C++ JEGYZET

A Kezdetek

Tartalom

Programkeret, amit mindig be kell írni:

```
#include <iostream>

//ide kerül az összes többi fejléc, amire szükségünk lehet

using namespace std;

int main()
{
//ez itt a program törzse
    return 0;
}
```

A megfelelő fejléc nélkül bizonyos parancsokat nem fog értelmezni a program!

A return mögötti érték megadja, hogy a függvény újbóli elindulásakor milyen értékkel rendelkezzen. Az int main gyakorlatilag a legfőbb függvény. Ha a beugróban olyasmit kérnek, hogy a függvény pl.: mindig 1 értéket vegyen fel, akkor a return mögötti értéket 1-re írjuk át.

Deklarációk:

```
int i=0; //egész szám deklarálása null kezdőértékkel
int a=0,b=0; // több egész szám deklarálása null kezdőértékkel
double c=0; // valós szám deklarálása null kezdőértékkel
char k=""; // karakter deklarálása
string s; // szöveg deklarálása <string>!
bool l; // logikai értéket tárol
ifstream bf ("nev.txt"); //befájl <fstream>!
ofstream kf("neve.txt"); //kifájl<fstream>!
stringstream sz; //fájlszerű szöveg <sstream>!
int tomb[10]; // egész számokat tartalmazó tömb deklarálása
```

Figyeljünk oda, hogy az egészekkel (integer) való műveletek esetén az osztás nem zárt az egész számok halmazára, ezért valós (double) számokkal operáljunk tovább!

A bool logikai értéket tárol, értéke IGAZ (true) vagy HAMIS (false) lehet, a false megfelel a nulla, a true a minden más integernek, ellenben olvashatóbbá és könnyebbé teszi a program kezelését.

A befájl fájlt olvas be, a kifájl fájlt teremt a zárójelben megadott névvel, ezt se felejtsük el.

A stringstream egy fájlszerűen kezelt szövegváltozó, a program az utolsó karakterig nézi jobb esetben.

A string, ifstream, ofstream, stringstream esetén ne felejtsük el a megfelelő fejléceket. TFH mindezek használva lesznek, ekkor így néz ki a program:

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
//ide kerül az összes többi feiléc, amire szükségünk lehet

using namespace std;

int main()
{

int i=0; //egész szám deklarálása null kezdőértékkel
int a=0,b=0; // több egész szám deklarálása null kezdőértékkel
double c=0; // valós szám deklarálása null kezdőértékkel
char k=""; // karakter deklarálása
string s; // szöveg deklarálása <string>!
bool l; // logikai értéket tárol
ifstream bf ("nev.txt"); //befájl <fstream>!
ofstream kf("neve.txt"); //kifájl<fstream>!
stringstream sz; //fájlszerű szöveg <sstream>!
int tomb[10]; // egész számokat tartalmazó tömb deklarálása

// ez itt a program további része

 return 0;
}
```

Elágazás:

Az elágazásnál a program bizonyos feltétel megléte esetén ezt, nem megléte esetén azt a parancsot hajtja végre (vagy semmit). Például: „Ha esik az eső, viszek neked esernyőt,” Ezen elágazás esetén, csak akkor történik meg az esernyő vitele, ha az esik az eső igaz értéket kap. Ezt bővíthetem azzal, ha a hamisság esetén is csináltatni szeretnék valamit. A különbség az else utáni és a teljes elágazás utáni parancssor között az, hogy az else csak akkor történik meg, ha hamis a feltétel, ha az elágazás után írjuk a parancsot, akkor igaz feltétel esetén, az igazon belüli parancs is és az elágazás utáni is végrehajtódik. Hamis feltétel esetén pedig csak, amit az elágazás után írunk.

