TDP004 - Tenta

2023-01-13

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

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 - Komplexa tal - klasser och operatorer

Komplexa tal kan vara bra att ha, de består av en reell och en imaginär del. I den givna filen `uppgift1.cc` finns ett huvudprogram som beror på att klassen Complex finns. Ditt jobb är att implementera denna klass. Klassen har 2 datamedlemmar av typen `double` som representerar den reella och imaginära delen av talet.

Utöver detta ska klassen ha en konstruktor och följande operatoröverlagringar:

- Utströmsoperatorn, se körexemplet.
- Den binära operatorn `+` som adderar 2 komplexa tal och skapar ett nytt tal som är summan av de två parametrarna. Vid addition ska det nya talets reella del vara summan av båda talens reella delar och den imaginära delen ska vara summan av båda talens imaginära delar

$$C_r = A_r + B_r$$
$$C_i = A_i + B_i$$

- Den binära operatorn `*` som multiplicerar 2 komplexa tal och skapar ett nytt tal som är produkten av de två parametrarna. Vid multiplikation blir det lite krångligare. För den reella delen tar man produkten av båda reella delarna och subtraherar produkten av båda imaginära delarna. För den imaginära delen tar man produkten av det första talets reella del och den andra talets imaginära del, från det adderar man sedan produkten av det första talets imaginära del och det andra talet reella del.

$$C_r = A_r \cdot B_r - A_i \cdot B_i$$
$$C_i = A_r \cdot B_i + A_i \cdot B_r$$

Körexempel:

```
a = 1+2i
b = 3+4i
a + b = 4+6i
a * b = -5+10i
```

Uppgift 2 - Makros - Behållare och funktioner

Inom programmering är ett `makro` något som liknar en variabel, men istället för att hålla koll på ett värde så håller den koll på en bit text. När exempelvis C++-kompilatorn stöter på ett makro så byter den ut makrot mot makrots definition. Tänk exempelvis att du skrive `Linköpings Universitet` ofta, då skulle man kunna skapa ett makro som heter `LiU` och definera det som `Linköpings Universitet`. Om jag exempelvis då skrev `Jag pluggar på LiU` så skulle det *expandera* till `Jag pluggar på Linköpings Universitet`.

Ett problem med makron är att deras definition kan hänvisa till andra makron, tänk att vi exemplvis har följande makron definerade:

```
IDA = Institutionen för datorvetenskap
LiU = Linköping Universitet
OGANIZATION = IDA / LiU
```

Då skulle texten `Kursen TDP004 ges vid ORGANIZATION` expandera till `Kursen TDP004 ges vid Institutionen för datorvetenskap / Linköpings Universitet`. Alltså måste vi expandera makron inuti andra makron, vilket kan ske i ett godtyckligt antal steg.

Din uppgift är att:

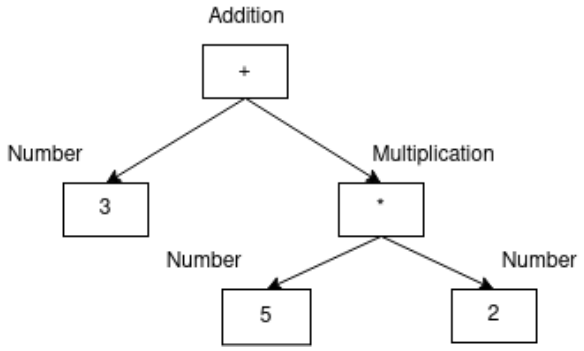
1. Skapa funktionen `define_macros` som tar en `ifstream&` och som läser in den givna filen `MACROS`. I filen finns ett makro per rad, först kommer namnet på makrot och sedan definitionen, dessa skiljs med tecknet `:`. Lagra dessa makron i en lämplig behållare och returnera behållaren.
   Tips: `getline` kan plocka ut namnet från makrot (leta efter `:`). Sedan kan du plocka ut definitionen med `getline` (leta efter \n)

