

7. Gyakorlat

Rövid elméleti összefoglaló

- Alapértelmezés szerinti (default) argumentumok

A C++ függvények prototípusában a paraméterekhez ún. alapértelmezés szerinti értékeket rendelhetünk. Az alapértelmezés szerinti értékekkel ellátott paraméterek *jobbról-balra* (azaz az utolsó paramétértől visszafelé) haladva folyamatosan helyezkednek el. Ha elhagyjuk a *default* paramétereket, akkor a híváskor *balról-jobbra* haladva kell megadnunk az argumentumokat, de nem hagyhatunk ki paramétért.

<pre>//MINTA7_01 #include <iostream> using namespace std; int osszeg(int x, int y = 2, int z = 3) { return x + y + z; } int main() { int s1, s2, s3; // alapértelmezés szerinti (default) argumentumok s1 = osszeg(1); cout << "osszeg(1) hivas eredménye (2,3 default): " << s1 << endl; s2 = osszeg(5,7); cout << "osszeg(5,7) hivas eredménye (3 default): " << s2 << endl; s3 = osszeg(4,3,5); cout << "osszeg(4,3,5) hivas eredménye: " << s3 << endl; cin.get(); return 0; }</pre>	<p>Argumentumok:</p> <table><tr><th>x</th><th>y</th><th>z</th></tr><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>5</td><td>7</td><td>3</td></tr><tr><td>4</td><td>3</td><td>5</td></tr></table>	x	y	z	1	2	3	5	7	3	4	3	5
x	y	z											
1	2	3											
5	7	3											
4	3	5											
<p>A program futásának eredménye:</p> <pre>osszeg(1) hivas eredménye (2,3 default): 6 osszeg(5,7) hivas eredménye (3 default): 15 osszeg(4,3,5) hivas eredménye: 12</pre>													

- Változó hosszúságú argumentumlista

A ... (három pont) megadásával használhatunk változó hosszúságú argumentumlistát a saját függvényeinkben, melyek feldolgozása a *cstdarg* fejlécállományban definiált makrókkal történik.

<pre>//MINTA7_02 #include <iostream> #include <cstdarg> #include <cmath> using namespace std; double MertaniKozep(int db,...); int main() { double mkozep; mkozep = MertaniKozep(2,1.0,2.0); cout << "Mertani kozep1 : " << mkozep << endl << endl; mkozep = MertaniKozep(4, 1.2, 2.5, 4.6, 8.2); cout << "Mertani kozep2 : " << mkozep << endl; cin.get(); return 0; } double MertaniKozep(int db,...) { va_list adat; double mk = 1; va_start(adat, db); // az első paraméter átlépése for (int i = 0; i<db; i++) { mk *= va_arg(adat, double); // double paraméter } mk = pow(mk,1./db); va_end(adat); return mk; }</pre>	<p>Megjegyzések a változó hosszúságú argumentumlista használatához:</p> <p>A függvényben <i>va_list</i> típusú változót kell deklarálni a <i>va_start()</i>, <i>va_arg()</i> és a <i>va_end()</i> makrók hívásához, mivel a makrók a <i>va_list</i> típusú mutatót használják az argumentumok eléréséhez.</p> <p>Kezdőértéket ad az argumentumok eléréséhez használt mutatónak:</p> <pre>void va_start(va_list p, elso_par);</pre> <p>Az argumentumlista következő elemét adja vissza:</p> <pre>type va_arg(va_list p,type);</pre> <p>Az argumentumok felhasználása után kötelezően kell hívni:</p> <pre>void va_end(va_list p);</pre> <p>A program futásának eredménye:</p> <pre>Mertani kozep1 : 1.41421 Mertani kozep2 : 3.26154</pre>
---	---

• Rekurzív függvények alkalmazása

A rekurzív függvények önmagukat hívják. A rekurzív megoldás általában rövidebb és áttekinthetőbb, mint az iteratív megoldás, azonban megnövekszik a feladat futási ideje és a memóriaigénye.

```
//MINTA7_03
#include <iostream>
using namespace std;
double kamatkiir(int evk, int ev,
                 double osszeg, double kamatlab);
int main()
{ double osszeg, kamatlab, ered;
  int ev;
  cout << "Kamatszamitas " << endl;
  cout << "Betet : "; cin >> osszeg;
  cout << "Kamatlab: "; cin >> kamatlab;
  do
  {
    cout << "Ev: "; cin >> ev;
  }while (0>= ev || ev > 10);
  ered = kamatkiir(1, ev, osszeg, kamatlab);
  cin.get(); cin.get();
  return 0;
}
double kamatkiir(int evk, int ev, double osszeg,
double kamatlab)
{
  if (ev == 0) return osszeg;
  else
  {
    osszeg = osszeg * (1+kamatlab/100);
    cout << "Az " << evk << ". ev vegen az osszeg: "
          << osszeg << endl;
    return kamatkiir(evk+1, ev-1, osszeg, kamatlab);
  }
}
```

A *kamatkiir()* rekurzív függvény adott *evk* (1) és *ev* között évenként kiírja a kamatos kamatot.

