# Függvények definiálása

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2023 szeptember 19.

# Saját függvények definiálása

Miért van szükség saját függvények definiálására?

- bizonyos feladatok többször is ismétlődhetnek a program futása során
- érdemes logikailag leválasztani a program egy részét
  - már a `main()` is eleve egy függvény volt

Tekintsük példaként a korábban már látott másodfokú egyenlet gyökeit számoló programot!

# Saját függvények definiálása

```
#include <iostream>
#include <cmath>

using namespace std;

double discr(double a, double b, double c)
{ return b*b-4*a*c;}

void roots(double a, double b, double c)
{
    double r1, r2;
    double d=discr(a,b,c);

    if (d>0)
    {d=sqrt(d);
    r1=(-b+d)/2/a; r2=(-b-d)/2/a;
    cout << " r1=" << r1 << endl;
    cout << " r2=" << r2 << endl;
    }
    else if (d==0.0)
    {r1=-b/2/a;
    cout << "egy gyok van r1=" << r1 << endl;}
    else {cout << "nincs valos gyok" << endl;}
}
```

- A függvényt a visszatérési értéke, a neve és a paramétereinek típusa azonosítja.
  Pl. `double discr ( double a, double b, double c)`

# Saját függvények definiálása

```
#include <iostream>
#include <cmath>

using namespace std;

double discr(double a, double b, double c)
{ return b*b-4*a*c;}

void roots(double a, double b, double c)
{
    double r1, r2;
    double d=discr(a,b,c);

    if (d>0)
    {d=sqrt(d);
    r1=(-b+d)/2/a; r2=(-b-d)/2/a;
    cout << " r1=" << r1 << endl;
    cout << " r2=" << r2 << endl;
    }
    else if (d==0.0)
    {r1=-b/2/a;
    cout << "egy gyok van r1=" << r1 << endl;}
    else {cout << "nincs valos gyok" << endl;}
}
```

- A függvényt a visszatérési értéke, a neve és a paramétereinek típusa azonosítja.
  Pl. `double discr ( double a, double b, double c)`
- A függvény definíciója a `{ ... }` részbe kerül

# Saját függvények definiálása

```
#include <iostream>
#include <cmath>

using namespace std;

double discr(double a, double b, double c)
{ return b*b-4*a*c;}

void roots(double a, double b, double c)
{
    double r1, r2;
    double d=discr(a,b,c);

    if (d>0)
    {d=sqrt(d);
    r1=(-b+d)/2/a; r2=(-b-d)/2/a;
    cout << " r1=" << r1 << endl;
    cout << " r2=" << r2 << endl;
    }
    else if (d==0.0)
    {r1=-b/2/a;
    cout << "egy gyok van r1=" << r1 << endl;}
    else {cout << "nincs valos gyok" << endl;}
}
```

- A függvényt a visszatérési értéke, a neve és a paramétereinek típusa azonosítja.
  Pl. `double discr ( double a, double b, double c)`
- A függvény definíciója a `{ ... }` részbe kerül
- `return`: a függvény visszatérési értéke

# Saját függvények definiálása

```
#include <iostream>
#include <cmath>

using namespace std;

double discr(double a, double b, double c)
{ return b*b-4*a*c;}

void roots(double a, double b, double c)
{
     double r1, r2;
     double d=discr(a,b,c);

     if (d>0)
     {d=sqrt(d);
     r1=(-b+d)/2/a; r2=(-b-d)/2/a;
     cout << " r1=" << r1 << endl;
     cout << " r2=" << r2 << endl;
     }
     else if (d==0.0)
     {r1=-b/2/a;
     cout << "egy gyok van r1=" << r1 << endl;}
     else {cout << "nincs valos gyok" << endl;}
}
```

- A függvényt a visszatérési értéke, a neve és a paramétereinek típusa azonosítja.
  Pl. `double discr ( double a, double b, double c)`
- A függvény definíciója a `{ ... }` részbe kerül
- `return`: a függvény visszatérési értéke
- `void` típusú függvény: pl `void roots(...)` nem számértéket ad vissza, hanem egy folyamatsort hajt végre

