# TDP004 - Tenta

2020-01-15

## Regler

- All kod som skickas in för rättning ska kompilera och vara väl testad.
- Inga elektroniska hjälpmedel får medtas. Mobiltelefon ska vara avstängd och ligga i jacka eller väska.
- Inga ytterkläder eller väskor vid skrivplatsen.
- Student får lämna salen tidigast en timme efter tentamens start.
- Vid toalettbesök ska pauslista utanför salen fyllas i.
- All form av kontakt mellan studenter under tentamens gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av assistent, vakt eller examinator under tentamens gång.
- Frågor om specifika uppgifter eller om tentamen i stort ska ställas via tentasystemets kommunikationsklient.
- Systemfrågor kan ställas till assistent i sal genom att räcka upp handen.
- Endast uppgifter inskickade före tentamenstidens slut rättas.
- Ingen uppgift kan kompletteras under tentamens sista kvart.
- En uppgift kan som regel kompletteras tills den är antingen "Godkänd" eller "Underkänd". En uppgift bedöms som "Underkänd" om ingen markant förbättring skett sedan tidigare inlämning.
- Kompilerande kod, fullständig kravuppfyllnad och följande av god stil och goda konventioner enligt god programmeringssed är krav för att en uppgift ska bedömas "Godkänd".

| | |
|---|---|
| Hjälpmedel | En C++-bok (t.ex. Stroustrup)<br>Ett A4-ark med egna anteckningar |

# Information

## Betygssättning vid tentamen

Tentamen består av ett antal uppgifter på varierande nivå. Uppgifter som uppfyller specifikationen samt följer god sed och konventioner ges omdömet "Godkänd". Annars ges omdömet "Kompletteras" eller "Underkänd". Tentamen kräver två godkända uppgifter för betyg 3. Alla betygsgränser ses i tabell 1 och 2. *För betyg 3 har du alltid hela tentamenstiden, varken mer eller mindre.* (För student som från LiU fått rätt till förlängd skrivtid förlängs betygsgränserna i proportion till den förlängda skrivtiden.)

| **Tid** | **Lösta uppgifter** | **Betyg** | **Tillgodo** |
|---|---|---|---|
| 4 h | 3 | 5 | inget |
| 2.5 h | 2 | 4 | inget |
| 4 h | 2 | 3 | inget |
| 4 timmar | 1 | \| | löst uppgift |

**Tabell 1:** Betygsättning vid förtentamen (dugga), 4 uppgifter ges

| **Tid** | **Lösta uppgifter** | **Betyg** |
|---|---|---|
| 3 h $+B$ | 3 | 5 |
| 4 h $+B$ | 4 | 5 |
| 4 h $+B$ | 3 | 4 |
| 2 h $+B$ | 2 | 4 |
| 5 timmar | 2 | 3 |

**Tabell 2:** Betygsättning vid sluttentamen, 5 uppgifter ges

## Bonustid ($+B$)

*All bonustid gäller endast under den första ordinarie tentamen i samband med kursen (januari).* Varje moment i kursen som ger bonus ger 5 minuter extra tid för högre betyg på sluttentamen, upp till maximalt 45 minuter. Detta är markerat med $+B$ i tabellen där bonus räknas.

## Tillgodoräknanden

*Tillgodoräknanden gäller endast under den första ordinarie tentamen i samband med kursen.* Om du på förtentamen endast lyckas lösa en uppgift kan du tillgodoräkna motsvarande uppgift på sluttentamen (se tabell 1). Du har då bara en uppgift kvar till betyg 3. För högre betyg (om du löser mer än en uppgift på förtentamen) kan du inte tillgodoräkna något. Om du på sluttentamen löser en andra uppgift snabbt och vill sikta på högre betyg kan du lösa den tillgodoräknade uppgiften "igen", men den räknas endast mot högre betyg, **inte** som andrauppgift för betyg 3.

