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 tentaklienten. 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

På tentamen har du tillgång till referensdelarna av `cppreference.com`. Du måste starta webbläsaren via skrivbordsikonen “Web access” för att komma åt sidan.

Terminalkommandon

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

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 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.
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.

Frågorna på del 1 bedöms med antingen *Godkänt* eller *Ej godkänt*. Vid *Ej godkänt* kan du försöka igen. Du kommer *endast* att få tips om vilken text från uppgiften som inte är ordentligt uppfylld. 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

Denna bonus är giltig 1 år från det du påbörjade kursen. Varje laboration i kursen har en extrauppgift. Genom att lösa den tillgodoräknas du motsvarande uppgift i del 1:

Uppgift	
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:

Poäng	Kriterie
3p	Samtliga av följande punkter är uppfyllda: • Klasser har ett tydligt ansvar och funktioner en väl avgränsad uppgift. • Klasser är inkapslade med god resurshantering och felhantering. • Efterfrågade språkkonstruktioner används. • Efterfrågad funktionalitet är huvudsakligen implementerad. **Observera att kod som inte kompilerar eller inte fungerar fullt ut fortfarande kan ge poäng om du undviker övriga avdrag.**
-1p	Stora brister. Max 2p avdrag. Avdrag med 1p för **varje** av följande fel: • Koden kompilerar inte. • Givna instruktioner är inte fullständigt uppfyllda. • Koden har brister som kan leda till felaktig beteende. Varje brist räknas. Se exempelsidor nedan.
-1p	Mindre brister. Max 1p avdrag. Avdrag med 1p om minst **två** av följande brister förekommer: • Koden följer inte konsekvent och tydlig stil. • Koden bryter mot god C++-konvention. Varje felkategori räknas. Se exempelsidor nedan. • Koden har brister som inte ger felaktigt beteende men som hämmar vidareutveckling eller underhåll. Varje brist räknas. Se exempelsidor nedan.

I del 2 kommer du att få återkoppling på din inlämning som ger dig ledning till vad som behöver åtgärdas (men inte hur) för att undvika avdrag. 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

Denna bonus är giltig 1 år från det du påbörjade kursen. Genom visat engagemang i kursen 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 är bonusen skillnaden mellan betyg U och 3.
Löser du två uppgifter med 2p på varje är bonusen skillnaden mellan betyg 4 och 5.

1 Exempel på brister med -1p per brist (max -2p)

Notera att listan inte är uttömmande. Detta är de vanligaste.

1.1 Brister som kan leda till felaktigt beteende

saknad generalisering
: Leder till saknad funktionalitet. Antag uppgiften att läsa in och summera tal. Fel: lösningen klarar 10 inmatade tal. Rätt: lösningen klarar N inmatade tal för valfritt N.

fullständig uppräkning
: Leder ofta till saknad generalisering (ovan) och alltid till hämmad vidareutveckling. Fel: Ett unikt kodstycke för varje upptänkligt data att behandla. Rätt: En loop som itererar all data som ska behandlas.

minnesläcka
: Kan leda till krasch eller behov av omstart. Fel: Någon allokering avallokeras inte. Rätt: Varje allokering avallokeras en gång.

slicing
: Leder till saknad funktionalitet. Fel: Härledd klass trunkeras till basklass. Rätt: Härledd klass bibehålls.

off by one
: Leder till fel. Fel: Loop kör ett varv för lite eller för mycket. Indexering just utanför vektor eller array.

ogiltig avreferering
: Leder till krasch. Fel: användning av ogiltig referens, index out of bound, avreferering av ogiltig pekare

undefined behaviour
: Användning av konstruktion där förväntat utfall inte är specificerat i språket.

användning före initiering
: Fel: En variabel eller medlem används innan den fått ett initialvärde.

2 Exempel på brister med -1p per två brister (max -1p)

Notera att listan inte är uttömmande. Detta är de vanligaste.

2.1 Brott mot god konvention

fel parametermode
Det är inte tydligt om en parameter är avsedd för indata eller utdata, eller så leder valt mode till onödig ineffektivitet.

nyckelord används ej
Språket tillhandahåller nyckelord som ger kompilatorn möjlighet kontrollera programmerarens intentioner, men dessa nyckelord används inte där så är möjligt.

operatorkonvention följs inte
En operator implementeras med ett beteende som normalt inte förväntas av operatorn, eller följer inte förväntat mode på parametrar och returvärde.

kompileringsvarning
Kompilatorn ger varning vid kompilering med rekommenderade varningar på-slagna (`w++17`). Leder lätt till att nya viktiga varningar går obemärkt förbi. Varningen i sig indikerar ofta förekomst av någon annan brist som kan leda till felaktigt beteende.

kod i c-stil
Kod i c-stil har brister i felhantering.

saknad headerguard
Ger svårighet att återanvända program-modulen.

using namespece i h-fil
Leder till att programmeraren som inkluderar oväntat använder funktioner från std, som ibland är annan än avsedd funktion.

