát amiben tárolni tudunk egy ilyen pontot. Majd olvassuk ki a file-ból mindet egy dinamikusan foglalt tömbbe. Végül írjuk ki azokat a pontokat amiknek a koordinátáik összege fibonacci szám.

Megoldás tervezése

A struktúránknak két egészet kell tárolnia. Érdemes lenne megírni egy olyan függvényt mely eldönti egy adott egész számról, hogy fibonacci-e vagy sem, majd ezt felhasználni a vizsgálatkor. A dinamikus tömb foglalásnál az általunk létrehozott stuktúrákat tároló tömböt kell lefoglalnunk, ez semmiben nem különbözik a beépített típusok lefoglalásától.

Elõször írjuk meg a fibonacci vizsgáló függvényt. Majd kezeljük le az argumentumokat, nyissuk meg a file-t. Végül olvassuk ki a file-ból a pontokat és nézzük meg hogy a koordinátáik összege fibonacci-e.

A megoldás

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

// A függvény 0-t fog adni ha nem fibonacci az adott szám, és 1-et ha az. C-ben 0 a hamis és 1 az
int fibonacci_e(int n){
    int x = 1; // 0.
    int y = 1; // és 1. fibonacci szám, y-ban lesz mindig a nagyobb, x-ben az elõzõ fibonacci szám
    if(n == 0 || n == 1) // Úgy döntöttünk, hogy a 0 és 1 fibonacci, így egyet adunk vissza rögtön
        return 1;   // Itt igaz nem írtam {}-eket, de nem is kell, ha nem teszem ki, akkor az if a
```

```
    while(y < n){   // Megkeressük azt a fibonacci számot ami egyenlõ vagy nagyobb a kapott számun
        int temp = y;   // Elmentem az y értéket, így tudom csak megoldani, hogy majd x-nek értékü
        y = y + x;      // F_{i+1} = F_{i} + F_{i-1}
        x = temp;       // F_{i}
    }

    if(y == n)          // Ha azért álltunk meg mert egyenlõ volt a kapott szám egy fibonacci-val
        return 1;       // Akkor igazzal térünk vissza
    else                // Különben hamissal
        return 0;       // Itt ugye az else maga nem is kellett volna, lehetett volna írni a retur
                        // mert ha y == n igaz volt, akkor úgyis kilép a függvény, szóval a return
}

// A struktúránk 2 int-et tárol:
typedef struct{
    int x;
    int y;
} Pont;

// Argumentumos main-t kell írnunk mert argumentumként kapunk két dolgot is
int main(int argc, char* argv[]){
    FILE* fp;   // Ebbe nyitjuk majd meg a file-t
    int i;      // Ciklusváltozó
    int n;      // Ebbe mentjük a pontok darabszámát (2. argumentum)

    fp = fopen(argv[1], "r");   // Megnyitjuk az elsõ argumentumként kapott számot olvasásra

    n = atoi(argv[2]);      // A második argumentumként kapott számot elõbb string-bõl int-é kell

    // Foglaljuk le a tömböt a pontoknak
    Pont* t = (Pont*)malloc(n * sizeof(Pont));  // Ahogy mondtam a lefoglalás nem különbözik a kor

    // Olvassuk be a file-ból a pontokat:
    for(i = 0; i < n; i++){             // n-ig megyünk mivel n darab pontunk van
        fscanf(fp, "%d", &(t[i].x));    // Elõször az x koordinátát olvassuk be. t[i] az i. pontun
        fscanf(fp, "%d", &(t[i].y));    // Aztán ugyanígy beolvassuk az y koordinátát is
    }

    // Egy külön ciklusban megnézzük hogy melyik pontok koordinátáinak az összege fibonacci és eze
    for(i = 0; i < n; i++){
        if(fibonacci_e(t[i].x + t[i].y)){       // A függvény vizsgálja a számokról hogy fibonacci
            printf("(%d, %d)\n", t[i].x, t[i].y);   // A kiírás pl így fog kinézni: (3, 2)
        }
    }

    return 0;       // return 0 ahogy a main végén szokásos
}
```

Példa futtatás linux-on, terminálból: ./program pontok.txt 10

Ahol a pontok.txt pl a következõ:

```
6 7
4 5
3 5
7 4
7 3
1 1
4 6
7 3
23 5
75 2
```

Felraktam a megoldást és a pontok.txt-t: konzult.c, pontok.txt

Gyakorlati feladatok megoldásai

Max

link a gyakra

Írj programot, ami pontosan 5 számot kér be a felhasználótól, majd kiírja közülük a legnagyobbat. (Érdemes valami értelmes szöveggel pl: "http://wiki.math.bme.huA legnagyobb: %d"http://wiki.math.bme.hu)

Megoldás

