

8. Gyakorlat

Rövid elméleti összefoglaló: Fájlkézelés

- **Karakter írása és olvasása**
 - `#include <fstream>` szükséges a fájlkézelő függvények használatához.
 - Állomány nyitása írásra, például:
`ofstream Fir;`
`Fir.open("adatok.dat");`
 - Egy karakter írása a megnyitott állományba, például:
`char kar = 'A';`
`Fir.put(kar);`
 - Állomány nyitása olvasásra, például:
`ifstream Folv;`
`Folv.open("adatok.dat");`
 - Hibavizsgálat: Ha a fájlművelet alatt nem volt hiba a *fail()* tagfüggvény 0-át ad vissza, különben 1-et.
 - Egy karakter olvasása az olvasásra megnyitott állományból: `Folv.get(kar);`
 - Az állomány végének vizsgálata: az *eof* (end of file - állomány vége) tagfüggvény 0-át ad vissza, ha van még adat, 1-et, ha vége van az állománynak.
`while(!Folv.eof())`
`{ }`
 - Ha csak vizsgálat miatt olvastunk be egy karaktert, azt visszatehetjük a kimenetre:
`Folv.putback(kar);`
 - Az állományok bezárása: `Fir.close(); Folv.close();`

```
//MINTA8_01
#include <iostream>
#include <fstream>
using namespace std;
main()
{ char kar;
  ifstream finp; ofstream fout;
  finp.open("szoveg.dat");
  if (finp.fail())
  {
    cerr << "szoveg.dat input allomany nem letezik!\n";
    cin.get(); exit(1);
  }
  fout.open("masol.dat");
  if (fout.fail()) // vizsgálat nyitási művelet írásra
  {
    cerr << "masol.dat output allomany nyitási hiba!\n";
    cin.get(); exit(1);
  }
  finp.get(kar); // egy karaktert olvasunk be
  while (kar != '?') // vizsgálat a ? karakterre
  {
    fout.put(kar); // karakterenként olvasunk
    finp.get(kar);
  }
  finp.putback(kar); fout.put('!');
  finp.close(); fout.close();
  finp.open("masol.dat"); // megnyitjuk a masol.dat
  if (finp.fail())
  {
    cerr << "masol.dat input allomany nem letezik!\n";
    cin.get(); exit(1);
  }
  finp.get(kar); //olvasunk egy karaktert az állományból
  while (!finp.eof())
  {
    cout << kar; // kiírjuk a karaktert a képernyőre
    finp.get(kar); // egy karaktert olvasunk az állományból
  }
  finp.close(); // lezárjuk az állományt
  cin.get();
}
```

A *szoveg.dat* állomány tartalma:

Konnyu a C++ programozasi nyelv?

A feladat, hogy a ? jel helyett a *masol.dat* állományba ! jelet írjunk.

A program futásának eredménye:

Konnyu a C++ programozasi nyelv!

A *fail()* tagfüggvénnyel megvizsgáljuk, hogy létezik-e az állomány. Ha nem létezik, akkor hibajelzést adunk, és kilépünk a programból.

Karakterenként olvasunk.

Ha nem ?-t olvastunk, akkor a beolvasott karaktert a *masol.dat* állományba írjuk.

Ha ?-t olvastunk, a ciklus utáni utasítás, a *putback* a ? karaktert visszahelyezi az input folyamra és ! karaktert írunk a ? helyett.

A *masol.dat* állomány tartalmát kiíratjuk a képernyőre.

• Fájlkezelés különféle típusú adatokkal

A különféle adatok be-, illetve kivitele az alábbi módon történik:

- A beszúrás műveleti jellel (<<) írhatunk ki információt a kimeneti adatfolyamba.
- A kiemelés műveleti jellel (>>) olvashatunk fájlban tárolt információt a bemeneti adatfolyamból.

