

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 "2021-10-20: 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/2021/2021-10-20/index.sv.shtml>

Regler - Tider

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

Regler - Bonus

- Bonustid gäller endast under ordinarie tentamen samma år som bonustiden erhöles (**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 "2021-10-20: 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:

- 2021-10-20: Regler (07:00-12:00)
- 2021-10-20: Anteckningar (07:00-12:00)
- 2021-10-20: Rast (09:00-12:00)
- 2021-10-20: STL Kod (09:00-12:00)
- 2021-10-20: STL PDF (09:00-12:00)
- 2021-10-20: Mallar Kod (09:00-12:00)
- 2021-10-20: Mallar PDF (09:00-12:00)

I “2021-10-20: STL Kod” lämnar du in koden till din lösning för STL-uppgiften och i “2021-10-20: 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 “2021-10-20: 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 “2021-10-20: 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::list<float>` 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 noll eller mindre sätt `x` till 2.
5. Utför följande operation på varje heltal `n` i `data`: $(x - 1) * n / (x + 1)$
6. Ta bort alla negativa värden från `data`.
7. Skriv ut varje element i `data` separerade med mellanslag.

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. Ange två fördelar med att använda STL algoritmer jämfört med egenskrivna loopar. Förklara noggrant varför dessa är fördelar.
2. Varför kan man inte sortera `std::list` med `std::sort` algoritmen? Vad krävs av en behållare för att den ska gå att sortera med `std::sort`? Ge exempel på en behållare (inte `std::vector`) som går att sortera med `std::sort`.
3. 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.

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

`std::set` och `std::map` är som bekant sorterade databehållare. Detta betyder att vissa databehållare har funktionalitet utöver att bara lagra data. I den här uppgiften ska vi skapa en databehållare som har utökad funktionalitet, men på ett annorlunda sätt. Det är nämligen så att i många fall vill vi endast spara data som uppfyller ett visst kriterium.

I filen `mallar.cc` finns det en klass `Constrained_Container` som implementerar en grundläggande skiss på hur en sådan databehållare kan tänkas se ut. Det som saknas i denna implementation är två saker:

1. Just nu är kriteriet hårdkodat (i klassen `Is_Positive`), medan vi vill att detta ska vara utbytbar mot en godtycklig klass med funktionen `check` (se t.ex. `Has_Length`).
2. Den sparar alltid datan som en `std::vector<int>` men vi vill att den ska kunna lagra värden av vilken datatyp som helst. Den ska dessutom kunna lagra denna data i en valfri databehållare. Detta så att användaren exempelvis kan använda en `std::set` om de också vill ha sortering.

Din uppgift är att färdigställa implementation så att den fungerar för alla typer av databehållare (förutom `std::map`). Detta gör du genom att göra `Is_Positive` och `Constrained_Container` till klassmallar. Specifikt så ska `Constrained_Container` ta *exakt* två mallparametrar:

- `Container` representerar vilken datatyp som datan ska lagras i.
- `Condition` representerar vilken klass som definierar kriteriet.

För att få godkänt måste du testa minst två *olika* databehållare samt både `Is_Positive` och `Has_Length`.

`Has_Length` behöver endast fungera för `std::string`, så den behöver inte vara en mall. Se `mallar.cc` för ett exempel på hur detta kan göras.

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

VG

För att få VG på denna uppgift så måste du lägga till en konstruktor som tar två iteratorer från en *godtycklig* behållare (är inte nödvändigtvis samma typ av behållare som `Container`) och stoppar in alla värden som spänns upp av iteratorerna till din `Constrained_Container`. Kom ihåg att alla värden som stoppas in i `Constrained_Container` måste uppfylla kravet som definieras av `Condition`.

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. Hur får vi reda på vad för datatyp `Container` innehåller?
2. Givet:

```
template <typename T>
T read(std::istream& is)
{
    T data;
    is >> data;
    return data;
}
```

Hur anropar vi `read` så att den läser in en `std::string` från strömmen? Varför måste man göra på det sättet?

3. Varför, och på vilket sätt blir anropet till `read` enklare om vi istället skriver såhär:

```
template <typename T>
void read(std::istream& is, T& data)
{
    is >> data;
    return data;
}
```

Tips: När kan kompilatorn avgöra själv vad `T` är? Varför? Varför inte?

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