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 "2021-08-25: 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/2021/2021-08-25/index.sv.shtml`

Regler - Tider

- De givna filerna publiceras kl 09:00.
- Tentan ska senast vara inlämnad kl 12:00.

Regler - Bonus

- Bonustid gällar endast under ordinarie tentamen samma år som bonustiden erhölls (**Bonustid gäller alltså inte under omtentamen**.
- 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 "2021-08-25: Anteckningar" i Lisam innan du börjar skriva tentan (går att lämna in fr.o.m kl 07:00).

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:

- 2021-08-25: Regler (07:00-12:00)
- 2021-08-25: Anteckningar (07:00-12:00)
- 2021-08-25: Rast (09:00-12:00)
- 2021-08-25: STL Kod (09:00-12:00)
- 2021-08-25: STL PDF (09:00-12:00)
- 2021-08-25: Mallar Kod (09:00-12:00)
- 2021-08-25: Mallar PDF (09:00-12:00)

I “2021-05-31: STL Kod” lämnar du in koden till din lösning för STL-uppgiften och i “2021-08-25: 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 “2021-08-25: 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 “2021-08-25: 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

Tidsplan

Vi rekommenderar att du lägger 45-60 minuter på att skriva kod och att du sedan lägger resten av tiden på att svara på frågorna.

Uppgift

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

1. Skapa en `std::vector<double>` vid namn `data`.
2. Fyll `data` med flyttal från `std::cin` tills användaren trycker `ctrl+D`.
3. Beräkna medelvärdet av de inlästa flyttalen och spara resultatet i variabeln `x`. Du kan anta att `data` har *minst* ett värde.
4. Om `x` är `0.0`, sätt `x` till `1.0`.
5. Utför följande operation på varje heltal `n` i `data`: `(n - x) * (n + x)`
6. Ta bort alla dubbletter från `data`.
7. 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 två olika algoritmer du använde i din lösning (minst en av dessa måste vara algoritmen du använde för att lösa steg 6). Förklara *varför* dessa algoritmer är lämpliga. Notera att kodexempel kan underlätta, men det är inte ett krav.
2. Finns det några för- och nackdelar med att standardalgoritmer använder iteratorer istället för att direkt jobba med behållare? Diskutera *varför* dessa är för- respektive nackdelar.
3. Varför kan inte alla algoritmer användas med alla typer av iteratorer? Ge ett exempel och redogör vilka iteratorer som den algoritmen inte fungerar för.

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

Tidsplan

Vi rekommenderar att du lägger ungefär 60 minuter på att skriva kod och 30 minuter på att svara på frågorna. Om du får ont om tid är det bättre att du skriver bra svar på frågorna än att du lägger tid på att göra klart koden.

Uppgift

I den givna filen `mallar.cc` implementeras en klass `Repeater` som innehåller en inre klass `iterator`. Det implementeras också en funktion `repeater`.

Repeater är en klass som innehåller en referens till en `std::vector<int>` och en räknare `count`. Hela poängen med den här klassen är att den exponerar en iterator som tillåter användaren att loopa igenom samma behållare flera gånger (`count` gånger).

Exempel: Antag att `count` är 2 och vektorn är `{1, 2, 3}`. Då kommer iteratorn i `Repeater` att loopa igenom värdena `1 2 3 1 2 3`.

iterator innehåller de grundläggande sakerna som behövs för att kunna implementera funktionaliteten för upprepning.

repeater är en funktion som tar emot en databehållare (`std::vector<int>` just nu) och en `count` och sedan returnerar ett `Repeater` objekt initierat med dessa parametrar.

Idén här är att vi ska kunna loopa igenom samma behållare flera gånger i en och samma `for`-loop. Just nu fungerar den endast för `std::vector<int>`, men vi vill att den ska fungera för alla databehållare med alla olika värdetyper. `Repeater::iterator` är inte heller en komplett iterator.

För att bli **godkänd** på uppgiften måste du uppfylla följande krav:

- `Repeater` ska vara en klassmall som tar emot en godtycklig databehållartyp `Container` som mallparameter.
- `iterator` ska vara en inre klass i `Repeater`.
- `iterator` ska innehålla en inre typ vid namn `It` som är iterator-typen av `Container`.
- `repeater` ska vara en funktionsmall som tar emot en godtycklig databehållare och returnerar ett `Repeater` objekt kopplat till den databehållare som skickades in.

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

Som nämndes tidigare så är den givna iteratorn inte komplett. För att få VG på denna uppgift så måste du göra den komplett genom att lägga till postfix inkrement operatorn (`it++`). Du måste även lägga till alla typalias som krävs (bland annat `value_type` och `iterator_category`). Dessa måste representera rimliga datatyper (vissa av dem kan vara samma typer som de är i `Container::iterator`). Sist så måste avrefereringsoperatorn returnera en referens till det underliggande elementet.

För att få VG måste du uppfylla följande krav:

- `Repeater::iterator` får **INTE** vara en klassmall.
- `iterator` ska implementera alla typalias som krävs av iterator-konceptet: `https://en.cppreference.com/w/cpp/named_req/Iterator`
- `iterator` ska ha en postfix inkrement operator (`it++`).
- Det ska vara möjligt att modifiera det avrefererade värdet från avrefereringsoperatorn (`*it`) d.v.s. den ska returnera en referens till det underliggande elementet.

Det finns ett antal testfall för detta i den givna koden i `mallar.cc`.

Tips: Fundera på vilken iterator kategori som `Repeater::iterator` tillhör. Vilka krav uppfyller den? Varför kan den inte (nödvändigtvis) tillhöra samma iterator kategori som `Container::iterator`?

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. Vad tycker du är bra, respektive dåligt i designen av `Repeater` i din lösning? Du kan fundera på (men är inte begränsad till) läslighet, användbarhet och skalbarhet (hur lätt det är att utöka funktionaliteten).
2. Varför har vi introducerat `repeater` funktionen? Vad för förbättring introduceras av att vi har denna funktion till vårt förfogande? Diskutera.
3. Vad är det som menas när man säger att “en funktionsmall inte är en funktion”? På vilket sätt skiljer sig en funktionsmall från en funktion? Förklara.

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