Kodprov IOOPM HT14

Instruktioner

Hämta filerna: `curl -O http://wrigstad.com/ioopm/w4711.zip`[1]

Nu får du en zip-fil med koden till uppgifterna. Denna gång kommer inlämning etc. att ske helt på mail, och inte via git som på tidigare kodprov.

Lämna in tentan genom att skicka hela den som ett zip-arkiv till `tobias.wrigstad@gmail.com`. Om din katalog heter `USERNAME` kan du skapa ett zip-arkiv för denna katalog med kommandot `zip -r USERNAME.zip USERNAME` om du står i katalogen ovanför.

För att den automatiska rättningen skall fungera *måste* du lägga uppgift ett i en katalog som heter `uppgift1`, uppgift två i en katalog som heter `uppgift2`, etc. Katalogen du lämnar in ska alltså ha följande struktur:

```
USERNAME --+--- uppgift1
           |
           +--- uppgift2
           |
          ...
```

Den automatiska rättningen kommer att gå till så att vi kör dina inlämningar mot vissa testfall. Du har fått ut testfall eller motsvarande i detta prov som du kan använda för att kontrollera din lösning. Om du har löst uppgifterna på rätt sätt och testfallen som du får ut passerar är det *troligt* att du är godkänd. Man kan förstås aldrig vara helt säker med test, och på provomgången av kodprovet såg vi lösningar som fungerade enbart för de testfall som lämnades ut. Till exempel såg vi lösningar där man, istället för att göra en generell (korrekt) lösning med `strlen`, testade om en sträng var ”foo” och i så fall returnerade 3, om det var ”quux” returnerade 4, etc. där ”foo” och ”quux” var strängarna från testfallen. En sådan lösning klarar förstås inte de testfall som vi kör mot när vi rättar, och förklarar också varför vi inte lämnar ut samma testfall i tentan som vi kör vid rättningen.

Tillåtna hjälpmedel

- Det är tillåtet att använda `man`-kommandot för att ta fram dokumentationen om C-funktioner (skriv t.ex. `man strcmp`).
- Du får också använda Oracles JavaDoc på `http://docs.oracle.com/javase/7/docs/api/`. Övriga sidor på Internet är inte tillåtna. Du får också logga in på din studentmail för att lämna in.
- Du får använda medtagna böcker, en om Java och en om C.
- Du får använda Emacs, vi, gdb, Eclipse, Netbeans, gcc, java, javac. Fråga om du är osäker.
- I övrigt gäller samma regler som på en salstenta, dvs. inga mobiltelefoner, inga SMS, inga samtal med någon annan än lärarna oavsett medium, etc. Inte läsa gamla lokala filer på ditt konto, kopiera gammal kod, etc.

[1] Notera att `-O` avser bokstaven ’O’, inte en nolla.

Uppgift 1 – strängkonkatenering & jämförelse

Denna uppgift går ut på att skriva ett litet program, `c` som tar som argument tre strängar, konkatenerar de första två, och kontrollerar om resultatet är samma som den tredje. Om de är samma skall texten ”Samma!” skrivas ut, annars ”Inte samma!”. Till exempel:

```
$ ./c foo bar foobar
Samma!

$ ./c foo bar "foo bar"
Inte samma!

$ ./c foo bar barfoo
Inte samma!

$ ./c foo foo
För få argument!
```

1. Programmet skall ha tre funktioner:
   (a) `main` på sedvanligt maner.
   (b) `concatenate_string` som tar två C-strängar och returnerar den konkatenerade strängen, allokerad på heapen. Du får **inte** använda biblioteksrutiner (t.ex. `strcat`) för konkateneringen.
   (c) `strcmp_concat` som tar tre C-strängar och returnerar `true` (alltså vad som helst som inte är 0) om den sista är samma som konkateneringen av de första två, annars `false`. Du **får** använda `strcmp` för jämförelsen.

   Funktionen `strcmp_concat` skall använda sig av `concatenate_string`. Du behöver inte implementera jämförelsen själv, utan kan använda till exempel `strcmp`. Funktionerna `strcmp_concat` och `concatenate_string` skall betraktas som självständiga funktioner, det vill säga, de skall gå att använda i godtyckliga program. Till exempel får inte `concatenate_string` förutsätta att den enbart anropas från `strcmp_concat`.

Icke-funktionella krav

– Utöver att programmet skall vara korrekt enligt ovanstående specifikation och till exempel korrekt översätta de inledande exemplen måste du skriva programmet helt själv till 100%.

– Verifiera med `valgrind` att ditt program inte läcker minne. När du kör

```
valgrind --leak-check=full ./c foo bar barfoo
```

skall texten ”All heap blocks were freed – no leaks are possible” skrivas ut.

Notera att `valgrind` bara är åtkomligt på institutionens linuxsystem. För att logga in på en linuxmaskin, kör `linuxlogin`.

– Du kan inte förutsätta någon given övre gräns för strängars längd.

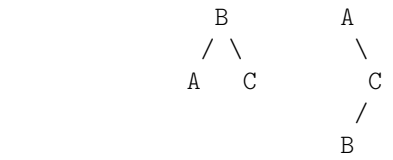
Inlämningsinstruktioner

Uppgiften måste lämnas in i katalogen `uppgift1`. Den skall endast innehålla:

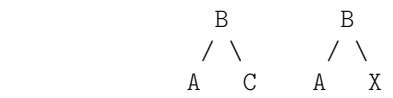
– `c.c`

Uppgift 2 – strukturell jämförelse

Filen `Uppgift2.java` innehåller ett enkelt Java-program som skapar två binära sökträd. Uppgiften går ut på att skriva en `equals`-metod så att `t1.equals(t2)` returnerar `true` om träden `t1` och `t2` innehåller samma element. Till exempel är följande träd lika:



. . . men inte följande:



Med *samma element* avses strukturell likhet för elementen, inte identitet.

När du kör programmet skall alltså se följande:

```
$ java Uppgift2
Träden (B(A)(C)) och (A()(C(B))) är lika
Träden (A()(C(B))) och (B(A)(C)) är lika
```

Notera att vi testar med betydligt större träd.

Icke-funktionella krav

- Utöver att programmet skall vara korrekt enligt ovanstående specifikation och till exempel korrekt skilja mellan identitet och strukturell ekvivalens ovan måste du skriva koden helt själv till 100%. (Utöver den kod som är given, naturligtvis.)
- Du får bara modifiera klasserna `Tree` och `Node`.
- Du får inte ändra klassernas nuvarande funktion, till exempel slå ihop dem till samma klass etc.
- Du får inte använda strängjämförelser.
- `equals`-metoderna skall ”overridea”/specialisera den i klassen `Object`

Inlämningsinstruktioner

Uppgiften måste lämnas in i katalogen `uppgift2`. Den skall endast innehålla:

- `Uppgift2.java`