

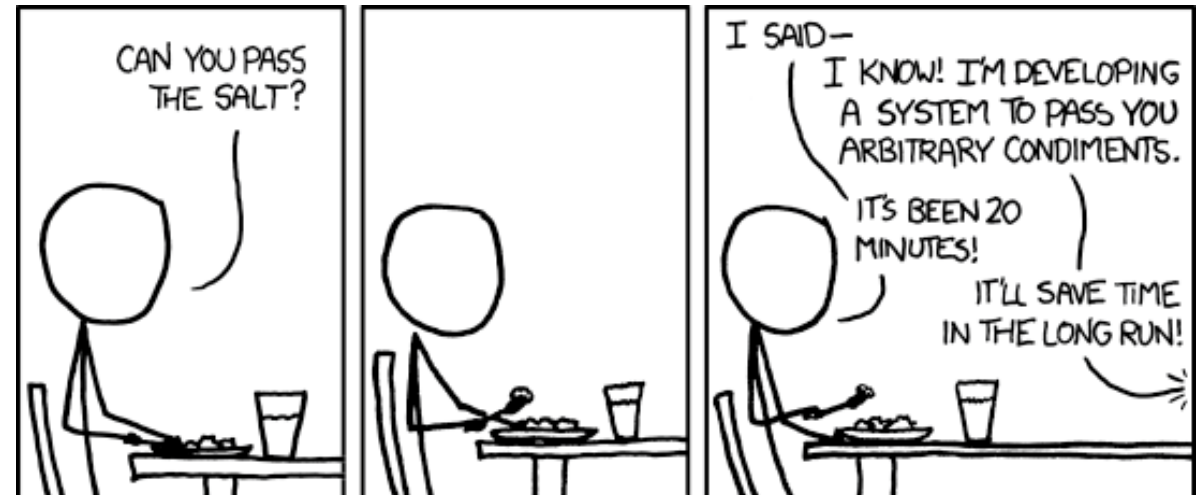
Debugolható programok

LEGO Kör robotika tanfolyam

„Clean code” alapvetések

Avagy hogyan írjunk könnyen debugolható kódot

- Törekedjünk az egyszerűsége
 - „KISS – Keep It Simple and Stupid”
 - A legegyszerűbb működő megoldás szinte mindig a legjobb is
 - A lehető legkevesebb dolog tud elromlani
- Konzisztencia: egyféleképpen csináljuk, egy mód legyen mindenre
- Találjuk meg a felmerült hibák gyökerét

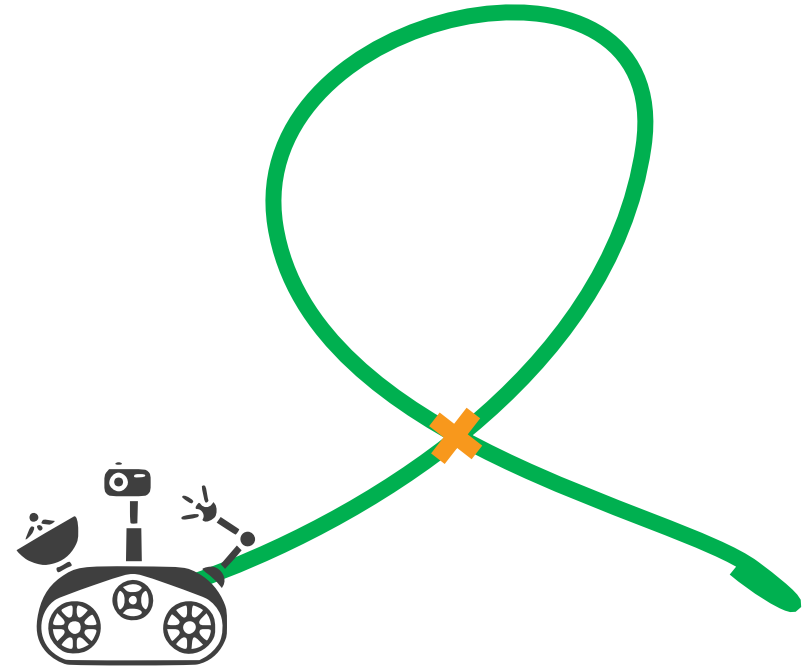


Túlbonyolítás...

```
task feladat1() {
  int corr=0;
  while(/*Sensor(SEN_COL)!=INPUT_GREENCOLOR*/true) {
    while(Sensor(SEN_COL)==KERESZTCOL) {
      OnFwd(OUT_AB,F1_SPD);
      Wait(700);
    }
    ClearScreen();
    NumOut(0,LCD_LINE1,corr);
    while(Sensor(SEN_COL)==INPUT_BLACKCOLOR||corr>400) {
      OnFwd(OUT_A,F1_SPD);

      if(Sensor(SEN_COL)==INPUT_BLACKCOLOR) {corr=0;}
      if(corr>400) {OnFwd(OUT_B,F1_SPD*-1);} else {OnFwd(OUT_B,F1_SPD/4);}
    }


    OnFwd(OUT_B,F1_SPD);
    OnFwd(OUT_A,F1_SPD/3);
    Wait(1);
    corr++;
    /*corr+=F1_CORR;
    OnFwd(OUT_A,F1_SPD+corr);
    OnFwd(OUT_B,F1_SPD-corr);
    Wait(100);*/
  }
}
```




