



Bevezetés a programozásba

7. Előadás: Függvények 1.

Helló

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

int main()
{
    cout << "Hello";
    return 0;
}
```

Helló

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

int main()
{
    cout << "Hello";
    return 0;
}
```

```
int main()
{
    int a,b;
    cout << "Add meg a két számot:";
    cin >> a >> b;
    int ooa=0;
    for(int i=1;i<=a;i++) {
        if (a%i==0) ooa+=i;
    }
    int oob=0;
    for(int j=1;j<=b;j++) {
        if (b%j==0) oob+=j;
    }
    cout << "Az osztóösszegek:"
        << ooa <<" ," << oob<<endl;
    return 0;
}
```

```
int main()
{
    int a,b;
    cout << "Add meg a két számot:";
    cin >> a >> b;
    int ooa=0;
    for(int i=1;i<=a;i++) {
        if (a%i==0) ooa+=i;
    }
    int oob=0;
    for(int j=1;j<=b;j++) {
        if (b%j==0) oob+=j;
    }
    cout << "Az osztóösszegek:"
        << ooa <<" ," << oob<<endl;
    return 0;
}
```

```
alprogram{
    int ooX=0;
    for(int i=1;i<=X;i++) {
        if (X%i==0) ooX+=i;
    }
}

int main()
{
    int a,b;
    cout << "Add meg a két számot:";
    cin >> a >> b;
    alprogram X < a
    alprogram X < b
    cout << "Az osztóösszegek:"
        << oa << ", " << ob<<endl;
    return 0;
}
```

```
int alprogram{
    int eredmeny=0;
    for(int i=1;i<=X;i++) {
        if (X%i==0) eredmeny+=i;
    }
    return eredmeny;
}

int main()
{
    int a,b;
    cout << "Add meg a két számot:";
    cin >> a >> b;
    int ooa = alprogram X < a
    int oob = alprogram X < b
    cout << "Az osztóösszegek:"
        << ooa <<" ," << oob<<endl;
    return 0;
}
```

```
int alprogram(int X) {
    int eredmeny=0;
    for(int i=1;i<=X;i++) {
        if (X%i==0) eredmeny+=i;
    }
    return eredmeny;
}

int main()
{
    int a,b;
    cout << "Add meg a két számot:";
    cin >> a >> b;
    int ooa = alprogram(a);
    int oob = alprogram(b);
    cout << "Az osztóösszegek:"
        << ooa <<" ," << oob<<endl;
    return 0;
}
```


Függvények írásmódja

- Ha egy C++ kódban ezt látjuk:
típusnév bármilyen név (tetszőleges deklarációk)
{
....
 return *valami*;
}
- Akkor az egy függvény

Függvények írásmódja

- Ha egy C++ kódban ezt látjuk:
típusnév bármilyen név (tetszőleges deklarációk)
{
....
 return *valami*;
}
- Akkor az egy függvény

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    cout << "Hello";  
    return 0;  
}
```

Függvény

- A függvény belsejében egy program van
- Minden C/C++ programban van egy fő függvény, ennek a neve „main”
- Amikor a teljes program elindul, akkor a main függvény indul el, és ha ez a függvény befejeződik, akkor a teljes program is befejeződik
- Mi a lényeg? Függvények más függvényeket hívnak meg.
- Ez lényegében egy új programkonstrukció

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

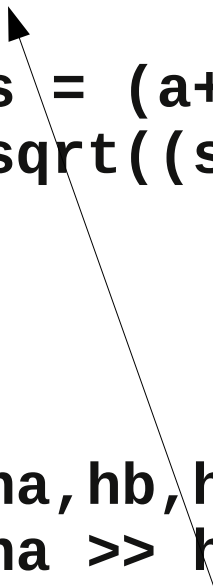
int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc;
    double t = terület(ha, hb, hc);
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvényhívás

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

double terulet(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc;
    double t = terulet(ha, hb, hc);
    cout << "terulet: " << t << endl;
    return 0;
}
```

