Laboration 5

BOSTADSKÖ

OBJEKTBASERAD PROGRAMMERING I C++
Ver 11

Mål: Laborationen avser att belysa användning av

- dynamiskt länkade listor
- en abstrakt datatyp (ADT) i form av en kö
- iteratorer

Redovisning: Koden skall vara lättläst där indragningar, kommentarer och variabelnamn är logiska och konsekventa.

Skriv en laborationsrapport enligt anvisning i WebCT.

Skicka in källkod och laborationsrapport via WebCT.
Packa alla filerna i en zip-fil.

Regler för inlämning:

Genom att du lämnar in detta arbete försäkrar du att alla svar är skapade av dig själv. Du är även ansvarig att se till att det inte finns någon plagierad text i dokumentet. När du refererar och citerar andra verk måste korrekta källhänvisningar finnas och i fallet citering ska den citerade texten vara tydligt markerad. http://www.bib.miun.se/student/skriva/referenser

Om plagierad text finns i dokumentet riskerar du att stängas av från studier.

Om samarbete sker utan att detta har stöd i instruktionen för examinationen utgör det normalt en disciplinförseelse och du som student riskerar att stängas av från dina studier.

Bostadskö

Inledning

I denna uppgift ska du skriva ett program som administrerar en bostadskö. Kön ska vara av FIFO-typ.

För de personer som ställs i kön ska man kunna lagra följande data:

- namn (förnamn och efternamn)
- adress (gatuadress, postnummer och postadress)
- personnummer och skonummer

För att hantera dessa data har du i tidigare laborationer gjort klassen Person. Använd denna klass!

Ett lämpligt sätt att implementera en kö är att använda en länkad lista som är av kötyp. I en kö kan man endast lägga till data sist i kön och ta bort data först i kön. Om man t.ex. vill ändra på data i kön så går detta inte med dessa funktioner. För att kunna hantera kön utan att påverka själva kön, skapar man en s.k. *iterator*. En *iterator* är en slags pekare som kan peka på enskilda element (noder) i kön. Med en *iterator* kan man läsa och ändra köns element utan att påverka kön.

Du ska i denna laboration skriva klasser för:

- en kölista
- en nod till kölistan. I noden ska data vara av typen Person.
- en iterator till kölistan.
- en klass som hanterar gränssnittet mellan användaren och köklassen

Specifikationer för klasserna ges nedan. Till klasserna kölista, node och iterator får du klassdeklarationerna. Din uppgift blir att implementera medlemsfunktionerna i de fall detta inte redan är gjort. För gränssnittsklassen blir din uppgift att skriva klassdeklarationen och implementera medlemsfunktionerna.

Du ska slutligen skriva ett program som:

- skapar ett objekt av gränssnittsklassen.
- kör medlemsfunktionen run() i vilken en meny för hantering av gränssnittsklassens medlemsfunktioner ligger

Menyalternativ:

- Ställ en person sist i bostadskön.
- Erbjud en person bostad
- Skriv ut hela bostadskön
- Skriv ut data om en person
- Ta bort en person ur bostadskön.
- Spara kön
- Sluta

Headerfil till kölist- och iteratorklasserna

Placera klasser för kö, iterator och nod i **samma** header- och definitionsfil.

Gör följande typedef i h-filen: typedef Person Item; där Person är den klass som du har använt i laboration 2 och 3.

```
// Headerfil
// Filnamn: queue.h
typedef Person Item;
class Node;
class QIterator
{
  private:
    Node *node;
  public:
    QIterator();                // Förvald konstruktor
    QIterator(Node *n);         // Initieringskonstruktor

    Item &operator*() const;
    QIterator &operator++();    // prefix   ++i
    QIterator operator++(int);  // postfix   i++
    bool operator!=(const QIterator &qi) const;
};
//---------------------------------------------------------------------------
class QList
{
     private:
     Node *first, *last;
   public:
      QList():first(0),last(0){};
     ~QList();
     void enque(Item item);
     bool deque(Item &item);
     bool del(Item item);
     bool isEmpty()const;

     QIterator begin()const {return QIterator(first);}
     QIterator end() const { return QIterator(NULL);}
};
```

```
// Defintionsfil
// Filnamn: queue.cpp
//---------------------------------------------------------------------------
// Nodeklassen placeras i cpp-filen för att den ska bli åtkomlig ENBART för
// klasserna QList och QIterator. På detta sätt kan datamedlemmarna göras
// public.
class Node
{
   public:
     Node *next;
     Item data;
     Node (Node *n, Item newData) : next(n), data(newData) {}
};

// Fyll på med funktionsdefinitioner för medlemsfunktionerna i QIterator och
// QList nedan!
```

Specifikationer på nästa sida!

