BME TMIT
2022

14/2

Németh Gábor
Janky Ferenc Nándor vázlatai alapján

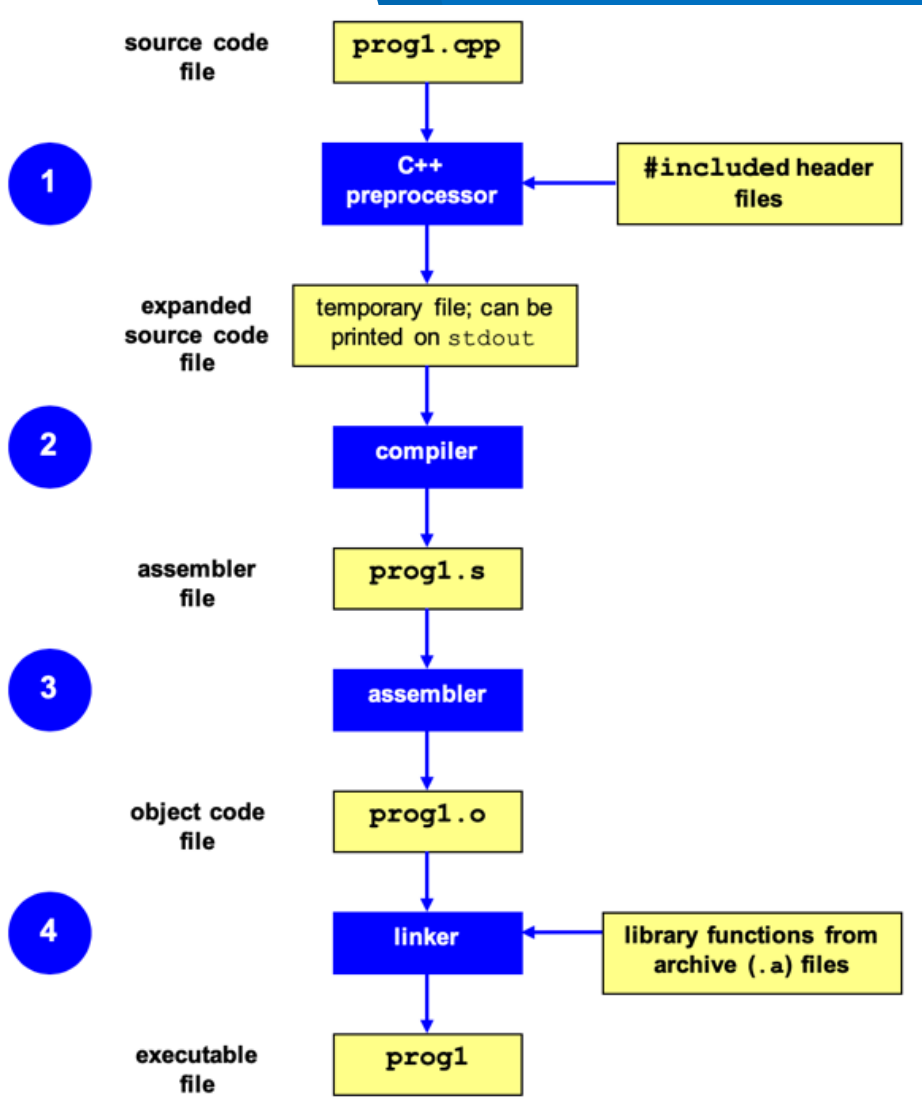
Funkcionális programozás C++-ban

Variadikus makrók, függvények, sablonok

Variadikus függvények

- Nem definiált aritású függvény
  - tetszőleges számú paramétert tud „kezelni”
- C++-ban három félét használhatunk:
  - variadikus makrófüggvény
  - variadikus függvény
  - variadikus sablon
    - függvény
    - osztály
    - változó
    - stb. (nagyjából bárhol, ahol eddig sablonparaméter állhatott)