▪ Karaktorsorozat (szöveg) fájlkezelése

```
//MINTA8_02
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char sor1[80], sor2[80], sor3[80];
    ifstream inp; // bemeneti fájlfolym
    ofstream out; // kimeneti fájlfolym
    out.open("szoveg.txt"); //írás szoveg.txt állományba
    out << "első sor a kezdet" << endl;
    out << "második sor a folytatás" << endl;
    out << "harmadik sor a vege" << endl;
    out.close(); // az állomány lezárása
    inp.open("szoveg.txt"); // szöveg visszaolvasása
    inp >> sor1;
    inp >> sor2;
    inp >> sor3;
    inp.close(); // az állomány lezárása
    cout << sor1 << endl;
    cout << sor2 << endl;
    cout << sor3 << endl;
    cin.get();
    return 0;
}
```

```
ifstream inp; bemeneti fájlfolym
ofstream out; kimeneti fájlfolym
```

Szöveg írása a *szoveg.txt* állományba:

```
első sor a kezdet
második sor a folytatás
harmadik sor a vege
```

Szöveg visszaolvasása a *szoveg.txt* állományból.

A program futásának eredménye:

```
első
sor
a
```

Megjegyzés: A bemeneti adatfolyam a **cin**-hez hasonlóan az elválasztó (jelen esetben a szóköz) karakterig olvasott, ezért a három sorban az első sor szóközzel elválasztott karaktorsorozata jelenik meg.

▪ Olvasás a sor végéig: *getline()*

- A *getline()* tagfüggvény a sor végéig ('\n') olvas, de a beolvasott karaktorsorozat nem tartalmazza a sort lezáró ('\n') karaktert. A függvény második paramétere a sztring hossza, harmadik paraméter a sort záró karakter ('\n').

```
ifstream& getline(char*, int, char = '\n');
```

```
//MINTA8_03
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char sor1[80], sor2[80], sor3[80];
    ifstream inp; ofstream out;
    out.open("szoveg.txt");
    out << "első sor a kezdet" << endl;
    out << "második sor a folytatás" << endl;
    out << "harmadik sor a vege" << endl;
    out.close(); // állomány lezárása
    // szoveg.txt állomány megnyitása olvasásra
    inp.open("szoveg.txt");
    inp.getline(sor1, sizeof(sor1)); // olvasás a sor végéig
    inp.getline(sor2, sizeof(sor2));
    inp.getline(sor3, sizeof(sor3));
    inp.close(); // állomány lezárása
    cout << sor1 << endl;
    cout << sor2 << endl;
    cout << sor3 << endl;
    cin.get();
    return 0;
}
```

A *szoveg.txt* állományból a sor végéig olvas a *getline()*.

A program futásának eredménye:

```
első sor a kezdet
második sor a folytatás
harmadik sor a vege
```

- **Numerikus adatok írása, olvasása**

- Bemeneti állomány esetén:
Olvasás egy valós változóba: `inp >> a;`
- Kimeneti állomány esetén:
Kiírás fixpontosan
`out.setf(ios::fixed);`
A tizedespont és a tizedespont utáni vezető nullák kiírásra kerülnek:
`out.setf(ios::showpoint);`
A értékes jegyek száma is beállítható
`out.precision(3);`
Valós számok kiírása soremeléssel:
`out << ossz << endl;`
`out << atlag << endl;`

```
//MINTA8_04
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    int db = 0;
    double a, ossz = 0, atlag;
    ifstream inp;
    ofstream out;
    char InpFnev[20], OutFnev[20];
    cout << "Input fajlnev: ";
    cin >> InpFnev; // bemeneti állomány neve (adat.txt)
    inp.open(InpFnev);
    if (inp.fail()) // vizsgálat: létezik-e az állomány
    {
        cout << InpFnev << " nem letezik!" << endl;
    }
    cout << "Output fajlnev: ";
    cin >> OutFnev; // kimeneti állomány neve (ered.txt)
    out.setf(ios::fixed); // kiírási formátum beállítása
    out.setf(ios::showpoint);
    out.open(OutFnev); // kimeneti állomány létrehozása
    if (out.fail())
    {
        cout << OutFnev << " létrehozás nem sikerült!"
        << endl;
        exit(1);
    }
    out.precision(3);
    cout << "A beolvasott adatok " << endl;
    inp >> a; // egy valós típusú adatot olvas
    while (!inp.eof())
    {
        cout << a << " "; // kiírja az állományba
        ossz += a; // az adatok összegét képezi
        db++; // megszámlálja az adatok számát
        inp >> a; // olvas egy újabb adatot
    }
    atlag = ossz/db; // kiszámítja az adatok átlagát
    //megjeleníti a képernyőn az összeget és az átlagot
    cout << endl << "Összeg: " << ossz << endl;
    cout << "Átlag: " << atlag << endl;
    // kiírja az állományba is
    out << ossz << endl; out << atlag << endl;
    inp.close(); // lezárja a bemeneti állományt
    out.close(); // lezárja a kimeneti állományt
    cin.get(); cin.get();
    return 0;
}
```

