

# Függvények definiálása

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2020. szeptember 14.

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

# Saját függvények definiálása

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  double discr(double a, double b, double c) {
6      return b * b - 4 * a * c;
7  }
8
9  void roots(double a, double b, double c) {
10     double d = discr(a, b, c);
11     if (d < 0) {
12         printf("No real solution.\n");
13         exit(-1);
14     } else if (d == 0) {
15         double r = -b / 2 / a;
16         printf("r = %f\n", r);
17     } else {
18         d = sqrt(d);
19         double r1 = (-b - d) / 2 / a;
20         double r2 = (-b + d) / 2 / a;
21         printf("r1 = %f, r2 = %f\n", r1, r2);
22     }
23 }
24
25 int main(int argc, char* argv[]) {
26     if (argc < 4) {
27         printf("Not enough arguments.\n");
28         exit(-1);
29     }
30     double a = atof(argv[1]);
31     double b = atof(argv[2]);
32     double c = atof(argv[3]);
33     roots(a, b, c);
34     return 0;
35 }
```

- A függvényt a visszatérési értéke, a neve és a paramétereinek típusa azonosítja

# Saját függvények definiálása

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  double discr(double a, double b, double c) {
6      return b * b - 4 * a * c;
7  }
8
9  void roots(double a, double b, double c) {
10     double d = discr(a, b, c);
11     if (d < 0) {
12         printf("No real solution.\n");
13         exit(-1);
14     } else if (d == 0) {
15         double r = -b / 2 / a;
16         printf("r = %f\n", r);
17     } else {
18         d = sqrt(d);
19         double r1 = (-b - d) / 2 / a;
20         double r2 = (-b + d) / 2 / a;
21         printf("r1 = %f, r2 = %f\n", r1, r2);
22     }
23 }
24
25 int main(int argc, char* argv[]) {
26     if (argc < 4) {
27         printf("Not enough arguments.\n");
28         exit(-1);
29     }
30     double a = atof(argv[1]);
31     double b = atof(argv[2]);
32     double c = atof(argv[3]);
33     roots(a, b, c);
34     return 0;
35 }
```

- A függvényt a visszatérési értéke, a neve és a paramétereinek típusa azonosítja
- A függvény definíciója a { ... } részbe kerül

# Saját függvények definiálása

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  double discr(double a, double b, double c) {
6      return b * b - 4 * a * c;
7  }
8
9  void roots(double a, double b, double c) {
10     double d = discr(a, b, c);
11     if (d < 0) {
12         printf("No real solution.\n");
13         exit(-1);
14     } else if (d == 0) {
15         double r = -b / 2 / a;
16         printf("r = %f\n", r);
17     } else {
18         d = sqrt(d);
19         double r1 = (-b - d) / 2 / a;
20         double r2 = (-b + d) / 2 / a;
21         printf("r1 = %f, r2 = %f\n", r1, r2);
22     }
23 }
24
25 int main(int argc, char* argv[]) {
26     if (argc < 4) {
27         printf("Not enough arguments.\n");
28         exit(-1);
29     }
30     double a = atof(argv[1]);
31     double b = atof(argv[2]);
32     double c = atof(argv[3]);
33     roots(a, b, c);
34     return 0;
35 }
```

- A függvényt a visszatérési értéke, a neve és a paramétereinek típusa azonosítja
- A függvény definíciója a `{ ... }` részbe kerül
- **return**: ha csak vissza kell adni valamilyen értéket
- **exit**: csak hibakezeléskor

# Saját függvények definiálása

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  double discr(double a, double b, double c) {
6      return b * b - 4 * a * c;
7  }
8
9  void roots(double a, double b, double c) {
10     double d = discr(a, b, c);
11     if (d < 0) {
12         printf("No real solution.\n");
13         exit(-1);
14     } else if (d == 0) {
15         double r = -b / 2 / a;
16         printf("r = %f\n", r);
17     } else {
18         d = sqrt(d);
19         double r1 = (-b - d) / 2 / a;
20         double r2 = (-b + d) / 2 / a;
21         printf("r1 = %f, r2 = %f\n", r1, r2);
22     }
23 }
24
25 int main(int argc, char* argv[]) {
26     if (argc < 4) {
27         printf("Not enough arguments.\n");
28         exit(-1);
29     }
30     double a = atof(argv[1]);
31     double b = atof(argv[2]);
32     double c = atof(argv[3]);
33     roots(a, b, c);
34     return 0;
35 }
```

