Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen
2026-01-15

Länkad lista

Mål

I den här laborationen skall du lära dig hantera pekare och dynamiskt minne i C++. Speciellt ska du lära dig hur konstruktorer, destruktorer och operatorer används för att hantera dynamiskt minne i en klass.

Uppgift

Du skall skapa en rak dubbellänkad sorterad lista innehållandes heltal. Det skall finnas en klass som representerar hela listan med funktioner för att utföra ett antal operationer. Klassen ska också hantera kopiering av hela listan korrekt i alla lägen.

Internt, som en inre klass, skall varje element i listan byggas upp av ett eget klassobjekt som håller reda på det lagrade värdet och en pekare till nästa element och föregående element. Själva listklassen ska endast innehålla två pekare: en till första elementet och en till sista.

Listklassen ska *som minst* ha funktioner för

- sorterad insättning av nya värden (dubbletter tillåts).
- borttagning av värdet på ett specifikt index.
- åtkomst av värdet på ett specifikt index. Index 1 är första värdet i listan. Om ett negativt index anges motsvarar det att räkna från slutet av listan. D.v.s index -1 är sista värdet i listan.
  Krav: För negativa index ska listan itereras från slutet.
- utskrift av hela listans innehåll på en given utskriftsström.

Listan ska kunna skapas antingen tom eller med givna startvärden enligt följande exempel.

```
List empty_list {};            // tom lista skapas
List initialized_list {2,7,5}; // lista med värdena 2,5,7
```

Listklassen ska även implementera de fem speciella medlemsfunktionerna så att kopiering och tilldelning av listor fungerar korrekt i alla lägen.

För att kunna använda (och testa) listan på ett vettigt sätt kommer du behöver lite fler funktioner än minimikraven ovan. Tänk på om alla dessa ska vara synliga eller om några bara är relevanta internt i klassen.

Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2026-01-15

Krav

Samma krav som i tidigare uppgifter gäller. Utöver det ska din implementation också uppfylla nedanstående krav.

- Klassen för listan ska garantera korrekt minnesanvändning och förhindra minnesläckage i alla lägen.
- Programmeraren som använder listklassen skall inte någonsin behöva veta (eller bry sig om) hur listan är uppbyggd internt. Nodklassen ska alltså hållas privat och oåtkomlig för allt utom listklassen.
- Alla funktioner du skriver ska testas med testfall som täcker in alla undantagsfall som kan inträffa, exempelvis kan insättning först i en lista behöva uppdatera andra pekare än insättning mellan två befintliga element. Fler sådana fall kan finnas beroende på lösningsmetod och använda algoritmer. Det krävs att du utöver din klass även redovisar ditt testprogram.
- Du ska noga tänka igenom vad som behöver finnas i det publika gränssnittet och vad som inte ska finnas där. Det är lämpligt att fundera på hur en tilltänkt kollega skulle vilja använda en lista och skapa ett gränssnitt som gör det enkelt att göra precis det. Listklassen ska dölja allt som är inte är relevant för en extern användare. Alla delar av det publika gränssnittet ska testas i testprogrammet.
- I lägen där listan upptäcker fel som den inte själv kan åtgärda ska dessa rapporteras till anropande kod. Listan ska aldrig skriva ut några felmeddelande i terminalen. Se istället till så att felet kan upptäckas programmatiskt och sedan hanteras av anropande kod.

Tillvägagång

1. Jobba enligt TDD för att implementera en del i taget. Börja med att skriva tester för en funktion. Implementera efter det koden som behövs för att testet ska fungera.
2. Upprepa samma process för att implementera alla delar av uppgiften.
3. Kontrollera att du uppnått varje krav.
4. Redovisa och lämna in.
   Du ska lämna in tre filer. En headerfil med tillhörande implementationsfil för din listklass samt ett testprogram.
5. Din assistent utvärderar om du uppnått kraven. Brister ger komplettering.

Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2026-01-15

Koncept som ingår

Alla listade koncept ingår i uppgiften. Endast listade koncept ingår i uppgiften. D.v.s. problem som inte är lämpliga att lösa med nedanstående koncept kan lösas med enbart förkunskaper från kursen eller tidigare kurser.

- Pekare
  - Adressoperatorn &
  - Innehållsoperatorn, avreferering med *
  - Piloperatorn, medlemsåtkomst med ->
- Inre klasser
- Speciella medlemsfunktioner
  - Konstruktorer (defaultinitiering eller från andra värden/typer)
  - Kopieringskonstruktor (djup kopiering till nytt objekt)
  - Kopieringstilldelning (djup kopiering till befintligt objekt)
  - Flyttkonstruktor (snabb flytt från döende till nytt objekt)
  - Flytttilldelning (snabb flytt från döende till befintligt objekt)
  - Destruktor (djup destruktion)

Exempel

Ett bra tips är att rita ut varje operation på ett papper. Rita ut varje steg som behövs en operation. Nedan visas en lista före och efter insättning av ett värde 4. Fundera på vilka delsteg som behövdes på vägen. Tänk på att hålla koll på alla noder och vart varje pekare pekar.

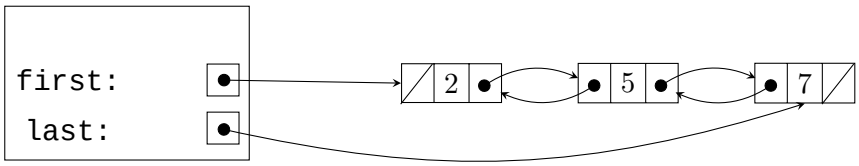

Figure 1: Lista innehållandes värdena 2, 5 och 7

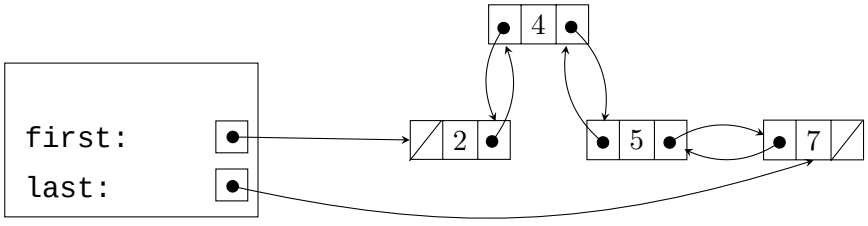

Figure 2: Lista efter insättning av 4