```
if (//feltétel
)
{//ide kerül a feltétel megléte esetén végrehajtandó parancssor
}
else
    {//ide kerül a feltétel hamissága esetén végrehajtandó parancssor

}
```

A feltételnél relációkat vizsgálunk, erre a megfelelő utasítások a következők:

```
a == b // egyenlőségvizsgálat
a > b   b>c //nagyobb
a <= b // kisebb, vagy egyenlő
a != b // nem egyenlő vizsgálata
```

Amikor bonyolítjuk és több feltétel meglétét szeretnénk vizsgálni elkerülhetetlen egyéb logikai művelet használata. (és vagy) alkalmazása, megfelel a nulladrendű logika szabályainak, and/or a legegyszerűbb, de velük ekvivalens a $and \iff \&\&$ $or \iff ||$

Ciklusok:

Alapvetően kétféle ciklusra lesz szükségünk, a for-ra és a while-ra. A for használata akkor célszerű, amikor konkrét futóindexet kell futtatnunk, szummázásnál, produktumnál, soroknál etc. A while-nál ezzel szemben nincsen szükség futóindexre.

```
for (//futóindex deklarálása
     ;//feltétel
;//futóindex futtatása
    )
    {
        //végrehajtandó parancsor
    }
```

Célszerű a cikluson belül deklarálni a futóindexet, mert akkor a későbbiekben egy másik cikluson kívüli ciklusban ismét felhasználható a futóindex megnevezése. Ez mindaddig, míg a feltétel teljesül, visszaugrik a bajuszvégtől a for elejéig. KEZDŐÉRTÉKET ne felejtsünk adni, illetve futtatni az indexet.

$$i = i + 1 \iff i++$$
$$i = i - 1 \iff i--$$

```
while (//feltétel
        )
{
    // végrehajtandó parancssor
}
```

Sima beolvasás, sima kiírás:

```
cin >> a; //beolvassa az a változó értékét
cout << a << endl; // kiírja az a változó értékét és sort tör
```

Fontosabb matematikai műveletek:

```
abs(a) // a változó abszolútértékét adja meg <cmath>
pow(a,b) // a-t b-re emeli <cmath>
sqrt(a)        // gyököt von <cmath>
a%b // a b-vel való osztásának maradéka
```

A <cmath> fejléct ne felejtsük el abszolútérték, hatvány, gyök esetén.

Fájl használata: FEJLÉC!

Ha adott egy txt fájlunk, akkor ezzel különböző műveleteket tudunk csinálni, ehhez deklarálnunk kell egy fájl változót, amiben hivatkozunk a fájlra, ahogyan azt fentebb is láthattuk. Fájl tartalmát elmenthetjük, attól függően, hogy számra, vagy szövegre van szükségünk belőle. Ezt aképp tehetjük meg, hogy a fájlból beolvasunk annyi karaktert, amennyi a keresett szám előtt van, majd egy számot

olvasunk be, és így megyünk tovább, amíg azt nem kérjük, hogy álljon meg, pl. ugorjon a következő sorra (getline). Gyakorlatilag karakterenként futunk végig (egyelőre), kivéve, amikor úgy látjuk, hogy szám van és mi azt számként mentenénk el. Például, amit a mozinál is alkalmaztunk:

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream f("mozi.txt"); //deklarálja az f nevű változót, melv a mozi nevű fájlt kezeli
    char kar; //deklarál egy karaktertípusú változót
    int b=0,c=0,s=0,ma=0,
    string z,d;


    while (f.good()){ //addig megy a ciklus, amíg a fájl értéke jó (azaz nincs vége - utolsó enterig)
        f >> kar >> b >> kar >> kar >> c >> kar;
        //beolvas a kar változóba egy karaktar
        //beolvassa azokat, amik a karakterek után vannak és számok (egy számként)
        //felülírja a karaktert arra, ami az előző utsó számjegy után jön
       //következő karakterre írja felül a karaktert
       //beolvassa a következő számjegyeket egyetlen számként
       //majd az utsó karakter jön
        getline(f,z); // a z sztringváltozóba elmenti a maradék összes karaktert a sortörésig
        s=c-b; // a mozis feladatban ez fogja kiszámolni a hosszt ( a két beolvasott számból)
        if (s>ma){ //ez a maximumkeresés tétele
            ma=s;
            d=z;
        }
        cout << s << z; //kiírjuk a nekünk éppen szükséges, beolvasott, megtalált kiszámolt dolgokat
    return 0;
}
```

Az a legbiztosabb, ha a fájl legvégébe mindenképpen rakunk entert, mert máskülönben az f.good függvény nem értelmezett és nem találja meg a fájl legvégét.

