Pedagogisk filosofi – läs mig först!

Sedan flera år tillbaka har varje kurs publika och fastslagna kursmål som anger vad som ingår på kursen. Dessa är menade som ett stöd till studenterna, men i min (Tobias) mening är det få eller inga studenter som aktivt drar nytta av kursmålen i sina studier, t.ex. som ledning inför vilka avsnitt i kurslitteraturen som endast är kursiva, etc. Till yttermera visso: för studenter som faktiskt läser kursmålen blir kopplingarna mellan dem och momenten på kursen svaga eftersom målen ofta (av goda själ) är vidlyftigt specificerade, eller för att det ligger i sakens natur att man ofta inte helt kan förstå kursmålen innan man faktiskt har klarat av kursen.

Jag stöter ofta på studenter (på många olika kurser och utbildningar) som lämnar in samtliga uppgifter och blir godkända, men som har svårt att förstå ”vad de kan”. Den som examinerar lovar vackert att ”om du klarar av samtliga moment på kursen så uppfyller du dessa mål”, men som student är det ofta svårt att förstå varför och ibland även att vara övertygad om att det faktiskt stämmer.

På denna kurs försöker vi undvika detta genom att expandera kursmålen och därmed göra det mer explicit vad man som student måste *förstå/behärska* för att bli godkänd och vad ökad *kvalitet* i praktiken innebär för att öppna dörrarna till högre betyg.

Mina mål är bland annat:

- Att göra det lättare för studenten att se inte bara vilka moment som är avklarade, utan också vilka konkreta kursmål som avklarats.
- Att lägga över mer ansvar på studenten för sin egen utbildning och samtidigt ge större frihetsgrader i hur man tar till sig ett ämne och i vilken ordning.
- Att tydliggöra för studenten vad hon faktiskt kan och därigenom bygga både självinsikt och självförtroende.
- Att frångå modellen där man i praktiken skickar meddelandet ”vad bra att du kan implementera ett program som gör X” till en modell där meddelandet är ”vad bra att du förstår länkade strukturer”.
- Att flytta kursen närmare ”problembaserat lärande” som jag upplever bygger starkare och intelligentare studenter (än traditionell undervisning) genom att kräva mer sökande och värderade av information.
- Att göra så stor del av examinationen i dialogform så att återkopplingen uppstår naturligt (nästan ingen hämtar ut sin tenta och läser lärarens kommentarer i de flesta kurser).
- Att undvika artificiella examinationsformer (t.ex. salstenta med programmering på papper utan de vanliga hjälpmedlen) i möjligaste mål, och på samma sätt examinationsformer som beror på dagsformen.
- Att skapa anledning till interaktion med de många och kunniga assistenterna utöver ”rättning” eller hjälp med felsökning.
- Att kräva en stor arbetsinsats av dig som student, på ett roligt sätt som också är lärorikt och utvecklande. Man kan nämligen bara lära sig programmering genom praktisk tillämpning – genom att skriva egen kod och läsa andras kod. Dessutom hoppas jag att kursen skall vara utvecklande på fler plan än programmering och systemutveckling.

Min förhoppning är att årets kurs skall vara betydligt bättre än tidigare års, och att det system som vi nu prövar ut (2013) skall fungera i praktiken utan allt för stora barnsjukdomar. Som vanligt behöver vi *din* återkoppling för att förstå hur det fungerar och om några ändringar behöver göras.

Den nya kursen har utvecklats tillsammans med flera av de seniora assistenter som arbetar på kursen sedan många år tillbaka, speciellt Elias Castegren och Tobias Skoglund. Jag har haft stor hjälp också från av nyare assistenter och tidigare års studenter, och det är bland annat tack vare deras entusiasm som jag vågar göra en så omfattande förändring!

Beskrivning av kursmål

Detta dokument beskriver de expanderade kursmålen på kursen imperativ- och objektorienteradprogrammeringsmetodik, IOOPM.

För varje mål anges vilken nivå målet ligger på och hur det kan redovisas. Totalt finns 43 mål på nivå 3, 22 mål på nivå 4 och 11 mål på nivå 5. Du förväntas bocka av mål löpande under kursens gång (se instruktioner på kursens webbsida). För att få betyget 3 på kursen måste samtliga mål på nivå 3 vara uppfyllda, för betygen 4 måste samtliga mål på nivå 3 och 4 vara uppfyllda och för betyget 5 måste samtliga mål i detta häfte vara uppfyllda. Observera att det inte finns en linjär relation mellan antalet mål och tidsåtgång – och många mål kan bockas av samtidigt och inom ramarna för samma uppgift.

Kod	Beskrivning	# 3	# 4	# 5
L	Redovisas i samband med labbtillfälle	32	18	5
G	Redovisas i samband med gruppmöte	12	17	10
T	Redovisas på frivillig tenta (se även essäuppgiften)	1	2	5
I	Intygas	1		
R	Redovisas genom en rapport	5		
W	Skickas in via epost		1	

Målen är uppdelade i tre kategorier med följande fördelning med avseende på nivåerna på de olika målen:

Typ av mål	# 3	# 4	# 5
Kunskapsmål	25	18	9
Verktygsmål	9	2	1
Strukturmål	9	2	1
$\sum = 76$	43	22	11

Metainstruktioner

I resten av detta dokument beskrivs samtliga mål. I vissa beskrivningar ges exempel för ökad tydlighet, men det är alltså inte meningen att det är dessa exempel som skall implementeras! Skilj mellan uppgifter och mål: uppgifterna (*inte* i detta dokument) ger specifikationen för program etc. som skall implementeras; målen (i detta dokument) skall redovisas inom ramarna för dessa uppgifter och en av de viktiga insikterna som du måste nå för att klara av denna kurs är vilka mål som är lämpliga att redovisa i vilken uppgift, och hur många mål faktiskt hänger samman eller till och med blir lättare om man löser dem tillsammans. Därigenom står det också klart att högre betyg inte handlar om kvantitet, utan om syntes och djupare nivå av insikt, alltså kvalitet. Utan dessa blir det nämligen omöjligt att ens *hinna* redovisa de totalt 76 målen för betyget 5.

Du har god hjälp av assistenterna här, särskilt under fredagsmötena, och naturligtvis också av kursledningen. Om du fastnar skall du alltid be om hjälp!

Notera att du förväntas att *själv söka efter information*, både i de länkar och bokreferenser som finns i kursmaterialet, men också fritt med hjälp av t.ex. Google och Wikipedia. Föreläsningarnas syfte är att måla en övergripande bild och introducera viktiga koncept och ibland även göra praktiska övningar. Screencasts används för mer detaljerade utvikningar om mindre ämnen. Förslag på kursböcker finns på kursens webbsida, men det finns ingen specifik bok som vi följer under kursens gång.

Mål

Först följer en förteckning över alla mål. Efter förteckningen kommer en genomgång av målen igen med en utvikning för varje mål och information om hur målen kan redovisas.

