Kodprov 2019-10-23

1 Instruktioner

Öppna en terminal och kör följande kommandon:

1. `cd` (detta tar dig till din hemkatalog)
2. `mkdir kodprov191023`
3. `cd kodprov191023`
4. `curl http://wrigstad.com/ioopm19/misc/kodprov191023.zip -o k.zip`
5. `unzip k.zip`

Nu har du fått ett antal filer och kataloger:

`uppgift1` – katalog med alla filer för uppgift 1

`uppgift2` – katalog med alla filer för uppgift 2

`Makefile` – en makefil för att lämna in kodprovet

1.1 Inlämning och rättning

Inlämning går till så här: ställ dig i katalogen kodprov191023. Om du har tappat bort dig i filsystemet kan du skriva cd; cd kodprov191023. 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. Vid behov kan du köra make handin flera gånger - endast den sista inlämningen räknas. Den automatiska rättningen kommer att gå till så att vi kör dina inlämningar mot omfattande testfall. Du har fått ut mindre omfattande testfall eller motsvarande i detta prov som du kan använda som ett stöd för att göra en korrekt lösning. Experiment med att lämna ut mer omfattande tester har visat sig skapa mer[1].

[1] Att lämna ut exakt samma test som används vid rättning är heller inte lämpligt, då det har förekommit fall då studenter försökt simulera en korrekt lösning genom att bara hacka output för specifika testvärden.

Om du har löst uppgifterna på rätt sätt och testfallen som du får ut passerar är du förhoppningsvis godkänd.

1.2 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 man-sidor 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 ofta tillämpat ett system där vi ger rest för mindre fel. Det är oklart om detta system tillämpas för detta kodprov men det betyder att det är värt att lämna in partiella lösningar, t.ex. en lösning som har något mindre fel.

Lycka till!

2 C-uppgiften

A *priority queue* is an abstract data type which is like a regular queue, but where additionally each element has a priority associated with it. In a priority queue, an element with high priority (lower values) is served before an element with low priority (higher values). If two elements have the same priority, they are served according to the order in which they were enqueued.

The task is to implement the following functions:

- `priority_queue_insert_with_priority`

- `priority_queue_merge`
- `priority_queue_size`
- `priority_queue_destroy`
- `priority_queue_pull_highest_priority_element`

Example of `priority_queue_insert_with_priority`

First random queue 'a' `'[(1,s), (4,v), (4,k)]'`

Insert `(1,a)`

Result `'[(1,s), (1,a), (4,v), (4,k)]'`

Example of `priority_queue_merge`

First random queue 'a' `'[(1,s), (4,v), (4,k)]'`

Second random queue 'b' `'[(4,i)]'`

Merging queues a and b `'[(1,s), (4,i), (4,v), (4,k)]'`

Instructions can be found in comments in the code. Code for the priority queue is given, in addition to the functions that you will implement. All code you need to write is in `yourcode.c` and all test code is in `driver.c`.

NOTE !! Only `yourcode.c` is submitted! You do not need to change driver.c but you may want to do so (for example, commenting on tests that do not work, etc.) It can also be helpful to read the code in driver.c. As usual, you must not leak memory, read unallocated memory etc.

2.1 Testa din lösning

`make compile` kompilerar `driver.c` och `yourcode.c` till `a.out`.

`make test` kompilerar enligt `make compile` och kör sedan testerna.

`make memtest` kör testerna enligt `make test` under valgrind.

Du måste själv läsa outputen och tolka den för att se om allt gick bra!

3 Java-uppgiften

The task is to implement a *multiset* of arbitrary elements, all of which implement the `CompareTo` interface.

A multi set differs from a regular set in that the same element can occur several times, e.g. $\{1, 3, 3, 3, 5, 8, 8\}$ or {"abba", "abba", "catempire", "donovan"}

You can add elements to the multiset (up to a certain number of different elements) and ask the multiset whether a particular element is a member of it or not, and if so, how many times. Just as in a regular set, the order between elements does not matter.

