2D1387 Programsystemkonstruktion med C++
Lösningsförslag, tentamen 22 oktober 2002

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

Uppgift 1

`A A B A`

Uppgift 2

a) Exempel på program:

```
template<class T, class It>
T accumulate(T t, It beg, It end)
{
    for(; beg != end; ++beg)
        t += *beg;
    return t;
}
```

b) Startvärdet `""` kan inte användas med `operator+=` eftersom man inte kan addera pekare. Lösning på problemet är att använda en standardsträng: `accumulate<std::string>("", b + 0, b + 3)` eller `accumulate(std::string(""), b + 0, b + 3)`. Man kan även skriva en specialisering för `accumulate<const char *>`.

Uppgift 3

Klassen `BigNum` för godtyckligt stora tal:
a) Deklarationer:

- `BigNum &operator++();`
- `const BigNum operator++(int);`
- `const BigNum operator+(const BigNum &) const;`
- `BigNum &operator+=(const BigNum &);`

b) Typer för returvärden och argument:

- Returtyper: `const BigNum` när ett nytt värde skapats (i t.ex. `operator+` ovan). Detta är en temporär och bör inte ändras. Följdaktligen är `BigNum` överflödig och borde vara `const BigNum`. `BigNum &` används när man vill ändra den returnerade variabeln. `const BigNum &` används när man inte vill kopiera hela objektet men inte vill riskera att någon modifierar datat.
- Argumenttyper: `BigNum &` tolkas som en variabel som kommer att ändras i funktionen. Argumentet kan därför användas som returvärde eftersom det förmedlar data till anroparen. `const BigNum &` används när man inte

vill kopiera hela objektet. `BigNum` används när man vill ha en kopia för sidoeffekternas skull (ovanligt) eller när datatypen är så liten att referensanrop ger onödigt arbete. `const BigNum` är faktiskt exakt samma sak som att bara ange `BigNum`. Det är t.ex. ganska vanligt att man anger att en vektorstorlek ska skickas som `int` i deklarationen och väljer att göra den `const int` i definitionen för att inte råka ändra värdet.

- Konstanta funktioner: Konstanta objekt kan bara anropa funktioner som är `const`, dvs funktioner som lovar att inte ändra objektet. Annars skulle ett konstant objekt kunna råka ändras om man anropar en icke-konstant funktion.

c) I en global funktion kan även det vänstra objektet konverteras. Ta t.ex. addition: `BigNum b; 1+b;`. I detta fall kan inte `operator+` anropas om operatorn är en medlemsfunktion. Är operatorn global så används en konstruktor för att konstruera ett `BigNum`objekt av `1`, och sedan sker addition. Alla operatorer i **a** går att göra globala. (Däremot går inte tilldelningsoperatorn `operator=` att göra global, bl.a. eftersom kompilatorn ger denna implicit. Det skulle alltså inte gå att avgöra för kompilatorn om någon har lagt till ytterligare en (global) tilldelningsoperator i någon annan implementationsfil, och du skulle få länkningsfel.)

Uppgift 4

a) Alla fyra funktionerna anropas rekursivt för alla medlemar (ex: defaultkonstruktorn kör defaultkonstruktorn för alla medlemmar). Är det en inbyggd typ (t.ex. `int` eller `void *`) kopieras datat bitvis. En konstruktor anropar dessutom basklassens konstruktorer. Destruktorn anropar basklassernas destruktorer.
b) Tilldelningsoperator och defaultkonstruktor kan inte skapas eftersom `A` innehåller en referens.

```
A::A(const A &o) : s(o.s), p(o.p), r(o.r), i(o.i)
{ memcpy(a, o.a, sizeof(a)); /* eller for-loop */ }
A::~A() { s.~std::string();  /* kör destruktor */ }
```