Abstraktion	Nivå
Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar	3

Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt	3
Demonstrera förståelse för designprincipen informationsgömning i ett C-program med hjälp av `.c` och `.h-filer`	4

Arv	Nivå
Använda arv, metodspecialisering och superanrop i ett program som drar nytta av subtypspolymorfism	3
Använda både överlagrade och specialiserade konstruktorer på lämpligt sätt i programmering	3
Förklara hur arvsbegreppet har använts i ett program för att separera genomskärande åtaganden	4
Förklara hur man kan undvika s.k. ”bräckliga basklasser” vid arv	5

Design	Nivå
Förklara hur designprinciperna modularisering och ”divide and conquer” har använts i ett eller olika program	3
Förklara begreppet designmönster samt visa hur minst ett designmönster har använts i ett icke-trivialt program	3
Demonstrera grundläggande förståelse för objektorienterad analys och design genom att redogöra för ett icke-trivialt programs design och analysen som ligger till grund för designen	4
Visa hur man kan separera gränssnitt från implementation med hjälp av Java-interfaces	4

Dokumentation	Nivå
Dokumentera icke-triviala modulers gränssnitt så att någon utomstående kan programmera mot dem	3

Genericitet	Nivå
Använda void-pekare i C-program för att uppnå genericitet på ett relevant sätt, t.ex. en datasamling som kan lagra godtyckligt data	3
Använda parametrisk polymorfism för ökad ”statisk typsäkerhet” vid interaktion med Javas standardbibliotek	3
Designa med parametrisk polymorfism för att göra relevanta delar av ett program mer generellt	4

Imperativ programmering	Nivå
Översätta mellan rekursiva och iterativa lösningar av samma problem samt diskutera för- och nackdelar med sido-effekter	3
Ta fram loopinvarianter för loopar och resonera om deras korrekthet	4
Förklara skillnaden mellan deklarativ och imperativ; diskutera deras respektive för- och nackdelar	5

Inkapsling	Nivå
Använda kopiering eller liknande för att undvika representationsläckage i ett C- eller Javaprogram	3
Använda åtkomstmodifierare för att styra åtkomst	3
Använda nästlade och inre klasser för att kapsla in privata beståndsdelar i ett sammansatt objekt	4
Resonera kring hur inkapsling både kan försvåra och underlätta testning (ta gärna hjälp av kodexempel)	5

Klasser och objekt	Nivå
Skriva minst två jämförelsemetoder för minst två sammansatta objekt	3
Redovisa förståelse för skillnaden mellan värdesemantik och referenssemantik med hjälp av ett kodexempel	3

Beskriva en klass klassinvarianter i kommentarer, samt resonera om hur inkapslingen slår vakt om invarianterna	4
Redovisa förståelse för abstrakta klasser och metoder, samt deras relation till Java-interface	5

Metodik	Nivå
Förklara innebörden av, och tillämpa på ett konsekvent sätt, defensiv programmering i ett program	3
Fånga på lämpliga platser och hantera på ett lämpligt sätt kontrollerade och okontrollerade (relevanta) undantag i ett program	3
Utöver föregående mål, demonstrera fördjupad förståelse för undantagshantering i Java genom att kasta existerande och egendefinierade kontrollerade och okontrollerade undantag i ett program	4

Minneshantering	Nivå
Demonstrera förståelse för skillnaden mellan allokering på stacken och allokering på heapen med hjälp av ett C-program	3
Demonstrera förståelse för minneshantering i C genom att skriva ett program med dynamiska strukturer som är fritt från minnesläckage och argumentera för varför det är så (och verifiera med valgrind)	4
Förklara skillnaden mellan C:s manuella minneshantering och hur Java hanterar minne och beskriv för ett lämpligt Java-program när minne allokeras och frigörs	4
Jämför två metoder för automatisk skräpsamling	5

Modularisering	Nivå
Specificera tydliga gränssnitt mellan moduler i ett program som bör brytas ned i flera moduler (och implementera)	3
Givet ett icke-trivialt program uppdelat i moduler som i ovanstående mål, resonera kring begreppen coupling och cohesion för några av modulerna	4
Utveckla resonemanget i (4) med en diskussion om separation of concerns och hur detta har uppnåtts (alt. kan/inte kan) i programmet i fråga	5

Parallelprogrammering	Nivå
Använda Fork/join i java.util.concurrent för att parallellisera relevanta delar av ett tidigare program	3
Utöver (3), mät körtiden före och efter och förklara mätresultaten	4
Utöver (4), justera eventuella parametrar (t.ex. sequential cutoff, antalet trådar) för att se hur prestanda påverkas, mät och förklara igen	5

Pekare	Nivå
Översätta mellan C:s array-notation och pekararitmetik	3
Använda pekare för att skapa länkade strukturer och använda pekare till stackvariabler för värdeöverföring mellan funktionsanrop	3
Använda pekare till pekare (dubbelpekare) på ett lämpligt sätt i ett program	4

Pragmatics	Nivå
Demonstrera på lämpligt sätt förståelse för termerna kompilering, länkning, interpretering och JIT-kompilering och hur dessa används i C och Java	3
I samband med ovanstående, förklara statisk och dynamisk bindning och exemplifiera med programkod	4

Profilering och optimering	Nivå

Använda profileringsverktyg för att visa var ett redan skrivet program tillbringar mest tid.	3
Med ledning resultatet från föregående mål, optimera programmet på ett förtjänstfullt sätt.	4
Samma som mål (3) och (4) men för minnesanvändning	5

Refactoring	Nivå
Tillämpa enkla refactoringmönster på ett program från en tidigare sprint i ett verktyg med särskilt stöd för refactoring	3
Utöver föregående, använd mer komplicerade mönster och diskutera hur programmet förbättras	4

Testning	Nivå
Ta fram lämpliga enhetstester för ett icke-trivialt program, motivera dem, och förklara vad testerna visar	3
Ta fram en uppsättning regressionstester för en del av ett program, motivera dem, och förklara vad testerna visar	4
Tillämpa automatiserad kontinuerlig integrationstestning, motivera testerna och förklara vad de visar	5

Verktyg/Debuggning	Nivå
Använda GDB för debuggning	3
Jämföra GDB och Netbeans/Eclipse	4

Verktyg/Dokumentation	Nivå
Använda doxygen eller JavaDoc för att extrahera kommentarer från kod till lämplig API-dokumentation	3
Använda LaTeX	3

Verktyg/Emacs	Nivå
Använda Emacs för editering	3
Utöver (3), dra nytta av Emacs inbyggda stöd för programmering	4
Utöver (4), dra nytta av Emacs möjligheter till utökning för programmering	5

Verktyg/Kompilering	Nivå
Använda Make	3
Använda Lint	3
Använda valgrind	3

Verktyg/Terminalen	Nivå
Behärska grundläggande terminalkommandon	3

Verktyg/Versionshantering	Nivå
Behärska grundläggande versionshantering	3

Strukturmål/Misc	Nivå
Implementera minst tre uppgifter komplett och i två olika språk	3
Arbeta i minst fyra olika par under projektet (Intygas)	3
Redovisa ett mål ur G eller T-kategorin som en essä	4
Hålla minst en välstrukturerad presentation vid gruppmöten (Bockas av vid tillfället)	3
Hålla en välstrukturerad verktygspresentation vid gruppmöte (Bockas av vid tillfället)	4

Visa förmåga att förmedla sin kunskap genom att handleda ett pass	5

Strukturmål/Projekt	Nivå
Använda parprogrammering	3
Tillämpa test-driven utveckling i projektet	3
Tillämpa tillämpliga delar av Scrum eller Kanban med hjälp av Trello	3
Konsekvent tillämpa en kodstandard	3
Tillämpa kodgranskning (redovisas via inlämnat protokoll)	3
Presentera och försvara projektets design och planer i en poster	3

Beskrivningar

A Abstraktion

	Mål	Nivå	Redovisas
A 1	Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar	3	L

Beskrivning Abstraktion är en av de viktigaste programmeringsprinciperna. Vi vet att djupt, djupt nere under huven är allt bara ettor och nollor (redan detta är en abstraktion!), men ovanpå dessa har vi byggt lager på lager av abstraktioner som låter oss tala om program t.ex. i termer av struktar och procedurer.

