Wigner GPU Lab

1. fejezet

A C++ nyelv alapjai

Haladó Alkalmazott Programozás
ELTE, 2019

A C++ nyelv

Tervezési irányelvek: (olvasnivaló)

- Valódi alkalmazásokhoz hasznos nyelv legyen
- Ne erőltesse a fejlesztőket egyetlen programozási stílusba
- Közvetlenül átképezhető legyen a hardverre
- Ne sérüljön implicit módon a típus rendszer
- Kompatibilis legyen a C-vel

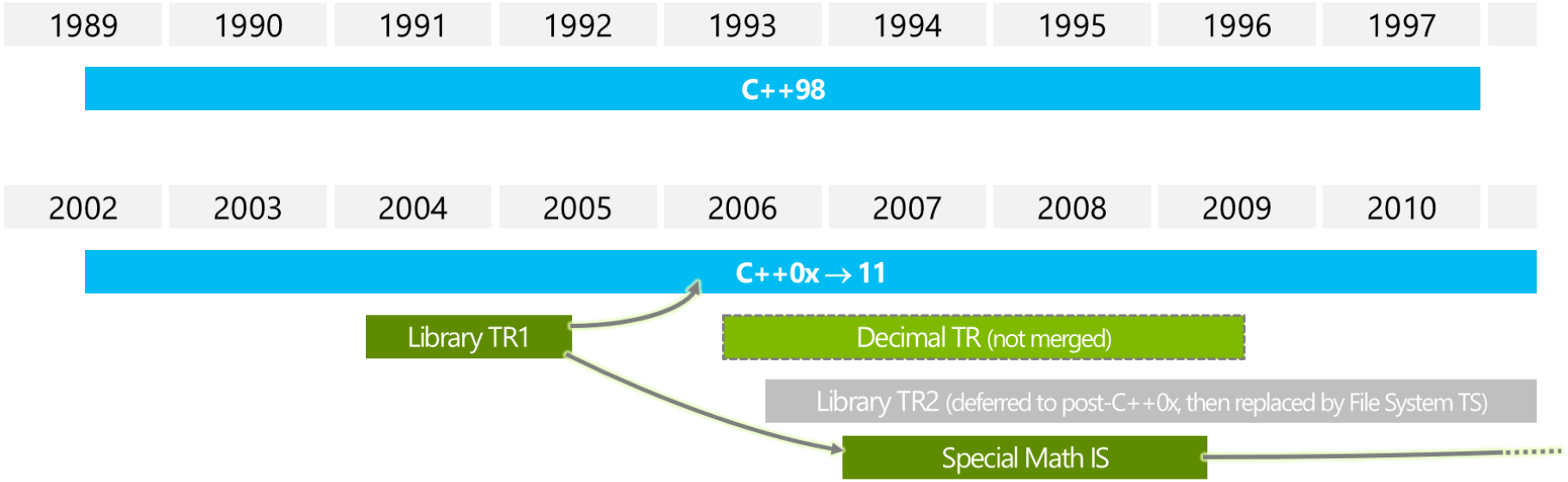
Bjarne Stroustrup
Előadása az ELTE-n, 2014-ben
és a BME-n 2016-ban.

- Ne fizessünk azért, amit nem használunk (zero-overhead rule)

A C++ nyelv

A nyelv kb. 1983-ban különült el végleg a C-től,
Az első hivatalos könyv és fordító 1985-ben jelent meg

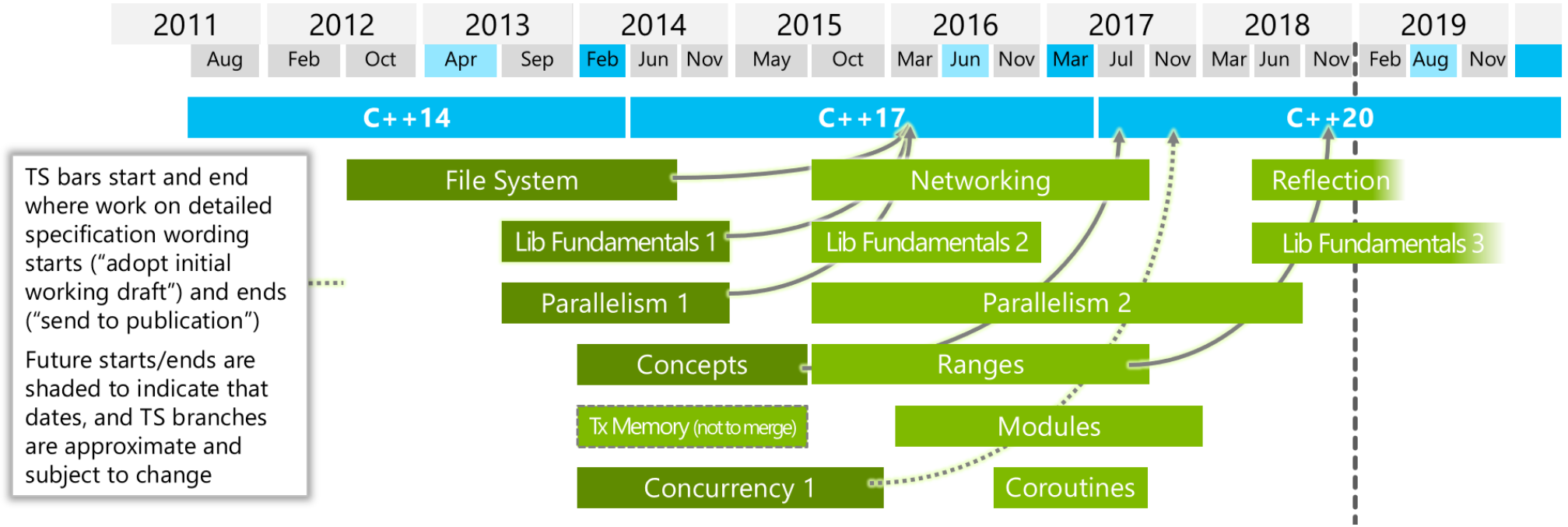
A standardizáció folyamata:



