Tentamensregler

Hjälpmedel

Följande får tas med på tentan:

- En bok om C++. För boken gäller följande regler:
  - Kommentarer eller noteringar som direkt rör text och exempel på sidan i fråga får finnas i sidmarginalen.
  - Egna sidflikar för att enkelt kunna hitta t.ex. de olika kapitlen är tillåtna.
  - Inga extra ark eller lappar, lösa eller fastsatta, får finnas.
  - Tomma sidor, insidan av pärmarna, försättsblad, etc., får inte innehålla programkod.
- Ett A4-ark med valfritt innehåll på vardera sida (går ej att ersätta med två enkelsidiga ark).
- Penna för att anteckna under tentan. Du kan be tentamensvakt om kladdpapper.

Följande får INTE tas med:

- Elektroniska prylar. Dit hör till exempel miniräknare, mobiltelefon, hörlurar, tangentbord, smartklocka etc.

Utloggning

När du är nöjd med ditt betyg (som står i tentaklienten) kan du avsluta tentan. Stäng alla program och spara alla filer. Sedan loggar du ut som vanligt i menyn. Lämna inte din plats innan vanliga inloggningsskärmen syns.

Frågor om uppgifter

Frågor om tentan i stort eller uppgiftspecifika frågor ska ställas via tenta-klienten. Detta för att vi ska ha en historik av konversationen samt för att vi ska kunna ge samma hjälp till olika studenter.

Systemfrågor

Om du har systemfrågor som t.ex. problem med tentaklienten eller terminalen räcker du upp handen så kommer en assistent och hjälper till.

C++ referens

Det finns tillgång till valda delar av `cppreference.com`. Du måste starta webbläsaren via skrivbordsikonen “Web access” för att komma åt sidan.

Alias för kompilering

Under tentan finns det tre alias att använda sig av för kompilering med c++17:
`w++17` Rekommenderas!
`e++17` Kompilering med alla varningar som fel.
`g++17` Kompilering utan varningar.

Bedömning

Tentamen har två delar. Du måste bli godkänd på del 1 innan inlämning på del 2 bedöms.

Bedömning Del 1

Del 1 består av 4 områden med 1 fråga på varje. Detta är grundfrågor på kursinnehållet du helt enkelt ska kunna. Varje fråga är designad för att kunna besvaras inom 15 minuter även för den som läser och skriver i maklig takt. Områdena behandlar:

1. Absoluta grunder. Exempel: klass, objekt/instans, konstruktor, inkapsling, datamedlem, medlemsfunktion, public, private, ändringsbarhet.
2. Operatorer. Exempel: medlemsoperator, fri operator, C++-standard för operatorer, pre-inkrement, post-inkrement, jämförelseoperatorer, aritmetiska operatorer, tilldelningsoperatorer, inmatningsoperator, utmatningsoperator, (indexoperator, funktionsoperator).
3. Minneshantering. Exempel: statiskt minne, dynamiskt minne, allokering, avallokering, stack-minne, heap-minne, adress, referens, pekare, destruktor, speciella medlemsfunktioner, kopieringskonstruktor, kopieringstilldelning, flyttkonstruktor, flytt-tilldelning, minnesläcka, ägandeskap.
4. Klassrelationer. Exempel: arv, basklass, härledd klass, delegerande konstruktor, polymorfi, virtual, override, pure virtual, virtuell destruktor, abstrakt klass, skyddad medlem, UML, komposition, aggregation, association.

Frågorna på del 1 bedöms med antingen *Godkänt* eller *Ej godkänt*. Vid *Ej godkänt* kan du försöka igen. Du kommer *inte* få några tips om vad som behöver åtgärdas. Uppgiften är så liten att du väntas läsa frågan igen för att uppfylla allt som står enligt god konvention. Om du försöker flera gånger utan konkreta framsteg sätts *Underkänt*. Du är då underkänd på tentamen och kan gå hem så snart tentamensvakten tillåter att du lämnar salen.

Bonus på del 1

Varje laboration i kursen har en extrauppgift. Genom att lösa den tillgodoräknas du motsvarande uppgift i del 1:

Uppgift	Tillgodo från
1	Måste alltid lösas på tentan
2	Tillgodo från Klockslagets extrauppgift
3	Tillgodo från Listans extrauppgift
4	Tillgodo från Pacmans extrauppgift

Bedömning Del 2

Del 2 på tentamen består av två större uppgifter. Här visar du att du kan kombinera flera av kunskaperna från kursen för att lösa ett givet problem. Varje uppgift bedöms med upp till tre poäng:

3p För att en uppgift ska anses godkänd för 3 poäng krävs följande:

- Uppgiften är fullständigt löst enligt givna instruktioner.
- Alla krav uppfylls i generella fallet (körexempel är bara enstaka exempel).
- Koden följer konsekvent och tydlig stil.
- Koden följer god C++-konvention (t.ex. inga kompileringsvarningar).
- Klasser har ett tydligt ansvar och funktioner en väl avgränsad uppgift.
- Klasser är inkapslade med god resurshantering och felhantering.
- Det finns inga övriga brister att tala om.

