# 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

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, slicing.

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 | 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:

| Poäng | Kriterie |
|---|---|
| 3p | Samtliga av följande punkter är uppfyllda:<br>• Klasser har ett tydligt ansvar och funktioner en väl avgränsad uppgift.<br>• Klasser är inkapslade med god resurshantering och felhantering.<br>• Efterfrågade språkkonstruktioner används.<br>• Efterfrågad funktionalitet är huvudsakligen implementerad.<br>**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:<br>• Koden kompilerar inte.<br>• Givna instruktioner är inte fullständigt uppfyllda.<br>• Koden har brister som kan leda till felaktig beteende.<br>Varje brist räknas. Se exempelsidor nedan. |
| -1p | Mindre brister. Max 1p avdrag. Avdrag med 1p om minst **två** av följande brister förekommer:<br>• Koden följer inte konsekvent och tydlig stil.<br>• Koden bryter mot god C++-konvention.<br>Varje felkategori räknas. Se exempelsidor nedan.<br>• Koden har brister som inte ger felaktigt beteende men som hämmar vidareutveckling eller underhåll.<br>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 (närvaro på lektioner, ligga i fas, noggrannhet vid komplettering) 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 för att representera en rektangel. När ett objekt av klassen skapas ska två flyttal kunna anges. De representerar bredden och höjden på rektangeln. Det ska finnas funktionalitet i klassen för att rotera rektangeln 90 grader. Ett befintligt objekt ska garantera att den längsta sidan inte är mer än två gånger den kortare sidan.

*TIPS:* Lösningsförslaget är cirka 30 rader kod. Denna uppgift kräver inga operatorer, speciella medlemsfunktioner eller tester. All kod skrivs i `uppg1.cc`.

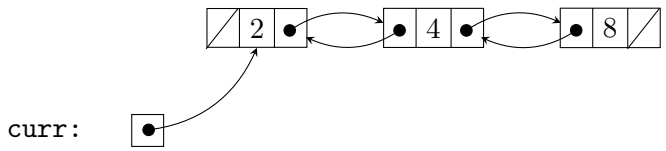
## Uppgift 2

Kopiera din `uppg1.cc` till `uppg2.cc` och utöka klassen i uppgift 1 med en operator som jämför två rektanglar. En rektangel är mindre än en annan om dess area är mindre.

*KRAV:* Din operator ska implementeras som en medlem.

*TIPS:* Lösningsförslaget lägger till cirka 6 rader till uppgift 1.

## Uppgift 3




Du har ovanstående struktur för en lista. Skriv kod i `uppg3.cc` som sätter in ett nytt värde 6 mellan de existerande elementen med värde 4 och 8. Det du har tillgång till är variabeln `curr`. Antag noder av typ `struct Node{ Node* prev; int val; Node* next; }`.

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

*TIPS:* Förväntad lösning är cirka 3 rader kod. Komplett kompilerande program krävs ej.

## Uppgift 4

Du har klasserna `Enemy`, `Player` och `Tree` för att representera objekt i ett spel.

Klassen `Enemy` har en position, hastighet och kan skjuta projektiler mot andra objekt. Fiender flyger runt och försöker undvika att bli träffad av spelaren. Om fienden träffas tre gånger försvinner den. `Player` har ett antal liv, en position och en hastighet. Spelaren flyttas med hjälp av piltangenterna. Klassen `Tree` ser ut att stå stilla på en position, men rör sig i själva verket mycket mycket långsamt. Träd försvinner när de blivit träffade fem gånger.

Alla klasser har en funktion för att kontrollera om de kolliderar med ett annat objekt.

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

# Del 2: bedöms först när del 1 är godkänd

## Uppgift 5

Nystartade mäklarfirman ”3House”har beställt en app för att snabbt kunna sätta ett lägsta grundpris på sina objekt. Appen ska byggas objektorienterat och centreras kring klassen `House`. Ett hus har ett antal rum (egen klass), avstånd till närmaste stadskärna (km), och om det har källare eller inte. När ett hus skapas ska avståndet krävas, men huset är utan källare om inte annat anges. Givet ett klassobjekt av typen hus ska det gå att lägga till rum och beräkna husets värdering. Rum ska gå att lägga till både med operatorn `+=` och medlemsfunktionen `add_room`.