Forrás: isocpp.org

A C++ nyelv

A standardizáció folyamata:



Forrás: isocpp.org

A C++ nyelv

Programozási paradigmák:

- Imperatív
- Objektum-orientált (osztályok, öröklődés)
- Funkcionális (kifejezetten since C++11-óta: lambda függvények)
- Generikus programozás (sablonok, templatek)

A C++ nyelv - fordítók

Ingyenes fordítók:

- Gnu Compiler Collection (GCC):
  Leginkább a Linux/Mac platformon használt,
  Aktuális verzió 8.2, C++17 teljesen támogatott.
- Clang (LLVM):
  Az egyik leggyorsabban fejlődő és leginkább standard kompatibilis platform
  Könnyű fejlesztői eszközöket építeni vele
  Aktuális verzió: 8.0, C++17 teljesen támogatott
  Kompatibilis a GCC-vel. Megy Linux-on, Mac-en, szinte tökéletesen windows-on is
- Visual Studio (IDE) MSVC (Visual C++ fordító):
  A leglassabban fejlődő de már majdnem teljesen C++17 kompatibilis,
  Windows specifikus, ezért sok nem standard viselkedése van

Fizetős fordító:

- Intel C++ compiler: a legjobb teljesítmény, különösen Intel processzorokon,
  még hiányos a C++17 támogatás

A C++ nyelv - fordítók

Online fordítók:

Hasznosak kipróbálni 1-1 új nyelvi elemet, vagy rövidebb programokat ellenőrizni

- godbolt.org — szinte minden létező fordítót támogat, több verzióban is, nem futtatja a programot, csak megmutatja a generált assembly kódot
- coliru — gcc 8.2, clang 5.0, web API
- Tutorialspoint — gcc 7.1.1, teljes shell, akár több file-t is összefordít
- Ideone — gcc 6.3/clang 4.0, futtatja is a programot

A C++ nyelv - fordítók

Ajánlások:

- Használjuk a clang-ot!
  Az egyik legstandardabb fordító, ennek a legérthetőbbek a hibaüzenetei

- Mindig teszteljük le a kódot több fordítóval
  miért? Fordító hibák, nem standard viselkedések
  Linux/Mac alatt gcc + clang, windows alatt msvc + clang

- Használjunk egy integrált fejlesztői rendszert!

A C++ standard

Mi ebben a félévben számos C++17-es újdonságot fogunk használni. Ez még nem mindenhol alapértelmezett, beállítások szükségesek.

Fordítók:

g++, clang: --std=c++17

MSVC: /std:c++17

CMAKE fájl:

```
project (hello)
add_executable(hello main.cpp)
set_target_properties(hello PROPERTIES CXX_STANDARD 17
CXX_STANDARD_REQUIRED ON CXX_EXTENSIONS OFF)
```

A C++ nyelv – fejlesztői rendszerek

Integrált fejlesztői rendszerek:

Jelentősen megkönnyítik a bonyolultabb programok összerakását, fejlesztését, de minimum a kódolást jelentősen támogatják

- Szintaxis kiemelés / színezés
- Kódkiegészítés, átalakítás
- Statikus analízis/fordítási hibák, már beíráskor
- Integrált debuggolás
- Profilozás
- ...

A C++ nyelv

A C++ nyelv

A C++ nyelv

C++ egy bonyolult nyelv, nagyon mélyen el lehet benne merülni, és sokáig tart teljesen kitapasztalni

Ennek ellenére, számos dolgot egyszerűen is meg lehet oldani benne.

A továbbiakban példákon keresztül mutatjuk be a nyelvi elemelet és felhasználásukat

bővebb információk mindig a hivatkozásokban érhetőek el

A C++ nyelv

Ha kérdés merül fel:

1. Google
2. A google valószínűleg ide mutat: stackoverflow.com
3. Ha nincs találat, akkor cppreference.com
4. Kérdezzetek minket☺

A C++ nyelv elemei

A C++ nyelv elemei

Egy C++ kódban a következő elemekkel találkozhatunk:

- Preprocesszor direktívák
- { } blokkok (scope, névterek, statement block-ok)
- Szimbólumok deklarációi és definíciói
  (akár típus, akár függvény, akár érték)
- Literal értékek, függvények (implementációs kód)
- Származtatott adattípusok

A C++ nyelv elemei

Egy definíció
bevezetünk egy szimbólumot, és megadjuk a jelentését is:

```
int a = 5;
```