Proceduren `ritaEnCirkel(int radie, koordinat center)` utför beräkningar och tänder individuella pixlar på en skärm, men i och med att dessa rutiner kapslats in i en procedur med ett vettigt namn, där indata är uttryckt i termer av koordinater och radie har vi abstraherat bort dessa detaljer, och det blir möjligt att rita cirklar tills korna kommer hem utan att förstå hur själva implementationen ser ut.

Väl utförd abstraktion döljer detaljer och låter oss fokusera på färre koncept i taget.

Du bör ha en klar uppfattning om bland annat:

- Varför det är vettigt att identifiera liknande mönster i koden och extrahera dem och kapsla in dem i en enda procedur som kan anropas istället för upprepningarna?
- Abstraktioner kan ”läcka”. Vad betyder det och vad får det för konsekvenser?
- Vad är skillnaderna mellan ”control abstraction” och ”data abstraction”? (Du kan läsa om dessa koncept på t.ex. Wikipedia.)

	Mål	Nivå	Redovisas
A 2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt	3	L

Beskrivning I samband med redovisning bör du kunna förklara skillnaden/likheterna mellan procedurell abstraktion och objektorienterad abstraktion.

	Mål	Nivå	Redovisas
A 3	Demonstrera förståelse för designprincipen informationsgömning i ett C-program med hjälp av `.c` och `.h`-filer	4	L,G

Beskrivning

Informationsgömning på engelska översätts till ”information hiding”.

Relatera gärna till koncept från Java, som gränssnitt respektive implementation.

B Arv

	Mål	Nivå	Redovisas
B 4	Använda arv, metodspecialisering och superanrop i ett program som drar nytta av subtypspolymorfism	3	L

Beskrivning Metodspecialisering på engelska översätts ofta till ”overriding” men även ”[method] specialisation”.

Metodspecialisering avser alltså när en subklass tillhandahåller en mer specifik implementation av en metod. Ponera klasserna `Egendom` och `Hus` där `Hus` ärver av `Egendom`. Man kan tänka sig att det finns en metod `skatt()` för `Egendom` som beräknar en viss grundskatt baserat på något givet värde, men att `Hus` tillhandahåller en egen implementation av `skatt()` som är *mer specifik* för just hus, t.ex. gör vissa avdrag etc. som gäller specifikt för hus. Man kan tänka sig att den mer specifika `skatt()`-metoden anropar den mer generella för återanvändning (räkna ut grundskatten).

Polymorfism är möjligheten att behandla olika typer av värden genom ett gemensamt gränssnitt eller möjligheten att applicera en funktion på flera olika typer av värden. Det finns många olika typer av polymorfism – studera detta begrepp vidare!

Subtypspolymorfism avser polymorfism mellan subtyper, t.ex. `Hus` och `Egendom` ovan och husets möjlighet att uppträda/bli behandlat som en generell form av egendom. (Javas dynamiska bindning ger här att huset ändå bibehåller sitt specifika beteende.)

	Mål	Nivå	Redovisas
B 5	Använda både överlagrade och specialiserade konstruktorer på lämpligt sätt i programmering	3	L

Beskrivning Ytterligare beskrivning överflödig.

	Mål	Nivå	Redovisas
B 6	Förklara hur arvsbegreppet har använts i ett program för att separera genomskärande åtaganden	4	G,T

Beskrivning Fördjupa dig i konceptet ”genomskärande åtaganden”!

Genomskärande åtaganden är en försvenskning av begreppet ”separation of concerns”. Ett åtagande är här något som (en del av) programmet måste göra, ”ett stycke funktionalitet” slarvigt uttryckt, som kan vara ”direkt” (beräkna X) eller ”indirekt” (logga händelse under beräkning av X så att programmets beteende kan följas vid en krasch).

Separation av åtaganden handlar om att inte blanda (eng. tangle) de olika implementationerna av olika funktionalitet. Om man t.ex. vill ändra på hur den ovan nämnda loggningen går till skall man inte behöva blanda in implementationen av beräkningen av X.

Jämför gärna med aspektorienterad programmering!

	Mål	Nivå	Redovisas
B 7	Förklara hur man kan undvika s.k. ”bräckliga basklasser” vid arv	5	G,T

Beskrivning Bräckliga basklasser heter på engelska ”fragile base classes”. Vad betyder detta? Finns det strategier för att undvika dem? Vad får det för konsekvenser?

C Design

	Mål	Nivå	Redovisas
C 8	Förklara hur designprinciperna modularisering och ”divide and conquer” har använts i ett eller olika program	3	G,T

Beskrivning Modularisering förklaras bl.a. ytligt i C 11 nedan. ”Divide and conquer” syftar på det successiva nedbrytandet av en uppgift i allt mindre beståndsdelar som blir allt enklare att lösa.

	Mål	Nivå	Redovisas
C 9	Förklara begreppet designmönster samt visa hur minst ett designmönster har använts i ett icke-trivialt program	3	G

Beskrivning Begreppet designmönster avser kända designlösningar eller ”best practises” för ett specifikt ”problem”. Återanvändandet och kodifieringen av kända lösningar på *designnivå* var ett stort steg framåt för datavetenskapen (och mjukvaruindustrin i stort) eftersom kända lösningar är *välförstådda* t.ex. med avseende på sido-effekter av designvalet.

	Mål	Nivå	Redovisas
C 10	Demonstrera grundläggande förståelse för objektorienterad analys och design genom att redogöra för ett icke-trivialt programs design och analysen som ligger till grund för designen	4	G

Beskrivning Hur kommer man fram till vilka egenskaper eller vilket beteende en klass skall ha? Hur kommer man fram till vilka klasser som finns i ett system? Hur kommer man fram till vilka åtaganden dessa klasser skall ha?

Nosa gärna också på tankarna på icke-funktionella krav, s.k. ”-ilities” för att skapa en förståelse för ytterligare en dimension av design som komplicerar processen avsevärt. (Detta ämne täcks djupare av en annan, valbar, kurs på år 3.)

	Mål	Nivå	Redovisas
C 11	Visa hur man kan separera gränssnitt från implementation med hjälp av Java-interfaces	4	L,G

Beskrivning Separation av gränssnitt från implementation är viktigt för att möjliggöra interna förändringar av implementationen utan att externa klienter behöver t.ex. kompileras om.

Javas **`interface`**-konstruktion är ett *förtingligat* gränssitt som kan knytas till en klass som implementerar det. Ett **`interface`** kan ses som ett kontrakt och implementationen av ett **`interface`** kan ses som ett löfte att uppfylla kontraktet. En klass som implementerar ett **`interface`** måste åtminstone nominellt uppfylla detta kontrakt.

Detta mål handlar inte bara om att förstå principen för *hur* man gör en separation av detta slag utan också *varför*. Det sistnämnda bör guida det praktiska användandet och kan med fördel förklaras vid en redovisning.

D Dokumentation

	Mål	Nivå	Redovisas
D 12	Dokumentera icke-triviala modulers gränssnitt så att någon utomstående kan programmera mot dem	3	L

Beskrivning Bra dokumentation är av största vikt vid utveckling. Vid en kort kurs som denna är det svårt att uppleva nyttan med dokumentation som man själv skriver eftersom det inte hinner gå tillräckligt lång tid under kursen för att sådant man utvecklat skall falla tillräckligt i glömska[1].