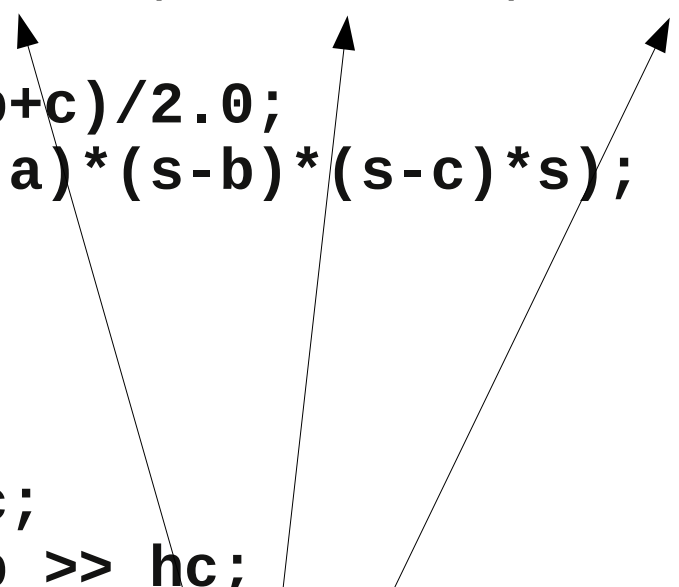


Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc;
    double t = terület(ha, hb, hc);
    cout << "terület: " << t << endl;
    return 0;
}
```

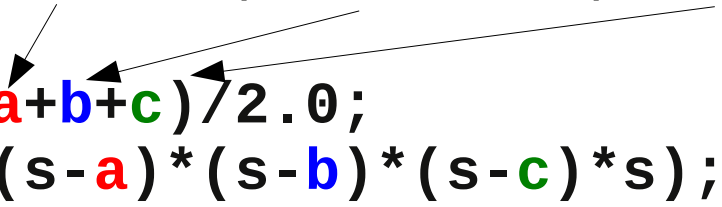
A diagram illustrating argument passing. Three arrows originate from the arguments 'ha', 'hb', and 'hc' in the function call 'terület(ha, hb, hc);' within the main function. Each arrow points to the corresponding parameter 'a', 'b', and 'c' in the function definition 'double terület(double a, double b, double c)'. The parameter 'a' is highlighted in red, 'b' in blue, and 'c' in green, matching the color of the arguments in the call.

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc;
    double t = terület(ha, hb, hc);
    cout << "terület: " << t << endl;
    return 0;
}
```

A diagram illustrating function argument passing. Three arrows originate from the variables 'ha', 'hb', and 'hc' in the 'terület' function call within the 'main' function. These arrows point to the parameters 'a', 'b', and 'c' in the 'terület' function signature. A fourth arrow points from the parameter 'a' to its use in the expression '(a+b+c)' within the function body.

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

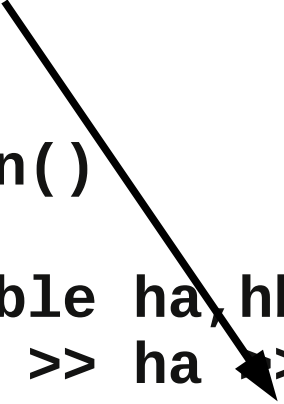
int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc;
    double t = terület(ha, hb, hc);
    cout << "terület: " << t << endl;
    return 0;
}
```


Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc;
    double t = terület(ha, hb, hc);
    cout << "terület: " << t << endl;
    return 0;
}
```



Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = terület(ha, hb, hc);
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = terület(3,4,5);
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = (3+4+5)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = terület(3,4,5);
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = 6;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = terület(3,4,5);
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = 6;
    return sqrt((6-3)*(6-4)*(6-5)*6);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = terület(3,4,5);
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = 6;
    return sqrt(3*2*1*6);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = terület(3,4,5);
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = 6;
    return sqrt(36);
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = terület(3,4,5);
    cout << "terület: " << t << endl;
    return 0;
}
```


Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = 6;
    return 6;
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = terület(3,4,5);
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvényhívás

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

double terület(double a, double b, double c)
{
    double s = 6;
    return 6;
}

int main()
{
    double ha, hb, hc;
    cin >> ha >> hb >> hc; //ha=3, hb=4, hc=5
    double t = 6;
    cout << "terület: " << t << endl;
    return 0;
}
```

