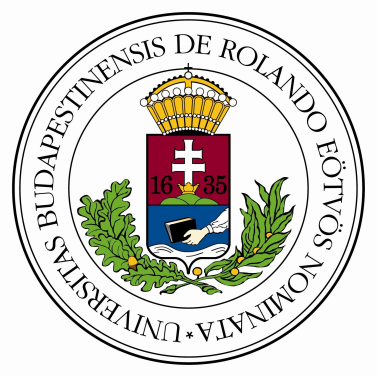
Imperatív programozás

Áttekintés

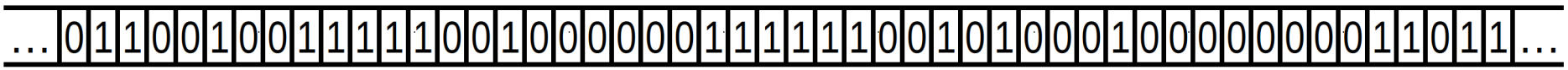


Kozsik Tamás és mások

ELTE Eötvös Loránd Tudományegyetem

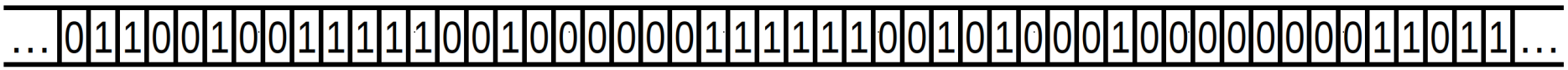
Imperatív programozás

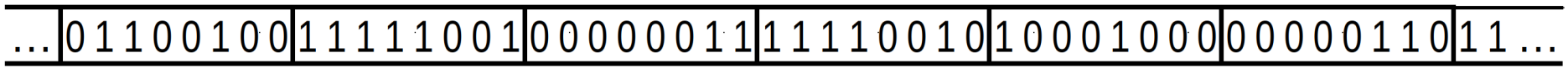
- Utasítások, vezérlési szerkezetek
- Memória írása, olvasása
- C programozási nyelv (link!)



Imperatív programozás

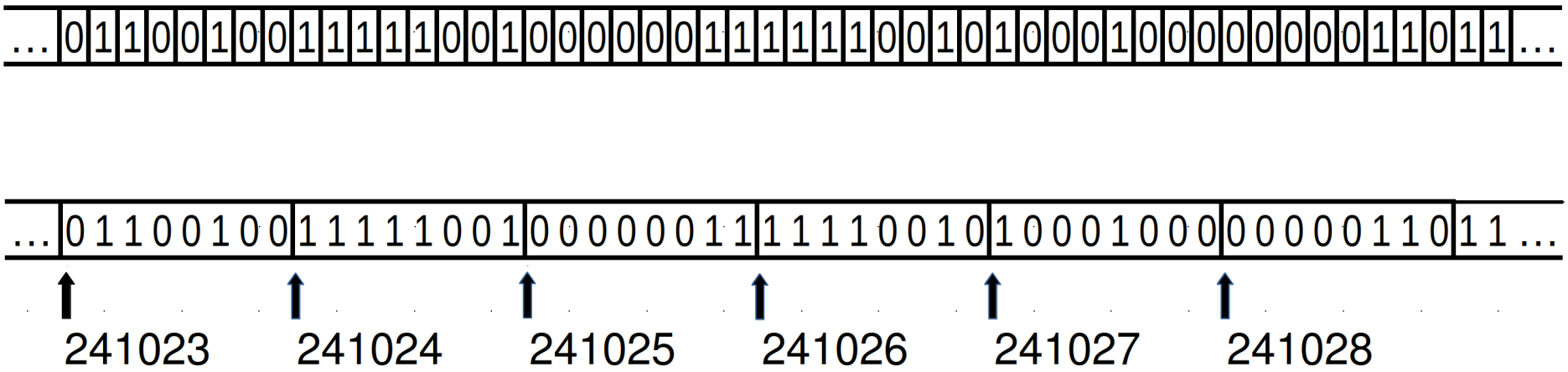
- Utasítások, vezérlési szerkezetek
- Memória írása, olvasása
- C programozási nyelv (link!)





