

Programozás

(GKxB_INTM114)

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM114.git

2024. április 8.

Feladat:

- két város nevének beolvasása,
- városok közötti távolság megjelenítése.
- Kilépés azonos városok megadása esetén.

Megoldás:

- városok nevét vektorban tároljuk
- a városok közötti távolságokat pedig mátrixban, és
- a városok vektorbeli indexével indexeljük a mátrixot is

Kimenet

Varosok kozotti tavolsag kiszamitasa

Kilepes azonos varosok megadasaval.

Indulo varos: Budapest

Erkezesi varos: Salakszentmotoros

Nem letezo varos!

Gyor

Tavolsag: 121km

Indulo varos: Gyor

Erkezesi varos: Gyor

varosok1.cpp

```
36  int main() {
37      int honnan, hova;
38      cout << "Varosok kozotti tavolsag kiszamitasa\n"
39           << "Kilepes azonos varosok megadasaval.\n";
40      do {
41          cout << "Indulo varos: "; honnan = varosldx();
42          cout << "Erkezesi varos: "; hova = varosldx();
43          if(honnan != hova) {
44              cout << "Tavolsag: "
45                   << tav(honnan, hova) << "km\n";
46          }
47      } while(honnan != hova);
48      return 0;
49  }
```

varosok1.cpp

```
5 #define VAROSOK 7
6 int varosldx() {
7     string vlista[VAROSOK] = {
8         "Budapest", "Gyor", "Szeged", "Debrecen",
9         "Veszprem", "Dunaujvaros", "Eger"
10    };
11    string vnev;
12    do {
13        cin >> vnev;
14        for(int i=0; i<VAROSOK; i++) {
15            if(vlista[i] == vnev) {
16                return i;
17            }
18        }
19        cout << "Nem letezo varos!\n";
20    } while(true);
21 }
```

varosok1.cpp

```

23 int tav(int honnan, int hova) {
24     int tavMtx[VAROSOK][VAROSOK] = {
25         { 0, 121, 174, 231, 115, 83, 139 },
26         { 121, 0, 287, 377, 82, 176, 285 },
27         { 174, 287, 0, 218, 278, 161, 298 },
28         { 231, 377, 218, 0, 368, 320, 131 },
29         { 115, 82, 278, 368, 0, 103, 275 },
30         { 83, 176, 161, 320, 103, 0, 228 },
31         { 139, 285, 298, 131, 275, 228, 0 }
32     };
33     return tavMtx[honnan][hova];
34 }

```

Észrevétel: a mátrixnak több, mint fele elhagyható! ($m_{i,j} = m_{j,i}$, $m_{i,i} = 0$)

	Budapest	Győr	Szeged	Debrecen	Veszprém	Dunaújváros	Eger
Budapest	0	121	174	231	115	83	139
Győr	121	0	287	377	82	176	285
Szeged	174	287	0	218	278	161	298
Debrecen	231	377	218	0	368	320	131
Veszprém	115	82	278	368	0	103	275
Dunaújváros	83	176	161	320	103	0	228
Eger	139	285	298	131	275	228	0

Probléma: a mátrixnak minden sora azonos elemszámú

Megoldás: eltérő elemszámú vektorokat címző vektor létrehozása (mutatótömb, alsó háromszögmátrix)

Észrevétel: még a vektorokat címző mutatók is megtakaríthatók, ha sorfolytonosan, egyetlen vektorban tároljuk a főátló alatti értékeket!

	Bp. [0]	Győr [1]	Szeged [2]	Debr. [3]	Veszp. [4]	Duv. [5]
Győr [1]	121					
Szeged [2]	174	287				
Debrecen [3]	231	377	218			
Veszprém [4]	115	82	278	368		
Dunaújváros [5]	83	176	161	320	103	
Eger [6]	139	285	298	131	275	228

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
121	174	287	231	377	218	115	82	278	...

