KTH Information and
Communication Technology

ID1218
Johan Montelius

Tillämpad Programmering (ID1218)
2012-12-10 09:00-12:00

Förnamn: _____

Efternamn: _____

Personnummer: _____

Regler

Du får inte ha något materiel med dig förutom skrivmateriel. Mobiler etc, skall lämnas till tentamensvakten.

Instruktioner

- Tentamen har totalt 36 poäng och skall skrivas på 3 timmar.
- Läs igenom hela tentamen innan du börjar.
- Svaren skall lämnas på dessa sidor, använd det utrymme som finns under varje uppgift för att skriva ner ditt svar. Om du inte får plats med sitt svar så kan du använda ytterligare sidor. Dessa sidor skall då vara märkta med namn och personnumer. Det skall även tydligt framgå i svaret under uppgiften att ytterligare sidor har använts.
- Du skall lämna in hela denna tentamen.
- Svar skall skrivas på svenska.

Betyg

< 16	F
≥ 16	E
≥ 20	D
≥ 24	C
≥ 28	B
≥ 32	A

Erhållna poäng

Skriv inte här, detta är för rättningen.

Uppgift	1	2	3	4	5	6	Σ
Max	6	6	6	6	6	6	36
Poäng							

Totalt antal poäng:

Betyg:

Namn:_____Personnummer:_____

1 Funktionell Programmering [totalt 6 poäng]

1.1 Ett träd [2 poäng]

Skriv en funktion *update(Key, Tree)*, som tar en nyckel och ett ordnat binärt träd, och returnerar ett nytt träd där värdet för nyckeln *Key* har ökats med *1*. Trädet skall fortfarande vara ordnat. Om nyckeln inte finns skall en ny nod läggas till i trädet där värdet för nyckeln är *1*.
Vi kan representera ett träd med antingen atomen *nil* (det tomma trädet) eller en tupel *{node, Key, Value, Left, Right}*. Antag att vi kan ordna nycklar med den vanliga $<$ *operatorn*.

Svar:

```
update(Key, nil) ->
    {node, Key, 1, nil, nil};
update(Key, {node, Key, V, L, R})  ->
    {node, Key, V+1, L, R};
update(Key, {node, K, V, L, R}) when Key < K ->
    {node, K, V, update(Key, L), R};
update(Key, {node, K, V, L, R}) when Key > K ->
    {node, K, V, L, update(Key, R)}.
```

Namn:_____Personnummer:_____

1.2 RLE [2 poäng]

Skriv en funktion *rle(List)* i Erlang som kompakterar en lista av atomer. Kompakteringen skall ske genom så kallad "run length encoding" där en obruten sekvens av identiska atomer representeras som en tuple *{I, N}* där *I* är atomen och *N* antalet förekomster i sekvensen. Den kompakterade versionen av listan

```
[b,a,c,c,c,a,a,b,c]
```

representeras som följer

```
[{b,1}, {a, 1}, {c,3}, {a,2}, {b,1}, {c,1}]
```

Svar:

```
rle([]) -> [];
rle([H|T]) -> rle(T, H, 1).

rle([], I, N) -> [{I,N}];
rle([H|T], H, N) -> rle(T, H, N+1);
rle([H|T], I, N) -> [{I,N}|rle(T, H, 1)].
```

Namn:_____Personnummer:_____

1.3 Två lika stora delar [2 poäng]

Skriv en funktion *split(List)* som returnera en tuple *{A, B}* där *A* och *B* tillsammans innehåller alla element ur den ursprungliga listan och är så lika långa som möjligt (lika om listan har ett jämnt antal element, och en diff på *1* om den har ett ojämnt antal).
Den ursprungliga listan skall bara gås igenom en gång. Du kan inte använda exempelvis *length/1* för att ta reda på längden av listan.

Svar:

```
split(L) -> split(L, [], []).

split([], A, B) -> {A, B};
split([H|T], A, B) -> split(T, [H|B], A).
```

Namn:_____Personnummer:_____

2 Högre-ordningens funktioner [6 poäng]

2.1 map, filter och fold [2 poäng]

Skriv en funktion som tar en lista med heltal och producerar: *summan av de kvadrater som är jämna* (lista [2,3,4] resulterar i 4+16 dvs 20). Du skall använda dig av följande högre ordningens funktioner: filter/2, map/2, foldl/3 och/eller foldr/3, se appendix för hur de anropas. Operatorn *rem* ger resten vid heltalsdivision (*X rem 2 == 0* evalueras till *true* om X är ett jämnt tal).