A függvény paraméterei még a betét összege (*osszeg*) és a kamatláb (*kamatlab*) paraméterben.

A program futásának eredménye:

```
Kamatos kamat szamitasa
Betet : 100
Kamatlab: 10
Ev: 10
Az 1. ev vegen az osszeg: 110
Az 2. ev vegen az osszeg: 121
Az 3. ev vegen az osszeg: 133.1
Az 4. ev vegen az osszeg: 146.41
Az 5. ev vegen az osszeg: 161.051
Az 6. ev vegen az osszeg: 177.156
Az 7. ev vegen az osszeg: 194.872
Az 8. ev vegen az osszeg: 214.359
Az 9. ev vegen az osszeg: 235.795
Az 10. ev vegen az osszeg: 259.374
```

• Függvények átdefiniálása (overloading)

A C++ nyelvben több függvényt definiálhatunk ugyanazzal a névvel, ha a definiált függvények paraméterlistája eltér egymástól. Az azonos nevű függvényváltozók közül a fordító választja ki az éppen szükségeset az argumentumok száma és típusa alapján.

```
//MINTA7_04
#include <iostream>
using namespace std;
double Osszeg(double a, double b);
int Osszeg(int a, int b);
double Osszeg(double a, int b);
double Osszeg(int a, double b);
int main()
{
  cout << "2.5 + 4.2 = " << Osszeg(2.5,4.2) << endl;
  cout << "2 + 4 = " << Osszeg(2,4) << endl;
  cout << "2.3 + 5 = " << Osszeg(2.3,5) << endl;
  cout << "3 + 5.6 = " << Osszeg(3,5.6) << endl;
  cin.get();
  return 0;
}
double Osszeg(double a, double b)
{ return a+b; }
int Osszeg(int a, int b)
{ return a+b; }
double Osszeg(double a, int b)
{ return a+b; }
double Osszeg(int a, double b)
{ return a+b; }
```

Számítsuk ki két szám összegét, ha az adatok a következő típusúak lehetnek! (Az eredmény típusa is változik.)

```
double + double → double
int + int → int
double + int → double
int + double → double
```

A feladat megvalósításához 4 függvény szükséges.

A program futásának eredménye:

```
2.5 + 4.2 = 6.7
2 + 4 = 6
2.3 + 5 = 7.3
3 + 5.6 = 8.6
```

- **A main() függvény paraméterei és visszatérési értéke**

A **main()** függvényt általában 0,1 vagy 2 paraméterrel használjuk!

```
int main ( int argc , char* argv[ ])
```

argc - az *argv* tömbben található sztringek száma.

argv - karaktermutatókat (**char***) tartalmazó tömb, az *argv[0]* a program nevére mutat.

```
//MINTA7_05
#include <iostream>
#include <cstdlib>
using namespace std;
int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        cerr << "Ket parameter szukseges: ";
        cerr << "az input és az output fajl neve" << endl;
        return EXIT_FAILURE;
    }
    cout << "A program neve: " << argv[0] << endl;
    cout << "Input fajl neve: " << argv[1] << endl;
    cout << "Output fajl neve: " << argv[2] << endl;
    cin.get();
    return EXIT_SUCCESS;
}
```

A **main()** függvény paraméterként két állomány nevét várja.
A lefordított **MAINPAR.EXE** programot a következőképpen indítjuk:

```
mainpar be.dat ki.dat
```

A program futásának eredménye:

```
A program neve: mainpar
Input fajl neve: be.dat
Output fajl neve: ki.dat
```

Megjegyzés: Az **EXIT_FAILURE** és az **EXIT_SUCCESS** szabványos konstansokat a **cstdlib** fejlécállomány tartalmazza.

- **Sztringargumentum**

A karaktersorozat-argumentumok egydimenziós karaktertömbökként (**char s[]**) adódnak át a függvénynek. (A **char*** mutatóval még tömörebben lehet a feladatokat megoldani.)

```
//MINTA7_06
#include <iostream>
using namespace std;
int KarSzamlal(const char s[],
               const char karakter);
int main()
{
    int kar_db;
    char szoveg[81], kar;
    cout << "szoveg (max.80 karakter): ";
    cin.getline(szoveg,81);
    cout << "Keresett karakter: "; cin >> kar;
    kar_db = KarSzamlal(szoveg, kar);
    cout << "" << kar << ""
         << " szama a szovegben: " << kar_db;
    cin.get(); cin.get();
    return 0;
}
int KarSzamlal(const char s[],
               const char karakter)
{
    int db = 0;
    for (int i = 0; s[i]; i++)
    {
        if (s[i] == karakter) db++;
    }
    return db;
}
```

A **const** a paraméterlistán azt jelenti, hogy a paraméterek értéke nem változik meg a függvényben.