Variadikus függvények és a fordítási folyamat

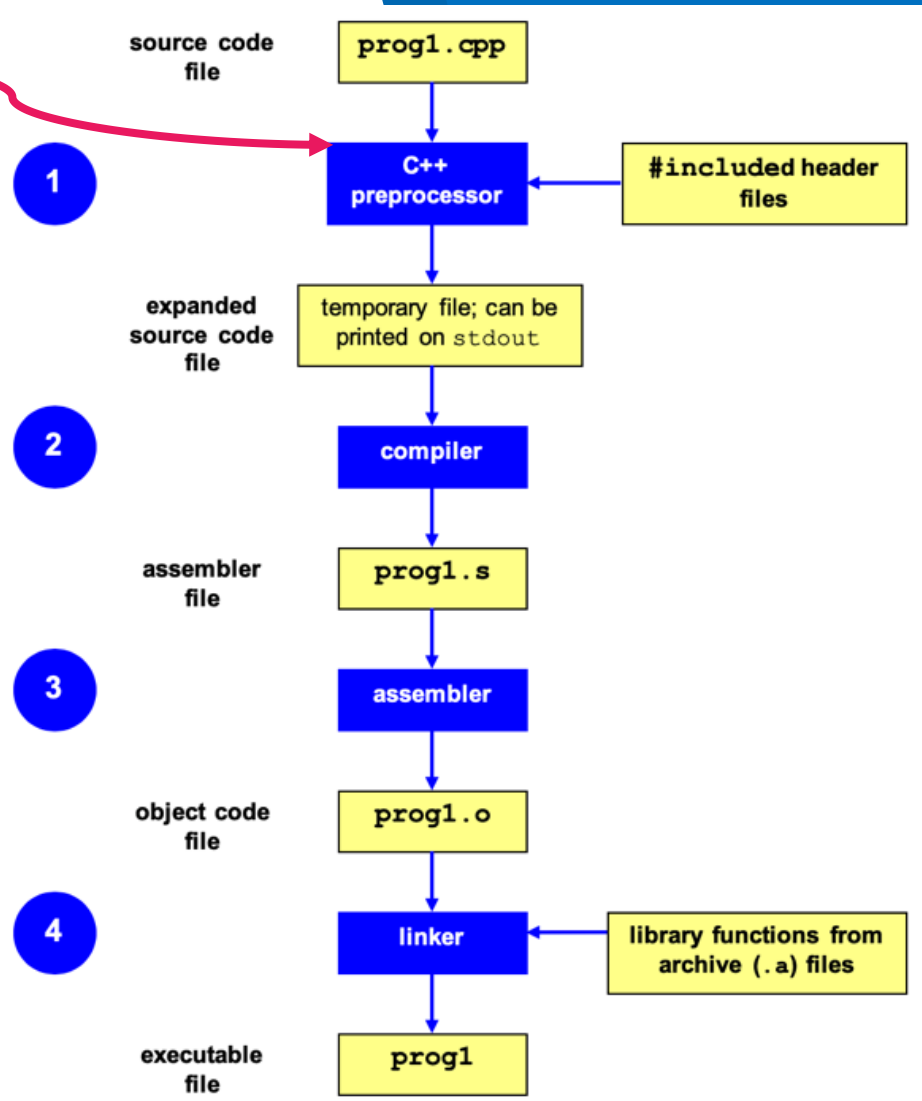


forrás:http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/Images/build.png

Variadikus függvények és a fordítási folyamat

- Variadikus makrók: a preprocesszálás alatt

```
#define PP_LOG(...) \
  ::print_line(__FILE__,__LINE__,__function,## __VA_ARGS__)
```



forrás:http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/Images/build.png

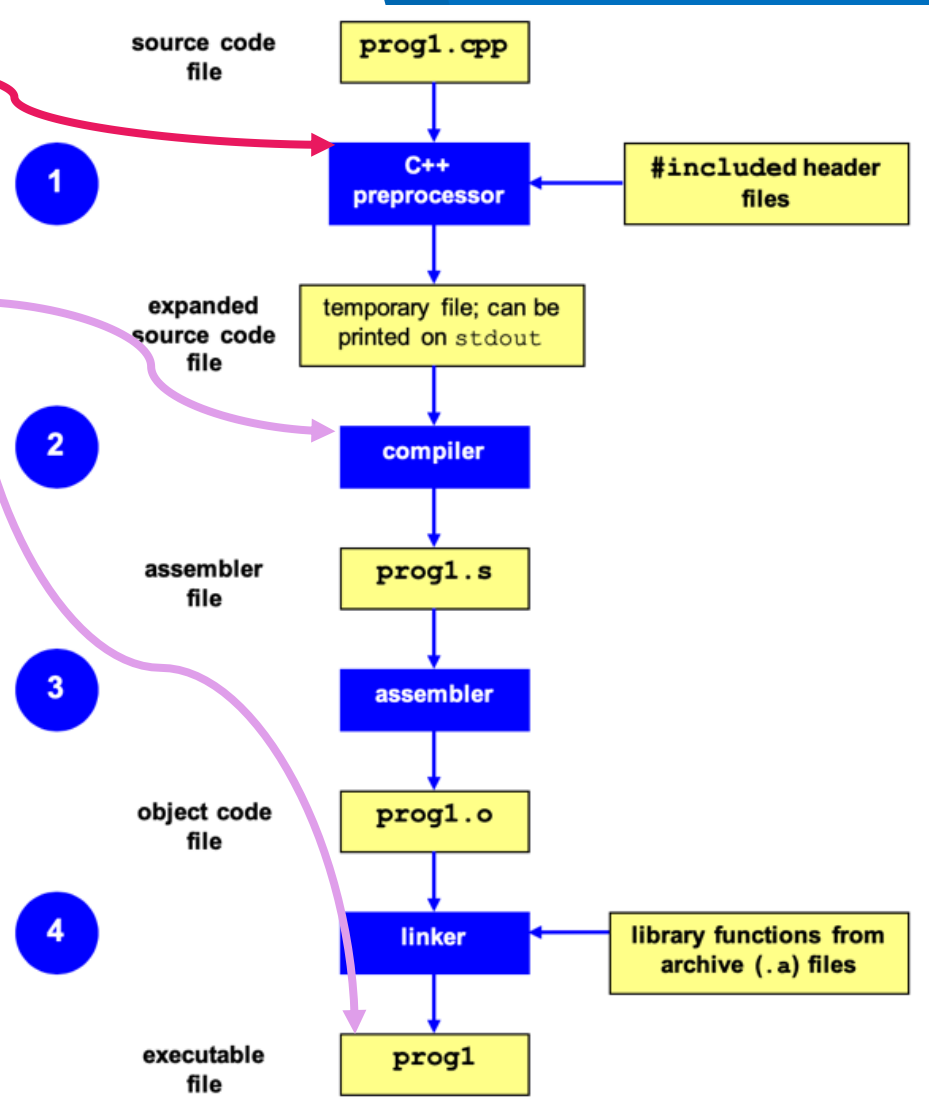
Variadikus függvények és a fordítási folyamat

