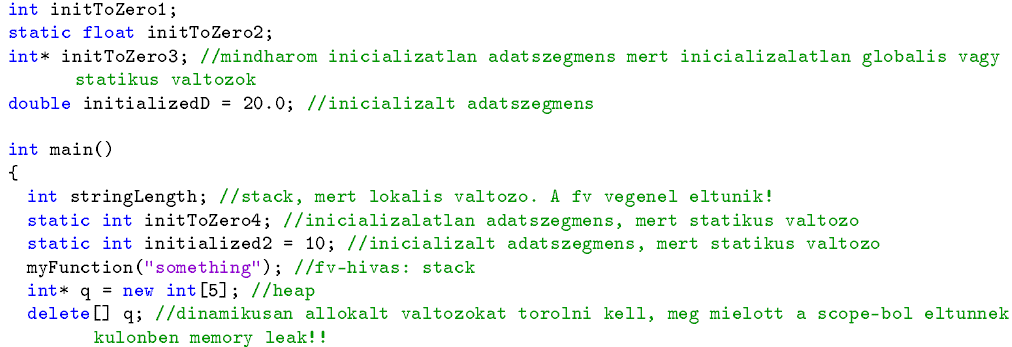
C++

1. **Mi a különbség a stack, heap, adatszegmens között? Melyik változó hova kerül?**

A heap (halom) a dinamikusan foglalt változókat tárolja (new/delete). A stack (verem) egy LIFO tároló, ami a függvények adatait tárolja (paraméterek, lokális változók). Az adatszegmens a globális, és a statikus változókat tartalmazza. /Kódszegmens: a futtatható kódot tartalmazza. /

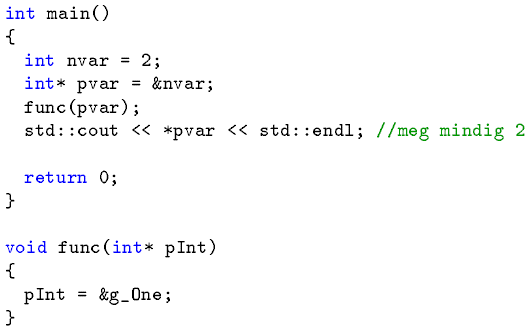
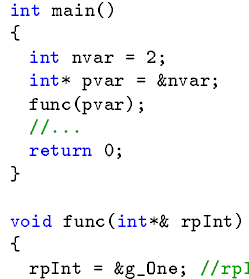


1. **Mit jelent a pointer? Mutassa be a 2 fajta operátort egy példán keresztül!**

Egy objektum hozzáférhető, ha ismert a címe, típusa. Erre való a pointer, ami egy adott típusú memóriaterületre mutat, ahol az objektum adatai találhatók.   
-Referencia operátor: objektumból címet ad vissza (&a)  
-De-referencia operátor: címből objektumot, értéket ad vissza (\*a)

1. **Mi a különbség az alábbi példában a 2 megvalósítás között?**

A baloldali esetben pointer kerül átadásra a függvénynek, ilyenkor a változó csak másolásra kerül, a függvényben hiába változik meg az értéke, visszatérés után a main-ben nem változik meg az érték. Ezzel szemben a jobb oldali példában referenciaként kerül átadásra a változó így, ha a függvényben megváltozik az értéke, az kihatással van a main-beli értékére is.



1. **Írjon példát egy double visszatérésű függvényre, melynek 2 paramétere double\*, és int! Mi a függvény pointer szintaktikája? Hogy hívható meg egy függvény pointeren keresztül? Módosítható-e a függvény dereferencia után?**

* double fuggveny(double\*,int)
* szintaktika: double (\* fvpointer)(double\*,int)
* fvpointer=&fuggveny;  
  fvpointer(&d,2);
* Nem módosítható a függvény, mert kódszegmensben található

1. **Milyen 3 lépésben történik a forráskód programmá alakítása? Linkelés során mire kell figyelni? Extern mit jelent?**

* Preprocesszálás: makrók, #include-ok betöltése, eredménye: futtatási egység
* Futtatási egység feldolgozása, minden (.cpp) fájl külön lefordítása
* Linkelés: külön lefordított részek egyesítése egy futtatható állományba
* Linkelés során fontos, hogy ne includeoljunk, illetve ne definiáljunk 2x semmit (header guard)
* Extern: Másik fájlban már definiálva van a változó

