Wigner GPU Lab

6. fejezet

Stringek, streamek és a fájlrendszer

Haladó Alkalmazott Programozás
ELTE, 2019

C++ példák

Stream-ek, string-ek

C++ stream-ek

A stream-ek adatfolyamokat reprezentálnak, amik potenciálisan végtelenek is lehetnek és csak egy irányban írhatok és/vagy olvashatóak.

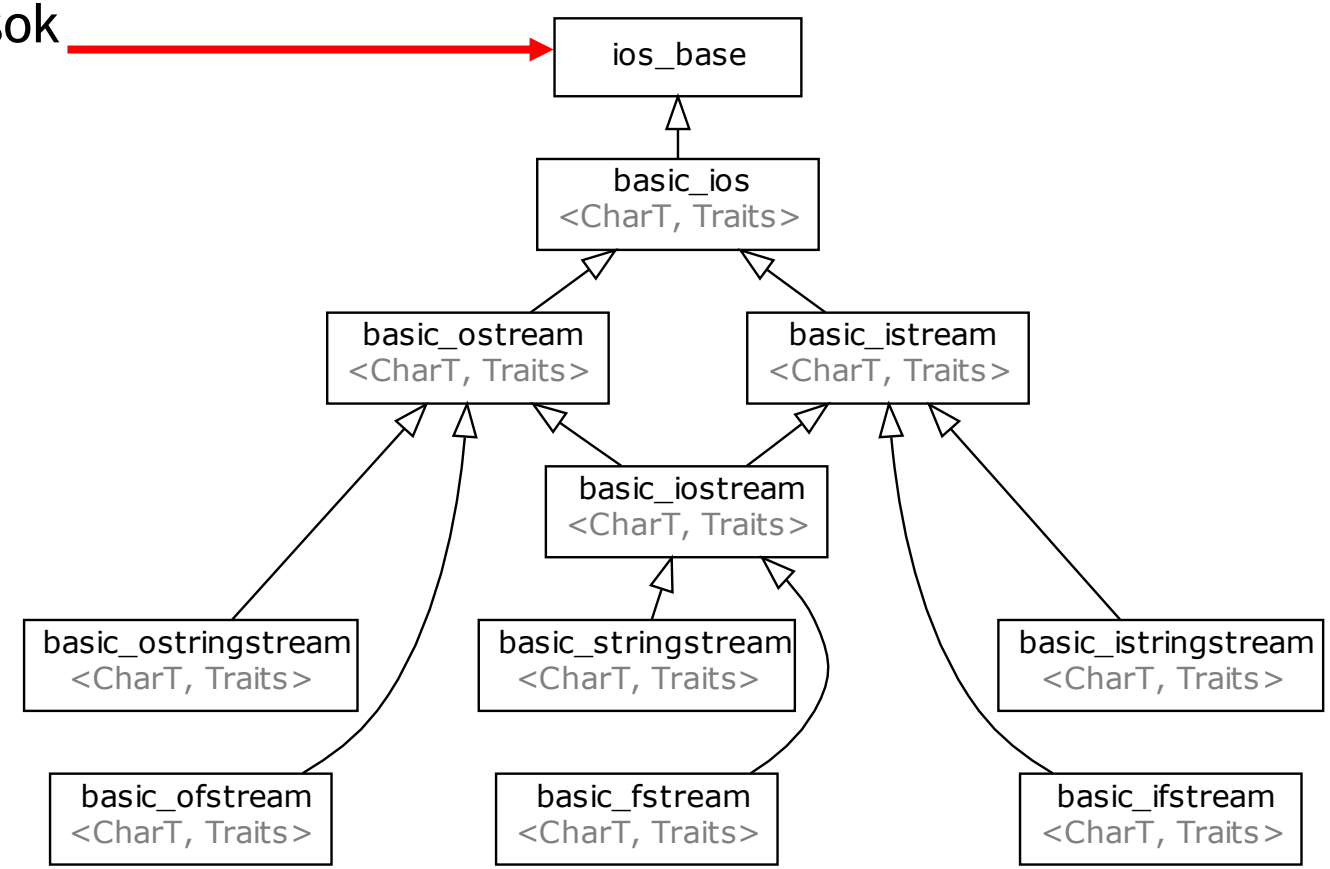
Példák:

- Fájlok
- Standard kimenet, bemenet, hiba
- Külső eszközök
- Hálózat

C++ stream-ek

A C++ a ki/bemeneteket kezelő osztályok hierarchiája:

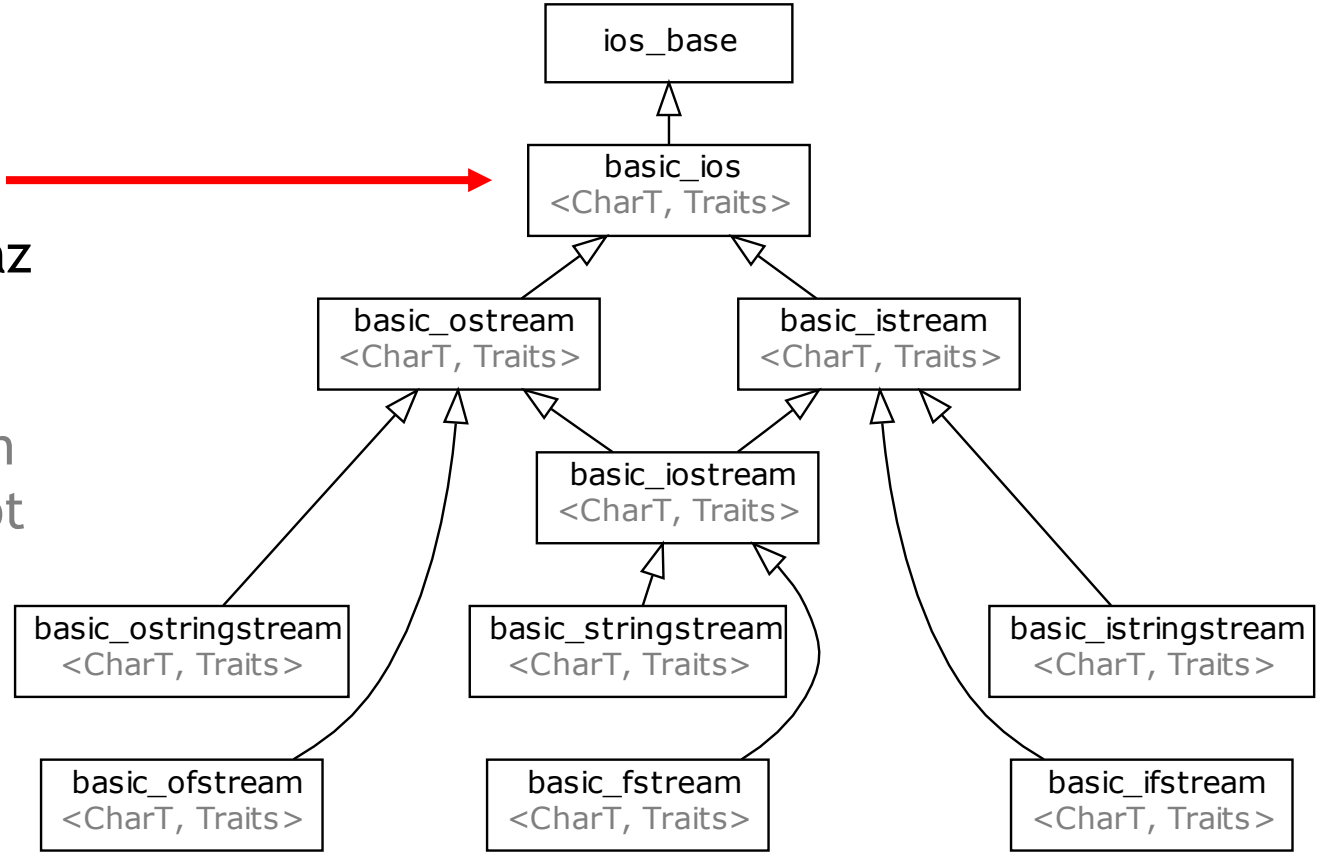
Az ios_base felelős a beépített típusok formázásáért és a stream hibaállapotainak kezeléséért



C++ stream-ek

A C++ a ki/bemeneteket kezelő osztályok hierarchiája:

A basic_ios tartalmaz egy buffert (std::basic_streambuf), amibe átmenetileg tárolja a hozzá tartozó stream egy részét és számon tartja az írási, olvasási pozíciót benne.
Maga a stream általában alacsony szintű operációs rendszer interfészen keresztül éri el a fizikai adatfolyamot