```
#include<stdio.h>

int main(void){
    int legnagyobb;     // Ebbe tároljuk majd mindig az eddig talált legnagyobbat
    int szam;           // Ebbe tároljuk az éppen megkapott számot
    int i;  // Ciklusváltozó

    scanf("%d", &legnagyobb);   // A legelsõ számot a legnagyobba olvassuk be, mert 1 elemû halmaz
    for(i = 0; i < 4; i++){     // 4-et kell márcsak beolvasnunk, mert az elsõt már beolvastuk
        scanf("%d", &szam);     // Beolvasunk egy számot a szam változóba
        if(szam > legnagyobb){  // Ha nagyobbat találunk az eddigi legnagyobbnál
            legnagyobb = szam;  // Akkor beállítjuk a legnagyobbat erre
        }
    }
    // Miután ez lefutott a legnagyobb-ban biztosan a legnagyobb beadott szám lesz
    printf("A legnagyobb: %d\n", legnagyobb);
    return 0;
}
```

min.c

Prímtényezõ keresés

link a gyakra

Írj programot, ami megkeresi egy a felhasználó által adott szám prímtényezõit és sorban kiírja azokat. (A hiba elkerülése végett elõször vizsgáljuk meg, hogy nem 0-t vagy 1-et kaptunk-e.)

Nem kell bonyolultra gondolni azonnal, meg lehet oldani úgy is, hogy egyesével megpróbáljuk elosztani az adott számunkat 2-tõl kezdve egyesével haladva egész számokkal, amíg 1-hez nem jutunk.

Emlékezzünk, hogy a maradék képzés (modulo) jele C-ben is a %

Megoldás

```
#include<stdio.h>

int main(void){
    int n;  // Ebbe tároljuk majd a számot
    int i;  // Ciklusváltozó (bár nem olyan lesz mint általában)
```

```
    scanf("%d", &n);

    if(n == 0 || n == 1){       // Ha 0-t vagy egyet adtunk
        printf("0 vagy 1-hez nehez primosztokat mondani");  // Akkor figyelmeztetjuk a felhasználó
        return 0;       // Majd befejezzük a program futását (visszatérünk a main-bõl)
    }

    for(i = 2; i <= n;){ // i-t nem léptetjük minden ciklusléptésben, 2-rõl kezdünk mert nem szeret
        if(n % i == 0){     // Ha n osztható i-vel
            printf("%d\n", i);      // Akkor kiírjuk mert prímtényezõ
            n = n / i;              // És elosztjuk vele n-et
        }
        else{
            i++;        // Ha n nem osztható i-vel, akkor léptetjük i-t, nagyobb prímosztóval prób
        }               // Összetett számokkal nem lesz gond, hisz az õ prímosztóikkal korábban má
    }
    return 0;
}
```

primtenyezo.c

Intervallumba esõk kiírása

Írjatok függvényt, ami bemenetként kap egy double tömböt és két int-et és kiírja printf-el a tömb azon elemeit amik beleesnek az adott int-ek által meghatározott intervallumba.

Függvényeknek tömböt a következõ módon (is) átadhattok (az elsõ paraméter a tömb):

```
void fv(int t[], int v){...}
```

A main függvényetek megírásában használjátok a következõ tömböt az egyszerûség kedvéért (így lehet egyszerre megadni egy tömb elemeit, a []-be nem is kell feltétlen beleírni az elemszámot):

```
double tomb[10] = { 2.5, 7.6, 1.2, -8.9, 4.6, -4.8, 12.4, 1.1, 3.5, -6.7 };
```

A main-ben ne felejtsétek el letesztelni a függvényeteket!

Megoldás

```
#include<stdio.h>

void interv(double t[], int min, int max, int n){      // t a tömb, min és max az intervallum két
    int i = 0;      // Ciklusváltozó
    for(i = 0; i < n; i++){         // Végigmegyünk a tömbön
        if(min < t[i] && t[i] < max){       // Nyílt intervallumot nézünk, a min < t[i] < max nem
            printf("%lf\n", t[i]);      // Ha intervallumba esõt találtunk kiírjuk, a double ugye
        }
    }
}

int main(void){
    double tomb[10] = { 2.5, 7.6, 1.2, -8.9, 4.6, -4.8, 12.4, 1.1, 3.5, -6.7 }; // Ezzel a tömbbel
    interv(tomb, 1, 4, 10);     // tömb, min, max, tömbméret
    return 0;
}
```

intervallum.c

Tömb min és max

link a gyakra

Írj függvényt, ami megkeresi egy adott double tömb minimumát és maximumát is, és pointerek segítségével adja vissza.

Bemenet: double tömb, double min pointer, double max pointer

Használhatjátok a korábbi double tömböt:

```
double tomb[10] = { 2.5, 7.6, 1.2, -8.9, 4.6, -4.8, 12.4, 1.1, 3.5, -6.7 };
```

Érdemes még meggondolni, hogy amikor függvényekben dolgozunk tömbökkel, akkor hasznos lehet átadni a tömbök méretét a függvénynek, hogy általánosabb függvényt írhassunk.

Megoldás

```
#include<stdio.h>

void minmax(double t[], double* min, double* max, int n){    // tömb, min pointer, max pointer, tö
    int i;  // Ciklusváltozó

    *min = t[0];    // Adunk kezdeti értéket a min és max-nak
    *max = t[0];    // min és max pointer, szóval az általuk mutatott memóriaterületre *min és *ma
    for(i = 1; i < n; i++){ // Végigmegyünk a tömbön, elég az 1-es indexûtõl menni, mert a 0-st má
        if(t[i] < *min){    // Ha kisebbet találtunk mint az eddigi minimum
            *min = t[i];    // Akkor ez az új minimum
        }
        if(*max < t[i]){    // Ha nagyobbat találtunk mint az eddigi maximum
            *max = t[i];    // Akkor ez az új maximum
        }
    }
    // Mikor lefutott ez a for ciklus, a min és max által mutatott memóriaterületeken a minimuma é
}

int main(void){
    double tomb[10] = { 2.5, 7.6, 1.2, -8.9, 4.6, -4.8, 12.4, 1.1, 3.5, -6.7 };
    double minimum;     // Létre kell hozni a változókat amibe majd menti a minmax függvény az ért
    double maximum;
    minmax(tomb, &minimum, &maximum, 10);   // tömb, min pointer, max pointer, tömb méret

    printf("Min: %lf\n", minimum);
    printf("Max: %lf\n", maximum);
    return 0;
}
```

minmax.c

Bet?raktár-kezel?

link a gyakra

Megoldás

beturaktar_kesz_kommentezve.c

Átlaghoz közel file-al

- Írjátok meg az 5. gyakorlat 1. feladatát úgy, hogy file-ból olvassa be a bemeneteket, és egy másik file-ba mentse a kimenetet.
- A program argumentumként kapja meg:
  - A beolvasandó file nevét
  - a kiírandó file nevét,
  - azt, hogy hány darab szám található a file-ban
- Az elõzõ gyakorlat 1. feladatának egy megoldása itt található:
  - Megoldás
  - Megoldás kommentezve

Megoldás

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

int main(int argc, char* argv[]){
    int i; // ciklusváltozó
    int m; // ebbe olvasom be a tömb méretét
    double *t; // ez lesz a tömbböm, most ugye még csak egy pointer, de foglalunk neki helyet dina
    double osszeg; // az összeg tárolására
    double atlag; // az átlag tárolására
    double legkoz; // ebbe tárolom majd az eddigi legközelebbet
    FILE* fp1;  // ebbõl olvassuk a számokat
    FILE* fp2;  // ebbe mentjük a legközelebbit

    fp1 = fopen(argv[1], "r"); // az elsõ argumentumként kapott file-t olvasásra nyitjuk meg
    fp2 = fopen(argv[2], "w"); // a második argumentumként kapott file-t írásra nyitjuk meg

    m = atoi(argv[3]);  // a számok darabszámát a 3. paraméter tárolja, de ezt még számmá kell ala
    t = (double*)malloc(m * sizeof(double));

    for(i = 0; i < m; i++){
        fscanf(fp1, "%lf", &t[i]); // itt olvasom be az m darab lebegõpontos számot, a &t[i] helye
    }

    atlag = 0; // kezdõérték az átlagnak
    for(i = 0; i < m; i++){
        osszeg = osszeg + t[i]; // kiszámolom az összeget, ezt akár az elõzõ for ciklusban is tehe
    }

    atlag = osszeg / m; // az átlag

    legkoz = t[0]; // kell valamilyen kezdõértéket adni neki, legyen ez a legelesõ
```

```
    for(i = 0; i < m; i++){
        if(abs(legkoz - atlag) > abs(t[i] - atlag)){ // ha az eddig talált legközelebbi távolabb v
            legkoz = t[i]; // akkor a most vizsgált közelebb van az átlaghoz.
        }
    }

    fprintf(fp2, "%lf", legkoz); // végül kiírjuk a legközelebbit a file-ba

    return 0;
}
```

legkozelebb_kommentezve_file.c, bemenet.txt

Egyszer? struktúra kezelés (vektorok)

link a gyakra

Megoldás (nem mind)

vektor.c