Függvények

```
double terület(double a, double b, double c)
```

- Szignatúra:
 - Visszatérési típus (a függvény típusának is nevezik)
 - Függvény neve
 - Zárójelben
 - Paraméter 1, ha van
 - Paraméter 2, ha van
 - ..
- A szignatúrából minden kiderül arról, hogyan kell használni a függvényt

Függvény szerkezete

- Szignatúra
- Törzs
 - program
 - benne **return** visszatérési érték
- Egy függvény több **return**-t is tartalmazhat, de már az első **return** végrehajtásakor véget ér a függvény
- Tehát bármit írsz egy **return** mögé, az sosem hajtódik végre

A void típus

- Ez a „semmilyen” típus
- Ilyen típusú változót nem lehet csinálni
- Ezzel jelezzük, ha egy függvénynek nincs visszatérési értéke, tehát a visszatérési típusa „semmilyen”
- `void kepernyotorles()`
- `void irdakonzolra(T1 ezt, T2 eztis)`
- ```
if (teleakepernyo) {
 kepernyotorles();
}
```

# Egyebek

- Mi a teendő, ha nem egy változót szeretnék visszaadni, hanem többet?
  - Türelmesen várj a következő előadásokig
- Mi történik, ha void visszatérési típusú függvényt hívok?
  - Nem használható kifejezésben, vagy értékadásban a függvényhívás, mert „semmilyen” a típusa
- Lehet-e függvényhíváskor a paraméter egy másik függvény visszatérési értéke?
  - Természetesen, a visszatérési értékkel rendelkező függvény tetszőleges kifejezés része lehet

# Szimbolikus és aktuális paraméter

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

double terület(double a, double b, double c)
{
 double s = (a+b+c)/2.0;
 return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
 double ha, hb, hc;
 cin >> ha >> hb >> hc;
 double t = terület(ha, hb, hc);
 cout << "terület: " << t << endl;
 return 0;
}
```

szimbolikus

aktuális

# Szimbolikus és aktuális paraméter

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

double terület(double a, double b, double c)
{
 double s = (a+b+c)/2.0;
 return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
 double ha, hb, hc;
 cin >> ha >> hb >> hc;
 double t = terület(ha, hb, hc);
 double t2 = terület(ha+1, hb+1, hc+1);
 cout << "terület: " << t << endl;
 return 0;
}
```

szimbolikus

aktuális



# Paraméterek érvényessége és láthatósága

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

double terület(double a, double b, double c)
{
 double s = (a+b+c)/2.0;
 return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
 double a, b, c;
 cin >> a >> b >> c;
 double t = terület(a, b, c);
 cout << "terület: " << t << endl;
 return 0;
}
```

szimbolikus: csak a sorrend számít

aktuális

# Paraméterek érvényessége és láthatósága

- Szabad, és nem kötelező ugyanazokat a változóneveket használni aktuális és formális paraméterként
- A függvény paramétere lokális változó, ami csak a függvényben él
- Minden olyan változó, ami függvényben lett deklarálnva, lokális változó
- A main() függvény változói sem használhatóak a többi függvényben
- A függvények paraméterekkel kommunikálnak

# Paraméterek érvényessége és láthatósága

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

double terület(double a, double b, double c)
{
 double s = (a+b+c)/2.0;
 return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
 double ha, hb, hc;
 cin >> ha >> hb >> hc;
 double t = terület(ha, hb, hc);
 cout << "terület: " << t << endl;
 return 0;
}
```