2.2 Brister som hämmar vidareutveckling och underhåll

datamedlemsinitieringslista saknas
Leder till extra arbete att införa referens- och const-medlemmar, kan leda till användning före initiering.

koduppprepning
Leder till onödig kod att underhålla och kan leda till fel när en upprepning glöms bort medan övriga uppdateras.

bristande felhantering
Leder till svårigheter att felsöka. Efterstäva ett unikt felmeddelande för varje felorsak och felrapportering inte kräver explicita kontroller i anropande kod för att upptäckas.

otillräcklig funktionsuppdelning
Leder till svårighet att återanvända funktioner.

bristande ansvarsfördelning
Kan t.ex. leda till koduppprepning, otillräcklig funktionsuppdelning, eller svårighet veta vem som har ansvar för avallokering med minnesläcka som följd.

Del 1: måste lösas före del 2

Uppgift 1

Skapa en klass som representerar ett kylskål. Ett kylskåp har en kapacitet som är ett flyttal. Den har också ett flyttal för att visa hur fyllt kylskåpet är. När man skapar ett kylskåp ska man kunna ange kapaciteten och kylskåpet ska vara tomt. Det ska finnas en funktion som förbrukar en angiven mängd plats i kylskåpet. Den har ett flyttal som parameter. Din klass ska garantera att kylskåpets kapacitet aldrig är negativ samt att kylskåpets innehåll inte överskrider kapaciteten.

TIPS: Förväntad lösning är ca 25 rader kod.

Uppgift 2

Implementera en operator som jämför en sträng med ett heltal. Operatorn ske ge ut `true` om strängen har färre tecken än heltalet har siffror, annars `false`.

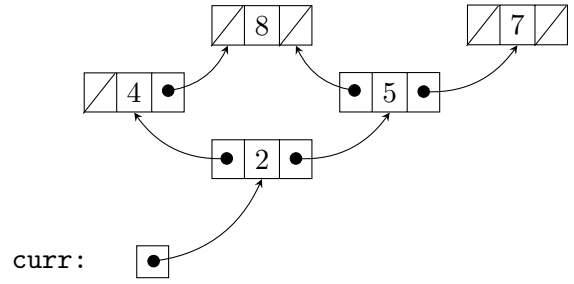
KRAV: Din operator ska implementeras som en fri funktion.

TIPS: Förväntad lösning är ca 4 rader kod.

TIPS: Konvertera heltalet till en sträng och räkna tecken.

Uppgift 3

Skriv kod som skapar den länkade strukturen nedan.



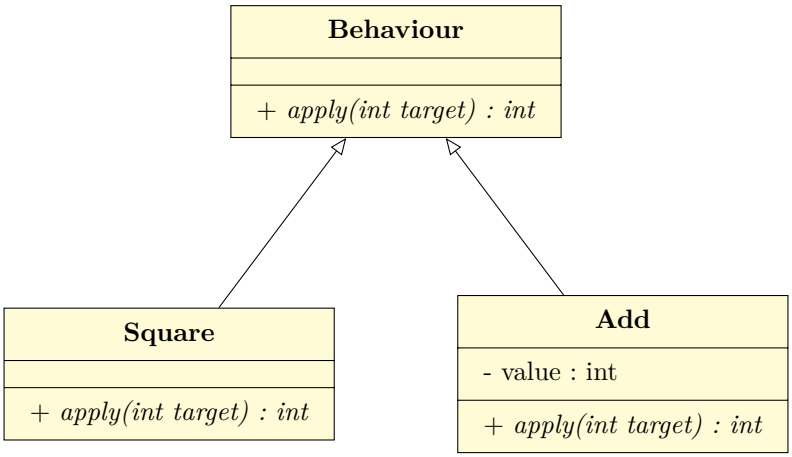
KRAV:

- Du måste använda den givna `Node` från `uppg3.cc`.
- Samtliga element ska vara dynamiskt allokerade. Du behöver inte avallokera elementen.
- Du får inte deklarera några fler variabler än de som redan är deklarerade i `main()`.

TIPS: Förväntad lösning är 2 - 7 rader kod.

Uppgift 4

Implementera följande klasshierarki:



- `Behaviour::apply(int target)` har inget definierat beteende.
- `Square::apply(int target)` returnerar `target * target`
- `Add::apply(int target)` returnerar summan av `target` och datamedlemmen `value`

KRAV:

- Alla dina klasser ska vara deklarerade och definierade i *en* källkodsfil.
- Den givna filen `uppg4.cc` har ett main-program som ska fungera oförändrat.

Funktionella krav: Körning av programmet ska ge följande utdata i terminalen:

```
100
```

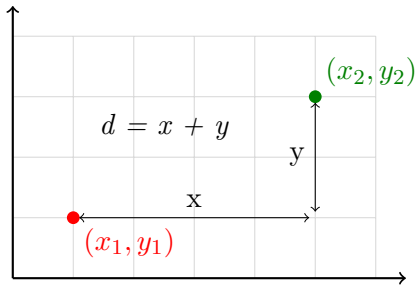
TIPS: Förväntad lösning är ca 34 rader kod.