Olvasás az *adat.txt* állományból:

A *adat.txt* tartalma:

2.5
4.1
5.6
4.8

A program futásának eredménye:

Input fajlnev: adat.txt
Output fajlnev: ered.txt
A beolvasott adatok
2.5 4.1 5.6 4.8
Összeg: 17
Átlag: 4.25

Az *ered.txt* állomány tartalma:

17.000
4.250

A létrehozás sikerességének vizsgálata.

Olvasás a bemeneti állományból az *a* változóba

Az *ossz* és az *atlag* kiírása a kimeneti állományba soremeléssel.
Az állományok lezárása.

- **Az ios osztály használata**

Megnyitási mód	Hatása
<code>ios::app</code>	Megnyitja az állományt (<i>append</i>) hozzáfűzési módba, és a fájlmutatót a fájl végére helyezi.
<code>ios::ate</code>	A fájlmutatót a fájl végére helyezi.
<code>ios::in</code>	A fájl bevitelre (<i>input</i>) nyitja meg, <i>ifstream</i> esetén ez az alapértelmezés.
<code>ios::out</code>	A fájl kivitelre (<i>output</i>) nyitja meg, <i>ofstream</i> esetén ez az alapértelmezés.
<code>ios::binary</code>	A fájl bináris módban nyitja meg.
<code>ios::trunc</code>	Ha a fájl nem létezik, akkor létrehozza, ha létezik, megnyitja és felülírja.
<code>ios::nocreate</code>	Ha a megadott fájl nem létezik, akkor nyitási művelet hibát jelez.
<code>ios::noreplace</code>	Ha a fájl létezik, akkor nyitási művelet hibát jelez, kivéve ha az <i>ate</i> vagy <i>app</i> van beállítva.

```
//MINTA8_05
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream inp;
    ofstream out;
    char fnev[20];
    int db;
    double adat;
    cout << "Az állomány neve: ";
    cin >> fnev;
    out.open(fnev);
    cout << "Adatok száma: "; cin >> db;
    for (int i = 1; i <= db; i++)
    {
        cout << "Adat: "; cin >> adat;
        out << adat << endl;
    }
    out.close();
    out.open(fnev, ios::app);
    cout << "Új adatok száma: ";
    cin >> db;
    for (int i = 1; i <= db; i++)
    {
        cout << "Adat: "; cin >> adat;
        out << adat << endl;
    }
    out.close(); // állomány lezárása.
    inp.open(fnev);
    cout << fnev << " állomány tartalma:\n";
    inp >> adat;
    while (!inp.eof())
    {
        cout << adat << " "; inp >> adat;
    }
    inp.close(); // az állomány lezárása
    cin.get(); cin.get();
    return 0;
}
```

Az állományhoz való hozzáfűzés bemutatása (*app*).

Állomány nevének beolvasása.

Állomány megnyitása kivitelre.
Adatok számának beolvasása.

Adatok fájlba írása.

Az állomány lezárása.
Állomány megnyitása *app*, azaz hozzáfűzés módban.
Új adatok darabszámának beolvasása,

majd az új adatok beolvasása,
és fájlba írása hozzáfűzéssel.

