LINKÖPINGS TEKNISKA HÖGSKOLA
Institutionen för datavetenskap
Avdelningen för Programvara och system
Tommy Olsson

2013-08-13

Checklista för grundläggande konstruktion av klasser

1. Följer din klass principen att en klass ska göra *en* sak och göra det *bra*?
   Kommentar: *En sak* kan vara komplicerad och att göra *en* sak bra innebär vanligtvis att klassen är komplett avseende *grundläggande* funktionalitet.

2. Har du valt lämpligt åtkomstskydd för klassmedlemmar?
   Tumregel: Datamedlemmar och medlemsfunktioner som inte ska kunna användas direkt av annan kod ska vara **private**. I basklasser kan **protected**-deklarerade medlemsfunktioner vara ett sätt att ge åtkomst till privata datamedlemmar från subklasser (s.k. åtkomstfunktioner).

3. Har du **const**-deklarerat alla medlemsfunktioner som inte ändrar (ska kunna ändra) på ett objekts tillstånd?
   Kommentar 1: **const**-deklaration av medlemsfunktioner är viktigt i flera avseenden; tydlig markering i koden att funktionen inte ändrar objekt (kodningsstil); kompilatorn kontrollerar att konstanthet uppfylls (säkerhet); endast **const**-deklarerade funktioner kan anropas för konstanta objekt (användbarhet).
   Kommentar 2: Ibland kan en medlemsfunktion som logiskt sett inte ändrar på ett objekts tillstånd, ändra en intern variabel. Då kan man ändå **const**-deklarera medlemsfunktionen om man **mutable**-deklarerar den interna variabeln.

4. Har din klass konstruktorer så att objekt kan initieras på önskat sätt?
   Tumregel: en klass ska normalt ha en konstruktor som kan anropas utan argument (*defaultkonstruktor*) om inte specifikationen säger annat eller om det motsägs av annan anledning.
   Kommentar: I en klasshierarki kan det tänkas att vissa konstruktorer ska vara publika, medan andra bara ska kunna användas av subklasser och då ska skyddas. En klass kan också tänkas ha konstruktorer som endast klassens egna medlemsfunktioner ska ha tillgång till och då vara privata.

5. Har du definierat *kopieringskonstruktor*, *kopieringstilldelningsoperator* (=) och *destruktor* för en klass, om de kompilatorgenererade versionerna av dessa speciella medlemsfunktioner inte fungerar?
   Tumregel: Om en klass har en icke-trivial destruktor ska den normalt också ha användardefinierad kopieingskonstruktor och kopieringstilldelningoperator. En icke-trivial destruktor är t.ex. en som återlämnar dynamiskt minne eller frisläpper andra resurser.
   Kommentar: För vissa klasser kan inte kopiering tillåtas av någon anledning och då eliminerar man kopieringskonstruktorn och kopieringstilldelningsoperatorn genom att deklarera dem som *deleted*.

6. Om du har definierat en egen kopieringskonstruktor och en egen kopieringstilldelningsoperator är det troligt att det även ska finnas en *move-konstruktor* och en *move-tilldelningsoperator.*

7. Har du deklarerat medlemsfunktion som ska ha polymorft beteende **virtual**?

8. Har den översta basklassen i en klasshierarki (rotklassen) en *virtuell destruktor*?
   Tumregel: En klass som har en virtuell medlemsfunktion ska ha en publik virtuell destruktor.
   Kommentar 1: Även om inte basklassen i sig behöver ha en destruktor måste ändå en virtuell destruktor deklareras för att säkerställa att subklassers destruktorer körs i alla sammanhang och speciellt i polymorfa sammanhang.
   Kommentar 2: Om klasserna i en klasshierarki aldrig kommer att användas polymorft behövs ingen virtuell destruktor och i sådant fall ska det inte heller finnas några virtuella medlemsfunktioner.

9. Har du förhindrat automatisk typomvandling med typomvandlande konstruktorer, om det *inte* är önskvärt?
   Kommentar 1: En typomvandlande konstruktor kan anropas med *ett* argument av annan typ än klassen ifråga. Typomvandling från en klass till en annan typ med typomvandlande operator ska man vara mycket restriktiv med, sådana ställer ofta till med mer problem än de löser.