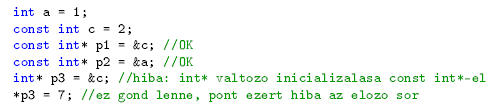
1. **Mik a referenciák? Miért vezették be őket? Hány féle van? Hogyan jelöljük őket? Pointer és referencia különbsége? Van értelme pointer referenciájának? És referencia pointerének?**

Referencia hasonló a pointerhez, csak ez ugyan olyan szintaxissal hivatkozható, mint maga az objektum, mindig arra az objektumra mutat, amelyre inicializáltuk, ezenkívül nagy előnye a pointerekkel szemben, hogy muszáj valamire mutasson, nem létezik null reference, ezért ezt nem kell ellenőrizni. Első sorban azért vezették be, mert a pointer szintaxisa sokszor átláthatatlan volt: \*, -> , pointer más időpillanatban más objektumra mutathat. Ha függvénynek referenciát adunk át, az változtathatja annak értékét. A referencia tulajdonképpen egy alias. 3 fajtája:

* lvalue referencia: megváltoztatandó objektumra hivatkozik, inicializása csak lvalue-n keresztül lehetséges  
  int i=1;  
  int& r=i;  
  int& r=1; //error, mert nem lvalue az 1  
  r++; //i=2 lesz
* const referencia: konstans objektumra hivatkozik, hivatkozható, de nem írható, inicializálható rvalueval is, más típussal is
* rvalue referencia: olyan objektumra hivatkozik, melyek értékét használat után nem akarjuk már felhasználni, destruktív olvasásra használt, ilyenkor az adat nem kerül másolásra, hanem „elmozdításra” kerül, csak rvalue-val hozható létre, jelölése &&  
  string&& r3{„Valami”}

A pointer referenciájának van értelme, a referenciára azonban nem lehet mutatni, mert annak nincs címe.

1. **Melyik sor hibás?**



int const\* p3=&c; //OK  
int const\* p4=&d; //OK  
int \* const p2=&c; //error

int const\* az ugyan az, mint const int\*.

1. **Hogyan foglalható, és szabadítható fel dinamikus memória? Mutasson példát int tömb foglalására! New, delete nyers használata miért nem jó?**

* int \*q=new int[5];  
  delete[] q;
* Nem szerencsés a nyers használatuk, mert memory leak lehet, vagy 2x-es törlés, ami hibát okoz, ezért konstruktorban és destruktorban érdemes használni őket.

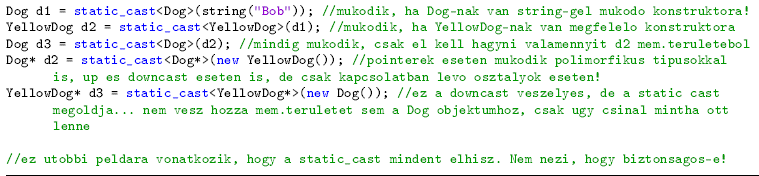
1. **Hány darab konstruktor készíthető egy osztályhoz, van-e felső korlát? Mutasson példát X osztály copy konstruktorára!**

Nincs felső korlát, akármennyit lehet csinálni, csak az argumentumok száma, vagy fajtája különböző kell legyen.   
X osztály copy konstruktora: X(const X& x)

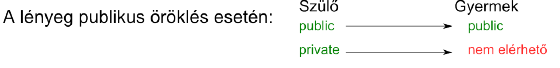
1. **A lenti példa alapján mi a különbség a static és a dinamic cast között?**

**dynamic\_cast**: nullptr-t ad vissza, ha nem megfelelő objektumra próbálunk castolni, referencia esteén kivételt ad. Ha dynamic\_cast <Sz\*> (p) hivásban p egy Sz típusú pointer, vagy G típusú (G gyereke Sz-nek), akkor p pointer memóriaterülete kezelhető úgy, mint egy Sz típusú objektum. Ha ferrari& referenciát akarunk dynamic\_cast-olni auto& referenciára, minden oké, mert a ferrari objektum tartalmazza magában az auto objektum memóriaterületét is, abból csak le kell vágni. Fordítva viszont nem biztonságos ez a dolog. Nem működik, ha: a 2 osztály nem polimorfikus, vagy ha adott példány biztonságosan nem konvertálható a másikba (pl. gyémánt-alakzat virtuális öröklés nélkül).

