TDDI82 – Tentaregler

Hjälpmedel

Följande får tas med på tentan:

- En bok om c++. För boken gäller följande regler:
  - Kommentarer/noteringar som direkt rör text och exempel på sidan i fråga får finnas i sidmarginalen.
  - Egna sidflikar för att enkelt kunna hitta t.ex. de olika kapitlen är tillåtna.
  - Inga extra ark eller lappar, lösa eller fastsatta, får finnas.
  - Tomma sidor, insidan av pärmarna, försättsblad, etc., får inte innehålla programkod.
- Maximalt ett A4-ark med egna anteckningar (dubbelsidigt)
- Penna för att anteckna under tentan. Ni kommer förses med blanka papper

Följande får INTE tas med:

- Elektroniska hjälpmedel såsom miniräknare, mobiltelefon och smartklockor.

Utloggning

När du är nöjd med ditt betyg (som står i tentaklienten) kan du avsluta och logga ut som vanligt i menyn. Klicka sedan på ok följt av knappen avsluta tentan. Observera att du när du gjort detta inte kan logga in igen. Lämna inte din plats innan vanliga inloggningsskärmen syns.

Frågor om uppgifter

Frågor om tentan i stort eller uppgiftspecifika frågor ska ställas via tenta-klienten. Detta för att vi ska ha en historik av konversationen samt för att vi ska kunna ge samma hjälp till olika studenter.

Systemfrågor

Om du har systemfrågor som t.ex. problem med tentaklienten eller terminalen räcker du upp handen så kommer en assistent och hjälper till.

Tentaregler

Tentan består av fyra uppgifter indelade i två kategorier; standardbibliotek (STL) och mallar. För godkänt betyg på tentan krävs godkänd lösning av en uppgift inom vardera kategori. För högre betyg krävs lösning av uppgifter inom en viss tid enligt tabell 1.

Tid (h)	Antal uppgifter		Betyg
	STL	Mallar	
2.5 + B	1	1	5
4 + B	2	1	5
4 + B	1	2	5
4 + B	1	1	4
5	1	1	3

Tabell 1: Tidsgränser för olika slutbetyg, B är bonus från labserien (se nedan)

För att en uppgift ska anses godkänd krävs följande:

- att man noga följt alla instruktioner och krav ställda i uppgiften
- din kod följer god programmeringsstil (se labseriens rättningsguide)
- att din kod har bra inkapsling och resurshantering
- att standardbibliotekskomponenter används på ett bra sätt

Bonus från labserien

Bonus från labserien (benämnd B i tabell 1) ger ett visst antal minuter (15 eller 30) extra på respektive tidsgräns för högre betyg. Bonusen är endast giltig det år den erhölls.

C++ referens

Det finns tillgång till valda delar av `cppreference.com`. Du måste starta webbläsaren via ikonen ”Web access” på skrivbordet.

Alias för kompilering

Det finns tre alias att använda sig av för kompilering med c++17:
`g++17` Kompilering utan varningar.
`w++17` Rekommenderas!
`e++17` Kompilering med alla varningar som fel.

Uppgift 1 - Mallar

I denna uppgift ska du skapa en generell mallfunktion `get_pairs` som tar emot två iteratorer från en godtycklig databehållare och returnerar en `std::vector` med alla möjliga par av element i iterator intervallet.

Exempel: Givet följande databehållare:

```
std::vector<int> v { 1, 2, 3 };
```

så ska `get_pairs(v.begin(),v.end())` returnera denna `std::vector<std::pair<int,int>>`:

```
{ { 1, 2 }, { 1, 3 }, { 2, 3 } }
```

I uppgiften är vi endast intresserade av unika par av element, dock behöver inte värdena i sig vara unika. Det betyder att t.ex. dessa element: `{ 1, 0, 1 }` ska resultera i följande uppsättning par: `{ { 1, 0 }, { 1, 1 }, { 0, 1 } }`.

Notera: `get_pairs` ska alltid returnera en `std::vector` som innehåller `std::pair`, men det som paret innehåller beror på *värdetypen* (`value_type`) av de inskickade iteratorerna.

Tips: Notera att du inte kan göra `+` operatorn på en godtycklig iterator. Använd istället `std::next`.

Tips: Ta iteratorer som en godtycklig mallparameter.

Det finns ett givet testprogram i filen `pairs.cc`, detta ska fungera som tänkt (se körexempel) och du ska inte behöva modifiera det överhuvudtaget.

Körexempel

```
==== Testfall #1 ====
1 + 2 = 3
1 + 3 = 4
1 + 4 = 5
2 + 3 = 5
2 + 4 = 6
3 + 4 = 7
==== Testfall #2 ====
ERIC CHRISTOFFER
MALTE ERIC
WILHELM MALTE
```

Uppgift 2 - Mallar

Det är förvånansvärt vanligt att iterera över två stycken behållare samtidigt för att processera elementen parvis. I C++ skulle detta innebära att man behöver två iteratorer som inkrementeras samtidigt, vilket är omständigt. Istället vill vi baka ihop det beteendet till en egen iterator för att göra koden något simplare för andra programmerare. Låt oss kalla vår egna iterator för `zip_iterator`.

`zip_iterator` är en klassmall som tar två mallparametrar `It1` och `It2` som representerar iteratorerna till de två behållare vi vill iterera över.

