

Programozás

(GKxB_INTM114)

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

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

2024. március 25.

Feladat:

Készítsünk olyan függvényt, ami a két paraméterének értékét felcseréli!

Problémák:

- Érték szerinti paraméter-átadás
- Csak egy visszatérési értéke lehet a függvénynek

csere1.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 void nyomtat(int a, int b) {
5     cout << "a = " << a << ", b = " << b << '\n';
6 }
```

```
eredeti ertekek:      a = 1, b = 2
csere1 után:          a = 1, b = 2
```

csere1.cpp – Második próbálkozás, csere2

```
14 struct ketszam { int a, b; };
15
16 ketszam csere2(int a, int b) {
17     ketszam cs = {b, a};
18     return cs;
19 }
```

csere1.cpp – A main részlete

```
31 ketszam ksz = csere2(a, b); a = ksz.a; b = ksz.b;
32 cout << "csere2 utan:\t\t"; nyomtat(a, b);
```

Kimenet részlete

```
csere2 utan:          a = 2, b = 1
```

Mi az a *mutató* (pointer), és mire használható?

- Memóriacím tárolására használható típus
- Többféle típusa létezik, hogy kifejezze az ott tárolt adat típusát
- Technikai megvalósítása hasonlít az egész számokéhoz
- Mutatódefiníció: *alaptípus** azonosító;

Néhány lehetséges mutatódefiníció

```
struct koordinata {  
    int x, y;  
};  
/* ... */  
char* pc;          // karaktert címző mutató  
int* pi;            // egészlet címző mutató  
double* pd;         // valós számot c. m.  
koordinata* pk;     // struktúrát c. m.  
void* pv;           // ismeretlen típusú adatot címző mutató
```

```

5  int i;
6  cout << "Az i valtozo memoriacime: " << &i << '\n';
7  struct koordinata { int x, y; } k;
8  cout << "A k struktura memoriacime: " << &k
9      << "\nk.x helye: " << &k.x << ", k.y helye: " << &k.y << '\n';
10 double dt[2];
11 cout << "A dt tomb memoriacime: " << &dt
12      << "\ndt[0] helye: " << &dt[0] << ", dt[1] helye: " << &dt[1] << '\n';

```

```
Az i valtozo memoriacime: 0x7ffeb6837cdc
A k struktura memoriacime: 0x7ffeb6837ce0
k.x helye: 0x7ffeb6837ce0, k.y helye: 0x7ffeb6837ce4
A dt tomb memoriacime: 0x7ffeb6837cf0
dt[0] helye: 0x7ffeb6837cf0, dt[1] helye: 0x7ffeb6837cf8
```

indirekcio.cpp – Adott címen lévő érték elérése: * (indirekció, dereference) operátorral

```
1 #include <iostream>
2 using namespace std;
3
4 int main(void) {
5     int i = 3;
6     int* pi;
7     pi = &i;
8     *pi += 2; // i += 2;
9     cout << "i = " << i << endl;
10    return 0;
11 }
```

Kimenet

```
i = 5
```

```
csere2 utan:      a = 2, b = 1
csere3 utan:      a = 1, b = 2
```


teglalap2.cpp Téglalapok rajzolása, bekerBFX: Akarja folytatni a bevitelt? Ha igen, mi a koordináta?

```

38 bool bekerBFX(int db, int min, int max, int* k) {
39     bool folytat;
40     do {
41         cout << db << ". téglalap BF sarok X: [" << MINX
42             << ", " << MAXX-1 << "] (negatívra vege) ";
43         cin >> *k;
44         folytat = *k>=0;
45     } while(folytat && (*k<MINX or *k>MAXX-1));
46     return folytat;
47 }
48
49 int beker(int db, string s, int min, int max) {
50     int k;
51     do {
52         cout << db << ". téglalap " << s << '[' << min << ", " << max << "] ";
53         cin >> k;
54     } while(k<min or k>max);
55     return k;
56 }

```

teglalap2.cpp

```

58 int main() {
59     teglalap tt[MAXALAK]; int db; bool folytat = true;
60     cout << "Rajzprogram – adja meg a téglalapok adatait!\n";
61     for(db=0; db<MAXALAK and folytat; db++) {
62         folytat = bekerBFX(db+1, MINX, MAXX-1, &tt[db].bf.x);
63         if(folytat) {
64             tt[db].bf.y = beker(db+1, "BF sarok Y", MINY, MAXY-1);
65             tt[db].ja.x = beker(db+1, "JA sarok X",
66                                 tt[db].bf.x+1, MAXX);
67             tt[db].ja.y = beker(db+1, "JA sarok Y",
68                                 tt[db].bf.y+1, MAXY);
69             cout << db+1 << ". teglalap rajzoló karaktere: ";
70             cin >> tt[db].c;
71         }
72     }
73     rajzol(tt, db);
74     return 0;
75 }

```

Néhány további tudnivaló mutatókkal kapcsolatban

- **Vigyázz!** `i` egész, `pi1` és `pi2` viszont egészet címző mutatók!
- A `*` körül tetszőleges számú szóköz elhelyezhető
- Mutató is kaphat inicializálással kezdőértéket

mutatok1.cpp