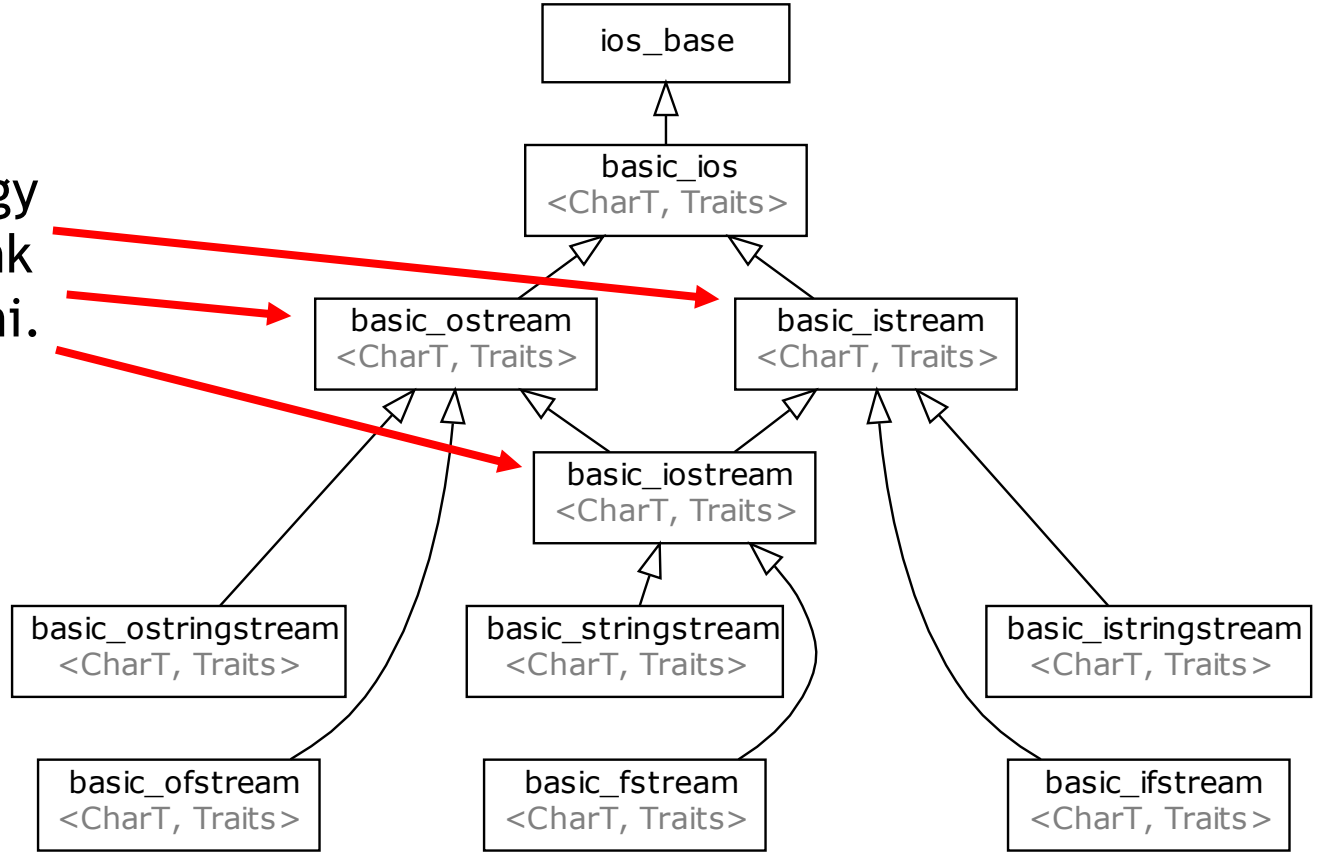


C++ stream-ek

A C++ a ki/bemeneteket kezelő osztályok hierarchiája:

A basic_i/o/iostream objektum teszi elérhetővé azokat a magasszintű műveleteket, amikkel formázott, vagy formázatlan (bináris) adatokat tudunk a streamekbe írni, vagy onnan olvasni.

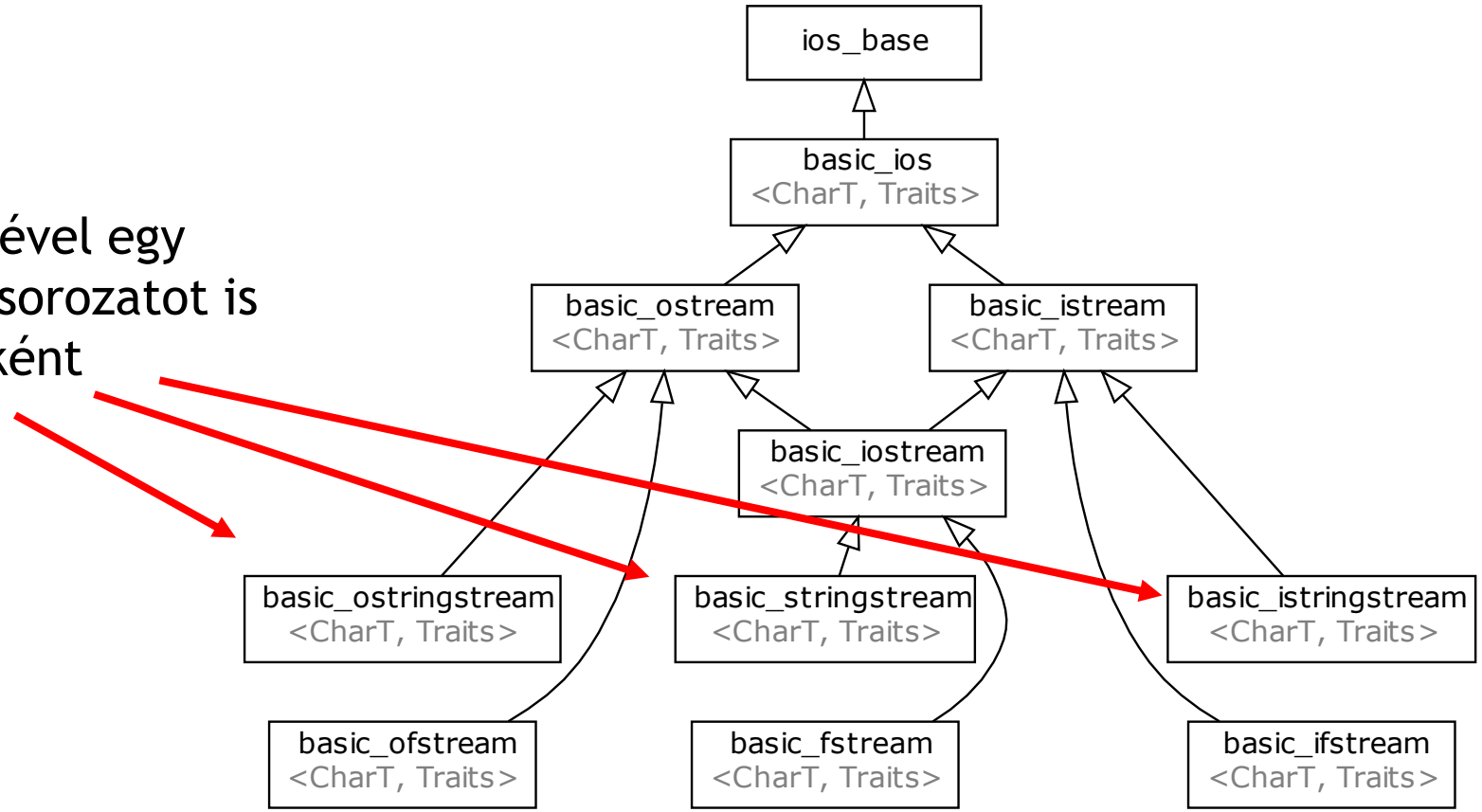
Ez adja az ismerős >> és << operátorokat is, illetve az új túlterheléseket ezzel a típussal kell bevezetni.



