TDP004 – Tentamen

2024-10-31
08:00–13:00

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) Ett A4-ark med egna anteckningar

Examination

Tentamen består av två delar, Del I och Del II. Båda bedöms under tentamens gång och återkoppling ges. Detta ger möjligheten att korregera uppgiften och skicka in igen.

Alla uppgifter i Del I ges betyget *Godkänd*, *Kompletteras* eller *Underkänd*. Betyget *Godkänd* sätts på en uppgift när uppgiften är komplett och uppfyller alla specifierade krav. Betyget *Kompletteras* sätts om det finns vissa problem som behöver åtgärdas. Då kan uppgiften skickas in igen. Betyget *Underkänd* ges om inga framsteg gjort sedan förra inlämningen. Då finns det ingen möjlighet att skicka in uppgiften igen.

Uppgifterna i Del II bedöms med ett antal poäng. Varje uppgift är värt 10 poäng. Uppgiften kan skickas in igen så länge *signifikanta* förbättringar görs. Om inga signifikanta förbättringar görs ges en sista chans att skicka in uppgiften, sedan låses uppgiften till den uppnådda poängen.

Betygskriterier

För ett godkänt betyg måste alla uppgifter i Del I vara godkända *och* 10 poäng insamlas i Del II.

För ett högre betyg krävs godkänt på alla uppgifter i Del I samt ett antal poäng enligt tabellen nedan.

Insamlade poäng	**Betyg**
10p	3
17p	4
24p	5

Bonuspoäng

Bonus gäller endast under den första ordinarie tentamen i samband med kursen.

För varje av kursens 6 labborationer som lämnades in i tid och blev godkänd med max en komplettering ges 0.5 bonuspoäng i Del II. Detta ger maximalt 3 bonuspoäng.

Tillgodoräknanden

Tillgodoräknanden gäller endast under den första ordinarie tentamen i samband med kursen.

Om du på duggan klarade av Del I kan du tillgodoräkna denna på tentamen. Då kan du gå direkt till Del II.

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.

Del I

I denna del ska du lösa tre uppgifter. Din lösning på dessa uppgifter måste uppfylla alla specifierade krav, följa god programmeringsed och vara korrekt C++-kod.

Uppgift #1 – STL

```
struct Date
{
  int month;
  int day;
};

std::vector<Date> dates {
  Date{1, 3}, Date{5, 18}, Date{4, 13}, Date{1, 7}, Date{4, 27}
};
```

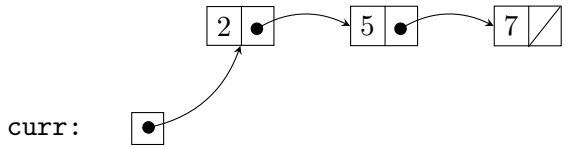
Skriv kod som sorterar vektorn `dates` så att månaden ökar och dagarna inom en månad minskar. I första hand ska vektorn sorterad på månad i stigande ordning. Om månaden är samma ska de i andra hand sorteras på dagen i fallande ordning.

Väntat resultat:

```
Date{1, 7}, Date{1, 3}, Date{4, 27}, Date{4, 13}, Date{5, 18}
```

Icke-funktionella krav: Du måste använda en eller flera lämpliga STL-algoritmer tillsammans med lämpliga lambda-uttryck så att din lösning går att applicera på en godtycklig vektor fylld med dates, inte bara den som är given i uppgiften.

Uppgift #2 – Minne



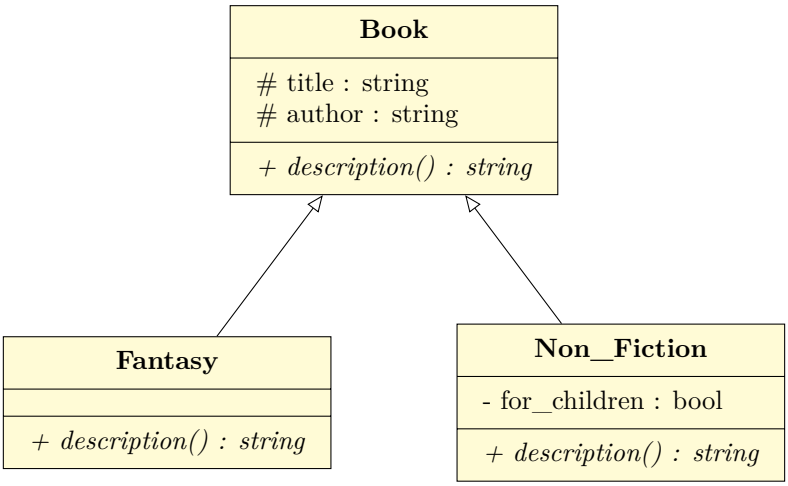
Skriv kod som skapar följande struktur. Modifiera sedan strukturen för att tar bort elementet med värde 5 ur listan. Ditt huvudprogram måste innehålla en variabel `curr` som pekare på första elementet i den länkade strukturen. När du är klar ska `curr` fortfarande peka på elementet med värde 2. Antag noder med strukturen

```
struct Node
{
  int val;
  Node* next;
}
```

Icke-funktionella krav: Du får inte byta plats på värden, endast flytta pekare. Samtliga element ska vara dynamiskt allokerade och koden får inte generera minnesläckor.