A kimeneti állomány visszaolvasása és az adatok megjelenítése.

A program futásának eredménye:

```
Az állomány neve: adat.txt
Adatok száma: 3
Adat: 2
Adat: 5
Adat: 4
Új adatok száma: 2
Adat: 9
Adat: 12
adat.txt állomány tartalma:
2 5 4 9 12
```

- **Bináris fájlkezelés**

Az összetett adattípusokat (tömböt és struktúrákat) kényelmesebb binárisan fájlba írni (*write*), illetve onnan olvasni (*read*).

– **Írás:** *ostream::write()* tagfüggvénye

```
ostream& write(const signed char*, int n);
ostream& write(const unsigned char*, int n);
```

– **Olvasás:** *istream::read()* tagfüggvénye

```
istream& read(signed char*, int);
istream& read(unsigned char*, int);
```

- **Struktúra adatainak kiírása bináris állományba**

```
//MINTA8_06
#include <iostream>
#include <fstream>
using namespace std;
struct aru{
    char azonosito[15];
    double sulya;
    int ara;
};

int main()
{
    aru r;
    int db;
    ofstream out_aru;
    out_aru.open("aruk.dat");
    cout << "Arufajtak szama: "; cin >> db;
    for (int i=0; i<db; i++)
    {
        cout << endl;
        cout << "Aru azonositoja: ";
        cin >> r.azonosito;
        cout << "Aru sulya: "; cin >> r.sulya;
        cout << "Aru ara: "; cin >> r.ara;
        out_aru.write((char*)&r, sizeof(aru));
    }
    out_aru.close(); // lezárjuk az állományt
    cin.get(); cin.get();
    return 0;
}
```

Struktúra kiírása bináris állományba.

Rákérdezzünk az áru fajták számára (*db*).

Beolvassuk az áruk azonosítóját, súlyát és árát.

Kiírjuk a struktúra adatait a bináris *aruk.dat* állományba.

A program futásának eredménye:

Arufajtak szama: 3

Aru azonositoja: alma

Aru sulya: 2

Aru ara: 100

Aru azonositoja: szilva

Aru sulya: 3

Aru ara: 150

Aru azonositoja: banan

Aru sulya: 4

Aru ara: 200

- **Struktúra adatainak visszaolvasása bináris állományból**

```
//MINTA8_07
#include <iostream>
#include <fstream>
using namespace std;
typedef struct aru{
    char azonosito[15];
    double sulya;
    int ara;
};

int main()
{
    aru r;
    int db;
    double RaktarErteke = 0, OsszSuly = 0;
    ifstream inp_aru;
    inp_aru.open("aruk.dat");
    inp_aru.read((char*)&r, sizeof(aru));
    while (!inp_aru.eof())
    {
        cout << end << "Aru azonositoja: "
              << r.azonosito << endl;
        cout << "Aru sulya: " << r.sulya << endl;
        cout << "Aru ara: " << r.ara << endl;
        OsszSuly += r.sulya;
        RaktarErteke += r.sulya*r.ara;
        inp_aru.read((char*)&r, sizeof(aru));
    }
    inp_aru.close();
    cout << "\nAruk ossz sulya: " << OsszSuly << endl;
    cout << "Aruk ossz erteke: "
          << RaktarErteke << endl; cin.get();
    return 0;
}
```

A program beolvassa a *MINTA8_06* által írt *aruk.dat* állományt tartalmát.

Az *aruk.dat* állományban binárisan tárolt *aru* struktúra adatainak visszaolvasása. Az állományból való olvasáskor összeadjuk az áruk súlyát, és kiszámítjuk az összértéküket.

