TDDI82 – Tentaregler

Hjälpmedel

Följande får tas med på tentan:

- En bok om c++. För boken gäller följande regler:
  - Kommentarer/noteringar som direkt rör text och exempel på sidan i fråga får finnas i sidmarginalen.
  - Egna sidflikar för att enkelt kunna hitta t.ex. de olika kapitlen är tillåtna.
  - Inga extra ark eller lappar, lösa eller fastsatta, får finnas.
  - Tomma sidor, insidan av pärmarna, försättsblad, etc., får inte innehålla programkod.
- Maximalt ett A4-ark med egna anteckningar (dubbelsidigt)
- Penna för att anteckna under tentan. Ni kommer förses med blanka papper

Följande får INTE tas med:

- Elektroniska hjälpmedel såsom miniräknare, mobiltelefon och smartklockor.

Utloggning

När du är nöjd med ditt betyg (som står i tentaklienten) kan du avsluta och logga ut som vanligt i menyn. Klicka sedan på ok följt av knappen avsluta tentan. Observera att när du gjort detta inte längre kan logga in igen. Lämna inte din plats innan vanliga inloggningsskärmen syns.

Frågor om uppgifter

Frågor om tentan i stort eller uppgiftspecifika frågor ska ställas via tenta-klienten. Detta för att vi ska ha en historik av konversationen samt för att vi ska kunna ge samma hjälp till olika studenter.

Systemfrågor

Om du har systemfrågor som t.ex. problem med tentaklienten eller terminalen räcker du upp handen så kommer en assistent och hjälper till.

Tentaregler

Tentan består av fyra uppgifter indelade i två kategorier; standardbibliotek (STL) och mallar. För godkänt betyg på tentan krävs godkänd lösning av en uppgift inom vardera kategori. För högre betyg krävs lösning av uppgifter inom en viss tid enligt tabell 1.

Tid (h)	Antal uppgifter		Betyg
	STL	Mallar	
2.5 + B	1	1	5
4 + B	2	1	5
4 + B	1	2	5
4 + B	1	1	4
5	1	1	3

Tabell 1: Tidsgränser för olika slutbetyg, B är bonus från labserien (se nedan)

För att en uppgift ska anses godkänd krävs följande:

- att man noga följt alla instruktioner och krav ställda i uppgiften
- din kod följer god programmeringsstil (se labseriens rättningsguide)
- att din kod har bra inkapsling och resurshantering
- att standardbibliotekskomponenter används på ett bra sätt

Bonus från labserien

Bonus från labserien (benämnd B i tabell 1) ger ett visst antal minuter (15 eller 30) extra på respektive tidsgräns för högre betyg. Bonusen är endast giltig det år den erhölls.

C++ referens

Det finns tillgång till valda delar av `cppreference.com`. Du måste starta webbläsaren via ikonen ”Web access” på skrivbordet.

Alias för kompilering

Det finns tre alias att använda sig av för kompilering med c++17:
`g++17` Kompilering utan varningar.
`w++17` Rekommenderas!
`e++17` Kompilering med alla varningar som fel.

Uppgift 1 - Mallar

I denna uppgift ska du skapa en generell mallfunktion vid namn `median()` som tar emot en godtycklig behållare och ett godtyckligt funktionsobjekt eller lambda. `median()` ska sedan använda funktionsobjektet som en jämförelsefunktion vilken används för att sortera behållaren och sist returnera medianen. Se exempel i `median.cc`.

Notera att medianen beräknas genom att sortera en datamängd och sedan tar man det elementet som ligger i mitten (d.v.s. `index = size / 2`). Som känt kan sortering göras på olika sätt, och beroende på sorteringskriterium varierar medianen.

Krav: Du får *inte* använda `auto` i denna uppgift.

Krav: Du får *inte* sortera den ursprungliga behållaren.

Tips: Ett funktionsobjekt är en godtycklig typ vilket betyder att det kan tas som en vanlig mallparameter. Använd gärna en lämplig sorteringsfunktion från STL.

Det finns givna testfall i `median.cc`, modifiera den givna koden så att den inte tar onödiga kopior av stora datatyper.

Uppgift 2 - STL

I fabriker är det vanligt att man har system som övervakar potentiella fel i en produktionslinje. Det finns många sätt att detektera att något *potentiellt* är fel, men i den här uppgiften görs det genom att analysera densiteten på produkter i följd.

I denna uppgift ska du använda lämpliga STL algoritmer för att skriva ett program som detekterar eventuella fel i en given produktionskedja. I filen `factory.txt` finns det en lista av produkter. Produkterna förekommer i den ordningen som de har producerats, d.v.s. de första tre fälten är den första produkten, fält fyra till sex är andra produkten, o.s.v.:

```
<namn> <vikt> <volym>
```

Notera att rader har ingen betydelse. I filen `factory.cc` finns en given `Product` struct tillsammans med en funktion `density()` som beräknar densiteten av produkten. Det finns även tillhörande in- och utströmsoperatorer för `Product`.

