Hlutapróf 3

English follows

1. **Krossaspurningar (25%)**
   Staðsetning: Hlekkurinn Exam á skjáborði tölvunnar

2. **BST: insert (25%)**
   Staðsetning: Í möppunni Exam á skjáborði vélarinnar: Mappan ***BinarySearchTree***
   "Highscore" tafla hefur verið útfærð með tvíundarleitartré (***HighScoreBST***).
   Klárið að útfæra fallið *addScore()*. Fallið bætir einu *HighScore* inn í tréð. Þið megið skrifa og nota öll þau hjálparföll sem þarf í klasann *HighScoreBST*.
   Einnig megið þið nota minna en (<) virkjann á *Highscore* klasanum og/eða yfirskrifa aðra virkja eða bæta föllum við þann klasa.
   Athugið að hægt er að commenta inn og út köll á test föll í *main()*. T.d. væri hægt að kalla einungis á *testInsert()* á meðan verið er að leysa þetta verkefni. Skráin "expectedOutput.txt" gerir ráð fyrir að öll test föllin séu keyrð.
   Ekki þarf að hafa áhyggjur af *minnisleka* í þessu verkefni. Á meðan ekki er búið að útfæra *removeNode()* lekur eyðirinn *~HighScoreBST()* minni.

3. **BST: removeNode (25%)**
   Staðsetning: Í möppunni Exam á skjáborði vélarinnar: Mappan ***BinarySearchTree***
   Þar hefur "highscore" tafla verið útfærð með tvíundarleitartré (***HighScoreBST***).
   Klárið að útfæra fallið **removeNode()**.
   remove() hefur þegar verið útfært, sem og einhver hjálparföll, en eitt fallanna kallar á **removeNode().** Þið verðið að klára útfærsluna á því falli. Nota má fallið **removeAndReturnScoreFromLeftMost()** sem hjálparfall.
   Athugið að hægt er að commenta inn og út köll á föll í *main()*, t.d. má kalla einungis á *removeTest()* á meðan verið að útfæra þennan hluta, en skráin *"expectedOutput.txt"* miðar við að öll þrjú test föllin séu keyrð.
   Það er í lagi að breyta bæði yfirlýsingum og útfærslum á þeim föllum sem þegar hafa verið útfærð, t.d. *remove()* og *removeAndReturnScoreFromLeftmost()*. Núverandi útfærsla notar pointer reference, en ef þið viljið frekar nota bara benda, og til dæmis skila gildi úr hverju falli, til að hengja nóðurnar aftur saman á leiðinni upp, þá er það í lagi, og skilagildum falla þarf þá að breyta í samræmi við það.
   Gangið úr skugga um að *forritið leki ekki minni* þegar verið er að fjarlægja stök úr trénu.
   Í möppu verkefnisins er *valgrind.sh* skrifta sem hægt er að tvísmella á til að keyra.

Verkefni 4 á næstu síðu

4. **Erfðir og STL (25%)**

   Staðsetning: Í möppunni Exam á skjáborði vélarinnar: Mappan **ParkingLot**

   Útfærið klasann *Car* sem yfirskrifar << *virkjann*. Það notar *virkjann* til að kalla í föll sem skrifa út á straum nafn bílsins (klasans). Útfærið einnig klasana *Toyota* og *Subaru* sem erfa *Car*. Útfærið þvínæst *Corolla* og *Yaris* sem erfa *Toyota* og *Forester* sem erfir *Subaru*. Hver þessara klasa á að skrifa út nafn foreldris síns, ásamt eigin nafni til viðbótar. Þannig skrifar *Corolla* út "**Car: Toyota Corolla**" á meðan *Toyota* skrifar út "**Car: Toyota**" en *Car* skrifar bara út "**Car**". Hver klasi á þó bara að skrifa út sinn hluta nafnsins en láta foreldrið um að skrifa út byrjunina.

   Ef illa gengur að forrita þessa virkni gangið þá bara úr skugga um að hver tegund skrifi eitthvað ólíkt þegar kallað er á << *virkjann*.