Sättet `zip_iterator` fungerar på är att den lagrar en begin och en end iterator för vardera behållare. Låt oss kalla dessa lagrade iteratorer för `begin1`, `end1`, `begin2` och `end2`, där `begin1` och `end1` är av typen `It1` medan `begin2` och `end2` är av typen `It2`. Alla dessa ska initieras med en lämplig konstruktor.

`zip_iterator` måste implementera `operator*` (avreferering), `operator==`, `operator!=` samt båda `operator++`. Detaljer följer nedan:

- `operator*` ska returnera ett `std::pair` som innehåller värdet av `begin1` och värdet av `begin2`. **Tips:** Använd `std::make_pair`.
- Båda versionerna av `operator++` ska inkrementera `begin1` och `begin2`. Utöver det ska de även ha ”standard” beteende. Notera att du inte behöver göra några kontroller i denna operator.
- För att iteratorn ska fungera som väntat måste vi implementera `operator==` på följande sätt:

```
if (begin1 == end1 && other.begin1 == other.end1)
{
  return true;
}
else if (begin2 == end2 && other.begin2 == other.end2)
{
  return true;
}
return begin1 == other.begin1 && begin2 == other.begin2;
```

  Där `other` är den iterator vi tar emot som argument till operatorn. Anledningen till detta är att vi vill att end iteratorn för `zip_iterator` ska nås när *någon* av behållarna når slutet. `operator!=` ska implementeras genom att invertera resultatet från `operator==`.

I `zip.cc` finns det ett givet testprogram. Detta ska fungera med C++17 utan några modifieringar.

Uppgift 3 - STL

I denna uppgift ska du skriva ett program som givet en dokumentmall fyller i de olika fälten i mallen med något som användaren anger på kommandoraden.

Filen `mall.txt` visar hur formatet på en dokumentmall ser ut. En mall innehåller ett antal parametrar som har formen `#N` där `N` är ett index som startar från `0`. Tanken är att användaren anger filen som första kommandoradsargument följt av ett antal ord. Dessa ord ska sedan ersätta motsvarande parameter i mallen. Så om t.ex. användaren kör programmet såhär:

```
$ ./a.out mall.txt first second third
```

Så ersätts alla förekomster av `#0` i `mall.txt` med `first`, `#1` med `second` och `#2` med `third` o.s.v.

För att implementera detta program, följ dessa steg:

1. Läs in innehållet från filen till en vektor vid namn `text`.
2. Skapa en lämpligt databehållare `arguments` (läs resten av stegen för att avgöra vad som är lämpligt).
3. Gå igenom alla kvarvarande kommandoradsargument. Associera strängen `"#0"` med det första kvarvarande kommandorandsargumentet, associera strängen `"#1"` med nästa kommandoradsargument, och fortsätt på detta vis tills det inte finns några fler kommandoradsargument. Alla dessa associationer ska sparas i `arguments`.
4. Gå igenom alla ord i `text` och utför följande:
   (a) Om ordet börjar med #, kolla om det ordet finns som nyckel i `arguments`.
   (b) Om det finns som nyckel, ersätt ordet i `text` med värdet associerat med nyckeln.
   (c) Om det inte finns som nyckel, skriv ut ett felmeddelande som berättar att denna parameter inte har fått ett värde och avsluta sedan programmet.
   (d) Skriv ut ordet följt av ett mellanslag.
5. Skriv ut en radbrytning.

Krav: Du får maximalt göra en uppslagning i `arguments` per ord i `text`.

OBS: Du kan alltid göra antagandet att parameterar i dokumentmallen (d.v.s. ord på formen `#N`) *inte* har några extra tecken i början eller slutet.

OBS: Denna uppgift behandlar databehållare, så du *behöver* inte använda STL algoritmer. Dock kan det underlätta.

Körexempel (fetstilt är användarinmatning)

```
$ ./a.out mall.txt first second third fourth
In this text first is a parameter, and the same is true for
second meaning that first and second can be whatever we
want. Also, as a test we add fourth as well, thus skipping
the third argument (but as a test we include third here).
```

Uppgift 4 - STL

Rövarspråket är ett kodspråk som används bland barn. Språket följer en enkel regel; efter varje konsonant lägger man till ett `o` följt av samma konsonant igen (d.v.s. konsonanter såsom `f` ersätts med `fof`). Exempel: `kaffe` blir `kokafoffofe` och `oavsett` blir `oavovsosetottot`.
I denna uppgift ska du skapa ett program som:

1. Läser in ord från `cin` till en vektor tills filslut (`ctrl-D`)
2. Konverterar alla ord i vektorn till rövarspråket (lägg själva logiken för att konvertera ett ord till rövarspråk i en egen funktion)
3. Skriver ut alla ord på rövarspråket (separerade med ett mellanslag)

Du får inte använda några manuella loopar, utan du måste använda algoritmer från standardbiblioteket. Du får inte heller använda `std::for_each`. Du får inte heller använda fullständig uppräkning för att ta reda på om en bokstav är en vokal eller konsonant. Använd istället strängen `vowels`, som finns i den givna filen `robber.cc`, för att se om tecknet finns med där. För att se om tecknet är en konsonant kan använda `std::isalpha` för att ta reda på det är en bokstav och sedan kontrollera att tecknet *inte* förekommer i `vowels`.

Notera: Ditt program behöver **inte** hantera å, ä eller ö.

Körexempel (fetstilt är användarinmatning)

```
Mata in din text: vi ska koka kaffe imorgon
vovi soskoka kokokoka kokafoffofe imomororgogonon

Mata in din text: det ska vara kul att skriva c++
dodetot soskoka vovarora kokulol atottot soskokrorivova coc++
```