Kodprov 2019-08-30

1 Instruktioner

Öppna en terminal och kör följande kommandon:

1. `cd` (detta tar dig till din hemkatalog)
2. `mkdir kodprov190830`
3. `cd kodprov190830`
4. `curl http://wrigstad.com/ioopm19/misc/kodprov190830.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 `kodprov190830`. Om du har tappat bort dig i filsystemet kan du skriva `cd; cd kodprov190830`. 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 stress än hjälp (tänk fler testfall som fallerar)[1].

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-uppgift

Uppgiften går ut på att ta bort alla dublettelement från en (osorterad) länkad lista. Dvs om samma element länkas till från flera platser i listan så skall alla länkar till det elementet utom det första tas bort ur listan.

Funktionen du skall skriva heter `list_remove_dups()`. Utöver den skall du också implementera funktionerna `list_size()` som returnerar storleken på en lista och `list_destroy()` som tar bort en lista ur minnet, tillsammans med dess innehåll. Intruktioner finns i kommentarer i koden.

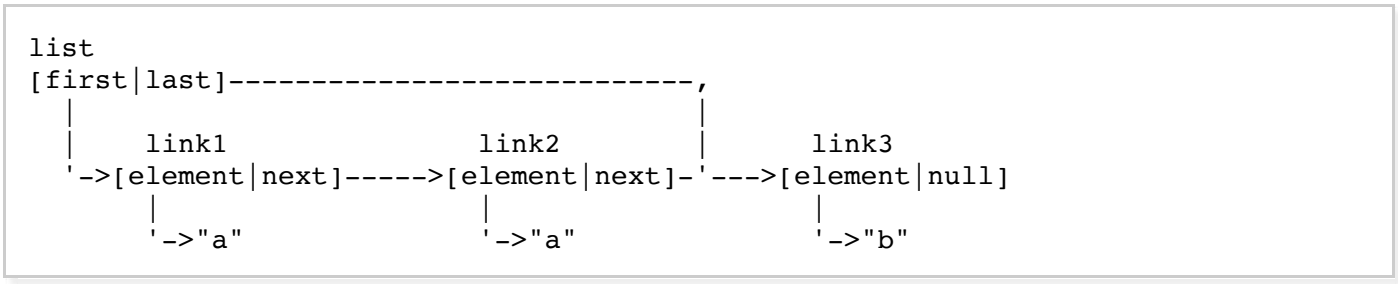
Kod för listorna är given, förutom funktionerna som du skall implementera. All kod du skall skriva är i `yourcode.c` och all testkod är i `driver.c`. **OBS!!** Endast `yourcode.c` lämnas in!

Du behöver inte ändra i `driver.c` men du kan vilja göra det för att t.ex. kommentera bort tester som inte fungerar, etc. Det kan också vara hjälpsamt att läsa koden i `driver.c` för ledning.

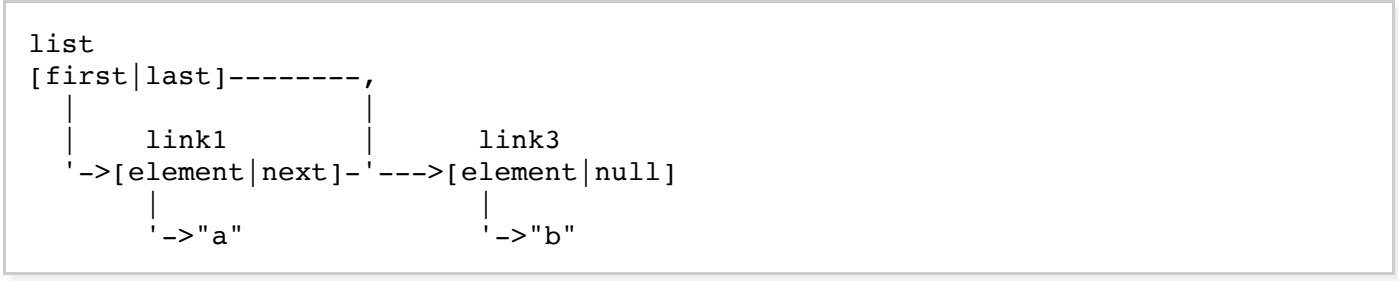
Som vanligt så får du inte läcka minne, läsa oallokerat minne etc.

2.1 Exempel

Ponera att du har den här listan:



Efter ett anrop till `list_remove_dups(list)` skall minnet se ut som följer: aldrig över.):



Tänk på att funktionen skall fungera oavsett hur många olika element som har dubletter och var dubletterna ligger. Om det inte finns några dubletter alls så skall den förstås lämna listan oförändrad.

2.2 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.

Testerna anropar `list_remove_dups(list)` på tre olika listor. **Du måste själv läsa outputen och tolka den för att se om allt gick bra!**

---

3 Java-uppgift

Uppgiften går ut på att implementera en *multimängd* (eng. multiset) av godtyckliga element som alla implementerar interfacet `CompareTo`.

