2D1387 Programsystemkonstruktion med C++
Lösningsförslag, tentamen 30 augusti 2001

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

Uppgift 1

a) `A(A)` kopieringskonstruktor för argumentet `a`, `A()` konstruktor för `b`, `A=A` kopiering `b = a`, `~A` destruktor för `a`, `~A` destruktor för `b`.
b) `A(A)` kopieringskonstruktor för `b`, `A(A)` kopieringskonstruktor för `c`, `~A` destruktor för `b`, `~A` destruktor för `c`.

Uppgift 2

```
template<class T>
T *my_new(unsigned int count)     // undvik neg storlekar
{
    if(count == 0)                // allok. minst ett elem
        count = 1;
    void *p = malloc(sizeof(T) * count);  // allok. minne
    if(!p)                        // slut på minne
        throw std::bad_alloc();

    T *tmp = static_cast<T *>(p); // iterator
    T *ret = new (tmp) T;         // returvärde, konstruera
    tmp++;                        // första pos avklarad ovan
    for(unsigned int i = 1; i < count; i++, tmp++)
        new (tmp) T;              // konstruera objekten

    return ret;
}

template<class T>
void my_delete(T *p, unsigned int count)
{
    T *tmp = p;                   // iterator
    for(unsigned int i = 0; i < count; i++, tmp++)
        tmp->~T();                // kalla dtor explicit
    free(p);                      // frigör minne
}
```

Uppgift 3

Följande fyra defekter ger kompilatorfel:

- `A b = 3.14;` Denna syntax, som i övrigt är ekvivalent med `A b(3.14)`, är inte tillåten då konstruktorn är explicit.

- `bar("xyz");` Funktionen `bar(A &)` får en temporär (implicit skapad av konstruktorn `A("xyz")`) som argument. Språket tillåter inte icke-konstanta referenser att referera till temporärer.
- `foo(3.14)` En implicit konvertering genom konstruktion kan inte ske eftersom konstruktorn som tar `double` är `explicit`.
- `foo(A())` En implicit given defaultkonstruktor ges bara av kompilatorn om inga andra konstruktorer deklarerats. Det finns alltså ingen defaultkonstruktor.

Uppgift 4

a) Man förhindrar användare av klassen att skapa instanser, men i klassen själv och i nedärvda klasser kan man fortfarande skapa instanser och kopior.
b) Några exempel på användningar av `protected` och `private`:

- Man kan förhindra objekt på stacken genom att göra destruktorn privat (för att skapa objekt krävs då en s.k. factoryfunktion och en `destroy()`-funktion som gör `delete this`).
- Man kan förhindra objekt på heapen genom att göra operatorn `new` privat.
- Man kan förhindra kopior av ett objekt genom att låta både kopieringskonstruktor och tilldelningsoperator vara private och sakna definition. Vissa objekt lämpar sig inte för kopiering, t.ex. ett objekt av typen 'operativsystem'.