Programozás
(GKxB_INTM114)

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

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

2024. április 16.

SZÉCHENYI EGYETEM
UNIVERSITY OF GYŐR

INFORMATIKA TANSZÉK
DEPARTMENT OF INFORMATICS

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 }
```

csere1.cpp – Első próbálkozás, csere1

```
 8 void csere1(int a, int b) {
 9   int cs = a;
10   a = b;
11   b = cs;
12 }
```

csere1.cpp – A main részlete

```
27 int main(void) {
28   int a = 1, b = 2;
29   cout << "eredeti ertekek:\t"; nyomtat(a, b);
30   csere1(a, b); cout << "csere1 utan:\t\t"; nyomtat(a, b);
```

Kimenet részlete

```
eredeti ertekek:        a = 1, b = 2
csere1 utan:            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;        // egészet címző mutató
double* pd;     // valós számot c. m.
koordinata* pk; // struktúrát c. m.
void* pv;       // ismeretlen típusú adatot címző mutató
```

cim.cpp – Memóriacím képzése az & (címképző, address-of) operátorral lehetséges

```
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';
```

Kimenet

```
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
```

csere1.cpp – Harmadik próbálkozás, csere3

```
21 void csere3(int* a, int* b) {
22   int cs = *a;
23   *a = *b;
24   *b = cs;
25 }
```

csere1.cpp – A main részlete

```
33   csere3(&a, &b); cout << "csere3 utan:\t\t"; nyomtat(a, b);
34   return 0;
35 }
```

Kimenet részlete

```
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 << ". teglalap BF sarok X: [" << MINX
42          << ", " << MAXX-1 << "] (negativra 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 << ". teglalap " << 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 memoriacime, ertelmetlen
```

- 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 (≈ 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` ≡ `d->nap`

naptar2.cpp

```
23 bool ellenoriz(const datum* d) {     // datum tartalmi ellenorzese
24   if(d->ho<1 or d->ho>12) return false;
25   int n = napok(d->ev, d->ho);
26   if(d->nap<1 or d->nap>n) return false;
27   return true;
28 }
29
30 int evNapja(const datum* d) {  // ev napjanak meghatarozasa
31   int n = d->nap;               // ev, ho, napbol
32   for(int h=1; h<d->ho; h++) {
33     n += napok(d->ev, h);
34   }
35   return n;
36 }
```

naptar2.cpp

```
77  int main(void) {
78    datum d = {2024, 3, 20};
79    cout << "A megadott datum " << (ellenoriz(&d)?"helyes":"hibas")
80         << ".\n" << d.ev << '.' << setw(2) << setfill('0') << d.ho
81         << '.' << setw(2) << setfill('0') << d.nap << " az ev "
82         << evNapja(&d) << ". napja, " << hetNapja(&d) << ".\n";
83    datum kar = {2024, 12, 24};
84    cout << "Hany nap van karacsonyig? " << kulonbseg(&d, &kar);
85    int evNapja = 300;
86    d = hoEsNap(d.ev, evNapja);
87    cout << '\n' << d.ev << ' ' << evNapja << ". napja: "
88         << setw(2) << setfill('0') << d.ho << '.' << d.nap << endl;
89    return 0;
90  }
```

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[]` ≡ `int* const t`
- A tömb tartalma csak olvashatóvá tehető:
  `const int t[]` ≡ `const int* const t`
- Hátulról 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 → a tényleges cím a mutatott adat méretének többszörösével változik
- Mutatók összehasonlíthatóak (relációk)
- tömbelemCíme = tömbKezdőcíme + index*sizeof(tömbelemTípusa)
- `tomb[index]` ≡ `*(tomb + 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 << "Elso elem erteke (cime):\t" << pi[0] << " (" << t
8        << ")\nMasodik elem erteke (cime):\t";
9   pi++;
10  cout << *pi << " (" << pi
11       << ")\nHarmadik elem erteke (cime):\t" << *(t+2)
12       << " (" << t+2 << ")\n";
```

Kimenet

```
Elso elem erteke (cime):        100 (0x7ffc05f63510)
Masodik elem erteke (cime):     200 (0x7ffc05f63514)
Harmadik elem erteke (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 fejfájl: `cstdlib` vagy `stdlib.h`
- Kezdőérték: `void srand(unsigned int seed);`, ahol `seed` a kezdőérték
- Véletlen számok: $0 \leq$ `int rand(void);` $\leq$ `RAND_MAX`

Példák:

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

Probléma: azonos seed $\rightarrow$ azonos számsorozatok
Megoldás:

- seed minden programindításnál más legyen $\rightarrow$ pontos idő
- Szükséges fejfájl: `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; }
```