Det är svårt att balansera mängden dokumentation som krävs för att beskriva något. Vem vänder man sig till? Vad kan man förvänta sig hos den som läser? Vad vill denne åstadkomma?

Vad är en bra balans mellan för lite information och för mycket? Vad är en lämplig detaljnivå?

Hur mycket av den interna implementationen bör man beskriva? Varför?

Hur beskriver man komplexa och tvetydiga processer?

I ML är pre- och postvillkor bra sätt att dokumentera förväntningar och löften på ett sätt som inte exponerar onödiga detaljer. Javaprogram har i regel tonvis med sido-effekter – vad får det för konsekvens för pre- och postvillkor?

E Genericitet

	Mål	Nivå	Redovisas
E 13	Använda void-pekare i C-program för att uppnå genericitet på ett relevant sätt, t.ex. en datasamling som kan lagra godtyckligt data	3	L

Beskrivning Det är inte en bra idé att ha kopior av kod med någon liten förändring, t.ex. en separat implementation av en heltalslista och en flyttalslista. Den underliggande logiken är densamma, och hittar man ett fel i en kopierad del måste man komma ihåg att ändra på samma ställe i alla kopior, etc.

[1] Ta gärna fram någon gammal inlämningsuppgift i ML från PKD och försök följa logiken och ändra i den.

I C använder man s.k. `void`-pekare för att göra en datastruktur generell, t.ex. gå från en lista av heltal till en lista som kan hålla godtyckligt data. Detta är ett extremt vanligt C-idiom som även förekommer i C:s standardbibliotek. En `void`-pekare är en pekare till en minnesplats med okänt innehåll, dvs. C-kompilatorn vet inte vad som finns där och hur man skall använda minnet, eller hur stort det är. Visa att du behärskar dessa idom och förstår deras konsekvenser genom att använda dem på ett lämpligt sätt i ett program.

	Mål	Nivå	Redovisas
E 14	Använda parametrisk polymorfism för ökad ”statisk typsäkerhet” vid interaktion med Javas standardbibliotek	3	L

Beskrivning Typsäkerhet avser att ett värde alltid används på tillåtet sätt, t.ex. att en bit minne som råkar hålla en baseballspelare med ett heltal för skostorlek på bytes 8–11 används på detta sätt. C är inte ett typsäkert språk vilket tillåter oss att t.ex. spara strängen `"Hej"` på skostorlekens plats, och sedan läsa dessa *fyra* bytes som en (abnorm) sko. Java är ett typsäkert språk, och priset man betalar för detta är kontroller under programmets körning som orsakar *undantag* om en operation skulle leda till ett typfel.

Ponera en lista i Java där elementpekarna är av typen `Object`. En sådan lista kan innehålla vad data som helst, men vanligt är att man är intresserad av en lista av basebollspelare etc.

Javas stöd för parametrisk polymorfism (även kallat generics) tillåter oss att skapa datastrukturer parametriserade över typer; parametrar vilka måste instantieras vid användande av strukturerna. Javas standardbibliotek formligen kryllar av klasser med typparametrar. Typparametrarna tillåter oss att uttrycka i kod att ”detta är en lista av hermeliner” vilket medger möjligheten att signalera kompileringsfel om man försöker smyga in katt ibland hermelinerna.

Visa att du förstår konceptet parametrisk polymorfism och hur det kan användas för att flytta fel från körning (undantag) till kompilering (kompileringsfel).

	Mål	Nivå	Redovisas
E 15	Designa med parametrisk polymorfism för att göra relevanta delar av ett program mer generellt	4	L,G

Beskrivning Vid designen av en eller flera klasser – dra nytta av parametrisk polymorfism för att slippa ange en typ vid definitionen (detta är en äppellåda) och istället ange den vid användandet av lådan (detta är en låda, som i detta fall råkar innehålla enbart äpplen).

En informell guldstjärna utgår för användanden som inte är samlingsklasser (collections).

F Imperativ programmering

	Mål	Nivå	Redovisas
F 16	Översätta mellan rekursiva och iterativa lösningar av samma problem samt diskutera för- och nackdelar med sido-effekter	3	L

Beskrivning Tänkvärt: Alla iterativa lösningar kan beskrivas med rekursion men gäller det omvända?. Ibland ger rekursion en ”historik” över vilken väg genom en datastruktur man tagit som försvinner vid en naiv översättning till iteration.

	Mål	Nivå	Redovisas
F 17	Ta fram loopinvarianter för loopar och resonera om deras korrekthet	4	L,G

Beskrivning Studera konceptet loopinvarianter. Det ingår inte i kursen att kunna göra formella bevis, men man bör skaffa sig en uppfattning eller kanske snarare en intuition för varför en loopinvariant är korrekt.

	Mål	Nivå	Redovisas
F 18	Förklara skillnaden mellan deklarativ och imperativ; diskutera deras respektive för- och nackdelar	5	G,T

Beskrivning Detta är delvis en fråga med tydliga rätt och fel, men delvis väldigt filosofisk. I viss utsträckning är det viktigare att kunna diskutera frågan än att kunna svara på den.

G Inkapsling

	Mål	Nivå	Redovisas
G 19	Använda kopiering eller liknande för att undvika representationsläckage i ett C- eller Javaprogram	3	L

Beskrivning Ett objekts (denna term omfattar även C-struktar) representation är de ”subobjekt” som bygger upp det sammansatta objektet. En länkad listas representation innehåller t.ex. dess länkobjekt eftersom dessa *ägs* av listan, men inte listdatat eftersom det inte är en del av själva listobjektet.

Representationsläckage uppstår när subobjekt är synliga utanför det objekt till vilket de konceptuellt tillhör. Om man t.ex. kunde få ett handtag till en länk i en sorterad länkad lista utanför listan kan man göra borttagningar eller insättningar som förstör listans sorteringsordning.

Det är vanligt att subobjekt flyttas mellan objekt och byter ägare, eller skapas utanför objektet och flyttas in i det vid initiering. Mainstreamprogramspråk saknar dock koncept för ägarskap vilket kan leda till representationsläckage om man inte är aktsam.

	Mål	Nivå	Redovisas
G 20	Använda åtkomstmodifierare för att styra åtkomst	3	L

Beskrivning Styr åtkomst till både privata metoder och data på ett lämpligt sätt och motivera användandet.

	Mål	Nivå	Redovisas
G 21	Använda nästlade och inre klasser för att kapsla in privata beståndsdelar i ett sammansatt objekt	4	L,G

Beskrivning Notera den syntaktiskt subtila men relativt viktiga skillnaden mellan inre och nästlade klasser. Ett annat namn på sammansatta objekt är aggregat.

	Mål	Nivå	Redovisas
G 22	Resonera kring hur inkapsling både kan försvåra och underlätta testning (ta gärna hjälp av kodexempel)	5	G,T

Beskrivning Konkreta exempel hjälper. Man skjuter sig i foten (och lurar sig själv på ett dåligt sätt) om man försöker komma till insikt utan att göra praktiska experiment med implementationer. Försök t.ex. kapsla in ”allt” och försök sedan testa. På motsvarande sätt – kapsla inte in någonting och försök sedan uttala dig om möliga intervall på värden etc.

H Klasser och objekt

	Mål	Nivå	Redovisas
H 23	Skriva minst två jämförelsemetoder för minst två sammansatta objekt	3	L

Beskrivning Sammansatta objekt kallas ibland för aggregat – ett objekt som består av flera andra objekt. Detta är fallet för de flesta intressanta objekt. En jämförelsemetod i ett objekt A tar som argument ett objekt B och returnerar `true` om objekten är *lika*, annars `false`. Hur ”lika” definieras kan vara olika – ibland används konceptet *identisk* för att avse att A och B är samma objekt; ibland används konceptet *strukturellt lika* eller *ekvivalenta* för att avse att A och B är två olika objekt men som ändå i enligt den gällande formen av likhet kan anses vara ”utbytbara”, t.ex. A och B är två olika kakburkar som innehåller samma kaka.