- Variadikus makrók: a preprocesszálás alatt

```
#define PP_LOG(...) \
  ::print_line(__FILE__,__LINE__,__function,## __VA_ARGS__)
```

- Variadikus függvények: fordítás + kvázi „runtime támogatás”

```
#include <stdarg.h>
int add(size_t numargs, ...)
{
    va_list vl;
    va_start(vl, numargs);
    int accu = 0;
    for (size_t i = 0; i < numargs; ++i)
        accu += va_arg(vl, int);
    va_end(vl);
    return accu;
}
```



forrás:http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/Images/build.png

Variadikus függvények és a fordítási folyamat

- Variadikus makrók: a preprocesszálás alatt

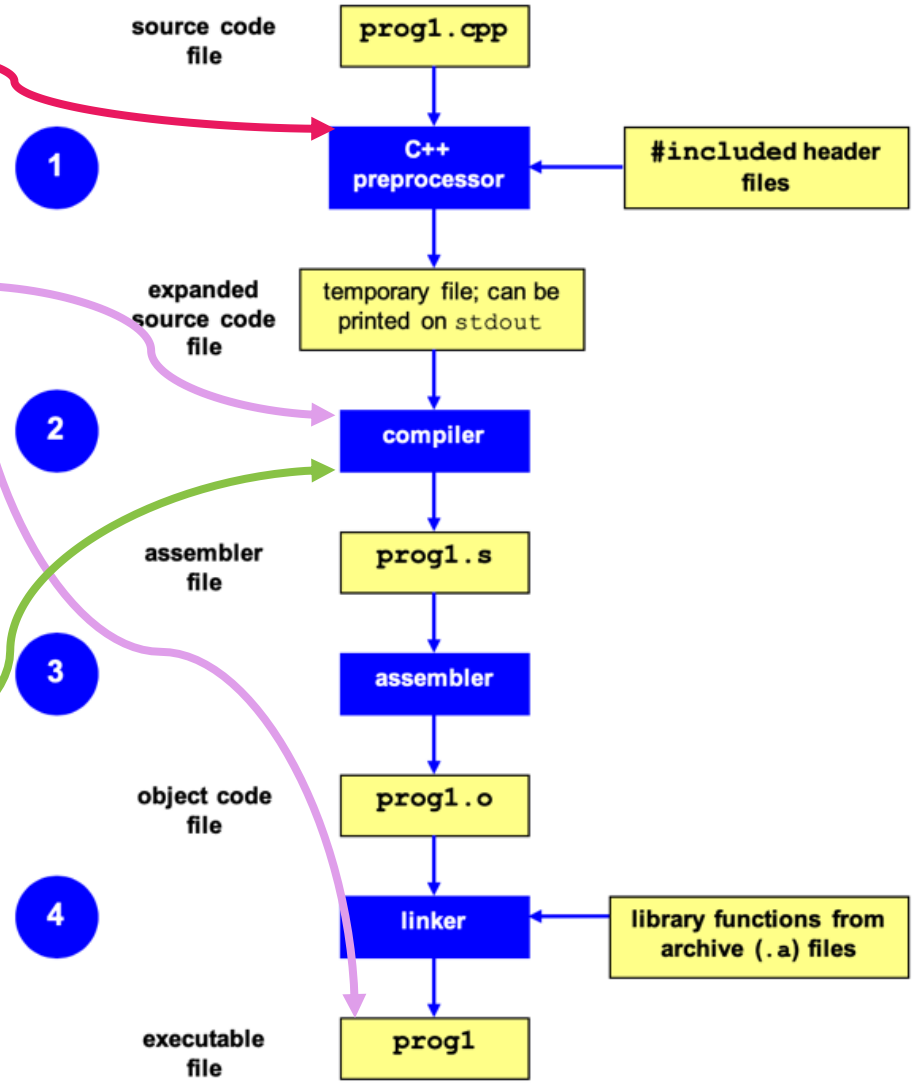
```
#define PP_LOG(...) \
  ::print_line(__FILE__,__LINE__,__function,## __VA_ARGS__)
```

- Variadikus függvények: fordítás + kvázi „runtime támogatás”

```
#include <stdarg.h>
int add(size_t numargs, ...)
{
    va_list vl;
    va_start(vl, numargs);
    int accu = 0;
    for (size_t i = 0; i < numargs; ++i)
        accu += va_arg(vl, int);
    va_end(vl);
    return accu;
}
```