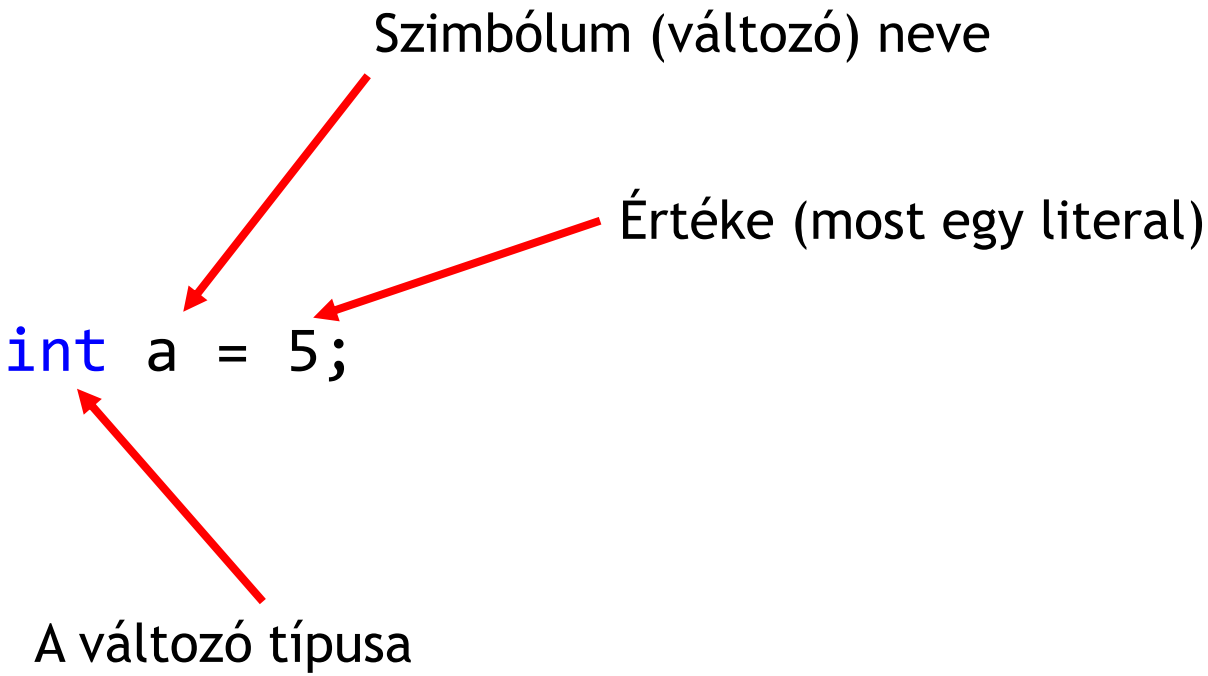
Egy deklaráció
csak bevezetünk egy szimbólumot, csak a típusát mondjuk meg:

```
int a;
```

A definíció egyben deklaráció is.

A C++ nyelv elemei

Egy definíció:



A C++ nyelv elemei

A C++-ban mindennek van típusa.
Néhány beépített típus elsőre:

Előjeles egész szám: `int`
Előjel nélküli egész szám: `unsigned int`

Egyszeres pontosságú lebegőpontos szám: `float`
Dupla pontosságú lebegőpontos szám: `double`

Igaz/hamis értéke: `bool`
ASCII karakter: `char`

Literal érték példa:

Egészek:
```
42
42u
```

Lebegőpontos:
```
42.0f
42.0
```

```
true false
'A'
```

A C++ nyelv elemei

Névterek (a neveket csoportosítják)

```
int a;

namespace A
{
      int b1;
}

namespace B
{
      int b2;
}
```

Mindig van egy legkülső (nem jelölt) globális, minden honnan látható névtér, ha mást nem írunk a szimbólumok abban léteznek.

A névterekkel szabályozhatjuk, mi honnan látszik. Alap esetben a névterekben csak azok a szimbólumok láthatóak, akik a befoglaló névterekben látszanak (egészen felfelé a globális-ig)

Itt most a látszik mindenhol, b1 csak A-ban, b2 csak B-ben.

A C++ nyelv elemei

Névterek (a neveket csoportosítják)

```
int a;

namespace A
{
      int b;
      namespace C
      {
            int c;
      }
}
```

Most az A névtérben látszik ‘a’ és ‘b’, de ‘c’ nem. C-ben látszik mindhárom változó. Globálisan csak ‘a’ látszik.

A C++ nyelv elemei

Névterek (a neveket csoportosítják)

```
int a;
namespace A
{
	int b;
	namespace C
	{
		int c;
	}
}
int x = A::C::c;
```

A névtereken belüli szimbólumokat a névtér neve után ::-al érj-k el

A C++ nyelv elemei

Függvények:

```
int main();
```

Deklaráció (visszatérési típus, függvénynév, argumentum lista)

```
int main()
{
	return 0;
}
```

Definíció (az előzőeken felül a függvény törzsét is tartalmazza (ami nagyjából egy névtér is egyben))

A C++ nyelv elemei

A kötelező hello world:

```
#include <iostream>

int main()
{
	std::cout << "Hello World" << std::endl;
	return 0;
}
```