- A függvényt a visszatérési értéke, a neve és a paramétereinek típusa azonosítja
- A függvény definíciója a `{ ... }` részbe kerül
- **return**: ha csak vissza kell adni valamilyen értéket
- **exit**: csak hibakezeléskor
- **void** típusú függvény: nem számértéket ad vissza, hanem egy folyamatort hajt végre

# Függvény definiálása vs. függvényhívás

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  double discr(double a, double b, double c) {
6      return b * b - 4 * a * c;
7  }
8
9  void roots(double a, double b, double c) {
10     double d = discr(a, b, c);
11     if (d < 0) {
12         printf("No real solution.\n");
13         exit(-1);
14     } else if (d == 0) {
15         double r = -b / 2 / a;
16         printf("r = %f\n", r);
17     } else {
18         d = sqrt(d);
19         double r1 = (-b - d) / 2 / a;
20         double r2 = (-b + d) / 2 / a;
21         printf("r1 = %f, r2 = %f\n", r1, r2);
22     }
23 }
24
25 int main(int argc, char* argv[]) {
26     if (argc < 4) {
27         printf("Not enough arguments.\n");
28         exit(-1);
29     }
30     double a = atof(argv[1]);
31     double b = atof(argv[2]);
32     double c = atof(argv[3]);
33     roots(a, b, c);
34     return 0;
35 }
```

## • Függvény definiálása

- visszatérési érték típusa a név előtt
- paraméterek típusa a nevek előtt

# Függvény definiálása vs. függvényhívás

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  double discr(double a, double b, double c) {
6      return b * b - 4 * a * c;
7  }
8
9  void roots(double a, double b, double c) {
10     double d = discr(a, b, c);
11     if (d < 0) {
12         printf("No real solution.\n");
13         exit(-1);
14     } else if (d == 0) {
15         double r = -b / 2 / a;
16         printf("r = %f\n", r);
17     } else {
18         d = sqrt(d);
19         double r1 = (-b - d) / 2 / a;
20         double r2 = (-b + d) / 2 / a;
21         printf("r1 = %f, r2 = %f\n", r1, r2);
22     }
23 }
24
25 int main(int argc, char* argv[]) {
26     if (argc < 4) {
27         printf("Not enough arguments.\n");
28         exit(-1);
29     }
30     double a = atof(argv[1]);
31     double b = atof(argv[2]);
32     double c = atof(argv[3]);
33     roots(a, b, c);
34     return 0;
35 }
```

## • Függvény definiálása

- visszatérési érték típusa a név előtt
- paraméterek típusa a nevek előtt

## • Függvény hívása

- sehol sincs kírva a típus
- a paraméter kifejezés is lehet



```
1 def discr(a,b,c):
2     return b * b - 4 * a * c
3
4 def roots(a,b,c):
5     d = discr(a, b, c)
6     if d < 0:
7         print("No real solution.")
8     elif d == 0:
9         r = -b / 2 / a
10        print("r = ", r)
11    else:
12        d = sqrt(d)
13        r1 = (-b - d) / 2 / a
14        r2 = (-b + d) / 2 / a
15        print("r1 = ", r1, ", r2 = ", r2);
16
17 roots(1.1, 3.3, 2.2)
```

## • Függvény definiálása

- **def** utasítással
- visszatérési érték típusát nem kell megadni

```
1 def discr(a,b,c):
2     return b * b - 4 * a * c
3
4 def roots(a,b,c):
5     d = discr(a, b, c)
6     if d < 0:
7         print("No real solution.")
8     elif d == 0:
9         r = -b / 2 / a
10        print("r = ", r)
11    else:
12        d = sqrt(d)
13        r1 = (-b - d) / 2 / a
14        r2 = (-b + d) / 2 / a
15        print("r1 = ", r1, ", r2 = ", r2);
16
17 roots(1.1, 3.3, 2.2)
```

