TDP004 - Tentamen

2018-01-12

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 ($+B$)

Avklarade deadlines ger bonus för högre betyg. Varje bonus från labserien ger 5 minuter extra till gränserna och den totala bonusen symboliseras med B i tabell 2.

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

Innan du loggar in ska du välja "Exam system" från sessionsmenyn i nedre vänstra hörnet av inloggningsrutan. Därpå loggar du in med dina LiU inloggningsuppgifter. Du kommer nu in i tentainloggningen och ska börja med att välja språk. Ta den flagga som ser minst Engelsk ut så blir det Svenska (valet spelar ingen roll för dig som kan båda språken). Följ instruktionerna på skärmen så långt det går tills du ska mata in ett engångslösenord. Tag fram ditt LiU-kort och visa det för assistent eller vakt i sal för att få detta lösenord.

Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö (Mate-session i Linux Mint) med grön skrivbordsbakgrund. 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 *experimentiell* 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. Då lösningen är på experimentell nivå kan det hända att problem uppstår, t.ex. att proxyn inte går att nå. Tag då hjälp av assistent i sal.

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. Hemkatalogen heter alltid `/home/student_tilde` om du undrar.

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 tryck på knappen märkt "Exit" i menyn längst ner på skärmen och välj "ok". Vänta ett tag och tryck sedan på knappen "Avsluta tentamen" när det är möjligt. När detta är gjort är det omöjligt att logga in igen. Lämna inte datorn förrän du ser den vanliga inloggningsskärmen med blå bakgrund. Anmäl till assistent eller vakt om inloggningsskärmen inte dyker upp inom en minut så åtgärdar vi problemet.

Testning av kod

Några av uppgifterna kan ha testfall skrivna med catch. För att ni inte ska behöva kompilera om catch vid varje ändring av er egen kod finns det en given fil `given_files/test_main.cc`. Denna behöver endast kompileras en gång och sedan kan man länka ihop den med de testfall man vill kompilera just nu. Gör följande:

1. Kompilera (men länka inte) `test_main.cc` till en objektfil (skapar filen `test_main.o` i nuvarande katalog):
   `g++17 -c given_files/test_main.cc`
   Observera att detta steg ska endast behöva göras en gång.

2. Vid kompilering av din kod läggs objektfilen till (skapar då en `a.out`):
   `w++17 test_main.o myfile.cc`

Uppgift 1 – Schack

I spelet schack finns det pjäser som rör sig på olika sätt. I denna uppgift ska du skapa en klasshierarki för att representera dessa rörelsemöster samt en klass som har ett rörelsemönster.

Börja med att skapa en abstrakt klass `Movement_Behavior` med en medlemsfunktion `valid_move` som tar emot två rutor och returnerar `true` om det är ett giltigt drag att gå mellan de två angivna rutorna. Skapa två direkta subklasser till `Movement_Behavior`, `Rook_Behavior` och `Pawn_Behavior` för att symbolisera ett torn och en bonde. Ett drag för ett torn är giltigt om det går rakt, antigen radvis eller kolumnvis. En bonde kan gå ett steg framåt (kolumnvis). Vid första draget för en bonde får den även gå två steg framåt.
Skapa även en subklass till `Rook_Behavior`, `Queen_Behavior` för att representera drag för en dam. Utöver att gå rakt får en dam gå diagonalt.

Skapa en till klass, `Chess_Piece`, som initieras med en position och en `Movement_Behavior` och har en medlemsfunktion `move`. `move` tar emot en position och flyttar dit om det är en legal förflyttning enligt det givna rörelsemönstret. Filen `given_files/chess.cc` har ett kort testfall på hur dessa klasser kan tänkas användas samt en enkel klass för att representera en position som du ska använda i din lösning.

Obs: vid förflyttningar utgår vi från att pjäsen är ensam på brädet och att positionerna som anges är innanför brädet.

Uppgift 2 – Sudoku

I pusselspelet sudoku har man en 9x9 stor matris av rutor där man ska fylla i siffrorna 1-9. Om man har gjort det korrekt har varje rad, kolumn och varje 3x3 delmatris siffrorna 1-9. Se figur 1 för ett exempel.

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Figur 1: Ett olöst sudokupussel. De tjocka linjerna markerar delmatriser.

I denna uppgift ska du skapa en klass för att representera ett sudokupussel. Internt ska det lagras som en tvådimensionell array (dvs `array<array<int, 9>, 9>`). Följande operationer ska stödjas:

Konstruktor Det ska gå att initera ett pussel med ett fullständigt bräde (dvs tvådimensionell array). Dessutom ska det finnas en defaultkonstruktor som genererar ett bräde. Genereringen ska, för varje ruta i brädet, slumpa en siffra mellan 0 och 9 (0 motsvarar en tom ruta).

int get(row, col) Ger värdet sparat på rad `row` och kolumn `col`.

void set(row, col, val) Sätter värdet på rad `row` och kolumn `col` till `val`.

bool is_solved() Ger true om det interna brädet är en godkänd lösning (se inledningstexten). Tips: `std::set` kan vara bra att ha här.

Till din hjälp finns det i filen `given_files/sudoku.cc` ett exmepelprogram som har en giltig lösning till ett sudoku.