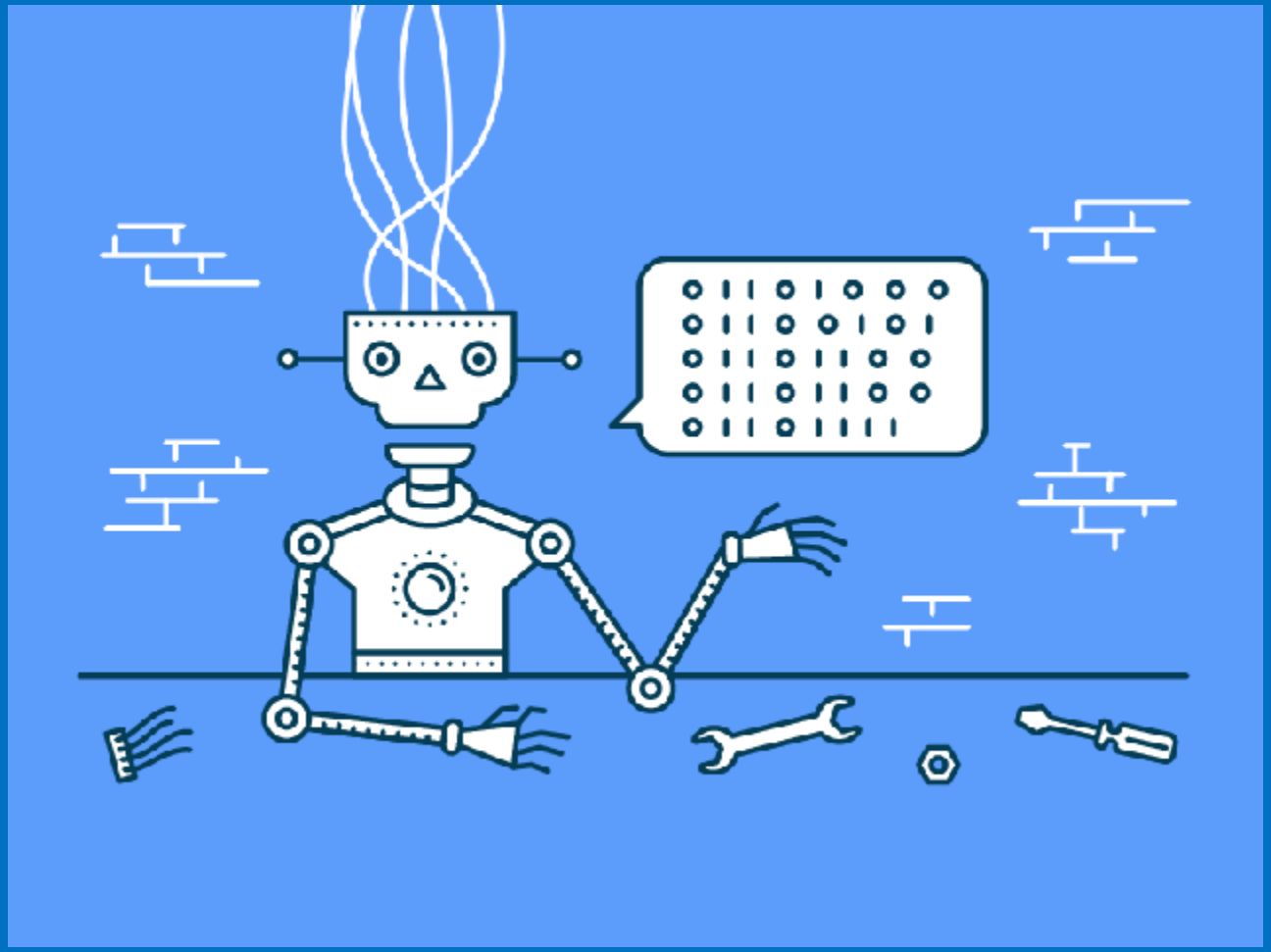
- Variadikus sablonok: csak fordítási időben

```
template<typename T,typename ... Ts>
auto add(T arg, Ts ... args) {
  return (arg + ... + args);
}
```



forrás:http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/Images/build.png

ex_0: variadikus függvények



Variadikus sablonok „anatómiája”

- ‘...’ , ellipszis „ placeholder” jelöli a változó típus és argumentum listát
  - változó hosszúságú template parméter lista

```
template<typename ... Ts> /*...*/
```

  - *non-type* template paraméter helyén is állhat, pl.:

```
template<bool ... Booleans> struct all{/*...*/};
```