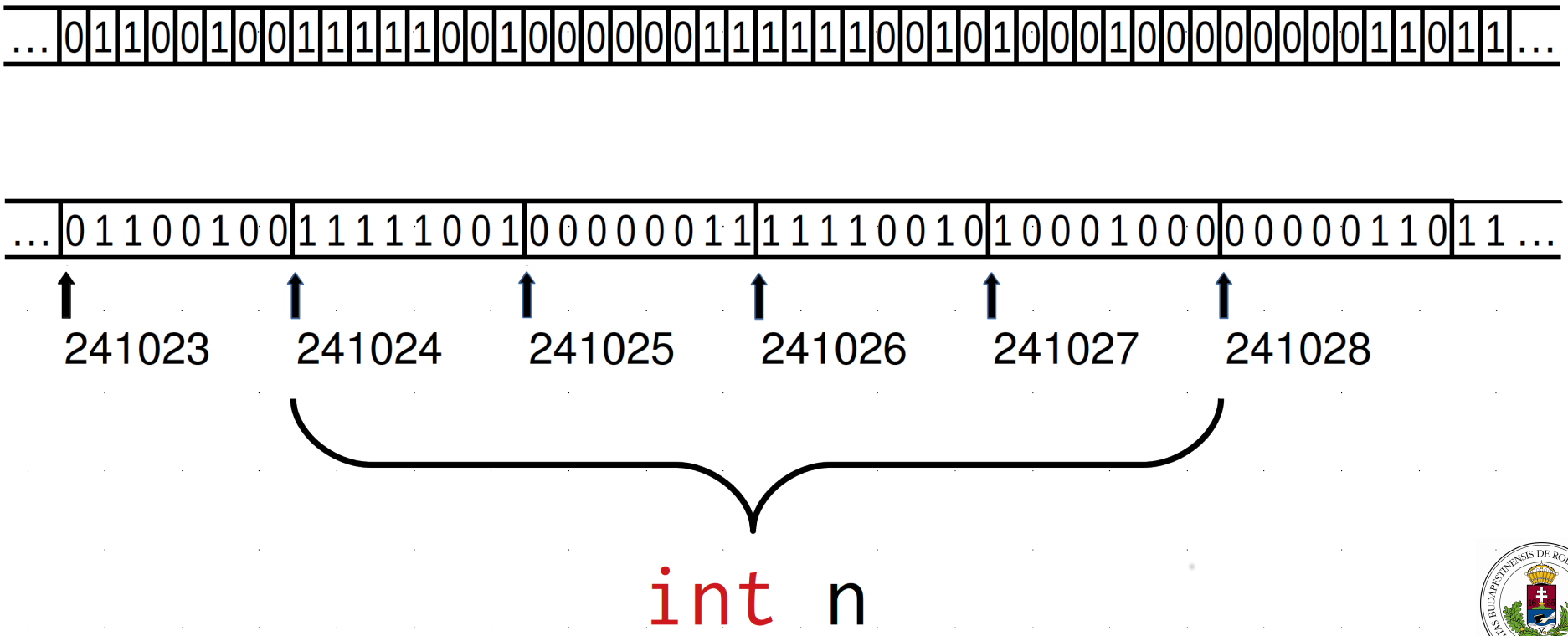
Imperatív programozás

- Utasítások, vezérlési szerkezetek
- Memória írása, olvasása
- C programozási nyelv (link!)



Imperatív programozás

- Utasítások, vezérlési szerkezetek
- Memória írása, olvasása
- C programozási nyelv (link!)