A C++ nyelv elemei

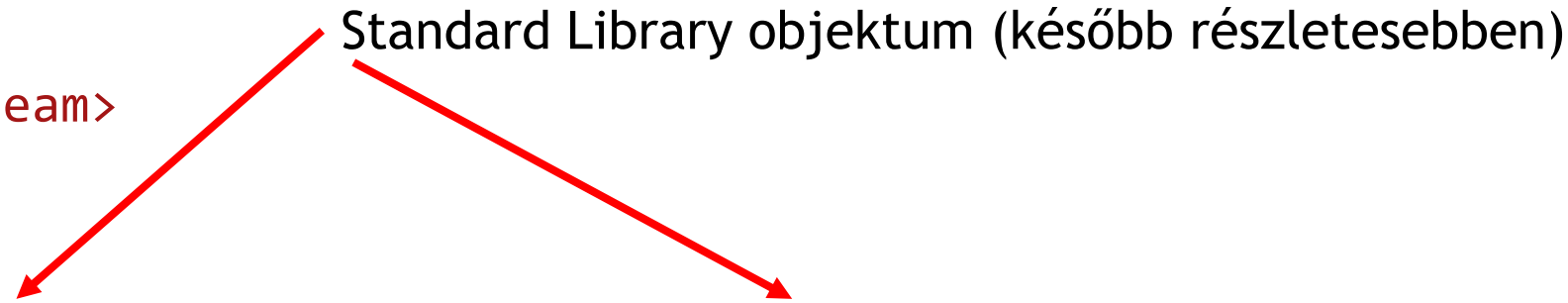
A kötelező hello world:

Preprocesszor direktíva
(lényegében bemásolunk egy másik forrás fájlt (header-t))

```
#include <iostream>

int main()
{
	std::cout << "Hello World" << std::endl;
	return 0;
}
```

A C++ nyelv elemei

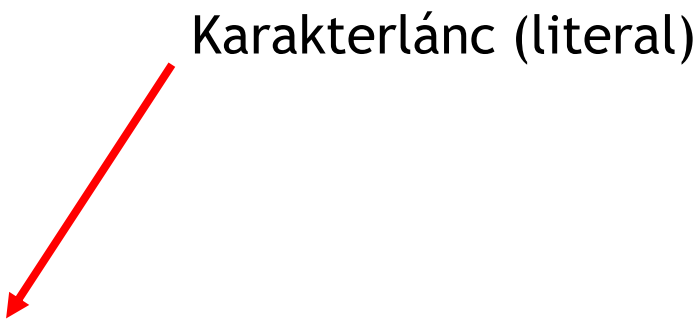
A kötelező hello world:




```
#include <iostream>

int main()
{
	std::cout << "Hello World" << std::endl;
	return 0;
}
```

A C++ nyelv elemei

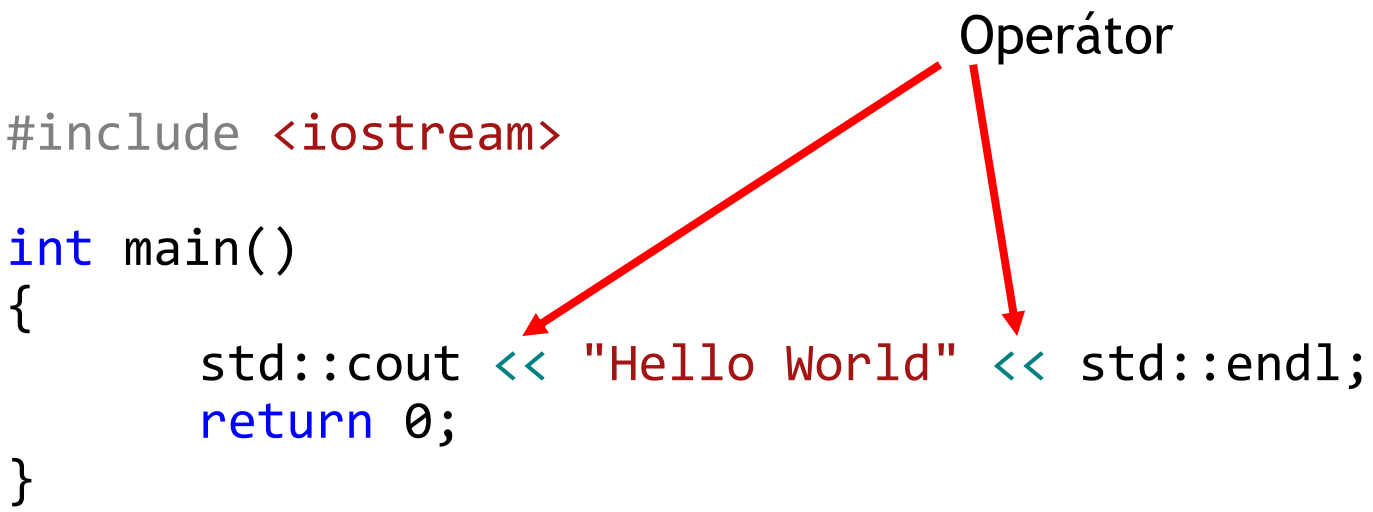
A kötelező hello world:




```
#include <iostream>

int main()
{
	std::cout << "Hello World" << std::endl;
	return 0;
}
```

A C++ nyelv elemei

A kötelező hello world:

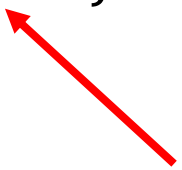



```
#include <iostream>

int main()
{
	std::cout << "Hello World" << std::endl;
	return 0;
}
```

A C++ nyelv elemei

A kötelező hello world:

```
#include <iostream>

int main()
{
	std::cout << "Hello World" << std::endl;
	return 0;
}
```



Függvény végeredményének visszaadása, az utána jövő kifejezés lesz a függvény visszatérési értéke.