Uppgift 3 – Taggar

Du har en fil, `given_files/names.txt`, där varje rad innehåller ett namn, ett kolon och flera mellanslagsseparerade taggar (ord). Din uppgift är att skapa ett program som låter användaren mata in ett antal taggar och sedan skriver ut alla namn som har samtliga taggar. Detta ska göras så långt som möjligt med hjälp av standardbibliotekets containrar och algoritmer enligt nedanstående algoritm:

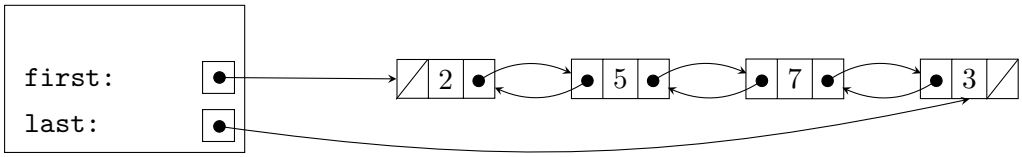
1. Öppna filen för läsning
2. Skapa en map för att koppla en tag till alla namn som har den taggen (alltså rekommenderas typen `map<string, vector<string>>`) kallad `tags`. Observera att taggen alltså är nyckeln.
3. För varje rad i filen, läs in namnet och lägg till det i vectorn som hör samman med varje tag på raden.
4. Sortera alla namn-vektorer
5. Låt användaren mata in taggar, spara dem i en vector
6. Skapa en ny `vector<string>`, `common_names` och initiera den med namnen som hör samman med den första inmatade taggen
7. För alla inmatade taggar utom den första, skapa en ny `vector<string>` där du sparar de namn som förekommer både i `common_names` och bland namnen som hör till aktuell tag. Gör detta med hjälp av `set_intersection`. Skriv sedan över `common_names` med namnen i den nya vectorn.
8. Skriv ut namnen i `common_names`, ett namn per rad.

```
Mata in taggar: MEXICO SWEDEN
Gemensamma namn
SOFIA

Mata in taggar: SWEDEN NORWAY ICELAND FEMALE
Gemensamma namn
ANNA
EMMA
SARA
```

Uppgift 4 – Kö

En kö är en datastruktur där man sparar värden sekvensiellt i den ordning de stoppas in. När man lägger in ett värde läggs det i slutet och när man tar ut ett värde tas det från början (en FIFO-kö, First In First Out). I denna uppgift ska du implementera en kö som en dubbellänkad lista. En dubbellänkad lista är en länkad datastruktur där varje nod har en pekare till föregående nod och en pekare till nästa nod. Figur 2 visar schematiskt hur denna datastruktur kan se ut.



Figur 2: Dubbellänkad lista innehållandes värdena 2,5,7 och 3

Din klass ska heta `Queue` och ha följande medlemsfunktioner:

void enqueue(double) Stoppar in ett värde (skapar en ny nod) sist i kön

double dequeue() Tar ut och returnerar det första värdet i kön. Tar bort första noden i listan. Om kön är tom ska ett lämpligt undantag kastas.

Speciella medlemsfunktioner Det ska vara möjligt att flytta men inte kopiera en kö. Självklart måste du se till att hantera minnet korrekt så en konstruktor och destruktor krävs.

bool empty() Tar reda på om kön har element eller inte.

Uppgift 5 – Vektor

Inom matematiken är en vektor en samling av numeriska värden i en dimension (till skillnad mot en matris som har flera dimensioner). I denna uppgift ska du skapa en klass som symboliserar en sådan vektor. Klassen `Vektor` ska internt ha en `vector<double>` för att lagra värdena. Skillnaden mot en `std::vector` är att en `Vektor` alltid har ett fast antal värden, dessutom ska du implementera addition samt utskrift av Vektorn. I filen `given_files/test_vektor.cc` finns det några catch-baserade testfall för din slutliga klass. Följande ska stödjas:

Initiering En Vektor initieras antingen med en storlek och ett värde som varje index ska ha. Om inte värdet anges ska detta vara 0.0, storleken är som standard 1 om det inte anges. En storlek mindre än 1 ska inte vara tillåtet. Man ska dessutom kunna initiera en Vektor med en `initializer_list<double>`.

size_t size() Returnerar antalet element i Vektorn.

Indexering `double operator[](size_t idx)` returnerar värdet på index `idx`. Kastar undantag om indexet går utanför Vektorns element.

Utskrift Ska skriva ut elementen på en given utmatningsström. Elementen skrivs ut separerade med komman.

Addition Man ska kunna addera en Vektor, både med en annan Vektor eller med en skalär. Vid addition mellan två Vektorer ska de adderas elementvis och summan (dvs en ny Vektor med samma storlek) returneras. Här krävs att de två Vektorerna har samma storlek annars ska ett undantag kastas. Vid addition med en skalär ska returvärdet vara en ny vektor där varje element är summan av skalären och motsvarande element i ursprungsvektorn. Nedanstående figur försöker visa dessa två typer av additioner.

```
[1, 2, 3] + [4, 5, 6] => [5, 7, 9]

[1, 3, 5] + 5 => [6, 8, 10]
```