Fájlt mi is létrehozhatunk, ekkor a deklarációnkban ifstream helyett ofstreamet kell használni. A fájlba írás ugyanúgy működik, csak a másikirányú kacsacsőrrel. Erre jó példa volt a sakktáblánk, vagy, ha akarjuk az aldeterminánsok előjele ;)

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ofstream f("sakktabla.txt"); //fájlteremt a megadott néven
    for (int i=0;i<8;i++) // ez egy jól látható for ciklus, a sorokért felel
    {
        for (int j=0; j<4;j++) //emez pedig a soron belüli részekért
        {
    if (i%2!=0) {
       f <<"* ";
    }
    else{
        f <<" *";
    }
    }
    f<<endl; //a sortörés is ugyanúgy működik fájlbaírás esetén is
    }
    return 0;
}
```

Emellett órán alkalmaztuk a maximumkeresés tételét, és a bizonyos események leszámlálását.

Maximumkeresés:

Ebben az esetben általában egy ciklussal dolgozunk, ami végigfut valamilyen adatsoron, s ebben kell nekünk elraktározni a legnagyobb értéket. Ez akár az adatsoron belüli műveletekre is vonatkozhat, hogy a műveletek közül arra kíváncsi, amelyik a legnagyobb értéket adta, ez a komplexitás volt a moziban is megtalálható.

A ciklusunkból az adatot egy elágazással el kell mentenünk egy cikluson kívüli változóba, abban az esetben, ha az adatunk nagyobb, mint az előzőleg elmentett, azaz a változó. Így a következő iterálás után is megmarad az épp legnagyobb érték, mindaddig, míg egy annál nagyobb nem következik. Ekkor a nagyobbik elmentésre kerül. A ciklus lefutása után a maximumértékkel azt kezdünk, amit csak szeretnénk. Az alábbi programrészlet csak pozitív számok esetén értelmezett, így a következő óra ad pontosabb útmutatást.

```
#include <iostream>

using namespace std;

int main()
{ int b=0,maximum=0;
   for (int i=0;i<8;i++) {
   cin >> b; //beolvas egy számot
   if (b>maximum)
    //megnézi, hogy nagyobb-e
    {
    maximum=b; //itt mentjük el az értéket, mert nagyobb mint az előző
   }
   }
   cout << maximum;
    return 0;
}
```

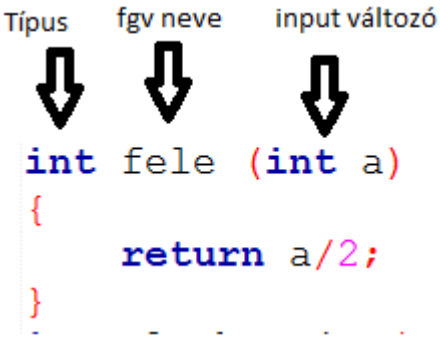
Remélem érthetőre és hasznosra sikeredett. (Néhány műveletet most nem tisztáztunk, de ha segítség kell, vagy eljutunk oda, arra is sor kerül, pl.: szöveghossz, karakterkeresés, nagybetű etc.)

Jó tanulást kívánok mindenkinek! ☺

A jegyzet néhol szorosan kapcsolódni fog az előző órai jegyzethez, mert a kettő egymásra épül, és szerves részét képezik egymásnak, így együttesen tanulandóak.

Függvény:

A függvény egy komplett programrészlet, maga a program is egy függvény. A függvény célja, hogy leegyszerűsítse a programunkat, könnyen lehessen kezelni és javítani. Ha van egy adott programrészlet, amit sokszor meg kell ismételnünk, gyakorlatilag függvény nélkül belebukhatunk. Egy hibát vétünk, akkor azt a sok ezer hibát ki kell javítanunk. Valamint ha nem használnánk függvényt, a programunk érthetetlenül sok sorrá válna.

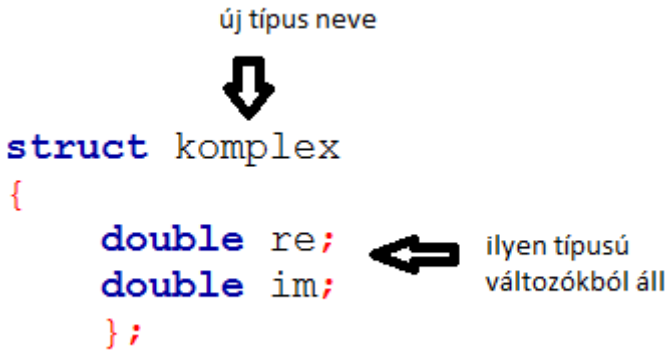


A függvényeket a főfüggvényen (main) kívül kell létrehozni. A függvény típusa a függvény névleges értékére vonatkozik. Erre olyankor van szükség, amikor a függvénnyel egyéb dolgunk lesz még. Láthattunk ilyenekre példákat. (egyenlőségvizsgálat, szövegkiírás etc.) A függvény névleges értékét, azaz a visszatérési értéket a „return valami” adja meg. A függvény neve után az input változók szerepelnek sima vesszővel elválasztva. Az input változók típusát ne felejtsük el megadni! Ennek meg kell egyeznie azzal, amit később a főfüggvényben elvárnánk.

Az input változó olyan változó, mely a függvényen belül van értelmezve. Azért van rá szükség, hogy így a megfelelő függvényen kívüli változókat meghívhassuk és a függvényen belül tudjunk operálni. A program később függvényhíváskor ezt az input változót teszi egyenlővé azzal a függvényen kívüli változóval, amelyek nevét a zárójelbe megfelelő sorrendben írjuk. Ez addig tart, amíg a függvény véget nem ér, következő meghíváskor ismét egyenlővé tehetjük egy másik függvényen kívüli változóval.