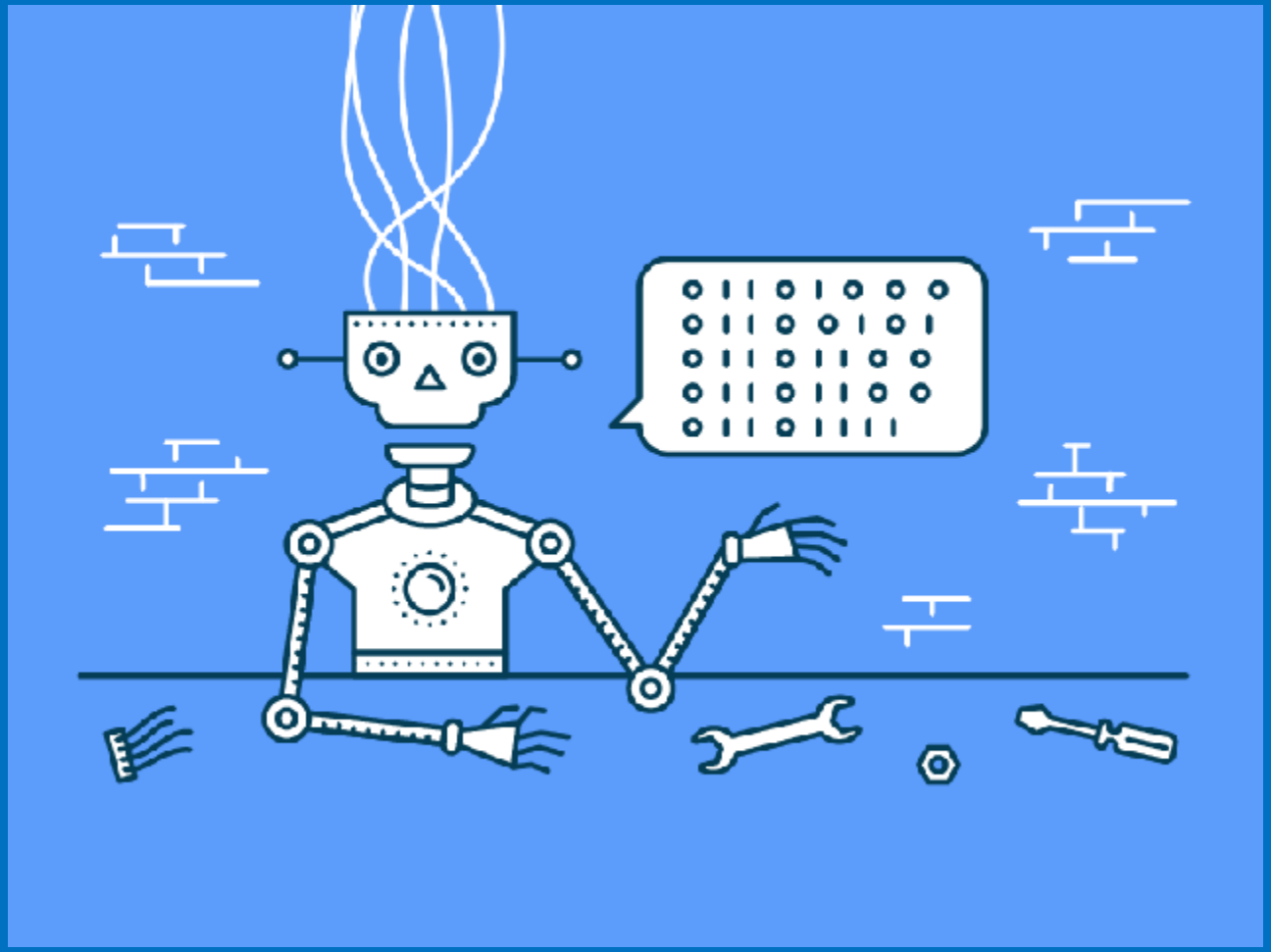
  - füffvény szignatúrájában a *parameter-pack*-ot jelöli

```
{ args... ;}
```

- a *parameter-pack*-ot expandálni kell használat előtt az ellipszis operátorral!

```
auto my_func(Ts const & ... args){/*...*/}
```

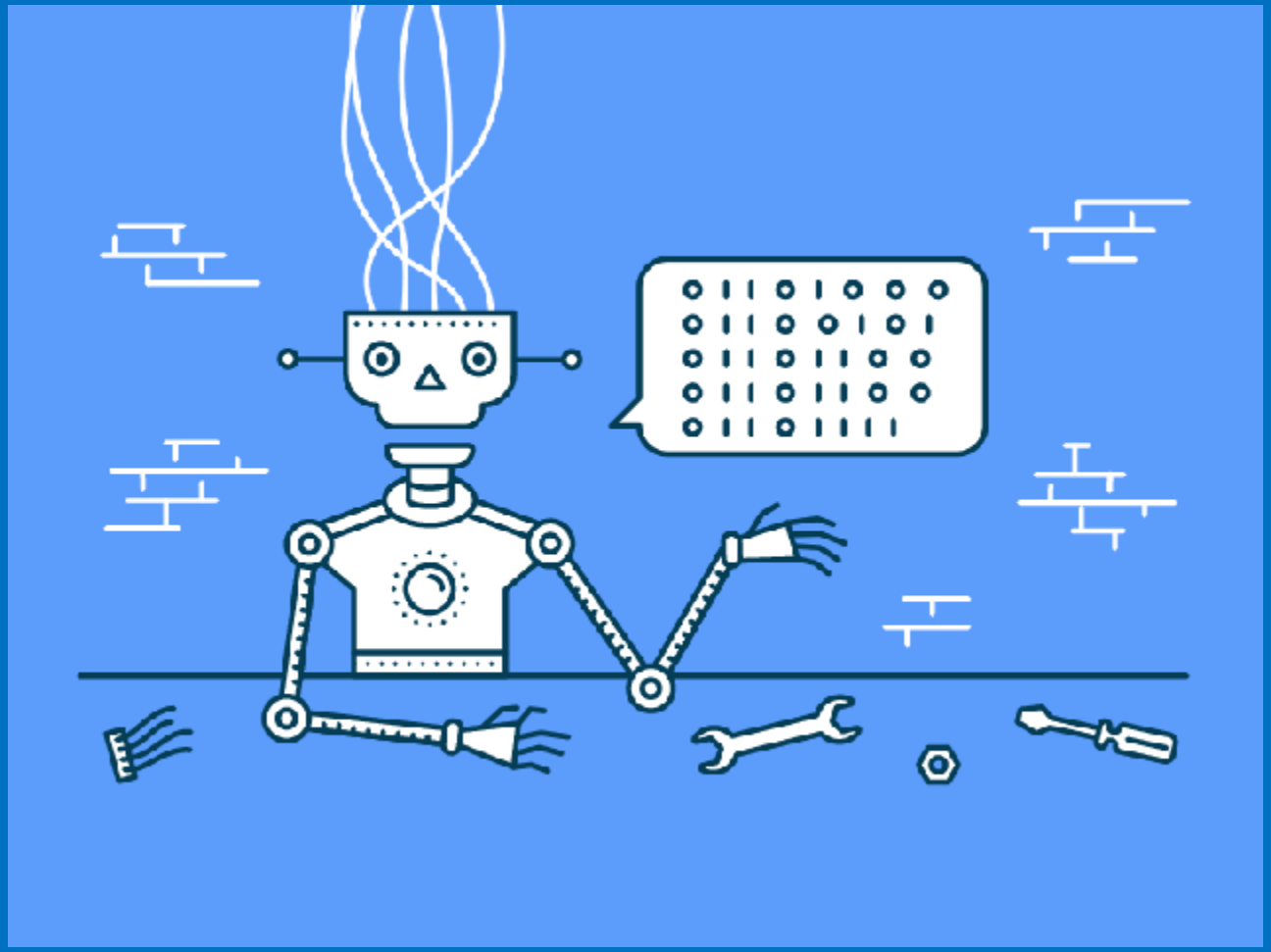
ex_1: variadikus sablonok



Variadikus sablonok „anatómiája” (folytatás)

