

Első C programok

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2021. szeptember 8.

Miből áll egy program?

1 Adatmodell

- a memóriaterületet a konkrét feladat céljára alakíthatjuk ki
- a tárterület logikai kiosztása az adatmodell
- a logikai adatmodellt a programnyelvvvel segítségével írjuk le
- az adatmodellt a memóriában a fordítóprogram valósítja meg

2 Algoritmus

- egy műveleti **folyamatsort** ír le, amit az adatokon el kell végezni
- az algoritmus a programnyelv utasításaival írjuk le
- pl. lehetnek benne **ciklusok** és **feltételes elágazások**
- az algoritmus a processzor kódjára a fordítóprogram alakítja át

3 Kimenet és bemenet

- az adatmodellt fel kell tölteni adattal (pl. fájlból)
- az algoritmus eredményét ki kell írni

Példa adatmodellre és algoritmusra 1.

Másodfokú egyenlet megoldóképlete

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

Adatmodell

- három valós szám a memóriában az együtthatóknak
- egy valós szám a diszkriminánsnak
- két valós szám a gyököknek

Algoritmus:

- olvasd be a három számot
- számítsd ki a diszkrimináns
- ha a diszkrimináns negatív, írd ki egy hibát, és állj meg
- ha a diszkrimináns nulla, írd ki a gyököt és állj meg
- ha a diszkrimináns pozitív, írd ki a két gyököt és állj meg

Mátrix elemeinek kiírása

Adatmodell

- két egész szám M , N a mátrix méretének tárolására
- két egész szám, amivel a mátrixelemeket indexelni tudjuk
- $M \times N$ valós szám a mátrix elemeinek tárolására
- a mátrix elemeit tároljuk soronként, a memóriában folytonosan!

Mátrixkiírás algoritmus:

- olvasd be a mátrixot
- egy ciklussal futtasd az i indexet 0 és M között
- egy belső ciklussal futtasd a j indexet 0 és N között
- számítsd ki az i és j indexekből a mátrixelem memóriacímét
- írd ki a mátrixelemet
- ha a belső ciklus a végére ért, tegyél be egy sortörést

Az adatmodell és az algoritmus erősen összefügg

- pl. nem mindegy, hogy a mátrixot a memóriában soronként vagy oszloponként tesszük el (pl. FORTRAN)
- ezért a kettőt egyszerre kell kitalálni

Az adatmodell megszabhatja az algoritmus futási idejét

- pl. ha valamit meg kell keresni egy listában, nem mindegy, hogy a listát milyen módon tároljuk

Számos univerzális adatmodell létezik, amikből a legtöbb algoritmus építkezik

- ezek az adatmodellek magukból az adatstruktúrákból és az azokat kezelő elemi algoritmusokból állnak
- pl. lista megvalósítása, új elem hozzáfűzése, elem kivétele

A minimális C program

A minimális C program a `main` függvény megvalósítását jelenti:

```
1 int main(int argc, char* argv[]) {  
2     return 0;  
3 }
```

A minimális C program

A minimális C program a `main` függvény megvalósítását jelenti:

```
1 int main(int argc, char* argv[]) {  
2     return 0;  
3 }
```

- `main`: ez a függvény neve
 - a program végrehajtása mindig a `main`-nel kezdődik
 - a függvénynév általában tetszőleges karaktersor, a `main` speciális
- `int`: ez a függvény visszatérési értékének típusa, később részletesebben megtárgyaljuk
 - az `int` jelentése két bájtos egész szám
 - a `main` esetében mindig `int` a visszatérési érték típusa
 - később majd látjuk, hogy általában mire való a visszatérési érték
- `return 0;` - a függvény visszatérési értéke
 - a `main` esetében ez magának a programnak a visszatérési kódja
 - általában 0, ha nem 0, akkor valami hibakódot jelent
- `(int argc, char* argv[])` - parancssori argumentumok
 - ezek a `main` függvény paraméterei
 - ezekre később visszatérünk

Egy másik egyszerű példa

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```


Egy másik egyszerű példa

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```

- A C-ben nincsen *beépített* függvény
 - de vannak standard könyvtárakat
 - ezeket az `#include` direktívával kell meghivatkozni

Egy másik egyszerű példa

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```

- A C-ben nincsen *beépített* függvény
 - de vannak standard könyvtárakat
 - ezeket az `#include` direktívával kell meghivatkozni
- Ha nem akarunk a parancssorról beolvasni a `main` argumentumlistája üres

Egy másik egyszerű példa

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```

- A C-ben nincsen *beépített* függvény
 - de vannak standard könyvtárakat
 - ezeket az **#include** direktívával kell meghivatkozni
- Ha nem akarunk a parancssorról beolvasni a **main** argumentumlistája üres
- **printf** függvény
 - szöveg kiírása a konzolablakba
 - a függvényhívás a függvény nevéből áll, melyet egy zárójeles rész követ
 - a zárójelek között a paramétereket kell felsorolni
 - paraméter lehet egyetlen változó, de összetett kifejezés is

Másodfokú egyenlet megoldóképlete

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

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     double a, b, c;
6     double d, r1, r2;
7     a = 1.0;
8     b = 3.0;
9     c = 2.0;
10    d = b * b - 4 * a * c;
11    d = sqrt(d);
12    r1 = (-b + d) / (2 * a);
13    r2 = (-b - d) / (2 * a);
14    printf("r1 = %f, r2 = %f\n", r1, r2);
15    return 0;
16 }
```

Az első valódi program

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     double a, b, c;
6     double d, r1, r2;
7     a = 1.0;
8     b = 3.0;
9     c = 2.0;
10    d = b * b - 4 * a * c;
11    d = sqrt(d);
12    r1 = (-b + d) / (2 * a);
13    r2 = (-b - d) / (2 * a);
14    printf("r1 = %f, r2 = %f\n", r1, r2);
15    return 0;
16 }
```