Outline

Programok felépítése

- Kulcsszavak, literálok, operátorok, egyéb jelek, azonosítók
- Kifejezések
- Utasítások
- Alprogramok (függvények/eljárások, rutinok, metódusok)
- Modulok (könyvtárak, osztályok, csomagok)

Példa

Python

```
def factorial(n):
    result = 1
    for i in range(2,n+1):
        result *= i
    return result
```

Példa

Python

```python
def factorial(n):
    result = 1
    for i in range(2,n+1):
        result *= i
    return result
```

C

```c
int factorial( int n )
{
    int result = 1;
    int i;
    for(i=2; i<=n; ++i)
    {
        result *= i;
    }
    return result;
}
```

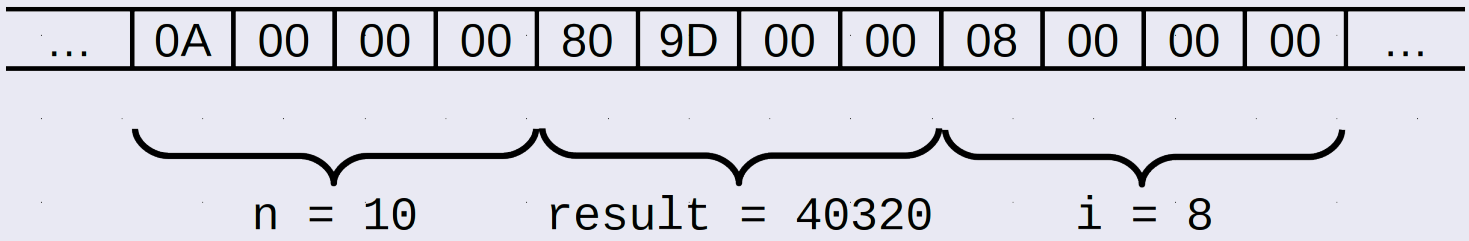
Példa

Python

```python
def factorial(n):
    result = 1
    for i in range(2,n+1):
        result *= i
    return result
```

C

```c
int factorial( int n )
{
    int result = 1;
    int i;
    for(i=2; i<=n; ++i)
    {
        result *= i;
    }
    return result;
}
```



Kifejezések

n

"Hello world!"

100

n+1

++i

range(2,n+1)

employees[factorial(3)].salary * 100

Utasítások

```
result = 1;
```

```
        result *= i;
```

```
                                    return result;
```

```
for( i=2; i<=n; ++i ){ result *= i; }
```

```
                        while(1) printf("Gyurrrika szép!\n");
```

Egyszerű utasítások

- értékadás
- üres utasítás
- alprogramhívás
- visszatérés függvényből

Vezérlési szerkezetek

- elágazások
- ciklusok stb.

Python

```
def gcd(n,m):
    while n != m:
        if n > m:
            n -= m
        else:
            m -= n
    return n
```

C

```
int gcd( int n, int m )
{
    while( n != m )
        if( n > m )
            n -= m;
        else
            m -= n;
    return n;
}
```

Kapcsos zárójelek vezérlési szerkezetekben

Elhagyott kapcsos zárójelek

```
int gcd( int n, int m )
{
    while( n != m )
        if( n > m )
            n -= m;
        else
            m -= n;
    return n;
}
```

Bolondbiztos megoldás

```
int gcd( int n, int m )
{
    while( n != m ){
        if( n > m ){
            n -= m;
        } else {
            m -= n;
        }
    }
    return n;
}
```

Csellengő else (dangling else)

Ezt írtam

```
if( x > 0 )
    if( y != 0 )
        y = 0;
else
    x = 0;
```

Ezt jelenti

```
if( x > 0 )
    if( y != 0 )
        y = 0;
    else
        x = 0;
```

Ezt akartam

```
if( x > 0 ){
    if( y != 0 )
        y = 0;
} else
    x = 0;
```

Lásd még...

goto-fail (Apple) link!

Kiírás a szabványos kimenetre

Kiírunk egy egész számot és egy soremelést (*newline*)

Python

```
print( factorial(10) )
```

C

```
printf("%d\n",factorial(10));
```

Bonyolultabb kiírás

Python

```
print( "10! =", factorial(10), ", ln(10) =", log(10) )
```

C

```
printf("10! = %d, ln(10) = %f\n", factorial(10), log(10));
```

Bonyolultabb kiírás

Python

```
print( "10! =", factorial(10), ", ln(10) =", log(10) )
```

C

```
printf("10! = %d, ln(10) = %f\n", factorial(10), log(10));
```

```
10! = 3628800, ln(10) = 2.302585
```