```
5  int i=3, *pi1, *pi2;  
6  pi1 = pi2 = &i; // OK  
7  double d=1.5;  
8  double* pd = &d; // inicializalas
```

- Ha valaminek nincs memóriacíme, akkor az & operátor sem tudja előállítani

mutatok1.cpp

```
9 // pd = &12.34;
10 // error: lvalue required as unary '&' operand
11 // literalnak nincs memóriacíme, értelmetlen
```

- Értékadás általában csak azonos típusú mutatók között lehetséges

mutatok1.cpp

```
12 // pd = pi1; error: cannot convert 'int*'
13 //      to 'double*' in assignment
```

- Kivétel: `void*` bármilyen más mutató értékét felveheti (\approx típusinformáció eldobása)

mutatok1.cpp

```
14 void *pv;  
15 pv = pi1; // OK
```

- Fordítva már nem megy: nem lehetünk benne biztosak, hogy azon a címen milyen típusú adat van
- Explicit típuskonverzióval persze rávehető a fordító a műveletre, de vajon van értelme?

mutatok1.cpp

```
16 // pi1 = pv;  
17 // error: invalid conversion from 'void*' to 'int*'  
18 pi1 = (int*)pv; // Programozo felelossegere
```

- Nem tudni, hogyan kell megjeleníteni az ismeretlen típusú adatot

mutatok1.cpp

```
19 // cout << *pv;  
20 // error: 'void*' is not a pointer-to-object type
```

- A NULL / nullptr speciális memóriacím: semmilyen adatot nem tárolnak ott, és
- hiba, vagy valami hiányának jelzésére használják,
- bármilyen típusú mutatóhoz hozzárendelhető érték
- NULL: a 0 értékhez készített makró (C örökség, elavult), nullptr: mindenképpen mutató

mutatok1.cpp

```
22 pv = NULL; pv = nullptr;
```

Probléma:

struktúrák általában nagyok, függvényhívásnál a paraméter átadás a másolás miatt időigényes

Megoldás:

- adjuk át a struktúra címét!
- **Veszély!** Ha a *hívott* fv. módosítja a paramétert, annak a *hívó* függvényben is lesz hatása!
- Ha a *hívott* függvénynek nem célja módosítani a paramétert: **const** csak olvashatóvá teszi azt (bármilyen más változónál is használható *típusmódosító*)
- Indirekció + tagelérés: **->** operátorral, pl. $(*d).nap \equiv d->nap$

naptar2.cpp

```
22 bool ellenoriz(const datum* d) {      // datum tartalmi
23     if(d->ho<1 or d->ho>12) return false; // ellenorzese
24     int n = napok(d->ev, d->ho);
25     if(d->nap<1 or d->nap>n) return false;
26     return true;
27 }
28
29 int evNapja(const datum* d) { // ev napjanak
30     int n = d->nap;           // meghatározása
31     for(int h=1; h<d->ho; h++) { // ev, ho, napból
32         n += napok(d->ev, h);
33     }
34     return n;
35 }
```


naptar2.cpp

```
75 int main(void) {
76     datum d = {2018, 3, 15};
77     cout << "A megadott datum "
78           << (ellenoriz(&d)? "helyes": "hibas")
79           << ".\n" << d.ev << '.' << d.ho << '.' << d.nap
80           << " az ev " << evNapja(&d) << ". napja, "
81           << hetNapja(&d) << ".\n";
82     datum kar = {2018, 12, 24};
83     cout << "Hany nap van karacsonyig? " << kulonbseg(&d, &kar);
84     int evNapja = 300;
85     d = hoEsNap(d.ev, evNapja);
86     cout << '\n' << d.ev << ' ' << evNapja << ". napja: "
87           << d.ho << '.' << d.nap << endl;
88     return 0;
89 }
```

buborek2.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  void buborek(int t[], int n) {
5      for(int i=n-1; i>=1; i--) {
6          for(int k=0; k<i; k++) {
7              if(t[k] > t[k+1]) {
8                  int csere = t[k];
9                  t[k] = t[k+1];
10                 t[k+1] = csere;
11             }
12         }
13     }
14 }
```

buborek2.cpp

```

16 int main() {
17     int szamok[] = {12, 3, 54, -4, 56, 4, 7, 3};
18     int n = sizeof(szamok)/sizeof(szamok[0]);
19     buborek(szamok, n);
20     cout << "Rendezes utan:\n";
21     for(int i=0; i<n; i++) {
22         cout << szamok[i] << '\t';
23     }
24     cout << endl;
25     return 0;
26 }
    
```

Kimenet

Rendezes utan:

| | | | | | | | |
|----|---|---|---|---|----|----|----|
| -4 | 3 | 3 | 4 | 7 | 12 | 54 | 56 |
|----|---|---|---|---|----|----|----|

Újdonságok:

- Tömb elemszámát nem *kell* megadni a formális paraméterlistán (de a fv.-nek valahonnan tudnia kell, hány tömbelemet kell rendezni)
- A *hívott* függvény **módosította** a paraméter tömböt!