A **getline()** hívás szóközt tartalmazó karaktersorozatot is be tud olvasni, azonban meg kell adni a karaktertömb méretét is:

```
cin.getline(szoveg,80);
```

A **KarSzamlal()** függvénynek két paramétere van: egy string (karaktersorozat) és egy karakter.

A függvény a *karakter* paraméterben megadott karaktert keresi az *s* sztringben és megszámolja, hogy hányszor fordult elő.

A ciklusban a feltétel:

```
s[i] != '\0'
```

mivel a ciklus a sztring végét jelző 0-s bájt nál áll le.

A program futásának eredménye:

```
szoveg (max.80 karakter): Ma szep az ido.
Keresett karakter: a
'a' szama a szovegben: 2
```

Feladatok

1. Tervezzük meg a *Szamol()* függvényt, melynek 3 valós és két alapértékkel megadott egész paramétere van. (GYAK7_1)

```
double Szamol(double x, double y, double z, int jel=0, int h = 3);
```

Ha a *jel* értéke 0, akkor a három valós szám összegét osszuk el a *h* értékével!

Ha a *jel* értéke 1, akkor a három valós szám szorzatából vonjunk *h* értékének megfelelő gyököt!

Ha a *jel* értéke 2, akkor a három valós szám összegét emeljük a *h* értékének megfelelő hatványra!

Aktiváljuk a *Szamol()* függvényt, változtatva az alapértelmezett paramétereket is!

A program futásának lehetséges eredménye:

```
1. adat: 1
2. adat: 2
3. adat: 3
Eredmeny: 2
Eredmeny: 1.81712
Eredmeny: 216
```

2. Tervezzünk a *MertaniKozep()* függvényt aktiváló *main()* függvényt, amely változó hosszúságú argumentumlistát használ. Az első paraméter a mértani közép számításához szükséges első adat, a változó paraméterlista végét a 0-ás adat zárja. (GYAK7_2)

```
double MertaniKozep(int elso,...)
{
    va_list p;
    int db = 1;
    int mk = elso, adat;
    double ered;
    va_start(p, elso); // az első paraméter átlépése
    while (adat = va_arg(p, int))
    {
        mk *= adat; // int paraméter
        db++;
    }
    ered = pow(mk, 1./db);
    va_end(p);
    return ered;
}
```

A függvényt legalább kétszer aktiváljuk különböző darabszámú adattal!

```
double mkozep;
mkozep = MertaniKozep(1,2,0);
cout << "Mertani kozep1 : " << mkozep << endl << endl;
mkozep = MertaniKozep( 1, 2, 4, 8,0);
cout << "Mertani kozep2 : " << mkozep << endl;
...
```

A program futásának eredménye:

```
Mertani kozep1 : 1.41421
Mertani kozep2 : 2.82843
```

3. Kamatos kamat számítására használjuk fel a rekurzív módon működő kamat függvényt! Írjunk hozzá *main()* függvényt, az eredményeket jelenítsük meg! (GYAK7_3)

```
double kamat(int ev, double osszeg, double kamatlab)
{
    if (ev == 0) return osszeg;
    else
    {
        osszeg = osszeg * (1+kamatlab/100);
        return kamat(ev-1, osszeg, kamatlab);
    }
}
```

A program futásának eredménye:

```
Kamatos kamat szamitasa
Betet : 100
Kamatlab: 10
Ev: 10
Kamatos kammattal megnovelt osszeg: 259.374
```

4. A téglalap kerületének és területének kiszámítására írjunk függvényeket, amelyek valós és egész adatokkal is működnek! (GYAK7_4)

Használjuk fel a függvények alábbi prototípusait!

```
void TeglalapKerulet(double a, double b, double& ker);
double TeglalapTerulet(double a, double b);
void TeglalapKerulet(int a, int b, int& ker);
int TeglalapTerulet(int a, int b);
```

A program futásának eredménye:

```
Valos adatok megadasa
Teglalap a oldala: 1.5
Teglalap b oldala: 2.5
Teglalap kerulete: 8
Teglalap terulete: 3.75
```

```
Egesz adatok megadasa
Teglalap a oldala: 1
Teglalap b oldala: 2
Teglalap kerulete: 6
Teglalap terulete: 2
```

5. Tervezzünk programot, amelyben a *main()* függvény két paramétert vesz át, egy betű karaktert és egy szöveget! Írjunk *masol* függvényt, amely a *t* karaktertömbben kapja a szöveget, a *k* karakter tömb 0. elemében a betoldandó karaktert! A *b* tömb tartalmazza a bővített szöveget. (GYAK7_5)

```
void masol(char t[], char k[], char b[]);
```

A program futtatása:

```
Project1 - bemutat
```

A program futásának eredménye:

```
Eredeti szoveg: bemutat
Betoltando betu: -
Bovitett szoveg: b-e-m-u-t-a-t-
```