# Paraméterek érvényessége és láthatósága

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

double terület(double a, double b, double c)
{
 double s = (a+b+c)/2.0;
 return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
 double a,b,c;
 cin >> a >> b >> c;
 double t = terület(a,b,c);
 cout << "terület: " << t << endl;
 return 0;
}
```

„Ez az **a** nem az az **a**”

# Névválasztás

- Érdeemes a formális paraméter nevét a függvénybeli szerep szerint elnevezni
- Az aktuális paraméter lehet akár kifejezés is
- Néha előfordul, hogy ugyanolyan nevű változók vannak az aktuális paraméterben és a formális paraméterben
  - ez nem baj, de az első programoknál érdemes kerülni, amíg a lokális változók fogalma nem tisztázódik
  - később akár jó is lehet, segítheti az áttekinthetőséget

# Változó láthatósága

- Egy változó látható a programkód azon soraiban, ahol a nevének leírásával használható
  - C++ nyelvben a lokális változók nem láthatóak más függvényekben
  - A mindenhol látható változót függvényen kívül kell deklarálni: globális változó
    - A globális változókat ahol lehet, kerüljük. Néhány értelmes használata van, ezekre általában kitérünk majd, minden másnál a jó stratégia az, hogy paraméterben adjuk át a szükséges adatokat
    - Ha mégis használjuk, azokban a függvényekben lesz látható, amelyek a deklaráció alatt vannak

# Változó érvényessége

- A változó érvényes a program futásának azon szakaszain, ahol az értékét megtartja
- A lokális változó a függvényben levő deklarációjától számítva addig él, amíg a deklarációjának blokkja be nem fejeződik.
- A paraméterek a függvény végéig élnek
- Ha egy A függvényből egy B függvényt hívunk, az A lokális változói ugyan nem láthatóak B-ben, de érvényesek maradnak, mert mikor B befejeződik, A-ban továbbra is használhatóak, és értéküket megtartották ezalatt.

# Érvényesség

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

double terület(double a, double b, double c)
{
 double s = (a+b+c)/2.0;
 return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
 double ha, hb, hc;
 cin >> ha >> hb >> hc;
 double t = terület(ha, hb, hc);
 cout << "A " << ha << ", " << hb << ", " << hc;
 cout << " haromszog terulete: " << t << endl;
 return 0;
}
```



# Procedurális (vagy hierarchikus) programozás

- Programtervezési elv: a feladatot osszuk részfeladatokra, és függvényekben csoportosítsuk azokat
  - „dekompozíció”, „redukció”
- int nagyonnehézfeladat(P1 P2 P3 ..) {  
    egyikkicsitkönnyebb(P2 P4 ..);  
    másikkicsitkönnyebb(P1 P2 ..);  
    ...  
}
- Top-down vagy Bottom-up felépítés
- 80-as évekig nagy szoftvereknél egyeduralkodó

# Trükk

- Mi történik, ha függvényhívások „körbeérnek”?
  - Excel: „feloldhatatlan körbehivatkozás”
  - OpenOffice: „522 Körkörös hivatkozás; A képlet közvetlenül vagy közvetetten önmagára hivatkozik, és az Eszközök - Beállítások - OpenOffice.org Calc - Számítás panel **Iterációk** beállítása nincs kijelölve.”
  - A C++ program így könnyen végtelen ciklusba kerülhet, de ezen lehet segíteni
- Rekurzió, rekurzív függvény
- Eleinte nehéz megérteni, hogy egy függvény több példányban is várjon visszatérési értékre

# Rekurzív függvény

```
int faktor(int p)
{
 if (p>1) {
 return p*faktor(p-1);
 } else {
 return 1;
 }
}

int main()
{
 cout << faktor(5) << endl;
 return 0;
}
```

# Rekurzió

- Nem kötelező anyag
- A megértést megkönnyítő fogalmak, jelenségek
  - A lokális változók annyi példányban léteznek, amilyen „mély a rekurzió”
  - A paraméterek és visszatérési értékek egy ún. verem segítségével közlekednek
    - „Amit először beleteszek, azt veszem ki utoljára”
  - Mindig van elágazás, ami alapján visszatérünk
    - különben végtelen ciklusunk van, „megtelt a verem” futásidejű hibaüzenettel, vagy ennek következményével (0xC0000005)

# Összefoglalás

- A nagyobb problémákat függvényekre bontjuk
- Az ismétlődő kódrészleteket egyszer írjuk csak le, és az eredeti több helyről meghívjuk
- A változók a másik függvényben nem láthatóak, ezért paraméterekkel és visszatérési értékkel kommunikálunk
- Érvényesség, láthatóság, lokális változók
- Függvények hívhatják egymást
  - ha ez körbeér, az a rekurzió, ezt eleinte kerüljük.