C++ stream-ek

A C++ a ki/bemeneteket kezelő osztályok hierarchiája:

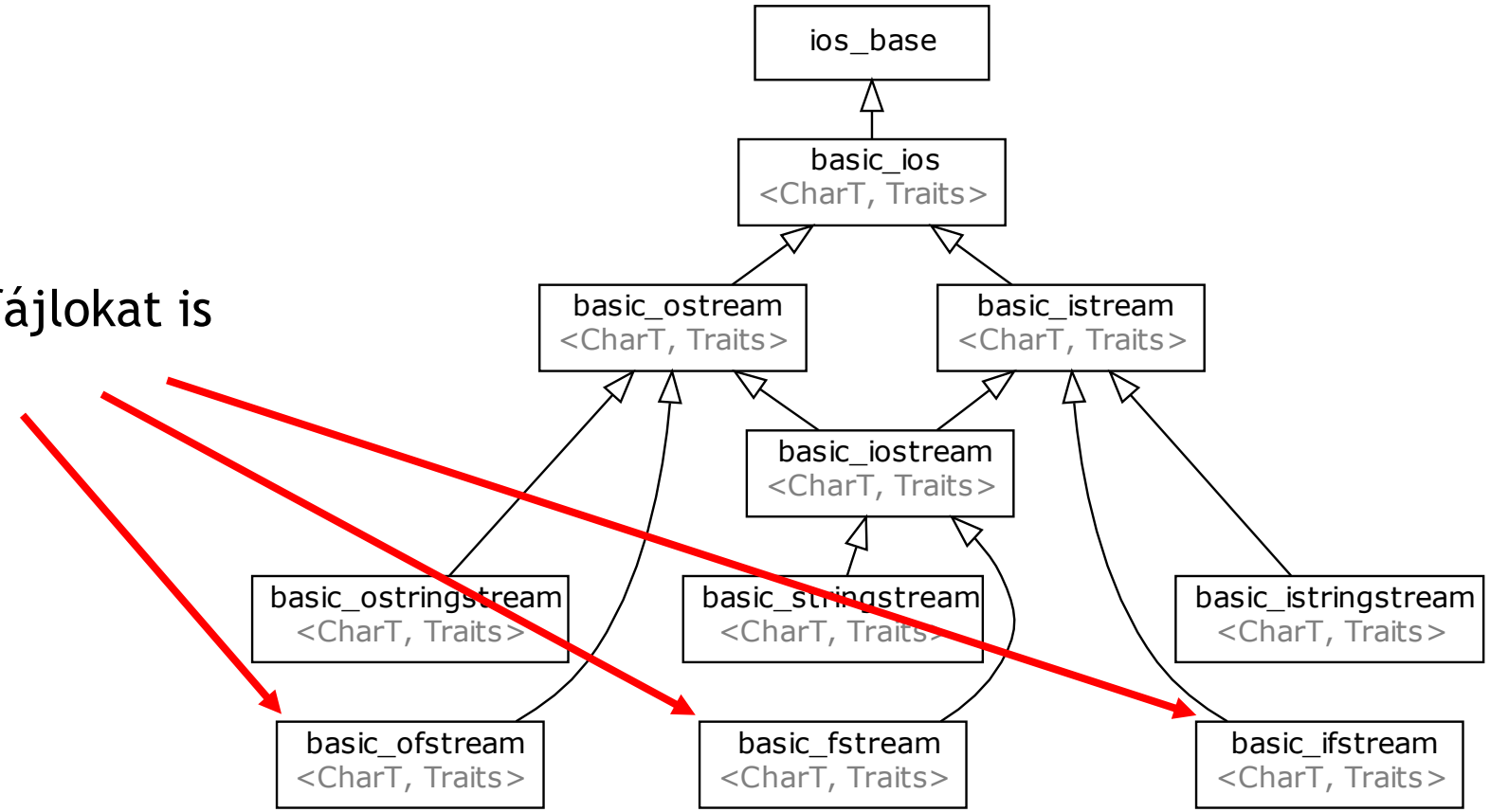
A standard library segítségével egy memóriában levő karaktersorozatot is kezelhetünk i/o/iostreamként



C++ stream-ek

A C++ a ki/bemeneteket kezelő osztályok hierarchiája:

Valamint természetesen fájlokat is



C++ stream-ek

A C++ I/O nagyon összetett és kicsit rosszul megtervezett, de ennek ellenére sok mindent meg lehet oldani 1 sorban vele.