Ett rum beskrivs av ett namn (t.ex. ”kök”) och rummets yta (kvm). När rum skapas ska det även gå att ange om rummet är i behov att renoveras eller ej, men om inget anges antas att rummet är i modernt och gott skick. För att räknas som ett rum måste ytan vara minst 7 kvm. Ytterligare kriterier[1] ett rum måste uppfylla ignoreras i denna första version av appen.

Värdet på ett hus beräknas enligt en helt *ny revolutionerande formula*[2] som *areavalue * basementfactor * distancefactor - renovationcost*. *Areavalue* är total yta multiplicerat med 30000 kr/kvm. *Basementfactor* är 0.75 om huset har källare, annars 1.0. *Distancefactor* är $4/(citydistance+3)+0.6$. *Renovationcost* är 10000 kr/kvm multiplicerat med total yta som är i behov att renoveras.

Mäklarfirman ser framför sig att de kommer vilja utöka både rum och hus med många nya attribut i framtiden. Det är därför av största vikt att framtida utökning av respektive klass kan göras med minimal påverkan på kod som i det läget redan använder klasserna. Din uppgift är att skapa klasserna `Room` och `House`.

*KRAV:* Korrekt utförd filuppdelning krävs för `House`. `Room` kan ligga tillsammans med `House`.

*TIPS:* I filen `uppg5.cc` finns huvudprogram att utgå från. Notera att de givna körexemplen inte är kompletta och testar varken all funktionalitet eller alla datamängder.

## Körexempel

```
 0.0 km => 5130000.0
 1.5 km => 3863333.3
 3.8 km => 3018888.9
 7.1 km => 2455925.9
12.2 km => 2080617.3
19.8 km => 1830411.5
31.2 km => 1663607.7
48.3 km => 1552405.1
73.9 km => 1478270.1
```

[1] Fastighetsmäklarinspektionens definition av ett rum: Med ett rum avses ett utrymme som har en golvyta på minst sju kvadratmeter och som har direkt dagsljus. Som rum räknas inte kök, kokvrå eller hygienutrymme. Som rum räknas inte heller matrum i anslutning till kokvrå om matrummet tillsammans med kokvrån anges som kök. Utrymmen som är mindre än sju kvadratmeter redovisas inte som rum.

[2] Det är ofta viktigt att tolka “ny revolutionerande formula” i sin rätta betydelse, dvs som rent hittepå utan vetenskapligt belägg.

# Del 2: bedöms först när del 1 är godkänd

## Uppgift 6

Din uppgift är att skapa klasser för ett ritprogram. Programmet låter användaren bygga upp figurer genom att kombinera grundläggande former. Först ut är att skapa klasser för att representera rektanglar och cirklar. En rektangel vet om sitt övre vänstra hörns position (x,y) samt bredd, höjd och utritningssymbol. En cirkel har en centerposition (x,y) samt en radie och utritningssymbol. Respektive form kan tillfrågas om den täcker en viss koordinat (x,y) samt vilken utritningssymbol den har.

En cirkel täcker positionen (px,py) om avståndet från dess centrum (cx,cy) till positionen är mindre än dess radie. Avståndet beräknas som $\sqrt{(px-cx)^2+(py-cy)^2}$. Det är okej att jämföra om avståndet i kvadrat är mindre än radien i kvadrat för att slippa använda kvadratroten.

En rektangel täcker en position (px,py) om $leftside <= px < rightside$ och $topside <= py < bottomside$ (y växer nedåt, $rightside = leftside + width$, $bottomside = topside + height$).

*TIPS:* I filen `uppg6.cc` finns huvudprogram att utgå från. Notera att den givna koden kan behöva justeras för att erhålla dynamisk bindning.

## Körexempel (något nedkortat av utrymmesskäl)

```
          ;;;;;;;;;
         ;;;;;;;;;;;
         ;;;;;;;;;;;
         ;;;;;;;;;;;
          ;;;;;;;;;

                  ###############
      %%%%%       ###############
      %%%%%       ###############
      %%%%%         ###########
                    ###########
      ####          ###########
      ####          ###########
      ####          ###########
  ################################
  ################################
  ################################
     xxxxx     xxxxx     xxxxx
   xxxxxxxxx xxxxxxxxx xxxxxxxxx
   xxxxxxxxx xxxxxxxxx xxxxxxxxx
   xxxxxxxxx xxxxxxxxx xxxxxxxxx
     xxxxx     xxxxx     xxxxx
```