NOTE! Write all code in the Multiset.java file that you must create yourself. Only this file should be submitted.

3.1 Requirements

For simplicity, all elements of the multiset must be stored in an array (called `elements`) whose size is fixed - ie. when creating a multiset it is determined how many (different) elements it can hold up to a maximum.

The number of occurrences of a certain element (the multiplicity) is stored in a parallel array (called count).

The elements in `elements` must be arranged ie. the following shall always apply to `i> = 0` and `i + 1 < elements.length`:

```
T a = elements[i];

T b = elements[i+1];

a.compareTo(b) < 0; ///is always true!
```

Since it is a bit tricky to create an array of generic type in Java, I publish that code here:

```
T [] elements = (T []) new Comparable [maxCapacity];
```

However, this leads to complaints from the compiler: Note: Multiset.java uses unchecked or unsafe operations. It is OK. Of course, things are going well - or better! - having an array of `Comparable []`. The Multiset class is declared as follows:

```
public class Multiset <T extends Comparable <T>>
```

i.e. the type `Multiset <Foo>` means a multiset of elements of the type `Foo` such that `Foo instanceof Comparable <Foo>` always applies. The `CompareTo` interface defines a method, int `compareTo (T o)`, such that:

`a.compareTo(b)` returns a negative number if a occurs before b in the order implemented

`a.compareTo(b)` returns 0 if a and b are equal

`a.compareTo(b)` returns a positive number if `a` comes after `b` in the order being implemented

(Thus equivalent to, for example, how `strcmp ()` works in C.)

Since `Foo instanceof Comparable <Foo>` is always valid, instances of `Foo` can only be compared with other instances of `Foo` using `compareTo ()`.

You therefore need at least two arrays and a maximum capacity of different elements.

3.2 Behaviour

The multitude should support the following operations:

- Constructor that takes as argument the maximum number of different elements
- Constructor that takes no arguments but delegates to the first constructor using 16 (default value) as maximum number of different elements
- `public void add (T elem)` that adds an occurrence of `elem` in the multiset if space is available. Whether an element is a member or not is determined using `compareTo ()`
- `private int indexOf (T elem)` which returns the location in `elements` where `elem` exists, alternatively should have existed if the item was included. To be able to distinguish between exists and should have existed, we use negative indexes for the latter, and since 0 is neither positive nor negative, we increase negative indexes by 1. Thus: -5 means that "if the searched element had been in elements it would have been on index 4 "while 4 means" the searched element is in place 4 ". Similarly, -1 means "would have existed first in elements", and 0 "first existed in elements".
- `public int contains (T elem)` which returns the number of times that `elem` appears in the multiset, ie 0 if the element is missing.
- `int size ()` which returns the size of the multiset, that is, the total number of occurrences of elements, initially 0. (For example, the multiset {1, 3, 3, 3, 5, 8, 8} has size 7.)
- A comparison method (`equals ()`). `a.equals (b)` must return true if `a` and `b` are aliases, or if `a` and `b` contain identical objects. Please note that it is not possible to make a safe version of this method! The compiler will complain about *Note: `Multiset.java` uses unchecked or unsafe operations...* Don't worry about this.
- Plus any other private methods you may need.

3.3 Non-functional requirements

- The only way to change the multiset must be through `add ()`.
- An attempt to create a multiset with negative maximum capacity, or call `add()` or `contains()` with `null` should throw the exception `IllegalArgumentException`. (There is a standard exception set available.)

- If a maximum number of different elements is exceeded, a `MultisetFullException` should be thrown. (NOTE: You have to write this yourself! Do it in `Multiset.java` and hard - otherwise, not even `Driver.java` will compile.)
- You should use `indexOf()` in the implementation of `add()` to find out if the new element already exists and if not (and where to place it).

3.4 Testing

You can test your submission with the help of `make`. Then the program runs `Driver.java`, which creates a few multisets and performs a few operations on them. In case of errors, a message is printed.

`make test` runs all tests without interrupting in case of error. `make test-hard` runs all tests and stops in case of error.