Regler

Regler - generellt

- Frågor ställs genom Zoom. Klicka på knappen (Ask for Help) så hjälper vi dig så fort som möjligt.
- Slutinlämningen av din kod ska vara skriven enligt god programmeringssed för C++. Det betyder att tekniker som finns för att underlätta användning, utveckling, felsökning och underhåll ska användas.
- Du ska sitta i en ostörd miljö utan andra personer i samma rum. Du ska hela tiden vara uppkopplad och synlig i Zoom. Om du är klar med en deluppgift och har lämnat in den kan du sätta en lapp framför din kamera där det står "Rast". Om du är klar med tentan kan du logga ut.
- Du kommer behöva legitimera dig under tentamen. Ha ditt foto-id redo.
- Var beredd på att i efterhand kunna redogöra för dina svar.
- Om du behöver ta en rast, lämna in de filer du jobbar på just då till "2021-05-31: Rast" i Lisam.
- Alla former av regelbrott medför att din tentamen underkänns direkt.
- Information och dina givna filer kommer att publiceras under tentans gång på följande sida: `https://www.ida.liu.se/~TDDI82/exam/2021/2021-05-31/index.sv.shtml`

Regler - Tider

- De givna filerna publiceras kl 14:30.
- Tentan ska senast vara inlämnad kl 17:30 + eventuell bonustid.

Regler - Bonus

- Bonustid gällar endast under ordinarie tentamen samma år som bonustiden erhölls.
- Varje laboration som blev inlämnad och godkänd i tid ger 15 minuter extra tid på tentamen.
- Den maximala bonustiden är därför 30 minuter extra.

Regler - hjälpmedel

- All kommunikation är förbjuden, undantaget frågor till kurspersonal.
- All form av kopiering eller avskrift är förbjuden.
- Alla källor du tar inspiration av ska redovisas.
- Du får använda `cppreference.com` fritt.
- Du får använda en valfri C++ bok
- Du får använda en A4 sida med anteckningar. Du måste lämna in dina anteckningar i inlämningen "2021-05-31: Anteckningar" i Lisam innan du börjar skriva tentan (går att lämna in fr.o.m kl 13:00).

Regler - betyg

Tentan består av två uppgifter, en för STL och en för Mallar. Varje del bedöms som U, G eller VG. För att få G på en uppgift måste du ha uppfyllt *majoriteten* av kraven som presenteras i uppgiften. Du måste även ha gjort ett försök till att svara på alla frågor i uppgiften. Detta betyder alltså att du kan få godkänt även om du inte har en fullt fungerande lösning.

I tabellen nedan presenteras betygsgränserna:

Betyg	STL	Mallar
3	G	G
4	G	VG
4	VG	G
5	VG	VG

Det krävs också att du fyllt i tentans regelinlämning, vilket bekräftar att du läst tentans regler och tänker följa dem.

Regler - inlämning

Inlämning görs via Lisam, på följande sida: `https://studentsubmissions.app.cloud.it.liu.se/Courses/TDDI82_2021VT_YV/`. Du kan även hitta sidan genom att gå in på TDDI82 i `https://lisam.liu.se` och sedan klicka på “Inlämningar” i menyn.

Följande inlämningar kommer vara tillgängliga vid de specificerade tiderna:

- 2021-05-31: Regler (13:00-17:30)
- 2021-05-31: Anteckningar (13:00-17:30)
- 2021-05-31: Rast (14:30-17:30)
- 2021-05-31: STL Kod (14:30-17:30)
- 2021-05-31: STL PDF (14:30-17:30)
- 2021-05-31: Mallar Kod (14:30-17:30)
- 2021-05-31: Mallar PDF (14:30-17:30)

I “2021-05-31: STL Kod” lämnar du in koden till din lösning för STL-uppgiften och i “2021-05-31: STL PDF” lämnar du in dina svar på frågorna till STL uppgiften som en PDF fil. Likadant för Mall-uppgiften.

Regelinlämning

Innan du börjar arbeta på första uppgiften måste du göra en inskickning till “2021-05-31: Regler” i Lisam (se ovan).

Du kan skicka ett meddelande via Lisam där du bekräftar att du har läst reglerna och att du tänker följa dem.

Kom ihåg att om du har en sida med anteckningar måste du skicka in denna i “2021-05-31: Anteckningar” i Lisam. Om det är datorskrivet kan du skicka in dem som de är. Om de är handskrivna kan du antingen skanna in dem eller ta en bild.