```
int b;
 cin >> b;
    int a1 = fele(6);
    int a2 = fele(b);
```

Struct



Új típus létrehozására való. Később ezen típussal is deklarálhatunk változókat. Ez szintén a programunk leegyszerűsítésére szolgál. Ezt láthattuk a Complex változó esetén is. Hiszen ez a változó két double típusból áll, a valós (real) értékből és kapcsolódik hozzá egy imaginárius (imaginary) érték is. Struct nélkül állandóan ezzel a két változóval kéne dolgoznunk. Ellenben, ha létrehozunk egy új típust, akkor később elég lesz egyetlen változót használnunk.

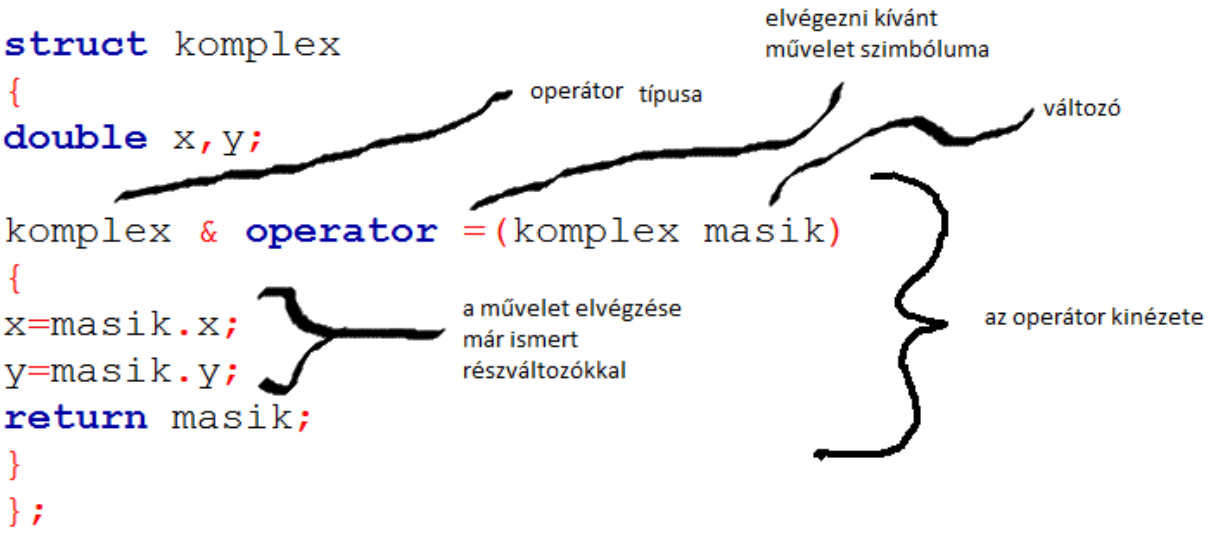
A struct törzsében deklarálnunk kell, hogy milyen típusú változókból áll az új típusunk. Itt adjuk meg, hogy hogy néz ki eredetileg is ismert típusokból összerakva az új típus. Az új változó részeire hivatkozhatunk a következőképpen:

