Programozás
(GKxB_INTM114)

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

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

2024. április 1.

Feladat:

- Fejlesszük tovább úgy a buborék rendezőalgoritmust bemutató példát, hogy a felhasználó adhassa meg a rendezendő adatokat! (Adatbevitelnek vége negatív adatra.)
- Nem adhat meg több számot, mint a tömb elemszáma!

Problémák:

- Fordítási időben tudni kellene a rendezendő adatok számát
- Túl kicsi tömb $\rightarrow$ nem férnek el az adatok
- Túl nagy tömb $\rightarrow$ pazaroljuk a memóriát
- Inkább nagyobb legyen a tömb, mint túl kicsi!

Kimenet – leállás végjelre

```
Adjon meg nemnegativ szamokat!
1. szam: 2
2. szam: 4
3. szam: 1
4. szam: 3
5. szam: -1
Rendezes utan:
1       2       3       4
```

Kimenet – leáll amikor a tömb megtelik

```
Adjon meg nemnegativ szamokat!
1. szam: 5
2. szam: 4
3. szam: 3
4. szam: 2
5. szam: 1
Rendezes utan:
1       2       3       4       5
```

buborek5.cpp

```
3  #define OSSZES 5

37 int main() {
38   int hasznal; // Ennyi elemet hasznalunk eppen
39   int szamok[OSSZES];
40   cout << "Adjon meg nemnegativ szamokat!\n";
41   beolvas(szamok, &hasznal);
42   buborek(szamok, hasznal);
43   cout << "Rendezes utan:\n";
44   tombKiir(szamok, hasznal);
45   return 0;
46 }
```

buborek5.cpp

```
5  void beolvas(int* szamok, int* hasznal) {
6    int uj;
7    *hasznal = 0;
8    do {
9      cout << *hasznal + 1 << ". szam: ";
10     cin >> uj;
11     if(uj>=0 and *hasznal<OSSZES) {
12       *(szamok + *hasznal) = uj;
13       (*hasznal)++;
14     }
15   } while(uj>=0 and *hasznal<OSSZES);
16 }
```

Dinamikus memóriakezelés

- A programozó dönt a dinamikus változók élettartamáról
- Memória foglalás:
  - C++ `new` operátorral → megfelelő típusú mutatót ad vissza, nincs szükség típuskényszerítésre
  - C `malloc()`, `calloc()` függvények, `void*` mutatóval térnek vissza
- Terület felszabadítása:
  - C++ `delete` operátorral
  - C `free()` függvénnyel
    - Ugyanaz a terület nem szabadítható fel többször
    - NULL/nullptr mutató felszabadítása nem okoz gondot

- Foglalt terület átméretezése:
  C++ nincs eszköz, a problémát specifikus osztály(sablon)okkal kerülik el, pl. `std::string`, `std::vector` → következő félév anyaga
  C `realloc()`
- A C/C++ memóriakezelő megoldásai nem keverhetők egymással (pl. memóriafoglalás `new`-val majd `realloc()`)

dinamikus1.cpp

```
5   char *pc; // deklaraciok
6   int* pi;
7   double* pd;
8
9   pc = new char;   // memoriafoglalasok
10  pi = new int;
11  pd = new double;
```

dinamikus1.cpp

```
13    *pc = 'X';   // ertekadasok
14    *pi = 42;
15    *pd = 3.14;
16
17    delete pc;   // memoria felszabaditas
18    delete pi;
19    delete pd;
```

dinamikus2.cpp – A lefoglalt terület inicializálható

```
 9    pc = new char('X');   // memoriafoglalasok
10    pi = new int(42);     // inicializaciok
11    pd = new double(3.14);
```

Struktúráknak is foglalható memória dinamikusan, amit akár inicializáció is követhet

dinamikus3.cpp

```
4  struct hallgato {
5    string nev;
6    int eletkor;
7  };

10   hallgato h = { "Gizi", 19 }; // Lokalis valtozo
11   hallgato* ph1 = new hallgato; // Memoria foglalas
12   ph1->nev = "Mari"; // Ertekadasok
13   ph1->eletkor = 20;
14   // Mar lezeto strukturaval inicializalhato
15   hallgato* ph2 = new hallgato(h);
16   hallgato* ph3 = new hallgato(*ph1);
17   hallgato* ph4 = new hallgato{ "Lili", 21 }; // C++11
```