Kommentar 2: Definiera gärna optimerade versioner av t.ex. överlagrade operatorer för att undvika automatisk typomvandling med tillhörande temporära objekt i typblandade uttryck.

10. Följer dina egendefinierade operatorfunktioner den semantik som gäller för motsvarande inbyggda operatorer?
    Kommentar: tilldelningsoperatorer, t.ex., ska normalt returnera en referens till vänsteroperanden.
11. Klasser som inte ska kunna användas som basklasser kan märkas med ’final’ i specifikationen.

Vanliga fel i samband med klasser

Olika översättningssystem för C++ kan detektera ett visst fel i olika faser av översättningen, antingen under kompileringen, fel meddelas vanligtvis i termer av ”`error:`…”, eller vid länkningen, fel meddelas i termer av ”`ld:`...” eller ”`ild:`…” (*incremental*).

1. *Kompilatorn/länkaren säger att definitionen för en deklarerad medlemsfunktion saknas*.
   Det kan stämma – du har helt enkelt glömt bort den separata definitionen.
   Det finns en definition – varför hittas den inte?
   Om definitionen finns på en separat implementeringsfil kan problemet bero på att du har glömt att ta med implementeringsfilen i kompileringen/länkningen.
   Du kan ha glömt klassprefixet framför namnet i definitionen.
   Om funktionen var **const**-deklarerad i deklaration kan du kan ha glömt **const** i definitionen (**const** är särskiljande vid överlagring och överridning).
   Det kan finnas någon annan avvikelse i funktionshuvudena för deklarationen och definitionen, t.ex. i parameterdeklarationerna (antal, typ, ordning).
2. *Kompilatorn/länkaren säger något om virtual table,* `__vtable`*, eller liknande*.
   Gäller något fel kopplat till virtuella medlemsfunktioner. Felet kan i många fall vara av samma slag som anges under punkten ovan, dvs av någon anledning kan inte en definition som motsvarar en viss deklaration hittas.
3. *Kompilatorn signalerar fel i en deklaration, t.ex. i en deklarationen av en parameterlista och du ser absolut inget fel och speciellt inte där kompilatorn anger att det skulle vara fel*.
   Det kan bero på att någon parameter är av standardbibliotekstyp, t.ex. `string`, och du har glömt att inkludera motsvarande ”header” (`<string>`), eller glömt att öppna namnrymden `std`, eller glömt att använda namnrymdsprefixet `std::`, t.ex. `std::string`.
4. *Programmet kraschar oväntat med t.ex. ett kryptiskt meddelande som ”bus error” eller liknande. Du har svårt att detektera exakt var felet inträffar men det verkar vara kopplat till användningen av klassobjekt (t.ex. retur från funktion, tilldelning).*
   Det kan hänga ihop med att du har en icke-trivial klass och du har glömt att definiera egna versioner av de speciella medlemsfunktionerna, t.ex. kopieringskonstruktor och kopieringstilldelning, vilket kan leda till minneshanteringsfel, speciellt om en destruktor har definierats.
   Programmet kan ha ”skrivit sönder minnet”, t.ex. genom att ha stegat för lång i fält (inte sällan teckenfält) och därigenom ändrat i minne som tillhör en annat objekt.
5. *Programmet kraschar med meddelande ”Segmentation fault”*.
   Typiskt relaterat till pekarfel: en tompekare avrefereras, en pekare som används är oinitierad, …
   Programmet kan även ha hamnat i en oändlig rekursion och vilket medfört att exekveringsstacken har förbrukats.

Det kan skilja en hel del i felmeddelanden mellan olika system och ibland kan kompilering med olika kompilatorer vara till hjälp för att försöka förstå annars svårbegripliga felmeddelanden.

Kolla alltid kompileringsfelen som kommer först i listan – ibland kan det vara enkla fel som genererar mängder av felmeddelanden. Leta efter nyckelord som ”fel”/”error”, ”varning”/”warning”.