Első C++ program

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2023 szeptember 12

Miből áll egy program?

1. Adatmodell
   - a számítógép memóriaterületét egy konkrét feladat céljára alakíthatjuk ki
   - a memóriaterület logikai kiosztása az adatmodell
   - a logikai adatmodellt a programnyelv 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
   - lehetnek benne pl. **ciklusok**, **feltételes elágazások**, stb.
   - az algoritmust a processzor által futtatható kódra 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 a, b, c együtthatóknak
- egy valós szám a diszkriminánsnak
- két valós szám a gyököknek

Algoritmus:

- olvasson be a három számot
- számítsa ki a diszkriminánst
- ha a diszkrimináns negatív, írjon ki egy hibaüzenetet és álljon meg
- ha a diszkrimináns nulla, írja ki a gyököt és álljon meg
- ha a diszkrimináns pozitív, írja ki a két gyököt és álljon meg

Példa adatmodellre és algoritmusra 2.

Mátrixban tárolt adatok 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:

- egy ciklussal fusson az i indexet 0 és M között
- egy belső ciklussal futtasson a j indexet 0 és N között
- számítsa ki az i és j indexekből a mátrixelem memóriacímét
- írja ki a mátrixelemet
- ha a belső ciklus a végére ért, kezdjen új sort a kiírás

Az adatmodell és az algoritmus kapcsolata

Az adatmodell és az algoritmus erősen összefügg

- pl. nem mindegy, hogy a mátrixot a memóriában sorfolytonosan (pl. C nyelv) vagy oszlopfolytonosan(pl. FORTRAN) tároljuk
- 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 C++ nyelv elemei

Egy C++ kódban következő elemekkel találkozhatunk:

- preprocesszor direktívák
- {...} blokkok
- szimbólumok deklarációi és definíciói
- karakterláncok
- függvények
- származtatott adattípusok

A minimális C++ program

A minimális C++ program a `main()` függvény megvalósítását jelenti:

```
1  int main() {
2    return 0;
3  }
```

A minimális C++ program

A minimális C++ program a `main()` függvény megvalósítását jelenti:

```
1  int main() {
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
  - mindig meg kell adni, milyen típusú eredményt ad vissza egy függvény
  - az `int` jelentése 4 bájtos egész szám (lásd később)
  - a `main` esetében mindig `int` a visszatérési érték típusa
- `return 0;` - a függvény visszatérési értéke
  - általában 0, ha nem 0, akkor valami hibakódot jelenthet (operációs rendszertől függően)
- `main()` ennek a függvénynek nincsennek bemenő paraméterei
  - bemenő paramétereket a `(...)` között lehet felsorolni
  - pl lehet átadni parancssori argumentumokat a `main`-nek
  - erre később visszatérünk
- `{...}` a függvény törzsét kapcsos zárójelek jelölik ki

“Hello world” példa

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7     cout << "Hello world!" << endl;
8     return 0;
9  }
```

‘Hello world” példa

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello world!" << endl;
  return 0;
}
```

- Sok feladat előre megírt, standard könyvtárak segítségével oldható meg
  - ezeket az `#include` direktívával kell meghivatkozni
  - a `<...>` közötti név a használni kívánt könyvtár ún. **header**-je
  - lényegében egy külső kódrészletet tudunk ezzel beilleszteni

‘Hello world” példa

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello world!" << endl;
  return 0;
}
```

- Sok feladat előre megírt, standard könyvtárak segítségével oldható meg
  - ezeket az `#include` direktívával kell meghivatkozni
  - a `<...>` közötti név a használni kívánt könyvtár ún. **header**-je
  - lényegében egy külső kódrészletet tudunk ezzel beilleszteni
- Előre megírt könyvtárat használunk pl. az standard adatbevitelre/kiírásra (input/output)
  - a kiíráshoz a `cout` objektumot használjuk (**standard output**)
  - a `endl` egy másik Standard Library objektum, ún **manipulator** (a helyes kiíratáshoz kell)

