Information relevant för både VAKTER och STUDENTER:

DEL 1: Teoridel. Löses först *utan* inloggning på dator.

Tillåtna hjälpmedel: Penna, tomma papper, egen hjärnkapacitet.
Som student rekommenderas du spendera *max* ca 40 minter på del 1. Du får använda mer tid.

DEL 2: Praktisk del. Engångslösenord till dator delas ut då DEL 1 lämnas in.

Tillåtna hjälpmedel: Penna, tomma papper, och en bok: *C++ Primer* alt. *C++ Direkt*.
Som student rekommenderas du spara *minst* 3h till denna del.

Information relevant endast för STUDENTER:

Betygsättning och Bonus

Tentamens båda delar ger vardera 30 poäng. Totalt kan alltså 60 poäng erhållas. För betyg 3 eller högre gäller att praktiska delens obligatoriska uppgift måste vara Godkänd och minst 30 poäng. Sedan räknas betyget fram genom heltalsdivisionen `betyg=totalpoäng/10`.

Under kursen har en antal bonuspoäng (procent räknad från tentamens maxpoäng) erhållits. Dessa adderas på din totala poäng då den obligatoriska praktiska uppgiften är *Godkänd*. Har du full (10%) bonus får du alltså 6 extra poäng när du klarat den praktiska delens obligatoriska uppgift. Eventuell avrundning sker nedåt.

Kommunikationsklient

Under tentamens praktiska del kommer en klient att finnas tillgänglig. Alla **skall** starta sin klient (högerklicka på bakgrunden/skrivbordet och välj från menyn). Klienten används för att:

1. Begära förtydliganden av uppgifter eller hjälp/råd då systemet krånglar.
2. Få tips, korrigeringar eller annan viktig information från Examinator.
3. Skicka in den obligatoriska uppgiften för rättning (enbart obligatoriska uppgiften).

Om mer akut hjälp behövs (systemet krånglar eller låser sig), räck up handen!

Poängsättning och krav på praktiska delens lösningar

För *Godkänd uppgift* krävs full funktionalitet helt enligt specifikationen (uppgiften). Dessutom skall koden uppfylla de krav på stil och algoritm som kan ställas enligt god programmeringssed. Som exempel på detta gäller: Konsekvent indentering, inga globala icka-konstanta variabler, ingen fullständig uppräkning, och korrekt minneshantering.

Den obligatoriska uppgiften betygsätts på skalan *Godkänd*, *Kompetteras* eller *Underkänd* direkt under tentamen. Vid betyg *Kompletteras* ges möjlighet att lämna in en ny, korrigerad, version av lösningen. *Underkänd* sätts då inga väsentliga framsteg skett mellan inlämningarna.

All poängsättning sker **efter** tentamens slut. För den praktiska delens **tre** uppgifter gäller att:

Partiell funktionalitet (uppfyller ej kraven för Godkänd uppgift)	0-1p
Uppgiften uppfyller kraven för Godkänd uppgift	5p
Mycket god programmeringsstil	+1p
God användning av STL	+1p
God och relevant ansvarsfördelning i funktioner och/eller klasser	+3p

Tabellen skall tolkas så att endast **en** av de två första raderna kan ge poäng, medan rader med ett plus kan ge poäng oberoende av varandra. Examinator kan, om examinator bedömer det motiverat, frångå ovan bedömningsmall. Totalt kan alltså maximalt 10 poäng erhållas på en uppgift. En uppgift som ej uppfyller kraven kan *teoretiskt* ändå ge maximalt 6 poäng.

Teoretisk del. Skriv tydligt.

Tidplan

Börja med att läsa alla uppgifter, både teoretiska och praktiska, och gör en tidplan.

Strömmar och in-/utmatning

```
int age;
string name;
while ( getline( (infile >> age >> ws), name) )
{
   outfile << setw(30) << name << age << endl;
}
```

Du har just stött på ovan kod. `infile` och `outfile` är vanliga filströmmar. Du vet att resultatet av inre parentesen (som utförs först) är `infile`, som skickas vidare till `getline`.

1. Vad drar du för slutsats om kodens funktion? Vad åstadkommer koden? (2p)
2. Förklara vid vilka två situationer loopen kommer att avbryta. (2p)
3. Finns möjligheter att utskriften blir “oformaterad” eller “rörig” på något sätt? (1p)

Pekare och referenser

```
void confuse_reader(int*& a, int& b, int* c)
{
  *c = 3;
  c = &b;
  a = c;
  *c = 2;
  *a = 1;
}
int main()
{
  int *x, y, z;
  confuse_reader(x, y, &z);
  cout << *x+y+z << endl;
}
```

4. Precis innan funktionen `confuse_reader` returnerar, vart hänvisar `a`, `b` och `c`? (3p)
5. Vilken summa skrivs ut av huvudprogrammet? (2p)

Minneshantering

```
List_Node* one = new List_Node;
List_Node* two = one->next;
one->value = 6;
delete one;
two->next = new List_Node;
delete one->next;
```

I ovanstående kod har klassen `List_Node` två publika datamedlemmar (8 byte) och saknar både konstruktor och destruktor. Koden har korrekt syntax och kompilerar utan problem.