Keresett elem feletti sorokban lévő adatok száma sorról sorra számtani sort alkot

$$S_n = \frac{[2a_1 + (n-1)d] \cdot n}{2}, \text{ speciálisan } a_1 = 1, d = 1 \rightarrow S_n = \frac{(n+1) \cdot n}{2}$$

varosok3.cpp

```

24  int tav(int honnan, int hova) {
25      if(honnan == hova) return 0;
26      int tavMtx[(VAROSOK*(VAROSOK-1))/2] = {
27          121,
28          174, 287,
29          231, 377, 218,
30          115, 82, 278, 368,
31          83, 176, 161, 320, 103,
32          139, 285, 298, 131, 275, 228
33      };
34      if(honnan < hova) {
35          int csere = honnan;
36          honnan = hova;
37          hova = csere;
38      }
39      return tavMtx[(honnan)*(honnan-1)/2 + hova];
40  }

```

Probléma:

- a lokális változók alapértelmezetten *automatikusak* (ld. `auto`, C++11-ig):
élettartam a definíció pillanatától a blokk elhagyásáig tart
- a vektorok és tömbök újrafoglalása időigényes

Megoldás: `static` minősítő használata

- élettartam: program indulásától leállásig → értéküket megőrzik a függvényhívások között
- hatókör: változatlan (deklarációtól blokk végéig)
- implicit módon inicializált (feltöltés zérus értékű bitekkel)

varosok4.cpp

```
6  int varosIdx() {
7      static string vlista[VAROSOK] = {
8          "Budapest", "Gyor", "Szeged",
9          "Debrecen", "Veszprem",
10         "Dunaujvaros", "Eger"
11     };
12     string vnev;
13     do {
14         cin >> vnev;
15         for(int i=0; i<VAROSOK; i++) {
16             if(vlista[i] == vnev) {
17                 return i;
18             }
19         }
20         cout << "Nem letezo varos!\n";
21     } while(true);
22 }
```

Feladat:

- hozzuk létre futásidőben a vektort és a háromszögmátrixot!
- töltsük fel őket felhasználó által adott értékekkel!

varosok5.cpp

```
57 int main() {
58     string* varosLista;
59     int varosDb;
60     int** varosTavok;
61     int honnan, hova;
62     cout << "Varosok kozotti tavolsag kiszamitasa\n"
63          << "Varosok szama: ";
64     cin >> varosDb;
65     varosLista = varosBe(varosDb);
66     cout << "Adja meg a varosok kozti tavokat!\n";
67     varosTavok = tavokBe(varosLista, varosDb);
68     cout << "Kilepes azonos varosok megadasaval.\n";
```

varosok5.cpp – main

```
69  do {
70      cout << "Indulo varos: ";
71      honnan = varosIdx(varosLista, varosDb);
72      cout << "Erkezesi varos: ";
73      hova = varosIdx(varosLista, varosDb);
74      if(honnan != hova) {
75          cout << "Tavolsag: "
76              << tav(varosTavok, honnan, hova) << "km\n";
77      }
78  } while(honnan != hova);
79  tavokFelszab(varosTavok, varosDb);
80  delete[] varosLista;
81  return 0;
82 }
```

varosok5.cpp

```
5  string* varosBe(int db) {
6      string* vlista = new string[db];
7      for(int i=0; i<db; i++) {
8          cout << i+1 << ". varos neve: ";
9          cin >> vlista[i];
10     }
11     return vlista;
12 }
```

varosok5.cpp

```

27 int varosldx(const string* vlista , int db) {
28     string vnev;
29     do {
30         cin >> vnev;
31         for(int i=0; i<db; i++) {
32             if(vlista[i] == vnev) {
33                 return i;
34             }
35         }
36         cout << "Nem letezo varos!\n";
37     } while(true);
38 }