# Saját függvények hívása

```
int main()
{
    double a, b, c;

    cout << "Kerem a szamokat:\n";
    cout << "  a="; cin >> a;
    cout << "  b="; cin >> b;
    cout << "  c="; cin >> c;

    roots(a,b,c);

     return 0;
}
```

- A saját függvényeket a `main()` előtt definiáltuk
- Ezután jön a függvényhívás a `main()`-ben: `roots(a,b,c);`

# Függvény definiálás vs függvény hívás

```
double discr(double a, double b, double c)
{ return b*b-4*a*c;}

void roots(double a, double b, double c)
{
    double r1, r2;
    double d=discr(a,b,c);
```

- **Függvény definiálása**
  - visszatérési érték típusa a név előtt
  - paraméterek típusa a nevek előtt

```
int main()
{
   double a, b, c;

   cout << "Kerem a szamokat:\n";
   cout << "  a="; cin >> a;
   cout << "  b="; cin >> b;
   cout << "  c="; cin >> c;

   roots(a,b,c);

    return 0;
}
```

- **Függvény hívása**
  - nincs kiírva a függvény típusa
  - nincs kiírva az átadott paraméterek típusa
  - függvényparaméterként kifejezést is megadhatunk, aminek értéke a híváskor kiértékelődik

# A végrehajtási sorrend függvényhíváskor

1. A program egy **függvényhíváshoz** ér
   - sorban kiértékelődik az összes paraméter értéke (ha kifejezések)
   - majd a program futása a meghívott függvény első során folytatódik
2. Ha a program egy `return` utasításhoz ér, akkor a függvény **visszatér**
   - ha van visszatérési érték, akkor azt a `return` utasításnak kell beállítania
   - a visszatérés utána a program a **függvényhívást követő** utasítástól folytatódik
3. Ha a program nem talál `return` utasítást (`void` típusú függvény), akkor a függvény az utolsó utasítás után tér vissza
4. A függvényhívások egymásba ágyazhatóak
   - két függvény hívhatja egymást kölcsönösen
   - sőt, egy függvény akár saját magát is hívhatja $\Rightarrow$ rekurzív hívás

# Változók hatálya, paraméterek átadása

Mint korábban már láttuk, a változók hatályát a kapcsos zárójelek jelölik ki

- a függvényeken belül létrehozott változók *lokálisak*
- csak a függvényből látszanak
- ha a függvény egy másik függvényt hív, a lokális változót át kell adni a hívott függvénynek (ha használni akarjuk)

# Változók hatálya, paraméterek átadása

Mint korábban már láttuk, a változók hatályát a kapcsos zárójelek jelölik ki

- a függvényeken belül létrehozott változók *lokálisak*
- csak a függvényből látszanak
- ha a függvény egy másik függvényt hív, a lokális változót át kell adni a hívott függvénynek (ha használni akarjuk)

A fenti példákban a függvényparamétereket **érték** szerint adtuk át a hívott függvénynek

- az átadott paraméterek csak a függvényen belül érhetők el
- ugyanúgy működnek, mint a lokális változók
- pl. át lehet írni az értéküket, de ez nem változtatja meg a külső függvény lokális változójának értékét!

# Változók hatálya, paraméterek átadása

```
int func(int x)
{x=x+1;
return x
}

int main()

int xmain=0;

cout << func(xmain) << "\n"; \\ kiirt ertek: 1
cout << xmain <<"\n";        \\ kiirt ertek: 0
```

Később majd látunk példát arra, hogy nem csak érték szerint lehet paramétereket átadni

# Globális változók

Változók deklarálhatók függvényeken kívül is

- *globális változók*
- ezek minden függvényből látszanak
- nem kell őket expliciten átadni függvényhíváskor
- a használatukat érdemes kerülni