Använd Java och ge prov på båda formerna av likhet.

	Mål	Nivå	Redovisas
H 24	Redovisa förståelse för skillnaden mellan värdesemantik och referenssemantik med hjälp av ett kodexempel	3	L

Beskrivning Slå upp termerna, pröva gärna även engelska: ”value semantics” respektive ”reference semantics”. Se även parameteröverföring.

	Mål	Nivå	Redovisas
H 25	Beskriva en klass klassinvarianter i kommentarer, samt resonera om hur inkapslingen slår vakt om invarianterna	4	L,G,T

Beskrivning En invariant är en utsaga som alltid är sann. En klassinvariant är en utsaga som alltid är sann för alla instanser av den klassen. T.ex. kan en invariant för klassen ”Räkning” att dess kund inte är `null` eller att dess belopp alltid är ≥ 0, etc.

Instuderingstips: Du kan läsa om klassinvarianter på nätet. Wikipedia har en bra startsida, men använd ett eget exempel från ett program som du redan har skrivit.

	Mål	Nivå	Redovisas
H 26	Redovisa förståelse för abstrakta klasser och metoder, samt deras relation till Java-interface	5	G,T

Beskrivning Beakta bland annat.:

- Hur skiljer de sig?
- Hur är de lika?
- När används de och vad har de för konsekvenser på programmen i vilka de används?
- Varför är multipelt interface-arv tillåtet och väldefinierat medan multipelt implementationsarv inte är det? (I Java, alltså.)

I Metodik

	Mål	Nivå	Redovisas
I 27	Förklara innebörden av, och tillämpa på ett konsekvent sätt, defensiv programmering i ett program	3	L,G

Beskrivning Studera konceptet defensiv programmering. Vid tillämpning, använd checklistor som finns i kursmaterialet och fundera på hur man kan kritisera dem.

	Mål	Nivå	Redovisas
I 28	Fånga på lämpliga platser och hantera på ett lämpligt sätt kontrollerade och okontrollerade (relevanta) undantag i ett program	3	L,G

Beskrivning Fundera över vad som är en lämplig plats och hur man på ett lämpligt sätt hanterar ett fel. Är det t.ex. någonsin relevant att fånga ett `NullPointerException`? Vilka fel kan t.ex. hanteras utan att programmet snarare borde följa principen ”crash, don’t trash”?

	Mål	Nivå	Redovisas
I 29	Utöver föregående mål, demonstrera fördjupad förståelse för undantagshantering i Java genom att kasta existerande och egendefinierade kontrollerade och okontrollerade undantag i ett program	4	L,G

Beskrivning Demonstrera med relevanta undantag, du kan eventuellt utgå från ett program som du redan skrivit.

J Minneshantering

	Mål	Nivå	Redovisas
J 30	Demonstrera förståelse för skillnaden mellan allokering på stacken och allokering på heapen med hjälp av ett C-program	3	L,G

Beskrivning Läs ”C: addendum om stack & heap” som finns bland materialet från 2012.

Pröva gärna också att t.ex. jämföra prestanda mellan ett program som allokerar mycket på stacken med ett som allokerar mycket på heapen (t.ex. det rekursiva fibonacci-programmet från fas 0/sprint 0 där samtliga heltal mallokeras på heapen). Vad får det för effekt på uträkningens prestanda?

	Mål	Nivå	Redovisas
J 31	Demonstrera förståelse för minneshantering i C genom att skriva ett program med dynamiska strukturer som är fritt från minnesläckage och argumentera för varför det är så (och verifiera med valgrind)	4	L

Beskrivning Redovisa bland annat för:

- Hur uppstår minnesläckage?
- Vilken del av ett program ansvarar för att frigöra minne?
- När avgör man att en bit allokerat minne är ”färdiganvänt” och går att frigöra?

– Vad kan hända om ett allokerat utrymme frigörs ”för tidigt”?
– På vilket sätt hjälper valgrind dig att undvika minnesläckage?

	Mål	Nivå	Redovisas
J 32	Förklara skillnaden mellan C:s manuella minneshantering och hur Java hanterar minne och beskriv för ett lämpligt Java-program när minne allokeras och frigörs	4	L,G

Beskrivning Redovisa bland annat för:

– När är ett objekt att betrakta som skräp som kan städas bort?
– När städas skräp bort?
– Hur kan jag som programmerare ta reda på när ett visst objekt städats bort?
– Finns det någon typ av minnesfel som jag kan få i C med pekare som jag inte kan få med referenser i Java?
– När allokeras minne och hur tar Java reda på hur många bytes ett visst objekt kräver?
– Hur tar jag som programmerare reda på hur många bytes ett visst objekt kräver?

Javas automatiska skräpsamlare kan konfigureras på många sätt, så tänk dig för varenda gång du säger ”så här fungerar Javas GC”.

Bonus: hur fungerar ”weak references”? Hur relaterar de till pekare?

	Mål	Nivå	Redovisas
J 33	Jämför två metoder för automatisk skräpsamling	5	G,T

Beskrivning Lämpliga metoder avser t.ex. mark-sweep GC, reference counting och generational GC. Diskutera deras för- och nackdelar.

K Modularisering

	Mål	Nivå	Redovisas
K 34	Specificera tydliga gränssnitt mellan moduler i ett program som bör brytas ned i flera moduler (och implementera)	3	L,G

Beskrivning Att dela upp ett program i mindre beståndsdelar är i regel att föredra framför ett *monolitiskt* program. En anledning till detta är att det underlättar *parallellutveckling* – dvs. flera programmerare kan vara inblandade i ett projekt samtidigt och jobba på olika delar och därigenom förkorta utvecklingtiden (och öka kvaliteten).

Naturligtvis duger det inte med att göra vilken uppdelning som helst! Man bör beakta vad som är en naturlig uppdelning; ett program som modellerar ett brädspel kanske kan delas upp i en modul för spelets logik (hur pjäserna interagerar med brädet och varandra) och en modul för spelets grafik. Genom att separera logiken från grafiken underlättar man också för ett framtida utbytande av spelets grafik.

Pröva din *modulariseringsstrategi* genom att implementera ett program och notera vilka förändringar du måste göra när din insikt i programmet fördjupas.

	Mål	Nivå	Redovisas
K 35	Givet ett icke-trivialt program uppdelat i moduler som i ovanstående mål, resonera kring begreppen coupling och cohesion för några av modulerna	4	L,G

Beskrivning

Termerna ”low coupling” och ”high cohesion” beskriver två egenskaper som krävs för att ett program skall anses välmodulariserat. Med low coupling avses att två moduler endast skall vara löst sammanknutna, så att det är möjligt att byta ut en av dem utan att göra några egentliga förändringar i den andra. Med high cohesion avses att alla delar som finns i en modul skall ha en ”hög grad av samhörighet”, dvs. att alla delar av modulen avser samma aspekt av programmet. (Varför?)

Termerna ovan är enkla att beskriva separat från faktisk kod, men för att förstå dem måste du kunna relatera dem till din egen kod, beskriva coupling mellan moduler, cohesion inom moduler, och och eventuellt göra motiverade förändringar i den tidigare designen.

