Dolgok, amelyekkel őrületbe kergetheted a Prog 1 tanárodat

Preprocesszor:

tedd kevésbé észrevehető helyre:

```
#define puts(a) puts(#a)
```

ezután hívd meg:

```
puts(hello world!)
```

---

tedd kevésbé észrevehető helyre:

```
#define i __LINE__
```

majd:

```
printf("%d,", i);
printf("%d,", i);
printf("%d", i);
```

ami pl.: 5, 6, 7, ha az első *printf* az ötödik sorban van.

Globális tér:

A globális változó 0-ra inicializálódik. pl.:

```
int valtozo1, valtozo2, valtozo3; // mindegyik 0-val egyenlő
```

Ha egy globális változónak / tömbnek / függvénynek nincs jelölve a típusa, akkor az int. pl.:

```
valtozo1; // azt jelenti: int valtozo = 0;
valtozo[3]; // azt jelenti: int valtozo[3] = {0, 0, 0};
f(){ return 5; } // azt jelenti: int f(int){ return 5; }
```

A pointer jelölés nyelvtanilag nem a típushoz tartozik, hanem a változóhoz ezért szintén vonatkozik rá a fenti szabály. pl.:

```
*ptr; // azt jelenti: int *ptr;
```

Ha egy függvény paramétereinek nincs jelölve a típusa,

akkor az szintén default int. pl.:

```
f(a,b){ return a + b; }
```

Ha azonban több típusú paraméter van egymás mellett,

vagy pointer típust inicializálsz, akkor (sajnos) kell explicit típusmegadás.

Attól még, hogy a K&R-féle paraméter megadás "kiment a divatból",

te még nyugodtan használhatod a kód átláthatatlanná tételére:

```
int func(a, b)
int a, b;
{
// függvény belseje
}
```

„Lássuk csak, *func* függvény deklarálva, *a*, *b* globális változók...

...mi az Isten, ez a blokk, hova tartozik?!”

A K&R és a default int szabályt ötvözve szintén nehezen értelmezhető kódot kapunk. Itt *argc*-ről tudjuk, hogy int lesz, ezért csak *argv* típusát kell meghatározni.

```
main(argc,argv)
char **argv;
{
```

Függvények:

Ha van egy rövid segédfüggvényed, mindig használj rekurziót. pl.:

```
sum_array(int *arr, int len){
    return len ? arr[--len]+sum_array(arr, len) : 0;
}
```

mennyivel szebb és nehezebben karbantarthatóbb mint ez:

```
int sum_array(int *arr, int len){
    int sum;
    int i;
    for (sum = i = 0; i < len; i++){
        sum += arr[i];
    }
    return sum;
}
```

Egyéb:

A <*condition*> ? <*if true*> : <*if false*> dolgot ismeritek.

Ha ezt használod if helyett, még nem morog senki. De ha véletlenül...

```
if (a < b){
    puts("LOL");
}
```

helyett azt írod (if):

```
a < b && (
    puts("LOL");
)
```

vagy (unless):

```
a < b || (
    puts("ROFL");
)
```

vagy egy if-else ág:

```
a < b && (
    puts("LOL");
) || (
    puts("ROFL");
)
```

...akkor ott nagy csodálkozások lesznek.

A vessző a barátunk. Ha pl. *g*-ben *f*-et meg akarjuk hívni, de rögtön vissza is akarunk térni, stílusosabb ezt egy sorban elintézni.

```
void f(){ puts("LOL"); }
int  g(){ return f();  }
```

Mivel *void* típusú a meghívandó függvény, a fordító kiírja, hogy:

> error: void value not ignored as it ought to be

Ezért azt csináljuk, hogy:

```
void f(){ puts("LOL");   }
int  g(){ return f(),5;  }
```

Így az *5* értékkel térünk vissza és a fordító nem szól.

Ha egy nagy szám literált akarsz berakni a kódodba, nehogy, ismétlem, nehogy számokat pötyögj be. Helyette karakter literálokat:

```
printf("my big number: %d", 'abc');
```

jelen esetben *'abc'* értéke 6382179, mivel úgy veszi, mintha egy integert töltenénk fel byte-onként egy számmal.
Pl. 'abc' = ('a'<<16)|('b'<<8)|'c' = (97<<16)|(98<<8)|99 = 6382179
(4 karakter a maximum mivel 4 byte egy int általában, illetve lehet hogy fordított sorrendben tölti fel a fordító. vc++ így, gcc fordítva)

Mindig használd fordítva a szögletes zárójelet. Pl.

```
int arr[] = {1, 2, 3};
1[arr]; // 2
```

Konstansok:

Használj suffix-eket, hogy olvashatatlan legyen a kód:

```
1111 == (long long int)11;
0xfalu == 250;
```

Pointerek:

Ha egy függvényben a paraméterek nem ugyanarra a helyre mutatnak, hogy megkönnyítsd a fordító dolgát, használd a restrict kulcsszót:

```
int func(int *a, int * restrict b);
```

Így a tanárnak valószínűleg utána kell olvasnia mi a francot jelent a restrict.

Változók:

Ha nem fontos egy változó utólagos értéke, vagy amúgy is inkrementálnád:

```
a + b; a++;
```

helyett

```
a+++b;
```

Akinek szerencséje van, még egy ilyet is leírhat:

```
b += 1; a + b; a += 1;
```

helyett

```
a+++ ++b;
```

Tömbök:

Egy ritkán használt ám annál izgalmasabb dolog,

a tömb kezdeti értékadásához köthető:

```
int array[] = {[0 ... 99] = 5, 0, [37] = 2};
```

az array 0-tól 99-ig fel lesz töltve az 5 értékkel.

Utána pedig a 100-as indexnél egy 0.

De közben meggondoljuk magunkat és a 37-es indexre inkább

5-ös helyett 2-est írunk.

Érdekes dolgokat lehet vele csinálni, pl.: (gcc-vel fordítva)

```
int a[] = {[0 ... 3] = ',cba', '\0cba'};
```

Ha ezt kiíratjuk, a következő szöveg lesz látható:

abc,abc,abc,abc,abc

Néha szükséged lehet egy tömbre gyorsan, névtelenül, egy sorban.

Tegyük fel, hogy *a* értéke 5, 6, 7 és te erre 5, 13, 1 értéket

akarsz visszaadni ebben a sorrendben. Mit tehetsz?

```
(int []){5, 13, 1}[a-5];
```

Ciklusok:

Ha le akarsz számolni egy ciklusban, használd ezt a rendkívül informatív alakot:

```
while (i --> 0){
```