Számos interneten fellelhető példakódban a file I/O elég „fapadosan” van megoldva...

A beolvasáshoz a különböző iterátorok és iterátor adaptorok nagyon hasznosak.

C++ stream-ek

Egy text fájl megnyitása és int-ek másolása belőle egy std::vector-ba:

```
std::vector<int> data;


std::ifstream input("data.txt");
if( input.is_open() )
{
  std::copy( std::istream_iterator<int>(input),
             std::istream_iterator<int>(),
             std::back_inserter(data) );
}
else{ std::cout << "Could not open input file\n"; }
```

C++ stream-ek

Egy text fájl megnyitása és `int`-ek másolása belőle egy `std::vector`-ba:

```
std::vector<int> data;

std::ifstream input("data.txt");
if( input.is_open() )
{
  std::copy( std::istream_iterator<int>(input),
             std::istream_iterator<int>(),
             std::back_inserter(data) );
}
else{ std::cout << "Could not open input file\n"; }
```

A fáljt bemeneti streamként reprezentáló objektum

Ellenőrizni kell, hogy sikerült-e megnyitni a fájlt!

A fálj eleje iterátor

A fájl vége iterátor

Spec iterátor a vectorra, ami a véghez told, ha írnak bele

C++ stream-ek

Egy text fájl megnyitása és `int`-ek másolása bele egy `std::vector`-ból:

```
std::vector<int> data = ...;


std::ofstream output("data.txt");
if( output.is_open() )
{
  std::copy( data.begin(),
             data.end(),
             std::ostream_iterator<int>(output, " ") );
}
else{ std::cout << "Could not open output file\n"; }
```

C++ stream-ek

Egy text fájl megnyitása és `int`-ek másolása bele egy `std::vector`-ból:

```
std::vector<int> data = ...;

std::ofstream output("data.txt");
if( output.is_open() )
{
  std::copy( data.begin(),
             data.end(),
             std::ostream_iterator<int>(output, " ") );
}
else{ std::cout << "Could not open output file\n"; }
```

A fáljt kimeneti streamként reprezentáló objektum

Ellenőrzés, hogy nyitva van-e

Adatok eleje iterátor

Adatok vége iterátor

`int`-eket kiíró, és azokat szóközzel elválasztó iterátor

C++ stream-ek

Még egyszer a különbség:

Beolvasás:

```
std::copy( std::istream_iterator<T>(input),
           std::istream_iterator<T>(),
           std::back_inserter(data) );
```

Kiírás:

```
std::copy( data.begin(),
           data.end(),
           std::ostream_iterator<T>(output, " ") );
```

C++ stream-ek

A stream-ek formázott beolvasás és kiírás operátorai:

```
std::ifstream ifile("data.txt");

int i = 0;
ifile >> i;
```

Átalakítja a streamben aktuálisan levő karaktereket `int`-é ha lehetséges. Ha nem sikerül, `i` nem módosul.

```
std::ofstream ofile("data2.txt");
ofile << i;
```

Átalakítja `i`-t karakterekké és kiírja a stream-be.

C++ stream-ek

Általában a tárolóknak van konstruktora iterátor párból, így `std::copy` nélkül is beolvashatunk:

```
std::ifstream file("data.txt");

std::vector<int> data{std::istream_iterator<int>(file),
                      std::istream_iterator<int>()};
```

Karakter sorozat, string-ek

C++ string-ek

A karaktersorozatokat a C-stílusú `const char*` helyett `std::string`-ekként célszerű kezelni:

```
std::string str("mytext");
```

Az `std::string` ugyan úgy heap-en dinamikusan allokált tároló, mint az `std::vector`, csak némi plusz funkcionalitással.

Minden `""` közötti dolog igazából egy `const char[]` C-s tömb, és a `''` jelekkel egyetlen karaktert jelölünk pl.: `'x'`, ami `const char` típusú.

C++ string-ek

A karaktersorozatokat a C-stílusú `const char*` helyett `std::string`-ekként célszerű kezelni:

```
std::string str("mytext");
```

C++14 óta létezik literál formája is az `std::string`-nek:

```
using namespace std::string_literals;
auto str = "mytext"s;
```

Ez pontosan ugyan azt a stringet írja le, mint az előző példa, és a literál típusa `std::string`.

C++ string-ek

Az std::string egyik konstruktora egy iterátor párt vár, ezzel is beolvashatunk egy fájlt, csak vigyázzunk a formázásra:

```
std::ifstream file("data.txt");
std::string str1{ std::istream_iterator<char>(file),
                  std::istream_iterator<char>()};

file.clear();
file.seekg(0);
std::string str2{ std::istreambuf_iterator<char>(file),
                  std::istreambuf_iterator<char>()};
```

C++ string-ek

Az std::string egyik konstruktora egy iterátor párt vár, ezzel is beolvashatunk egy fájlt, csak vigyázzunk a formázásra:

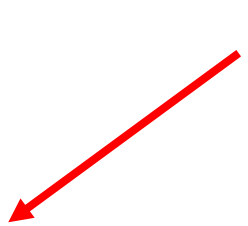
```
std::ifstream file("data.txt");
std::string str1{ std::istream_iterator<char>(file),
                  std::istream_iterator<char>()};

file.clear();
file.seekg(0);
std::string str2{ std::istreambuf_iterator<char>(file),
                  std::istreambuf_iterator<char>()};
```