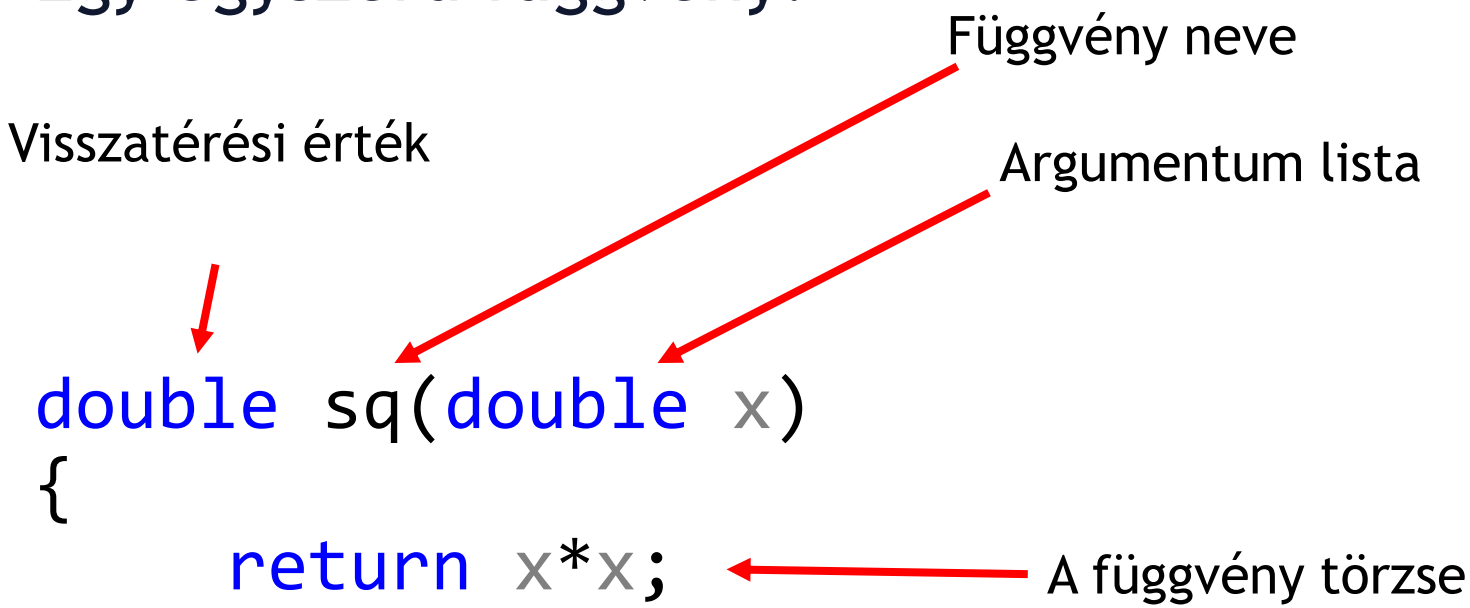
A C++ nyelv elemei

Egy egyszerű függvény:

```
double sq(double x)
{
    return x*x;
}
```

A C++ nyelv elemei

Egy egyszerű függvény:



A C++ nyelv elemei

Egy egyszerű függvény:

A C++11 lehetővé tette, hogy a visszatérési értéket a függvény argumentum listája után adjuk meg, ha előre `auto`-t írunk

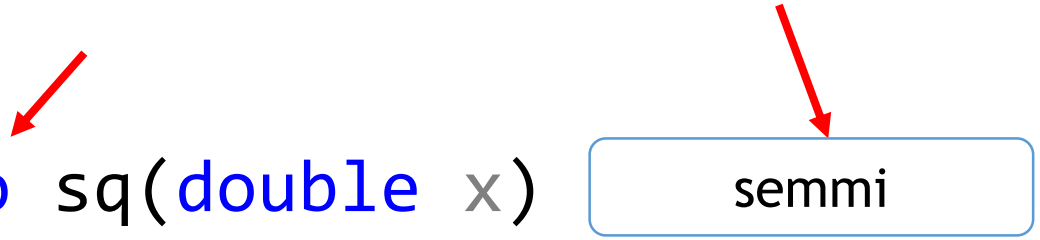
Hogy miért, arra a templatek-nél kitérünk

```
auto sq(double x)->double
{
    return x*x;
}
```

A C++ nyelv elemei

Egy egyszerű függvény:

A C++14-ben pedig teljesen elhagyható, mert a fordító kitalálja a `return` statementekből.



A C++ nyelv elemei

Lambda függvények (C++11): függvény kifejezések („függvény literalok”)

```
double sq(double x)
{
    return x*x;
}
```

```
[](double x)->double{ return x*x; }
```

A C++ nyelv elemei

Lambda függvények (C++11): függvény kifejezések

```
double sq(double x)
{
    return x*x;
}
```

A fenti függvény egy statement block, szemben az alsóval, ami egy kifejezés. Ennek ellenére, ugyan azt fejezik ki.

```
[](double x)->double{ return x*x; }
```

A C++ nyelv elemei

Lambda függvények (C++11): függvény kifejezések

```
double sq(double x)
{
    return x*x;
}
```

Ha névhez akarjuk kötni, akkor csak `auto`-t használhatunk. A lambda függvény típusát csak a fordító ismeri.