Gör detta innan du börjar jobba på tentan!

Tidsplan

Vi rekommenderar följande tidsfördelning på uppgifterna:

- STL: 45-60 minuter för att skriva kod
- STL: 30-45 minuter för att skriva svar på frågorna
- Mallar: 60 minuter för att skriva kod
- Mallar: 30 minuter för att svara på frågorna

Eventuell bonus rekommenderar vi att du använder som marginal för att fixa till småproblem eller för att förfina svaren.

STL

Uppgift

I denna uppgift ska du använda STL algoritmer. Du ska undvika loopar i den mån du kan. Din uppgift är att skapa ett program som utför följande steg:

Notera: Steg 3, 6 och 7 varierade per person under tentan. Alla möjliga alternativ presenteras nedan.

1. Skapa en `std::vector<std::string>` vid namn `data`.
2. Fyll `data` med ord (separerade med blanksteg) från `std::cin` tills användaren trycker `ctrl+D`.
3. - Ta bort alla ord som börjar med stor bokstav. Tips: `std::isupper` kan vara användbar.
   - Ta bort alla ord som är kortare än 3 tecken.
   - Ta bort alla ord som börjar och slutar på samma tecken.
   - Ta bort alla ord vars längd är udda.
   - Ta bort alla ord som *inte* slutar på en bokstav. Tips: `std::isalpha` kan vara användbar.
   - Ta bort alla ord som börjar med `'a'` eller `'A'`.
4. Om alla ord togs bort i förra steget, avsluta då programmet med ett lämpligt meddelande.
5. Summera längderna av alla ord i `data`. Dela sedan summan med antalet ord för att få ut genomsnittet. Resultatet ska sparas i en `double` variabel.
6. - Dela listan i två delar, med alla tal som är kortare än genomsnittet till vänster och de som är större till höger.
   - Ersätt alla ord som är kortare än genomsnittet med det första ordet i listan.
   - Skriv ut antalet ord som är längre än genomsnittet.
   - Skriv ut det första ordet som är mindre eller lika med genomsnittet.
   - Lägg till tecknet `'*'` i slutet av alla ord som är lika långa som genomsnittet.
7. - Sortera orden i stigande ordning med avseende på längden.
   - Sortera orden i fallande ordning, d.v.s. från Z till A.
   - Sortera orden i stigande ordning med avseende på det sista tecknet i varje ord.
8. Skriv ut varje element i `data` på varsin rad.

Notera att det är ditt ansvar att visa på hur väl du kan standardbiblioteket. Det är därför viktigt att du demonstrerar så mycket som bara möjligt. Det är bättre att ha utkommenterade delar som är nästan rätt än att ha en fungerande loop.

Frågor

Svara på ***ALLA*** frågor nedan angående din lösning. Skriv dina svar i ett separat PDF dokument. Detta dokument ska som mest vara 1000 ord långt.

1. Redogör för vilka algoritmer du använde för att implementera steg 3 och steg 6. Förklara *varför* dessa algoritmer är lämpliga. Notera att kodexempel kan underlätta, men det är inte ett krav.
2. Algoritmer som ska "ta bort" element från en databehållare flyttar ofta de borttagna elementen till slutet av databehållaren. Varför gör den så istället för att bara ta bort elementen direkt? Diskutera vad du tror.
3. Vad för fördelar finns det med att använda lambda funktioner istället för vanliga funktioner när vi anropar algoritmer? Finns det något tillfälle då vanliga funktioner inte går att använda? Diskutera.

Svara på frågorna ovan så gott du kan. Vi letar inte nödvändigtvis efter ett specifikt svar utan vi vill se din tankegång.

Betyg

Denna uppgift har inget speciellt du behöver göra för VG. Vi kommer istället bedöma enligt följande kriterier:

G: Uppgiften löst med flera algoritmer utan allvarliga brister. Rimliga svar på frågorna.

VG: Uppgiften löst fullt ut med algoritmer utan misstag. Rimliga svar på frågorna med djupa resonemang och god motivering.

Detta innebär alltså att du måste lösa uppgiften enligt din bästa förmåga och svara på frågorna för att få chans till VG.

Mallar

Uppgift

I den givna filen `mallar.cc` implementeras klasserna `Map_Filter` och `Remove_And_Increase`.