Tömböknek is lehet dinamikusan memóriát foglalni

dinamikus4.cpp

```
7   char* pc = new char[10]; // Tomb foglalasa
8   // Futasidoben kiszamolt meret sem okoz gondot
9   srand(time(NULL));
10  int* pi = new int[rand()%50];
11  double* pd = new double[100];
12
13  // Beepitett tipusokbol allo tomb
14  // nem inicializalhato :(
15
16  delete[] pc; // Felszabaditas
17  delete[] pi;
18  delete[] pd;
```

buborek6.cpp – Dinamikus tömb

```
49 int main() {
50   int hasznal; // Ennyi elemet hasznalunk eppen
51   int* szamok; // Tomb cime
52   cout << "Adjon meg nemnegativ szamokat!\n";
53   szamok = beolvas(&hasznal);
54   buborek(szamok, hasznal);
55   cout << "Rendezes utan:\n";
56   tombKiir(szamok, hasznal);
57   delete[] szamok;
58   return 0;
59 }
```

buborek6.cpp – Betelt a tömb? Átméretezés a kétszeresére

```
4  int* beolvas(int* hasznal) {
5    int uj, osszes = 2;
6    int* szamok = new int[osszes];
7    *hasznal = 0;
8    do {
9      cerr << "\t[Felhasznalva: " << *hasznal
10          << ", tombelemek szama: " << osszes << "]\n";
11     cout << *hasznal + 1 << ". szam: "; cin >> uj;
12     if(uj >= 0) {
13       if(*hasznal == osszes) {
14         cerr << "\t[Memoriafoglalas + mozgatas]\n";
15         osszes *= 2;
16         int* szamok2 = new int[osszes];
17         for(int i=0; i<*hasznal; i++) {
18           szamok2[i] = szamok[i];
19         }
```

buborek6.cpp

```
20            delete [] szamok;
21            szamok = szamok2;
22          }
23          szamok[*hasznal] = uj;
24          (*hasznal)++;
25        }
26    } while(uj >= 0);
27    return szamok;
28  }
```

Kimenet

```
Adjon meg nemnegativ szamokat!
        [Felhasznalva: 0, tombelemek szama: 2]
1. szam: 1
        [Felhasznalva: 1, tombelemek szama: 2]
2. szam: 2
        [Felhasznalva: 2, tombelemek szama: 2]
3. szam: 3
        [Memoriafoglalas + mozgatas]
        [Felhasznalva: 3, tombelemek szama: 4]
4. szam: 4
        [Felhasznalva: 4, tombelemek szama: 4]
5. szam: 5
        [Memoriafoglalas + mozgatas]
        [Felhasznalva: 5, tombelemek szama: 8]
6. szam: 6
        [Felhasznalva: 6, tombelemek szama: 8]
7. szam: -1
Rendezes utan:
1       2       3       4       5       6
```

Alakítsuk át a téglalap rajzoló programot is hasonlóan, de

- most mindig ugyanannyival növeljük a tömb méretét, ha elfogy a hely
- a tömböt lefoglaló és feltöltő függvény adja vissza az elemek számát, a tömb címét pedig írja a paraméterként kapott címre!

teglalap3.cpp

```
92  int main() {
93    teglalap* tt; int db;
94    cout << "Rajzprogram - adja meg a téglalapok adatait!\n";
95    db = bekeres(&tt);
96    rajzol(tt, db);
97    delete[] tt;
98    return 0;
99  }
```

teglalap3.cpp

```
58 int bekeres(teglalap** tt) {
59   int db=0, osszes = 2, bfx;
60   bool folytat;
61   *tt = new teglalap[osszes];
62   do {
63     cerr << "\t[Felhasznalva: " << db << ", elemszam: "
64          << osszes << "]\n";
65     folytat = bekerBFX(db+1, MINX, MAXX-1, &bfx);
66     if(folytat) {
67       if(db == osszes) {
68         cerr << "\t[Memoriafoglalas + mozgatas]\n";
69         osszes += 2;
70         teglalap* tt2 = new teglalap[osszes];
71         for(int i=0; i<db; i++) {
72           *(tt2+i) = *((*tt)+i); // tt2[i] = (*tt)[i];
73         }
```

teglalap3.cpp

```
74          delete[] *tt;
75          *tt = tt2;
76        }
77        (*tt)[db].bf.x = bfx;
78        (*tt)[db].bf.y = beker(db+1, "BF sarok Y",
79                               MINY, MAXY-1);
80        (*tt)[db].ja.x = beker(db+1, "JA sarok X",
81                               (*tt)[db].bf.x+1, MAXX);
82        (*tt)[db].ja.y = beker(db+1, "JA sarok Y",
83                               (*tt)[db].bf.y+1, MAXY);
84        cout << db+1 << ". teglalap rajzoló karaktere: ";
85        cin >> (*tt)[db].c;
86        db++;
87      }
88    } while(folytat);
89    return db;
90  }
```