```
komplex b;
   b.re=1;
   b.im=2;
```

Valtozo_neve.reszvaltozo_neve

Operátor

Hogy még esztétikusabbá tegyük a programot az újonnan létrehozott változókhoz különböző operátorokat is létrehozhatunk. Erre is szükségünk van, csak gondoljunk bele: Tudjuk, mit tegyünk egy valós számmal, egy szöveggel, de, hogyan értelmezzük két komplex szám egyenlőségét? Na, azt nem tudjuk. Éppúgy, mint azt sem, hogy két pont mikor lesz önmaga. Ezekre mi rájöhetünk, de a programolvasók nem fogják tudni, anélkül, hogy mi ne mondanánk meg neki. Ez a megmondás az operátor. Ennek szájbarágása nélkül csak részváltozókkal tudunk operálni, és ez nyilvánvalóan megnövelné a programunk terjedelmét, mikor mi minimálra törekszünk. Csak képzeljük el, mihez kezdenénk egy 10 dimenziós vektortér esetén. $V \in \mathbb{R}^{10}$.



Az operátornak a fent említett módon típust kell adni, majd a benne használandó változókat deklarálni kell a zárójelben. Az operátor törzsébe kerül azon részváltozókkal végzendő műveletek, melyeket a program már képes értelmezni és mi szeretnénk, hogy történjen. Ezek után a program már értelmezni képes a megadott típusok közötti szimbólumot.

A cout << komplex_neve értelmeztetése konkrétan így történik:

```
friend ostream & operator << ( ostream& output,komplex masik)
{
output<<masik.re<<"+i*"<<masik.im;
return output;
}
```

Mivel a cout nem egy változótípus, ezért csak így lehet megadni, bármilyen típus kiírását is szeretnénk értelmeztetni. Itt látható, hogy valóban elég a két double tárolása, mert az „+i*” könnyedén kiíratható vele.

Innentől kezdve a fájlba írást is értelmezni tudja, ezért ajánlott megérteni, megjegyezni.

Ennek leellenőrzésére készítsünk egy olyan programot, mely egy adott komplex számot deklarál, majd azt egyenlővé teszi egy másik komplexszámmal, és ezt a komplex számot írjuk ki a képernyőre és egy fájlba.

spoiler!

```
#include <iostream>

#include <fstream>
using namespace std;

struct komplex
{
double re,im;

komplex & operator =(komplex masik)
{
re=masik.re;
im=masik.im;
return masik;
}

friend ostream & operator << ( ostream& output,komplex masik)
{
output<<masik.re<<"+j*"<<masik.im;
return output;
}
};

int main()
{
      komplex b;
         b.re=1.0;
         b.im=2.0;

        komplex a;
        a=b;
     cout<<a;
     ofstream f("Complex.txt");
     f<<a;
     return 0;
}
```

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <vector>
#include <sstream>
```

Alapvetőbb feladatrészek:

Vektor megtöltése ciklussal+ összegzés tétele:

Vektor esetén ne felejtsük el az #include <vector> fejlécet!

Alapkérdés, hogyan töltsünk meg egy vektort csupa egyessel, majd szummázuk őket (összegzés tétele).