`Map_Filter` är en klass som tar emot en databehållare och lagrar `begin` och `end` iteratorerna för denna behållare. Den tar också emot en instans av `Remove_And_Increase` (beskrivs längre ned).

`Map_Filter` har två publikt tillgängliga funktioner `next()` och `done()`. Tanken är att vi använder den här klassen för att iterera igenom de två iteratorer som klassen tog emot, *men* med skillnaden att den hoppar över vissa av elementen och transformerar om de element som inte hoppas över. Varje gång vi anropar `next()` så får vi nästa element, och detta gäller fram tills `done()` är klar.

Klassen `Remove_And_Increase` innehåller beteendet för `Map_Filter`. D.v.s. den specificerar vilka element som ska hoppas över och på vilket sätt varje kvarvarande element ska förändras.

Tanken med `Remove_And_Increase` är att den hoppar över alla element som är lika med ett annat element. Därför måste `Remove_And_Increase` ta emot det värde `target` som ska skippas.

`Remove_And_Increase` har två funktioner för att uppfylla detta:

valid är en funktion som tar emot ett element och returnerar `false` om elementet ska skippas. Det ska alltså bli `false` om det elementet är lika med `target` och `true` annars.

Man kan också se det som att `valid` returnerar `true` om elementet ska vara med i sekvensen av element som erhålls från `Map_Filter`.

transform är en funktion som beskriver hur varje giltigt värde ska förändras. Den tar emot ett element, adderar 1 till det och returnerar sedan resultatet.

Som det ser ut just nu i den givna koden så hanterar `Map_Filter` endast `std::vector<int>` och `Remove_And_Increase` kan endast handskas med `int`.

För att bli **godkänd** på denna uppgift måste du:

- Göra om `Map_Filter` så att den kan hantera alla typer av databehållare, inte bara `std::vector`. Detta betyder att `Map_Filter` är en databehållare som tar mallparametern `Container`.
- Skriva om `Remove_And_Increase` så att den handskas med en godtycklig datatyp `T`. Vi antar dock att `T` går att addera med `1`.
- Uppdatera `value_type` och `iterator` i `Map_Filter` så att de kommer från `Container` istället för `std::vector<int>`.

Testa din lösning genom att lägga till eller ändra testerna i `main` (du kan återanvända de givna testerna för olika datatyper).

Notera att det finns frågor under VG delen som du måste svara på för godkänt.

VG

Varför har vi brutit ut beteendet i två klasser? Jo, för att vi vill göra det möjligt för användaren att byta ut `Remove_And_Increase` mot en annan klass som ger ett nytt beteende, utan att behöva skriva om hela `Map_Filter` klassen.
För att få VG på denna uppgift så måste du:

1. Lägga till ytterligare en mallparameter `Policy` som ersätter `Remove_And_Increase`.
2. Skapa en ny klassmall vid namn `Negative_To_Positive` som tar emot en datatyp `T` (precis som `Remove_And_Increase`).
3. Lägga till *minst* ett testfall där du använder `Map_Filter` tillsammans med `Negative_To_Positive`.

Detta betyder att du ska modifiera `Map_Filter` så att det går att byta ut `Remove_And_Increase` mot något annat. Sen ska du implementera ett exempel på en ny klass som kan ändra beteendet på `Map_Filter` genom att skapa klassen `Negative_To_Positive`.
`Negative_To_Positive` ska göra att `Map_Filter` hoppar över alla element som är positiva och förvandlar alla negativa element till positiva element. Det är OK att anta att `Negative_To_Positive` endast används med heltals- och flyttalstyper.

Frågor

Svara på ***ALLA*** frågor nedan angående din lösning, både för G och för VG. Skriv dina svar i ett separat PDF dokument. Detta dokument ska som mest vara 1000 ord långt.

1. Varför måste vi ta emot `Remove_And_Increase` som en parameter till `Map_Filter`? Förklara.
2. Vad ändras om vi ändrar `Map_Filter` så att den tar emot iteratorer istället för en behållare? Finns det några för- respektive nackdelar med denna ändring?
3. Ser du några fördelar med `Map_Filter` jämfört med algoritmerna `std::transform` och `std::remove_if`? Om inte, redovisa minst två nackdelar med `Map_Filter` istället.

Svara på frågorna ovan så gott du kan. Vi letar inte nödvändigtvis efter ett specifikt svar utan vi vill se din tankegång.