KÓDOLÁSI TANÁCSOK

Bjarne Stroustrup, a C++ nyelv kifejlesztője, a maga alkotta programozási nyelv leírását tartalmazó könyvének bevezetőjében, több mint húsz oldalon keresztül próbálja megosztani felgyülemlett tapasztalatait a leendő programozókkal, általános tanácsokkal ellátva őket:

Ahhoz, hogy megtanuljunk jól játszani egy hangszeren nagyon sok gyakorlásra és az erre áldozott időre van szükségünk. Gyakorlatilag hasonló mondható a programozás elsajátításáról is, bár némileg gyorsabban válhatunk jó programozóvá. Sajnos azonban nem annyival könnyebben és gyorsabban, mint ahogy azt a legtöbben szeretnénk.

…

Egy kisebb programot megírhatunk „nyers erővel", még akkor is, ha felrúgjuk a jó stílus minden szabályát. Nagyobb programoknál ez egyszerűen nincs így. Ha programunknak rossz a felépítése, azt fogjuk tapasztalni, hogy ugyanolyan gyorsan keletkeznek az újabb hibák, mint ahogy a régieket eltávolítjuk.

…

A tanácsok útmutatásul szolgálnak, csak ott kövessük, ahol értelme van. A saját tapasztalatot, józan észt és jó ízlést nem lehet semmivel kiváltani.

Az alábbiakban mi is néhány egyszerű tanáccsal szeretnénk szolgálni:

Ne programozzunk „visszafelé"!

Miután sikerül jól megoldanunk egy kitűzött feladat valamelyik részfeladatát, és rátérünk az újabb részfeladat megoldására, időnként úgy tűnhet, korábbi kódrészletünk némi átalakítással ezt is megoldaná. Ekkor nagy a kísértés, hogy visszalépjünk a kódban és annak átalakításával „két legyet üssünk egy csapásra"! Ne engedjünk a kísértésnek!

A tapasztalat szerint az átalakítás legtöbbször elrontja a korábban jól működő algoritmust, változóink „összeakadnak" mivel a két eljárás egyszerre akarja írni értékeiket, egyes eljárások „maradványértékei" nem várt, nehezen kideríthető hibákat okoznak, más típusú kivételek keletkeznek az egyes eljárásokban, amelyek kezelése egyre átláthatatlanabb kódot eredményez, stb. Természetesen lehet létjogosultsága egy többet tudó, rugalmas kódrészletnek, ennek precíz kialakítása azonban időigényes!

Programozzunk modulárisan!

A különböző feladatok megoldására fejlesszünk önálló modulokat, és ne akarjunk túl sok funkciót belezsúfolni, persze túl keveset se! Egy fotelágy fotel is, meg ágy is, viszont a tapasztalat szerint fotelnek sem, ágynak sem igazán kényelmes. A fotelből viszont ne hagyjuk el a párnázást, mert ez természetes módon hozzátartozik. Moduljainknak legyenek jól körülhatárolt feladatai, ne legyenek „áthallások" az egyes modulok között. Áttekinthetőbb a kód, könnyebb a hibakeresés.

Használjunk tömör, „beszédes” változóneveket!

Egy rosszul megválasztott változónév, az általa keltett pontatlan asszociációk révén kifejezetten gátolhatja, hamis irányba viheti gondolkodásunkat. A túl hosszú változózó neveket sokáig tart begépelni, és kiolvasni és képesek nehézkessé tenni gondolatainkat. A túlzottan rövid, elvont változónevek egy idő után tartalmilag kiüresednek, semmilyen hasznos képzettársítást nem okoznak, néhány perc, de legalábbis néhány nap múlva elfelejtjük bevezetésük célját.

Tervezzük meg előre összetett típusú változóink szerkezetét!

Egy jól megválasztott összetett adattároló struktúra, például rekordok tömbje, vagy egy osztály jelentősen egyszerűsítheti algoritmusaink bonyolultságát, igaz, megnövekszik a felhasznált memóriaterület és a tárhoz fordulások száma is.

Kódírás előtt modellezzük logikailag a megoldást!

Amikor a rendelkezésünkre álló bemeneti adathalmazból ki kell „bányásznunk” egy információt, és nem tudjuk, hogyan kellene hozzákezdeni, gondoljuk végig, mit tennénk akkor, ha csak egy papírlapunk és egy tollunk lenne, és persze rengeteg időnk. Programunknak pontosan ezen végiggondolt lépéseket kell lekövetnie.

Fejlesztés közben gyakran teszteljük programunkat!

Kódírás közben minden lényeges változtatás után érdemes azonnal tesztelni! Ha túl sokáig várunk, egyre több új elem jelenik meg kódunkban, így egyre nehezebb lesz beazonosítani a felmerülő hibákért felelős kódrészletet.

Fordítsunk figyelmet az összetett logikai kifejezések használatára!

A programozási nyelvek különbözőképpen értékelik ki az összetett logikai kifejezéseket. Egyes nyelvek például az *A és B* típusú összetett kifejezések *B* állítás részét még akkor is kiértékelik, ha az *A* állítást hamisnak találták, noha *A* hamissága már biztosan hamissá teszi *A és B* –t, függetlenül *B* logikai értékétől. Ez kellemetlen és nem várt kivételt okozhat például feltételes ciklus esetén. Használjunk korrekt zárójelezést, még akkor is, ha a zárójelek egyébként a precedencia szabályok miatt elhagyhatók lennének: *Ha ((A) és (B)) akkor …* . Összetett logikai kifejezések tagadására különös figyelmet fordítsunk! Például az alábbi tagadó kifejezés a logikai eseménytér hét(!) különböző részhalmazának unióját jelöli: *(Nem (((A) és (B)) és (C))).*