Kodprov 2018-08-30

Danger

As long as this note remains on this page, any information **is subject to change** and broken links, etc. are to be expected. Please be restrictive in reporting any errors for now.

1 Instruktioner

Öppna en terminal och kör följande kommandon:

1. `cd` (detta tar dig till din hemkatalog)
2. `mkdir kodprov 180830`
3. `cd kodprov 180830`
4. `curl http://wrigstad.com/ioopm/ mandel.zip -o k.zip`
5. `unzip k.zip`

Nu har du fått ett antal filer och kataloger:

Objekt	Beskrivning
`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 `kodprov 180830`. Om du har tappat bort dig i filsystemet kan du skriva `cd; cd kodprov 180830`. 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 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.

Lycka till!

---

2 C-uppgift

2.1 En cirkulärlänkad lista

Denna uppgift går ut på att implementera en cirkulärlänkad, dubbellänkad lista.

En dubbellänkad lista är en lista där varje nod har en pekare, inte bara till sitt nästa element, utan också till sitt föregående. En sådan lista är lämplig att använda när man vill kunna traversera listan i "båda riktningar".

En cirkulärlänkad lista är en lista som inte termineras genom att den sista länkens `next`-pekare pekar på `NULL`, utan där den sista länken pekar på den första länken. Följande loop skulle i en cirkulärlänkad lista, alltså inte terminera:

```
while (cursor)
  {
    printf("%d", cursor->element); // Skriv ut elementet
    cursor = cursor->next; // Gå vidare till nästa element
  }
```

En lista som är både dubbellänkad och cirkulärlänkad går alltså att traversera åt båda hållen och termineras **icke** av `NULL` i någondera riktning. För enkelhets skull kommer listans element att vara heltal.

Din uppgift är att skriva färdigt programmet `list.c` så att det implementerar alla funktionerna i `list.h` och passerar testerna i `driver.c`. Vad du måste göra är klart markerat med `///TODO` i `list.c`.

Listing 1: list.h (utdelad i kodprovet)

```
#ifndef __kodprov__
#define __kodprov__

#include <stdbool.h>

typedef struct list list_t;

/// Skapar en tom lista
list_t *ioopm_list_create();

/// Tar bort en lista ur minnet
void ioopm_list_destroy(list_t *list);

/// Stoppar in ett element FÖRST i listan
void ioopm_list_prepend(list_t *list, int element);

/// Stoppar in ett element SIST i listan
void ioopm_list_append(list_t *list, int element);

/// Tar bort elementet på plats index. Negativa
/// index räknas bakifrån, dvs. -1 är sista elementet.
/// Valida index är [-N,-1] och [0,N) för en lista med N element.
/// \returns true om elementet togs bort.
bool ioopm_list_remove(list_t *list, int index);

/// Returnerar elementet på plats index. Negativa
/// index räknas bakifrån, dvs. -1 är sista elementet.
/// Valida index är [-N,-1] och [0,N) för en lista med N element.
/// \returns en pekare till int:en i listan -- NULL om den inte fanns
int *ioopm_list_get(list_t *list, int index);

/// Kontrollerar om en lista innehåller ett viss element
/// \returns true om elementet finns i listan.
bool ioopm_list_contains(list_t *list, int element);

/// OBS! Implementeras i O(n) tid, inte O(1).
/// \returns antalet element i listan.
int ioopm_list_size(list_t *list);
#endif
```

2.1.1 Tips 1: sentinel i listan

Om du implementerar din länkade lista så att den alltid har en nod, som är tom – en såkallad sentinel – som första-nod, förenklar du implementationen avsevärt (eftersom du inte behöver göra några `NULL`-test för en tom lista). Då kan prepend E implementeras som "stoppa E efter första-noden" och append E som "stoppa E före första-noden". Sentinel-noden går inte att ta bort, utan försvinner först vid destroy.

2.1.2 Tips 2: rita!

Detta är verkligen en uppgift där det lönar sig att rita upp vad som händer!

