TDP004 - Datortenta (DAT2)

2012-12-19

Regler

- All kod som skickas in för rätting skall 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 tentans start.
- Vid toalettbesök eller rökpaus ska pauslista utanför salen fyllas i.
- All form av kontakt mellan studenter under tentans gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av tentavakt i samband med tentans start samt under tentans gång.
- Frågor om specifika uppgifter eller om tentan i stort ska ställas via tentasystemets kommunikationsklient.
- Systemfrågor kan ställas till assistent i sal genom att räcka upp handen.
- Ingen uppgift rättas efter tentatidens slut.
- Ingen uppgift kan kompletteras under tentans sista kvart.
- En praktisk uppgift kan kompletteras för högre poäng tills den är poängsatt med "Klar". "Klar" sätts vid bedömningen att ingen nämnvärd förbättring skett sedan tidigare inskickning.
- En teoretisk uppgift kan inte kompletteras. Undantagsvis kan examinator be om förtydligande av ett svar.
- Kompilerande kod är ett grundkrav för poängsättning. Kod som inte kompilerar skall beskrivas och kommenteras ut före inskickning.

Antal uppgifter	7
Totalt antal poäng	20
Hjälpmedel	En C++-bok (t.ex. C++ Primer 5th ed.) En A4-sida med egna anteckningar

Betygssättning

Tentan är uppdelad i två delar, en teoretisk och en praktisk. Delarna är värda 5 respektive 15 poäng och för godkänt betyg krävs minst 11 poäng totalt. Poäng som krävs för de olika betygen kan ses i tabell 1.

Poängsumma	**Betyg**
0 – 10	U
11 – 14	3
15 – 17	4
18 – 20	5

Tabell 1: Poängfördelning för betygssättning

Information

Inloggning

Logga in på tentakontot med följande användaruppgifter:

```
Användarnamn: examx
Lösenord:     kluring1
```

Följ menyvalen så långt det går tills du ska mata in ett engångslösenord. Tag fram ditt LiU-kort och visa det för tentavakten för att få detta lösenord.

När du är inloggad är det viktigt att du startar tentaklienten genom att högerklicka på bakgrunden och välja “tentaklient” i menyn.

Avslutning

Tryck på knappen märkt “exit” i menyn längst nere på skärmen och välj ok. Vänta ett tag och tryck sedan på knappen “Avsluta tentamen” när det är möjligt. När detta är gjort är det omöjligt att logga in igen. Lämna inte datorn förrän du ser den vanliga inloggningsskärmen.

Teoretisk del

Dessa uppgifter kan *INTE* kompletteras. Skriv ditt svar på en fil med namnet *teoriN.txt* (där *N* är uppgiftsnumret) och skicka in filen för rättning. En fil och en inskickning för varje uppgift.

1. Nedan asciifigur visar en pekare p och en struct/class med två medlemmar som representerar en koordinat. Ange ett C++-uttryck som refererar till minnesutrymmet markerat med en asterix (*). [1 p]

```
  +------+   +---------+
p|   ------>| +------+|
  +------+   |x| 2.54 ||
             | +------+|
             | +------+|
             |y|* 8.9 ||
             | +------+|
             +---------+
```

Svar:

(*p).y eller p->y

2. Nedan anges fyra koncept och fyra meningar. Koppla ihop rätt koncept (a,b,c,d) med rätt mening (1,2,3,4) [2 p]

```
a) Koppling (coupling)
b) Samhörighet (cohesion)
c) Inkapsling (encapsulation)
d) Abstraktion (abstraction)
```

```
1) Hur mycket detaljer olika klasser känner till om varandra
2) Hur relevanta medlemmar är för en klass
3) När metoder och data som hör ihop samlas och paketeras tillsammans
4) Ett namn anger vad något åstadkommer utan att vi behöver veta hur
```

Svar:

1a, 2b, 3c och 4d. 1 rätt ger 0 poäng. 2 rätt ger 1 poäng. 3 rätt går inte att få. 4 rätt ger 2 poäng.