A program futásának eredménye:

Aru azonositoja: alma

Aru sulya: 2

Aru ara: 100

Aru azonositoja: szilva

Aru sulya: 3

Aru ara: 150

Aru azonositoja: banan

Aru sulya: 4

Aru ara: 200

Aruk ossz sulya: 9

Aruk ossz erteke: 1450

Feladatok

1. Olvassunk be egy karaktersorozatot a pontig a *SZOVEG.DAT* állományból! Számláljuk meg magánhangzók és az üres helyek számát. (GYAK8_1)

Például a *SZOVEG.DAT* tartalma:

A C++ programozási nyelv megtanulható.

Használjuk az alábbi definíciót, valamint a *SZOVEG.DAT* beolvasása és kiértékelése:

```
char kar;
int osszkar=0, maganh=0, space=0;

ifstream finp;
finp.open("szoveg.dat");
if (finp.fail())
{
    cerr << "szoveg.dat input allomany nem letezik!\n";
    cin.get();
    exit(1);
}
finp.get(kar);
cout << "A beolvasott mondat: " << endl;
while (kar != '.')
{
    cout << kar;
    osszkar++;
    switch (kar)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'u':
        case 'o': maganh++; break;
        case ' ': space++; break;
    }
    finp.get(kar);
}
cout << kar << endl;
finp.close();
. . .
```

A program futásának eredménye:

A beolvasott mondat:
A C++ programozási nyelv megtanulható.

Az összes karakter szama: 37
Magánhangzok szama: 11
Üres helyek szama: 4

2. Generáljunk 10 véletlen számot 1 és 10 között, és írjuk ki azokat a *VELETLEN.TXT* állományba! A *VELETLEN.TXT* állományból olvassuk vissza, és jelenítsük meg a beolvasott adatokat! Számláljuk meg a páros, a páratlan és a 4-gyel osztható számokat. A párosakat adjuk össze, számítsuk ki a páratlanoknak összegét és átlagát. (GYAK8_2)

szam = rand()%10+1;

véletlenszám-generátor kezdőértéke: srand(unsigned(time(NULL)));

```
#include <cstdlib>
#include <fstream>
#include <ctime>
using namespace std;
int main()
{
    int i, db = 10, vszam, parosdb= 0, paratlandb = 0, negyoszt = 0;
    int parosossz= 0, ptosszeg = 0;
    double ptatlag;
    ifstream inp;
    ofstream out;
```

```

. . .
srand((unsigned)time(NULL));
for(i = 0; i<db; i++)
{
    vszam = rand()%10+1;
. . .
}

```

3. Használjuk az alábbi definíciót!

```

struct utazas{
    char uticel[15];
    int letszam;
    double ara;
    int napokszama;
};

```

Az adatok feltöltésekor legyen az úti cél Berlin is! Írjuk ki az *utazas* struktúra adatait az *UTAZAS.DAT* bináris állományba! (GYAK8_3)

4. Olvassuk be az *UTAZAS.DAT* állomány tartalmát, és számláljuk meg az utak számát, az összes utasok számát, valamint azt, hogy hányan utaznak Berlin városába! Melyik a legdrágább és melyik a legolcsóbb út! (GYAK8_4)

A struktúra azonos 3. feladatnál megadottal, használjuk az alábbi definíciókat!

```

utazas ut;
int i, utakszama = 0, osszutas = 0, berlindb = 0;
double dragajegy = 0, olcsout;
ifstream inp;
. . .

```

Az *UTAZAS.DAT* állományból táblázatosan írjuk vissza az adatokat:

```

cout << "        Uticel        Ara    Letszam" << endl;
while (!inp.eof())
{
    cout << setw(15) << ut.uticel;
    cout << setw(8) << ut.ara;
    cout << setw(7) << ut.letszam << endl;
    . . .
}

```

Példaként a program futásának eredménye, ha az *UTAZAS.DAT* állománynak ez a tartalma:

Uticel	Ara	Letszam
Berlin	15000	12
Praga	25000	22
Budapest	28000	12
Szofia	14000	13
Oszlo	2000	22

```

Osszes utas szama: 81
Utak szama: 5
Berlibe utazok szama: 12
Legdragabb jegy: 28000
Legolcsobb ut: 2000

```