```


varosok5.cpp

```

14  int** tavokBe(const string* vlista , int db) {
15      int** tavMtx = new int*[db-1];
16      for(int honnan=1; honnan<db; honnan++) {
17          tavMtx[honnan-1] = new int[honnan];
18          for(int hova=0; hova<honnan; hova++) {
19              cout << vlista[honnan] << " —> "
20                  << vlista[hova] << ": ";
21              cin >> tavMtx[honnan-1][hova];
22          }
23      }
24      return tavMtx;
25  }
```

varosok5.cpp

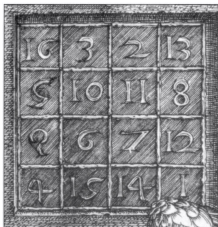
```
40 int tav(int** tavMtx, int honnan, int hova) {
41     if(honnan == hova) return 0;
42     if(honnan < hova) {
43         int csere = honnan;
44         honnan = hova;
45         hova = csere;
46     }
47     return tavMtx[honnan-1][hova];
48 }
```

varosok5.cpp

```
50 void tavokFelszab(int** tavMtx, int db) {
51     for(int i=0; i<db-1; i++) {
52         delete [] tavMtx[i];
53     }
54     delete [] tavMtx;
55 }
```

Tulajdonságok

- Bűvös négyzet: olyan (ált. 1 és n^2 közötti) egész számokat tartalmazó négyzetes mátrix, melynek
 - minden sorösszege,
 - minden oszlopösszege,
 - főátlójában és
 - mellékátlójában lévő számok összege azonos.
- További érdekességek



Albrecht Dürer: Melencolia I (részlet)



Sagrada Família, Barcelona

Páratlan rendű bűvös négyzetek konstrukciója

- 1 A mátrix első sorának középső oszlopába írjunk 1-et!
- 2 A mátrix minden további elemének értéke legyen eggyel nagyobb a korábbinál ($2, 3, \dots, n^2$)!
- 3 A következő elemet úgy választjuk ki, hogy **jobbra és felfelé** lépünk egyet.
 - Ha a meghatározott elem már korábban ki lett töltve, akkor az utoljára kitöltött elem **alatti** elemmel kell folytatni a műveletet.
 - Ha az így meghatározott elem kívül esne a mátrixon, akkor a szemközti oldalon lévő első elemet kell használni (pl. a „legfelső feletti” sor esetén a legalsót).

1. lépés			2. lépés			3. lépés			4. lépés			5. lépés			6. lépés		
	1			1			1			1			1			1	6
						3			3			3	5		3	5	
					2			2	4		2	4		2	4		2

7. lépés			8. lépés			9. lépés		
	1	6	8	1	6	8	1	6
3	5	7	3	5	7	3	5	7
4		2	4		2	4	9	2

buvos.cpp

```
46 int main(void) {
47     int meret;
48     do {
49         cout << "Buvos negyzet merete: ";
50         cin >> meret;
51     } while(meret%2 == 0);
52     int** buvos = eloallit(meret);
53     nyomtat(buvos, meret);
54     felszabadit(buvos, meret);
55     return 0;
56 }
```

buvos.cpp

```
4 // Csak paratlan rendu matrixszal mukodik!
5 int** eloallit(int meret) {
6     // Memoriafoglalas
7     int** mtx = new int*[meret];
8     for(int s=0; s<meret; s++) {
9         mtx[s] = new int[meret];
10        for(int o=0; o<meret; o++) {
11            mtx[s][o] = 0;
12        }
13    }
```

buvos.cpp

```
14 // Feltoltes
15 int s=0, o=meret/2;
16 for(int n=1; n<=meret*meret; n++) {
17     mtx[s][o] = n;
18     int i = s-1; if(i== -1) i=meret-1;
19     int j = o+1; if(j==meret) j=0;
20     if(mtx[i][j] != 0) {
21         s++;
22     } else {
23         s = i;
24         o = j;
25     }
26 }
27 return mtx;
28 }
```


buvos.cpp

```
30 void nyomtat(int** mtx, int meret) {
31     for(int s=0; s<meret; s++) {
32         for(int o=0; o<meret; o++) {
33             cout << mtx[s][o] << '\\t';
34         }
35         cout << endl;
36     }
37 }
38
39 void felszabadit(int** mtx, int meret) {
40     for(int s=0; s<meret; s++) {
41         delete [] mtx[s];
42     }
43     delete [] mtx;
44 }
```