- Egymásba ágyazott expanzió is lehetséges, (belülről-kifelé asszociatív)
  - ```
    template <typename... Funcs>
    struct InvokeAll {
        template <typename... Ts>
        void operator()(Ts&&... args) const
        {
            (Funcs {}(std::forward<Ts>(args)...), ...);
        }
    };
    ```
    - Vigyázat!!!: *std::forward<>* nem mindig kívánatos, ha a paramétereket iterációban *forward<>*-oljuk!
    ```

ex_2: rekurzió elkerülése



Pattern-matching

Pattern-matching I.

- Számos függvény intuitívan és egyszerűen definiálható *pattern-matching* („mintaillesztés”) segítségével.
- Szintaktikai kifejezések sorozata – minták – és az ezekhez tartozó eredmények halmaza
- Ha az n-ik minta illeszkedik akkor a kifejezés értéke az a n-ik eredmény
  - pl.: Haskell-ben a logikai negálás következőképpen definiálható:

```
data Bool = True | False
not :: Bool -> Bool
not True = False
not False = True
```

- C++-ban lehetőségek *pattern-matching*-re:
  - overload resolution
  - template specializáció

Pattern-matching II.

- negálás példa overload resolution-nel:

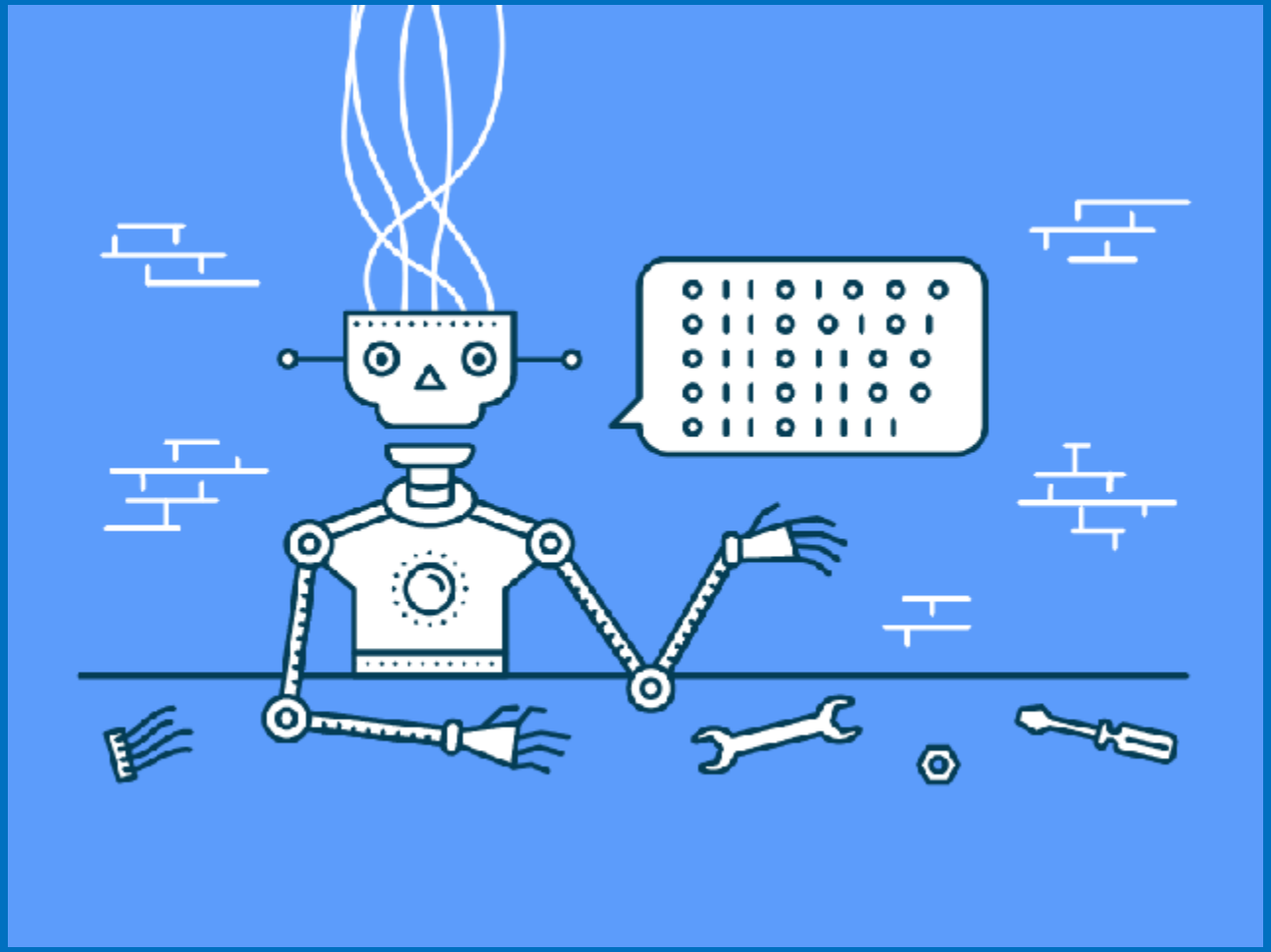
```
#include <type_traits>
std::true_type not(std::false_type){}
std::false_type not(std::true_type){}
```

- negálás template specializációval
  - jelenleg nem lehetséges a függvények parciális specializációja
  - Ilyen esetekben éredemes mindig osztályokat használni