6. Ange vilket (eller inget) minne som används felaktigt i kodsekvensen. (1p)
7. Hur mycket (eller inget) minne är allokerat utan att blivit avallokerat efter koden? (2p)
8. Hur mycket (eller inget) minne är avallokerat utan att blivit allokerat efter koden? (2p)

Kodstil med mera

Vilket viktigt krav skall enligt god konvention ställas på en kommentar? (1p)

Vilket viktigt krav skall enligt god konvention ställas på ett variabel- eller funktionsnamn? (1p)

Vilka två huvudsakliga mål har kodformatering, t.ex. indentering? (2p)

Standarsbiblioteket i C++ (STL)

Du vill räkna hur många gånger orden “if”, “while”, “for”, “switch”, “class”, “struct”, “enum” och “typedef” förekommer i en stor fil.

9. Ange vilken av `std::list`, `std::vector` och `std::map` du skulle använda. (0p)
10. Motivera varför ditt val ovan är korrekt. (3p)

Klasser

```
vector<Animal> v;
Cat    p(“Pelle Svanslös”);
Dragon t(“Temeraire”);
Lion   a(“Alex”);
v.push_back(p);
v.push_back(t);
v.push_back(a);
list.at(2)->speak();
```

Det finns en implementation av funktionen `speak` i vardera av de fyra klasserna `Animal`, `Cat`, `Lion` och `Dragon`. Koden innehåller tyvärr lite felaktigheter som gör att den inte fungerar.

11. Vad krävs av klassdefinitionerna och i form av korrigeringar av koden ovan för att anropet på sista raden skall ske till medlemsfunktionen `speak` i klassen `Lion`? (3p)
12. Inuti funktionen `speak` finns nyckelordet `this` att tillgå. Vilken av variablerna ovan kommer nyckelordet `this` att referera till? Tänk på att första index är 0. Du får anta att koden korrigerats enligt kraven du angav i föregående uppgift. (2p)
13. Inkapsling, Aggregation, Konstruktor, Djup kopiering, Destruktor, Abstraktion, Kohesion är viktiga begrepp. Välj *ett* av dem och beskriv det för en nybörjare. (3p)

Uppgift 1, Praktisk del. Obligatorisk. (frequency_table.cc)

Inom programmering är det mycket vanligt att analysera och visualisera statistiska data.

I denna uppgift skall du räkna förekomsten av varje siffra 0-9 i en fil. Resultatet skall sedan visualiseras i ett stapeldiagram enligt nedan exempel. Om någon siffra inte förekommer alls så skall den stapeln hoppas över.

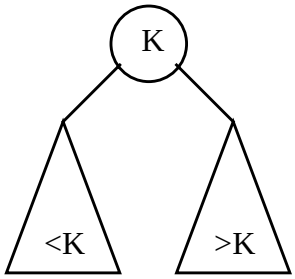
Namnet på filen anges på kommandoraden, precis efter programmets namn. Programmet skall således inte fråga användaren efter någon data. Om filnamnet saknas på kommandoraden, eller om filen inte kunde öppnas så skall programmet skriva ut ett felmeddelande på standard felutmatningsström och sedan avsluta utan att rita ut stapeldiagrammet.

Du skall anta att indatafilen kan vara mycket stor, många gigabytes. Ditt program skall fungera utan problem oavsett filstorlek.

```
zaza1 <1> bargraph given_files/example_1.1
-------------
0             | 13
-------------
---------------------------------------------------------------
1                                                             | 62
---------------------------------------------------------------
-----------------
2                | 16
-----------------
---------
3        | 9
---------
-------------
4            | 12
-------------
------------------
6                 | 17
------------------
-----------
7          | 11
-----------
----------
8         | 10
----------
----------
9         | 10
----------
```

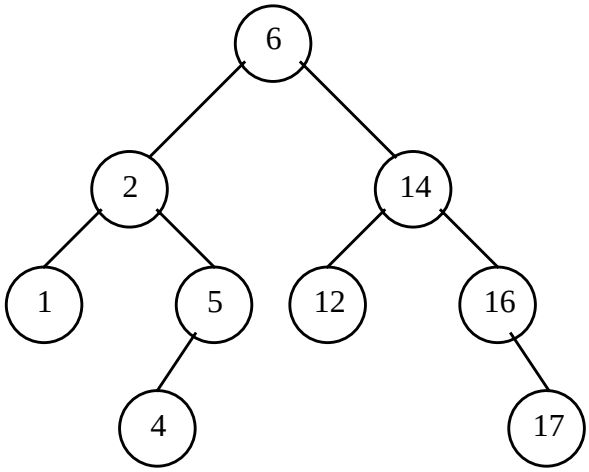
Uppgift 2, Praktisk del. (`Tree.cc, Tree.h, tree_main.cc`)

Binära sökträd används ofta på grund av den effektivitet de erbjuder när det gäller att söka upp ett värde i trädet.