Mivel ez egy értékadás, kell a pontosvessző a végére!

```
auto sq = [](double x)->double{ return x*x; };
```

A C++ nyelv elemei

Lambda függvények (C++11): függvény kifejezések

```
auto sq(double x)
{
    return x*x;
}
```

Ugyan úgy, mint a függvényeknél, itt is ki tudja találni a visszatérési típust a fordító, nem kell kiírni:

```
auto sq = [](double x) semmi { return x*x; };
```

A C++ nyelv elemei

Generikus lambda függvények (C++14):

Ha ugyan az a működés kell több típusra:

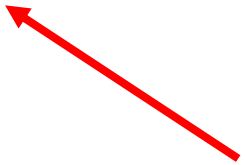
```
auto sq = [](auto x){ return x*x; };
```

A C++ nyelv elemei

Generikus lambda függvények (C++14):

Ha ugyan az a működés kell több típusra:

```
auto sq = [](auto x){ return x*x; };
```



x típusa majd akkor derül ki, ha meghívjuk a függvényt, és akkor a fordító ellenőrzi, hogy a bemenő típusokra helyesek-e a törzsben leírtak

A C++ nyelv elemei

Az `auto` kulcsszóról:

Az `auto` használható nagyon sok helyen, ahol típus ki kéne írnunk, de nem akarjuk, például az előzőeken túl definíciókban:

```
auto alpha = 1.0 / 137.035999139; //ez double lesz
```

Ajánlás: ha csak egyetlen típussal lesz használva a kifejezés / szimbólum, és nincs más indokunk (nem nagyon bonyolult a típus neve) akkor inkább írjuk ki és ne használjunk auto-t, mert így olvashatóbb lesz a kód!

A C++ nyelv elemei

Függvények túlterhelése (ad-hoc polimorfizmus)

Azonos névvel bevezethetünk több függvényt, amíg az argumentumlistájukon keresztül egyértelműen elkülöníthetőek

```
double sq( double x){ return x*x; } //OK

int    sq( int    x){ return x*x; } //OK

double sq( int    x){ return (double)x*x; } //Hiba!
```

Csak a visszatérési érték alapján nem lehet túlterhelni!
Ez a változat nem különíthető el a 2. változattól.

A C++ nyelv elemei

Az operátorok csak „normál” függvények speciális szintaxissal:
Például a korábban használt << operátor igazából ilyesmi:

```
std::ostream& operator<<(std::ostream& o, const char* ch);
```

Számos más operátor szimbólum van, ezek egyrésze egyedi típusokra egyedi jelentéssel ruházható fel (túlterhelés, overloading)

Érdemes tudni a precedenciákat és asszociativitást.

A C++ nyelv elemei

Beépített függvények:

Vannak olyan függvények, amiket nem mi vezetünk be, hanem a nyelv, pontosabban a standard library részei, számunkra elsőre a matematikai függvények a legfontosabbak:

Általános matematikai függvények:

Pl.: abs, fma, exp, log, pow, sqrt, cbrt, sin, cos, tan, ezek inverzei, hiperbolikus fv-ek, és inverzeik, erf, erfc, tgamma, lgamma, kerekítés és egészre vágás, stb.

Speciális matematikai függvények: (C++17)

Elliptikus-, exponenciális integrálok, (gömbi és henger) Bessel fv, beta fv, Riemann zeta fv, Laguerre, Legendre, Hermite polinomok, stb.

Egyéb hasznos: gcd, lcm, min, max

A C++ nyelv elemei

Beépített függvények:

Általános matematikai függvények

Speciális matematikai függvények:

gcd, lcm, min, max

Ajánlás: hacsak nincs alapos indokunk rá, ne írjunk saját matematikai függvényt, használjuk a beépítetteket!

A C++ nyelv elemei

Példa:
3 elemből álló vektor hossza:

```
#include <cmath>
double length(double x, double y, double z)
{
  return std::sqrt(x*x+y*y+z*z);
}
```

A C++ nyelv elemei

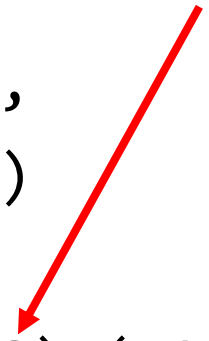
Példa:
2 db 3 elemből álló vektor távolsága:

```
#include <cmath>
double distance(double x1, double y1, double z1,
                double x2, double y2, double z2)
{
return std::sqrt( (x1-x2)*(x1-x2)+(y1-y2)*(y1-x2)+(z1-z2)*(z1-z2) );
}
```

A C++ nyelv elemei

A függvények azért vannak, hogy okosabban szervezzük a kódot, és ne kelljen ugyan azt sokszor leírni, mert úgy is elrontjuk!

```
#include <cmath>
double distance(double x1, double y1, double z1,
                double x2, double y2, double z2)
{
return std::sqrt( (x1-x2)*(x1-x2)+(y1-y2)*(y1-x2)+(z1-z2)*(z1-z2) );
}
```



A C++ nyelv elemei

Egy jobb megoldás:

```
#include <cmath>

double sq(double x){ return x*x; }

double distance(double x1, double y1, double z1,
                double x2, double y2, double z2)
{
  return std::sqrt( sq(x1-x2) + sq(y1-y2) + sq(z1-z2) );
}
```

A C++ nyelv elemei

Vezérlési szerkezetek:

```
//solve a*x+b = 0
void solve(double a, double b)
{
  if( a == 0.0 )
  {
    std::cout << "No solution\n";
  }
  else
  {
    std::cout << "The solution is: " << -b/a << "\n";
  }
}
```

A C++ nyelv elemei

Feladat:

Írjunk egy függvényt, ami megoldja a másodfokú egyenletet, és kiírja képernyőre a megoldás(okat), ha van(nak).