Kimenet 1/2

```
Rajzprogram - adja meg a téglalapok adatait!
        [Felhasznalva: 0, elemszam: 2]
1. teglalap BF sarok X: [0, 78] (negativra vege) 0
1. teglalap BF sarok Y[0, 23] 0
1. teglalap JA sarok X[1, 79] 5
1. teglalap JA sarok Y[1, 24] 5
1. teglalap rajzoló karaktere: *
        [Felhasznalva: 1, elemszam: 2]
2. teglalap BF sarok X: [0, 78] (negativra vege) 2
2. teglalap BF sarok Y[0, 23] 2
2. teglalap JA sarok X[3, 79] 7
2. teglalap JA sarok Y[3, 24] 7
2. teglalap rajzoló karaktere: |
        [Felhasznalva: 2, elemszam: 2]
3. teglalap BF sarok X: [0, 78] (negativra vege) 4
        [Memoriafoglalas + mozgatas]
3. teglalap BF sarok Y[0, 23] 4
3. teglalap JA sarok X[5, 79] 9
3. teglalap JA sarok Y[5, 24] 9
3. teglalap rajzoló karaktere: -
        [Felhasznalva: 3, elemszam: 4]
4. teglalap BF sarok X: [0, 78] (negativra vege) -1
```

Kimenet 2/2

```
******
******
**||||||
**||||||
**||------
**||------
  ||------
  ||------
    ------
    ------
```

A C nyelv karakterláncai

- C-ben nincs `string` típus → karakteres tömbök tárolják a jelek ASCII kódjait, a lánc végét a 0 kódú karakter (`'\0'`) jelzi
- Minden eddig látott karakterlánc literál valójában konstans C karakterlánc (tömb) volt!
- `char str[] = "abc";` ← Automatikusan bekerül a lánczáró karakter a tömb végére!
- Karakterláncok kezelése: a `cstring` (`string.h`) függvényeivel, pl.
  - `size_t strlen(const char *s);` hossz lekérdezése
  - `char *strcat(char *dest, const char *src);` összefűzés
  - `char *strcpy(char *dest, const char *src);` másolás
- A programozó felelőssége, hogy az eredmény elférjen a `dest` helyen!
- C++ `string` → C-karakterlánc: `string::c_str()`

stringek.cpp

```
1  #include <iostream>
2  #include <string>
3  #include <cstring> // strlen, strcopy, strcat
4  using namespace std;
5
6  void kiir(const char* cstr) {
7    const char* ment = cstr;
8    cout << "ASCII:\t";
9    cstr--;
10   do {
11     cstr++;
12     cout << int(*cstr) << '\t';
13   } while(*cstr!='\0');
14   cout << "\nOlv.:\t";
15   for(cstr=ment; *cstr!='\0'; cstr++) {
16     cout << *cstr << '\t';
17   }
18   cout << endl;
19 }
```

stringek.cpp

```
21 int main() {
22   kiir("");
23   kiir("C");
24   kiir("C-stilus");
25   string h = "Hello";
26   kiir(h.c_str());
27   char v[] = " vilag!\n";
28   char* cs = new char[h.length() + strlen(v) + 1];
29   strcpy(cs, h.c_str());
30   strcat(cs, v);
31   cout << cs;
32   delete[] cs;
33   return 0;
34 }
```

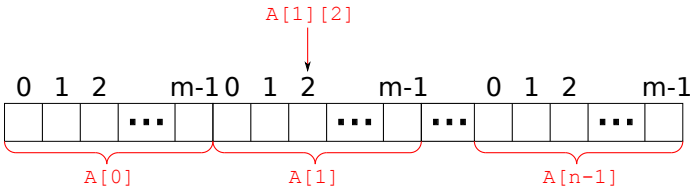
Kimenet

```
ASCII:  0
Olv.:
ASCII:  67      0
Olv.:   C
ASCII:  67      45      115     116     105     108     117     115     0
Olv.:   C       -       s       t       i       l       u       s
ASCII:  72      101     108     108     111     0
Olv.:   H       e       l       l       o
Hello vilag!
```

Mátrix: azonos típusú elemek kétdimenziós tömbje. A C++-ban csak egydimenziós tömbök léteznek, de ezeket tetszőleges mélységben egymásba lehet ágyazni!
mátrix = vektorokból álló vektor

$$A = \begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \end{bmatrix}$$

```
int A[3][4] = {
    {11, 12, 13, 14},
    {21, 22, 23, 24},
    {31, 32, 33, 34} };
```



