TDP004 - Tenta

2022-03-18

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 genom skrivbordgenvägen (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 genvägen Webaccess på skrivbordet. 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 - Kö - Klasser och operatorer

Bara ett par årtionden tillbaka i tiden verkade tillväxten av datakraft vara ändlös. Idag ser situationen annorlunda ut. Oavsett finns det omständigheter där det är viktigt att skriva effektiv kod, ett exempel på detta är när man väldigt ofta vill komma åt viss data och mer sällan vill komma åt annan. I denna uppgift ska du skapa en klass som kan användas för att se hur många gånger filer öppnas. Du ska skapa klassen `Optimizer` som har 2 datamedlemmar. Den ena datamedlemmen, `table`, ska vara av typen `std::map` som motsvarar tabellen med fil-idn som nycklar och hur många gånger varje fil öppnats som värden. Den andra datamedlemmen, `limit` ska vara av typen `int` och motsvarar det minsta antalet gånger en fil öppnats för att få vara med i utskriften (se nedan). Följande medlemsfunktioner skall implementeras:

- Konstruktor som tar ett heltal och sätter datamedlemmen `limit` till detta heltal
- Konstruktor som inte tar några parametrar och sätter datamedlemmen `limit` till `0`
- `void set_limit(int)` som sätter datamedlemmen `limit` till värdet som skickas in
- `void read(vector<long unsigned int>)` som tar emot en vector av fil-idn. Medlemsfunktionen ska gå igenom alla fil-idn och lägga till dem som nycklar i datamedlemmen `table` om de inte redan finns och värdet till 1. Om nyckeln redan existerar ska motsvarande värde ökas med 1.
- `void print()` skriver ut alla nyckel-värde-par där värdet är minst lika stort som datamedlemmen `limit` på formatet i körexemplet nedan.

Ditt program ska fungera med det givna huvudprogrammet och skrivas med korrekt filindelning. Du får endast modifiera huvudprogrammet i den utsträckning som krävs för att inkludera dina filer.

Körexempel:

```
File 1: 3
File 2: 2
File 4: 2

File 1: 3

File 1: 6
File 2: 2
File 4: 5
File 5: 1
File 128: 3
```

Uppgift 2 - Stackström - Klasser och operatorer

Strömmar är ett centralt koncept inom C++ (och programmering i allmänhet). Oftast kopplar vi strömmar till olika resurser (t.ex. filer, terminaler, etc.), men detta är inte alltid nödvändigt: vi kan t.ex. skapa strömmar som är kopplade till strängar med `std::stringstream`.

I denna uppgift ska du konstruera en stack som fungerar likt en ström. Skapa en klass vid namn `Stack` som har följande medlemsfunktioner:

- `unsigned int size()` som returnerar hur många element som har lagts till i stacken.
- `std::string to_string()` skapar en strängrepresentation av stacken. Ordningen elementen skrivs ut i ska vara i samma ordning som de är i stacken. Det elementen som ligger högst upp i stacken (lades till sist) är det första elementet i utskriften.

I en vanlig stack skulle vi också lägga till `push()` och `pop()`, men eftersom att stacken ska agera som en ström, låter vi istället `operator<<` och `operator>>` ta dessa roller.

- `operator<<` implementeras som en medlemsfunktion och tar en `int` parameter. Denna måste också returnera en referens till stacken. Denna operator kommer sedan lägga till den inskickade int:en till stacken (tänk push).
- `operator>>` implementeras också som en medlemsfunktion och tar en `int` referens. Även denna måste returnera en referens till stacken. Denna operator sätter den inskickade referensen till det element som ligger överst på stacken och tar sedan bort (tänk pop) det översta elementet i stacken.

Stacken ska fungera med det givna huvudprogrammet utan några modifikationer av main funktionen.

Tips: Du kan internt implementera stacken m.h.a. en `std::vector<int>`.

Körexempel:

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

Uppgift 3 - Gränssnitt för listor - Polymorfi

Ett användningsområde för polymorfi inom datavetenskapen är att definiera gränssnitt. Klasser som utökar gränssnittet måste sedan implementera alla funktioner som specificeras i gränssnittet. I denna uppgift ska du skapa tre klasser: `List`, `Enumerate` och `Description`.

`List` ska vara ett gränssnitt och ska kräva att alla klasser som utökar klassen behöver implementera följande funktioner:

- `void display()`
- `void push_back(std::string)`

`Enumerate` ska implementera gränssnittet i `List` och har en `std::vector<std::string>` som datamedlem. Enumerate är en lista som skrivs ut med index för varje element och implementerar funktionerna enligt följande beskrivning:

- `void display()` skriver för varje element i vectorn ut index och strängen som sparas i vectorn (se körexempel)
- `void push_back(std::string)` tar strängen som skickas in som parameter och lägger till den sist i vectorn

`Description` ska implementera gränssnittet i `List` och har en `std::vector<std::pair<std::string, std::string>>` som datamedlem. Description är en lista av par där första strängen i paret är namnet på ett spel och andra strängen är beskrivningen av det spelet och implementerar funktionerna enligt följande beskrivning:

