2D1387 Programsystemkonstruktion med C++
Lösningsförslag, tentamen 9 januari 2003

Obs: Dessa lösningar är just lösningsförslag. Rättningen kan se något annorlunda ut.

Uppgift 1

```
A B B B
```

Uppgift 2

a) Exempel på implementation:

```
template<class T>
const T &min(const T &t1, const T &t2)
{
    return t1 < t2 ? t1 : t2;
}

template<class T>
const T &max(const T &t1, const T &t2)
{
    return t1 < t2 ? t2 : t1;
}
```

b) Exempel på implementation:

```
template<class T>
void swap(T &t1, T &t2)
{
    T tmp(t1);
    t1 = t2;
    t2 = tmp;
}
```

c) För att byta innehåll i vektorn byter vi plats på dess medlemmar. Exempel på implementation:

```
template<>
void swap(Vector &v1, Vector &v2)
{
    swap(v1.size, v2.size);
    swap(v1.data, v2.data);
}
```

Uppgift 3

Problemen och förslag på åtgärd:

```
class A {
public:    // 1: i synlig från main
    static int i;
public:    // 2: j synlig från B(int)
    int j;
public:
    virtual void foo() {}
};
struct B : public A {
    B(int i) : c(i) { j = i; }   // 3: c finns inte
private:
    virtual void foo() {}
};
int A::i;    // 4: A::i saknar definition
int main()
{
    A a;
    B b(1);   // 5: B har ingen defaultkonstr
    a.i = 7;
    b.j = 3;
    A &ar = b;
    ar.foo();
    return 0;
}
```

Uppgift 4

a) Alla anrop till `A`-objektet måste gå via en pekare, vilket tar tid. `A`-objektet måste allokeras dynamiskt, vilket kan ta tid om det blir många små objekt som ska allokeras och avallokeras. Det dynamiskt allokerade minnet kan ligga på en annan plats i minnet, vilket kan ge cachemissar. Detta är extra illa om man itererar över en vektor med `B`-objekt, där varje dynamiskt allokerat `A`-objekt ligger på en egen minnessida. Dessutom måste man implementera det interface som man annars hade ärvt från `A`.
b) Arv är nödvändigt vid åtkomst av medlemmar som är `protected`, om `B` ska användas istället för `A`-objekt (t.ex. då `B *` konverteras till `A *`) och om ett `A`-objekt inte kan skapas dynamiskt pga att konstruktorn eller `operator new()` är `protected`.