Ez az iterátor eldobja a white space karaktereket (space, tab, sortörés)

C++ string-ek

Az std::string egyik konstruktora egy iterátor párt vár, ezzel is beolvashatunk egy fájlt, csak vigyázzunk a formázásra:

```
std::ifstream file("data.txt");
std::string str1{ std::istream_iterator<char>(file),
                  std::istream_iterator<char>()};

file.clear();
file.seekg(0);
std::string str2{ std::istreambuf_iterator<char>(file),
                  std::istreambuf_iterator<char>()};
```



Ez az iterátor nem hagyja el a white space karaktereket

C++ string-ek

Az std::string egyik konstruktora egy iterátor párt vár, ezzel is beolvashatunk egy fájlt, csak vigyázzunk a formázásra:

```
std::ifstream file("data.txt");
std::string str1{ std::istream_iterator<char>(file),
                  std::istream_iterator<char>()};

file.clear();
file.seekg(0);
std::string str2{ std::istreambuf_iterator<char>(file),
                  std::istreambuf_iterator<char>()};
```

A file elejére visszamenéshez először törölni kell a file vége jelet a stream-ből,

Majd vissza lehet állítani a beolvasási pozíciót az elejére

C++ string-ek

A string konverziókra C++11-óta van standard módszer:

`std::string` vagy `std::wstring` átalakítása adott numerikus típusra,
pl.: `std::stoi` ha `int`-et akarunk kapni

string-é alakítás: `std::to_string`, `std::to_wstring`

vagy más szemantikával ugyan ezek a funkciók C++17-től:
`std::to_chars`, `std::from_chars`

C++ string-ek

Néhány hasznos művelet string-ekkel:

String-ek összefűzése:

```
std::string s1 = "abc";
std::string s2 = "123";
std::string s3 = s1 + s2;
std::string s4{s1};
s4 += s2;
auto s5 = '*' + s1 + "*";
```

C++ string-ek

Wigner GPU Lab

Néhány hasznos művelet string-ekkel:

String-ek összefűzése:

```
std::string s1 = "abc";
std::string s2 = "123";
std::string s3 = s1 + s2;
std::string s4{s1};
s4 += s2;
auto s5 = '*' + s1 + "*";
```

A + operátor stringeket egymás után rak
s3 értéke `"abc123"`

A += operátor ennek megfelelően az összefűzés eredményét a bal oldali argumentumba írja.

Az operátornál elég, ha az egyik argumentum `std::string` és a másikra pedig van konstruktora az `std::string`-nek, és akkor automatikusan megtörténik az átalakítás

C++ string-ek

Néhány hasznos művelet string-ekkel:

String-ek összehasonlítása:

```
std::string s1 = "abc";
std::string s2 = "abc";
std::string s3 = "ab";
std::string s4 = "xy";
bool b = s1 == s2; //true
bool b2 = s3 < s1; //true
bool b3 = s4 < s3; //false
```

C++ string-ek

Néhány hasznos művelet string-ekkel:

find:

```
std::string s1 = "abcdefghijk"s;
s1.find('f');   //értéke 5
s1.find("ghi"); //értéke 6
s1.find("x");   //értéke npos
```

Az npos egy speciális érték az std::string-ben, ami a -1 átalakítva a find vissza térési értékére, ami egy előjel nélküli egész, nem létező helyet, vagy a string végét, vagy sikertelenséget kódol.

C++ string-ek

Wigner GPU Lab

Néhány hasznos művelet string-ekkel:

substr:

```
std::string s1 = "abcdefghijk"s;

std::string s2 = s1.substr(5);    //értéke "fghijk"
std::string s3 = s1.substr(4, 3); //értéke "efg"
```

C++ stringstream

Egy karakter stringet is használhatunk stream-ként az `std::stringstream` segítségével:

```
std::stringstream ss;
ss << sqrt(2.0);
```

Kiírunk egy lebegőpontos számot a streambe

```
std::string s = ss.str();
```

Le tudjuk kérdezni a stringet

```
float y = 0.0f;
ss >> y;
```

Vissza tudjuk olvasni a stream-ből a karaktereket lebegőpontos számként

```
if( ss ){ std::cout << "Conversion to float succeeded.\n"; }
else    { std::cout << "Conversion to float failed.\n"; }
```

C++ stringstream

A streamek belső bufferén keresztül hatékonyan tudunk teljes adatfolyamokat másolni (formázatlanul):

```
std::ifstream file("data.txt");
std::stringstream ss;

if(file)
{
      ss << file.rdbuf();
}
```

operator >> és <<

Hogyan kell megírni a formázási operátorokat,
azaz a beolvasót >>, és a kiírót << saját típusokra?