Del 2 - Uppgift 5

Ett lagerhus förvarar varor som ska till ett antal butiker. En lastbil utgår från lagerhuset och kör ut till alla butiker. För att hantera butiksdestinationerna och hitta den optimala vägen behöver lastbilen en klass som håller koll på var lagerhuset är och vart alla destinationer är.

Din uppgift är att implementera två datatyper enligt instruktionerna nedan.

Destination är ett aggregat (`struct`) som sparar ett namn på en plats samt två heltal x och y för att representerar platsens koordinater. Utöver det ska du implementera en fri funktion för att beräkna avståndet mellan två destinationer. Lastbilen kan inte köra fågelvägen (då skulle den krascha in i hus) så avståndet beräknas som "manhattanavståndet". D.v.s. att vi summerar distansen i x-led och distansen i y-led. Observera att distansen alltid ska vara positiv.



Route_Planner är en klass som ger en lastbilschafför information om de destinationer hen ska besöka. Klassen behöver lagra

- en lista med destinationer. Listan representerar de butiker som ska besökas.
- en destination som representerar lagerhuset.

Med ett objekt av klassen ska man kunna

- beräkna den totala distansen en lastbil skulle behöva köra om den åkte fram och tillbaka från lagerhuset till varje destionation i listan.
- lägga till en till destination i listan genom att anropa `operator+=`. Om den ny totala distansen enligt ovan skulle bli större än 100 ska destinationen inte läggas till i listan. Istället ska ett undantag kastas enligt körexemplet.
- plocka ut den destination i listan som är närmast en given startdestination.
- skriva ut alla destinationer i listan på given utström genom `operator<<`.

Körexempel

```
Med 'ica' blir den totala distansen är större än vad en lastbil klarar av!
=== Alla butiker ===
# Total distans: 94
willys: [10, 10]
coop: [3, -1]
hemköp: [-8, 4]
lidl: [-4, -5]

======
Närmaste butik är: lidl
```

Del 2 - Uppgift 6

Denna uppgift liverättas ej. Kontrollera noga att den fungerar och är välskriven innan du skickar in. Du kan fortfarande skicka in så många gånger du vill. Det senast inskicket bedöms.

I ett nytt äventyrsspel med 2D grafik ska en spelare kunna röra sig runt och interagera med objekt i spelet. I spelet ska det finnas minst två typer av spelobjekt som påverkar spelaren. Spelaren representeras av `struct Player` som finns given i filen `uppg6.cc`. Där finns även ett huvudprogram för att testa de skapade spelobjekten. Varken spelaren eller huvudprogrammet får modifieras. Alla positioner i spelet är heltal.

Utöver basklassen som representerar spelobjekt (`GameObject`), som du designar själv, ska du implementera två klasser:

Portal: En portal har en position på spelplanen och äger (ansvarar för) ett annat spelobjekt som fungerar som portalens destination. På ett portalobjekt ska det gå att anropa tre medlemsfunktioner:

- `void move_player_here(Player& p)`: Sätter spelarens position till portalens position.
- `bool collides(Player const& p) const`: Returnerar sant om spelarens position är samma som portalens position.
- `void affect(Player& p) const`: Sätter spelarens position till destinationens position ***och*** låter sedan destinationen utföra sin påverkan på spelaren.
- Kopieringskonstruktor och kopieringstilldelsningsoperator tas explicit bort (eller implementeras korrekt) eftersom denna klass ansvarar för ett annat objekt. Borttagning av en medlemsfunktion görs genom att deklarera funktionen och avsluta med `= delete`.

Bouncer: En studsare har en position på spelplanen och håller reda på sin studsförmåga (ett heltal). På ett studsobjekt ska det gå att anropa tre medlemsfunktioner:

- `void move_player_here(Player& p)`: Sätter spelarens position till studsarens position.
- `bool collides(Player const& p) const`: Returnerar sant om spelarens position är samma som studsarens position.
- `void affect(Player& p) const`: Multiplicerar spelarens hastighet med studsförmågan f och subraherar produkten från spelarens position. Matematiskt skulle det se ut så här:
  $p_x = p_x - v_x * f$
  $p_y = p_y - v_y * f$

Icke-funktionella krav:

- Det ska inte finnas koduppprepning. Gör lämplig basklass `GameObject`.
- Det givna testprogrammet ska fungera oförändrat. Gör lämpliga konstruktorer utifrån hur de anropas i testprogrammet.
- Det ska inte förekomma minnesläckor så som testprogrammet är skrivet. Gör lämplig(a) destruktor(er).

Funktionella krav: Korrekt körning av programmet ska endast ge 7 utskrifter av “ok” i terminalen.