2. Skapa funktionen `expand` som tar en sträng med en rad text och behållaren som returneras från `define_macros`. Denna funktion ska expandera alla makron i texten och returnera den expanderade texten. För att implementera denna funktion ska du gå igenom raden ord för ord. Kontrollera om ordet är ett makro genom att leta efter ordet i din behållare. Om ordet är ett makro så anropar du `expand` med **definitionen av makrot** som har det namnet och sparar resultatet i strängen som kommer returneras. Om ordet inte är ett makro så kan du bara spara det i strängen som kommer returneras (lägg till ett mellanslag också).
   Tips: Använd `istringstream` och operatorn `>>` för att gå igenom raden text ord för ord.
   Tips: Det är viktigt att vi anropar `expand` med definitionen av makrot i denna funktion istället för att göra det i `define_macros` eftersom vi inte kan känna till alla makron innan `define_macros` har körts färdigt.

Körexempel (användarinmatning i fetstil):

`Pontus works at IDA and teaches FULL_COURSE`
`Pontus works at Department of Computer and Information Science and teaches TDDE18 at Department of Computer and Information Science / Linköping University`
`ctrl+D`

Uppgift 3 - Matte med klasser - klasser och 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:

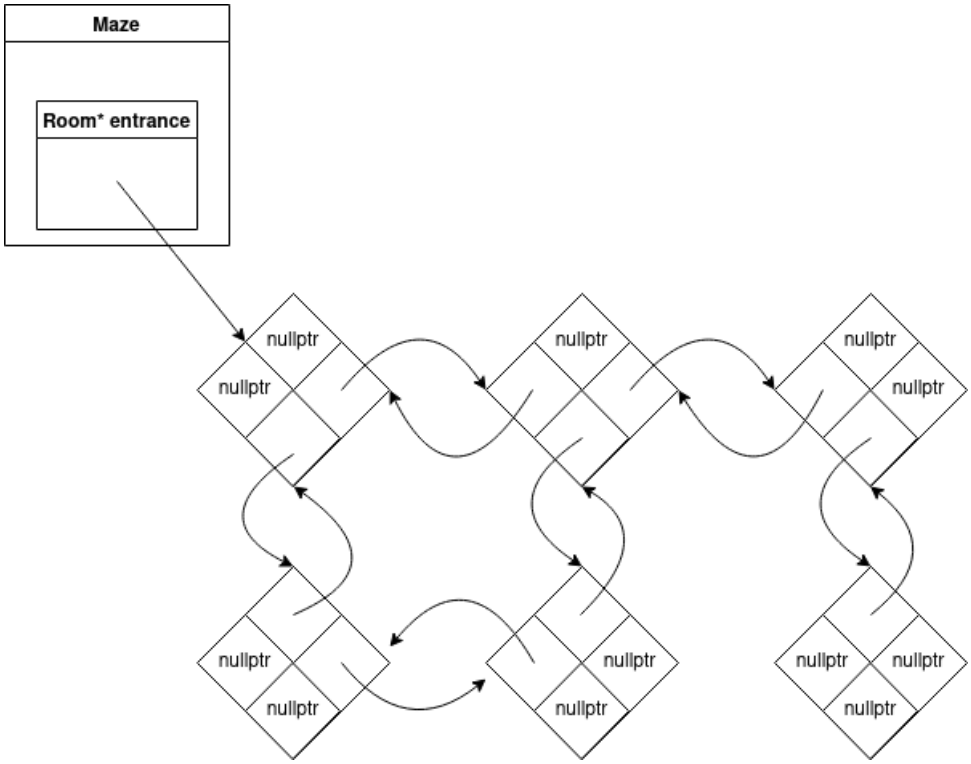
$$3 + 5 \cdot 2$$

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.