## Inloggning

Logga in på datorn i labbsalen med ditt liu-id och lösenord. Detta är samma inloggningsuppgifter som du använder i Lisam.

### Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö (Mate-session i Ubuntu). Efter en stund kommer även din kommunikationsklient att dyka upp automatiskt. Startmenyn är nedskalad till enbart det som examinator bedömt relevant. Andra program kan fortfarande finnas tillgängliga genom att starta dem från en terminal. Observera att en del program som använder nätverkstjänster inte fungerar normalt, eftersom nätverket inte är åtkomligt.

*När du är inloggad är det viktigt att du har tentaklienten igång hela tiden. Om den inte dykt upp fem minuter efter inloggning och inte heller när du startar den manuellt från menyn (fisken) tar du kontakt med assistent eller vakt i sal.*

### Terminalkommandon

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

### C++ referenssidor

På tentan har du tillgång till referenssidorna på `http://www.cppreference.com/` via webbläsaren Chrome. Starta `chromium-browser` i terminalen eller välj lämpligt alternativ från startmenyn. Observera att allt utom referenssidorna är avstängt. Om du inte kan komma åt en sida du tycker hör till referenssidorna (som kanske blockerats av misstag) kan du skicka ett meddelande via tentaklienten. Tag hjälp av assistent i sal om det inte fungerar.

### Givna filer

Eventuella givna filer finns i katalogen `given_files`. Denna underkatalog är skrivskyddad, så det är ingen risk du råkar ändra på dessa filer. Skrivskyddet gör dock att du måste kopiera in de givna filer du vill använda till tentakontots hemkatalog. Hur du listar och kopierar filer ska du kunna. Hemkatalogen står du i från början, och du kommer alltid tillbaka till den genom att bara exekvera kommandot `cd` i terminalen.

### Avslutning

När dina uppgiftsbetyg och ditt slutbetyg i kommunikationsklienten stämmer överens med det du förväntar och du är nöjd, eller när tentamenstiden är slut, är det dags att logga ut. Hinner du inte se ditt betyg får du höra av dig till examinator via epost efter tentamen.

Avsluta alla öppna program och logga ut ur datorn. Lämna inte datorn innan du ser att du är utloggad.

# Uppgift 1 - Speciella kort


Kapten Picard vill ha en app till spelet Gloomhaven. Appen ska hantera speciella spelkort och behöver vara ruskigt effektiv, därför vill han implementera delar av den i c++. Appen kommer hantera kort och i Gloomhaven består varje kort av tre delar: en övre effekt, en undre effekt och ett initiativ.

I `uppgift1.cc` finns ett givet huvudprogram, du ska skapa de klasser som krävs för att detta huvudprogram ska fungera. Du ska skapa 3 klasser enligt följande beskrivning.

`Effect` är ett aggregat (en `class` eller `struct` med enbart publika datamedlemmar) med 2 datamedlemmar. Datamedlemmarna är `name` och `description`. Det är lämpligt att båda dessa datamedlemmar är av typern `std::string`.

`Card` är en klass som har 3 datamedlemmar. Datamedlemmarna `top` och `bottom` är av typen `Effect`. Datamedlemmen `initiativ` är ett heltal. `Card` behöver också implementera operatorer för jämförelse och för att skrivas till utström. Det räcker att implementera operatorn 'mindre än' för jämförelse. Ett kort med lägre initiativ är mindre än ett kort med högre initiativ.