Magyarázat:

- A tömbök általában nagyok → **mindig** a címet adják át!
- A tömbök azonosítója egy *konstans mutató* (a mutatót nem, de a mutatott helyen lévő értéket lehet módosítani), pl.
`int t[] \equiv int* const t`
- A tömb tartalma csak olvashatóvá tehető:
`const int t[] \equiv const int* const t`
- Hátról előre olvasva: *t* egy *const*-ans mutató (*), ami olyan *int*-et címez ami *const*-ans.

buborek3.cpp

```
16 void tombKiir(const int* const t, int n) {
17     for(int i=0; i<n; i++) {
18         cout << t[i] << '\t';
19     }
20     cout << endl;
21 }
22
23 int main() {
24     int szamok[] = {12, 3, 54, -4, 56, 4, 7, 3};
25     int n = sizeof(szamok)/sizeof(szamok[0]);
26     buborek(szamok, n);
27     cout << "Rendezes utan:\n";
28     tombKiir(szamok, n);
29     return 0;
30 }
```

Mutatóaritmetika: hasonlóan végezhető művelet mutatókkal, mint egészekkel:

- Mutató növelhető, csökkenthető \rightarrow a tényleges cím a mutatott adat méretének többszörösével változik
- Mutatók összehasonlíthatóak (relációk)
- $\text{tömbElemCíme} = \text{tömbKezdőcíme} + \text{index} * \text{sizeof}(\text{tömbElemTípusa})$
- $\text{tomb}[\text{index}] \equiv *(\text{tomb} + \text{index})$
- A `void*` mutató kivételes: a mutatott elem mérete ismeretlen
- Azonos tömb elemeit címző mutatók különbsége képezhető

mutatok2.cpp

```
5  int t[] = { 100, 200, 300 };
6  int* pi = t;
7  cout << "Első elem értéke (cime):\t" << pi[0] << " (" << t
8      << ")\nMasodik elem értéke (cime):\t";
9  pi++;
10 cout << *pi << " (" << pi
11      << ")\nHarmadik elem értéke (cime):\t" << *(t+2)
12      << " (" << t+2 << ")\n";
```

Kimenet

```
Első elem értéke (cime):      100 (0x7ffc05f63510)
Masodik elem értéke (cime):   200 (0x7ffc05f63514)
Harmadik elem értéke (cime): 300 (0x7ffc05f63518)
```

buborek4.cpp

```
16 void tombKiir(const int* t, int n) {  
17     for(const int* vege=t+n; t<vege; t++) {  
18         cout << *t << '\\t';  
19     }  
20     cout << endl;  
21 }
```


A C nyelvből örökölt megoldás:

- **Álvéletlen számok** előállítása (PseudoRandom Number Generator, PRNG)
- Szükséges fejléc: `cstdlib` vagy `stdlib.h`
- Kezdőérték: `void srand(unsigned int seed);`, ahol `seed` a kezdőérték
- Véletlen számok: $0 \leq \text{int rand}(\text{void}); \leq \text{RAND_MAX}$

Példák:

- $x = (\text{double})\text{rand}() / \text{RAND_MAX}$ ahol $\{x | x \in \mathbb{R}, 0 \leq x \leq 1\}$
- $x = \text{MIN} + \text{rand}() \% (\text{MAX} - \text{MIN} + 1)$ ahol $\{x | x \in \mathbb{Z}, \text{MIN} \leq x \leq \text{MAX}\}$

Probléma: azonos seed → azonos számsorozatok

Megoldás:

- seed minden programindításnál más legyen → pontos idő
- Szükséges fejléc: `ctime` vagy `time.h`
- `time_t time(time_t *t);`
- V.t. érték: `time_t (long)` típusban az 1970-01-01 00:00:00 +0000 (UTC) (Unix-idő, epoch) óta eltelt **másodpercek (!)** száma, amit `t` címen is eltárol, ha az nem `nullptr`

A C++ nyelv sokkal kifinomultabb képességekkel rendelkezik.

További tudnivalók a véletlenszám generálással kapcsolatban.

tipp.cpp

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <ctime>
4 #define MIN 1
5 #define MAX 100
6 using namespace std;
7
8 int main() {
9     srand(time(NULL));
10    int tipp, szam = MIN + rand()%(MAX-MIN+1);
11    cout << "Talald ki a " << MIN << " es " << MAX << " kozotti szamot!\n";
12    do {
13        cout << "Tipp: "; cin >> tipp;
14        if(tipp < szam) cout << "Nagyobbra gondoltam.\n";
15        else if(tipp > szam) cout << "Kisebbre gondoltam.\n";
16    } while(tipp != szam);
17    cout << "Eltalaltad!\n";
18    return 0; }
```