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 den här uppgiften ska du skapa en generell funktionsmall `sum_max()` som tar en mallparameter `It` som representerar en godtycklig iterator. Funktionsmallen tar två iteratorer, `begin` och `end` och beräknar sedan summan av alla element inom detta iterator intervall. Samtidigt som den beräknar summan ska den även hitta det största enskilda elementet i intervallet. Summan av elementen samt värdet av det största element ska sedan returneras som ett `std::pair`.

Exempel: Antag att du har följande behållare: `{ 1, 3, 2 }` då ska `sum_max()` returnera paret `{ 6, 3 }`, då summan är `1 + 3 + 2 = 6` och värdet `3` är störst i intervallet.

Tips: Du kommer behöva plocka ut värdetypen av iteratorn för att kunna skapa lokala variabler. Detta går *inte* att lösa med endast `auto`.

Du ska dessutom skapa en funktionsmall `print_pair()` som tar emot ett `std::pair` vars båda fält är av den godtyckliga typen `T`. Denna ska skriva ut paret på format givet i körexemplet.

Krav: `sum_max()` ska fungera för *alla* behållare som innehåller element som går att jämföra med `operator>` och addera med `operator+`.

Krav: `print_pair()` ska ta emot endast `std::pair` objekt som innehåller två stycken `T`.

Körexempel

Det finns givna testfall i `givna_filer/sum_max.cc`, dessa ska resultera i följande utskrift:

```
==== Testfall #1 ====
The sum is: 10
The biggest(>) element is: 4
==== Testfall #2 ====
The sum is: PONTUS CHRISTOFFER ERIC MALTE NILS EDVIN
The biggest(>) element is: PONTUS
```

Uppgift 2 - STL

Ett problem jag tror vi alla känner igen är att välja en film när det är filmdags, då alla involverade har olika åsikter om vilka filmer som är intressanta (folk har fel smak helt enkelt).

I denna uppgift ska du använda STL behållare för att hjälpa till att lösa detta problem. Det är därför av yttersta vikt att välja lämpliga behållare som gör lösningen på problemet så enkel som möjligt.

Nedan följer en lista på implementationskrav:

- Läs in ett godtyckligt antal personer. En person och dess associerade data skrivs på en rad till `std::cin` enligt följande format: `<namn> <film(er)...>` där `<namn>` representerar ett namn på personen och `<film(er)...>` är noll eller fler namn på filmer. Både namn på filmer och personer separeras med mellanslag, vilket betyder att namnen i sig inte innehåller några mellanslag. För att avbryta inmatningen matar användaren in `ctrl+D`.
- För varje person ska du lagra alla *unika* filmer som har associerats med personen (d.v.s. vilka filmer denne kan tänka sig att se).
- För varje film ska du lagra hur många personer som är intresserad av att se den.
- Skriv ut alla filmer som *varje* person är intresserad av. Detta görs enklast genom att kontrollera vilka filmer vars lagrade värde är lika med mängden personer.

Körexempel

```
On each line, enter a person's name followed by their movie preferences.
To exit, press ctrl+D.
Alice Titanic Avatar Gladiator Coco Inception
Bob Interstellar Gladiator Gravity Up Inception Titanic
Charlie Braveheart Inception Gladiator Titanic Memento Alien
David Inception Jaws Gladiator Titanic Brave
Eve Titanic Skyfall Gladiator Amélie Inception

The candidates for watching:
 - Gladiator
 - Inception
 - Titanic
```

Uppgift 3 - Mallar

I denna uppgift ska du skapa en klassmall vid namn `Ping_Pong` som tar en mallparameter vid namn `T`. Tanken är att denna klassmall representerar en behållare som är *oändligt* stor. Sättet detta uppnås på är att "ping pong:a" fram och tillbaka över en given datamängd. Notera att `T` representerar värdetypen av behållaren.

Exempel: Antag att vi har datamängden `{ 1, 2, 3 }` så kan vi upprepa detta åt höger på följande vis:

```
Data : 1 2 3 | 3 2 1 | 1 2 3 | ...
Index: 0 1 2 | 3 4 5 | 6 7 8 | ...
```

Notera att vi upprepat itererar datamängden men byter håll varje gång vi har itererat hela mängden (bytet har markerats med `|` i exemplet ovan).