`Hand` är en klass med en datamedlem som är en behållare som kan innehålla ett godtyckligt antal `Card`. `Hand` har två medlemsfunktioner. `draw` som lägger till ett kort i databehållaren och `print` som skriver ut alla kort i databehållaren till en godtycklig `std::ostream`. Medlemsfunktionen för utskrift ska använda ostream operatorn för `Card`

**Körexempel:**

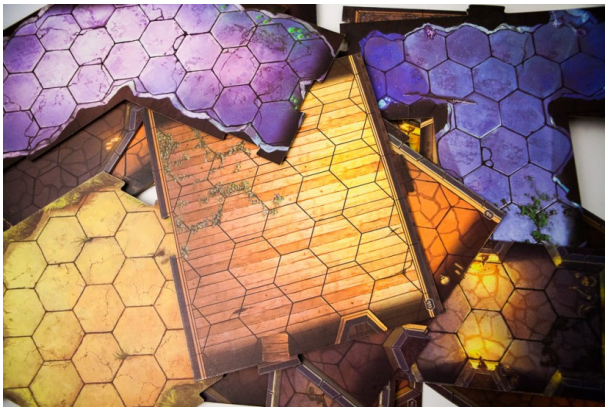
```
$ ./a.out
Card1:
Top: Aid from the Ether
Initiative: 91
Bottom: Summon Mystic Ally

Top: Aid from the Ether
Initiative: 91
Bottom: Summon Mystic Ally

Top: Ice Lance
Initiative: 25
Bottom: Ride the Wind

card1 is less than card2: false
card2 is less than card1: true
```

## Uppgift 2 - Delar av en spelplan



Kapten Cisco har gett dig jobbet att utveckla ett system för att hantera en del av spelplanen i spelet Gloomhaven. I Gloomhaven byggs en spelplan upp genom att mindre delar kopplas ihop och en sådan del kalla för en `Tile`. En `Tile` består av flera hexagoner. Varje hexagon kan ha ett godtyckligt antal saker ståendes på sig. Vi representerar i denna uppgift en hexagon som en `std::string` där varje `char` är en sak. Den saken som står högst upp på hexagonen är den sista bokstaven i strängen.

Du ska skapa klassen `Tile`.

Klassen `Tile` är en bit av spelplanen och har datamedlemmarna `name` som är en sträng och `hexes` som är en lämplig databehållare. `hexes` ska lagra godtyckligt många hexagoner. Varje hexagon skall lagras associerad med en x- och y-koordinat. `std::map` kan vara en lämplig start. Förutom att klassen ska fungera med den givna koden finns följande krav:

- En `Tile` ska skapas med enbart ett namn.
- Man skall sedan kunna lägga till nya hexagoner med en funktion som tar en x/y-koordinat och en `std::string` med funktionen `create_hex`.
- Man kan lägga till saker på en hexagon med en funktion (`push`) som tar en koordinat och en `char`. Den saken hamnar då högst upp på hexagonen.
- En hexagon skall kunna skrivas ut med en funktion (`print_hexagon`) som tar en koordinat.
- En hel `Tile` skall kunna skrivas ut med en funktion (`print_tile`).
- Vid utskrift av en hexagon så skriv endast den översta saken ut.

**Körexempel:**

```
Hexagon(0, 0): [X]
b1:
Hexagon(0, 0): [O]
Hexagon(0, 1): [H]
```

Tips: Det kan vara bra att fundera över hur du vill representera dina koordinater. En bra början kan vara std::pair.

## Uppgift 3 - Motståndare med gemensam basklass

I Gloomhaven finns det flera olika typer av monster. Kapten Georgiou håller på att skapa en digital version av Gloomhaven och funderar över olika designtyper som kan vara lämpliga. Du argumenterar för att en objektorienterad design skulle lösa detta problem bäst men Georgiou är inte övertygad. Du lovar därför att lämna in en prototyp som visar på styrkorna av objektorientering och polymorfi. Du ska skapa tre klasser som fungerar enligt det givna huvudprogrammet och resultatet ser ut som körexemplet. Du kommer behöva göra några ändringar i den givna koden för att lösa problem med slicing och eventuella minnesläckor. Klasserna du ska skapa är `Adversary`, `BanditGuard` och `BanditArcher`.

