JEGYZŐKÖNYV
==========
Varga Márton
==========

Első hét:

OO szemlélet: A feladat nem tudtam önálló megoldani ezért Bátfai Norbert tanár úr forráskódjából vettem ihletet. A feladatot értelmeztem és sikeresen fordítottam, futtattam.

Yoda:Az aktualis Yoda feladatnál sikerult egy sajat kódot irni ami sikeresen fordult és futott.

Gagyi:A Gagyi feladat algoritmusa megmutatja a különbséget hogy a program hogy reagál arra ha -128nál kisebb számot adunk meg és a while ciklus végtelen lesz.

Homokozo:A program végleges megírásához tutort vettem segítségbe ami igy sikeresen futott és fordult.

Kodolas_From_Sracth:Az algoritmus félig tudtam önálló megoldani ezért Bátfai Norbert tanár úr forráskódját vettem igénybe. A feladatot értelmeztem és sikeresen fordítottam, futtattam.

Második hét:

Liskov: A feladat lényege a következő: “Ha egy alap madár osztályból származtatunk további osztályokat, akkor mindegyik örökölni fogja a repülést, viszont a pingvin nem tud repülni.” Sikeresen megirva és futattva.

Anti oo: Az algoritmust tutor segitségével sikerult megoldani.

Szulo-gyerek:A feladat java verziójával sikerült megoldani a feladatot viszont a c++ verziót még nem sikerült elkészíteni. FOlyamatban van.

Harmadik hét:

Bpel: A megoldást a keresztapukám segítségével sikerült megvalósítani.

Bpmn: A program végleges megírásához tutort vettem segítségbe ami igy sikeresen futott és fordult.

Esettan:Elolvastam az aktualis könyv részt.

Forward Engineering Uml: A megoldást a keresztapukám segítségével sikerült megvalósítani.

Reverse Engineering v: Reverse engineering UML osztálydiagram:Az eclipse IDE-ben feltelepitettem az aktuális plugint amivel legeneráltam a feladat által kért megoldást.

Negyedik hét:

Encoding: A netbeans alap karaterkodólása Utf-8 a fileok pedig olyan karakterkodlást tartalmaznak mai nincs beaállitva ezért át kell állitani windows-1252-re.

Full Screen: A feladat lényege java grafkis képeségét szemlélteti.

Leetranslator: A carlexer.ll fájl elején rövidítések vannak definiálva, amelyek értékei vagy konkrét karaktersorozatként, vagy reguláris kifejezésként vannak megadva. A lexer ezeket a rövidítéseket használja fel mintaként, ha pedig a megadott minták valamelyikével egyezést talál, akkor annak a szövegét megfelelő formátumúvá alakítja át, és ezt memóriacímen tárolja.

Ötödik hét:

Rsa: A pdf-ben látható kódrészletek begépelve, kipróbálva, sikerült titkosított szöveget létrehozni.

Másoló-mozgató szemantika: Udprogon találtam rá egy példát, azt már elkezdtem értelmezni, de még nem teljesen világos.

Jdk: Az eszé kifejti a feladat lényegét igy nem tartottam szükségesnek a meglétét.

Esszé: Készen van a githubon elérhető.

Hatodik hét:

Costum alloc: Bátfai Norbert tanárúr forráskódját sikeresen forditottam, futtattam majd értelemztem.

STL map érték: A rendezendő mapet egy párokat tartalmazó üres vektorba másoljuk, a vektort a párok második értékei alapján rendezzük, utána pedig kiíratjuk a rendezett vektort.

Alternativ Tabella: A Comparable interface használatával az adott class objektumai megfelelő sorrendbe rendezhetőek. Sikeresn fordult és futott.

Esszé: Folyamatba van…

Hetedikhét:

SamuCam: A samucam programban amikor a qmaket ráküldöm teljesen jól readgál a program, viszont mikor a maket küldöm rá fatal errort ad vissza az opencv miatt pedig fel van telepitve. a webcam kezeléséről: videoCapture.open (videoStream); résszel elinditja a felvételt. videoCapture.set (CV_CAP_PROP_FRAME_WIDTH, width); videoCapture.set (CV_CAP_PROP_FRAME_HEIGHT, height); videoCapture.set (CV_CAP_PROP_FPS, 10); résszel pedig beállitja a méretet és az fps-t. void SamuCam::run() résznél kéri be az XML-t és futtatja a felvételt.

Future tevékenység: Nem sikerült telepiteni a java fx-t valószinüleg szerve hiba miatt.

