Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2013

Øving 10

Frist: 2013-04-27

Mål for denne øvingen:

- Lære å bruke lenkede lister.
- Unntakshåndterings: «`try-catch`» blokker, kaste unntak (`throw` exceptions), «exception»-klasser

Generelle krav:

- bruk de eksakte navn og spesifikasjoner som er gitt i oppgaven.
- det er valgfritt om du vil bruke en IDE (Visual Studio, XCode), men koden må være enkel å lese, kompilere og kjøre.
- skriv den nødvendige koden for å demonstrere implementasjonene dine

Anbefalt lesestoff:

- Kapittel 17 & 18, Absolute C++ (Walter Savitch)
- It's Learning notater

1 Lenkede lister (30 poeng)

I denne øvingen skal du implementere en lenket liste (linked list). En lenket liste er en liste hvor hvert element, i tillegg til å inneholde data, peker til det neste elementet i listen. Bruk den gitte filen `"LList.h"`, eller kopier og lim den følgende koden, og implementer en lenket liste hvor hver *node* inneholder en *string* som data.

```
class ListNode{
private:
    string value;
    ListNode *next;
public:
    ListNode(const string &);
    string getValue() const;
    ListNode *getNext() const;

    friend class LinkedList;
};

class LinkedList{
private:
    ListNode *head;
    ListNode *last;
public:
    LinkedList();
    ~LinkedList();
    bool isEmpty();
    void insertAtFront(const string &);
    void insertAtBack(const string &);
    bool removeFromFront(string &value);
    bool removeFromBack(string &value);

    friend ostream &operator <<(ostream &stream, const LinkedList &list);
};
```

Implementer følgende funksjoner og operator:

- `ListNode::ListNode(const string &value)`

  - initialiserer noden.

- `string ListNode::getValue() const`

  - returnerer verdien til noden.

- `ListNode *ListNode::getNext() const`

  - returnerer pekeren til den neste noden.

- `LinkedList::LinkedList()`

  - konstruktør som lager en ny liste med head og last satt til `NULL`.

- `LinkedList::~LinkedList()`

  - destruktør som sletter alle elementene i listen for å frigjøre minne.

- `bool LinkedList::isEmpty() const`

  - returnerer true hvis listen er tom.

- `void LinkedList::insertAtFront(const string &value)`

  - som legger til et element fremst i listen.

- `void LinkedList::insertAtBack(const string &value)`

  - som legger til et element bakerst i listen.

- `bool LinkedList::removeFromFront(string &value)`

  - som fjerner et element fremst i listen.
  Verdien til elementet som fjernes skal lagres i string-referansen som gies som argument.

- `bool LinkedList::removeFromBack(string &value)`

  - som fjerner et element bakerst i listen.
  Verdien til elementet som fjernes skal lagres i string-referansen som gies som argument.

- `ostream& operator<<(ostream &stream, const &LinkedList)`

  - som skriver ut listen i et lesbart format

Den lenkede listen du nettopp lagde kan brukes til å implementere andre datastrukturer på en lett måte. Hvordan ville du brukt LinkedList klassen for å implementere en *stack* eller *queue*? (Du trenger ikke å implementere dem.)

2 Søke i den lenkede listen (10 poeng)

Legg til en medlemsfunksjon for å søke i den lenkede listen etter en gitt *string*-verdi. Funksjonen skal returnere en peker til den første noden som inneholder verdien, eller NULL hvis verdien ikke finnes i listen.

```
ListNode *LinkedList::search(const string &value);
```

3 Slette noder fra den lenkede listen (10 poeng)

Legg til en medlemsfunksjon som fjerner og sletter (`delete`) alle elementene i en lenket liste som har lik verdi som en gitt string (argumentet til funksjonen).

```
void LinkedList::remove(const string &value);
```

4 Lage en «template»-klasse (10 poeng)

Lag en *template*-klasse basert på implementsajonene du har laget så langt. Hensikten med dette er å kunne lage lenkede lister for alle datatyper, og ikke bare for string.

Hint: Template-implementasjonen skal lages i en ny *header*-fil. For å gjøre en *template*-funksjon *friend* av en template-klasse brukes forskjellige *typename* for de to:

```
template<typename T>
class MyClass() {
    template<typename U> friend int myFunc(MyClass<U> myClass);
};
```

5 Sortert dobbellenket liste (20 poeng)

Lag *template*-klasser for en sortert, dobbellenket liste (`SortedDoubleLinkedList`). Noder skal settes inn i listen i sortert rekkefølge, basert på bruk av «mindre enn» operatoren (<). Dobbeltlenket liste betyr at hver node skal ha pekere til både den forrige og den neste noden i listen. Listen skal støtte duplikatverdier. `SortedDoubleLinkedList`-klassen skal ha funksjonen `insert(T &)` for å legge til nye verdier og funksjonen `remove(T &)` for å fjerne noder.

Hint: Hvis du ønsker kan du implementere den dobbeltlenkede listen for string-verdier først, og skrive den om til *template*-klasser etterpå.

6 Unntakshåndtering (20 poeng)

I denne oppgaven skal dere skrive om Matriseklassen fra øving 6 til å bruke unntakshåndtering i stedet for å returnere ugyldige matriser. Hvis du ikke gjorde øving 6 kan du bruke koden i som er tilgjengelig i løsningsforslaget.

a) Fjern den tomme konstruktøren fra klassene `Matrix` og `Vector`, samt funksjonen `isValid()`. La de resterende konstruktørene kaste et `std::invalid_argument` unntak med en passende feilmelding dersom noen forsøker å lage en matrise med dimensjoner:

`0 x COLS, Rows x 0 eller 0 x 0.`

Skriv også om konverteringskonstruktøren i `Vector` slik at den kaster et `std::invalid_argument` med en passende feilmelding dersom noen forsøker å konstruere en `Vector` fra en `Matrix` som ikke har dimensjoner ROWS x 1.

NB! Matrise- og Vector-klassene vil ikke kompilere før du har gjort neste oppgave også.

b) Lag unntaksklassen `MatrixDimensionMismatchException` som skal arve fra `std::logic_error`. Gi unntaksklassen passende medlemsvariabler og konstruktør for å kunne returnere følgende typer feilmeldinger fra `what()` funksjonen.

`"Matrix Dimensions do not agree in operator+. Undefined for Matrix 2x3 + Matrix 5x5."`

Hint: Puss støv av stringstream.

c) Skriv nå om alle operatorene i `Matrix`-klassen som tidligere kunne returnere ugyldige matriser til å heller kaste ett `MatrixDimensionMismatchException` unntak. Det kan være du må legge til ekstra tester der du gjenbruker operatorer, for å skrive ut korrekte feilmeldinger. Du vil også trenge å rydde litt i koden din fordi du nå har fjernet `isValid()` funksjonen.

d) Skriv kode i `main()` som tester og fanger unntakene du nettopp har skrevet.