3. Kompileringsprocessen går igenom tre steg. Först sker preprocessing, sedan kompilering och sist länkning. Från vilket steg kommer vardera av följande tre felmeddelanden? [1 p]

(a)

```
<1> g++11 errors.cc
Undefined                       first referenced
 symbol                             in file
kung_fu_panda(int)                  /var/tmp//cceKGWw3.o
ld: fatal: symbol referencing errors. No output written to a.out
collect2: error: ld returned 1 exit status
```

(b)
```
<2> g++11 errors.cc
errors.cc: In function 'int main()':
errors.cc:7:24: error: too few arguments to function
                       'int kung_fu_panda(int)'
errors.cc:3:5: note: declared here
```

(c)
```
<3> g++11 errors.cc
errors.cc:1:28: fatal error: my_little_pony.h: No such file or
directory compilation terminated.
```

Svar:

(a) länkningsfel
(b) kompileringsfel
(c) preprocessorfel All rätt krävs för poäng.

4. Du skriver medlemsfunktioner till en klass. I flera av dessa upptäcker du att du behöver initiera och sedan använda en temporär variabel för att hålla reda på hur långt funktionen hunnit i sin beräkning. Ska du deklarara den temporära variabeln som en klassmedlem eller som en lokal variabel i varje funktion? Motivera svaret kort. [1 p]

Svar:

En lokal variabel ska användas.

Några motiveringar:

- Variabler skall inte ha längre livstid än strikt nödvändigt. En klassvariabel skulle finnas kvar även när den varken används eller behövs.
- Flera funktioner skall inte återanvända samma variabel för olika syften. Det finns då risk att en funktion via variabeln påverkar en annan funktion på fel sätt, i detta fall om en funktion t.ex. glömmer initiera variabeln innan den används.
- Det står ingenstans att temporärvariablerna är av samma typ. Är det olika typ behövs naturligtvis olika variabler.

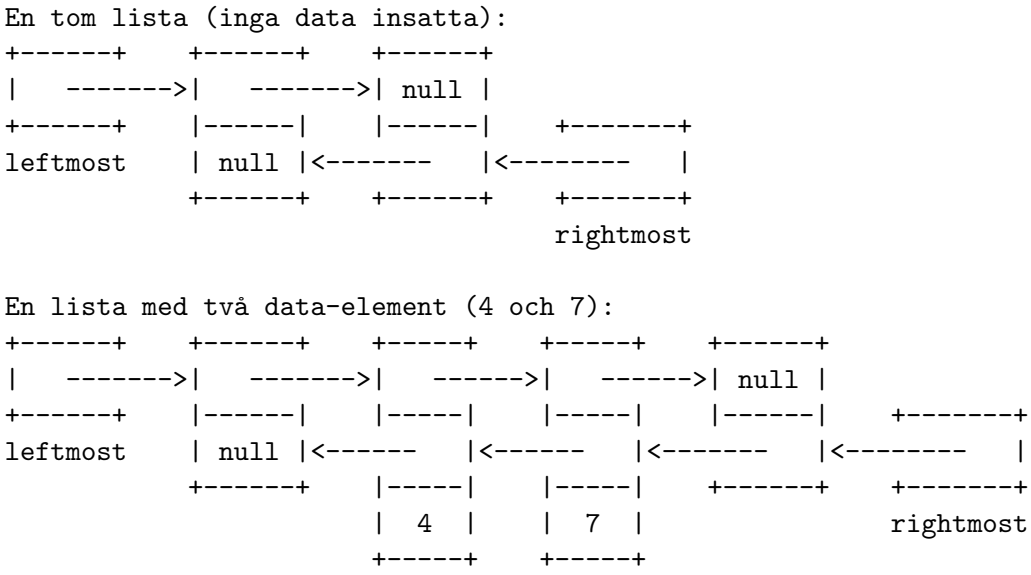
Praktisk del

En uppgift kan kompletteras tills den är poängsatt som "Klar". Poängen inom hakparenteser till höger anger vad maxpoängen för hela uppgiften är värd. Poängen inom vanliga parenteser till vänster anger vad motsvarande speciella krav är värt. Du skall alltid lösa så mycket av uppgiften du bedömer att du klarar *innan* du skickar in.

Kompilering mm

Använd aliaset *g++11* för att kompilera med C++11-standarden.
Använd aliaset *w++11* för att även få bra varningar.
Använd aliaset *g++11filter* för att även filtrera med gccfilter.
Använd aliaset *w++11filter* för att få varningar och gccfilter.
Använd *bcheck ./a.out* för att leta minnesläckor.

