Paraméterek megadása. Feltételes programelágazások

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2020. szeptember 14.

Emlékeztető: az első program

Másodfokú egyenlet megoldóképlete

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

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

int main() {
  double a, b, c;
  double d, r1, r2;
  a = 1.0;
  b = 3.0;
  c = 2.0;
  d = b * b - 4 * a * c;
  d = sqrt(d);
  r1 = (-b + d) / (2 * a);
  r2 = (-b - d) / (2 * a);
  printf("r1 = %f, r2 = %f\n", r1, r2);
  return 0;
}
```

Az előző program egy kicsit tömörebben

```
1   #include <stdio.h>
2   #include <math.h>
3
4   int main() {
5     double a = 1.0;
6     double b = 3.0;
7     double c = 2.0;
8     double d = sqrt(b * b - 4 * a * c);
9     double r1 = (-b + d) / (2 * a);
10    double r2 = (-b - d) / (2 * a);
11    printf("r1 = %f, r2 = %f\n", r1, r2);
12    return 0;
13  }
```

- a változók deklarálása összevonható az értékadással

Az előző program egy kicsit tömörebben

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

int main() {
  double a = 1.0;
  double b = 3.0;
  double c = 2.0;
  double d = sqrt(b * b - 4 * a * c);
  double r1 = (-b + d) / (2 * a);
  double r2 = (-b - d) / (2 * a);
  printf("r1 = %f, r2 = %f\n", r1, r2);
  return 0;
}
```

- a változók deklarálása összevonható az értékadással
- a függvények kifejezéseket is kaphatnak paraméterként
  - ilyenkor a kifejezés előbb kiértékelődik
  - a függvény a kapott eredményt kapja meg paraméterként

Az első program fő problémái

1. Az egyenlet együtthatói nem paraméterek, hanem konstansok
   - ez úgy mondjuk, hogy az értékek “hard code”-olva vannak
   - az ilyet mindig kerülni kell, kivéve ha valóban konstansokról van szó

Az első program fő problémái

1. Az egyenlet együtthatói nem paraméterek, hanem konstansok
   - ez úgy mondjuk, hogy az értékek “hard code”-olva vannak
   - az ilyet mindig kerülni kell, kivéve ha valóban konstansokról van szó

2. Hogyan lehetne az együtthatókat paraméterként beadni?
   - parancssori argumentumként
   - billentyűzetről beolvasva
   - fájlból beolvasva

Az első program fő problémái

1. Az egyenlet együtthatói nem paraméterek, hanem konstansok
   - ez úgy mondjuk, hogy az értékek “hard code”-olva vannak
   - az ilyet mindig kerülni kell, kivéve ha valóban konstansokról van szó
2. Hogyan lehetne az együtthatókat paraméterként beadni?
   - parancssori argumentumként
   - billentyűzetről beolvasva
   - fájlból beolvasva
3. Mi történik akkor, ha olyan együtthatókat nézünk, ahol a diszkrimináns negatív?

Parancssori paraméterek beolvasása

A parancssori argumentumokat a konzolablakban szeretnénk megadni futtatáskor, pl.:

```
1  $ ./roots 1 3 2
```

Parancssori paraméterek beolvasása

A parancssori argumentumokat a konzolablakban szeretnénk megadni futtatáskor, pl.:

```
1  $ ./roots 1 3 2
```

Ezeket az argumentumokat a `main` függvény paramétereiként kapjuk meg

- vigyázat: minden paraméter szövegként van kezelve
- át kell alakítani a szöveget számmá
- egyelőre itt a kész megoldás: `atof` függvény

Parancssori paraméterek beolvasása

A parancssori argumentumokat a konzolablakban szeretnénk megadni futtatáskor, pl.:

```
$ ./roots 1 3 2
```

Ezeket az argumentumokat a `main` függvény paramétereiként kapjuk meg

- vigyázat: minden paraméter szövegként van kezelve
- át kell alakítani a szöveget számmá
- egyelőre itt a kész megoldás: `atof` függvény

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

int main(int argc, char* argv[]) {
  double a = atof(argv[1]);
  double b = atof(argv[2]);
  double c = atof(argv[3]);
  double d = sqrt(b * b - 4 * a * c);
  double r1 = (-b + d) / (2 * a);
  double r2 = (-b - d) / (2 * a);
  printf("r1 = %f, r2 = %f\n", r1, r2);
  return 0;
}
```

- Az `argc` paraméter a parancssori paraméterek számát tartalmazza
  - az első (0. indexű) paraméter mindig a futó program neve
- Az `argv` paraméterben szövegként kapjuk meg a paramétereket

Feltételes elágazások, az if utasítás

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

- Az if (...) utasítás zárójelében egy feltétel szerepel
  - logikai vagy aritmetikai kifejezés

Feltételes elágazások, az if utasítás

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

- Az `if (...)` utasítás zárójelében egy feltétel szerepel
  - logikai vagy aritmetikai kifejezés
- Ha feltétel teljesül (értéke nem nulla), akkor az `if` utáni `{ ... }` block fut le

Feltételes elágazások, az if utasítás

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

- Az `if (...)` utasítás zárójelében egy feltétel szerepel
  - logikai vagy aritmetikai kifejezés
- Ha feltétel teljesül (értéke nem nulla), akkor az `if` utáni `{ ... }` block fut le
- Ha a feltétel nem teljesült, az `else` ág fut le
  - az `else` ág nem kötelező

Feltételes elágazások, az `if` utasítás

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

- Az `if (...)` utasítás zárójelében egy feltétel szerepel
  - logikai vagy aritmetikai kifejezés
- Ha feltétel teljesül (értéke nem nulla), akkor az `if` utáni `{ ... }` block fut le
- Ha a feltétel nem teljesült, az `else` ág fut le
  - az `else` ág nem kötelező
- `exit(0);` függvényhívás
  - ez egy speciális függvény, ami azonnal kilép a programból
  - a nem nulla argumentummal hibát lehet jelezni az operációs rendszer felé

Milyen sorrendben hajtódik végre a program?

1. Az utasítások felülről lefelé hajtódnak végre
2. A program futása a `main` függvény első sorával indul
   - ekkor a parancssori argumentumok már fel lettek dolgozva, az `argv` értéke fel van töltve
3. Ha a program `if`-hez ér, akkor három dolog történhet
   - ha a feltétel teljesül, akkor a főág fut le
   - ha a feltétel nem teljesül, de van `else` ág, akkor az fut le
   - ha a feltétel nem teljesül, és nincsen `else` ág, akkor a program az `if` főágát átugorja, és az `if` utáni első utasításnál folytatódik

Többágú elágazás egymásba ágyazott if-ekkel

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

- Az `if(...)` és az `else` után egy-egy utasításblokk jön

Többágú elágazás egymásba ágyazott if-ekkel

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

- Az `if(...)` és az `else` után egy-egy utasításblokk jön
  - de lehet egy másik `if` utasítás is