TDP004 - Tenta

2023-08-22

Regler

- All kod som skickas in för rättning ska kompilera och vara väl testad.
- Inga elektroniska hjälpmedel får medtas. Mobiltelefon ska vara avstängd och ligga i jacka eller väska.
- Inga ytterkläder eller väskor vid skrivplatsen.
- Student får lämna salen tidigast en timme efter tentamens start.
- Vid toalettbesök ska pauslista utanför salen fyllas i.
- All form av kontakt mellan studenter under tentamens gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av assistent, vakt eller examinator under tentamens gång.
- Frågor om specifika uppgifter eller om tentamen i stort ska ställas via tentasystemets kommunikationsklient.
- Systemfrågor kan ställas till assistent i sal genom att räcka upp handen.
- Endast uppgifter inskickade före tentamenstidens slut rättas.
- Ingen uppgift kan kompletteras under tentamens sista kvart.
- En uppgift kan som regel kompletteras tills den är antingen "Godkänd" eller "Underkänd". En uppgift bedöms som "Underkänd" om ingen markant förbättring skett sedan tidigare inlämning.
- Kompilerande kod, fullständig kravuppfyllnad och följande av god stil och goda konventioner enligt god programmeringssed är krav för att en uppgift ska bedömas "Godkänd".

Hjälpmedel	En C++-bok (t.ex. Stroustrup) Ett A4-ark med egna anteckningar

Information

Betygssättning vid tentamen

Tentamen består av ett antal uppgifter på varierande nivå. Uppgifter som uppfyller specifikationen samt följer god sed och konventioner ges omdömet "Godkänd". Annars ges omdömet "Kompletteras" eller "Underkänd". Tentamen kräver två godkända uppgifter för betyg 3. Alla betygsgränser ses i tabell 1 och 2. *För betyg 3 har du alltid hela tentamenstiden, varken mer eller mindre.* (För student som från LiU fått rätt till förlängd skrivtid förlängs betygsgränserna i proportion till den förlängda skrivtiden.)

Tid	Lösta uppgifter	Betyg	Tillgodo
4 h	3	5	inget
2.5 h	2	4	inget
4 h	2	3	inget
4 timmar	1	|	löst uppgift

Tabell 1: Betygsättning vid förtentamen (dugga), 4 uppgifter ges

Tid	Lösta uppgifter	Betyg
3 h $+B$	3	5
4 h $+B$	4	5
4 h $+B$	3	4
2 h $+B$	2	4
5 timmar	2	3

Tabell 2: Betygsättning vid sluttentamen, 5 uppgifter ges

Bonustid ($+B$)

All bonustid gäller endast under den första ordinarie tentamen i samband med kursen (januari). Varje moment i kursen som ger bonus ger 5 minuter extra tid för högre betyg på sluttentamen, upp till maximalt 45 minuter. Detta är markerat med $+B$ i tabellen där bonus räknas.

Tillgodoräknanden

Tillgodoräknanden gäller endast under den första ordinarie tentamen i samband med kursen. Om du på förtentamen endast lyckas lösa en uppgift kan du tillgodoräkna motsvarande uppgift på sluttentamen (se tabell 1). Du har då bara en uppgift kvar till betyg 3. För högre betyg (om du löser mer än en uppgift på förtentamen) kan du inte tillgodoräkna något. Om du på sluttentamen löser en andra uppgift snabbt och vill sikta på högre betyg kan du lösa den tillgodoräknade uppgiften "igen", men den räknas endast mot högre betyg, **inte** som andrauppgift för betyg 3.

Inloggning

Logga in på datorn i labbsalen med ditt liu-id och lösenord. Detta är samma inloggningsuppgifter som du använder i Lisam.

Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö (Mate-session i Ubuntu). Efter en stund kommer även din kommunikationsklient att dyka upp automatiskt. Startmenyn

är nedskalad till enbart det som examinator bedömt relevant. Andra program kan fortfarande finnas tillgängliga genom att starta dem från en terminal. Observera att en del program som använder nätverkstjänster inte fungerar normalt, eftersom nätverket inte är åtkomligt.