```
#include <type_traits>
template<typename T> struct not;
template<> struct not<std::true_type>{ using type = std::false_type;};
template<> struct not<std::false_type>{ using type = std::true_type;};
template<typename T> using not_t = typename not<T>::type;
```

- **Kulcs:** A típusrendszerbe van „kódolva” hogy mi a viselkedés, deklaratívan
- Az utóbbi megoldás általában generikusabb, könnyebben bővíthető.

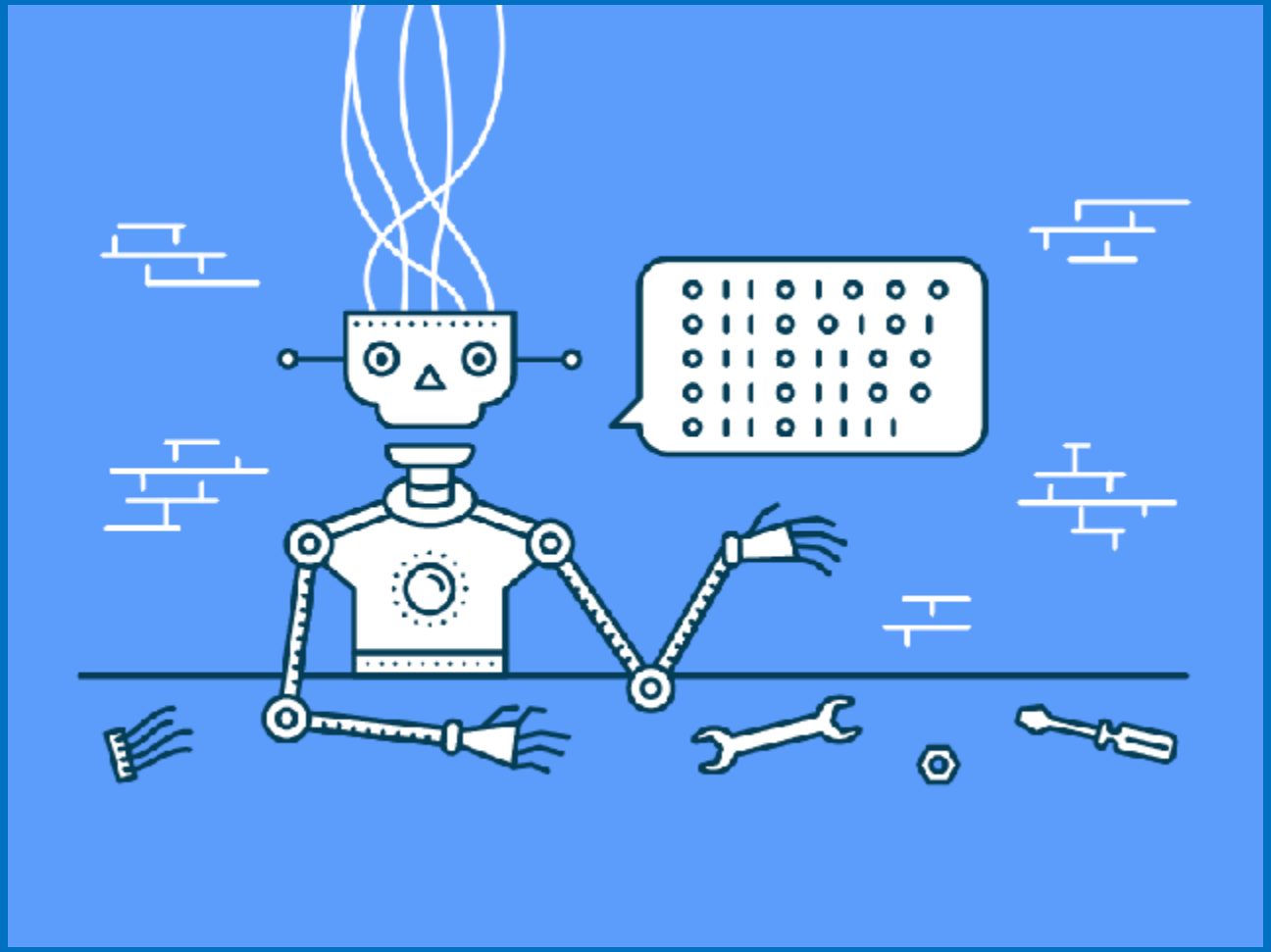
ex_3: sablonok specializációja



SFINAE (Substitution Failure Is Not An Error)

- A C++ *overload resoultion* egy bonyolult folyamat
- Ha az overload resolution közben a compiler fgv. template-et lát, először dedukálnia kell a template típus paramétereket, és be kell helyettesítenie azt a szignaturába, hogy az bekerülhessen az *overload set*-be
- Ha compiler hibára fut eközben a behelyettesítés közbe, az nem számít fordítási hibának, egyszerűen az a specializáció figyelmen kívűl hagyódik
- Ezt a mechanizmust még szofisztikáltabb mintaillesztésre is fel lehet használni (többek között)
