Függvénypointerek

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2019. szeptember 30.

Alapprobléma

Sokszor előfordul:

- ugyanazt a feladatot többször kell végrehajtani
- egy jól beprogramozott algoritmust (függvényt) különböző feladatok esetén szeretnénk felhasználni
  - pl. sorbarendezés, diffegyenlet integrálása

Egyszerű példa: az $f(x)$ függvény értékének kiíratása

- ciklus az x értékeken fut, az $f(x)$ értékek kiírása
- ami különbözik: maga a függvény, amit kiíratunk

Témába vágó példa: differenciálegyenlet-megoldó program

- az integrátor állandó (pl.: Euler vagy RK4)
- a differenciálegyenlet feladatról feladatra változik

Függvényre mutató pointer

Egy függvényre mutató pointer értéke az a memóriacím, ahol a függvény kódja elhelyezkedik, a pointer típusa pedig a függvény típusa.

Hogyan definiálhatjuk?

```
1 double (*functpt) (int, double)
```

`functpt` egy olyan pointer, amely **double** visszatérési értékkel rendelkező függvényre mutat, melynek egy **int** és egy **double** típusú paramétere van.

Példa függvénypointer használatára

Írjunk egy olyan függvényt, amely egy tetszőleges másik függvény értékeit írja ki valamilyen intervallumon!

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void printFunct(double xfrom, double xto, double dx,
5                  double (*fp)(double))
6  {
7    double x, y;
8    for (x = xfrom; x < xto; x += dx)
9    {
10     y = (*fp)(x);
11     printf("%f %f\n", x, y);
12   }
13 }
14
15 double sqr(double x)
16 {
17 return x*x;
18 }
19
20 int main() {
21 double a=1.0;
22 double b=2.0;
23 double dx=0.1;
24
25 printf("Az sqr fuggveny ertekei az %f, %f intevallumban,"
26        "%f lepessel\n",a,b,dx);
27   printFunct(a, b, dx, &sqr);
28   return 0;
29 }
```

- a `printFunct` függvény definíciója

Példa függvénypointer használatára

Írjunk egy olyan függvényt, amely egy tetszőleges másik függvény értékeit írja ki valamilyen intervallumon!

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void printFunct(double xfrom, double xto, double dx,
5                  double (*fp)(double))
6  {
7    double x, y;
8    for (x = xfrom; x < xto; x += dx)
9    {
10     y = (*fp)(x);
11     printf("%f %f\n", x, y);
12   }
13 }
14
15 double sqr(double x)
16 {
17 return x*x;
18 }
19
20 int main() {
21 double a=1.0;
22 double b=2.0;
23 double dx=0.1;
24
25 printf("Az sqr fuggveny ertekei az %f, %f intevallumban,"
26        "%f lepessel\n",a,b,dx);
27   printFunct(a, b, dx, &sqr);
28   return 0;
29 }
```

- a `printFunct` függvény definíciója
- a `printFunct` egyik paramétere egy double típusú függvényre mutató pointer

Példa függvénypointer használatára

Írjunk egy olyan függvényt, amely egy tetszőleges másik függvény értékeit írja ki valamilyen intervallumon!

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void printFunct(double xfrom, double xto, double dx,
5                  double (*fp)(double))
6  {
7    double x, y;
8    for (x = xfrom; x < xto; x += dx)
9    {
10     y = (*fp)(x);
11     printf("%f %f\n", x, y);
12   }
13 }
14
15 double sqr(double x)
16 {
17 return x*x;
18 }
19
20 int main() {
21 double a=1.0;
22 double b=2.0;
23 double dx=0.1;
24
25 printf("Az sqr fuggveny ertekei az %f, %f intevallumban,"
26        "%f lepessel\n",a,b,dx);
27   printFunct(a, b, dx, &sqr);
28   return 0;
29 }
```

- a `printFunct` függvény definíciója
- a `printFunct` egyik paramétere egy double típusú függvényre mutató pointer
- a függvénypointer értékének kiolvasása

Példa függvénypointer használatára

Írjunk egy olyan függvényt, amely egy tetszőleges másik függvény értékeit írja ki valamilyen intervallumon!

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void printFunct(double xfrom, double xto, double dx,
5                  double (*fp)(double))
6  {
7    double x, y;
8    for (x = xfrom; x < xto; x += dx)
9    {
10     y = (*fp)(x);
11     printf("%f %f\n", x, y);
12   }
13 }
14
15 double sqr(double x)
16 {
17 return x*x;
18 }
19
20 int main() {
21 double a=1.0;
22 double b=2.0;
23 double dx=0.1;
24
25 printf("Az sqr fuggveny ertekei az %f, %f intevallumban,"
26        "%f lepessel\n",a,b,dx);
27   printFunct(a, b, dx, &sqr);
28   return 0;
29 }
```

- a `printFunct` függvény definíciója
- a `printFunct` egyik paramétere egy double típusú függvényre mutató pointer
- a függvénypointer értékének kiolvasása
- az `sqr` függvény definíciója

Példa függvénypointer használatára

Írjunk egy olyan függvényt, amely egy tetszőleges másik függvény értékeit írja ki valamilyen intervallumon!