Svar:

```
sumesqr(List) ->
     Squares= map(fun(X) -> X*X end, List),
     Even = filter(fun(X) -> X rem 2 == 0 end, Squares),
     foldl(fun(X,A) -> X+A end, 0, Even).
```

Namn:_____Personnummer:_____

2.2 uppdatera ett träd [2 poäng]

Skriv en funktion *update(Fun, Tree)*, som tar en funktion och ett träd och returnerar ett nytt träd som har samma struktur som det givna trädet men där varje nods värde är resultatet som fås när funktionen appliceras på det ursprungliga värdet.
Antag att trädet representeras med *nil* för det tomma trädet och *{node, Key, Value, Left, Right}* för en nod med en nyckel och ett värde.

Svar:

```
update(Fun, nil) ->
      nil;
update(Fun, {node, Key, Value, Left, Right}) ->
      {node, Key, Fun(Value), update(Fun, Left), update(Fun, Right)}.
```

Namn:_____Personnummer:_____

2.3 foldr av träd [2 poäng]

Givet representationen av träd i föregående uppgift, skriv en högre ordningens funktion *traverse(Fun, Acc, Tree)* som tar en funktion, ett initialt värde och ett träd och returnerar det värde som är resultatet av att applicera funktionen på elementen i trädet och den ackumulerade parametern. Ordningen skall vara sådan att man börjar med trädets mest högra element och sedan arbetar sig mot vänster.
Följande anrop

```
Tree = {node, g, 3, {node, a, 2, nil, nil},
                {node, m, 4, nil, nil}},
traverse(fun(X,A) -> [X|A] end, [], Tree).
```

skall ge resultatet

```
Res = [2,3,4]
```

Resultatet blir i det är fallet en lista med "inorder traversering". Noter dock att vi har applicerat funktionen i omvänd ordning i.e. *Fun(2, Fun(3, Fun(4, [])))*.

Svar:

```
traverse(Fun, Acc, nil) ->
      Acc;
traverse(Fun, Acc, {node, Key, Value, Left, Right}) ->
      traverse(Fun, Fun(Value, traverse(Fun, Acc, Right)), Left).
```

Namn:_____Personnummer:_____

3 Concurrency [totalt 6 poäng]

3.1 liten process [2 poäng]

Skriv en funktion i Erlang som implementerar en process. Processen skall ha ett tillstånd som är en räknare och kunna ta emot följande meddelandet:

- **{add, N}** : addera N till tillståndet.
- **{sub, N}** : subtrahera N från tillståndet.

Svar:

```
open(State) ->
 receive
    {add, N} ->
        open(State + N);
    {sub, N} ->
        open(State - N)
 end.
```

3.2 lite större [2 poäng]

Utöka processen i uppgiften ovan så att den kan hantera följande meddelanden:

- **{read, Ref, From}**: där *From* är en process-id från en anropande process, svara med {state, Ref, State}.

Svar:

```
open(State) ->
 receive
     :
   {read, Ref, From} ->
       From ! {state, Ref, State},
       open(State)
 end.
```

Namn:_____Personnummer:_____

3.3 låsning [2 poäng]

Pricessen skall kunna befinna sig i två olika tillstånd *open* som du implemeterat hittills och *locked* som du nu skall implementera. Detta gör vi genom att lägga till följande meddelande till *open*:

- **{lock, Ref, From}**: där *From* är en process-id från en anropande process, svara med {locked, Ref, State} och gå över i det låsta tillståndet.

I det låsta tillståndet skall vi kunna hantera följande meddelande:

- **{add, Ref, N}** : addera N till tillståndet.
- **{sub, Ref, N}** : subtrahera N från tillståndet.
- **{release, Ref}**: gå över till det öppna tillståndet.

Notera att vi inte skall acceptera vilket meddelanden som helst. Vi skall inte låta lura oss utan endast ta emot meddelanden som har exakt samma referens *Ref* som användes när vi låste processen - alla andra meddelanden skall vara kvar i meddelandekön.

Svar:

```
open(State) ->
  receive
      :
    {lock, Ref, From} ->
        From ! {locked, Ref. State},
        locked(State, Ref)
  end.


locked(State, Ref) ->
  received
    {add, Ref, N} ->
         locked(State+N, Ref);
    {sub, Ref, N} ->
         locked(State-N, Ref);
    {release, Ref} ->
         open(State)
  end.
```

Namn:_____Personnummer:_____

4 Värden, pekare och arrayer i C++[totalt 6 poäng]

4.1 legala uttryck [2 poäng]

Vilka uttryck är legala uttryck i C++(som följer standarden C++03) ? Svara enbart ja eller nej, alla rätt ger 2 poäng, ett avdrag för varje fel.