	Mål	Nivå	Redovisas
K 36	Utveckla resonemanget i (4) med en diskussion om separation of concerns och hur detta har uppnåtts (alt. kan/inte kan) i programmet i fråga	5	L,G

Beskrivning

Undersök termen/begreppet ”separation of concerns” (leta bland annat men inte enbart i litteraturen om aspekt-orienterad programmering). Vad betyder detta begrepp? Relatera det till diskussionen om coupling och cohesion i föregående mål.

L Parallelprogrammering

	Mål	Nivå	Redovisas
L 37	Använda Fork/join i java.util.concurrent för att parallellisera relevanta delar av ett tidigare program	3	L

Beskrivning

	Mål	Nivå	Redovisas
L 38	Utöver (3), mät körtiden före och efter och förklara mätresultaten	4	L,G

Beskrivning

	Mål	Nivå	Redovisas
L 39	Utöver (4), justera eventuella parametrar (t.ex. sequential cutoff, antalet trådar) för att se hur prestanda påverkas, mät och förklara igen	5	L,G

Beskrivning

M Pekare

	Mål	Nivå	Redovisas
M 40	Översätta mellan C:s array-notation och pekararitmetik	3	L,G

Beskrivning Arrayer i C är egentligen inget mer än en behändig notation för pekararitmetik. Demonstrera att du förstår likheterna mellan pekare till konsekutiva minnesblock av element av en typ, och arrayer i C i ett i övrigt vettigt program.

Tips: Argumentvektorn till `main`-funktionen.

	Mål	Nivå	Redovisas
M 41	Använda pekare för att skapa länkade strukturer och använda pekare till stackvariabler för värdeöverföring mellan funktionsanrop	3	L,G

Beskrivning En länkad struktur är en struktur uppbygd av separat-allokerade objekt som pekar till varandra, t.ex. en länkad lista eller ett träd. Länkade strukturer växer och krymper dynamiskt, dvs. deras storlek varierar med körning och under körning och kräver i regel minnesallokering på heapen.

Läs ”C: addendum om stack & heap” som finns bland materialet från 2012.

Referenssemantik och värdesemantik tas upp redan i fas 0/sprint 0.

	Mål	Nivå	Redovisas
M 42	Använda pekare till pekare (dubbelpekare) på ett lämpligt sätt i ett program	4	L,G

Beskrivning

En pekare till en pekare är en väldigt kraftfull mekanism som kan vara förvirrande ibland (t.ex. att man blandar ihop vad de olika pekarna pekar på).

Tips: iteratorer till datasamlingar som ger möjlighet att länka in och länka ur.

Argumentvektorn i `main`-funktionen är visserligen en pekare till en pekare, men skall *inte* användas för demonstration här.

N Pragmatics

	Mål	Nivå	Redovisas
N 43	Demonstrera på lämpligt sätt förståelse för termerna kompilering, länkning, interpretering och JIT-kompilering och hur dessa används i C och Java	3	L

Beskrivning Fundera över skillnaderna mellan länkning i C och Java och när länkning sker.

Ledning: med hjälp av enkla prestandamätningar kan man mäta tiden för JIT-kompilering och även dess inverkan på presetanda hos kod efter att den kompilerats.

	Mål	Nivå	Redovisas
N 44	I samband med ovanstående, förklara statisk och dynamisk bindning och exemplifiera med programkod	4	L

Beskrivning Bland annat bör du:

- påvisa prestandaskillnaden mellan dessa två typer av bindning, och
- exemplifiera effekten på återanvändning av kod.

O Profilering och optimering

	Mål	Nivå	Redovisas
O 45	Använda profileringsverktyg för att visa var ett redan skrivet program tillbringar mest tid.	3	L,G

Beskrivning ”Premature optimisation is the root of all evil” är ett citat från Donald Knuth. Ett delmål i denna kurs är att lära sig att använda tillämpliga verktyg för att förstå ett programs beteende under körning med avseende på prestanda, dvs. efter att programmet är implementerat ta fram det data som sedan krävs för att göra en faktisk optimering.

Lämpligen tar du fram vilka delar av programmet som tar längst tid att köra respektive körs flest gånger. Några frågor att besvara:

- Vad är lämplig granularitetsnivå på ”delarna” som nämns ovan – och varför?
- Hur tar du fram detta data?
- Hur vet du att datat är ”rätt”?
- Hur kan du använda datat?

	Mål	Nivå	Redovisas
O 46	Med ledning resultatet från föregående mål, optimera programmet på ett förtjänstfullt sätt.	4	L,G

Beskrivning Optimera programmet med hjälp av profilen ovan, motivera ansatsen och demonstrera och förklara prestanda*ökningen* med hjälp av en ny profil för det optimerade programmet

	Mål	Nivå	Redovisas
O 47	Samma som mål (3) och (4) men för minnesanvändning	5	L,G

Beskrivning Beakta inte enbart exekveringstid och frekvens, men minnesåtgången för strukturer och funktioner/metoder. Kanske finns det någon samband mellan lång exekveringstid och högt minnesanvändande? Kan man minska exekveringstiden genom att minska mängden allokerat minne?

P Refactoring

	Mål	Nivå	Redovisas
P 48	Tillämpa enkla refactoringmönster på ett program från en tidigare sprint i ett verktyg med särskilt stöd för refactoring	3	L,G

Beskrivning Enkla refactoringmönster avser t.ex. namnändringar, och att flytta tillstånd och/eller beteenden från en plats till en annan. Både Netbeans och Eclipse har inbyggt stöd för refactoring. En bra utgångpunkt för att lära sig om refactoring är `refactoring.com`.

	Mål	Nivå	Redovisas
P 49	Utöver föregående, använd mer komplicerade mönster och diskutera hur programmet förbättras	4	L,G

Beskrivning Mer komplicerade refactoringmönster avser t.ex. extrahera en metod eller ersätt villkor med polymorfism.

Q Testning

	Mål	Nivå	Redovisas
Q 50	Ta fram lämpliga enhetstester för ett icke-trivialt program, motivera dem, och förklara vad testerna visar	3	L

Beskrivning Det står dig fritt fram att använda t.ex. CUNIT, JUnit, eller liknande. Detta kan redovisas både under projektet eller tidigare. Föreläsningar om testning kommer under fas 1.

Enhetstester fokuserar på enskilda funktioner och metoder. Ett bra enhetstest kan köras helt automatiserat, är läsbart och prövar endast en enda logisk funktion/koncept. Enhetstester har flera funktioner, bl.a. gränssnittsdokumentation genom exempel, stöd vid debuggning av kod genom dokumentation av felaktiga indata, och i testdriven utveckling (TDD) skrivs testerna före koden de skall testa för att bena ut specifikationer och sätta fingret på oklarheter.

	Mål	Nivå	Redovisas
Q 51	Ta fram en uppsättning regressionstester för en del av ett program, motivera dem, och förklara vad testerna visar	4	G

Beskrivning En uppsättning regressionstester är ingenting man tar fram, utan de växer fram gradvis över tid. Att *hantera och administrera* regressionstester handlar inte bara om att ta fram nya tester vid behov (vilka?) utan också om ta bort tester (varför och när?).

Regressionstestning är viktigt för underhåll av kod eller vidareutveckling av en stor kodbas där effekterna av en förändring är svåra och kostsamma att spåra ”manuellt”. Om man har en uppsättning regressionstester kan man köra dessa mellan förändringar för att se om förändringarna fått tester att misslyckas (alternativt börja fungera).