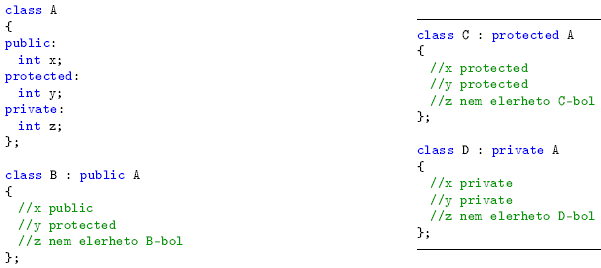
**static\_cast**: fordításidőben végzi a konverziót. Akkor tudja elvégezni, ha létezik a két objektum között definiált konverzió (van megfelelő konstruktor). Ha pointerről, referenciáról van szó, akkor működik, ha az egyik osztály a másikból származik, ilyenkor abban az esetben is működik, ha a konverzió nem biztonságos (up,downcast is).



1. **Public, protected, private öröklés? Melyiket mikor használjuk? Miért nem ajánlott a protected adattag?**

* Public: pl. a manager egyfajta employee, ekkor az employee publikus tagjaihoz hozzáférhet a manager bármely függvénye, barátfv.-e, manager objektumon keresztül bármilyen sima fv.
* Private: alapesetben ez történik, ekkor az employee publikus tagjaihoz hozzáférhet manager bármely fv-e, barátfv-e, de nem férhet hozzá sima fv. Ezért mondhatjuk azt, hogy a privát öröklés implementációs részletek elrejtéséhez hasznos.



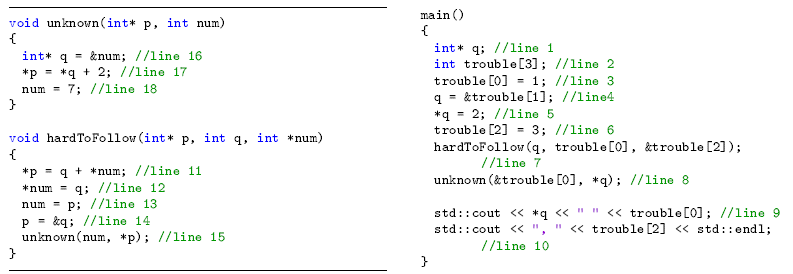
* Protected: implementációs részletet rejtünk el itt is, de a további öröklés lehetséges.
* A protected minősítés függvények esetén ok, de adattagok esetén nem ajánlott, mert nem tiszta kódhoz vezet. Ismeretlen embernek úgy tűnhet, ha valami protected, akkor azt valaki módosítani fogja, de ki és hol? Ha pedig nem akarjuk amúgy sem módosítani, akkor inkább legyen private alapból.

1. **Mi történik olyankor, amikor egy olyan típusú objektumot teszünk egyenlővé egy olyan típusú objektummal, amely típushoz nem definiáltunk konstruktort?**

Ebben az esetben a fordító létrehoz egy alapértelmezett copy konstruktort, ami csak annyit tesz, hogy egy az egyben lemásolja az objektum memóriaterületét. Ez addig ok is, amíg az objektumhoz nem tartozik dinamikus memória. Ha van foglalva dinamikus memória, akkor ezzel ugyan oda mutatna az új objektum mutatója is, ezért ebben az esetben célszerű saját copy konstruktort írni.

1. **Mik a virtuális függvények? Hogyan hozhatunk létre absztrakt osztályt?**

A virtuális függvényekkel lehetséges megmondani, hogy a szülőosztály egyes függvényei újradefiniálhatóak. Ilyenkor a linkelő összekapcsolja a megfelelő objektumot a megfelelő függvénnyel. A virtuális függvényeket nem kötelező a leszármazottban felüldefiniálni. Absztrakt osztály létrehozható úgy, ha csak pure virtuális függvények találhatóak az osztályban. (=0) Ha a virtuális függvény pure, akkor kötelező felüldefiniálni. Ezek segítségével működni tud a futás idejű polimorfizmus, pl.: szülő osztályú konténeren keresztül a leszármazott objektum is elérhető. Virtuális függvénnyel rendelkező osztályban a destruktor is legyen virtuális.

1. **Vezesse le az alábbi program működését!**