`Adversary` lagrar en datamedlem i form av ett heltal som representerar en fiendes hälsa. Klassen har medlemsfunktionerna `description` och `to_string`. `description` har inget grundbeteende som ska implementeras. `to_string` ska returnera en sträng som visar fiendens hälsa enligt formatet 'hp: x'.

`BanditGuard` utökar beteendet av `Adversary`. Klassen har förutom en datamedlem för hälsa också en data medlem för rörlighet. Objekt av denna typ implementerar nu medlemsfunktionen `description` som returnerar strängen 'Bandit guard'. `to_string` returnerar banditens hälsa och rörlighet.

`BanditArcher` utökar beteendet av `BanditGuard`. Ytterligare en datamedlem som representerar hur långt skytten kan skjuta finns nu med. `description` returnerar nu istället 'Bandit archer' och `to_string` returnerar även hur långt skytten kan skjuta.

**Körexempel:**

```
Adversaries:
=============
Bandit archer:
hp: 1 movement: 3 range: 2

Bandit guard:
hp: 2 movement: 4

Bandit guard:
hp: 3 movement: 3
```

## Uppgift 4 - Räkna ord

I Gloomhaven finns det ett litet minispel som kan ändra spelets riktning med extra skatter eller fler faror. Minispelet går ut på att räkna antalet förekomster av ord under de intensiva diskussionerna, vissa ord kan nämligen ge mer skatter medans andra ger fler faror som hjältarna möter. Kapten Janeway vill därför veta vilka ord som har sagts och hur många gånger de har uppkommit under diskussionerna och ber därför om din hjälp. Janeway har skrivit ner allt som har sagts och kan mata in dem i ditt program via terminalen (`std::cin`). Så din uppgift är att ta emot det som Janeway har skrivit in via terminalen och sedan visa en tabell med alla ord i inmattningen och hur många gånger de förekom (exempel hittar du nedan). Det finns dock en liten hake, Janeway gillar inte `std::map` (vilket egentligen hade varit perfekt för denna uppgift) så du får lösa uppgiften med andra algoritmer och behållare.

KRAV:
Du måste använda lämpliga algoritmer för att lösa problemet.

KRAV:
Varken `std::map` eller `std::for_each` får användas.

KRAV:
Det får inte förekomma några loopar, använd lämpliga algoritmer och behållare.

**Körexempel:**

```
$ ./a.out
I found loot over here
I too found loot

I: 2
found: 2
here: 1
loot: 2
over: 1
too: 1
```

## Uppgift 5 - Att bygga ihop en spelplan

I den givna koden `uppgift5.cc` finns delar av en länkad datastruktur implementerad. Denna länkade datastruktur representerar en karta i spelet Gloomhaven. En karta består av många `Tile`s som man pusslar ihop innan spelet börjar. På detta sätt kan man bygga många olika kartor. Kapten Kirk har börjat implementera klassen `GMap` för att representera kartor men får det inte riktigt att fungera på rätt sätt. Det är ditt jobb att skriva färdigt klassen.

`GMap` innehåller en pekare till den första kartbiten och innehåller sedan de medlemsfunktioner som krävs för att lägga till nya bitar så att varje ny bit blir den första. Det går också att ta bort den första kartbiten, efter borttagning blir den näst första kartbiten den första.

`Tile` innehåller en sträng med namnet som kartbiten har. Den innehåller också en pekare till nästa kartbit.

Följande problem finns med programmet som du behöver lösa:

- Klassen tar inte ansvar för att återlämna de resurser som den äger.
- Klassen implementerar inte och programmet testar inte flytt i den givna koden. Implementera korrekt flytt och skriv kod som testar att det fungerar som förväntat. Tips: `std::move` är en funktion du kan använda för att testa flytt.
- Klassen implementerar inte kopiering på önskat sätt. En karta skall inte gå att kopiera.

Det finns i denna uppgift inget körexempel som visar det önskade beteendet eftersom det är ditt jobb att skriva om huvudprogrammet så att det testar klassen på ett till fredställande sätt.