Kodprov IOOPM HT14

Instruktioner

Hämta filerna: `curl -O http://wrigstad.com/ioopm/jor-el.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, inte kopiera gammal kod, etc.

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

Uppgift 1 – skriv klart ett existerande program

Denna uppgift går ut på att skriva klart en implementation av en ordnad mängd som använder en länkad lista under huven. Det finns också ett litet testprogram till den ordnade mängden. För enkelhets skull ligger allt i filen `o.c`. Testprogrammet tar ett antal strängar som inparametrar och lägger till dem i mängden. Dubletter ignoreras. Strängar som föregås av ett minustecken tas bort och inparametrarna processeras i den ordning de kommer. Några exempel på körningar av programmet[2]:

```
$ ./o 1 3 4 2
{ 1, 2, 3, 4 }
Set size: 4
$ ./o 1 3 2 4 -6
{ 1, 2, 3, 4 }
Set size: 4
$ ./o 2 3 1 4 -3
{ 1, 2, 4 }
Set size: 3
$ ./o 3 2 1 0 1 2 3 -1 -2 -3 -0 -1 -2 -3
{  }
Set size: 0
$ ./o 3 -3 3
{ 3 }
Set size: 1
```

Uppgiften består av två delar:

1. Skriva klart implementationen `ordered_set_del` som tar bort ett element ur mängden på ett givet index.
2. Se till så att programmet frigör allt minne som allokeras genom att skriva färdigt funktionen `ordered_set_rm`. Använd `valgrind` för att verifiera att programmet är fritt från minnesläckage.

Om du kör programmet nu kommer du märka att borttagningar, á la `-4` ovan inte fungerar då `ordered_set_del` inte är implementerad ännu, dvs.:

```
$ ./o 1 -1
{ 1 }
Set size: 1
```

Icke-funktionella krav

– Utöver att programmet skall vara korrekt enligt ovanstående specifikation måste du skriva de angivna delarna helt själv till 100%.

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

```
valgrind --leak-check=full ./o 1 -1 1
```

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:

– `o.c`

[2]Notera att mängden i testprogrammet innehåller *strängar* och att 22 kommer före 3 i lexikografisk ordning.

Uppgift 2 – implementation av interface

Filen `Uppgift2.java` innehåller ett Java-interface för en datasamling. Uppgiften går ut på att skriva en klass `Solution` som implementerar interfacet. Ett antal testfall är givna som ytterligare specifikation för implementationen. I händelse av diskrepans mellan tester och specifikation är det testerna som har företräde[3]. Du får inte använda existerande klasser i Javas API som löser problemet. Exempelvis, om interfacet är en datasamling får du inte använda en existerande datasamling för att lösa uppgiften.

Du kan kontrollera att din implementation uppfyller specifikationen genom att köra:

```
$ java Uppgift2
Alla tester passerar
```

Detta är förstås inte någon garanti för att lösningen är korrekt (t.ex. för att din lösning råkar fungera för just de värden vi testar med och inga andra, eller för mindre testdata, etc.).

Icke-funktionella krav

- Utöver att programmet skall vara korrekt enligt ovanstående specifikation måste du skriva de angivna delarna helt själv till 100%.
- Du får inte modifiera någon kod som är given, såvida inget annat anges i koden.
- Använder du ett verktyg som kräver att koden ligger i ett paket, ta bort alla paketdeklarationer innan du lämnar in!

Inlämningsinstruktioner

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

- `Uppgift2.java`

[3] Detta är i så fall oavsiktligt – det finns inga ”slamkrypare” i kodprovet.