Tentamen
EDAF30 Programmering i C++

2016–08–16, 8.00–13.00

Hjälpmedel: En valfri C++-bok. OH-bilderna från föreläsningarna är *inte* tillåtna.

Du ska i dina lösningar visa att du behärskar C++ och även att du kan använda C++ standardklasser och algoritmer. Rena ”C-lösningar” på problem som med fördel kan lösas enligt mera objektorienterade principer, eller egen implementering av sådant som finns i standardbiblioteket, kan därför ge poängavdrag, även om de är korrekta. I bedömningen av en lösning kan även minneshantering beaktas där det är relevant. Om du använder saker ur standardbiblioteket i din kod, var tydlig med att du gör det. Använd fullt kvalificerade namn eller `using`-direktiv för de namn du använder.

Uppgifterna ger preliminärt $14 + 12 + 7 + 11 + 6 = 50$ poäng. För godkänt krävs preliminärt 25 poäng (betygsgränser 3/25, 4/33, 5/42). Observera att ordningsföljden eller poängantalet för uppgifterna inte nödvändigtvis avspeglar deras svårighet.

---

1. I kursboken (J. Skansholm: C++ direkt) hittar man följande funktion som använder sig av binärsökning för att leta upp positionen (returnerar en pekare till elementet) för ett heltal i en sorterad array (*”fält”*):

```
int *bin_sok(int sokt, int *forsta, int *sista) {
        if (forsta>sista)
                return nullptr;
        int *mitten = forsta + (sista-forsta)/2;
        if (sokt<*mitten)
                return bin_sok(sokt,forsta,mitten-1);
        else if (sokt>*mitten)
                return bin_sok(sokt,mitten+1,sista);
        else
                return mitten;
}
```

   a) Skriv om funktionen till en funktionsmall så att man i stället för att bara kunna leta i en int-array kan söka efter ett element i en sorterad array av godtycklig typ (med element som kan jämföras).

Uppgiften fortsätter på nästa sida...

b) Antag att vi vill använda din funktionsmall ovan för att söka efter element av klassen Bil enligt nedan. **OBS!** Sökningen ska ske baserat på *registreringsnumret* för bilen

```
class Bil {
public:
    Bil() : reg_nr(nullptr), owner(nullptr) { }
    void set(const string& r, const string& o) {
        delete reg_nr;          // delete nullptr is OK in C++.
        delete owner;           // Null checks not required.
        reg_nr = new string(r);
        owner = new string(o);
    }
    string get_reg_nr() const {
        return *reg_nr;
    }
    string get_owner() const {
        return *owner;
    }
private:
    string *reg_nr;
    string *owner;
};
```

Nedan finns ett litet exempel som visar hur det ska kunna fungera:

```
    Bil bil[4];
    bil[0].set("ABC123","Pelle");
    bil[1].set("BCD234","Lena");
    bil[2].set("CDE345","Oskar");
    bil[3].set("DEF456","Sofia");
    Bil s;
    s.set("BCD234","");
    Bil *b = bin_sok(s,bil,bil+3);
    if (b!=0) {
        cout << b->get_reg_nr() << endl << b->get_owner() << endl;
    } else {
        cout << "Fanns ej!" << endl;
    }
```

Komplettera koden för klassen `Bil` (dvs ändra ej existerande kod - lägg bara till ny kod) så att koden i exemplet ovan fungerar!

c) Ett generellt problem med klassen `Bil`, som den definieras i den föregående uppgiften, är att man riskerar att råka ut för en minnesläcka när man använder den.
Vad innebär en minnesläcka?

d) *Komplettera* koden (ändra ej existerande kod) för klassen `Bil` ovan så att användning av den inte medför en minnesläcka. Var observant så att du inte i och med ändringen introducerar fel som beror på för tidig avallokering av minne.

e) Om du hade fått lov att ändra i den ursprungliga koden för klassen `Bil` skulle det gå att åtgärda problemen med minnesläckor på ett mycket enklare sätt (utan att ändra klassens beteende utåt – det publika gränssnittet). Hur?

2. Betrakta följande vektorklassmall:

```
template <typename T>
class Vector {
public:
    Vector(size_t s) : v(new T[s]), sz(s) {
        for (size_t i = 0; i != sz; ++i) {
            v[i] = T();
        }
    }
    ~Vector() { delete[] v; }
    size_t size() const { return sz; }
    T get(size_t i) const { return v[i]; }
    void put(size_t i, T val) { v[i] = val; }
private:
    T* v;
    size_t sz;
};
```

a) När `Vector` används i följande program får man ett oväntat resultat:[1]

```
void print(Vector<int> v) {
    for (size_t i = 0; i != v.size(); ++i) {
        cout << v.get(i) << " ";
    }
    cout << endl;
}

int main() {
    Vector<int> v1(10);
    for (size_t i = 0; i != v1.size(); i++) {
        v1.put(i, i + 1);
    }
    print(v1);    // utskrift 1 2 3 4 5 6 7 8 9 10
    Vector<int> v2(5);
    print(v2);    // utskrift 0 0 0 0 0
    print(v1);    // utskrift 0 805306368 0 805306368 3 6 7 8 9 10
}
```

Den sista `print(v1)`-satsen borde ha gett samma utskrift som den första! Förklara varför det har blivit fel (du behöver inte förklara vad de stora värdena står för) och korrigera *klassmallen* så att programmet fungerar som förväntat.