Uppgift 4 - Labyrint - Pekarsemantik

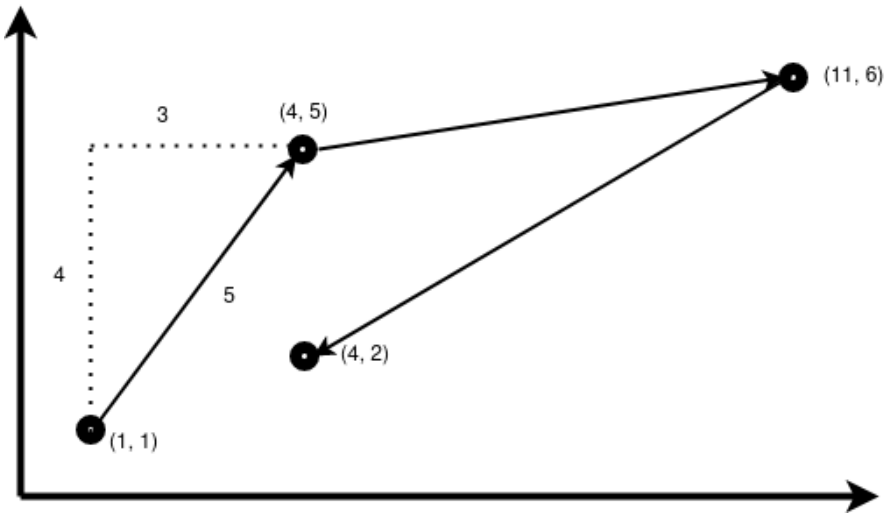


I den givna filen `uppgift4.cc` finns en påbörjad implementation av en länkad labyrint (`Maze`). Klassen har en datamedlem `entrance` av typen `Room*` som leder till det första rummet. Varje rum leder i sin tur till upp till 4 rum. Problemet är att klassen har ansvar för allt minne som dessa rum tar upp och saknar en destruktor, programmet läcker därför minne. Du ska i denna uppgift implementera destruktorn för klassen `Maze`.

Tänk på att labyrinten (du kan se labyrinten som finns i exempelprorammet i bilden) har cykler, det finns alltså flera sätt man kan komma till samma rum. Man behöver därför hålla koll på att man inte försöker ta bort ett rum fler än en gång och samtidigt ser till att man faktiskt tar bort alla rummen.

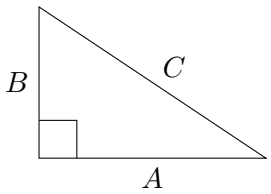
Testa om din lösning fungerar med `valgrind --leak-check=full ./a.out`

Uppgift 5 - Mäta avstånd - STL algoritmer



Denna uppgift utgår från den givna filen `WAYPOINTS`. I filen finns fyra vägpunkter (waypoints) som representeras med en x-koordinat och en y-koordinat som separeras med `-`. Din uppgift är att skapa ett program som med lämpliga algoritmer läser in en fil av denna typ och skriver ut det totala avståndet mellan dem om man skulle gå från punkt1 till punkt2 till punkt3 och så vidare. Bilden representerar punkterna givna i filen `WAYPOINTS`.

För att mäta avståndet mellan två punkter använder vi helt enkelt Pythagoras sats. I bilden är alltså den första sträckan lika med hyptenusan för en triangel där katetrarna är skilnaden i x- och y-koordinatens värden. Skillnaden mellan x-koordinaterna är alltså 3 och skillnaden mellan y-koordinaterna är 4. För att få ut hypotenusan tar vi alltså summan av 4 i kvadrat och 3 i kvadrat. Roten ur den summan är alltså hypotenusan, eller i det här fallet avståndet mellan punkterna. Se Pythagoras sats nedan:



$$C^2 = A^2 + B^2$$

Summan av alla avstånden för det givna exemplet ska bli ungefär 20.133.

TIPS: Du kanske behöver funktionalitet för att göra upphöjt till eller kvadratroten i denna uppgift. Dess finns i `<cmath>` och används som `std::pow(2, 3)` och `std::abs(-5)` för att göra 2 upphöjt till 3 och att plocka ut absolutbeloppen av minus 5.
KRAV: Denna uppgift ska lösas så långt som möjligt med **lämpliga** algoritmer från standard-biblioteket. I praktiken innebär detta att du inte får använda `for_each`, for-loopar, while-loopar, rekursion, etc.