operator >> és <<

Vegyünk egy saját típust, pl.:

```
struct Data
{
    int i;
    double x;
    std::string s;
};
```

operator >> és <<

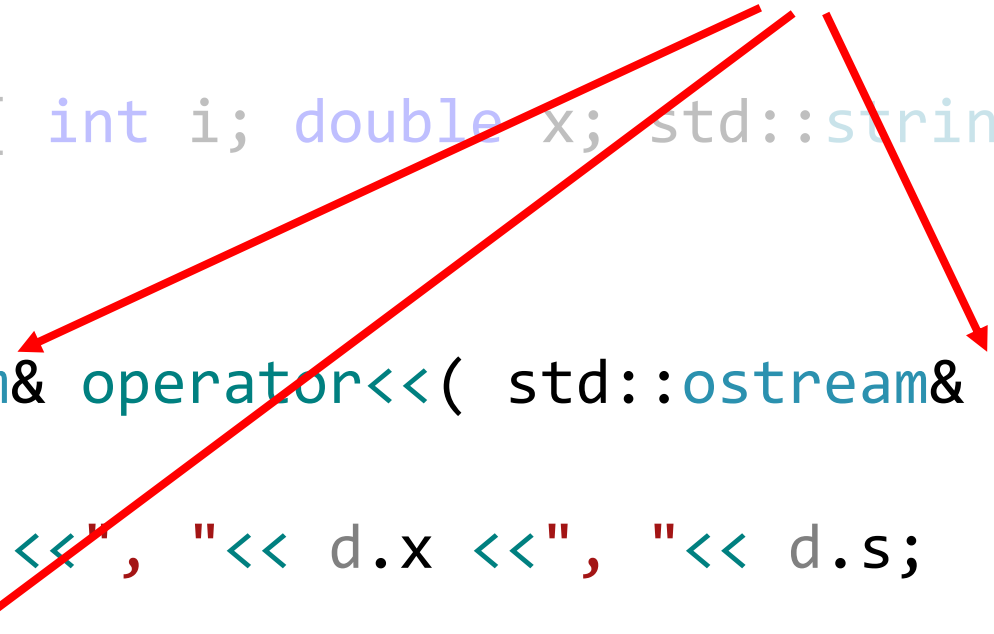
A kiíró operátor:

```
struct Data{ int i; double x; std::string s; };



std::ostream& operator<<( std::ostream& s, Data const  & d )
{
    s << d.i <<", "<< d.x <<", "<< d.s;
    return s;
}
```

operator >> és <<

A kiíró operátor:

A kimeneti stream objektumot referenciaként kötelező átvennünk, és ugyan így visszaadnunk!



```
struct Data{ int i; double x; std::string s; };



std::ostream& operator<<( std::ostream& s, Data const  & d )
{
    s << d.i <<", "<< d.x <<", "<< d.s;
    return s;
}
```

operator >> és <<

Beolvasás, a bemeneti stream-et most is referenciaként kapjuk és adjuk vissza:

```
std::istream& operator>>( std::istream& s, Data& d )
{
        std::string tmp;
        std::getline(s, tmp);
        if(tmp.size() > 0)
        {
            std::stringstream ss(tmp);
            std::getline(ss, tmp, ','); d.i = std::stoi(tmp);
            std::getline(ss, tmp, ','); d.x = std::stod(tmp);
            std::getline(ss, d.s);
         }
         return s;
}
```

operator >> és <<

```
std::istream& operator>>( std::istream& s, Data& d )
{
  std::string tmp;
  std::getline(s, tmp);
  if(tmp.size() > 0){
    std::stringstream ss(tmp);
    std::getline(ss, tmp, ','); d.i = std::stoi(tmp);
    std::getline(ss, tmp, ','); d.x = std::stod(tmp);
    std::getline(ss, d.s);
  }
  return s;
}
```

Olyan formátumot szeretnénk beolvasni, ahol vesszővel vannak elválasztva az értékek:

2, 3.14, abcd
5, 5.67, xyz

operator >> és <<

```
std::istream& operator>>( std::istream& s, Data& d )
{
  std::string tmp;
  std::getline(s, tmp);
  if(tmp.size() > 0){
    std::stringstream ss(tmp);
    std::getline(ss, tmp, ','); d.i = std::stoi(tmp);
    std::getline(ss, tmp, ','); d.x = std::stod(tmp);
    std::getline(ss, d.s);
  }
  return s;
}
```

Az std::getline beolvas egy adott határkarakterig mindent egy streamből egy stringbe.

A harmadik argumentum a határkarakter, alapértelmezésben a sortörés, tehát egy egész sort olvasunk be.

operator >> és <<

```
std::istream& operator>>( std::istream& s, Data& d )
{
  std::string tmp;
  std::getline(s, tmp);
  if(tmp.size() > 0){
    std::stringstream ss(tmp);
    std::getline(ss, tmp, ','); d.i = std::stoi(tmp);
    std::getline(ss, tmp, ','); d.x = std::stod(tmp);
    std::getline(ss, d.s);
  }
  return s;
}
```

Ellenőrizni kell, hogy sikeres volt-e a beolvasás.

operator >> és <<

```
std::istream& operator>>( std::istream& s, Data& d )
{
  std::string tmp;
  std::getline(s, tmp);
  if(tmp.size() > 0){
    std::stringstream ss(tmp);
    std::getline(ss, tmp, ','); d.i = std::stoi(tmp);
    std::getline(ss, tmp, ','); d.x = std::stod(tmp);
    std::getline(ss, d.s);
  }
  return s;
}
```