Brainb: A program forditásához és futtatásához szügséges dolgokat feltelepitettem majd a forditás és a futtatás sikeres volt. A qt slot-signal mechanizmusról: A signals and slots mechanizmus teljes egészében Qt-specifikus.A Qt-ban

Osm térkép: Folyamatban van.

Nyolcadikhét:

Junit teszt: Sikerült a kézzel számított mélységét és szórását bele dolgozni egy Junit tesztbe.

Összefoglaló: Esszé a 2. feladatról (AOP) Mi az az AOP? Legáltalánosabban megfogalmazva egy magasabb szintű absztrakciót vezet be az OO-hoz képest. Másképp fogalmazva: megadja nekünk az alkalmazáslogikát keresztbe-kasul vagdosó síkok (mint például a felhasználó-azonosítás) kiemelésének lehetőségét. Miért hívják aspektus-orientált programozásnak? Azért, mert a feladatokat bizonyos aspektusok szerint értelmezzük: például egy átutalás a naplózás vagy a tranzakció-kezelés szempontjából. Új aspektusokat bármikor létrehozhatunk. Tegyük fel, hogy létrehozunk egy osztályt, amely elvégez egy bizonyos dolgot. Ehhez mi írunk egy (vagy több) aspektust, amellyel olyan utólagos feladatokat tudunk elvégezni, mint például a naplózás. Ekkor a naplózás külön válik a főprogramtól, így azt bármikor anélkül tudjuk módosítani, hogy a főprogramba bele kellene nyúlni. Bármikor összeköthetjük az aspektusokat és az osztályokat, így a tervezés folyamatában nagyobb szabadságot élvezünk. Aspektusokat Java nyelven írhatunk, egy külön fájlban, melynek kiterjesztése '.aj'. Ha megvan a kapcsolódó aspektus, a szokásos 'javac' fordító helyett az aspektus fordítójával, az 'ajc' segítségével bele tudjuk "szőni" az aspektust a főprogramunkba, fordítás közben. Ezt láthatjuk a forrásban kommentelt parancsnál is: ajc *.java SzamolAspect.aj -cp aspectjrt-1.8.12.jar Több forrásfájl esetén mindegyikbe bele kell szőni, erre utal a *.java. Vegyük sorba a programunk részeit. Az egész nagyon hasonlít egy Java osztály deklarációjára. 1) Új aspektust hasonlóan hozunk létre, mint egy Java osztályt: public aspect ASPEKTUSNÉV 2) Aztán jönnek a változó deklarációk, amelyet használni

akarunk. Mi jelenleg a 'nulla' és 'egy' int típusú változókat használjuk arra, hogy a számlálók értékét elrakjuk. 3) Utána a pointcut-ok megadása. Join point: kisebb megszorításokkal a program bármely azonosítható futási pontja lehet: egy metódus hívása, egy változó értékadása, stb. Pointcut: ezek tartalmazzák és fogják egybe az alkalmazás join pointjait, őket definiáljuk, hiszen a join pointok a valós osztályokban vannak. A megfelelő függvényhez megfelelő pointcutokat hozunk létre, amelyeket majd az advice-oknál fogunk felhasználni. 3 metódust fogunk figyelni: Csomopont.ujNullasGyermek, ha új nullás kerül beszúrásra; Csomopont.ujEgyesGyermek, ha új egyes kerül beszúrásra; valamint a main() metódus. KITÉRÉS: az execution() és a call() programkiválasztó. Az execution magának a kijelölt metódusnak a végrehajtását kapja el, tehát a kijelölt metódus törzsébe szúrja a tanácsot. A call() a metódus hívását kapja el, tehát a tanácsot a metódus hívása elé és utána szúrja be. Normálisan ennek nincs jelentősége, de speciális esetekben fontos lehet. A Jáva alkalmazás main() metódusát pl. nem lehet call() kiválasztóval elkapni, mert ez a hívás nincs a Jáva programban. (Ugyancsak fontos lehet a különbség konstruktorok hívásánál.) Ez az oka annak, hogy a main() metódusnál nem call()-t látunk a forrásban. 4) Advice-ok/tanácsok. Az advice határozza meg, mi történjen az adott join pointnál. A before() tanács a metódus futása előtt, az after() tanács a metódus futása után hívódik meg. Megfelelő helyen növeljük a változóink értékét, a after():vege() tanács pedig jelzi, hogy a main() metódus futása után írjuk ki az eredményeket. Forrás: https://prog.hu/cikkek/905/aspektus-orientalt-programozas

AOP: A feladatot tutor segitségével sikerült megvalósitani sikeresen fordult es futott.