Kodprov IOOPM HT15

Instruktioner

Öppna en terminal och skriv omedelbart

```
mkdir kodprov151023.
```

Gå in i denna katalog med

```
cd kodprov151023.
```

Hämta kodprovet till din dator med följande kommando:

```
curl --remote-name http://wrigstad.com/ioopm/broman.zip
```

Nu får du en zip-fil med koden till uppgifterna som du kan packa upp så här:

```
unzip broman.zip
```

Nu har du fått ett antal filer och kataloger:

- `uppgift1` – filer för uppgift1
- `uppgift2` – filer för uppgift2
- `Makefile` – en makefil för att **lämna in** kodprovet

Inlämning

Inlämning går till så här: ställ dig i katalogen `kodprov151023`. Om du har tappat bort dig i filsystemet kan du skriva `cd ~/kodprov151023`. Nu kan du skriva `make handin` för att lämna in kodprovet. När du kör detta kommando skapas en zip-fil med de filer som du har uppmanats att ändra i (inga andra), och denna fil sparas sedan på en plats där vi kan rätta provet.

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 tidigare omgångar av kodprovet har vi sett lösningar som fungerar enbart för de testfall som lämnats ut. Till exempel har vi sett 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.

Allmänna förhållningsregler

- Samma regler som för en salstenta gäller: inga mobiltelefoner, inga SMS, inga samtal med någon förutom vakterna oavsett medium.
- Du måste kunna legitimera dig.
- Du får inte på något sätt titta på eller använda gammal kod som du har skrivit.
- Du får inte gå ut på nätet.
- Du får inte använda någon annan dokumentation än mansidor och böcker.
- Det är tillåtet att ha en bok på en läsplatta, eller skärmen på en bärbar dator. Inga andra program får köra på dessa maskiner, och du får inte använda dem till något annat än att läsa kurslitteratur.
- Du måste skriva all kod själv, förutom den kod som är given.
- Du får använda vilken editor som helst som finns installerad på institutionens datorer, men om 50 personer använder Eclipse samtidigt så riskerar vi att sänka servrarna.

Vi kommer att använda en blandning av automatiska tester och granskning vid rättning. Du kan inte förutsätta att den kod du skriver enbart kommer att användas för det driver-program som används för testning här. Om du t.ex. implementerar en länkad lista av strängar kan testningen ske med hjälp av ett helt annat driver-program än det som delades ut på kodprovet.

I mån av tid har vi tidigare år tillämpat ett system där vi ger rest för mindre fel. Det är oklart om detta system tillämpas i år, men det betyder att det är värt att lämna in partiella lösningar, t.ex. en lösning som har något mindre fel.

För dig som vill använda en IDE

- Se till att inte halva kodprovet går åt till att skapa ett projekt och läsa in filerna.
- Se till att rätt fil lämnas in till slut.
- Om du måste stoppa in klasser i paket, ta bort paketdeklarationerna innan du lämnar in.

Uppgifter

Uppgift 1: Implementera en kö i C

Den här uppgiften går ut på att implementera en FIFO-kö. Förutom att skapa och ta bort en kö finns operationerna `enqueue` som lägger in ett element *sist* i kön, och `dequeue` som tar bort elementet som ligger först i kön (FIFO står för ”first in, first out”: det element som lades in först är också det som kommer ut först). Den utdelade koden kommer i tre filer:

1. `q.h` – Innehåller hela gränssnittet för köstrukturen
2. `q.c` – Här ska du skriva implementationen för alla funktioner i `q.h`
3. `main.c` – Ett litet program som använder köstrukturen för att lägga in talen 0 till 9, och sedan plocka ut ett tal i taget tills kön är tom. *Om kön är korrekt implementerad ska talen skrivas ut i samma ordning som de läggs in.*

Det finns också en `makefile` som du kan använda för att bygga programmet.

För att bli godkänd på uppgiften ska du skriva klart `q.c` genom att implementera alla funktioner som listas i `q.h`. Du får förstås skriva fler funktioner om du behöver, men inte fler filer.

Observera att testerna i `main.c` är till för att hjälpa dig att hitta fel i din kod och att programmet passerar testerna inte nödvändigtvis betyder att programmet är korrekt eller att du är godkänd.

Uppgift 2: Implementera en länkad lista i Java

Javaprogrammet `Driver.java` läser data från kommandoraden, lägger det i en länkad lista och skriver sedan ut listan, ömsom från slutet och från början (t.ex. `a b c` blir `a b c`). Din uppgift är att implementera en länkad lista i `List.java` så att Driver-programmet fungerar och ger den output som förväntas (du kan verifiera mot exemplen nedan, finns även i `Makefilen`. Du skall alltså enbart skriva en list-datastruktur, inte göra någon sortering etc.

Driver-programmet är typat mot det generella interfacet `Sequence` som finns i `Sequence.java`. Du skall alltså implementera detta interface i din lista. Utöver detta skall det vara möjligt att skapa listor, både som initialt tomma och utifrån en existerande lista i vilket fall alla element i den listan skall stoppas in i omvänd ordning. Driver-programmet använder sig av båda: den första lista som skapas är alltid listan `z, y, x` som sedan skrivs us `z x y` som exemplen nedan visar.

```
public interface Sequence<T>
{
    /**
     * Append e to the end of the list
     */
    public void append(T e);

    /**
     * Returns the nth element in the list (starting from 0).
     */
    public T get(int index);

    /**
     * Append e to the front of the list
     */
    public void prepend(T e);

    /**
     * Removes and returns the nth element in the list (starting from 0).
     */
    public T remove(int nth);

    /**
     * Returns the length of the list
     */
    public int length();
}
```

Du skall endast ändra i filen `List.java`.

`Sequence` använder sig av parametrisk polymorfism (generics). Din implementation skall bibehålla samma typsäkerhet som interfacet.

Du kan använda `make all` för att kompilera programmet och `make test` för att köra några enklare tester. Notera att programmet inte kompilerar just nu, eftersom `List.java` är tom.

Observera att testerna är till för att hjälpa dig att hitta fel i din kod och att programmet passerar testerna inte nödvändigtvis betyder att programmet är korrekt eller att du är godkänd.

Körexempel

Observera att programmet alltid stoppar in en lista med elementen x, y och z.

```
$ java Driver
z x y
z x
x
```

```
$ java Driver a b c
z x y
z x
x
a c b
```

```
$ java Driver a a a a a
z x y
z x
x
a a a a a
```

```
$ java Driver a b c d e f
z x y
z x
x
a f b e c d
$ java Driver a c e f d b
z x y
z x
x
a b c d e f
```