När du är inloggad är det viktigt att du har tentaklienten igång hela tiden. Om den inte dykt upp fem minuter efter inloggning och inte heller när du startar den manuellt från menyn (fisken) tar du kontakt med assistent eller vakt i sal.

Terminalkommandon

`e++17` används för att kompilera med "alla" varningar *som fel*.
`w++17` används för att kompilera med "alla" varningar. **Rekommenderas.**
`g++17` används för att kompilera `utan` varningar.
`valgrind --tool=memcheck` används för att leta minnesläckor.

C++ referenssidor

På tentan har du tillgång till referenssidorna på `http://www.cppreference.com/` via webbläsaren Chrome. Starta `chromium-browser` i terminalen eller välj lämpligt alternativ från startmenyn. Observera att allt utom referenssidorna är avstängt. Om du inte kan komma åt en sida du tycker hör till referenssidorna (som kanske blockerats av misstag) kan du skicka ett meddelande via tentaklienten. Tag hjälp av assistent i sal om det inte fungerar.

Givna filer

Eventuella givna filer finns i katalogen `given_files`. Denna underkatalog är skrivskyddad, så det är ingen risk du råkar ändra på dessa filer. Skrivskyddet gör dock att du måste kopiera in de givna filer du vill använda till tentakontots hemkatalog. Hur du listar och kopierar filer ska du kunna. Hemkatalogen står du i från början, och du kommer alltid tillbaka till den genom att bara exekvera kommandot `cd` i terminalen.

Avslutning

När dina uppgiftsbetyg och ditt slutbetyg i kommunikationsklienten stämmer överens med det du förväntar och du är nöjd, eller när tentamenstiden är slut, är det dags att logga ut. Hinner du inte se ditt betyg får du höra av dig till examinator via epost efter tentamen.

Avsluta alla öppna program och logga ut ur datorn. Lämna inte datorn innan du ser att du är utloggad.

Uppgift 1 - Klockslag - klasser och operatorer

Året är 2024 och programmeringsstudenter som löst en klockslagslaboration har till slut tröttnat på hur klockan fungerar. 24 timmar på en dag, 60 minuter på en timme, och 60 sekunder på en minut... Sen ska det hållas på med 12- och 24-timmars klockor också, helt orimligt! Revolutionen var kort och inget blod spilldes. Den nya klockan `Dagklockan` mottogs med öpna armar av massorna. Den nya `Dagklockan` har bara en enhet, en dag, för att uttrycka kortare tider används `deciDag`, `centiDag`, och `milliDag` etc. Mitt på dagen är alltså nu 5 `deciDag`.

Ditt jobb är nu att skapa en klass och tillhörande operatorer för att representera denna nya klocka. Den minsta enheten vi behöver hantera i denna implementation är 1 `millDag` (1/1000 Dag). Utöver konstruktor(er)ska klassen ha:

- Medlemsfunktionen `to_string` som returnerar en strängrepresentation av klockslaget (returnerar en sträng som visar de nuvarande **milliDag**arna).
- Utströmsoperatorn, se körexemplet.
- Unär operator för post- och pre-inkrement.
- Binär operator för addition, både för `klocka + int` och `int + klocka`.

Ingen felkontroll krävs för konstruktor (användaren antas alltid göra rätt).

Krav: Men om vi ökar klockslaget måste vi se till att klockan slår över. Så om klockan är 995 `milliDag` och vi lägger på 5 `milliDag` så ska det nya klockslaget visa 0 `milliDag`.
Krav: Du får inte använda `friend` för att lösa denna uppgift.
Krav: Du ska använda korrekt filuppdelning (.cc och .h eller motsvarande) när du lägger till klassen.

Körexempel:

```
2
3
999
0
```

Tänk på att tiden behöver slå över, om klockan är 999 `milliDag` och användaren inkrementerar klockan med 1 `milliDag` så ska klockan bli 0 `milliDag`.