Mátrixok összeadása: $(A+B)[i,j] = A[i,j] + B[i,j]$, ahol A és B két $n \times m$ méretű mátrix.

mtxOsszead1.cpp

```
5  #define SOROK 3
6  #define OSZLOPOK 4
7
8  int main() {
9    // deklaracio, inicializacio
10   int a[SOROK][OSZLOPOK] = {
11     { 11, 12, 13, 14 },
12     { 21, 22, 23, 24 },
13     { 31, 32, 33, 34 }
14   };
15   int b[SOROK][OSZLOPOK], c[SOROK][OSZLOPOK];
16   srand(time(NULL));
17   for(int s=0; s<SOROK; s++) { // mtx. feltoltese
18     for(int o=0; o<OSZLOPOK; o++) {
19       b[s][o] = 10 + rand()%40;
20     }
21   }
```

mtxOsszead1.cpp

```
22    for(int s=0; s<SOROK; s++) { // mtx.-ek osszeadasa
23      for(int o=0; o<OSZLOPOK; o++) {
24        c[s][o] = a[s][o] + b[s][o];
25      }
26    }
27    for(int s=0; s<SOROK; s++) { // megjelenites
28      for(int o=0; o<OSZLOPOK; o++) {
29        cout << a[s][o] << ' ';
30      }
31      cout << (s==SOROK/2?"+ ":"  ");
32      for(int o=0; o<OSZLOPOK; o++) {
33        cout << b[s][o] << ' ';
34      }
35      cout << (s==SOROK/2?"= ":"  ");
36      for(int o=0; o<OSZLOPOK; o++) {
37        cout << c[s][o] << ' ';
38      }
39      cout << endl;
40    }
41    return 0; }
```

Kimenet

```
11 12 13 14   33 49 36 12   44 61 49 26
21 22 23 24 + 20 45 24 18 = 41 67 47 42
31 32 33 34   19 10 11 42   50 42 44 76
```

Hogyan adható át egy mátrix függvénynek?

OK ✓

```
void fv(int t[SOROK][OSZLOPOK]) { //...
void fv(int t[][OSZLOPOK]) { //...
void fv(int (*t)[OSZLOPOK]) { //...
```

Hiba X – Ez mutatótömb, nem mátrix!

```
void fv(int *t[OSZLOPOK]) { //...
```

mtxOsszead2.cpp

```
44 int main() {
45   int a[SOROK][OSZLOPOK], b[SOROK][OSZLOPOK],
46       c[SOROK][OSZLOPOK];
47   srand(time(NULL));
48   general(a);
49   general(b);
50   osszead(a, b, c);
51   megjelenit(a, b, c);
52   return 0;
53 }
```

mtxOsszead2.cpp

```
5  #define SOROK 3
6  #define OSZLOPOK 4
7
8  void general(int t[][OSZLOPOK]) {
9    for(int s=0; s<SOROK; s++) {
10     for(int o=0; o<OSZLOPOK; o++) {
11       t[s][o] = 10 + rand()%40;
12     }
13   }
14 }
15
16 void osszead(const int (*a)[OSZLOPOK], const int (*b)[OSZLOPOK],
17              int (*c)[OSZLOPOK]) {
18   for(int s=0; s<SOROK; s++) {
19     for(int o=0; o<OSZLOPOK; o++) {
20       c[s][o] = *(a[s] + o) + *(*(b+s) + o);
21     }
22   }
23 }
```

mtxOsszead2.cpp

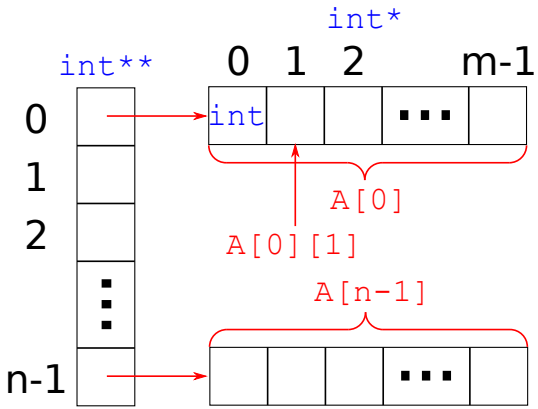
```
25 void megjelenit(const int a[][OSZLOPOK],
26                 const int b[][OSZLOPOK],
27                 const int c[][OSZLOPOK]) {
28   for(int s=0; s<SOROK; s++) {
29     for(int o=0; o<OSZLOPOK; o++) {
30       cout << a[s][o] << ' ';
31     }
32     cout << (s==SOROK/2?"+ ":"  ");
33     for(int o=0; o<OSZLOPOK; o++) {
34       cout << b[s][o] << ' ';
35     }
36     cout << (s==SOROK/2?"= ":"  ");
37     for(int o=0; o<OSZLOPOK; o++) {
38       cout << c[s][o] << ' ';
39     }
40     cout << endl;
41   }
42 }
```