Ledning: Felet är relaterat till hur man hanterar minne i C++.

b) Det är inte elegant att behöva utnyttja funktionerna `get` och `put` för att komma åt vektorelementen. Lägg till konstruktioner i `Vector` så att man kan komma åt dess element med indexering, till exempel `v1[i] = i + 1`. Lägg också till en funktion så att man kan skriva ut vektorer, till exempel `cout << v2 << endl << v1 << endl`.

[1] Programmet är kört på en Mac med Xcode version 6.1.1.

3. En programmerare har skrivit ett litet testprogram för att lära sig använda iostream, men vid testkörning fås ett oväntat resultat.

```
#include<iostream>
using std::cin;
using std::cout;
using std::endl;

int main() {
    const int prefix_len = 8;

    int nbr;
    char prefix[prefix_len];

    cout << "Enter number of times to loop:" << endl;
    cin >> nbr;

    cout << "Enter a prefix (max length: " << prefix_len << " chars):" << endl;
    cin >> prefix;

    cout << "Looping " << nbr << " times." << endl;
    while(nbr--) {
        char txt[100];
        cout << "Enter a text:" << endl;
        cin >> txt;
        cout << prefix << " : " << txt << endl;
    }
    cout << "Program finished" << endl;
    return 0;
}
```

Testkörning som fungerar:

```
Enter number of times to loop:
3
Enter a prefix (max length: 8 chars):
test
Looping 3 times.
Enter a text:
aaaa
test : aaaa
Enter a text:
bb
test : bb
Enter a text:
ccccc
test : ccccc
Program finished
```

Testkörning med oväntat resultat:

```
Enter number of times to loop:
5
Enter a prefix (max length: 8 chars):
abcdefgh
Looping 0 times.
Program finished
```

Programmeraren förväntade sig att programmet skulle fråga efter en sträng fem gånger, men det avslutas direkt. Förklara vad som händer i körningen med oväntat resultat.
Ledning: Tänk på hur en C-sträng representeras i minnet.

4. I ett program läser man kommandon och heltal och summerar talen. Exempel på konversation med programmet (kommentarerna ingår inte; `p`, `a`, `u`, `c` och `r` är kommandon):

```
p       // print the sum, 0 to begin with
Sum is now 0
a 4     // add 4
a 3     // add 3
a 2     // add 2
p
Sum is now 9
u       // undo last add
u       // multiple undo's are allowed
p
Sum is now 4
c       // commit changes (make them permanent so they cannot be undone)
u       // nothing happens
a 3
a 2
p
Sum is now 9
r       // rollback, undo all changes since last commit
p
Sum is now 4
```

Programmet ser ut så här:

```
#include <iostream>
#include "accumulator.h"

using std::cout;
using std::cin;
using std::endl;

int main() {
    Accumulator accum;
    char cmd;
    while (cin >> cmd) {
        switch (cmd) {
            case 'p':
                cout << "Sum is now " << accum << endl;
                break;
            case 'a': {
                int nbr;
                cin >> nbr;
                accum += nbr;
                break;
            }
            case 'u':
                accum.undo();
                break;
            case 'c':
                accum.commit();
                break;
            case 'r':
                accum.rollback();
                break;
        }
    }
}
```

Implementera klassen `Accumulator`.

5. I ett program som hanterar djur finns klasserna

```
class Pet {
public:
    Pet() =default;
    virtual ~Pet() =default;
    virtual string speak() const =0;
};

class Dog : public Pet {
public:
    virtual string speak() const override {return "Woof";}
};

class Cat : public Pet {
public:
    virtual string speak() const override {return "Meow";}
};

class Bird : public Pet {
public:
    virtual string speak() const override {return "Tweet";}
};
```

För att testa djurens läten har ett minimalt testprogram skrivits:

```
#include <iostream>
#include <vector>
#include "pet.h"

void test_pets()
{
    Bird b;
    Cat c;
    Dog d;

    std::vector<Pet> pets;

    pets.push_back(b);
    pets.push_back(c);
    pets.push_back(d);

    for(Pet& p : pets) {
        std::cout << p.speak() << std::endl;
    }
  }
```

Tyvärr går det inte att kompilera programmet: kompilatorn klagar på raden `std::vector<Pet> pets;`

a) Varför får man inte lov att skapa en `std::vector<Pet>`?

b) Ändra funktionen `test_pets` så att den fungerar. Funktionen ska skapa djur-objekten, skapa en `vector` och sen iterera över vektorn och anropa `speak()` för varje position så att utskriften blir

```
Tweet
Meow
Woof
```