- Expression SFINAE: pl. *decltype()* vagy *sizeof()* argumentumában, nem kiértékelt operandus esetén (C++20 előtt egyszerűbb *Concept* emulációra is használható)
- Leggyakoribb módja : std::enable_if<bool,T>
- SFINAE-vel lehet még pl.: tagfüggvényeket, tagváltozokat, stb. detektálni

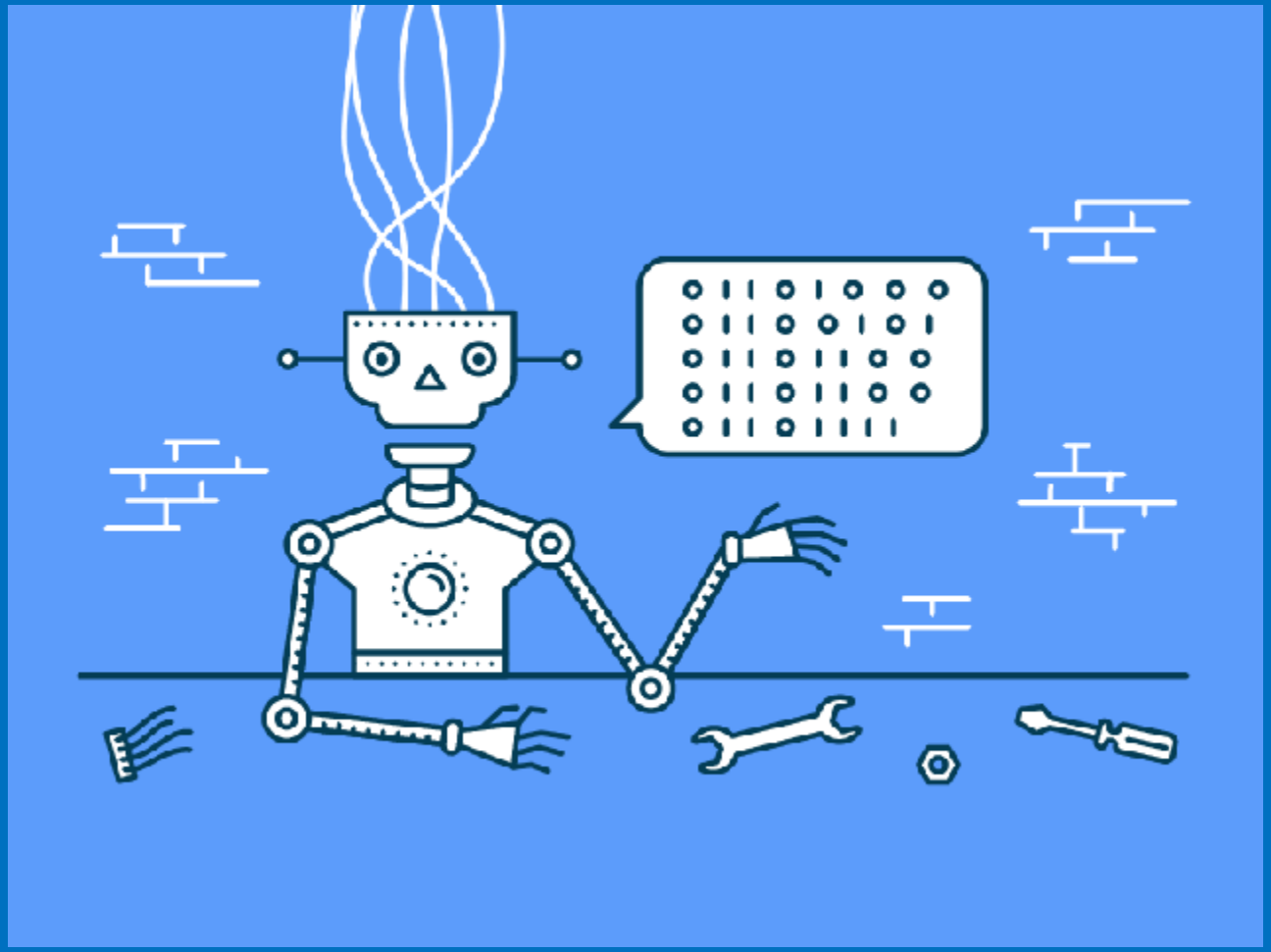
ex_4: SFINAE



Tag dispatch

- Valamely komplexebb típusfeltétel alapján történő mintaillesztés, amely egyszerre alkalmaz több technikát
  - overload resolution
    - az egyes minták függvények paraméterlistával
    - kiegészítve úgynevezett *„tag”*-ekkel
  - általában SFINAE technikával kiválasztjuk a hívás helyén a *tag*-et, majd az overload resolution-re bízzuk a mintaillesztést
- Ha függvény implementációk közül akarunk választani,és van alapértelmezett viselkedés, amire tudunk fallback-elni, akkor többnyire ezt a technikát éredemes alkalmazni
- *constexpr-if* : teljesen kiváltható vele, de olvashatóság, karbantarthatóság szempontjából még ma is releváns (megj.: ez is növeli a ciklomatikus komplexitást)

ex_5: Tag dispatch



Köszönöm a figyelmet!

Folytatjuk…