   Útfærið núna klasann *ParkingLot*. Hægt er að bæta tilvikum af taginu *Car** inn í *ParkingLot* gegnum fallið *add*. Athugið að **bendar** eru notaðir þannig að STL gagnagrind sem notuð er til að halda utan um bílana þarf að vera safn af *Car** (**<Car*>**) en ekki bara *Car*. Yfirskrifið << *virkjann* í *ParkingLot* og látið hann fara gegnum alla bílana sem bætt hefur verið á bílastæðið og senda þá á strauminn með <<. Ef þeir eru einfaldlega sendir sem *Car** ættu þeir sjálfkrafa að skrifa út, hver sína útgáfu af framleiðanda og tegund, ef erfðirnar hafa verið rétt útfærðar.

   Áætlað úttak má sjá í skránni *ExpectedOutput.txt* í verkefnismöppunni.

1. **Single choice and text questions (25%)**
   Location: The link Exam on the exam computer's desktop

2. **BST: insert (25%)**
   Location: The folder Exam on the exam computer's desktop: Folder ***BinarySearchTree***
   A "Highscore" table has been implemented usinga Binary Search Tree (***HighScoreBST***).
   Finish implementing the operation *addScore()*. The function adds one *HighScore* into the tree. You may write and use any helper functions you need on the class *HighScoreBST*.
   You can also use the less than (<) operator on the *Highscore* class and/or overwrite any other operators or add functions to that class.
   Note that you can comment out calls to the test functions in main(), for example to only call *testInsertOnly()* while solving this part. The file "expectedOutput.txt" assumes all test functions are run.
   You don't need to worry about *memory leaks* in this part. While *removeNode()* has not been implemented the destructor *~HighScoreBST()* will leak memory.

3. **BST: removeNode (25%)**
   Location: The folder Exam on the exam computer's desktop: Folder ***BinarySearchTree***
   A "Highscore" table has been implemented usinga Binary Search Tree (***HighScoreBST***).
   Finish implementing the function **removeNode()**.
   remove() has already been implemented, as well as some helper functions, but one of the functions calls **removeNode().** You must finish the implementation of that function. You may use the function **removeAndReturnScoreFromLeftMost()** as a helper function.
   Note that you can comment out calls to the test functions in main(), for example to only call *testRemoveOnly()* while solving this part. The file "expectedOutput.txt" assumes all test functions are run.
   It is OK to change both definitions and implementations of the functions that have already been implemented, like *remove()* and *removeAndReturnScoreFromLeftmost()*. The current implementation uses pointer references but if you would rather just use pointers and for example return values and then use those values to link nodes together on the way out of the recursion, then that is fine, but you must change the return values of functions accordingly.
   Make sure the pogram *doesn't leak memory* when items are removed from the tree. In the project folder ther is a *valgrind.sh* script that can be double clicked to run.

Problem 4 on the next page

4. **Inheritance and STL (25%)**

   Location: The folder Exam on the exam computer's desktop: Folder **ParkingLot**

   Implement the class *Car* that overloads the << *operator*. It uses *the operator* to call operations that write the name of the car (the class) to a stream. Also implement the classes *Toyota* and *Subaru* which inherit *Car*. Then implement *Corolla* and *Yaris* that inherit *Toyota* and *Forester* that inherits *Subaru*. Each of these classes should write their parent's name as well as their own. *Corolla* will write "**Car: Toyota Corolla**" while *Toyota* writes "**Car: Toyota**" but *Car* only writes "**Car**". Each class should still only write their own part of the name, but allow the parent to write the prefix.

   If you can not implement this functionality, just make sure that each class writes something different when the << *operator* is used (in order to finish the second part).

   Now implement the class *ParkingLot*. Instances of the type *Car** can be added to *ParkingLot* through the function *add*. Note that **pointers** are used so that a STL data structure used to contain the cars must be a collection of *Car** (**<Car*>**) but not just *Car*.

   Overload the << *operator* in *ParkingLot* and make it go through all the cars that have been added and write them to the stream using <<. Ef they are simly sent in as *Car** they should automatically write each their producer, model etc. if the inheritance has been correctly implemented.

   See the file *ExpectedOutput.txt* in the project folder.