Linköpings universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2016-10-25

Pekare och länkad lista

Mål

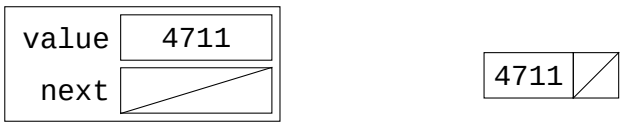
I den här laborationen skall du lära dig hantera pekare och dynamiskt minne. Speciellt ska du lära dig hur konstruktorer, destruktorer och operatorer i en klass kan hjälpa till med ansvaret att hålla reda på pekare och allokerat minne i en klass.

Läsanvisningar

- Rekursion (recursion)
- Pekare
  - Adressoperatorn `&`
  - Innehållsoperatorn, avreferering med `*`
- Inre klasser
- De fem viktiga
  - Kopieringskonstruktorn (djup kopiering till nytt objekt)
  - Tilldelningsoperatorn (djup kopiering till befintligt objekt)
  - Flyttkonstruktorn (snabb flytt från döende till nytt objekt)
  - Flytttilldelningen (snabb flytt från döende till befintligt objekt)
  - Destruktorn (djup destruktion)
- Slumpgenerering (`random_device, default_random_engine, uniform_int_distribution`, etc.)

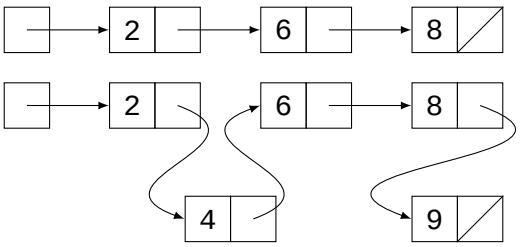
Uppgift: Enkellänkad sorterad lista

Du skall skapa en rak enkellänkad sorterad lista som kan innehålla heltal. Det skall finnas en klass som representerar hela listan, komplett med funktionalitet för att hantera kopiering och flytt av hela listan korrekt i alla lägen, och för att undvika såväl felaktig minnesanvändning som minnesläckage.
Internt, som en inre klass, skall varje länk i listan byggas upp av ett eget klassobjekt som håller reda på det lagrade värdet och en pekare till nästa länk. Det är dessa länkobjekt som tillsammans bygger upp en kedja som formar listan som helhet. Själva listklassen behöver därmed endast innehålla en pekare till den första länken, samt *som minst* funktioner för insättning och borttagning av värden samt utskrift av hela listan. Du kommer märka att du behöver lite fler funktioner för att kunna använda (testa) listan på ett vettigt sätt. Figuren nedan visar representationen i minnet för en länk med värde `4711` och null-pekare, samt hur densamma ritas förenklat.



Figur 1: En länk i minnet, utförligt och förenklat ritat

Den som använder listklassen skall inte någonsin behöva veta hur listan är uppbyggd internt. Länkklassen bör alltså hållas privat och oåtkomlig, alternativt oanvändbar, för allt utom listklassen.
Du bör börja med att tänka igenom hur insättning och borttagning skall fungera och rita operationerna på papper. Det är lämpligt att börja med en tom lista, sätta in `5`, `3`, `9`, `7`, ta bort dem i samma ordning och för varje steg fundera på vilka pekarvariabler som måste uppdateras och i vilken ordning. Figuren nedan ritar ut en lista som den ser ut efter insättning av `6`, `2` och `8`. Därefter hur den ser ut efter insättning av även `9` och `4`.
Alla funktioner du skriver ska testas enligt testdriven utveckling (TDD) med testfall som täcker in alla undantagsfall som kan inträffa, exempelvis kan insättning först behöva uppdatera andra pekare än insättning mellan två befintliga länkar. Fler sådana fall kan finnas beroende på lösningsmetod och använda algoritmer.



Givetvis skall alla tre operationer (insättning, borttagning och utskrift) fungera korrekt på alla listor, inklusive tomma. Likaså skall kopiering och kopieringstilldelning av listor fungera och skapa djupa kopior (ändringar i kopian påverkar inte originalet). Ditt testprogram skall tillgodose detta. Flyttkonstruktion och flytttilldelning ska lämna originalet som en tom lista.

KRAV: Du ska skriva insättningen *rekursivt* och borttagningen *iterativt*.
KRAV: Du ska använda *filuppdelning* för att strukturera programmet.

Valgrind: Att hitta minnesläckage är svårt och i stora program kan det vara nästintill omöjligt att hitta det på egen hand, för att underlätta detta finns det en rad olika verktyg. Verktyget som ni ska använda heter Valgrind och på följande länk finns en kort guide på hur ni kommer igång med det.
UPP-gruppens guide till Valgrind

GDB: Här finns även en kort guide för att komma igång med GDB: **Kodarens Guide till GDB**