- `void display()` skriver för varje element i vectorn ut namnet på spelet och beskrivningen av spelet (se körexemplet).
- `void push_back(std::string)` tar strängen som skickats som parameter och delar upp denna i två delar. Den första delen ska vara fram till och med den första förekomsten av ett mellanslag i strängen. Den andra delen är all text efter det första mellanslaget. Dessa 2 delar utgör första och andra delen av det par som ska knuffas in i vectorn. **Tips:** titta på vad det finns för medlemsfunktioner kopplade till std::string, det kanske finns stöd för att hitta positionen för första förekomsten av ett tecken i en sträng och stöd för att ta ut delsträngar.

```
0: Nemesis
1: Gloomhaven
2: VirginQueen

Nemesis:  SciFi survival horror game
Gloomhaven:  Fantasy legacy game
VirginQueen:  GMT wargame set in the Elizabethian era
```

Uppgift 4 - Nytt chiffer - STL

Datorvetenskapen har genom tiderna knäckt och upptäckt många chiffer. Ett vanligt sätt att dölja meddelanden är att blanda ihop olika texter och sedan plocka ut rätt bokstäver (utefter ett antal regler) för att läsa det hemliga meddelandet. I den här uppgiften får du tillgång till ett chiffer som har fyra typer av tecken: versaler(ABC...), gemener(abc...), mellanslag (' ') samt pipetecknet ('|'). Din uppgift är att läsa ut de två gömda meddelanden som finns i texten med hjälp av följande algoritm:

1. Läs in filen `cipher.txt`, tecken för tecken, till en lämplig behållare
2. Dela upp behållaren i två delar genom att lägga alla tecken med ascii-värden under 96 i en behållare och alla tecken med ascii-värden över 96 i en annan.
3. Skriv ut behållaren med tecken under 96 till standardströmen
4. För den andra behållaren behöver pipetecknet('|') bytas ut till mellanslag innan utskrift till standardströmmen

Körexempel:

```
./a.out
HELLO FRIEND
how are you doing today
```

Några ytterligare krav är att uppgiften:

- ska lösas med STL
- får inte innehålla manuella loopar

Tips för inläsning av tecken se: `istreambuf_iterator`

Du får bara använda dessa algoritmer: `copy`, `partition_copy`, `transform`, `rotate_copy`, `replace`, `sort`, `partition`, `accumulate`

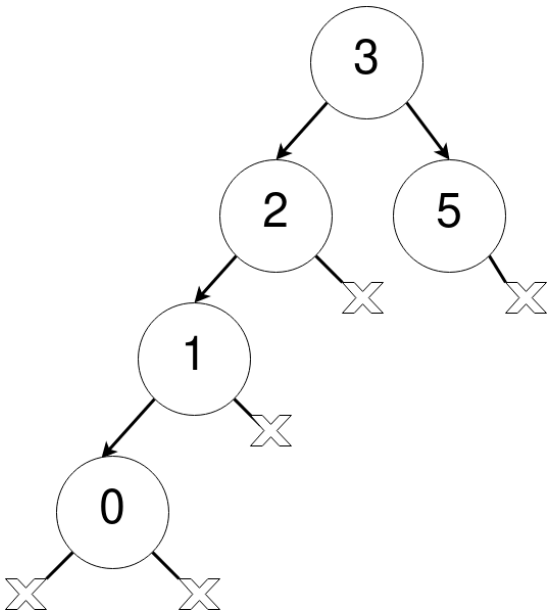
Obs: Det är inte meningen att du ska använda **alla** dessa algoritmer. Välj endast de som är lämpliga för att lösa problemet.

Uppgift 5 - Binärt träd - pekare och minneshantering

Ofta är det praktiskt att hålla data sorterad inom datavetenskap. En struktur man gör detta med är genom binära träd där varje nod i trädet har upp till två pekare till andra noder (ett vänstra och ett högra barn) och ett värde. För varje nod finns det ett antal regler:

- Om nuvarande noden har ett barn i sin vänstra gren så kommer den nodens värde vara mindre eller lika med värdet i nuvarande noden
- Om nuvarande noden har ett barn i sin högra gren så kommer den nodens värde vara större än värdet i nuvarande noden
- Där ett barn saknas ska motsvarande pekare vara satt till `nullptr`

Du ska implementera en klass som motsvarar ett binärt träd och en klass (eller ett aggregat) som representerar en nod i ett sådant träd.



Figur 1: Example tree after inserting 3, 5, 2, 1, 0

Node har 3 datamedlemmar:

- `value` som är värdet noden innehåller och är av typen `int`
- `left` som innehåller pekaren till nodens vänstra barn och är av typen `Node*`
- `right` som innehåller pekaren till nodens vänstra barn och är av typen `Node*`

Tree har 1 datamedlem `root` som är av typen `Node*` och har följande publika medlemsfunktioner:

- Default konstruktor som initierar ett träd där datamedlemmen `root` sätts till `nullptr`
- `void add(int)` som tar in ett värde och sätter in detta värde på lämplig plats i trädet. Det finns en påbörjad implementation av denna medlemsfunktion i den givna filen.
- `void print()` som skriver ut alla värden i trädet. Ordningen detta skall skrivas ut på är genom att utforska trädets vänsterled först. I figur 1 innebär det att 0 skall skrivas ut först, följt av 1, 2, 3, 5. Denna funktion måste lösa problemet rekursivt men kan göra det genom att nyttja en rekursiv stödfunktion (se påbörjade lösningen av `add`.
- Flyttkontstruktor och flyttilldelningsoperator ska implementeras och fungera som förväntat
- Destruktor ska implementeras och fungera som förväntat
- Kopieringskonstruktor och kopieringsoperator ska explicit stängas av.

Körexempel:

```
./a.out
t1: tree: 2, 3, 5,
t2: tree: 1, 2, 3, 5,
```