En multimängd skiljer sig från en vanlig mängd genom att samma element kan förekomma flera gånger, t.ex. {1, 3, 3, 3, 5, 8, 8} eller {" *abba* ", " *abba* ", " *catempire* ", " *donovan* "}

Man kan lägga till element i multimängden (upp till ett visst antal olika element) och fråga multimängden om ett visst element är medlem i den eller inte och i så fall hur många gånger. Precis som i en vanlig mängd så spelar ordningen mellan element ingen roll.

OBS! Skriv *all* kod i filen `Multiset.java` som du måste skapa själv. Endast denna fil skall lämnas in.

3.1 Tillstånd

For enkelhets skull skall alla element i multimängden lagras i en array (kallad `elements`) vars storlek är fix – dvs. när man skapar en multimängd bestäms hur många (olika) element den maximalt kan rymma.

Antalet förekomster av ett viss element (multipliciteten) lagras i en parallell array (kallad `count`).

Elementen i `elements` skall vara ordnade dvs. följande skall alltid gälla för `i >= 0` och `i+1 < elements.length`:

```
T a = elements[i];
T b = elements[i+1];
a.compareTo(b) < 0; /// är alltid sant
```

Eftersom det är lite meckigt att skapa en array av generisk typ i Java ger jag ut den koden här:

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

Detta leder dock till klagomål från kompilatorn: *Note: Multiset.java uses unchecked or unsafe operations*. Det är OK. Det går förstås också bra – eller bättre! – att ha en array av `Comparable[]`.

Klassen `Multiset` är deklarerad enligt följande:

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

dvs. typen `Multiset<Foo>` betyder en multimängd av element av typen `Foo` sådana att `Foo instanceof Comparable<Foo>` alltid gäller. Interfacet `CompareTo` definierar en metod, `int compareTo(T o)` sådan att:

- `a.compareTo(b)` returnerar ett negativt tal om `a` kommer före `b` i den ordning som implementeras
- `a.compareTo(b)` returnerar 0 om `a` och `b` är lika
- `a.compareTo(b)` returnerar ett positivt tal om `a` kommer efter `b` i den ordning som implementeras

(Alltså ekvivalent med t.ex. hur `strcmp()` fungerar i C.)

Eftersom `Foo instanceof Comparable<Foo>` alltid gäller så kan instanser av `Foo` endast jämföras med andra instanser av `Foo` med hjälp av `compareTo()`.

Du behöver alltså minst två arrayer och en maxkapacitet av olika element.

3.2 Beteende

Multimängden skall stödja följande operationer:

1. Konstruktor som tar som argument maximalt antal olika element
2. Konstruktor som inte tar några argument utan delegerar till den första konstruktorn med 16 som maximalt antal olika element
3. `public void add(T elem)` som lägger till en förekomst av `elem` i multimängden om utrymme finns. Huruvida ett element är medlem eller inte avgörs med hjälp av `compareTo()`.
4. `private int indexOf(T elem)` som returnerar platsen i `elements` där `elem` finns, alternativt *borde ha funnits* om elementet fanns med. För att kunna skilja mellan *finns* och *borde ha funnits* använder vi negativa index för det senare, och eftersom `0` är varken positivt eller negativt ökar vi negativa index med 1. Alltså: `-5` betyder att "om det sökta elementet hade funnits i `elements` skulle det ha funnits på index 4" medan `4` betyder att "det sökta elementet finns på plats 4". På samma sätt betyder `-1` "skulle ha funnits först i `elements`", och `0` "finns först i `elements`".
5. `public int contains(T elem)` som returnerar antalet gånger som `elem` finns i multimängden, alltså 0 om elementet saknas.
6. `int size()` som returnerar storleken hos multimängden, alltså totala antalet förekomster av element, initialt 0. (T.ex. har multimängden {1, 3, 3, 3, 5, 8, 8} storlek 7.)
7. En jämförelsemetod (vanlig `equals()`). `a.equals(b)` skall returnera `true` om `a` och `b` är alias, eller om `a` och `b` innehåller *identiska* objekt. Observera att det inte går att göra en typsäker version av denna metod! Kompilatorn kommer att klaga på *Note: Multiset.java uses unchecked or unsafe operations.*. **Bry dig inte om detta.**
8. Plus alla andra privata metoder som du kan tänkas behöva.

3.3 Icke-funktionella krav

1. Enda sättet ändra i multimängden skall vara genom `add()`.
2. Ett försök att skapa en multimängd med negativ maxkapacitet, eller anropa `add()` eller `contains()` med `null` skall kasta undantaget `IllegalArgumentException`. (Detta är ett standard-exception som finns.)
3. Vid överskridande av max antal *olika* element skall ett `MultisetFullException` kastas. (**OBS!** Detta måste du själv skriva! Gör det i `Multiset.java` och "fort" – annars kan inte ens `Driver.java` kompilera.)
4. Du skall använda dig av `indexOf()` i implementationen av `add()` för att ta reda på om det nya elementet redan finns och om inte – var det skall placeras.

3.4 Testning

Du kan testa din inlämning med hjälp av `make`. Då körs programmet i `Driver.java` som skapar några multimängder och utför några få operationer på dem. Vid eventuella fel skrivs ett meddelande ut.

`make test` kör alla test utan att avbryta vid fel. `make test-hard` kör alla test och avbryter vid fel.

Footnotes:

[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.

Author: Lars-Henrik
Created: 2019-08-30 Fri 04:01