2p För två poäng är punkterna ett och två för 3 poäng uppfyllda, och bland övriga punkter är det bara *en* som inte uppfylls.

1p Används inte.

0p *Två eller fler* av kraven för 3p är inte uppfyllda.

I del 2 kommer du att få återkoppling på din inlämning som ger dig viss ledning till vad som behöver åtgärdas för att uppnå nästa poängnivå. När du löst problemet kan du skicka in igen. Skickar du in flera gånger utan konkreta framsteg *fryser* vi uppgiften på uppnådd poängnivå. Du kan då inte försöka mer på den uppgiften.

För respektive betyg i kursen krävs lika många poäng:

Betyg	Poäng
3	3
4	4
5	5

Bonus på del 2

Genom visat engagemang i kursen (närvaro på lektioner, ligga i fas, noggrannhet vid kompletteringar) kan du få 1p extra på del 2. Denna bonus förs in under **Uppgift 7** när du uppnått minst 2 poäng.
Löser du en uppgift med 2p eller två uppgifter med 2p är bonusen skillnaden mellan betyg U och 3 eller betyg 4 och 5 respektive.

Del 1: måste lösas före del 2

Uppgift 1

Skapa en klass för att representera ett så kallat “four letter word”. När ett objekt av klassen skapas ska en sträng kunna anges. Givet ett objekt ska man kunna hämta ut strängen ur objektet eller ändra strängens värde. Ett befintligt objekt ska garantera att dess sträng alltid är fyra tecken lång.

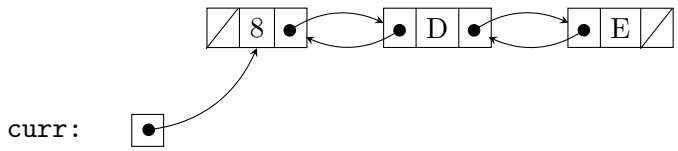
Denna uppgift kräver inga operatorer eller speciella medlemsfunktioner och all kod kan skrivas i filen `uppgift1.cc`.

Uppgift 2

Utöka klassen i uppgift 1 med en operator som förstärker ett “four letter word” genom att upprepa det flera gånger. Resultatet ska vara en sträng där ordet har upprepats det antal gånger som heltalsparametern anger. Din operator ska vara en medlemsoperator.

TIPS: std::string har redan en `operator+=` som eventuellt kan användas.

Uppgift 3



Du har ovanstående struktur för en lista. Skriv kod som tar bort elementet med värde D ur listan. Det du har tillgång till är variabeln `curr`. När du är klar ska `curr` fortfarande peka på elementet med värde 8. Antag noder av typ `struct Node{ Node* prev; int val; Node* next; }`. Koden behöver inte vara ett komplett program som kompilerar.

KRAV: Du får inte byta plats på värden, endast flytta pekare. Samtliga element är dynamiskt allokerade och koden får inte generera minnesläckor.

Uppgift 4

Du har klasserna Sailboat och Tanker som representerar olika typer av båtar.

Segelfartyget har en längd, ett antal master och håller koll på nuvarande vindstyrka. Det finns funktioner för att beräkna räckvidd och lastkapacitet. Vi kan också beräkna en säkerhetsfaktor utifrån båtens storlek.

Tankern har en stor dieseltank, en längd och ett antal vattentäta skott (bulkheads). Vi beräknar säkerhetsfaktorn utifrån storleken (som för andra typer av båtar) och antal vattentäta skott. Vi kan även beräkna en lastkapacitet och en räckvidd.

Lastkapacitet uppskattas på samma sätt för alla båtar medans räckvidd baseras på den specifika båtens egenskaper.

Din uppgift är att implementera en lämplig klassdefinition för basklassen Boat (dvs det som brukar finnas i en headerfil).

Del 2: bedöms först när del 1 är godkänd

Uppgift 5

En polygon består av ett antal sidor som bildar en sammansatt figur. I en konvex polygon är vinkeln mellan alla sidor mindre än 180 grader och en godtycklig linje genom polygonet korsar endast två sidor. Vi kan skapa en konvex polygon med n sidor genom att ange längden av $n-1$ sidor och sedan beräkna den sista sidan. Om sista sidans längd t.ex. sätts lika med den längsta av de övriga går det alltid[1] att rita upp dem som en konvex polygon.

I denna uppgift ska du skapa en klass vid namn `Polygon` som representerar en konvex polygon. Klassen ska lagra en `std::vector` med sidlängder.