Uppgift #3 – Polymorfi

Implementera följande klasshierarki.



Där `description()` har följande beteende för respektive klass:

Book returnerar bokens titel följt av dess författerare, separerade med ett komma.

Fantasy returnerar bokens titel följ av ordet “by” följt av dess författare.

Non_Fiction göra samma sak som för `Book` fast lägger till `" for children"` i slutet om variabeln `for_children` har värdet `true`.

Icke-funktionella krav:

- Alla dina klasser ska vara deklarerande och definierade i *en* källkodsfil.
- Den givna filen `uppgift3.cc` har ett main-program som ska fungera oförändrat.

Funktionella krav:

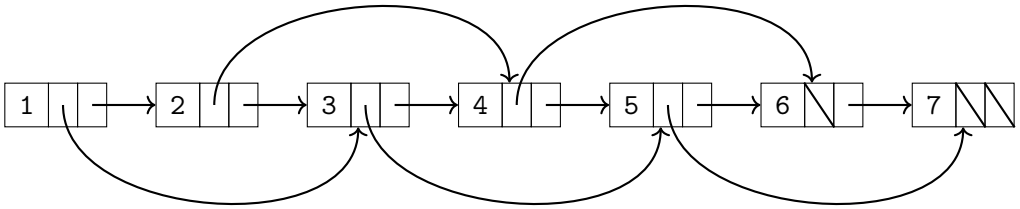
```
1984, George Orwell
Lord of the Rings by J.R.R. Tolkien
C++ Primer, Stanley Lippman
Grumpy Monkey, Suzanne Lang for children
```

Part II

Varje av de tre uppgifterna i Del II är värda 10 poäng vardera. Det innbär att totalpoäng för Del II är 30 poäng.

Uppgift #4 – Minne

I `given_files/uppgift6.cc` finns en del av en implementation av en något optimerad länkad lista, kallad `Jump_List`. Skillnaden mellan en vanlig länkad lista och vår `Jump_List` är att varje nod har två pekare istället för en. Pekaren `next` pekare på nästa element. Det finn också en pekare `jump` som pekare på elementet två steg framför. Detta låter oss stega genom listan mycket snabbare eftersom vi kan hoppa över varannat element (se bilden).



Din uppgift är att implementera följande medlemsfunktioner i `Jump_List`:

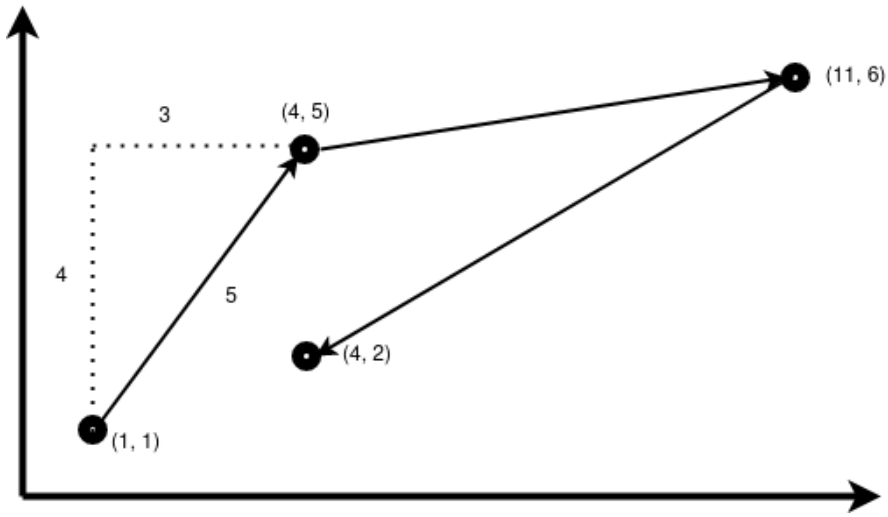
- Flyttkonstruktor
- Flyttilldelningsoperator
- Destruktor
- Medlemsfunktionen `push_front()` som tar ett värde och lägger till det i början av listan. Notera att pekaren `jump` måste uppdateras *om* det elementet finns.

Du ska också garantera att det inte går att kopiera objekt av typen `Jump_List` genom att förbjuda användningen av kopieringskonstruktorn och kopieringstilldelningsoperatorn.

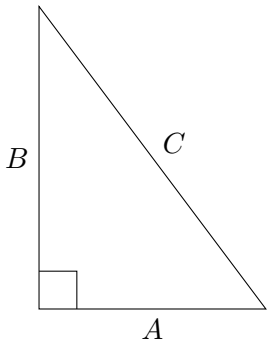
Icke-funktionella krav:

- Programmet måste köra utan minnesläckor.
- Du måste utöka det givna testprogrammet så det testar de speciella medlemsfunktionerna du implementerat.