Ett binärt sökträd byggs upp genom att varje nod associeras med ett värde och två barnnoder. Varje barnnod utgör roten av ett subträd. Ett viktigt krav är att alla noder i en nods vänstra delträd har värden som är mindre än nodens värde, och att alla noder i en nods högra delträd har större värden. I figuren till höger visas noden K med dess två subträd. Nya noder sätts alltid in som "löv" (längst ned) i trädet.

Roten i ett träd är farförälder till alla noder i trädet. I figuren nedan är noden med värdet 6 rotnod. Rodnoden där har två subträd som startar i noderna med värde 2 respektive 14.



Filerna `given_files/Tree.h`, `given_files/Tree.cc` och `given_files/tree_main.cc` innehåller implementationen av ett binärt sökträd och ett litet testprogram. Kopiera filerna till din arbetsmapp och slutför implementationen av medlemsfunktionen `remove_root`. Helt enligt namnet på funktionen så skall den ta bort rotnoden ur trädet. Oavsett hur trädet ser ut naturligtvis. När funktionen anropas upprepat blir trädet så småningom tomt.

Krav 1: Du får endast använda pekaroperationer när du tar bort roten. Du får alltså inte "fuska" genom att skapa ett nytt träd utan noden, eller genom att kopiera data från nod till nod.

Krav 2: Det är mycket viktigt att all minneshantering sker korrekt.

Krav 3: Du har varken behov eller tillåtelse att ändra i `tree_main.cc`.

Tips 1: Börja med penna och papper. Ta bort roten upprepat tills trädet är tomt (rita alla träd). För att skriva koden måste du förstå hur trädet förändras.

Tips 2: Du kommer upptäcka att du efter borttagning plötsligt sitter med ett delträd "över", som du inte vet vad du skall göra av. Det ena subträdet till den borttagna noden blir ny rot. Överväg noga var det andra delträdet skulle hamna om det sattes in i den nya roten. En nods vänstra barnnod kommer alltid att vara den minsta noden av alla noder i det högra delträdet. Betänk hur insättning av 10, 9, 8 och 7, i den ordningen, ska gå till. Då bör du få en bra känsla för var den minsta noden i ett delträd (delträdet med start i 14) kommer att hamna.

Uppgift 3, Praktisk del. (written_text.cc)

C++ har mycket språkstöd för objektorienterad programmering. I denna uppgift skall du nyttja språkstödet så långt du kan för att designa en klasshirarki för skriven text. Det givna huvudprogrammet i `given_files/written_text.cc` får inte ändras i sin struktur eller funktion, det ger poängavdrag. Dock får det uppgraderas till användning av STL, och det är tillåtet att lägga till tester så länge inte befintlig funktion tas bort. Klasserna du skall skapa:

`Written_Text` är basklassen. En skriven text har en författare, en titel samt funktioner för att komma åt dessa (utan att kunna ändra dem). Det skall inte gå att skapa en skriven text direkt, endast subklasser kan göra det.

`Book` representerar en vanlig bok. Förutom författare och titel har en bok en bindning. Det skall bara gå att skapa en bok om alla tre anges. En bok har två undertyper, `Hardback` som alltid har sydd (“stitched”) bindning och `Paperback` som alltid har limmad (“glued”) bindning. Det skall gå att hämta ut bindningstypen från en bok.

`Textfile` representerar en text i elektronisk form. Den har utöver författare och titel ett filformat. Alla tre attribut måste anges för att skapa en klass av denna typ. Det skall gå att fråga en textfil efter dess format.

`Magazine` representerar slutligen en slags tidning. Författare och titel måste anges för att få skapa en instans.

För att kunna mata ut en skriven text skall en operator för detta finnas, så att huvudprogrammet fungerar. Utskriften skall nyttja polymorfi. Utöver vad som angivits ovan skall inga synliga funktioner finnas i någon av klasserna, och så många funktioner som möjligt skall fungera på konstanta objekt.

Exempel:

```
Temeraire is written by Novik with sewn binding.

Potter is written by Rowling with stitched binding.

Eragon is written by Paolini with glued binding.

NY Times Magazine is written by Times company as a Journal.

Midnight Sun is written by Meyer in PDF format.

Temeraire, Novik, sewn
Potter, Rowling, stitched
Eragon, Paolini, glued
NY Times Magazine, Times company
Midnight Sun, Meyer, PDF
```

TIPS: Det går inte att göra `operator<<` för utskrift polymorf. Däremot går det utmärkt att göra polymorfa medlemsfunktioner.