Szignatúra:

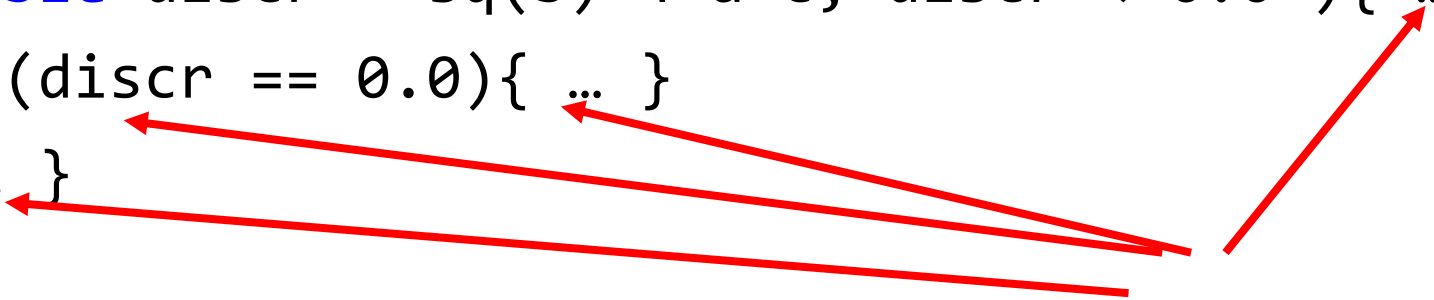
```
//solve a*a*x+b*x+c = 0
void solve(double a, double b, double c);
```

A C++ nyelv elemei

Vezérlési szerkezetek:

C++17 óta az `if` hasába lehet írni definíciót is:

```
if( double discr = sq(b)-4*a*c; discr < 0.0 ){ … }
else if(discr == 0.0){ … }
else{ … }
```



Ha bevezettünk egy nevet az `if` hasában (`discr`), akkor azt az egész elágazás szerkezetben használhatjuk, de utána már nem látszik!

A C++ nyelv elemei

Vezérlési szerkezetek: Ciklusok

```
double sum_of_squares(int n)
{
  double sum = 0.0;
  for(int i=0; i<=n; ++i)
  {
    sum += sq(i);
  }
  return sum;
}
```

A C++ nyelv elemei

Vezérlési szerkezetek: Ciklusok

```
double sum_of_squares(int n)
{
  double sum = 0.0;
  for(int i=0; i<=n; ++i)
  {
    sum += sq(i);
  }
  return sum;
}
```

Definíció

Feltétel (amíg igaz, addig megy a ciklus)

Léptetési kifejezés

Törzs: a Definícióban bevezetett név csak itt látható

A C++ nyelv elemei

Vezérlési szerkezetek: Ciklusok

```
double sum_of_squares2(int n)
{
  double sum = 0.0;
  int i = 0;
  do
  {
    sum += sq(i);
    ++i;
  }while(i <= n);
  return sum;
}
```

```
double sum_of_squares3(int n)
{
  double sum = 0.0;
  int i = 0;
  while(i <= n)
  {
    sum += sq(i);
    ++i;
  }
  return sum;
}
```

A C++ nyelv elemei

Vezérlési szerkezetek: Ciklusok

```
double sum_of_squares2(int n)
{
  double sum = 0.0;
  int i = 0;
  do
  {
    sum += sq(i);
    ++i;
  }while(i <= n);
  return sum;
}
```

```
double sum_of_squares3(int n)
{
  double sum = 0.0;
  int i = 0;
  while(i <= n)
  {
    sum += sq(i);
    ++i;
  }
  return sum;
}
```

Feltétel (addig megy a ciklus, amíg igaz)

A C++ nyelv elemei

Vezérlési szerkezetek: Ciklusok

```
double sum_of_squares2(int n)
{
  double sum = 0.0;
  int i = 0;
  do
  {
    sum += sq(i);
    ++i;
  }while(i <= n);
  return sum;
}
```

```
double sum_of_squares3(int n)
{
  double sum = 0.0;
  int i = 0;
  while(i <= n)
  {
    sum += sq(i);
    ++i;
  }
  return sum;
}
```

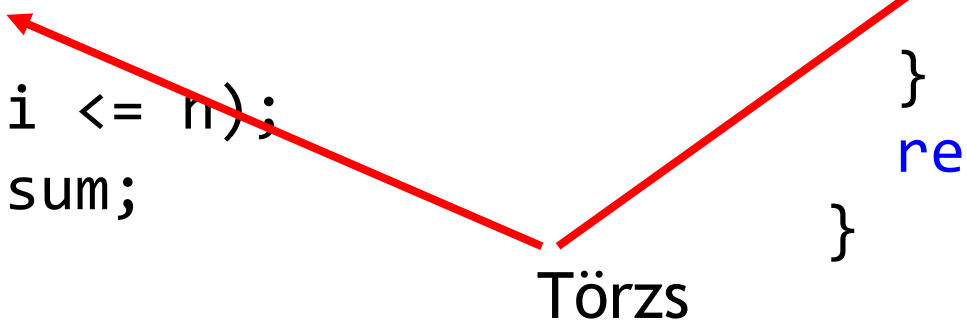
Itt csak az használható, ami a cikluson kívül lett definiálva!