- Függvény definiálása
  - `def` utasítással
  - visszatérési érték típusát nem kell megadni
- Programblokkok nincsenek { ... } -ben

```
1 def discr(a,b,c):
2     return b * b - 4 * a * c
3
4 def roots(a,b,c):
5     d = discr(a, b, c)
6     if d < 0:
7         print("No real solution.")
8     elif d == 0:
9         r = -b / 2 / a
10        print("r = ", r)
11    else:
12        d = sqrt(d)
13        r1 = (-b - d) / 2 / a
14        r2 = (-b + d) / 2 / a
15        print("r1 = ", r1, ", r2 = ", r2);
16
17 roots(1.1, 3.3, 2.2)
```

- Függvény definiálása
  - **def** utasítással
  - visszatérési érték típusát nem kell megadni
- Programblokkok nincsenek { ... } -ben
- a kiértékelendő feltételt nem kell (...) -be tenni

```
1 def discr(a,b,c):
2     return b * b - 4 * a * c
3
4 def roots(a,b,c):
5     d = discr(a, b, c)
6     if d < 0:
7         print("No real solution.")
8     elif d == 0:
9         r = -b / 2 / a
10        print("r = ", r)
11    else:
12        d = sqrt(d)
13        r1 = (-b - d) / 2 / a
14        r2 = (-b + d) / 2 / a
15        print("r1 = ", r1, ", r2 = ",r2);
16
17 roots(1.1, 3.3, 2.2)
```

- Függvény definiálása
  - **def** utasítással
  - visszatérési érték típusát nem kell megadni
- Programblokkok nincsenek { ... } -ben
- a kiértékelendő feltételt nem kell (...) -be tenni
- **else if** lerövidítve **elif**

- 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, akkor a függvény az utolsó utasítás után tér vissza
- 4 A függvényhívások *szinte* tetszőleges mértékben egymásba ágyazhatóak
  - két függvény hívhatja egymást kölcsönösen
  - sőt, egy függvény akár saját magát is hívhatja

# Függvények előre deklarációja

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  double discr(double a, double b, double c);
6  void roots(double a, double b, double c);
7
8  int main(int argc, char* argv[]) {
9      if (argc < 4) {
10         printf("Not enough arguments.\n");
11         exit(-1);
12     }
13     double a = atof(argv[1]);
14     double b = atof(argv[2]);
15     double c = atof(argv[3]);
16     roots(a, b, c);
17     return 0;
18 }
19
20 void roots(double a, double b, double c) {
21     double d = discr(a, b, c);
22     if (d < 0) {
23         printf("No real solution.\n");
24         exit(-1);
25     } else if (d == 0) {
26         double r = -b / 2 / a;
27         printf("r = %f\n", r);
28     } else {
29         d = sqrt(d);
30         double r1 = (-b - d) / 2 / a;
31         double r2 = (-b + d) / 2 / a;
32         printf("r1 = %f, r2 = %f\n", r1, r2);
33     }
34 }
35
36 double discr(double a, double b, double c) {
37     return b * b - 4 * a * c;
38 }
```

A függvényt korábban kell deklarálni, mint ahogyan hivatkozunk rá.

- ilyenkor csak a **szignatúrát** írjuk ki
- a sort **;** zárja

# Függvények előre deklarálása

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 double discr(double a, double b, double c);
6 void roots(double a, double b, double c);
7
8 int main(int argc, char* argv[]) {
9     if (argc < 4) {
10         printf("Not enough arguments.\n");
11         exit(-1);
12     }
13     double a = atof(argv[1]);
14     double b = atof(argv[2]);
15     double c = atof(argv[3]);
16     roots(a, b, c);
17     return 0;
18 }
19
20 void roots(double a, double b, double c) {
21     double d = discr(a, b, c);
22     if (d < 0) {
23         printf("No real solution.\n");
24         exit(-1);
25     } else if (d == 0) {
26         double r = -b / 2 / a;
27         printf("r = %f\n", r);
28     } else {
29         d = sqrt(d);
30         double r1 = (-b - d) / 2 / a;
31         double r2 = (-b + d) / 2 / a;
32         printf("r1 = %f, r2 = %f\n", r1, r2);
33     }
34 }
35
36 double discr(double a, double b, double c) {
37     return b * b - 4 * a * c;
38 }
```

A függvényt korábban kell deklarálni, mint ahogyan hivatkozunk rá.

- ilyenkor csak a **szignatúrát** írjuk ki
- a sort **;** zárja

A függvény definíciója már lehet később, mint az első hívás