Ezt forral a legcélszerűbb. A v.size() egy függvény, mely a vektor hosszára utal, ez mindig arra a számra tér, amit mi a deklarációban a v vektornak adtunk, mint méret. Egy tíz elemű vektor deklarálása: vector<típus> vektor_neve<10>. A fenti függvény ezt azt a számértéket veszi föl, ami itt a 10. Figyelni kell, mert a vektor első eleme a 0. elem. De ez megfogja könnyíteni a ciklus létrehozását.

```
vector<int> v(10);
        for (int i=0;i<v.size();i++) {
                v[i]=1;
    }
```

Látható, hogy a futóindexet elég a v.size-ig vinni. Összeadás esetén hasonlóképpen járunk el. Az összegzés tételét alkalmazzuk. E szerint deklarálnunk kell egy olyan változót, mely az összeadás értékét tárolja. Ha ezt a cikluson belül deklaráljuk null kezdőértékkel, akkor minden egyes visszatérés után lenullázódik az érték, így nem mentettük el, ezért a cikluson kívülre kell tennünk.

A cikluson belül ezt az értéket kell módosítanunk úgy, hogy ehhez hozzáadjuk az i. elemet.

```
vector<int> v(10);
double szumma=0;
        for (int i=0;i<v.size();i++) {
                szumma=szumma+v[i];
    }
```

Maximumkeresés tétele II:

Az előző fejezetben látott leírás csak addig jó, amíg a pozitív számok halmazán vagyunk. Legtöbb esetben vektorbeli elemeket kell majd összehasonlítanunk. Az előző programrészlet kiterjesztése negatív számokra macerás lenne, ezért mentsük el a bekért számokat egy vektorba. Ezt az előbb nézett módszerrel alkalmazva megtehetjük.

```
vector<int> v(10);
double b=0,maximum=v[0];
        for (int i=0;i<v.size();i++) {
                cin >> b;
                v[i]=b;
                if (v[i]>maximum)
                {
                    maximum=v[i];
                }
    }
cout << maximum;
```

Ezt úgy oldhatjuk meg, hogy a maximum kezdeti értékének a vektor 0. elemét adjuk meg. Innentől minden a már ismert módon hajtható végre.

A fejlécek átismétlése kedvéért kérlek sorold fel a lehető legtöbb általunk ismert fejléct! (6) Jobb felső sarokban megtalálhatóak.

Az eukleideszi távolság két pont között ahol a az A, b a B pontok koordinátái.

$$\sqrt{\sum_i (a_i - b_i)^2}$$

Ezt írjuk fel C++ nyelven, fejlécelni ne felejtsünk!

Spoiler:

z=sqrt(pow(abs(a.x-b.x),2)+ pow(abs(a.y - b.y),2));

(abs, ide nem feltétlenül kell, csak plusznak van benne)

Jó tanulást kívánok mindenkinek! ☺

Referencia

```
void fele (int &a)
{
a=a/2;
}
```

A függvényen kívüli változót a megadottak szerint módosítja és elmenti, referencia nélkül a függvény lefutása után a függvényen kívüli változó értéke az marad, ami a fgv előtt is volt. A referencia akkor használatos, amikor nem a függvényünkkel „számolnak” tovább, hanem például a függvényen kívüli változó értékével, viszont a függvényben elvégzett művelet értékét várják el tőlünk a változó helyén.

A képen látható példában például beolvastathatjuk a „szam” nevű változót, ekkor a függvény után a „szam” nevű változó értéke felére csökken.

Más szavakkal ’&’ nélkül a C++ beolvassa a függvény nélküli változó értékét az inputváltozóba, majd ezzel operál a függvényen belül, vagy ad a függvénynek értéket (return esetén) vagy nem. ’&’ láttán a program beolvassa a függvényen kívüli változót az input változóba, azzal operál a függvényen belül, majd a függvény_vége bajusz elérésekor egyenlővé teszi a függvényen kívüli változót a függvényben levő inputváltozó utolsó értékével. – Excelben jártasaknak ismerős lehet lsd függvény írásakor, mit jelent az ’$’ jel egyes cellák megnevezésében.

GETLINE (fájlból beolvasás)

Kizárólag stringtípusnál értelmezett!

getline(miből,hova_string,meddig)

Tabulátor/Enter szimbóluma:

\t tabulátor

\n enter

További hintekért lesd meg:

cplusplus.com

Jó tanulást kívánok mindenkinek! ☺