Specifikationer

QIterator

Specifikationer för medlemsfunktionerna i klassen **QIterator**:

Namn: **QIterator**
Uppgift: Sätter datamedlemmen node att peka på NULL
Indata: -
Utdata: -

Namn: **QIterator**
Uppgift: Initierar datamedlemmen node med den pekare som ges som indata
Indata: En pekare till en nod i listan, Node *node
Utdata: -

Namn: **operator***
Uppgift: Överlagrar dereferensoperatorn.
Indata: -
Utdata: Returnerar en referens till data som ligger i den nod som datamedlemmen *node* pekar på.
Returdatatyp: Item &, d.v.s. en referens till ett Person-objekt.
Med denna operator ska man kunna läsa **värden** från en nod i Qlist **och tilldela** nya värden till en nod i QList

Namn: **operator++ (prefix)**
Uppgift: Överlagrar ++ operatorn genom att flytta iteratorn till nästa nod. Denna syntax för överlagringen av ++-operatorn ger en prefix ++ operator.
Indata: -
Utdata: Returnerar innehållet i den nod som iteratorn pekar på **sedan ++** utförts, genom att returnera en referens till det som det egna objektet pekar på dvs. *this.
Returdatatyp: QIterator &

Namn: **operator++ (postfix)**
Uppgift: Överlagrar ++ operatorn genom att flytta iteratorn till nästa nod. Denna syntax för överlagringen av ++-operatorn ger en postfix ++ operator.
Indata: -
Utdata: Returnerar innehållet i den nod som iteratorn pekar på **innan ++** utförs.
Returdatatyp: QIterator

Namn: **operator!=**
Uppgift: Överlagrar olikhetsoperatorn för två QIteratorobjekt
Indata: Ett konstant QIteratorobjekt, const QIterator &qi
Utdata: true om det vänstra och det högra objektet **inte** är lika annars false

Kölista

I klassen QList (kölista) tillkommer följande medlemsfunktioner jämfört med en ”äkta” köklass:

- QIteator begin()
- QIterator end()
- bool del(Item item)

begin() och *end()* är inline-funktioner definierade i headerfilen ovan, medan *del()* specificeras nedan.

Specifikationer för medlemsfunktionerna i klassen **QList**:

Namn: **QList**
Uppgift: Sätter first=NULL och last =NULL
Indata: -
Utdata: -

Namn: **~QList**
Uppgift: Frigör den plats som allokerats för listans noder.
Indata: -
Utdata: -

Namn: **enque**
Uppgift: Lägg till en nod i sist listan och lägg in data för en person (item) i denna nod.
Låt last-pekaren peka på denna nod
Indata: Item item, d.v.s. data för en person (Person)
Utdata: -

Namn: **deque**
Uppgift: Tar bort en nod i början av listan. Data för den person som ligger i noden returneras via en referensdeklarerad parameter. Vidare returneras false om listan är tom annars true
Indata: -
Utdata: false om listan är tom annars true. Item &item, data för den person som tas bort från listan.

Namn: **del**
Uppgift: Tar bort den nod i listan som innehåller den person som skickas med som argument
Indata: Item item, d.v.s. data för den person som ska tas bort ur kön.
Utdata: true om personen angiven i item togs bort, annars false.

Namn: **isEmpty**
Uppgift: Undersöker om det finns några noder i listan.
Indata: -
Utdata: true om listan är tom annars false

Gränssnitt mellan användare och kölistan

Skriv en klass som hanterar **gränssnittet** mellan användaren och kölistan QList. Kalla den för HousingQ. Utforma den med samma struktur som klassen UserInterface i laboration 3.

Namn: **HousingQ**

Datamedlemmar:

- en kölista (QList)
- antal personer i bostadskön (int)
- filnamn (string)

Publika medlemsfunktioner:

- Konstruktor
- run() med innehållet:
  - filnamn anges (det ska inte ändras under programmets gång)
  - ev sparad fil läses in.
  - menyn körs. Se sidan 2 för vad menyn ska innehålla

Privata medlemsfunktioner

- Ställ en person sist i kön
  - Låt användaren mata in data från tangentbordet
- Erbjud en person bostad, d.v.s. ta bort en person från början av kön.
  - Skriv ut data för en person som blev erbjuden bostad.
  - Skriv meddelande om kön är tom.
- Skriv ut hela bostadskön tillsammans med totalt antal personer som står i bostadskön.
  - Vid varje person ska dess placering i bostadskön anges.
  - Skriv meddelande om kön är tom.
- Skriv ut data om en person på skärmen.
  - Sökning efter personen görs med personnummer som söknyckel.
  - Skriv också ut personens placering i bostadskön.
  - Skriv meddelande om kön är tom.
- Ta bort (stryk) en person från bostadskön.
  - Identifiera personen med personnummer.
  - Skriv meddelande om kön är tom.
- Spara kön på fil

Redovisning

Följande kod ska redovisas:

- medlemsfunktioner i köklassen (QList).
- medlemsfunktioner i iteratorklassen (QIterator).
- klassdeklaration och medlemsfunktioner i gränssnittsklassen (Housing Q).
- ett huvudprogram enligt specifikation i inledningen på sidan 2.

Kommentar:

I en kö ska det inte vara möjligt att radera ett element. Ändå ingår det i denna laboration att radera ett element! Anledningen till detta är att du i funktionen som raderar ett element får extra övning i att hantera pekarna i den länkade listan.