- standard könyvtárak

Az első valódi program

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     double a, b, c;
6     double d, r1, r2;
7     a = 1.0;
8     b = 3.0;
9     c = 2.0;
10    d = b * b - 4 * a * c;
11    d = sqrt(d);
12    r1 = (-b + d) / (2 * a);
13    r2 = (-b - d) / (2 * a);
14    printf("r1 = %f, r2 = %f\n", r1, r2);
15    return 0;
16 }
```

- standard könyvtárak
- változók **deklarálása** és értékadás
 - itt valójában ennyi az adatmodell: öt szám
 - **double** - dupla pontosságú tört szám

Az első valódi program

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     double a, b, c;
6     double d, r1, r2;
7     a = 1.0;
8     b = 3.0;
9     c = 2.0;
10    d = b * b - 4 * a * c;
11    d = sqrt(d);
12    r1 = (-b + d) / (2 * a);
13    r2 = (-b - d) / (2 * a);
14    printf("r1 = %f, r2 = %f\n", r1, r2);
15    return 0;
16 }
```

- standard könyvtárak
- változók **deklarálása** és értékadás
 - itt valójában ennyi az adatmodell: öt szám
 - **double** - dupla pontosságú tört szám
- változók kezdőértékeinek megadása (inicializálás)
 - C-ben mindig inicializáljuk a változókat!
 - később megnézzük, hogyan lehet a paramétereket kívülről beolvasni

Az első valódi program

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     double a, b, c;
6     double d, r1, r2;
7     a = 1.0;
8     b = 3.0;
9     c = 2.0;
10    d = b * b - 4 * a * c;
11    d = sqrt(d);
12    r1 = (-b + d) / (2 * a);
13    r2 = (-b - d) / (2 * a);
14    printf("r1 = %f, r2 = %f\n", r1, r2);
15    return 0;
16 }
```

- standard könyvtárak
- változók **deklarálása** és értékadás
 - itt valójában ennyi az adatmodell: öt szám
 - **double** - dupla pontosságú tört szám
- változók kezdőértékeinek megadása (inicializálás)
 - C-ben mindig inicializáljuk a változókat!
 - később megnézzük, hogyan lehet a paramétereket kívülről beolvasni
- matematikai műveletek elvégzése

Az első valódi program

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     double a, b, c;
6     double d, r1, r2;
7     a = 1.0;
8     b = 3.0;
9     c = 2.0;
10    d = b * b - 4 * a * c;
11    d = sqrt(d);
12    r1 = (-b + d) / (2 * a);
13    r2 = (-b - d) / (2 * a);
14    printf("r1 = %f, r2 = %f\n", r1, r2);
15    return 0;
16 }
```

- standard könyvtárak
- változók **deklarálása** és értékadás
 - itt valójában ennyi az adatmodell: öt szám
 - **double** - dupla pontosságú tört szám
- változók kezdőértékeinek megadása (inicializálás)
 - C-ben mindig inicializáljuk a változókat!
 - később megnézzük, hogyan lehet a paramétereket kívülről beolvasni
- matematikai műveletek elvégzése
- eredmények **formázott** kiírása a konzolablakba ("%f")

Az első valódi program

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     double a, b, c;
6     double d, r1, r2;
7     a = 1.0;
8     b = 3.0;
9     c = 2.0;
10    d = b * b - 4 * a * c;
11    d = sqrt(d);
12    r1 = (-b + d) / (2 * a);
13    r2 = (-b - d) / (2 * a);
14    printf("r1 = %f, r2 = %f\n", r1, r2);
15    return 0;
16 }
```

- standard könyvtárak
- változók **deklarálása** és értékadás
 - itt valójában ennyi az adatmodell: öt szám
 - **double** - dupla pontosságú tört szám
- változók kezdőértékeinek megadása (inicializálás)
 - C-ben mindig inicializáljuk a változókat!
 - később megnézzük, hogyan lehet a paramétereket kívülről beolvasni
- matematikai műveletek elvégzése
- eredmények **formázott** kiírása a konzolablakba ("%f")
- a program visszatérési értéke 0

A program lefordítása és futtatása

Ha a program a 1.c fájlban van, akkor a fordítás menete:

- `gcc 1.c -o firstprog -lm`

```
1 $ ls -lt
2 -rwxrwx--- 1 kor kor 286 szept 3 10:53 1.c
3 -rw-r--r-- 1 kor kor 94 szept 3 10:38 1.py
4 $ gcc 1.c -o firstprog -lm
```

Ez a parancs létrehoz egy `firstprog` nevű futtatható fájlt:

```
1 $ ls -lt
2 -rwxrwxr-x 1 kor kor 8344 szept 3 11:00 firstprog
3 -rwxrwx--- 1 kor kor 286 szept 3 10:53 1.c
4 -rw-r--r-- 1 kor kor 94 szept 3 10:38 1.py
```

Ezek után jön a futtatás:

```
1 $ ./firstprog
2 r1 = -1.000000, r2 = -2.000000
```

Az első program Python-ban

```
1 a = 1.0
2 b = 3.0
3 c = 2.0
4 d = sqrt(b**2-4*a*c)
5 r1 = (-b+d)/(2*a)
6 r2 = (-b-d)/(2*a)
7 print(r1,r2)
```

- A Python gyakorlattal szemben C-ben az egész program egy függvény.
- Pythonban nem kellett deklarálni, C-ben kötelező
- C-ben az utasítások végére ";" kell, Pythonban nem kell
- Pythonban nem kell format string a kiíráshoz