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void printFunct(double xfrom, double xto, double dx,
5                  double (*fp)(double))
6  {
7    double x, y;
8    for (x = xfrom; x < xto; x += dx)
9    {
10     y = (*fp)(x);
11     printf("%f %f\n", x, y);
12   }
13 }
14
15 double sqr(double x)
16 {
17 return x*x;
18 }
19
20 int main() {
21 double a=1.0;
22 double b=2.0;
23 double dx=0.1;
24
25 printf("Az sqr fuggveny ertekei az %f, %f intevallumban,"
26        "%f lepessel\n",a,b,dx);
27   printFunct(a, b, dx, &sqr);
28   return 0;
29 }
```

- a `printFunct` függvény definíciója
- a `printFunct` egyik paramétere egy double típusú függvényre mutató pointer
- a függvénypointer értékének kiolvasása
- az `sqr` függvény definíciója
- a `main` függvényben meghívjuk a `printFunct` függvényt, hogy kiírjuk az `sqr` által számolt eredményeket

A **typedef** használata

A typedef használatával olvashatóbbá lehet tenni a forráskódot.

```
1
2   #include<stdio.h>
3   #include<stdlib.h>
4
5   typedef double MATFV(double x); //egy double erteku matematikai fuggveny tipusa
6
7   void printFunct(double, double, double,  MATFV *); // printFunct prototipusa,
8                                                      // MATFV * jelzi a fuggvenypointert
9
10  double sqr(double x);   // az sqr fuggveny prototipusa
11
12
13  int main()
14  {
15  double a=1.0;
16  double b=2.0;
17  double dx=0.1;
18
19  printf("Az sqr fuggveny ertekei az %f, %f intevallumban, %f lepessel\n",a,b,dx);
20    printFunct(a, b, dx, &sqr);
21    return 0;
22  }
23
24  // a printFunct definicioja
25   void printFunct(double xfrom, double xto, double dx, MATFV *fp)
26   {
27    //utasitasok
28     }
29
30  // az sqr fuggveny definicioja
31   double sqr(double x)
32    {
33    return x*x;
34    }
```

Differenciálegyenlet-rendszer megadása

Közönséges differenciálegyenletek rendszere:

$$\frac{\mathrm{d}y_i}{\mathrm{d}t} = f_i(t, y_1, y_2, ..., y_i)$$

Program írásakor

- A koordinátákat nem érdemes külön-külön változóban tárolni, tegyük be mindet egy vektorba
- ugyanígy az egyenlet paramétereivel is
- az f_i függvényeket kell megírni
- érdemes az integrátor függvényt úgy megírni, hogy paraméterként egy függvényt vár, ami ki tudja számolni az f_i deriváltakat

Példa: harmonikus oszcillátor egyenlete

Deriváltak kiszámolása:

```
1  void harmonikusOszcillator(
2      double* param,
3      double* y,
4      double* dy)
5  {
6      double k = param[0];
7      double m = param[1];
8      double x = y[0];
9      double v = y[1];
10     dy[0] = v;
11     dy[1] = - k / m * x;
12 }
```

Ahol `y[0]` az előző időpontban vett hely és `y[1]` a sebesség.

Függvény a deriváltak kiszámolására

Általában a deriváltak függhetnek a független változótól (pl idő) is
Bonyolultabb függvény esetén érdemes a `typedef`-t használni

```
1  typedef void ODE(
2      double*,          // parameterek
3      double,           // fuggetlen valtozo
4      double*,          // fuggo valtozok
5      double*,          // derivaltak (kimeno)
6      int);             // egyenletek szama
```

Függvény a deriváltak kiszámolására

Általában a deriváltak függhetnek a független változótól (pl idő) is
Bonyolultabb függvény esetén érdemes a `typedef`-t használni

```
1  typedef void ODE(
2      double*,          // parameterek
3      double,           // fuggetlen valtozo
4      double*,          // fuggo valtozok
5      double*,          // derivaltak (kimeno)
6      int);             // egyenletek szama
```

Ennek segítségével pl az integrátor függvény prototípusa

```
1  void eulerLepes(
2      double t,         // fuggetlen valtozo
3      double dt,        // lepeshossz
4      double* p,        // parameterek
5      double* y,        // valtozok
6      double* dy,       // derivaltak
7      int n,            // egyenletek szama
8      ODE *);           // egyenletek
```