Kódcsipeteken keresztül kellett összevetni a C++11 másoló és mozgató szemantikáját, valamint a mozgató konstruktort a mozgató értékadásra kellett alapozni. Íme a teljes kód ami bemutatja mind a másoló konstruktort, másoló értékadást, mozgató konstruktort és mozgató értékadást: De nézzük elemezve a kódot. A program class private részében először is van két változó. Egy alap int típusú ez lesz az "a". A másik egy pointer amit \* jelölünk ez a "b". A legelső egy default két paraméteres konstruktor. Itt csak annyi történik, hogy amikor ez meghívódik akkor az "a"-t egyenlővé teszi az első elemmel, a "b" viszont mivel pointer jelölni kell egy \*-al, így mentjük el az értékébe az elem2-t. A kiíratás egy nyomkövető üzenet. Ezután a sorban a másoló konstruktor következik. Ezt akkor használjuk amikor úgy inicializálunk egy osztályt, hogy egy már kész ugyanolyan típusú osztály példány értékeit másoljuk át. Ez a konstruktor annyira fontos a C++-ban, hogy ha nem hozunk létre ilyet, automatikusan létre fog jönni egy. Azonban ezek a fordító által létrehozott konstruktorok csak sekély másolatot készítenek. Ez azt jelenti, hogy csak az eredeti objektumra való hivatkozást másolja és nem annak az értékét. Ezért ha pointerekkel és referenciákkal dolgozunk csak maga a pointer másolódik, ekkor kell létrehoznunk egy saját mélymásoló konstruktort. A mélymásoló külön memóriát foglal le a másolt információk számára. Vagyis a forrás és a másolat különböznek. Ha a mélymásolásnál megváltoztatjuk a másolat értékét akkor a forrás nem változik, nem úgy mint a sekély másolásnál. Tehát az egyik memóriahelyen végrehajtott módosítások nem befolyásolják a másikat. Ha pedig dinamikus memóriát pointerek segítségével allokálunk, akkor kell a felhasználó által definiált másoló konstruktor, hogy mind a két objektum különböző memóriahelyekre mutasson. Egy ilyen konstruktor látható fent is. A paraméterében egy objektumra vonatkozó referenciát kell tartalmaznia másképpen végtelen ciklus jön létre. A törzsben jelenleg egy nyomkövető üzenet van, egy sima int típusú érték átmásolás, valamint a b pointerbe másoljuk a másolandó példány b értékét. A másoló értékadást akkor használjuk amikor egy osztály értékeit egy másik már létező osztály értékeibe szeretnénk másolni. Meghívása: osztaly1 = osztaly2 Ez is ugyanúgy mint a másoló konstruktornál automatikusan létrejön a fordításkor, de ugyanaz lesz a helyzet itt is csak sekély másolás történik, így ahhoz, hogy pointerekkel és referenciákkal tudjunk dolgozni itt is létre kell hozni egy saját másoló értékadást. Ebben az esetben mint a másoló konstruktornál van egy sima int érték átmásolás és a b pointerbe másoljuk a másolandó példány értékét. Míg a másoló konstruktor és másoló értékadás célja, hogy másolatot készítsen egyik objektumról a másikra, addig a mozgató konstruktor és értékadásé, hogy átruházza az értékeket egyikről a másik objektumra. Ez a művelet sokkal kevesebb erőforrást igényel, mint a másolás, mivel itt nem kell lemásolni az értékeket csak átadni neki. Eközben az eredeti elveszti a tartalmát és használhatatlanná válik. A mozgató konstruktor és értékadás definiálása analóg történik. A másoló szemantika esetén a paraméter egy konstans bal érték referencia, amíg a mozgató szemantikánál egy nem konstans jobb érték referenciáról beszélünk. Ezt láthatjuk is a programban mivel a referencia & jellel van jelölve a paraméterben. A mozgató értékadás csak annyit tesz, hogy felcseréli a két példány tagjait a már beépített swap metódussal. A mozgató konstruktort pedig a mozgató értékadásra kellett alapozni. Ez úgy történik, hogy amikor meghívódik a konstruktor akkor amelyikbe mozgatni fogjuk az értékeket annak először kinullázzuk az értékeit, majd a \*this = move(adott); sorral meghívjuk a másoló értékadás függvényt, ez felcseréli a példányok értékeit, így a forrás kapja a nullákat és a másik a forrás értékeit. A program osztályban ezután csak egy print metódus van, hogy szebben láthassuk mi is történik. Az "a" értékét írjuk ki, majd a "b" értékét, majd a memóriacímet ahová a "b" mutat. Valamint kell még egy destruktor, ami törli a pointereket a program végén, ezzel memóriát szabadítva fel. A main metódusban először létrehozzuk az Alap példányt amit másolni fogunk és kiíratjuk az értékeit. Ezután lemásoljuk az Alap példányt az Alap\_masolatba és kiíratjuk az értékeiket. Létrehozunk egy Uj1 nevű példányt, amibe másoló értékadással átmásoljuk az Alap\_masolat értékeit. Már csak a mozgató konstruktor és értékadás maradt. Ezzel a sorral: program Uj2(move(Alap)); híjuk meg a konstruktort. Mivel a beépített move függvény az Alapból jobb érték referenciát hoz létre, így tudjuk meghívni. A program futás után: Ahogyan látszik a másoló konstruktornál az értékek ugyanazok a másolat példányánál, de a memória cím más. A másoló értékadásnál ugyanez látható. A mozgató konstruktornál és értékadásnál pedig az látszik, hogy a forrást nullára állítottuk és az Uj2 kapta meg az értékeket. Ahogy látszik, ha jól megtanuljuk az alapokat nem okozhat nehézséget ezeknek a konstruktoroknak és értékadásoknak a használata, sőt megkönnyíthetik az egyes programok működését ha megfelelően használjuk őket.