Regressionstester kan ha många olika format, bl.a. existerande enhetstest eller integrationstest, såväl som testfall skapade i samband med buggrapporter och buggfixar. I stora system kan regressionstesterna lätt bli svårhanterligt många.

	Mål	Nivå	Redovisas
Q 52	Tillämpa automatiserad kontinuerlig integrationstestning, motivera testerna och förklara vad de visar	5	L,G

Beskrivning Ett test är inte ett test om det inte är automatiserat. Integrationstester måste därför kunna köras på samma automatiska vis som enhetstester.

Integrationstestning är konsten att ta olika moduler och visa att de fungerar tillsammans. Integrationstestning är separat från enhetstestning som försöker visa funktioner i enskilda moduler, och kan också vara betydligt mer komplicerat än enhetstestning eftersom det kan vara svårt att koppla samman moduler eller påvisa i vilken modul ett fel uppstår.

R Verktyg/Debuggning

	Mål	Nivå	Redovisas
R 53	Använda GDB för debuggning	3	L

Beskrivning Avser att kunna använda GDB för att stega genom ett C-programs exekvering och sätta breakpoints (ex. b, c, s, n). Redovisa genom att återskapa ett förlopp från en skarp buggjakt med de relevanta stegen och förklara buggen.

	Mål	Nivå	Redovisas
R 54	Jämföra GDB och Netbeans/Eclipse	4	L

Beskrivning För ett enkelt resonemang om skillnaderna mellan verktygen och hur de underlättar debuggning.

S Verktyg/Dokumentation

	Mål	Nivå	Redovisas
S 55	Använda doxygen eller JavaDoc för att extrahera kommentarer från kod till lämplig API-dokumentation	3	L

Beskrivning Endast lämplig att demonstrera i ett större program med flera moduler där gränssnitten behövts dokumenteras.

	Mål	Nivå	Redovisas
S 56	Använda LaTeX	3	L

Beskrivning Avser användande av LaTeX för skriftlig produktion med samma kodhygien som för vanlig programmering, t.ex. i essä eller i projektrapporten.

T Verktyg/Emacs

	Mål	Nivå	Redovisas
T 57	Använda Emacs för editering	3	L

Beskrivning Avser förmåg att använda Emacs *utan mus* för att navigera och editera text, öppna filer, bläddra mellan och hantera buffrar, använda kill & yank och mark-stacken.

	Mål	Nivå	Redovisas
T 58	Utöver (3), dra nytta av Emacs inbyggda stöd för programmering	4	L

Beskrivning Avser stöd för kompilering och felsökning, textkomplettering el. snippets, samt användande av tags eller motsvarande för att navigera kod.

	Mål	Nivå	Redovisas
T 59	Utöver (4), dra nytta av Emacs möjligheter till utökning för programmering	5	G

Beskrivning Avser att konfigurera Emacs för Cedet & ECB och använda det i sin programmering. Reflektera gärna över likheter/skillnad med Eclipse/Netbeans.

U Verktyg/Kompilering

	Mål	Nivå	Redovisas
U 60	Använda Make	3	L

Beskrivning Avser att åtminstone skriva och köra enklare makeskript för att hantera *byggberoenden* i ett C-program.

	Mål	Nivå	Redovisas
U 61	Använda Lint	3	L

Beskrivning Avser att åtminstone förklara var i arbetsflödet Lint passar in, förklara utdata från Lint, samt vad för slags fel Lint kan och inte kan hitta och demonstrera hur det har använts framgångrsikt.

	Mål	Nivå	Redovisas
U 62	Använda valgrind	3	L

Beskrivning Avser användning av verktyget på ett eget program för att identifiera och åtgärda minnesläckage.

V Verktyg/Terminalen

	Mål	Nivå	Redovisas
V 63	Behärska grundläggande terminalkommandon	3	L,G

Beskrivning Avser åtminstone `cd`, `ls`, `grep`, `find`, `xargs`, `&`, `^z`, `bg`, `fg`, omdirigering, pipes och miljövariabler.

W Verktyg/Versionshantering

W 64

Mål	Nivå	Redovisas
Behärska grundläggande versionshantering	3	L

Beskrivning Avser åtminstone kommandona add, blame, clone, commit, pull, merge och log. Gärna även push.

Redovisas lämpligen med ett litet egenkomponerat scenario som exercerar samtliga kommandon.

X Strukturmål/Misc

X 65

Mål	Nivå	Redovisas
Implementera minst tre uppgifter komplett och i två olika språk	3	G

Beskrivning Att skriva ett komplett program driver fram problem och insikter som man inte får om man t.ex. bara skriver små tillrättalagda snuttar. För att bli godkänd på kursen måste du har skrivit minst tre kompletta program som minst uppfyller sin specifikation, och har en lämplig nivå av tester och dokumentation.

Redovisas med fördel för din gruppledare så snart du skrivit tre kompletta program. Gör en *mycket kort* dragning för alla där du för varje program anger vilka delar av specifikationen som var svårast att designa och implementera, och vad du skulle göra om ifall du var tvungen att skriva om dem från grunden. Undvik att ge ut lösningar på problem som andra kanske brottas med eller kommer att brottas med. Vid frågor på detta, prata med din gruppledare.

Projektarbetet räknas *inte* som ett av dessa tre program.

X 66

Mål	Nivå	Redovisas
Arbeta i minst fyra olika par under projektet (Intygas)	3	I

Beskrivning Parprogrammering blir allt vanligare i arbetslivet. Personkemi spelar stor roll för hur par fungerar och vi gör en poäng av att sätta som ett explicit krav att rotera par minst tre gånger under projektets gång.

X 67

Mål	Nivå	Redovisas
Redovisa ett mål ur G eller T-kategorin som en essä	4	W

Beskrivning Skriftlig framställning är en viktig del av samtliga kurser på universitetet. Det är oerhört viktigt och användbart att kunna lägga upp en text eller ett resonemang, och veta hur man skall rikta sig till en specifik målgrupp.

Betrakta formkrav och språkliga krav som lika viktiga som ämneskraven.

Här är några tips:

(a) För mycket text på en sida utan rubriker blir ogenomträngligt och osexigt

(b) Tänk alltid igenom vad ditt syfte är och vad du vill förmedla till läsaren (vad skall läsaren ta med sig)

(c) Fråga för varje ”kapitel” eller stycke vad de har för relation till det ovan nämnda budskapet – om det är oklart kan man fråga sig om det är motiverat att ha kvar

(d) En text skrivs eller läses sällan linjärt

(e) 66-72 tecken per rad är en optimal radlängd för ögat att följa

(f) Hur du formger din text är också en del av budskapet och hjälper till att driva hem poängerna

(g) Slarvfel och stavfel får dig tråkigt nog att framstå som oseriös och dum

(h) Använd verktyg (t.ex. `M-x flyspell` i Emacs) men förstå deras begränsningar

(i) Använd textverktyg som gör det enkelt att göra förändringar i layout utan för mycket handpåläggning

(j) ”Bara för att du har skrivit det är det inte en bra idé att ha det kvar i texten”

(k) När man granskar en text skall man läsa den både ”nära” (hur skall den här meningen vara) och ”på avstånd” (kanske poängen blir tydligare om man ändrar ordningen på kapitel 1 och 2)

(l) Be alltid någon annan läsa igenom texten och förklara för dig vad det faktiskt står i den

(m) Uppsala Universitet ger dig som student tillgång till professionell skrivhjälp, även för en uppgift som denna

(n) Du skall berätta en story och storyn hjälper läsaren att förstå och knyta samman allt.