Típusok

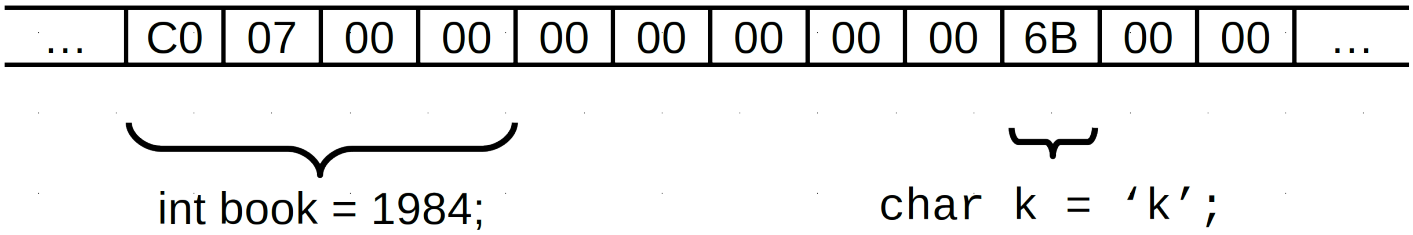
- Kifejezik egy bitsorozat értelmezési módját
- Meghatározzák, milyen értéket vehet fel egy változó
- Megkötik, hogy műveleteket milyen értékekkel végezhetünk el

C-ben

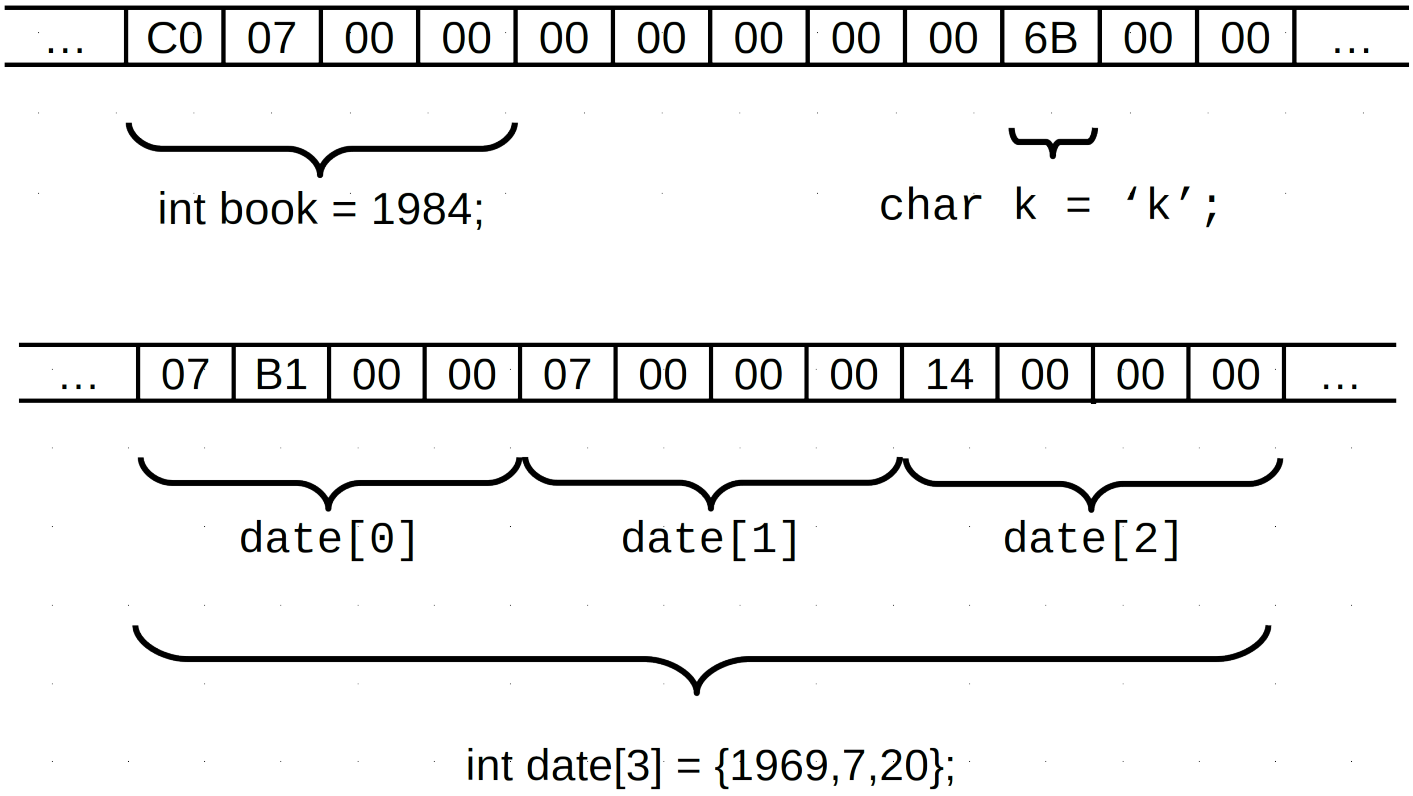
- `int` – egész számok egy intervalluma, pl. $[(-1) * 2^{31} \,..\, 2^{31} - 1]$
- `float` – racionális számok egy részhalmaza
- `char` – karakterek
- `char[]` – szövegek, karakterek tömbje
- `int[]`– egész számok tömbje
- `int*` – mutató (pointer) egy egész számra