Probléma:

rugalmatlan függvények, mert az oszlopok száma rögzített

Megoldás:

- hozzunk létre dinamikusan vektorokat (pl. `int*`-gal címezhetők), majd
- ezek címeit tároljuk egy újabb, dinamikus vektorban (`int**`, mutatótömb)!



mtxOsszead3.cpp

```
56 int main() {
57   srand(time(NULL));
58   int sorok = 1 + rand()%4;
59   int oszlopok = 1 + rand()%4;
60   int** a = lefoglal(sorok, oszlopok);
61   int** b = lefoglal(sorok, oszlopok);
62   int** c = lefoglal(sorok, oszlopok);
63   general(a, sorok, oszlopok);
64   general(b, sorok, oszlopok);
65   osszead(a, b, c, sorok, oszlopok);
66   megjelenit(a, b, c, sorok, oszlopok);
67   felszabadit(a, sorok);
68   felszabadit(b, sorok);
69   felszabadit(c, sorok);
70   return 0; }
```

mtxOsszead3.cpp

```
6  int** lefoglal(int sorok, int oszlopok) {
7    int** t = new int*[sorok];
8    for(int s=0; s<sorok; s++) {
9      t[s] = new int[oszlopok];
10   }
11   return t;
12 }
13
14 void general(int** t, int sorok, int oszlopok) {
15   for(int s=0; s<sorok; s++) {
16     for(int o=0; o<oszlopok; o++) {
17       t[s][o] = 10 + rand()%40;
18     }
19   }
20 }
```

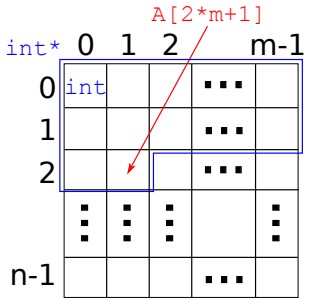
mtxOsszead3.cpp

```
22 void osszead(int** a, int** b, int** c,
23              int sorok, int oszlopok) {
24   for(int s=0; s<sorok; s++) {
25     for(int o=0; o<oszlopok; o++) {
26       c[s][o] = *(a[s] + o) + *(*(b+s) + o);
27     }
28   }
29 }

49 void felszabadit(int** t, int sorok) {
50   for(int s=0; s<sorok; s++) {
51     delete[] t[s];
52   }
53   delete[] t;
54 }
```

Alternatív megoldás:

- utánozzuk a „statikus” tömbök memóriabeli szerkezetét, azaz
- valójában vektornak foglalunk helyet, és erre képezzük le a mátrix elemeit



mtxOsszead4.cpp

```
44 int main() {
45   srand(time(NULL));
46   int sorok = 1 + rand()%4;
47   int oszlopok = 1 + rand()%4;
48   int* a = lefoglal(sorok, oszlopok);
49   int* b = lefoglal(sorok, oszlopok);
50   int* c = lefoglal(sorok, oszlopok);
51   general(a, sorok, oszlopok);
52   general(b, sorok, oszlopok);
53   osszead(a, b, c, sorok, oszlopok);
54   megjelenit(a, b, c, sorok, oszlopok);
55   delete[] a;
56   delete[] b;
57   delete[] c;
58   return 0;
59 }
```

mtxOsszead4.cpp

```
6   int* lefoglal(int sorok, int oszlopok) {
7     return new int[sorok*oszlopok];
8   }
9
10  void general(int* t, int sorok, int oszlopok) {
11    for(int s=0; s<sorok; s++) {
12      for(int o=0; o<oszlopok; o++) {
13        t[s*oszlopok + o] = 10 + rand()%40;
14      }
15    }
16  }
17
18  void osszead(int* a, int* b, int* c, int sorok, int oszlopok) {
19    for(int s=0; s<sorok; s++) {
20      for(int o=0; o<oszlopok; o++) {
21        c[s*oszlopok+o] = a[s*oszlopok+o] + b[s*oszlopok+o];
22      }
23    }
24  }
```