Klassen måste uppfylla följande krav:

- Det ska gå att skapa polygon-objekt genom att ange minst två sidor. *Den sista sidan beräknas som max av de övriga sidorna, oavsett hur många sidor som anges. Dvs en polygon kommer då alltid ha minst tre sidor. Observera att den sista sidan aldrig lagras då den måste beräknas på nytt när sidor läggs till.*
- Det ska vara möjliga att lägga till en sida genom att använda medlemsoperatorn += med ett heltal representerande en sidlängd som högerled.
- Det ska vara möjliga att skapa en ny polygon med en extra sida genom att använda additionsoperatorn + utan att ändra vänsterledet. Denna operator ska vara kommutativ (dvs både `p + n` och `n + p` ska fungera). Dessa ska implementeras med hjälp av +=.
- Det ska vara möjligt för användaren att skriva ut polygonen med hjälp av utströmsoperator i enlighet med nedan körexmpel.
- Det ska finnas en funktion vid namn `circumference()` som returnerar polygonens omkrets. Omkretsen beräknas genom att summera polygonets alla sidor (inklusive den beräknade).
- En polygon måste ha minst tre sidor (inklusive den beräknade) och alla sidlängder ska vara större än 0. Om dessa krav ej uppfylls måste konstruktorn kasta ett `std::invalid_argument` undantag.
- Du får **INTE** använda dig av nyckelordet `friend` i denna uppgift.
- Din kod ska ha korrekt filuppdelning `polygon.h,polygon.cc`.

I `given_files/initializer_list.cc` och `given_files/constructor_args.cc` finns det två givna huvudprogram. Du får **INTE** modifiera dessa program. Din lösning måste fungera med minst ett av dem.

Körexempel

```
$ ./a.out
Testing the Polygons.
[3, 4, 4]
[2, 6, 7, 7]
[2, 6, 7, 9, 9]
[2, 6, 7, 1, 7]
Bad polygon: Sides must be > 0.0
Bad polygon: Sides must be > 0.0
Bad polygon: To few sides
```

[1] Författarens förmodan. Lämnas till läsaren att (be)visa.

Del 2: bedöms först när del 1 är godkänd

Uppgift 6

I forskningen som bedrivs inom fälten medicin och biologi är det vanligt att simulera biologiska processer. I denna uppgift ska du implementera en enkel simulering av hur bakterier växer i en koloni baserat på tillgänglig mängd näring (Notera: denna simulering är **inte** baserad på riktig forskning). För att utföra detta behöver du implementera fyra klasser:

- **Bacteria_Colony** är basklassen för en bakteriekoloni. Ett heltal representerar antalet bakterier i kolonin och ett flyttal representerar mängden näring kolonin konsumerar per steg i simuleringen. Klassen har följande funktioner:

  `update()` tar emot ett flyttal som representerar tillgänglig näring. Observera att anroparens argument för detta kommer behöva modifieras av funktionen. Funktionen beräknar hur mycket kolonin växer och hur mycket näring som krävs i ett simuleringssteg enligt de regler som gäller för respektive specialiserad koloni.

  `print()` tar in en utström och skriver till den information om kolonins storlek m.m. enligt det som gäller för respektive koloni. Se körexempel för respektive koloni.

- **Ecoli** representerar en koloni av Escherichia coli (E. coli) bakterier. Klassen ärver från `Bacteria_Colony` och har två datamedlemmar av flyttalstyp vid namn `growthFactor=2.0` och `growthRate=1.5`.

  `update()` kontrollerar om det finns tillräckligt med näring och konsumerer i så fall `delta = number of bacteria * consumption rate`. När kolonin har konsumerat näring multipliceras antalet bakterier med `growthFactor`. `growthFactor` multipliceras sedan med `growthRate`. Om det inte fanns tillräckligt med näring sker ingenting.

- **Saureus** representerar en koloni av Staphylococcus aureus (S. aureus) bakterier. Klassen ärver från `Bacteria_Colony`.

  `update()` kontrollerar om det finns tillräckligt med näring och konsumerar i så fall `delta` (se Ecoli ovan) näring samt multiplicerar antalet bakterier med `3`.

- **Petri_Dish** innehåller en `std::vector` av bakteriekolonier och en datamedlem av typen `double` som representerar den totala mängden tillgänglig näring i petriskålen. Klassen har två medlemsfunktioner:

  `simulate()` kör varje kolonis `update()` funktion en gång med näringen som argument.

  `print()` kör varje kolonis `print()` funktion en gång och skriver till slut ut den totala mängden näring som finns kvar. Se körexmpel.

Det finns ett program i filen `given_files/bacteria.cc` att bygga ut.

Körexempel

```
$ ./a.out
E-coli count: 1 with growth factor: 2
S-aureus count: 1
Nutrients: 600
Simulating 5 time steps...
E-coli count: 1842 with growth factor: 15.1875
S-aureus count: 243
Nutrients: 140
```