stb.

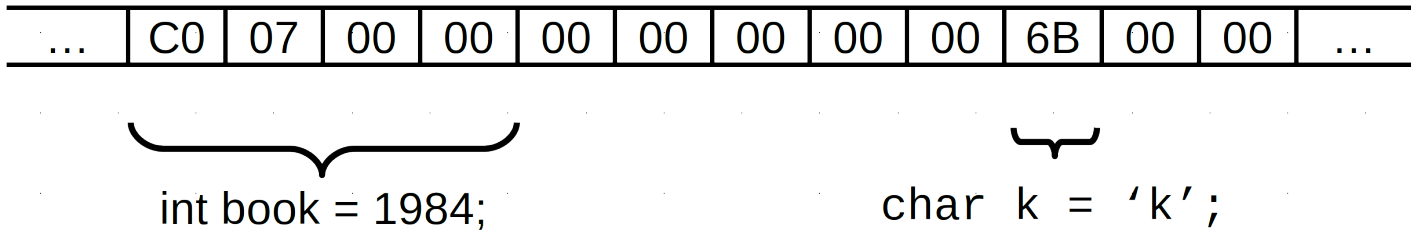
Különböző típusú értékek a memóriában

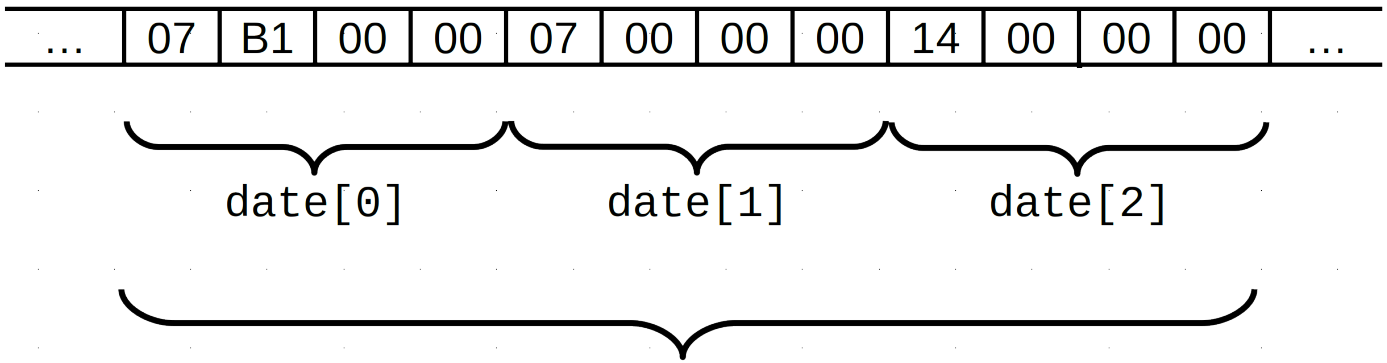


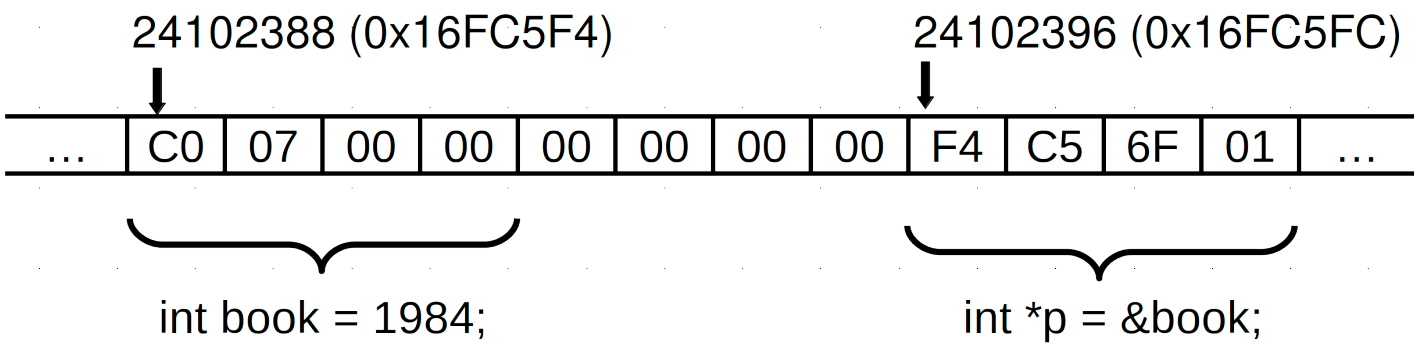
Különböző típusú értékek a memóriában



Különböző típusú értékek a memóriában







Típus szerepe

- Védelem a programozói hibákkal szemben
- Kifejezik a programozók gondolatát
- Segítik az absztrakciók kialakítását
- Segítik a hatékony kód generálását

Típusellenőrzés

- A változókat, függvényeket a típusuknak megfelelően használtuk-e
- A nem típushelyes programok értelmetlenek

Statikus és dinamikus típusrendszer

- A C fordító ellenőrzi *fordítási időben* a típushelyességet
- Pythonban *futási időben* az interpreter vizsgálja ezt

Erősen és gyengén típusos nyelv

- Gyengén típusos nyelvben automatikusan konvertálódnak értékek más típusúra, ha kell
  - Eleinte kényelmes
  - De könnyen írunk mást, mint amit szerettünk volna
- A C-ben és Pythonban viszonylag szigorúak a szabályok (erősen típusosak)

Alprogramok (subprograms)

- Több lépésből álló számítás leírása
- Általános, paraméterezhető, újrafelhasználható
- A program strukturálása – komplexitás kezelése
  - egy képernyőoldalnál ne legyen hosszabb
- Különböző neveken illetik
  - rutin (routine vagy subroutine)
  - függvény (function): kiszámol egy értéket és "visszaadja''
  - eljárás (procedure): megváltoztathatja a program állapotát
  - metódus: objektum-orientált programozási terminológia

Főprogram (main program)

Ahol a program végrehajtása elkezdődik

Python

Nincs jelölve, egy csomó utasítás egymás után

```
half = 21
print(2*half)
```

C

Egy megfelelő nevű alprogram: `main`

```
int main()
{
    int half = 21;
    printf("%d\n",2*half);
    return 0;                   /* sikeres végrehajtás */
}
```

Megjegyzés

[...]