Uppgift 2 - Tårtdagar

På din nya arbetsplats finns en regel som säger att företaget bjuder på tårta de dagar som firar ett jämt tiotal år sedan arbetstagarnas födsel. Alla på arbetsplatsen väljer att tolka regeln så flexibelt som möjligt, dvs om Anna fyller 28 år på samma dag som Petter fyller 32 så finns skäl att fira med 60-års tårta (28+32). Två år senare blir det förstås tårta igen när Anna fyller 30 år. När arbetsgruppen växer och det är fler och fler som fyller på samma dag börjar det snabbt bli komplicerat att hålla reda på alla tårtdagar och som ny anställd får du uppgiften att skriva ett program som hjälper till hålla reda på när det får (ska!) köpas in tårta. I grundversionen definieras en tårtdag som en dag då (1) minst en anställd fyller jämnt eller (2) summan av alla anställda som fyller på samma dag är jämn. Med "jämn" menas i detta sammanhang "jämnt delbar på 10". (Version två förväntas även räkna fram om något urval av personer som fyller på samma dag summerar jämnt, men detta är överkurs för denna uppgift.) Du har redan fått ihop ett huvudprogram inklusive testdata i `uppg2.cc`, komplettera med klassen `B_Day`. Den ska:

- hålla reda på födelsedatum. Datum felkontrolleras inte.
- kunna jämföras. Två B-days är lika om de infaller på samma datum. Årtalet beaktas inte.
- kunna subtraheras från ett heltal (heltal - B-day). Vid denna subtraktion dras födelseåret från heltalet. Datumet beaktas inte.
- kunna subtraheras med ett heltal (B-day - heltal). Vid denna subtraktion dras heltalet från födelseåret. Datumet beaktas inte.
- kunna konverteras till en std::string via medlemsfunktione `to_string()`. T.ex. 24/8 för 24:e augusti. Årtalet beaktas inte.

Krav: Du inte får använda nyckelordet `friend`, koden ska vara korrekt uppdelad i deklarations (*.h) och implementations (*.cc) fil, och implementationen ska nyttja de funktioner som finns i den mån det är lämpligt. Huvudprogrammet är givet i `uppgift2.cc`.

Körexempel:

```
./a.out | grep Cake
24/8 celebrates 26 and 50 years. Cake!
8/3 celebrates 30 years. Cake!
24/8 celebrates 24 and 50 years. Cake!
17/6 celebrates 30 and 54 years. Cake!
19/7 celebrates 26 and 50 years. Cake!
14/10 celebrates 23 and 60 years. Cake!
28/12 celebrates 30 and 98 years. Cake!
14/10 celebrates 37 and 60 years. Cake!
19/7 celebrates 24 and 50 years. Cake!
29/6 celebrates 30 and 57 years. Cake!
27/5 celebrates 20 years. Cake!
```

Uppgift 3 - Minneshantering

I filen `uppg3.cc` finns en implementation av en länkad struktur. Din uppgift är att komplettera koden så klassen `Queue` blir komplett, implementera de fem speciella medlemsfunktionerna och säkerställa korrekt minnehantering i alla lägen. Utöka också huvudprogrammet så att klassen du skapat testas.

En **Kö** är en datastruktur där man kan lägga till och ta bort data. Den fungerar snarlikt en **Stack** men insättning och borttagning ser lite annorlunda ut. I en **Stack** läggs alltid saker till först och tas alltid bort först. I en **Kö** så läggs saker till sist men tas bort först. Precis som om man ställde sig i vilken kö som helst.

Uppgift 4 - Polymorfi

Objektorienterad system kan byggas upp på många olika sätt. Ett sätt är att representera beteenden som klasser och sedan knyta de beteendena till en annan instans. Exempelvis kan vi tänka oss ett spelobjekt som har en x- och en y-koordinat. Ett beteende man kan koppla till spelobjektet skulle kunna vara att objektets koordinater ska öka med 1. Ditt jobb i denna uppgift är att skapa 4 klasser.