‘Hello world” példa

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello world!" << endl;
  return 0;
}
```

- Sok feladat előre megírt, standard könyvtárak segítségével oldható meg
  - ezeket az `#include` direktívával kell meghivatkozni
  - a `<...>` közötti név a használni kívánt könyvtár ún. **header**-je
  - lényegében egy külső kódrészletet tudunk ezzel beilleszteni
- Előre megírt könyvtárat használunk pl. az standard adatbevitelre/kiírásra (input/output)
  - a kiíráshoz a `cout` objektumot használjuk (**standard output**)
  - a `endl` egy másik Standard Library objektum, ún **manipulator** (a helyes kiíratáshoz kell)
- A `«` egy operátor
  - egy adott értéket beküldhetünk vele a `cout`-ba, vagyis a standard kimentre

‘Hello world” példa

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello world!" << endl;
  return 0;
}
```

- “Hello world“ egy karaktersor (literal), ezt akarjuk kiíratni
  - a kiíratáshoz a « operátorral beleküldjük a cout-ba
  - majd pedig a endl-t is ”utánnaküldjük“

‘Hello world” példa

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello world!" << endl;
  return 0;
}
```

- “Hello world“ egy karaktersor (literal), ezt akarjuk kiíratni
  - a kiíratáshoz a « operátorral beleküldjük a `cout`-ba
  - majd pedig a `endl`-t is ”utánnaküldjük“
- Definiálhatunk ún namespace-eket, vagyis névtéreket

Mire szolgálnak a névterek?

- a különböző típusú objektumokat, pl változókat, függvényeket stb csoportosíthajuk
- pl különböző feladatokra definiált, de azonos nevű függvények használatakor pontosan meg tudjuk adni, melyik függvényt akarjuk meghívni
- itt most az **std** névtérben definiált `cout` és `endl` objektumokat akarjuk használni: using namespace **std**;
- ez a névtér a iostream-ben van definiálva

Namespace

Példa namespace definiálásra:

```
namespace A
{
 int a;
 double peldafunc(double);
}
```

Namespace használatának másik szintakszisa

```
1  #include <iostream>
2
3  // using namespace std;//
4
5  int main()
6  {
7      std::cout << "Hello world!" << std::endl;
8      return 0;
9  }
```

- //.....// paranccsal megjegyzéseket helyezhetünk el a forráskódban
- a megjegyzések nem kerülnek feldolgozásra a program futásakor
- std::cout jelentése: a **std** namespace-ben található cout objektumot használja

A program fordítása

- a forráskód még nem futtatható
- **le kell fordítani** egy fodítóprogrammal, hogy futtatható állomány keletkezzen

Fordítók Linux (Mac, Windows) platformon

- Gnu Compiler Collection (gcc)
- Clang (LLVM)

Online fordítók

- Tutorialspoint
- Ideone
- coliru

A fóliákon g++-t és a `clang`-ot fogjuk használni

A program fordítása és futtatása parancssorból

Ha a program a `hello.cpp` fájlban van, akkor a fordítás menete Linux-on:

- érdemes kipróbálni kétféle fordítóprogramot is, segíthet a hibaüzenetek megértésében
- pl g++ és `clang`
  - `g++ -Wall hello.cpp -o hello`
  - `clang++ -Wall -std=c++14 hello.cpp -o hello`

- a fordítónak különböző opciók is átadhatóak
  - pl `-Wall`: figyelmeztetéseket is írjon ki, ne csak kritikus hibákat
  - `-std=c++14`: milyen nyelvi standard szerint történjen a fordítás
  - `-o`: ezután adhatjuk meg a futtatható állomány nevét

A program fordítása és futtatása parancssorból

Ha a program a `hello.cpp` fájlban van, akkor a fordítás menete Linux-on:

- érdemes kipróbálni kétféle fordítóprogramot is, segíthet a hibaüzenetek megértésében
- pl g++ és `clang`
  - `g++ -Wall hello.cpp -o hello`
  - `clang++ -Wall -std=c++14 hello.cpp -o hello`

- a fordítónak különböző opciók is átadhatóak
  - pl `-Wall`: figyelmeztetéseket is írjon ki, ne csak kritikus hibákat
  - `-std=c++14`: milyen nyelvi standard szerint történjen a fordítás
  - `-o`: ezután adhatjuk meg a futtatható állomány nevét

A fenti parancs létrehoz egy `hello` nevű futtatható fájlt
Ezek után jön a futtatás:

```
1 $ ./hello
```

Ilymódon lehet ellenőrizni a k8plex-en is, hogy rendben lefordul a program!