Detta ska också vara möjligt att hämta element *bakåt*, med negativa index t.ex.:

```
Data : ... |  1  2  3 |  3  2  1 |  1  2  3 | ...
Index: ... | -6 -5 -4 | -3 -2 -1 |  0  1  2 | ...
```

Krav: `Ping_Pong` ska uppfylla följande krav:

- Datamängden ska sparas med en lämplig behållare `data` i `Ping_Pong`.
- Det ska finnas en konstruktor som kan ta emot en *godtycklig* databehållare och kopiera innehållet till `data`.
- Det ska finnas en `operator[]` som tar emot en `int` index. Denna operator ska ta fram motsvarande element i den oändliga ordningen (se exemplet ovan).
- Det ska finnas en medlemsfunktion `append()` som tar emot ett värde av typen `T` och lägger till detta i slutet av `data`.

Tips: Notera att `p[i] == p[i + 2*data.size()]` för alla index `i`. Detta kan utnyttjas för att förenkla implementationen av `operator[]`, t.ex. genom att använda *modulo*.

Körexempel

I filen `givna_filer/ping_pong.cc` finns det ett antal testfall. Dessa ska **INTE** modifieras. Om de fungerar korrekt ske det generera följande resultat:

```
1 0 0 1 2 3 3 2 1 0 0 1 2 3 3 2 1 0 0 1 2
0 1 2 3 4 4 3 2 1 0 0 1 2 3 4 4 3 2 1 0 0
NILS MALTE ERIC PONTUS PONTUS ERIC MALTE NILS EDVIN EDVIN
NILS MALTE ERIC PONTUS PONTUS ERIC MALTE NILS EDVIN CHRISTOFFER
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
```

Uppgift 4 - STL

I spelet Bingo har man en s.k. bingobricka där man har ett antal nummer. En dragning av ett antal nummer genomförs gemensamt för alla spelare. Om en spelare har något av de dragna nummrerna på sin bricka kryssas dessa i. I denna uppgift ska du göra ett program som hjälper till att identifiera huruvida din bingobricka innehöll *minst* ett av de dragna nummrerna, och vart det första funna nummret på din bingobricka var.

Denna uppgift ska lösas med lämpliga STL algoritmer. I filen `givna_filer/bingo.cc` finns ett start på programmet givet. Din uppgift är nu att slutföra programmet genom att utföra följande steg:

1. Skriv klart den givna inmatningen.
2. Börja med att kontrollera att alla nummer i `numbers` är större eller lika `lower_limit` och mindre eller lika med `upper_limit`. Om minst ett nummer var utanför skrivs ett felmeddelande ut och programmet avslutas.
3. Sortera de vinnande nummrerna enligt avtagande ordning (d.v.s. störst först) och skriv ut dem i den nya ordningen.
4. Hitta den *första* förekomsten av ett vinnande nummer i `numbers`. Detta betyder alltså att programmet ska söka efter den första förekomsten av ett av numrerna från `winning_numbers`. Om ett vinnande nummer inte förekommer, skriv ut ett felmeddelande och avsluta programmet. Annars, skriv ut vilken position (d.v.s. vilket index i `numbers`) som det första vinnande nummret förekom på.

Krav: Inga loopar, rekursion eller `std:;for_each` får användas för att lösa denna uppgift.

Körexempel (fetstilt är användarinmatning)

```
Enter your bingo number limit: 1 10
Enter your bingo numbers: 1 5 22
You had a bingo number outside of the limits
```

```
Enter your bingo number limit: 1 10
Enter your bingo numbers: 1 5 2 10
Enter the winning numbers: 3 6 8
The winnning numbers in a descending sorted order:
8 6 3
No winning numbers were found.
```

```
Enter your bingo number limit: 1 10
Enter your bingo numbers: 1 5 2 10
Enter the winning numbers: 3 10 2
The winnning numbers in a descending sorted order:
10 3 2
First occurrence found at position: 2
```