1. `int &x;`

   Svar: nej: vi kan inte skapa en referens utan att säga vad den skall referera till

2. `int y = 3; int &x = y;`

   Svar: ja: vi definerar en referens x och låter den referera till y

3. `int y = 7; int *x = &y;`

   Svar: ja: inget problem att ta adressen av x och det är en int

4. `int x=32; int** y = &(&x) - 5;`

   Svar: nej: vi kan inte ta adressen av en adress (inte ett l-värde)

5. `int y = 7; int *x = y;`

   Svar: nej: y är av typen int, men x har typen int*

4.2 värdet på x: [2 poäng]

Givet nedanstående uttryck, vad är värdet på variabeln x?

```
int a[5] = {3,2,1,5,4};
int x = a[2] + a[3];
```

Svar: Variabeln x får värdet 6 eftersom arrayer är noll-indexerade.

4.3 värdet på y: [2 poäng]

Givet nedanstående uttryck, vad är värdet på variabeln y?

```
int a[5] = {3,2,1,5,4};
int y = *(&a[3] - 2);
```

Svar: Variabeln y får värdet 2 eftersom (&a[3] - 2) är &a[1] och *(&a[1]) är a[1] dvs 2.

Namn:_____Personnummer:_____

5 Klasser och objekt [totalt 6 poäng]

I följande frågor antar vi att vi inkluderar biblioteket *iostream* och använder dess namespace *std*. Vi har också definierat en klass *IntCell* enligt nedan.

5.1 inte riktigt som vi tänkt oss [2 poäng]

Givet är följande definitiom av *IntCell* och en procedur `swap(IntCell &x, IntCell &y)` som låter `x` och `y` byta värden.

```
class IntCell
{
  public:
     IntCell(int initialValue = 0 )
       { storedValue = new int(initialValue); }
     ~IntCell()
       { delete storedValue}
     int getValue() const
       { return *storedValue; }
     void setValue(int val)
       { *storedValue = val;}
  private:
       int *storedValue;
};

void swap(IntCell &x, IntCell &y) {
  IntCell tmp = IntCell();

  tmp = x;
  x = y;
  y = tmp;
}
```

Vi har även ett litet testprogram som använder sig av vår *swap-procedur*.

```
int test() {
   IntCell x(1);
   IntCell y(2);

   swap(x, y);

   cout << "x = " << x.getValue() << endl;
   cout << "y = " << y.getValue() << endl;
}
```

Namn:_____Personnummer:_____

```
int main() {
   test();
   cout << "ok" << endl;
}
```

Allting kompilerar men redan första körningen resulterar i en crash. Vad är det som går fel?

Svar: När `swap` anropas skapas ett temporärt objekt, *tmp*, som får en pekare till den *int* som är allokerad för *x*. När proceduren returnerar så avallokeras *tmp*. Detta kommer generera ett anrop till destruktorn *IntCell* vilket i sin tur kommer att göra *delete* på den *int* som *tmp* har en pekare till. Denna *int* var ju allokerad av *x* och borde inte alls avallokeras.
Redan när vi gör anropen *y.getValue()* är vi ute på hal is eftersom vi nu har en pekare till en *int* som vi gjort *delete* på. Har vi tur så överlever vi men utfallet är odefinierat.
När vi returnerar från `test` så kommer *x* och *y* att avallokeras men *y* har ju då en pekare till den *int* som en gång allokerades för *x* (och som vi redan gjort *delete* på). Vi kan ha tur men det troliga är, som i det här fallet, att det resulterar i en crash.

5.2 vilka objekt modifieras [2 poäng]

Givet programmet nedan, vad kommer att skrivas ut på `cout` och varför? Skriv en kort motivering till ditt svar.

```
class IntCell
{
  public:
     IntCell(int initialValue = 0 )
       { storedValue = new int(initialValue); }
     int getValue() const
       { return *storedValue; }
     void setValue(int val)
       { *storedValue = val;}
  private:
       int *storedValue;
};

int main()
{
  IntCell a(2);
  IntCell b = a;
  IntCell c;

  c = b;
  a.setValue(4);

  cout << a.getValue() << endl;
  cout << b.getValue() << endl;
  cout << c.getValue() << endl;
}
```

Svar: Det kommer att skrivas ut tre stycken 4:or. Anledningen är att objekten *b* och *c* kommer att peka på samma *int*. När vi modifierar denna genom *a.setValue(4)* så kommer alla anrop till *getValue()* returnera samma sak. Observera att *b* inte pekar på *a*, inte heller pekar *c* på *b*. Objektet *b* har en pekare till den *int* som blev allokerad för *a*.