Fontos „apróságok”

Változódeklaráció

```
int main(void) {  
    int n, sz, i;  
  
    scanf("%d", &n);  
    sz = 1;  
    for (i = 1; i <= n; i += 1)  
        sz *= i;  
    printf("%d! = %d\n", n, sz);  
  
    return 0;  
}
```



```
int main(void) {  
    int n;  
    scanf("%d", &n);  
  
    int sz = 1;  
    for (int i = 1; i <= n; i += 1)  
        sz *= i;  
  
    printf("%d! = %d\n", n, sz);  
  
    return 0;  
}
```



Változónevek

```
int t;  
int et; //elapsed time in hours
```



```
int elapsedTime;  
int elapsedHours;
```



Változónevek

```
bool sensorDisabled = true;  
...  
if (sensorDisabled) //?!?
```



```
bool sensorEnabled = false;  
...  
if (sensorDisabled)
```



Mágikus számok

```
OnFwdSync(OUT_AB, 60, 0);
```

```
OnFwdSync(OUT_AB, 60, 0);
```

```
#define SPEED 60
```

```
...
```

```
OnFwdSync(OUT_AB, SPEED, 0);
```

```
OnFwdSync(OUT_AB, SPEED, 0);
```

Függvények: hossz

LEGO

KÖR



Függvények: feladat



Függvények: név

```
void doTask3(void) {  
    // ...  
}
```



```
void goAroundBox(void) {  
    // ...  
}
```



Függvények: paraméterek

```
int calcGradeForStudent(const char* name, int  
midterm1grade, int midterm2grade, int ...
```