Készítsünk egy stream-et a beolvasott stringből

operator >> és <<

```
std::istream& operator>>( std::istream& s, Data& d )
{
  std::string tmp;
  std::getline(s, tmp);
  if(tmp.size() > 0){
    std::stringstream ss(tmp);
    std::getline(ss, tmp, ','); d.i = std::stoi(tmp);
    std::getline(ss, tmp, ','); d.x = std::stod(tmp);
    std::getline(ss, d.s);
  }
  return s;
}
```

Majd a stringstream-ből a következő vesszőig kiolvassuk a karaktereket és int-é alakítjuk

operator >> és <<

```
std::istream& operator>>( std::istream& s, Data& d )
{
  std::string tmp;
  std::getline(s, tmp);
  if(tmp.size() > 0){
    std::stringstream ss(tmp);
    std::getline(ss, tmp, ','); d.i = std::stoi(tmp);
    std::getline(ss, tmp, ','); d.x = std::stod(tmp);
    std::getline(ss, d.s);
  }
  return s;
}
```

Majd ismét a következő vesszőig kiolvassuk a karaktereket és double-é alakítjuk

operator >> és <<

```
std::istream& operator>>( std::istream& s, Data& d )
{
  std::string tmp;
  std::getline(s, tmp);
  if(tmp.size() > 0){
    std::stringstream ss(tmp);
    std::getline(ss, tmp, ','); d.i = std::stoi(tmp);
    std::getline(ss, tmp, ','); d.x = std::stod(tmp);
    std::getline(ss, d.s);
  }
  return s;
}
```

Majd ami maradt, azt egyenesen beolvassuk a Data struktúra s nevű string mezőjébe.

operator >> és <<

~~Elegánsabb megoldás: ha nem sikerül a beolvasás, állítsuk helyre a stream~~ beolvasás előtti állapotát:

```
std::istream& operator>>( std::istream& s, Data& d )
{
  const auto state = s.rdstate();
  const auto pos = s.tellg();
  //...
  if(error)
  {
    s.clear();
    s.seekg(pos);
    s.setstate(state);
  }
  return s;
}
```

operator >> és <<

~~Elegánsabb megoldás: ha nem sikerül a beolvasás, állítsuk helyre a stream~~ beolvasás előtti állapotát:

```
std::istream& operator>>( std::istream& s, Data& d )
{
  const auto state = s.rdstate();
  const auto pos = s.tellg();
  //...
  if(error)
  {
    s.clear();
    s.seekg(pos);
    s.setstate(state);
  }
  return s;
}
```

A stream kezdeti állapota és az olvasási pozíció benne.

Ha hiba történt a beolvasott string értelmezésében, vagy bármi másban, akkor vissza állítjuk a stream korábbi állapotát és a korábbi pozíciót visszatérés előtt!

operator >> és <<

Bővebb példa: részecske pályák beolvasása

Formátum:

```
név energia px py pz  px py pz  px py pz ...
```

structured_io.cpp

Másik példa: file_io.cpp

A stringek és a streamek eddigi példái sima ASCII karakterekkel dolgoztak (ami elrejtett template paraméter volt), ahol az elemi karakter elfér az általában 8 bit-es `char` típusban.

A szélesebb, reprezentációhoz a `w_char`, `char16_t`, `char32_t` típusokat és a rájuk specializált stream-eket és stringeket kell használni.

A lokalizáció (pl. ékezetes betűk) és a nem ASCII karakterkezelés, vagy az eltérő kódolások és szélességek közötti átalakítás valódi rémálom, amit sokszor nagyon nehéz hordozhatóan megoldani...

Fájlrendszer

- Fájlrendszer (C++17)

Fájlrendszer

Fájlrendszer kezelés (C++17, `#include <filesystem>`):

```
namespace fs = std::filesystem;
fs::create_directories("test/a/b");
std::ofstream("test/file1.txt");
std::ofstream("test/file2.txt");
for(auto& p : fs::directory_iterator("test"))
{
      std::cout << p << '\n';
}
fs::remove_all("test");
```

Fájlrendszer

Példa:

A legutóbb módosított fájl megkeresése a mappában:

filesystem_ts.cpp

Fájlrendszer

Vigyázat: a fájlrendszer iterátoroknak van belső állapotuk, referencia típusként működnek, azaz, ha másoljuk őket, mind a két iterátor ugyan arra az eredeti objektumra mutat, és azt is módosítják!

Házi feladat

Írjuk meg a korábbi Vector2 és Mátrix osztályhoz az `operator>>` és `<<` műveleteket, figyelve a stream error kezelésre, és tesztként stringstream-el végezzük a kiiratást, beolvasást, mert abból kinyerhető a karakterlánc és összehasonlítható az elvárt string eredménnyel.

Szorgalmi:

Próbáljuk ki a filesystem-et, írjunk olyan programot, ami egy adott mappában (amit a felhasználó futás közben paraméterként megad cin-ről a terminálban) megmondja a legnagyobb méretű fájlt és almappát (ez utóbbihoz rekurzívan be kell járni és felösszegezni a fájlok méreteit).

g++ alatt a filesystem használatához szükség van a CMakeLists.txt-ben az add executable után erre az utasításra:

```
target_link_libraries (${PROJECT_NAME} PRIVATE $<$<PLATFORM_ID:Linux>:stdc++-fs>)
```