Namn:_____Personnummer:_____

5.3 En student är också en person, eller? [2 poäng]

Givet följande definitioner:

```
class Person {
  string name;
public:
  Person(const string &nn) : name(nn) {};
  void print() const {
    cout << name;
  };
};

class Student : public Person {
  string prgm;
public:
  Student(const string &nn, const string &cc) : Person(nn), prgm(cc) {}
  void print() const {
    Person::print();
    cout  << "/" << prgm;
  }
};

void foo(const Person &p) {
  p.print();
};
```

Vad skrivs ut när vi kör programmet nedan? Varför och hur skull vi kunna ändra på beteendet?

```
int  main() {
  Student jim("Jim", "ICT");
  foo(jim);
}
```

Svar: Programmet skriver ut "Jim" eftersom C++ använder "statisk dispatch" och proceduren foo då kommer att ta för givet att *p* är en *Person*. Om vi vill att *foo* vid run-time skall kunna avgöra vilket objekt den får måste vi deklarera metoden *print* som virtuell (mha `virtual`). Vi skulle också kunn ha två olika versioner av *foo* eler att definera den med hjälp av ett template.

Namn:_____Personnummer:_____

6 Komplexitet [totalt 6 poäng]

6.1 Sökning i ordnat träd [2 poäng]

Antag att vi har representerat en mängd nyckel-värde par i ett ordnat träd och har en sökfunktion som plockar fram ett värde givet en nyckel. Vilken tidskomplexitet har en sådan funktion? Svara med den asymptotiska tidskomplexiteten exempelvis *O(n) där n är .....* Glöm inte att ange vad *n* är.

Svar: Sökning i ett träd är *O(log(n))* där n är antalet element i trädet. Att säga att sökning är *O(n)* där n är djupet i trädet är iofs rätt men frågan är väl då kanske hur djupet fårhåller sig till antalet element..

6.2 Hitta rätt lista [2 poäng]

Antag att vi har en representerar en mängd sekvenser som en lista av listor. Vi har en sökfunktion som letar efter en sekvens genom att traversera listan och jämföra sekvenser i listan med den sekvans som man söker efter. Jämförelsen görs genom att jämföra element för element av de två sekvenserna. Vilken tidskomplexitet har en sådan funktion? Svara med den asymptotiska tidskomplexiteten exempelvis *O(n) där n är .....* Glöm inte att ange vad *n* är.

Svar: Sökningen är *O(n)* där $n = k*l$, k är längden på listan och l är längden på de sekvenser vi arbetar med. Detta under förutsättning att vi kan avgöra om två element är lika i konstant tid. Att svara $O(n^2)$ där n är längden på den längsta av listan (inlkuderat listan som inehåller listorna) är inte riktigt rätt, antag att sekvenserna alla är l långa, hur beter sig exekveringstiden om vi dubblerar längden på listan som håller sekvenserna?

6.3 Sudoku [totalt 2 poäng]

Antag att vi har skrivit en sudokulösare som kan lösa sudokus av storlek 9x9. Vi har testkört den för ett antal sudokus och sett att den hittar en lösning grovt uppskattat en tid som är proportionell till $4^{n/4}$ där n är antalet okända (n typiskt ligger mellan vara mellan 40 och 60). Detta motsvarar ett sökträd med förgreningsfaktor 4 (antalet val man har när man måste gissa) och djup $n/4$ (antalet gissningar vi gör om vi hela tiden gissar rätt).
Antag att vi kan förbättra vår propagreringsalgoritm så att vi kan halvera antalet val vi har när vi måste gissa till 2 och antalet sökningar till $n/8$, vi kommer alltså ha svarstider som är proportionella till $2^{n/8}$. Den mer kraftfulla algoritmen tar dock ca 1000 gånger så läng tid att köra varje gång den används, är det värt att använda den? Gör en grov uppskattning av hur körtiderna skulle ändras om vi använde den förbättrade algoritmen.

Svar: Vi kommer gå från $2^{n/2}$ till $2^{n/8}$ antalet sökningar men varje propagering kostar en faktor 2^{10}. Om n är 40 till 60 så går vi från 2^{20} - 2^{30} till cirka 2^{15}

2^{18}. Vi vinner alltså i storleksordningn 2^5 - 2^{12}, med andra ord en rätt så bra affär.

Namn:_____Personnummer:_____

Appendix

Följande Erlang-funktioner kan komma till användning.
map(Fun, List): returnerar en lista där funktionen applicerats på varje element i den givna listan.
filter(Fun, List): returnerar en lista av de element i listan för vilka funktionen returnerar *true*.
foldl(Fun, Acc, List) och foldr(Fun, Acc, List): returnerar en värde som fås genom att applicera funktionen på varje element i listan och ett ackumulerat värde där Acc är det initiala ackumulerade värdet. Funktionen foldr börjar med det sista elementet och foldl med det första elementet