2D1387 Programsystemkonstruktion med C++

Lösningsförslag, tentamen 25 oktober 2001

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

Uppgift 1

`7 7 7 B B A` (på sex rader)
Medlemsdata har inte polymorfiskt beteende så variabeln `i` kommer att läsas från typen `A`.

Uppgift 2

Exempel på program:

```
template<class T>
std::string sum_elems(const char *s, T it1, T it2)
{
    return sum_elems(std::string(s), it1, it2);
}

template<class S, class T>
S sum_elems(S ret, T it1, T it2)
{
    for(T t = it1; t != it2; ++t)
        ret += *t;

    return ret;
}
```

Överkurs: Motivering till utelämnat `const T &` är att komplilatorn inte kan avgöra om referenstypen är `int *` eller `int[6]` i första anropet till `sum_elems`. Detta problem får man inte om man inte använder referenser, eftersom `int[6]` då alltid konverterar till `int *`.

Uppgift 3

Programmet innehåller bl.a. följande konstigheter:

- Semikolon saknas på `B1` och `B2`.
- `A1` har privat destruktor vilket gör att variablerna `a` och `b` inte kan frigöras med `delete`.
- Destruktorn i `A1` har ingen definition.
- Variabeln `a` allokeras med `new[]` och deallokeras med `delete` vilket ger odefinierat beteende.
- Variabeln `d`: `A2` är inte en basklass till `B1`.
- Destruktorn i medlemsvariabeln `B2::s2` körs inte eftersom basklassen `A1` inte har virtuell destruktor. Detta resulterar i en minnesläcka eftersom std::string har dynamiskt allokerat minne.

Uppgift 4

a) Det är alltid bra att skilja gränssnitt från implementation. Kompileringen går snabbare eftersom implementationsfilerna blir mindre till storleken. Kompileringstiderna minskar eftersom beroendena minskar (ta t.ex. extremfallet med endast en .h-fil och en .cpp-fil där allt kompileras om vid varje ändring). Det blir lättare att återanvända delar av koden om inte onödig kod följer med på köpet.
b) Definitioner bör inte ligga i headerfilen eftersom inkludering i flera .cpp-filer då ger multipla definitioner vilket ger länkningsfel.
c) Deras definitioner måste vara synliga vid användningen, annars kan inte `inline`-funktioner "klistras in" och `templates` instansieras.
d) Kompilatorn skulle lägga ut information om vilka `inline`-funktioner och `templates` som varje .cpp-fil definierar resp. använder. Länkaren samlar upp denna information och avgör vilken .cpp-fil som behöver vilken definition. Kompilatorn körs sedan för varje fil som använder `inline`-funktioner eller `templates`. Efter detta måste koden länkas.