5. I *given_files/dlist.cc* och *given_files/dlist.h* finns en implementation av en dubbellänkad lista. Listan börjar och slutar med ett "dummy-element" som inte lagrar någon data, utan endast markerar start och slut. Dessa finns för att slippa specialfall vid hantering (t.ex. insättning/borttagning) av de dataelement som sitter först och sist. Varje element i listan har en right-pekare till nästa element och en left-pekare till föregående. Figuren visar dels en tom lista och dels en lista med två dataelement; fyra och sju. [3 p]



Speciellt för listan är att dataelementen är mycket ineffektiva att kopiera, men listan behöver ändå sorteras. En algoritm för att sortera listan finns implementerad. Det enda som saknas är den del som byter plats på två intilliggande element som är i fel ordning (swap). Som bekant innebär sortering att data flyttas runt (kopieras) väldigt mycket under sorteringen så här gäller det att tänka till så bara pekare flyttas runt och ingen data.

Mål: Implementera det som behövs för att byta plats på två element i listan utan att kopiera data.

Tips: Var mycket tydlig och metodisk och gör en sak i taget. Rita varje steg på papper så du ser vad som händer.

Svar:

Sex pekare behöver uppdateras i rätt lösning för att genomföra swappen. Efter detta måste "current"-pekaren sättas att peka på det första av de två swappade elementen.

Om problemet löses genom att kopiera data till en ny nod och därefter sätta in den på rätt ställe ges poäng förutsatt att det är korrekt gjort utan minnesläckor.

```
if (current->getData() > current->right->getData())
{
  Node* one = current->left;
  Node* two = current->right; // number two in new order
  Node* three = current;     // number three in new order
  Node* four = current->right->right;

  one->right = two;
  two->right = three;
  three->right = four;

  four->left = three;
  three->left = two;
  two->left = one;

  current = two;
}
```

6. På två filer finns noll eller flera heltal sorterade i stigande ordning. Du kan enkelt skapa filer med testdata i emacs eller direkt i terminalen: [5 p]

```
<1> echo 1 3 5 7 9 21 25 > odd.txt
<2> echo 0 2 4 6 8 > even0.txt
<3> echo 2 4 6 8 > even2.txt
<4> echo > empty.txt
```

Mål: Skriv ett program som skriver ut talen från båda filerna i en sekvens sorterad i stigande ordning. Filnamnen anges på kommandoraden för högsta poäng, annars via vanlig inmatning. Om någon av de två indatafilerna saknas eller inte går att öppna ska programmet skriva ut ett felmeddelande och avslutas. Det skall inte spela någon roll vilken fil som anges först.

(3p) Fullt fungerande och korrekt program.

(1p) De båda filerna kan anges på kommandoraden utan att fel kan uppstå.

(1p) Programmet klarar att hantera filer som vardera är mycket större än den mängd internminne som finns tillgängligt för programmet.

Svar:

- Hårdkodade filnamn ger 2 poängs avdrag.
- Kontroll av kommandoradsargument
- Filhantering, öppning, kontroll, (stängning)

- Lagring - endast ett tal per fil behöver hanteras åt gången.
- Filerna kan ha (har troligen) olika längd...
- Vad händer med det sista talet som lästes in från den “fungerande” filen?

Se *merge.cc*.

7. Professor Sierpinski sitter på en konferens. Föredraget är inte så intressant så istället för att lyssna sitter han förstrött och ritar i sitt anteckningsblock. Han har ritat en triangel (a) och kommer fram till att om han ritar samma triangel både under och till höger om den ursprungliga triangeln så får han en ny triangel (b) med ett triangelformat hål. Om han kopierar den nya dubbelt så stora triangeln på samma sätt blir det en fyra gånger större triangel (c). Om processen upprepas ännu fler gånger börjar ett intressant mönster att framträda. Sierpinsi’s triangel är född. Figuren nedan visar de tre trianglarna (a), (b) och (c) omnämnda ovan. [7 p]

```
(a)
///
//
/
```

```
(b)
//////
// //
/  /
///
//
/
```

```
(c)
////////////
// // // //
/  /  /  /
///   ///
//    //
/     /
//////
// //
/  /
///
//
/
```

Professor Sierpinski involverar nu kollegan i stolen bredvid som tacksamt accepterar denna intressanta distraktion från det tråkiga föredraget. Tillsammans utarbetar de en metod för att rita en hyfsad *appromixation* av triangeln. De hinner lagom klart tills nästa föredrag (som är det de egentligen kommit för) startar. Deras metod ser ut som följer:

1. Välj en storlek på en liksidig triangel.
2. Välj hur många itereringar som skall utföras.
3. Det rätvinkliga hörnet högst upp till vänster är koordinat (0,0).
4. Definiera övriga två hörn utifrån storleken.
5. Slumpa fram en koordinat `Current=(x,y)` inom den kvadrat som bildas av de tre hörnen (det fjärde hörnet får du fram från de tre hörn du har).
6. Välj slumpvis ett av hörnen i triangeln.
7. Flytta `Current` halva sträckan (fågelvägen) till det slumvis valda hörnet.
8. Lägg till koordinaten `Current` till den samling koordinater som hör till triangeln.
9. Upprepa från punkt 6 så många gånger som valdes i punkt 2.
10. Gå avslutningsvis igenom alla koordinater rad för rad i från (0,0) till nedre högra hörnet i kvadraten från punkt 5. Rita ut ett '/' om punkten finns med i samlingen från punkt 8, eller rita annars ett blanksteg ' '. *Var medveten om att metoden är en approximation. Ett fåtal '/' kommer att hamna fel eller saknas.*

Mål: Implementera Sierpinski's metod att rita trianglarna. Det krävs ingen felkontroll av punkt 1 eller 2, men det är lämpligt att välja storlek mindre än 80 och minst 10000 iterationer. Om du bara löser uppgiften delvis måste du ändå lämna in ett fungerande program som testar de delar du gjort. Använd lämpliga funktioner för att abstrahera delproblem.

OBS! Den funktionalitet du har vid första inlämning är det du får poäng för. I resterande inlämningar kan du bara korrigera felaktigheter, inte lägga till funktionalitet.

(2p) Du har implementerat en korrekt klass för att representera en koordinat med korrekt användning av private och konstruktor.

(1p) Din koordinat-klass implementer jämförelseoperatorn "mindre än" och jämförelseoperatorn "lika med" så två koordinater kan jämföras direkt. För "mindre än" jämförs y-koordinaten i första hand och x-koordinataen i andra hand. (Du får ersätta detta krav med lambdafunktion om du vill.)

(1p) Du löser punkt 6 smidigt utan att använda if-satser eller switch-sats, alternativt att du använder abstraktion för att dölja sådana.

(1p) Du löser punkt 7 med en medlemsfunktion som flyttar koordinaten halvvägs till en koordinat som anges som parameter.

(2p) Du använder slumptal och STL-containers för att implementera Sierpinski's metod till fullo utan fel.

Svar:

Uppgiften täcker in grundläggande klass, enkel operatoröverlagring, slumptal, grundlägande STL och lite problemlösning. Fullt fungerande program som löst och testat (på något sätt) delar av uppgiften ger delpoäng enligt ovan.

Figuren blir inte perfekt med denna metod. En del steck hamnar fel.

Se *sierpinski_vector.cc*.

Tips: En rudimentär påminnelse till några vanliga saker.

Slumptal :

C-stil Inkludera `<cstdlib>` och `<ctime>`. `srand(time(nullptr))` initierar slumpgeneratorn. Se "man -s3c rand" i övrigt.

C++-stil Inkludera `<random>`. Ett objekt av typen `random_device` kan skapas och sedan anropas som funktion för att slumpa fram ett tal.

Operatorer :

Hur en klass A deklarerar några vanliga konstruktorer och operatorer:

```
class A
{
  public:
    A();
    A(A const&);
    ~A();
    A& operator=(A const&);

    bool operator==(A const&) const;
    bool operator<(A const&) const;

    A operator+(A const&) const;
    A operator*(A const&) const;

    friend ostream& operator<<(ostream& os, A const&);
    friend istream& operator>>(istream& os, A&);
};
```

STL-containers :

vector<T> En kontinuerlig sekvens. Osorterad. Insättning sist med `.push_back(T värde)`. Åtkomst med indexering eller iterering.

list<T> En länkad lista. Osorterad. Insättning först med `.push_front(T värde)`. Insättning sist med `.push_back(T värde)`. Åtkomst med iterering.

set<T> En uppsättning (endast nycklar). Sorterad. Insättning med `.insert(T nyckel)`. Effektiv åtkomst med `.find(T nyckel)`.

map<T1, T2> En uppsättning nycklar med tillhörande värden. Sorterad. Insättning med `.insert(pair<T1,T2> par)` eller indexering. Åtkomst med indexering eller iterering. Effektiv åtkomst utan insättning med `.find(T1 nyckel)`.

STL-algoritmer :

sort(begin, end)

find(begin, end, värde) returnerar iterator

for_each(begin, end, lambda)

unique(begin, end) returnerar iterator

copy(sourcebegin, sourceend, destinationbegin)