A C++ nyelv elemei

Vezérlési szerkezetek: Ciklusok

```
double sum_of_squares2(int n)
{
  double sum = 0.0;
  int i = 0;
  do
  {
    sum += sq(i);
    ++i;
  }while(i <= n);
  return sum;
}
```

```
double sum_of_squares3(int n)
{
  double sum = 0.0;
  int i = 0;
  while(i <= n)
  {
    sum += sq(i);
    ++i;
  }
  return sum;
}
```



A C++ nyelv elemei

Scope:

A függvények és vezérlési szerkezetek { } blokkjai szabják meg a változók láthatóságát és élettartamát.

```
int main()
{
    {
        int a = 5;
        {
            int b = 8;
        }

        {
            int c = 9;
        }
    }
}
```

A C++ nyelv elemei

Scope:

A függvények és vezérlési szerkezetek { } blokkjai szabják meg a változók láthatóságát és élettartamát.

```
int main()
{
    {
        int a = 5;
        {
            int b = 8;
        }

        {
            int c = 9;
        }
    }
}
```

'a' itt kerül bevezetésre,
és idáig látszik (él)

A C++ nyelv elemei

Scope:

A függvények és vezérlési szerkezetek { } blokkjai szabják meg a változók láthatóságát és élettartamát.

```
int main()
{
    {
        int a = 5;
        {
            int b = 8;
        }

        {
            int c = 9;
        }
    }
}
```

'b' itt kerül bevezetésre,
és idáig látszik (él)

A C++ nyelv elemei

Scope:

A függvények és vezérlési szerkezetek { } blokkjai szabják meg a változók láthatóságát és élettartamát.

```
int main()
{
    {
        int a = 5;
        {
            int b = 8;
        }

        {
            int c = 9;
        }
    }
}
```

'c' itt kerül bevezetésre,
és idáig látszik (él)

A C++ nyelv elemei

Scope:

A függvények és vezérlési szerkezetek { } blokkjai szabják meg a változók láthatóságát és élettartamát.

```
int f(double x)
{
  double sqx = sq(x);
  if(double y = sqx - 10.0;
     y < 5.0)
  {
    return y;
  }
  return sqx;
}
```

'sqx' itt kerül bevezetésre,
és idáig látszik (él)

A C++ nyelv elemei

Scope:

A függvények és vezérlési szerkezetek { } blokkjai szabják meg a változók láthatóságát és élettartamát.

'y' itt kerül bevezetésre,
és idáig látszik (él)

```
int f(double x)
{
  double sqx = sq(x);
  if(double y = sqx - 10.0;
     y < 5.0)
  {
    return y;
  }
  return sqx;
}
```

A C++ nyelv elemei

Scope:

A függvények és vezérlési szerkezetek { } blokkjai szabják meg a változók láthatóságát és élettartamát.

```
int f(double x)
{
  double sqx = sq(x);
  if(double y = sqx - 10.0;
     y < 5.0)
  {
    return y;
  }
  return sqx;
}
```

Ajánlás: minden változót a lehető legszűkebb { } -blokkban, scope-ban használjunk, ne vezessünk be indokolatlanul kívül változókat

A C++ nyelv elemei

Feladat:

Írjunk egy függvényt, ami Newton iterációval gyököt számol.

Link: https://en.wikipedia.org/wiki/Newton%27s_method#Square_root_of_a_number

Szignatúra:

```
//Find the square root of 'num' by starting the iteration from 'x0'
double sqrt_newton(double num, double x0);
```

Debuggolással ellenőrizzük, hogy az első néhány lépés értékei egyeznek-e a wikin írtakkal.

Házi feladat

Házi feladat

Válasszunk ki egy 1 dimenziós numerikus integrátort, például valamelyik kompozit Newton-Cotes formulát (link, 7-8. oldal) és implementáljuk egy kiválasztott függvényre.

Az intervallum határai és a felosztási pontok számai legyenek a függvény paraméterei. A módszerek hibatagjával (ami a valamelyik deriválttal arányos) nem kell most törődni.

Az implementálandó függvény szignatúrája:

```
double integrate(int n, double x0, double x1);
```

Házi feladat

Néhány esetre ellenőrizzük az integrál helyességét például a WolframAlpha segítségével:

Példa:

$$\int_{-1}^{3} \cos(x)\, e^{-x^2}\, \mathrm{d}x$$

WolframAlpha szintaxis:

```
Integrate[Cos[x]*Exp[-x^2], {x, -1, 3}]
```

Közvetlen link, közelítő érték: 1.346387956803450376698 16

Ellenőrizzük a konvergenciát úgy, hogy n értékét növelve kiíratjuk a kimenetre n-et és az integrál értékét.

Házi feladat

A kimenetre írás alapból csak kb. 6 tizedest ír ki, `double` értékeknél célszerű beállítani a pontosságot 16 jegyre:

```
std::cout.precision(16);
```

(ez a beállítás érvényben marad addig, amíg újra nem módosítjuk, ezért elég akár egyszer meghívni a munkánk elején)

Házi feladat

Beküldés:

Az ideális megoldás egy ingyenes kódrepozitóriumban elhelyezés (pl.: github), ebben az esetben oda töltsétek fel a megoldást (pl. minden óra egy mappa), és küldjétek el a linket rá.

Alternatíva:

Küldjétek el a kész fájlt e-mailben a következő címre:

wignergpulab (kukac) gmail (pont) com