Assignment #5 – STL



Denna uppgift kretsar kring den givna filen `WAYPOINTS.txt`. I filen finns fyra *waypoints* som representeras av en x- och ykoordinat separerade med ett `-` Din uppgift är att skapa ett program som med hjälp av *lämpliga* STL-algoritmer läser innehåller i filen och skriver ut det totala distansen om man skulle besöka varje *waypoint* i ordingen de står i filen. Vi antar att man färdas raka vägen mellan två *waypoints*. Diagrammet ovan visar de vägar som finns i `WAYPOINTS.txt`.



För att mäta avståndet mellan två punkter används Pythagoras sats. Detta betyder att distansen mellan de två första punkterna i diagrammet ovan ges av $\sqrt{(4-1)^2 + (5-1)^2}$ där $4-1$ är differansen mellan x-koordinaterna och $5-1$ är differansen mellan y-koordinaterna. I det generella fallet ges hypotenusan av

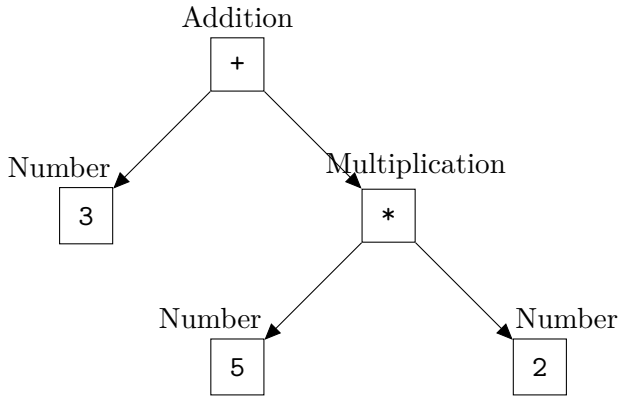
$$C = \sqrt{A^2 + B^2}$$

Ditt jobb är att beräkna avståndet mellan varje på varandra följande par av punkter och summera dessa för att få den totala distansen (det borde vara ≈ 20.133). Utdatan från programmet ska endast vara den totala distansen.

Du får **inte** använda några loopar eller rekursion. Syftet är att använda lämpliga algoritmer från standardbiblioteket för att lösa problemet.

Tips: Funktionen `std::sqrt` i `<cmath>` beräknar kvadratroten av ett tal.

Uppgift #6 – Polymorfi



Ett sätt att representera aritmetiska uttryck kan vara som ett träd av noder. Alla noder som inte är lövnoder(noderna längst ner i trädet) kan lagrar referenser till andra noder och på det sättet kan man sätta ihop trädet (se bilden, observera att vi alltså inte nödvändigtvis använder pekare för att lösa detta problem). I bilden har vi en additionsnod, en multiplikationsnod och 3 lövnoder som innehåller heltal. Det här trädet representerar det aritmetiska uttrycket:

$$3 + 5 \cdot 2$$

Och man skulle kunna utvärdera uttrycken genom att be additionsnoden (anropa en medlemsfunktion) utvärdera sig själv. Additionsnoden ber i sin tur alla sina barn (i det här fallet värdenoden med siffran 3 och multiplikationsnoden) att utvärdera sig. Sedan skickar additionsnoden tillbaka summan av värdena som barnen returnerar. Värdenoden returnerar heltalet 3, multiplikationsnoden i sin tur ber båda sina barn att utvärdera sig och skickar sedan tillbaka produkten av de utvärderingarna. Resultatet blir att multiplikationsnoden returnerar talet 10 och additionsnoden kan nu returnera summan av 3 och 10 vilket är 13.

I den givna filen `uppgift3.cc` finns ett givet huvudprogram. Ditt jobb är att implementera den klasshierarki som krävs för att detta program ska fungera. Du ska implementera 5 klasser i denna uppgift (de är små).

- Klassen `Node` är en basklass som representerar en godtycklig nod. Den har inga datamedlemmar och är abstrakt. Har en medlemsfunktion `int eval()` som saknar implementation.
- Klassen `Number` som härleds från `Node`. Klassen lagrar en datamedlem av typen `int` och implementerar `eval`, när eval anropas på instanser av denna typ ska datamedlemmen som lagras returneras.
- Klassen `Binary` är en abstrakt klass som härleds från Node och har två datamedlemmar av typen `Node&`
- Klassen `Addition` härleds från klassen `Binary` och överlagrar `eval`. Denna medlemsfunktion anropar medlemsfunktionen `eval` på båda datamedlemmarna och returnerar summan av returvärdena (datamedlemmarna som kommer till i och med arvet från `Binary`)
- Klassen `Multiplication` härleds från klassen `Binary` och överlagrar `eval`. Denna medlemsfunktion anropar medlemsfunktionen `eval` på båda datamedlemmarna och returnerar produkten av returvärdena.