Python

[...]

```
half = 21
print(2*half)                    # itt így írok megjegyzést
```

C

[...]

```
int main()
{
    int half = 21;
    printf("%d\n",2*half);
    return 0;                    /* itt így írok megjegyzést */
}
```

Modul

Modularitás: egységbe zárás, függetlenség, szűk interfészek

- Újrafelhasználható programkönyvtárak
  - pl. a nyelv szabványos könyvtára (standard library)
- A program nagyobb egységei
- Absztrakciók megvalósítása

Modulokra bontás

Újrafelhasználható `factorial`

- `factorial.c` – a `factorial` függvényt
- `tenfactorial.c` – a főprogramot

tenfactorial.c

```c
#include <stdio.h>

int factorial( int n );  /* deklaráljuk ezt a függvényt */

int main()
{
    printf("%d\n", factorial(10));
    return 0;
}
```

Outline

Forráskód

- Programozási nyelven írt kód
- Számítógép: gépi kód
- Végrehajtás
  - interpretálás (Python)
  - fordítás, futtatás (C)
- Forrásfájl
  - `factorial.py`
  - `factorial.c`

Forrásfájl Pythonban

factorial.py

```
def factorial(n):
    result = 1
    for i in range(2,n+1):
        result *= i
    return result

print(factorial(10))
```

Végrehajtás

```
python3 factorial.py
```

Parancsértelmező (interpreter)

- Például python3
- Forráskód feldolgozása utasításonként
  - Ha hibás az utasítás, hibajelzés
  - Ha rendben van, végrehajtás
- Az utasítás végrehajtása: beépített gépi kód alapján

Hátrányok

- Futási hiba, ha rossz a program (ritkán végrehajtott utasítás???)
- Lassabb programvégrehajtás

Előnyök

- Programírás és -végrehajtás integrációja
  - REPL = Read-Evaluate-Print-Loop
  - Prototípus készítése gyorsan
- Kezdők könnyebben elsajátítják

Forrásfájl C-ben

factorial.c

```c
#include <stdio.h>

int factorial( int n ){
    int result = 1;
    int i;
    for(i=2; i<=n; ++i){
        result *= i;
    }
    return result;
}

int main(){
    printf("%d\n", factorial(10));
    return 0;
}
```

Fordítás és futtatás szétválasztása

- Sok programozási hiba kideríthető a program futtatása nélkül is
- Előre megvizsgáljuk a programot
- Ezt csak egyszer kell (a *fordítás* során)
- Futás közben kevesebb hiba jön elő
- Cél: hatékony és megbízható gépi kód!

“Fordítási idő” és “futási idő”

Fordítás és futtatás

Forrásfájl

```
factorial.c
```

Fordítás

```
gcc factorial.c
```

Lefordított program

```
a.out
```

Futtatás

```
./a.out
```

A gcc -o kapcsolója

Forrásfájl

```
factorial.c
```

Fordítás

```
gcc -o fact factorial.c
```

Lefordított program

```
fact
```

Futtatás

```
./fact
```

Fordítási hibák

- Nyelv szabályainak megsértése
- Fordítóprogram detektálja

factorial.c-ben

```
int factorial( int n )
{
    int result = 1;
    for(i=2; i<=n; ++i)
    {
        result *= i;
    }
    return result;
}
```

gcc factorial.c

```
factorial.c: In function ‘factorial’:
factorial.c:6:9: error: i undeclared (first use in this function)
     for(i=2; i<=n; ++i)
         ^
```

Fordítási figyelmeztetések

- Nyelv szabályai betartva
- A fordítóprogram valamilyen furcsaságot detektál
- Valószínűleg hibát vétettünk
- **Cél: warning-mentes fordítás!**

factorial.c-ben

```
int factorial( int n )
{
    int result = 1, i;
    for(i=2; i<=n; ++i)
    {
        result *= i;
    }
}
```

gcc -W -Wall --pedantic factorial.c

```
factorial.c: In function ‘factorial’:
factorial.c:10:1: warning: control reaches end of non-void function
                                                    [-Wreturn-type]
 }
 ^
```

Előfeldolgozás

C preprocessor: (forráskódból) forráskódot állít elő

Makrók

```
#define WIDTH 80
...
char line[WIDTH];
```

Deklarációk megosztása

```
#include <stdio.h>
...
printf("Hello world!\n");
```

Feltételes fordítás

```
#ifdef FRENCH
printf("Salut!\n");
#else
printf("Hello!\n");
#endif
```