Det förväntas att densiteten av produkterna i produktionslinjen faller inom ett visst intervall (detta intervall anges av användaren i början av programmet). Om `n` stycken produkter i följd faller utanför intervallet ska programmet rapportera att det finns potentiella fel i produktionen. `n` anges också av användaren i början av programmet.

Om en felaktig sekvens hittas ska programmet skriva ut alla de `n` felaktiga produkterna. Dessutom ska indexet av den första av de funna felaktiga produkterna skrivas ut. Om inga fel hittades ska ett meddelande skrivas ut.

I denna uppgift får inga loopar eller rekursion förekomma. Du får endast använda STL algoritmer ur följande lista:

`distance()` `rotate()` `copy()` `copy_if()` `copy_n()`

`find_if()` `search_n()` `transform()` `adjacent_find()`

Notera att `distance()` strikt taget inte är en algoritm, men inkluderas för tydlighetens skull.

Körexempel (fetstil är användarinmatning)

```
$ ./a.out
Enter the lower density tolerance: 1000
Enter the upper density tolerance: 1500
Enter the sequential fault threshold: 3
Found a faulty sequence after 5 products: Banana [0.4 kg, 0.00045 m^3]
Watermelon [8 kg, 0.013 m^3] Orange [0.45 kg, 0.0003 m^3]
```

```
$ ./a.out
Enter the lower density tolerance: 0
Enter the upper density tolerance: 10000
Enter the sequential fault threshold: 10
No faults found
```

Uppgift 3 - STL

I webshoppar har kunder oftast en varukorg med köpta produkter. I denna uppgift ska du läsa kunders varukorgar från filen `customers.txt` och representera datan med en lämplig STL behållare.

Varje rad i `customers.txt` börjar med kundens namn följt av namnen på produkterna denne har i sin varukorg. Notera alltså att varje rad motsvarar en varukorg. Det finns inga begränsningar på antalet produkter i varukorgen.

Du måste ha ett uppslagsverk i ditt program med produkter och deras priser, vilka dessa ska vara hittas i filen `shop.cc`. Detta uppslagsverk ska representeras av en lämplig *konstant* behållare.

Huvudprogrammet ska börja med att läsa in alla kunders varukorgar från filen `customers.txt`, sedan ska användaren kunna mata in ett namn på en kund. Om kunden har en befintlig varukorg (d.v.s. om den finns i filen) så ska alla produkter skrivas ut följt av totala priset. Om kunden inte finns ska detta skrivas ut. Efter detta ska användaren kunna mata in ett nytt namn och repetera processen. Programmet ska avslutas med `ctrl+D`.

Krav: Det ska *enkelt* gå att slå upp en kunds varukorg, utan att behöva loopa igenom alla kunder.

Körexempel (fetstil är användarinmatning)

```
$ ./a.out
Enter name (ctrl+D to exit): Malte
Your cart contains:
Apples
Bananas
Watermelon
Your total is: 45
Enter name (ctrl+D to exit): Christoffer
Your cart contains:
Oranges
Grapes
Blueberries
Watermelon
Your total is: 90
Enter name (ctrl+D to exit): Pontus
This person isn't shopping!
Enter name (ctrl+D to exit): <ctrl+D>
```

Uppgift 4 - Mallar

I denna uppgift ska du skapa en klassmall vid namn `Split_Container` som tar en mallparameter vid namn `Container`. Tanken är att denna klassmall ska användas för att dela upp en behållare i ett antal lika stora grupper.

Konstruktorn till `Split_Container` tar en behållare av typen `Container` och ett heltal `groups` som representerar antalet grupper som behållaren ska delas upp i. Klassmallen ska lagra en kopia av behållaren och ett lämpligt heltal (antingen antalet grupper *eller* antal element per grupp).

`Split_Container` ska ha följande medlemsfunktioner:

- `set_groups()` tar ett heltal och sätter `groups` till detta nya värde.
- `get_group()` tar ett index och returnerar motsvarande grupp som en `std::vector`.
- `size()` returnerar antalet grupper.

Du kan anta att storleken på behållaren är jämnt delbart med antalet grupper, du behöver alltså **inte** hantera fallet när det inte går jämnt ut.

Det finns ett givet testprogram i `split.cc`, ändra det så att den inte tar onödiga kopior.

Exempel: Antag att du har listan:

```
1 3 2 5 4 6
```

Då kan vi dela upp den i 3 grupper såhär:

```
1 3  |  2 5  |  4 6
```

Och i 2 grupper såhär:

```
1 3 2  |  5 4 6
```

Krav: Du måste även modifera `print_groups()` funktionen så att den fungerar för din `Split_Container`.

Krav: Du får ej lagra grupperna, utan de ska beräknas i `get_group()` enligt följande:

```
begin = index * group_size
end   = begin + group_size
```

Där `group_size = container.size() / groups`. Notera att för att detta ska fungera med alla behållare behöver du använda iteratorer från ursprungsbehållaren.

Tips: Använd `std::next()` för att addera ett index på en iterator.