Krav: Du får inte använda friend för att lösa denna uppgift

Object:

- Object har 3 datamedlemmar. En x- och en y-koordinat som är heltal. En vector av `Behaviour*`.
- Medlemsfunktionen `add_behaviour` som saknar returvärde och tar en parameter av type `Behaviour*`. Funktionen lägger till paramtern i vectorn.
- Medlemsfunktionen `simulate` som saknar returvärde och parametrar. Funktionen ska iterera genom vectorn och anropa medlemsfunktionen `execute` på varje beteende vars pekare är lagrat i vectorn.
- Klassen har också publika `setters` och `getters` för x- och y-koordinaten.

Behaviour:

- Detta är en abstract klass som har en medlemsfunktion `execute`. Funktionen tar en `Object&` som parameter och saknar implementation.

Move:

- Klassen härleds från `Behaviour` och överlagrar funktionen `execute`. Funktionen ska öka objektets x- och y-koordinat med 1 (en referens till objektet tas emot som parameter till funktionen).

Display:

- Klassen härleds från `Behaviour` och överlagrar funktionen `execute`. Funktionen ska skriva ut objektets x- och y-koordinat (se körexemplet).

Körexempel:

```
1, 1
2, 2
3, 3
```

Tips: Det finns en utmaning här med cirkulära beroenden, problemet kan delvis lösas genom att bryta upp klasserna i h- och cc-filer, men det är inte nödvändigt eller ett krav för att lösa uppgiften.

Uppgift 5 - STL

Det finns en given indatafil, `data1.txt`, i mappen `given_files`.

I denna uppgift är det speciellt viktigt att använd standardbibliotekskomponenter i så hög grad som möjligt. Komplettera med lambdauttryck eller egna funktionsobjekt, ej vanliga funktioner, om det behövs för att lösa en uppgift. Inläsning och utskrift ska göras enligt följande.

- Programmet ska läsa indata från standardströmmen cin. Läsning från fil kan då göras genom omdirigering på kommandoraden, enligt följande exempel:

  ```
  a.out < given_files/data1.txt
  ```

- Utdata ska skrivas på standardströmmen cout.

Programmet ska göra följande, i den ordning som anges nedan.

1. Läs indata i form av heltal (int) och lagra talen i en lämplig sekvenscontainer, med tanke på vad som ska göras nedan. Ingen ytterligare container får användas i programmet för att lagra och bearbeta data.
2. Skriv ut talen i containern, med ett mellanrumstecken efter varje värde.

   ```
   149 tal har lästs in:
   91 67 108 9 ... 68 82 68 13 31 74 4
   ```

3. Sortera talen i containern i stigande ordning.
4. Ta bort alla dubbletter, så att endast ett värde av varje finns kvar i containern.
5. Skriv ut de nu sorterade, unika talen i containern med ett mellanrumstecken efter varje värde.

   ```
   Unika, sorterade tal:
   1 2 3 4 5 6 7 ... 105 106 107 108 109 110 111
   ```

6. Beräkna hur många tal som motsvarar 5 procent. Vid beräkningen ska resultatet avrundas nedåt till närmast mindre heltal. Om det är 111 tal motsvarar 5% exakt 5,55 tal, vilket alltså ska avrundas till 5.
7. Ta bort de 5% minsta och de 5% största talen ur containern. Om containern till exempel innehåller 111 tal ska alltså de fem minsta och de fem största, sammanlagt tio tal, tas bort. Skriv sedan ut de återstående talen i containern, med ett mellanrumstecken efter varje värde.

   ```
   Efter att de 5% minsta och 5% största talen tagits bort:
   6 7 8 9 10 ... 102 103 104 105 106
   ```

8. Beräkna summan och medelvärdet av talen i containern och skriv ut dessa.

   ```
   Summan av talen är 5656
   Medelvärdet av talen är 56.0
   ```