2.1.3 Icke-funktionella krav

1. Insättning med append och prepend skall ske i $O(1)$ (uppnås i och med att det är en länkad lista och inte en arraylista).
2. Iteration baklänges och framlänges skall vara lika effektiva och ske i $O(n)$ där n är antalet element (uppnås om listan är dubbelänkad).
3. Interna pekare mellan noder i listan får inte vara `NULL` (uppnås om listan är cirkulärlänkad).
4. `ioopm_list_size()` skall implementeras genom att samtliga länkar räknas. Det skall alltså *inte* sparas en storlek på listan någonstans som `ioopm_list_size()` returnerar.
5. Ditt program får inte läcka minne. Efter att `ioopm_list_destroy()` har körts på en lista skall listan vara helt borttagen ur minnet. Vidare skall `ioopm_list_remove()` städa bort noder ur minnet löpande.

2.1.4 Testa din lösning (funktionella tester)

Du kan kompilera din lösning mot `driver.c` som utför enklare operationer på listan och rapporterar eventuella funna fel. Detta gör du med hjälp av den medföljande makefilen enligt följande:

`make test`

kompilerar driver.c och list.c till a.out och kör

`make memtest`

som make test, men kör i valgrind för att hitta läckage

Danger

Du får inte ändra i någon annan fil än `list.c` eftersom det är den enda fil som lämnas in.

---

3 Java-uppgift

Uppgiften går ut på att implementera en *mängd* (eng. set) av godtyckliga element som alla implementerar interfacet `CompareTo`, t.ex. $\{1, 3, 5, 8\}$ eller $\{$ "abba", "cat empire", "donovan" $\}$. Man kan lägga till element i mängden (upp till en viss storlek) och fråga mängden om ett visst element är medlem i den eller inte.

OBS! Skriv *all* kod i filen `Set.java` som du måste skapa själv.

3.1 Tillstånd

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

Elementen i arrayen 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: Set.java uses unchecked or unsafe operations.* Det är OK. Det går förstås också bra – eller bättre! – att ha en array av `Comparable[]`.

Klassen `Set` är deklarerad enligt följande:

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

dvs. typen `Set<Foo>` betyder en mängd 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 en array och en maxkapacitet.

3.2 Beteende

Mängden skall stödja följande operationer:

1. Konstruktor som tar som argument maximalt antal element
2. Konstruktor som inte tar några argument utan delegerar till den första konstruktorn med 16 som maximalt antal element
3. `public boolean add(T elem)` som lägger till `elem` i mängden om utrymme finns och `elem` inte redan är medlem. Huruvida ett element är medlem eller inte avgörs med hjälp av `compareTo()`.
4. `private int indexOf(T elem)` som returnerar platsen i arrayen 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 arrayen 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 arayen”, och `0` “finns först i arrayen”.
5. `public boolean contains(T elem)` som returnerar `true` om `elem` finns i mängden, annars `false`.
6. `int size()` som returnerar antalet element i mängden, initialt 0.
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: Set.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 mängden skall vara genom `add()`.
2. Ett försök att skapa en mängd 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 maxkapaciteten skall ett `SetFullException` kastas. (**OBS!** Detta måste du själv skriva! Gör det i `Set.java` och “fort” – annars kan inte ens `Driver.java` kompilera.) Observera att anrop till `add()` med element som redan är medlemmar i mängden inte växer mängden!
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 mängder 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.

Questions about stuff on these pages? Use our Piazza forum.

Want to report a bug? Please place an issue here. Pull requests are graciously accepted (hint, hint).

Nerd fact: These pages are generated using org-mode in Emacs, a modified ReadTheOrg template, and a bunch of scripts.

Ended up here randomly? These are the pages for a one-semester course at 67% speed on imperative and object-oriented programming at the department of Information Technology at Uppsala University, ran by Tobias Wrigstad.

---

[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.