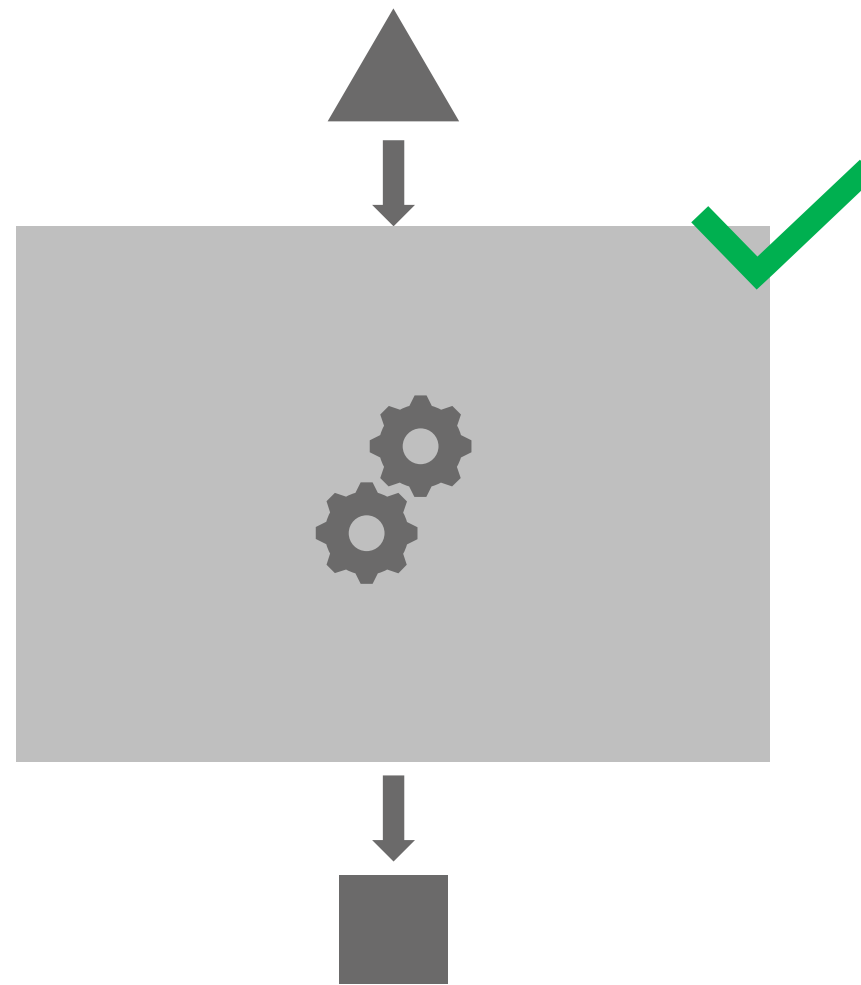
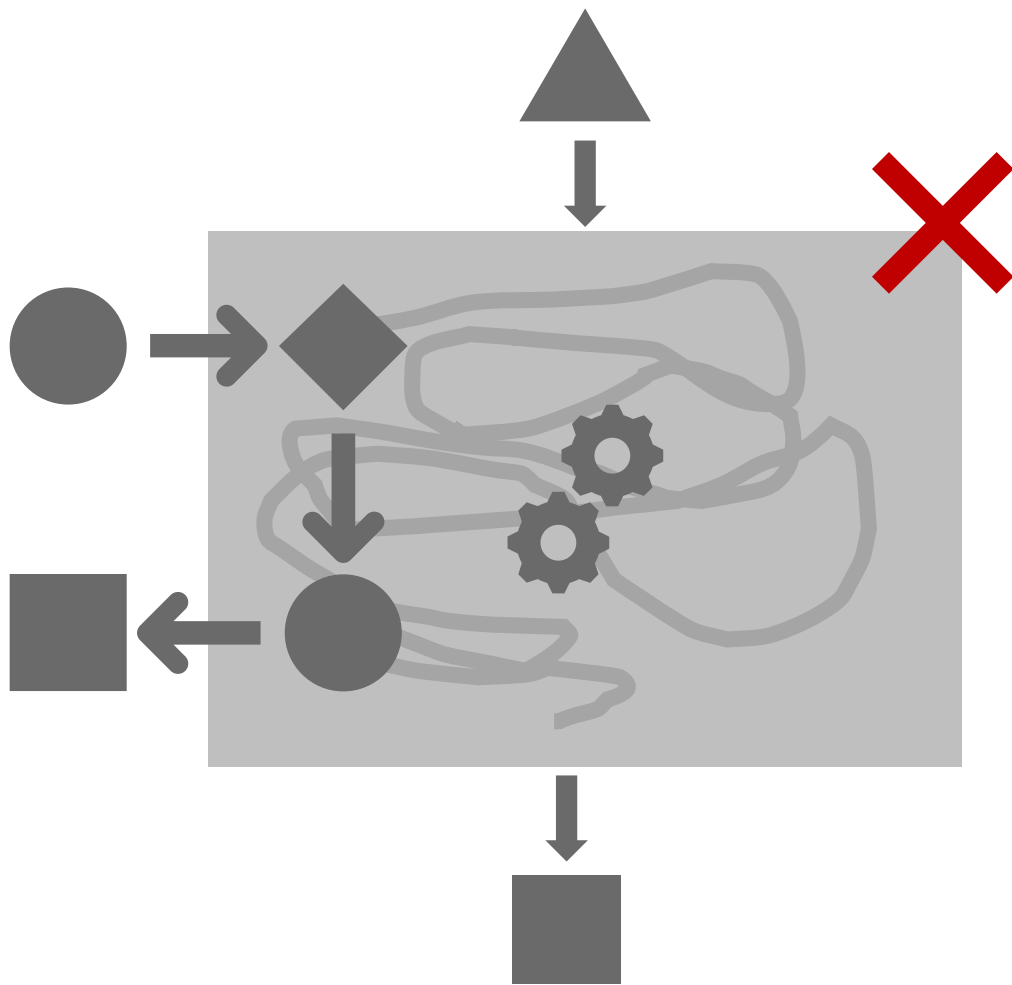


```
struct Student {  
    char* name;  
    int midterm1grade;  
    // ...  
};
```



```
int calcGradeForStudent(Student student);
```

Függvények: mellékhatás



- Rövid
- Egy dolgot csinál
- Beszédes neve van
- Nincs rengeteg paramétere
- Nincs mellékhatása
 - Legjobb: „pure function”
- Nincs „flag” paramétere, helyette több külön függvény
 - ALU analógia rossz
 - Nem beállítható multifunkciós szerszám gép, hanem célszerzám

- Legjobb eset: önmagyarázó kód
 - Lehetővé teszi: beszédes függvénynevek, beszédes változónevek
- Ha mégis kell komment:
 - Nyilvánvaló (pl. fv. névből látszó szinten) dolgot nem kommentelünk
 - Nem mit, hanem hogy miért
 - Félreértések megelőzésére
 - Figyelemfelhívásra

Egy NXT-s program írása

Egy NXT-s program írása

- Értelmezd a feladatot
- Bontsd a feladatot részfeladatokra
 - Állapotgép?
- Valósítsd meg a részfeladatokat (→ függvény?)
 - Képzeld magad a rover helyébe!
 - Folyamatosan próbáld ki, amit csinálsz
 - Nem jó, ha egyszerre mindent megírsz, majd csodálkozol, hogy nem megy
- A fizikai világ nem tökéletes
 - Szenzor zajos, nem veszi észre a szint, ...

Ha valami nem megy

- Használd az elérhető módszereket az állapotjelzésre!
 - Írd ki a változót LCD-re, sípolj a függvény végén, ...
- Bizonyosodj meg róla, meddig jó a kód
 - „Ez még megy, még ez is helyes, ééés itt romlik el”
- Szűkítsd a hiba helyét
 - Mi az ebben a részben, ami ilyen viselkedést okozhat?
 - Akkor is ezt csinálja, ha egyszerűsítem?