(o) Bara för att du gjorde X före Y behöver de inte komma i den ordningen i texten, även om det antyder att du gjorde dem i omvänd ordning

Fundera över vad för slags ton man bör hålla i en essä, hur man klara av att vara saklig utan att bli tråkig, och fundera över vilken din stil är.

Det stilistiska är alltid viktigt, oavsett om man skriver ett sms eller en avhandling. Nedanstående briljanta citat kan förhoppningsvis både skänka glädje och inspirera:

> This sentence has five words. Here are five more words. Five-word sentences are fine. But several together become monotonous. Listen to what is happening. The writing is getting boring. The sound of it drones. It's like a stuck record. The ear demands some variety. Now listen. I vary the sentence length, and I create music. Music. The writing sings. It has a pleasant rhythm, a lilt, a harmony. I use short sentences. And I use sentences of medium length. And sometimes, when I am certain the reader is rested, I will engage him with a sentence of considerable length, a sentence that burns with energy and builds with all the impetus of a crescendo, the roll of the drums, the crash of the cymbals-sounds that say listen to this, it is important. – Gary Provost

En begränsning är att essän skall vara på ca 1000 ord eller 6500 tecken. Längre är inte bättre, men det man skriver måste vara av en viss volym för att tillräckligt intressanta frågeställningar om skriftlig framställning skall uppkomma.

Du skall lämna in texten i inte mindre än två format:

– En PDF-fil genererad från LaTeX
– Filen typsatt i Google Docs

Vi kommer att använda PDF-annoteringar och/eller Google Docs för att kommentera.

Essän lämnas in per e-post till Tobias och din gruppledare. Tobias kan bli inbjuden att dela ett Google Docs-dokument på adressen `tobias.wrigstad@gmail.com`.

	Mål	Nivå	Redovisas
X 68	Hålla minst en välstrukturerad presentation vid gruppmöten (Bockas av vid tillfället)	3	G

Beskrivning Muntlig framställning är lika viktig som skriftlig. En presentation kan lätt bli tråkig om den som talar är oengagerad, och en talares nervositet smittar av sig på åhörarna som då tänker på allt annat än budskapet. Att läsa innantill från ett papper är döden, även om den text man skrivit är fantastisk. Man måste göra den levande!

En talare som glider in i oväntade kläder eller har ett ”tick” drar lätt uppmärksamheten till detta istället för till sitt budskap. Det kan vara en bra idé att undvika läderbyxorna (för någon definition av läderbyxor) och om det inte går själv berätta varför man måste ha dem på sig just idag, så åhöraren får det ur sitt system.

En presentation har på samma sätt som en skriven text ett budskap och man måste vara noga med hur man lägger upp presentationen för att vara säker på att åhöraren kan följa med. Det blir ännu viktigare att berätta vad man skall säga, och sammanfatta hela tiden vad man har sagt.

Bilder är bra, speciellt om man tänker sig att de skall användas som studiematerial senare, men de kan lätt göra presentationen stelbent och inflexibel.

Några tips:

– Ha inte för mycket text på varje bild

– Bilder *kan* vara bärare av mycket information, men kräver stor eftertanke vid skapandet
– Använd animationer etc. sparsamt
– Undvik att backa eller på annat sätt hoppa fram och tillbaka i presentationen
– Det är viktigt att vara engagerad – om du skall prata om något måste du övertyga dig själv om att det är spännande, för varför skulle man annars lyssna på dig?
– Be inte om ursäkt
– Lyssna på dig själv – säger du ”ehm...” mellan varje mening eller ”alltså” i början av varje mening? Går det att arbeta bort?

X 69

Mål	Nivå	Redovisas
Hålla en välstrukturerad verktygspresentation vid gruppmöte (Bockas av vid tillfället)	4	G

Beskrivning Se föregående mål. En verktygspresentation kan mer ha karaktären av en demonstration.

X 70

Mål	Nivå	Redovisas
Visa förmåga att förmedla sin kunskap genom att handleda ett pass	5	L

Beskrivning Att hålla en presentation utan förberedelser är vanligt – jämför t.ex. med ”elevator pitch” eller när någon frågar om något som man känner till. Vid labhandledning är det vanligt att man måste berätta om olika koncept etc.

Även om man inte kan en viss aspekt av kursen bör man kunna hjälpa någon med en diskussion, både lyssna och tala.

Redovisning sker med hjälp av ordinarie labassar som bitvis följer med dig och lyssnar på dina förklaringar etc. till andra. Obeservera att du inte får ta redovisningar.

Ett bra sätt att förbereda sig är att sätta sig in i labassens roll genom att titta på hur (särskilt de seniora) labassarna gör.

Y Strukturmål/Projekt

Y 71

Mål	Nivå	Redovisas
Använda parprogrammering	3	R

Beskrivning Redovisning sker genom en skriftlig personlig (enskild) reflektion som inkluderas i projektrapporten. Hur har parprogrammering fungerat? Hur har det påverkat kvaliteten och utvecklingstiden? Hur har parrotationerna fungerat? Vad kunde göras bättre? Etc.

Y 72

Mål	Nivå	Redovisas
Tillämpa test-driven utveckling i projektet	3	R

Beskrivning Redovisning sker genom en gemensam (för alla medlemmar i projektet) skriftlig reflektion över dess styrkor och svagheter i projektrapporten. Om ni skulle starta om projektet imorgon – vad skulle ni behålla och vad skulle ni göra annorlunda (och hur)?

Y 73

Mål	Nivå	Redovisas
Tillämpa tillämpliga delar av Scrum eller Kanban med hjälp av Trello	3	R

Beskrivning Redovisning sker genom en gemensam (för alla medlemmar i projektet) skriftlig beskrivning av den valda processen (på hög nivå) samt en reflektion över dess styrkor och svagheter i projektrapporten. Om ni skulle starta om projektet imorgon – vad skulle ni behålla och vad skulle ni göra annorlunda (och hur)?

Y 74

Mål	Nivå	Redovisas
Konsekvent tillämpa en kodstandard	3	R

Beskrivning Redovisas i projektrapporten genom en hänvisning till den kodstandard som använts med eventuella tillägg eller avvikelser, samt en *kort* diskussion om nyttan med kodstandard.[2]

Y 75

Mål	Nivå	Redovisas
Tillämpa kodgranskning (redovisas via inlämnat protokoll)	3	R

Beskrivning Redovisas i projektrapporten genom en *kort* diskussion om hur kodgranskning har

[2]Läs gärna Joel Spolskys åsikter i frågan i Great Software Writing I.

fungerat. Om ni skulle starta om projektet imorgon – vad skulle ni behålla och vad skulle ni göra annorlunda (och hur)?

Protokollen och anteckningarna från kodgranskningarna skall också bifogas, samt referenser till vilka filer som avses och vilken revision.

Y 76

Mål	Nivå	Redovisas
Presentera och försvara projektets design och planer i en poster	3	Se nedan

Beskrivning Redovisning sker i samband med posterseminariet. Information om detta kommer senare under kursen.

Z En första presentation

Z 77

Mål	Nivå	Redovisas
Kunna samtala avslappnat med andra gruppmedlemmar	–	G

Beskrivning Presentera dig kort på första gruppmötet. Berätta litet om vem du är och vad du vill åstadkomma med din utbildning generellt, och vad du har för ambitioner med denna kurs specifikt. Beskriv kort din programmeringsbakgrund. Detta kommer att underlätta för att forma programmeringspar under kursen.

Ta detta på allvar och använd det som en ”isbrytare” så att det blir lättare att delta aktivt i gruppmöten under kursen.